


2019

Multi-touch Detection and Semantic Response on Non-parametric Rear-projection Surfaces

Jason Hochreiter
University of Central Florida

 Part of the [Computer Sciences Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Hochreiter, Jason, "Multi-touch Detection and Semantic Response on Non-parametric Rear-projection Surfaces" (2019). *Electronic Theses and Dissertations*. 6823.
<https://stars.library.ucf.edu/etd/6823>

MULTI-TOUCH DETECTION AND SEMANTIC RESPONSE ON NON-PARAMETRIC
REAR-PROJECTION SURFACES

by

JASON HOCHREITER
B.S. University of Central Florida, 2011
M.S. University of Central Florida, 2014

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2019

Major Professor: Gregory Welch

© 2019 Jason Hochreiter

ABSTRACT

The ability of human beings to physically touch our surroundings has had a profound impact on our daily lives. Young children learn to explore their world by touch; likewise, many simulation and training applications benefit from natural touch interactivity. As a result, modern interfaces supporting touch input are ubiquitous. Typically, such interfaces are implemented on integrated touch-display surfaces with simple geometry that can be mathematically parameterized, such as planar surfaces and spheres; for more complicated non-parametric surfaces, such parameterizations are not available. In this dissertation, we introduce a method for **generalizable optical multi-touch detection and semantic response on uninstrumented non-parametric rear-projection surfaces** using an infrared-light-based multi-camera multi-projector platform.

In this paradigm, touch input allows users to manipulate complex virtual 3D content that is registered to and displayed on a physical 3D object. Detected touches trigger responses with specific semantic meaning in the context of the virtual content, such as animations or audio responses. The broad problem of touch detection and response can be decomposed into three major components: determining *if* a touch has occurred, determining *where* a detected touch has occurred, and determining *how to respond* to a detected touch. Our fundamental contribution is the design and implementation of a relational lookup table architecture that addresses these challenges through the encoding of coordinate relationships among the cameras, the projectors, the physical surface, and the virtual content.

Detecting the presence of touch input primarily involves distinguishing between *touches* (actual contact events) and *hovers* (near-contact proximity events). We present and evaluate two algorithms for touch detection and localization utilizing the lookup table architecture. One of the

algorithms, a bounded plane sweep, is additionally able to estimate hover-surface distances, which we explore for interactions above surfaces.

The proposed method is designed to operate with low latency and to be generalizable. We demonstrate touch-based interactions on several physical parametric and non-parametric surfaces, and we evaluate both system accuracy and the accuracy of typical users in touching desired targets on these surfaces. In a formative human-subject study, we examine how touch interactions are used in the context of healthcare and present an exploratory application of this method in patient simulation. A second study highlights the advantages of touch input on content-matched physical surfaces achieved by the proposed approach, such as decreases in induced cognitive load, increases in system usability, and increases in user touch performance. In this experiment, novice users were nearly as accurate when touching targets on a 3D head-shaped surface as when touching targets on a flat surface, and their self-perception of their accuracy was higher.

To my family and friends.

ACKNOWLEDGMENTS

I am fortunate to have had the opportunity to work with many wonderful people throughout my doctoral studies. Without their contributions, this dissertation would not have been possible.

First, I would like to extend my sincere gratitude to my advisor and committee chair, Prof. Gregory Welch, for his constant support over the last several years. I am extremely grateful for his thoughtful guidance and advice, whether academic in nature or simply about life in general. Likewise, I would like to thank my remaining committee members Prof. Gerd Bruder, Prof. Joseph LaViola Jr., Prof. Laura Gonzalez, and Prof. Juan Cendán. My committee's expertise covered a broad range, including human-computer interaction, 3D user interfaces, statistics, human-subject studies, and healthcare simulation, each of which played an invaluable role in the research described in this dissertation. I am extraordinarily appreciative of all of their many contributions.

Finally, I would like to express my deepest thanks to all of my colleagues and collaborators at the Synthetic Reality Lab, both current and former, and across the University of Central Florida, including (alphabetically) Mindi Anderson, Salam Daher, Desiree Díaz, Austin Erickson, Hassan Foroosh, Charles Hughes, Eric Imperiale, Kate Ingraham, Yazdan Jamshidi, Sungchul Jung, Kang-soo Kim, Barbara Lee, Myungho Lee, Arjun Nagendran, Nahal Norouzi, Andrew Raij, and Ryan Schubert. I could not have asked for a better group of collaborators.

Note: This material includes work supported in part by the National Science Foundation under Grant Number 1564065 (Dr. Ephraim P. Glinert, IIS) and Grant Number 1800961 (Dr. Tonya Smith-Jackson, IIS), the Office of Naval Research under Grant Number N00014-17-1-2927 (Dr. Peter Squire, Code 34), and the AdventHealth Endowed Chair in Healthcare Simulation (Prof. Welch). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the supporting institutions.

TABLE OF CONTENTS

LIST OF FIGURES	xvi
LIST OF TABLESxxiv
CHAPTER 1: INTRODUCTION	1
1.1 Challenges	3
1.2 Thesis Statement	8
1.3 High-Level Overview	9
1.4 Healthcare Training: Physical-Virtual Patients	11
1.5 Matched and Mismatched Physical-Virtual Content	16
1.6 Outline	17
CHAPTER 2: RELATED WORK	19
2.1 Touch Sensing Technology	21
2.1.1 Hardware-Based Touch Sensing	22
2.1.1.1 Resistive Sensing	22
2.1.1.2 Capacitive Sensing	23

2.1.1.3	Other	25
2.1.2	Camera-Based Touch Sensing	26
2.1.2.1	Frustrated Total Internal Reflection	26
2.1.2.2	Rear Illumination	29
2.1.2.3	Other	34
CHAPTER 3: METHODOLOGY		36
3.1	Overview	38
3.1.1	Lookup Table	41
3.2	Sensing Touch	46
3.2.1	Preprocessing Phase	47
3.2.1.1	Summary	62
3.2.2	Runtime Touch Detection	63
3.2.2.1	Projection Space Touch Detection	69
3.2.2.1.1	Algorithm	74
3.2.2.1.2	Touch Localization	78
3.2.2.1.3	Multi-Touch Detection	79
3.2.2.1.4	Multiple Projectors	80

3.2.2.2	Plane Sweep Touch Detection	80
3.2.2.2.1	Algorithm	85
3.2.2.2.2	Touch Localization	90
3.2.2.2.3	Multi-Touch Detection	92
3.2.2.3	Comparison of Approaches	92
3.2.2.4	Touch Labeling	95
3.3	Responding to Touch	97
3.3.1	Semantic Content Engine	97
3.3.1.1	Updates to Touch/Hover Classification Algorithms	104
3.3.1.1.1	Projection Space Touch Detection	104
3.3.1.1.2	Plane Sweep Touch Detection	105
3.3.2	Touch Messages	105
3.3.3	Rendering System	106
3.4	Head-Mounted Display Touch	107
3.4.1	IR Environment Mapping	108
3.4.2	Physical-Virtual Alignment	108
CHAPTER 4: REALIZATIONS		111

4.1	Software	111
4.1.1	Preprocessing Phase Implementation	114
4.1.1.1	Special Case: Plane	125
4.1.2	Runtime Implementation	126
4.1.2.1	Processing Camera Imagery	126
4.1.2.2	Touch/Hover Classification Algorithms	127
4.1.2.2.1	Base Touch Detector	127
4.1.2.2.2	Projection Space Touch Detector	128
4.1.2.2.3	Plane Sweep Touch Detector	129
4.1.2.2.4	Sending Touch Messages	130
4.1.2.2.5	Lookup Table Accesses	130
4.1.2.3	Semantic Content Engines	131
4.1.2.3.1	Projector-Space-Based Applications	131
4.1.2.3.2	Hover-Based Interactions	132
4.1.2.3.3	Unity Environments	132
4.1.2.3.4	Receiving Touch Messages	135
4.2	Hardware	136

4.2.1	Common Elements	137
4.2.2	Prototype Rig I: Small	139
4.2.3	Prototype Rig II: Large	141
CHAPTER 5: DEMONSTRATIONS		143
5.1	Plane	143
5.1.1	Preprocessing Phase	144
5.1.2	Semantic Content Engines	145
5.2	Bowl	150
5.2.1	Preprocessing Phase	150
5.2.2	Semantic Content Engines	151
5.3	Head	152
5.3.1	Preprocessing Phase	157
5.3.2	Semantic Content Engine	158
5.3.2.1	3D Model	159
5.3.2.2	Patient Simulation: Healthy and Stroke	161
5.3.2.3	Touch Interactions	162
5.4	Child	165

5.4.1	Preprocessing Phase	166
5.4.2	Semantic Content Engine	167
CHAPTER 6: SYSTEM EVALUATION		170
6.1	System Evaluation Metrics	170
6.2	Bowl	176
6.2.1	Projector Targets	176
6.2.1.1	Projection Space	176
6.2.1.2	Plane Sweep	177
6.2.1.3	Summary	182
6.2.2	Touch Targets	185
6.2.2.1	Projection Space	185
6.2.2.2	Plane Sweep	189
6.2.2.3	Summary	192
6.3	Head	195
6.3.1	Projector Targets	195
6.3.1.1	Projection Space	195
6.3.1.2	Plane Sweep	196

6.3.1.3	Summary	196
6.3.2	Touch Targets	203
6.3.2.1	Projection Space	203
6.3.2.2	Plane Sweep	207
6.3.2.3	Summary	210
6.4	Child	213
6.4.1	Projector Targets	213
6.4.1.1	Projection Space	214
6.4.1.2	Plane Sweep	219
6.4.1.3	Summary	224
6.4.2	Touch Targets	226
6.4.2.1	Projection Space	226
6.4.2.2	Plane Sweep	227
6.4.2.3	Summary	233
6.5	Algorithm Comparison	235
CHAPTER 7: USER STUDIES		238
7.1	Formative Study: Stroke Patient Simulation	238

7.2	Physical-Virtual Mismatches	242
7.2.1	Introduction	244
7.2.2	Experimental Setup	248
7.2.2.1	Touch Sensing	250
7.2.2.2	Visual Stimuli	251
7.2.2.3	Computing	253
7.2.3	Experiment	253
7.2.3.1	Participants	254
7.2.3.2	User Tasks	254
7.2.3.2.1	Touch Accuracy Phase	255
7.2.3.2.2	Cognitive Load Phase	256
7.2.3.3	Study Procedure	258
7.2.4	Results	259
7.2.4.1	Touch Performance	260
7.2.4.2	Cognitive Load	260
7.2.4.3	Subjective Responses	263
7.2.4.3.1	Task Load	263

7.2.4.3.2	Usability	264
7.2.4.3.3	Preferences	264
7.2.4.3.4	Rankings	265
7.2.5	Discussion	265
CHAPTER 8: CONCLUSIONS AND FUTURE WORK		269
APPENDIX: UNITY PROJECTION MATRICES		272
LIST OF REFERENCES		282

LIST OF FIGURES

1.1	Graphical touch interactions on rear-projection surfaces	3
1.2	Touch interactions on virtual patient simulators	4
1.3	Proximity-based interactions	7
1.4	Simplified high-level overview of the proposed method	9
1.5	Touch interactions in healthcare	12
3.1	High-level overview of the proposed method	40
3.2	Relationships among the coordinate spaces used to achieve touch sensing and response	44
3.3	Supplementing unobserved correspondences with back-projected rays	59
3.4	Supplementing unobserved correspondences with forward-projected inter- sections	61
3.5	IR image processing	65
3.6	The conversion of camera contours to their corresponding positions on the graphical mesh	67
3.7	A potential touch or hover event	69
3.8	The camera-to-projector conversions of a touch and a hover in a simplified 2D world	71

3.9	Multiple camera-to-projector conversions in a simplified 2D world	72
3.10	Touch convergence in projector space	73
3.11	Convergence of camera-to-projector contours of a touch and a hover	74
3.12	Example projector response masks	76
3.13	Ambiguity in projector space touch/hover classification	82
3.14	Plane sweep for a simulated touch	83
3.15	Plane sweep for a simulated hover	84
3.16	Camera uncertainty	90
3.17	Disjoint and convex planar projection union areas	91
3.18	General format of a touch message	106
4.1	Calibration captures of an asymmetric circle grid pattern	115
4.2	Two prototype touch sensing rigs	137
4.3	User interaction with the two prototype touch sensing rigs	138
4.4	Touch-sensitive surfaces on Prototype Rig I	140
4.5	The touch-sensitive child surface on Prototype Rig II	142
5.1	The touch-sensitive plane	144
5.2	Plane surface preprocessing phase	145

5.3	A touch-based painting application on the plane surface	147
5.4	A touch-based drag and drop interface for the plane surface	148
5.5	A theremin-like musical instrument interface based on hover distance from the plane surface	149
5.6	The touch-sensitive bowl surface	150
5.7	Bowl surface preprocessing phase	151
5.8	A touch-based painting application on the bowl surface	153
5.9	A touch-based drag and drop interface for the bowl surface	154
5.10	Touch-initiated contour lines on the bowl surface	155
5.11	The touch-sensitive physical-virtual head	156
5.12	Simulated patients on the head surface	157
5.13	Head surface preprocessing phase	158
5.14	General process for modeling, texturing, and animating the physical-virtual head	159
5.15	A touch-sensitive physical-virtual patient simulator	160
5.16	A touch-sensitive physical-virtual stroke patient simulator	162
5.17	Capillary refill on the head surface	163
5.18	Blendshapes on the head surface	164

5.19	Blendshapes on the stroke patient	165
5.20	Touch-triggered audio on the head surface	165
5.21	The touch-sensitive physical-virtual child	166
5.22	Child surface preprocessing phase	167
5.23	Capillary refill on the child surface	168
5.24	Blendshapes on the child surface	169
6.1	Results: bowl surface, projection space algorithm, projected-target-detection distance	178
6.2	Results: bowl surface, projection space algorithm, projected target multi-camera agreement score	179
6.3	Results: bowl surface, plane sweep algorithm, projected-target-detection distance	180
6.4	Results: bowl surface, plane sweep algorithm, projected target minimum union plane	181
6.5	Touch detection algorithm comparison: 3D projected-target-detection distance graph for the bowl surface	184
6.6	Results: bowl surface, projection space algorithm, touch-target-detection distance	186

6.7	Results: bowl surface, projection space algorithm, touch target multi-camera agreement score	187
6.8	Results: bowl surface, projection space algorithm, touch/hover classification .	188
6.9	Results: bowl surface, plane sweep algorithm, touch-target-detection distance	190
6.10	Results: bowl surface, plane sweep algorithm, touch target minimum union plane	191
6.11	Results: bowl surface, plane sweep algorithm, touch/hover classification . . .	192
6.12	Touch detection algorithm comparison: 3D touch-target-detection distance graph for the bowl surface	194
6.13	Results: head surface, projection space algorithm, projected-target-detection distance	197
6.14	Results: head surface, projection space algorithm, projected target multi-camera agreement score	198
6.15	Results: head surface, plane sweep algorithm, projected-target-detection distance	199
6.16	Results: head surface, plane sweep algorithm, projected target minimum union plane	200
6.17	Touch detection algorithm comparison: 3D projected-target-detection distance graph for the head surface	202

6.18	Results: head surface, projection space algorithm, touch-target-detection distance	204
6.19	Results: head surface, projection space algorithm, touch target multi-camera agreement score	205
6.20	Results: head surface, projection space algorithm, touch/hover classification .	206
6.21	Results: head surface, plane sweep algorithm, touch-target-detection distance	208
6.22	Results: head surface, plane sweep algorithm, touch target minimum union plane	209
6.23	Results: head surface, plane sweep algorithm, touch/hover classification . . .	210
6.24	Touch detection algorithm comparison: 3D touch-target-detection distance graph for the head surface	212
6.25	Results: child surface body projector, projection space algorithm, projected-target-detection distance	215
6.26	Results: child surface head projector, projection space algorithm, projected-target-detection distance	216
6.27	Results: child surface body projector, projection space algorithm, projected target multi-camera agreement score	217
6.28	Results: child surface head projector, projection space algorithm, projected target multi-camera agreement score	218

6.29	Results: child surface body projector, plane sweep algorithm, projected- target-detection distance	220
6.30	Results: child surface head projector, plane sweep algorithm, projected-target- detection distance	221
6.31	Results: child surface body projector, plane sweep algorithm, projected target minimum union plane	222
6.32	Results: child surface head projector, plane sweep algorithm, projected target minimum union plane	223
6.33	Touch detection algorithm comparison: 3D projected-target-detection dis- tance graph for the child surface	225
6.34	Results: child surface, projection space algorithm, touch-target-detection dis- tance	228
6.35	Results: child surface, projection space algorithm, touch target multi-camera agreement score	229
6.36	Results: child surface, projection space algorithm, touch/hover classification .	230
6.37	Results: child surface, plane sweep algorithm, touch-target-detection distance	231
6.38	Results: child surface, plane sweep algorithm, touch target minimum union plane	232
6.39	Results: child surface, plane sweep algorithm, touch/hover classification . . .	233

6.40	Touch detection algorithm comparison: 3D touch-target-detection distance graph for the child surface	235
7.1	The two experimental conditions of the stroke study	241
7.2	Participants interacted with four representations of a 3D virtual head that varied in physical form and visual display	243
7.3	The four study conditions	245
7.4	Midair touch condition	247
7.5	Study platform	249
7.6	Alignment of a virtual HoloLens hologram to a physical object	252
7.7	The set of 39 visual targets displayed to participants during the four study conditions, distributed across the 3D head model	255
7.8	Example <i>Cognitive Load Phase</i> trials	257
7.9	Results for the <i>Touch Accuracy Phase</i> , separated by study condition	261
7.10	Results for the target selection task in the <i>Cognitive Load Phase</i>	262
7.11	Results of the subjective questionnaires	267
7.12	Subjective rankings for the four experimental conditions	268

LIST OF TABLES

3.1	Example lookup table	45
3.2	Lookup table after observing sparse projector-camera and camera-camera correspondences	51
3.3	Lookup table after triangulating sparse camera-camera correspondences	53
3.4	Lookup table after creating the touch surface mesh	56
3.5	Dense lookup table after back-projecting all camera and projector pixels to 3D rays	60
3.6	Comparison between projection space and plane sweep touch detection algorithms	94
3.7	Lookup table augmented with semantic regions for animations	99
3.8	Lookup table augmented with generic semantic regions	100
3.9	Lookup table augmented with the closest vertex indices of the graphical mesh	103
6.1	Projector target results summary for the bowl surface	183
6.2	Touch target results summary for the bowl surface	193
6.3	Touch/hover classification results summary for the bowl surface	193
6.4	Projector target results summary for the head surface	201

6.5	Touch target results summary for the head surface	211
6.6	Touch/hover classification results summary for the head surface	211
6.7	Projector target results summary for the child surface	224
6.8	Touch target results summary for the child surface	234
6.9	Touch/hover classification results summary for the child surface	234
6.10	Summary of projection space and plane sweep algorithm results on the bowl, head, and child surfaces	237

CHAPTER 1: INTRODUCTION

One of the fundamental aspects of human life is our ability to *touch*: to allow for the use of tools, as a means to learn about our surrounding environment, or even as a form of communication with others. The first sense a human fetus is able to detect is touch, which remains central to the development of a growing child, who learns by touching objects in his or her environment [54,108]. As a communication tool, touch has been shown to be capable of conveying a wide variety of emotions and meanings, including anger, fear, love, compliance, support, greetings, departures, sympathy, and companionship [65,66,67,80,152,156,157]. In the modern world, a significant and growing number of technologies support touch input.

Inspired by this, there is a growing desire to study and leverage the psychological effects of human touch in human-computer interaction, especially in the context of three-dimensional interfaces [23,24,86,136,137,138]. Many interfaces consider ISO 9241-9, a standard which provides guidelines for “non-keyboard input devices,” including touch interfaces, along with mechanisms for comparing input methods [75]. Fitts designed a model representing the difficulty of a target selection task [49]; known widely as *Fitts’ Law*, the model suggests that the amount of time required for a user to select a target—whether by physical touch or mouse input—increases as the distance from the starting point to the target increases and as the width of the target decreases [12]. Touch may be incorporated either directly, through the use of tangible interfaces, or indirectly, through haptic simulation [104]. Touch interfaces that incorporate some form of tactile feedback, such as small vibrotactile devices that are attached to a user’s fingers or hands [3,111,128], can lead to decreases in task time and increases in accuracy [4,118]. In particular, user response times to tactile stimuli have been shown to be significantly faster than responses to visual or auditory stimuli [110].

User performance in touch tasks can be measured in a variety of ways:

- *Accuracy and precision*: How closely does a user’s detected touch position match the intended touch position? Can users correctly and consistently select specific touch input locations when many are available?
- *Response time*: How long does it take a user to select—both locate and touch—a specific target?
- *Cognitive load and usability*: What sorts of mental, physical, or temporal demands are placed on users of a system? Can they complete other secondary tasks simultaneously?
- *Subjective feedback*: Do users prefer one touch input system over another? Do they generally consider one to be easier or harder than another?

In this dissertation, we investigate the problem of uninstrumented multi-touch detection and semantic response on non-parametric rear-projection surfaces. Such surfaces provide physical affordances, requiring no simulation or alternate presentation of tactile feedback, and allow the direct tactile manipulation of virtual content such as graphical imagery (Figure 1.1). Often, we describe these as *physical-virtual surfaces*, being composed of a physically perceivable shape onto which virtual imagery is displayed. For example, Figure 1.2 shows interactions with virtual patient models on matching physical surfaces in which touch input can be used to examine the teeth and eyes of the simulated patients. Additionally, we show that physical surfaces with geometry that matches or closely matches the geometry of the virtual content promote improved touch performance, lower cognitive load, increased usability, and greater subjective enjoyment compared to other touch input paradigms. While the proposed methodology is designed to be general, we specifically consider supporting touch input in the context of physical-virtual agents for human

patient simulation, where—despite the importance of touch for both diagnostic and therapeutic reasons in healthcare—touch is not typically an available input modality.

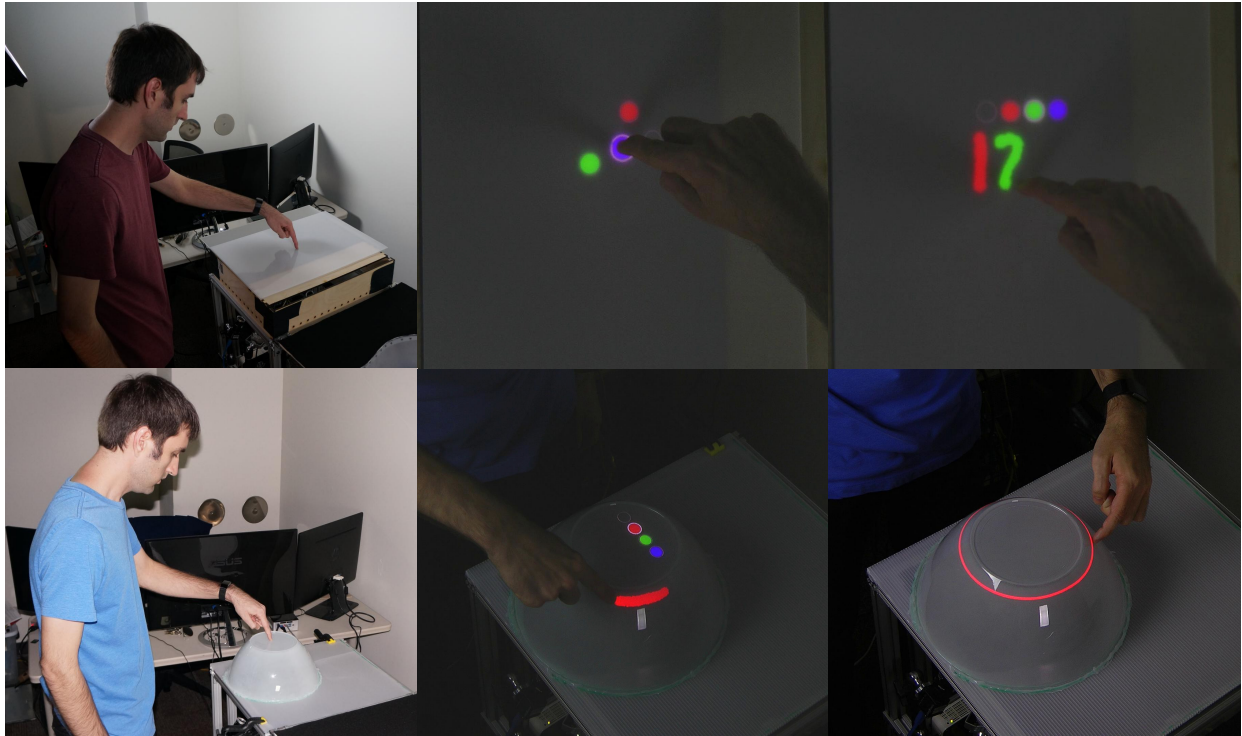


Figure 1.1: Graphical touch interactions on rear-projection surfaces. The imagery projected on the surface updates in response to user touch input.

1.1 Challenges

Supporting touch as an input modality in human-computer interfaces is a challenging problem. Each specific touch-sensitive interface imposes certain requirements on both itself and its users. For instance, interfaces that require the display of dynamic, touch-responsive imagery must have a mechanism for realizing that imagery. If the imagery is projected using spatial augmented reality [13], practical concerns include the position of the projector or projectors and the availability of touch sensing methods that can successfully operate alongside projected imagery. Alternatively, if

the touch-sensitive interface is directly integrated with a graphical display surface using hardware touch sensing elements, there may be practical limitations on the geometry of the surface, including shape and size, and on the resolution of touch input that can be achieved. In many ways, the nature of the touch-sensitive surface itself dictates the approaches that are available, since certain classes of touch detection methods are not capable of handling surfaces with arbitrary curvature.

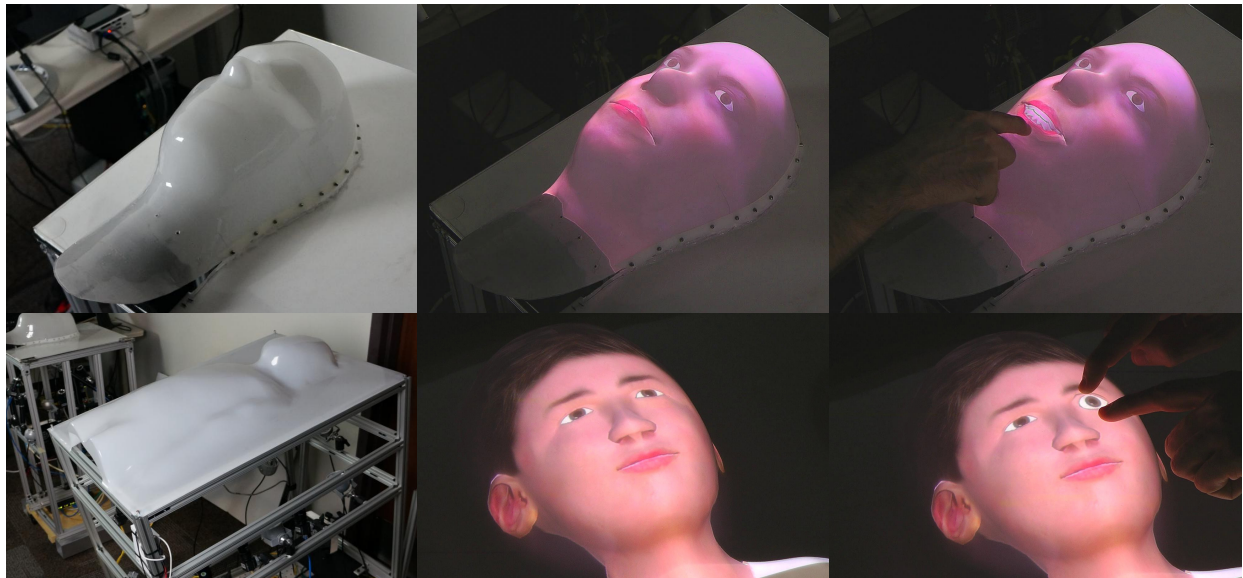


Figure 1.2: Touch interactions on virtual patient simulators. The graphical model projected onto each surface updates in response to touch input.

In this dissertation, we describe a general methodology to detect and respond to touch input on non-parametric rear-projection surfaces. Accordingly, we address the following specific assumptions and requirements:

- *Non-planar and non-parametric surfaces:* Typical touchscreens are often limited in form factor to flat surfaces, such as in smartphone and tablet devices. While suitable for many applications, such planar input surfaces may pose increased difficulty when users interact with three-dimensional virtual content. Additional manipulation techniques, such as rotating and

translating the virtual content, might be necessary for users to accurately touch specific virtual components, which may prove less intuitive than more direct interaction on a matching three-dimensional physical surface. Furthermore, the representation of 3D content on flat surfaces removes a variety of cues about the content’s geometry, such as its shape, its size, and the physical relationship between its components. Thus, we focus on extending touch sensing to non-parametric surfaces.

- *Generalizability*: Though we are motivated by applications to the healthcare domain, our proposed method is designed to support a variety of both parametric and non-parametric rear-projection surfaces through the same fundamental underlying software architecture. As such, it is appropriate for simpler surfaces, such as planes and hemispheres (Figure 1.1), and more complicated surfaces, such as human shapes for patient simulation (Figure 1.2).
- *Semantically defined touch responses*: To ensure highly responsive, interactive physical-virtual experiences, we describe a mechanism for assigning semantic meanings to touch input in a virtual content engine supporting a variety of output modalities, such as updated projected imagery and sound effects. The touch sensing and response systems are designed to be tightly linked so that detected touches rapidly trigger appropriate responses. For example, a healthcare practitioner can examine the gums of a patient by moving his or her lips, or an artist may paint an object via touch-based controls.
- *Dynamic imagery*: One of the core components of our proposed system is the ability for users to interact with and affect dynamic visual imagery registered to the touch surface. While flat touchscreens generally comprise integrated touch-display surfaces, our desire to support non-parametric surfaces is more suitable to imagery from multiple projectors. Moreover, since touch input is another integral aspect, we specifically rely on rear-projection solutions so that the user’s arms, hands, and fingers do not occlude the virtual imagery.

- *Multi-touch detection*: A key advantage of touch input methods compared to traditional computer mouse input is the ability to interpret multiple simultaneous touches, including multiple fingers and multiple hands, which provides a far richer set of available interaction techniques. Thus, our methodology is designed to detect, localize, and track multiple touches over time.
- *Uninstrumented touch*: Instrumenting users—for instance, with tracked markers or touch controllers—to allow for input may induce additional cognitive load that interferes with learning or training objectives. Likewise, instrumenting the surface itself to achieve touch sensing may disrupt projected imagery or might be limited in resolution, which could have similar negative effects. As opposed to any instrumentation of the users or the surface, we instead rely on cameras to detect touch input.
- *Accurate localization*: To assess the reliability of our methodology, we perform an evaluation of system consistency across multiple rear-projection surfaces. Furthermore, we investigate the distances between localized user touch input and intended targets.
- *Touch and hover disambiguation*: Certain touch sensing approaches potentially experience interference due to *hovers*—non-contact proximity events, such as a user holding his or her hand close to the surface without actually touching it. Our proposed methodology is designed to distinguish such events from actual touch input, which we demonstrate through user-collected datasets of touches and hovers. Additionally, we consider this problem from a secondary viewpoint that interprets hovers as intended input events by estimating the distance from the user’s fingers to the surface—for example, in the context of a musical instrument interface that emits notes determined by this distance (Figure 1.3).
- *Low-latency processing*: The latency between touch input and semantic output can greatly impact a user’s experience. For instance, in the context of touch-based interactions with a

virtual human, excessively high processing latency can diminish the user’s perceived realism of the virtual human, perhaps leading to unintended behavioral differences. As such, many of our design decisions focus on reducing latency throughout this process.

Our proposed methodology addresses these challenges by establishing specific relationships between optical *devices*—cameras used for touch sensing and projectors used for visual output—and the interaction *content*—the touch surface along with the accompanying graphical model and semantic behavior.

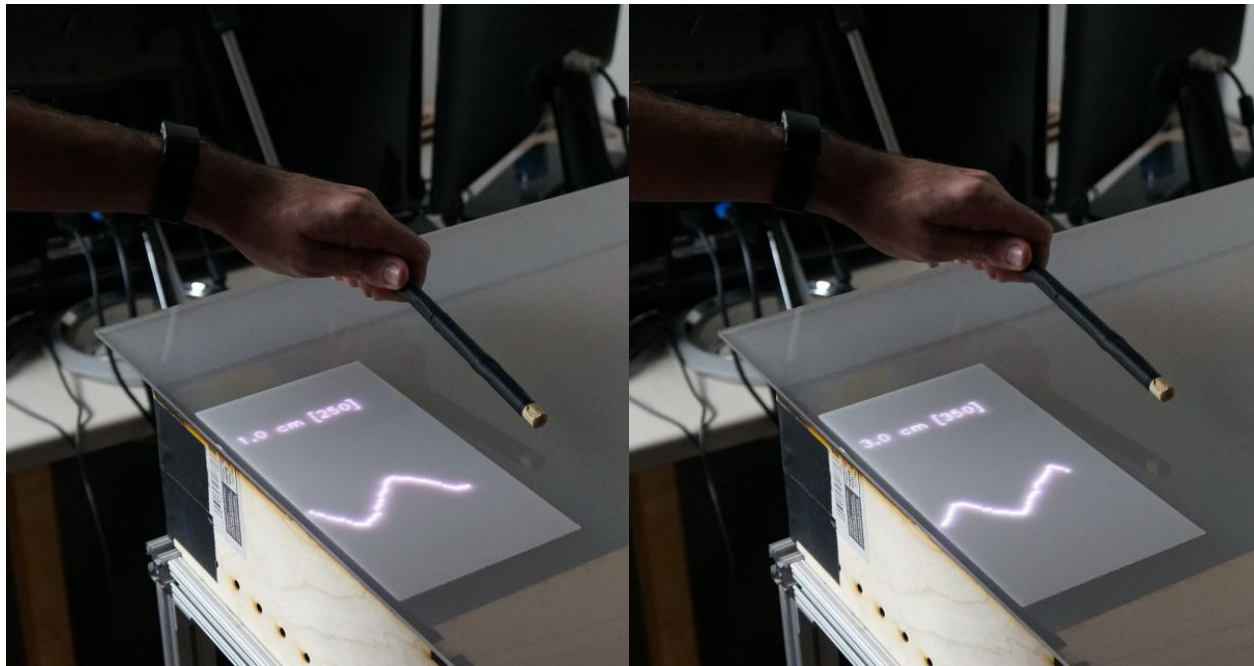


Figure 1.3: Proximity-based interactions for a musical instrument interface. The distance between the surface and the control wand is estimated and used to control the pitch of an emitted musical tone, which is visualized in a frequency graph.

1.2 Thesis Statement

TS1) A device-content relational lookup table architecture can be used to realize a generalizable system for rendering application-based content, with integrated and registered optical multi-touch sensing, for interactions on uninstrumented non-parametric surfaces.

TS2) The use of such a device-content relational lookup table architecture:

- (a) supports both non-parametric and parametric surfaces;
- (b) supports low-latency runtime processing linear in the number of touch input regions;
- (c) supports constant-time conversions between the coordinate frames of the optical devices (cameras and projectors), the physical surface, and the virtual content;
- (d) supports refinement based on existing reference models and functions;
- (e) supports direct registration of rendered content and touch sensing; *and*
- (f) encodes touch input to affect content-specific semantic output.

TS3) A bounded plane sweep method can be used to distinguish touches from hovers and to estimate hover-surface distances for interactions above the surface.

TS4) Touch sensing with a content-matched physical surface results in improved touch performance compared to both a non-matching physical surface and a virtual surface.

1.3 High-Level Overview

Building on methods initially developed for touch interaction and display on planar surfaces (such as Matsushita and Rekimoto’s HoloWall [103] and Wilson’s TouchLight [153]) and spherical surfaces (Benko, Wilson, and Balakrishnan’s Sphere [7]), we introduce a camera-based system with rear infrared (IR) illumination and rear-mounted projected imagery specifically designed to be generalizable to a variety of non-parametric surfaces. Our system comprises three tightly linked components:

1. The **touch sensing** system analyzes camera imagery for touches. It has two primary goals: *determining if and where* a touch has occurred on the surface. This includes distinguishing touches from near-contact hover events.
2. The **semantic content engine** determines *how to respond* to a detected touch, such as by updating the projected imagery or playing audio, via predefined state- and region-based semantic mappings between touch input and responses.
3. The **rendering system** *provides output* to the user in a variety of modalities, such as visual and audio, as instructed by the semantic content engine.

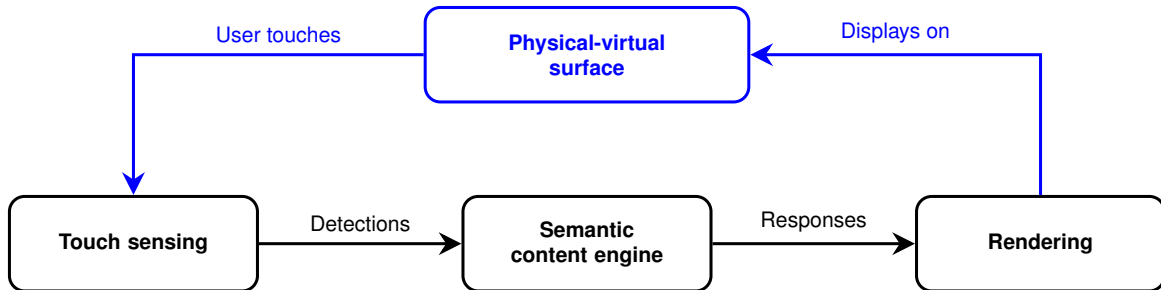


Figure 1.4: Simplified high-level overview of the proposed method, showing the connections between the touch sensing system, the semantic content engine, and the rendering system.

One of our core contributions is the design and implementation of a relational lookup table architecture that links these components to achieve touch sensing and response. At a high level, this lookup table encodes relationships between camera pixels, projector pixels, virtual content coordinates, and virtual content behavior. Figure 1.4 shows a simplified summary of the overall methodology.

The general physical realization of our proposed touch sensing methodology comprises multiple IR cameras and light sources located below and oriented toward a rear-projection surface. IR light from these light sources passes through the surface; as a user's fingers or hands approach the surface and eventually contact it, this IR light is increasingly reflected back down through the surface and imaged by the set of IR cameras. Subsequently, the system determines whether or not the user has touched the surface by processing these camera images, using information encoded in the lookup table. Fingers or hands in close proximity to the surface may produce imagery that is largely similar to images of actual touches in terms of pixel intensity and contour sizes; by considering the imagery from the entire collection of IR cameras, the touch sensing system can determine whether a true touch or a non-contact hover has occurred. Alongside the rear-mounted IR cameras and lights, visible light projectors augment the surface with registered dynamic imagery, such as from a three-dimensional virtual model or a graphical touch-based interface. As IR light is largely invisible to the human eye, it does not interfere with imagery projected onto the surface; likewise, since the projected imagery exists in the human visible light spectrum, it is invisible to the IR cameras and thus does not disrupt touch sensing.

When a touch has been detected, the touch sensing system again utilizes the lookup table relationships to process the same collection of camera imagery to determine the touch's location in the three-dimensional space of the virtual content so that an appropriate semantic response can be initiated. For planar or simple parametric objects, this localization step is straightforward, generally requiring at most the evaluation of a simple equation or set of equations whose inputs are

the two-dimensional touch coordinates as observed by the cameras and whose outputs are three-dimensional coordinates on the physical touch surface. However, for non-parametric objects, no such equation or set of equations exists. The lookup table serves as a substitute for these equations, providing instantaneous conversions between camera pixels, projector pixels, and 3D coordinates on the surface to support highly responsive touch interactions on such non-parametric surfaces.

The virtual content displayed to the user can range from lightweight touch-based interfaces to sophisticated 3D simulations with associated visual and audio behaviors. Interaction with this content is handled by the semantic content engine, which is responsible for assigning region- and state-based semantic meanings to detected touches, again via the lookup table. For instance, touch input at a particular location in one semantic state may trigger a sound effect, while touch input at the same location may initiate an animation in a different state. Detected touches are transmitted from the touch sensing system to the semantic content engine via messages that encode all of the relevant information used to determine the appropriate response; in general, at runtime, the semantic content engine simply needs to evaluate a few conditionals or mathematical expressions. Once the response has been identified, the rendering system outputs this to the user through the use of the rear-mounted projectors, speakers, or other output devices.

1.4 Healthcare Training: Physical-Virtual Patients

We are particularly interested in applying this technique to the healthcare training domain as a complement to existing patient simulator technologies, for which integrated touch input and response is generally not available. This is in spite of the fact that touch is a common component of patient care, including for diagnostic or comfort-related purposes (summarized in Figure 1.5), suggesting the value of supporting touch as an input modality for patient simulators. While the ability to touch a simulated patient is itself useful, the primary benefits arise from the manner in which

the simulator responds to or exhibits awareness of touch. In fact, in an exploratory study, we found that even external, manual control of virtual patient touch awareness by an observer monitoring trainee behavior over a video camera—often referred to as a Wizard of Oz paradigm [37]—can be a valuable component of a patient simulator [34]. However, in this dissertation, we are focused on supporting automatic touch detection and response, particularly for the kinds of touch input that are not practically achievable in such human-in-the-loop paradigms due to the level of accuracy, responsiveness, and dynamic interactivity needed—for instance, if a nurse needs to examine a patient’s teeth by manually moving his or her lips or touch a patient at extremely precise locations.




Type Purpose											
		Push	Palm	Two Hands	Multiple Fingers	Pinch/Spread	Object	Grasp	Tap	Rub	Stroke/Wipe
Diagnosis	Palpation	✓		✓	✓						
	Examine	✓		✓	✓	✓		✓	✓		
	Measure						✓				
	Differentiate						✓				✓
Therapy	Warm/Cool						✓				✓
	Massage	✓	✓	✓	✓					✓	✓

Figure 1.5: Common types and purposes of touch in patient care. Table contents developed together with (alphabetical) Kelly Allred, Laura Gonzalez, Mary Lou Sole, Steve Talbert, and Gregory Welch.

Rather than targeting this application exclusively, this dissertation presents a generalizable approach that is suitable to a variety of scenarios, including healthcare training. However, this specific motivation informed some of our design decisions, and so we briefly cover relevant aspects. Before we describe typical healthcare simulators, we first introduce a few important definitions.

Lombard and Ditton define *presence* as “the perceptual illusion of nonmediation” [95]. They further expand this notion as follows:

An “illusion of nonmediation” occurs when a person fails to perceive or acknowledge the existence of a medium in his/her communication environment and responds as he/she would if the medium were not there.

This concept has important implications on healthcare training, which often relies on patient simulation: as the presence of a particular medium increases, training can be more effective [41], users are more likely to treat the medium as a social entity, and user behavior becomes more typical of human-human interactions.

Other researchers have extended this definition, examining related and more specific types of presence. Biocca et al. define *social presence* as “the sense of being with another” [14]. *Co-presence* has been described as a person’s perception of another person and their sense of being perceived by the other person [56]; Harms and Biocca consider co-presence to be one of six sub-dimensions of social presence [62]. Furthermore, the *physicality* of an agent or avatar includes the fidelity of its physical occupancy—attributes such as size, shape, and position—and its ability to change or sense changes in the surrounding environment [28]. Increasing the physicality of agents and avatars has been shown to increase social presence and lead to more realistic behavior from users [29, 78, 89, 94, 123].

From these findings, there are immediate benefits arising from the methodology presented in this dissertation. When representing simulated patients, non-parametric surfaces afford increased physicality, which results in increased social presence. Likewise, surfaces of any shape and size can exhibit greater awareness of the environment with the addition of touch-triggered behavior, which further increases physicality and thus social presence. Such increases ultimately promote more realistic trainee behavior and thus more effective and engaging training.

Typically, patient simulators in modern healthcare training can be categorized as *standardized patients*, *computer-based* simulators, *mannequin-based* simulators, or *mixed-reality* simulators [93].

- *Standardized patients* are humans who have been trained to simulate a specific patient or patients in a consistent, replicable manner [96]. Along with mimicking symptoms, standardized patients present an accompanying patient history and simulate the various physical and emotional behaviors that an actual patient representing these symptoms would exhibit. Simulations with humans naturally benefit from an extremely high degree of realism, physical presence, and social presence. However, certain situations preclude the usage of standardized patients, such as symptoms that are difficult to realistically replicate or scenarios that involve pediatric patients.
- *Computer-based* simulators instead portray these symptoms and other aspects through computer screens and audio speakers [96]. Compared to standardized patients, computer-based simulators can be advantageous due to their ability to display dynamic imagery, potentially allowing for the portrayal of complicated visual symptoms. One notable disadvantage is the lack of physicality resulting from the two-dimensional representation of the patient, which may make simulations less compelling or prevent learners from treating the simulated patient in a realistic manner. As they are controlled via software, computer-based simulators can provide extremely consistent experiences through specific predefined behaviors; they can also be integrated with artificial intelligence control.
- *Mannequins* are life-sized, human-like models of the entire body or a particular subset [96]. Many mannequins support various physiological functions, including detectable pulses and simulated breathing capabilities. Like standardized patients, they exhibit a high degree of physicality as a result of their realistic shapes and sizes, but they are generally static in appearance and unable to simulate certain visual symptoms. A *task trainer* represents only

a subset of the human body, such as an arm, which is sometimes all that is necessary for a particular training scenario; if desired, both a task trainer and another simulation technique can be used in a hybrid approach.

- *Mixed-reality simulators* are a special class of computer-based simulators [96] across a wide spectrum of techniques. Virtual reality patients can be presented in a fully three-dimensional virtual environment via head-mounted displays, affording a high sense of spatial presence. Additionally, spatial augmented reality supports the addition of virtual imagery, generally via projectors, to physical objects in the user's environment, which could include virtual patients [13]. As they are computer-based, these simulations are all capable of displaying dynamic imagery, whether using head-mounted displays or projectors.

Each of the above patient simulators has benefits and disadvantages in terms of the variety of medical symptoms it can portray and its effectiveness in training. **Physical-virtual patients** [123] are a hybrid approach that aims to combine the valuable aspects of these simulation paradigms:

- Like mannequins, physical-virtual patients occupy real physical space, providing a sense of presence and physicality that can promote more compelling training scenarios and prompt more realistic behavior from trainees and learners.
- Being computer-based, physical-virtual patients support dynamic imagery through the use of augmented reality, allowing for the expression of complicated visual symptoms and a diversity of simulated patient appearances.
- Real humans can control physical-virtual patients, increasing the realism of speech and other behaviors.

Moreover, the increases in presence and physicality resulting from environmental awareness exhibited via touch are applicable to physical-virtual scenarios beyond patient simulation. General

virtual agents with sophisticated human-like capabilities are known as *intelligent virtual agents (IVAs)*, described as

...intelligent digital interactive characters that can communicate with humans and other agents using natural human modalities such as facial expressions, speech, gestures, and movement. They are capable of real-time perception, cognition, emotion, and action that allow them to participate in dynamic social environments [10].

With such capabilities, IVAs are natural candidates for computer-based simulations of virtual humans, including patients. However, IVA research has historically been limited to the visual and audio input/output domains [113]. Thus, among other such applications, one of our primary motivations for this research involves *extending touch input to physical-virtual patient simulators and to IVAs in general*, which is naturally handled by the presented methodology.

1.5 Matched and Mismatched Physical-Virtual Content

Our proposed method is designed to support touch interaction scenarios in which the geometry of the physical touch surface matches the geometry of the desired three-dimensional virtual content, such as a human-shaped surface with corresponding virtual imagery. While we emphasize that the ability to sense touch on such complicated surfaces is an inherent benefit of this approach, we also consider cases consisting of a mismatch between the physical and virtual geometry. Examples include the trivial case in which a user interacts with three-dimensional content on a two-dimensional touchscreen interface and situations for which no physical surface is present, which we compare to matched-geometry interactions in a human-subject study (Chapter 7).

There are two primary motivations for maintaining an explicit distinction between the geometries of the touch surface and the virtual model. First, this allows for the display of high-frequency

visual content on low-frequency objects as a means of supporting coarse animations without leading to distortions. For example, when projecting a virtual head onto a matching head-shaped surface, many head movements—such as nodding, shaking, or turning—would appear extremely unnatural in high-frequency regions such as the nose, possibly leading to breaks in presence, reducing perceived realism, or otherwise eliciting feelings of discomfort from users. With a smoother surface—perhaps one with no physical nose—reduced-distortion animations may be possible, but it may not be desirable to create a specific low-frequency virtual model when a high-frequency one is already available. In spite of such geometry inaccuracies, smoothed surfaces still promote increased realism and physicality over flat interfaces in touch interactions achieved via this proposed approach.

Additionally, this separation provides a mechanism for creating the touch-based semantic behavior of the virtual content directly in the space of that content such that it remains agnostic of the physical surface. In other words, a particular virtual model can be displayed on rear-projection surfaces of various geometries—whether matched or mismatched—and the system will interpret touches on each physical surface appropriately in the context of the virtual content.

1.6 Outline

This dissertation is structured as follows. Chapter 2 presents a survey of touch sensing implementations, ranging from capacitive hardware sensing elements to camera-based approaches. Our proposed approach is covered in detail in Chapter 3, including the design of the device-content lookup table, our algorithms for distinguishing touches from hovers, and mechanisms for achieving touch-triggered responses in the context of the virtual content. This discussion focuses on the fundamental aspects of the method, highlighting the manner in which it can be applied to a variety of non-parametric rear-projection surfaces. In Chapter 4, we present our practical implementation

of this methodology as a general software architecture. Additionally, we describe the design and construction of two physical prototypes we used to test this method on several rear-projection surfaces. Demonstrations of the method as applied to these rear-projection surfaces are provided in Chapter 5, including example photographs of touch-triggered responses and descriptions of the associated semantic content. System performance metrics are introduced and evaluated in Chapter 6, and human-subject studies based on this approach are described in Chapter 7. Finally, Chapter 8 discusses potential extensions along with ongoing and future work related to the proposed approach.

CHAPTER 2: RELATED WORK

Touch as an input modality in the field of human-computer interaction has a long and varied history. As an alternative to standard computer input devices, such as mice, touch interfaces have been studied for several decades [25,90]. This includes touch interfaces both with and without integrated graphical displays. Buxton et al. [25] list several main distinctions between touch interfaces and other input devices, including:

- No small devices that can be lost or damaged
- Consistency of position over time
- Ability to indicate multiple touches
- Integration into other equipment (e.g. desks)
- No moving parts
- Ability to partition physical input space into independent regions, whether physically (i.e. a physical template), graphically (i.e. with an integrated graphical display), or virtually (i.e. not presented to the user)
- The types of events that can be processed—e.g. whether *positioning* and *selecting* are independent, whether different degrees of pressure can be sensed, etc.
- Providing feedback to the user

Systems designed to sense and interpret touch must consider a number of important aspects, including:

- What *touch sensing technology* will be employed (e.g. capacitive sensors, IR light)?
- What *kinds of contact* can be sensed (e.g. multiple finger touch, single hand touch)?
- Are *multiple users* supported? Can they be identified uniquely?
- Are *other objects* supported? Do they cause interference?
- What *gestures* can be sensed (e.g. swipe, pinch)?
- Can *hand or finger hovers* be sensed? Can they be distinguished from touches? Do they cause interference?
- What *geometry* can be used for the touch surface (e.g. flat, spherical)? Are there shape or size limitations? Is the surface deformable or rigid?
- From what *material* is the touch surface constructed?
- How *accurate* are touch localizations?
- Does or could the system support *dynamic imagery* (e.g. via one or more projectors)?
- Has the system been *evaluated* for usability and cognitive load?
- Do users or does the surface need to be *instrumented*?
- With what *type of visual content* will users interact (two- or three-dimensional)?
- What *other modalities* are supported (e.g. audio, haptics)?
- Do the physical surface and virtual content have *matching geometry*?
- What kinds of *responses* can be triggered by touch?
- What *information* is associated with detected touch input (e.g. location, pressure)?

Due to the wide variety of touch sensing implementations and applications, many permutations of the above dimensions have been developed and evaluated. Below, we survey the literature and present many such methods, highlighting noteworthy paradigms and interesting extensions.

2.1 Touch Sensing Technology

There are a variety of methods designed to support touch interactions in interactive computer graphics applications [129]. The specific technique chosen has important impacts on the types of interfaces that are available—in terms of interaction capabilities, output modalities, supported surface geometry, and other characteristics. Generally, typical touch sensing approaches can be categorized as either *hardware-based* or *camera-based*.

Hardware-based touch sensing methods rely on capacitive sensors, resistive elements, or similar technologies; they require outfitting the surface with grids or other tight configurations of such sensing elements. While these approaches commonly assume flat surfaces, they have been successfully applied to surfaces with more complicated geometry. However, this often leads to practical considerations concerning how densely the sensing elements can be packed on high-frequency regions of the surface, which directly impacts the resolution of touch that can be sensed. Also, hardware-based approaches are often not amenable to dynamic imagery via projectors, as the hardware elements themselves might interfere with projected imagery. However, they are often advantageous in terms of space requirements: for instance, many hardware-based touch-sensitive devices, such as cell phones, are sufficiently small that they allow mobile, handheld use.

Camera-based touch sensing instead uses imagery from cameras, such as infrared (IR) cameras, depth sensors, or even marker-based motion capture systems. Imagery from cameras mounted either above or below the touch surface is continuously processed to detect touch or proximity

events. A popular approach based on frustrated total internal reflection has been commonly used to support multi-touch interactions on flat tabletop and wall surfaces, with a notable resurgence in 2005 [61]. Rear camera approaches in particular are suitable for dynamic imagery through rear projection. With the addition of rear IR lighting, camera-based methods have the ability to detect touches on more complicated surfaces, provided detected touches in 2D camera imagery can be reliably and quickly converted to their corresponding positions on the 3D touch surface. Compared to hardware-based approaches, camera-based methods tend to have increased space requirements due to the need to place cameras throughout the environment at appropriate distances to capture images of touch input on the surface. As a result, interfaces of this form are often presented on tabletop or wall surfaces. In some cases, the optical paths of the cameras can be folded to decrease the amount of space needed, such as through mirrors.

2.1.1 Hardware-Based Touch Sensing

Hardware-based touch sensing generally involves the construction of a grid or other configuration of sensors which experience a change in resistivity or capacitance when touched by a user's fingers or hands [68, 76, 90, 112, 149, 150]. Compared to camera-based approaches, these techniques require significantly less data to be processed, and the desired properties of touch events are more directly measured rather than inferred from images [68]. However, this often comes at the cost of decreased touch-sensing resolution [61], the inability to project imagery, and increased difficulties when considering surfaces with non-planar or non-parametric geometry.

2.1.1.1 Resistive Sensing

Force-sensitive resistance approaches feature arrays of sensors that experience an increase in conductance in response to increasing application of force [48, 68, 77, 98, 124, 147]. Because of this

relationship, they are capable of detecting variations in pressure. Resistive touch interfaces are generally cheaper than other hardware-based methods [45]. Unlike capacitive sensing approaches, they are capable of detecting input from devices like styli—though this limitation may be overcome in capacitive systems through the construction of special input devices (e.g. [116]).

The UnMousePad achieves multi-touch sensing through the use of resistive elements [124]. The authors call their technique *Interpolating Force Sensitive Resistance (IFSR)*, which they claim supports higher resolution than traditional resistive touch sensing approaches that are often limited in their ability to detect touches in locations between the hardware touch sensors. It is capable of detecting a large spectrum of pressure variation. The device is flexible and has been applied to cylindrical surfaces. In addition to detecting finger input, the authors also focused on high-resolution stylus sensing, noting that many other approaches are completely incapable of detecting a stylus unless contact occurs directly on one of the sensing elements.

TactileTape is a pliable resistive touch-sensitive material that can be applied to non-planar surfaces [73]. Primarily designed to allow for rapid prototyping of simple touch-sensitive objects, it is composed of a resistive and a conductive layer that forms a closed circuit when combined with user touch input. The proposed prototype is only capable of sensing the occurrence—but not location—of touch at some point on the strip. However, as it is made of common materials—a pencil, tin foil, and shelf liner—designers can quickly and easily experiment with applying it to a variety of objects to support simple touch sensing.

2.1.1.2 Capacitive Sensing

Similar to resistive touch sensing, capacitive touch sensing is generally achieved through a grid or other configuration of sensors which experience a change in capacitance when touched by fingers or hands [76, 90, 112, 149, 150]. The raw change in capacitance can be used as a measure of

touch pressure [90, 149]. Unlike camera-based approaches, capacitive sensing is not susceptible to interference from ambient lighting [143]. Also, capacitive sensing approaches typically require significantly less bandwidth than image-based ones. However, because the collection of sensors is discrete, by default touch sensing resolution can be low, and often a single touch affects the capacitance of several adjacent sensors. To improve the resolution of detectable touches, such approaches often include an interpolation scheme that considers the capacitance of many neighboring sensors [90, 149]; Westerman suggests that a 4 mm spacing of sensors can achieve a precision of 0.2 mm via interpolation [148], though it is important to note that such tightly packed grids may not always be achievable, especially for non-planar surfaces. Finally, unlike camera-based approaches, capacitive ones depend on specific electrical properties of the human body, often restricting touch input to only fingers [124]. However, it is possible to design special objects, such as styli, that can be tracked by such systems [116].

Using capacitive coupling, DiamondTouch supports touch from multiple users on a tabletop surface [43]. Location-specific electric fields are transmitted throughout the table using a series of antennas driven by a transmitter. Each user is capacitively coupled to his or her own receiver in a chair. When touch occurs, a circuit that runs from the transmitter to the table to the appropriate user's receiver and back to the transmitter is completed. Thus, detected touches can be immediately associated with the corresponding user, facilitating multi-user touch input. Depending on the array of antennas, multiple touches from a single user may be too close together to be distinguished. Interestingly, even conductive objects left on the surface do not complete the entire circuit, and so they do not interfere with DiamondTouch; the authors propose creating special objects that could be used to interact with the table. As an extension, Wu and Balakrishnan explore multi-finger and hand gesture input on DiamondTouch [154].

SmartSkin comprises a grid of capacitive sensors in the form of transmitter and receiver electrodes on a table- or tablet-sized surface [122]. Along with detecting the position of a user's hand on

the surface, it is also capable of computing the proximity of the hand to the surface. For the table-sized surface, the system is accurate to within 1 cm and is capable of sensing a hand within 5–10 cm of the surface. The authors suggest that SmartSkin is suitable for non-planar physical surfaces, though they do not explore this. Additionally, the authors experimented with applying conductive electrode barcode patterns to objects, which they refer to as “capacitance tags.” As the objects are ungrounded, they are not directly sensed by SmartSkin; when a user touches them, they become grounded and hence can be detected.

Villar et al. created an alternative computer mouse based on capacitive sensing [143]. While some similar approaches have difficulties with detecting multiple touches in close proximity, the Cap Mouse is able to sense the user’s fingertips separately. Internally, the mouse uses capacitive elements printed with conductive ink in a 5 mm grid, interpolating a touch’s position based on the specific subset of sensors whose capacitances are affected. Interestingly, the values read from the capacitive sensors are converted into a grayscale image, which is interpreted by the same touch sensing processing pipeline used by two camera-based multi-touch mice the authors created; however, as the source of this data is a conductive ink grid, we categorize this approach together with other capacitive sensing methods.

2.1.1.3 Other

Gu and Lee presented TouchString, a flexible multi-touch sensor that can be applied to objects of various shapes, including planes and cylinders [59]. Their design is somewhat general, supporting optical, capacitive, or resistive sensors. As examples, they used the TouchString structure with phototransistors to allow for multi-touch support on a cellphone frame, a plane, and a bottle, using form-fitting configurations of sensors. However, they note that the prototype is not flexible enough to be truly general purpose, and they experienced problems when repeatedly folding and unfolding

it. Also, the spatial resolution of TouchString is a limitation, as the sensors are placed around 18 mm apart. The authors considered the addition of an interpolation scheme once they reduce the cell spacing by half.

2.1.2 Camera-Based Touch Sensing

Compared to hardware-based approaches, touch sensing via camera imagery is generally capable of higher resolution touch input [61]. Such approaches are also suitable for dynamic projected imagery. Camera-based touch sensing tends to require more physical space than hardware-based approaches [124], but it can often support larger and more complicated touch surfaces.

2.1.2.1 Frustrated Total Internal Reflection

A popular method for camera-based touch sensing relies on a concept called *frustrated total internal reflection (FTIR)*. A light wave propagating in a medium with a higher refractive index than an adjacent medium at a sufficient angle of incidence is entirely reflected, known as total internal reflection [46, 61]. Another medium located sufficiently close to this boundary can disrupt (or *frustrate*) the internal reflection, causing the light to escape. Such light can be captured by cameras or other light sensors and interpreted to perform touch sensing [79, 135, 151]. However, FTIR does pose certain drawbacks. Most importantly, touch applications relying on FTIR are limited to either flat surfaces or surfaces with minimal curvature [125]. The traditional FTIR approach is not capable of detecting hovers, though this may be a benefit in some scenarios. Other IR light sources, such as sunlight or other optical tracking systems, can interfere with FTIR-based routines [44]. Additionally, fast hand movement can lead to intensity decreases in the camera imagery, which can make contact detection more difficult [74].

FTIR has been used to create images of fingerprints [53], including even as early as 1965 [155]. As a natural extension to this idea, Johnson and Fryberger obtained a patent in 1972 describing the use of FTIR to support touch input on a cathode ray tube (CRT) [79]. Other FTIR touch-sensitive CRT systems have been developed by Mallos [100] and Kasday [83], and similar methods have been designed to support CRT drawing applications via fingertip, brush, and pen input, either using photodetectors [109] or cameras [58] to capture the frustrated light. Han repopularized the use of FTIR-based touch sensing in 2005, noting several benefits of the technique: low cost, high resolution, high accuracy, and the ability to integrate with dynamic rear-projection graphics [61]. Davidson and Han later integrated touch pressure information into this approach, allowing users to “tilt” graphical objects by selectively applying pressure to place them on top of or below other ones [40].

Some researchers have investigated ways to augment traditional FTIR-based touch sensing approaches. Echtler et al. combined a standard FTIR multi-touch table with an overhead light source that allows for the detection of shadows as an additional input modality [46]. The traditional side-mounted IR LEDs used for touch sensing and the ceiling-mounted IR light for shadow tracking are enabled in an alternating fashion; as such, a single input comprises a contact image and a shadow image taken on consecutive frames. When the system detects a shadow, it places a cursor at its peak, shifted by some amount to prevent occlusion from the user’s hand. Once the user’s finger contacts the surface, a click is triggered at the cursor’s location. This is analogous to mouse input in computer interfaces, where placing the cursor over an object and selecting that object by clicking are two separate events. In a human-subject study, participants touched squares displayed on the table, either with or without the shadow-based cursors. With the cursors, participants were more accurate in touching the squares, but this doubled the amount of time required.

Iacolina et al. also experimented with shadow detections in an FTIR setup on a flat tabletop [74]. However, their prototype relies on natural, uncontrolled IR light in the user’s environment, such as

sunlight, rather than additional controlled IR light sources. The lack of control does not cause major issues: the standard IR tracking works better when there is less sunlight, and the supplemental shadow tracking performs better when there is more. As in similar setups, a primary motivation is the ability to detect proximity to the touch surface.

Similarly, Dohse et al. added an overhead camera to a traditional FTIR setup to perform hand tracking [44]. They had two primary motivations: assigning touch ownership to the appropriate user in a multi-user environment and making the system less susceptible to IR noise due to poor lighting conditions. The hand-tracking camera could additionally be used to recognize gestures above the surface, though the authors do not explore this.

Schöning et al. describe *interscopic multi-touch surfaces (iMUTS)*, which leverage both 2D and 3D interaction techniques with both monoscopic and stereoscopic visual content [130]. Their prototype comprises an FTIR-based multi-touch wall. They describe two interscopic interaction techniques. The *Windows on the World* metaphor presents a 2D overlay over 3D content; users can navigate in three dimensions throughout the virtual world via various touch interactions on the window. The second interaction technique allows users to cut and subsequently deform the 3D volume of the presented data by selectively dragging on certain parts of the wall. To simplify the interface, predefined cutting templates are provided.

While FTIR-based approaches are generally limited to flat surfaces, Weiss et al. designed Bend-Desk, an interactive hybrid desk featuring a horizontal tabletop and a vertical board connected by a curve [145]. The imagery from three cameras is interpreted to detect touches across the entire surface, including the curve. The authors' findings indicate that users tended to think about the surface in terms of its three constituent components rather than as a single one, and they often avoided interacting "across" the curve. Likewise, Villar et al. created a curved multi-touch mouse based on FTIR [143]. The surface of the FTIR Mouse is a smooth arc that still supports total internal

reflection. However, due to the shape and camera placement, the mouse is only capable of sensing touch in a small area toward the front.

Using FTIR, Roudaut et al. investigated how surface convexity and concavity affect touch input [125]; due to light bleeding, they made a few modifications to their curved surfaces, including applying silicone spray and manually smoothing the surfaces to reduce IR hotspots. They found that pointing accuracy increases as surface convexity increases and that users were less accurate when touching targets on concave surfaces.

2.1.2.2 *Rear Illumination*

In an alternative approach, known as *rear illumination (RI)*, IR light sources are placed below the touch-sensitive surface. Some amount of IR light passes through the surface; when an object, such as a finger or hand, comes in contact with the surface, this light is reflected and sensed by IR cameras [129]. In some RI setups, a diffuse material is applied to the touch surface, referred to as *rear-diffused illumination (RDI)*. In general, setups without a diffuser are better able to interpret events that occur beyond the display surface, though RDI approaches do not entirely prevent this. This separates RI methods from FTIR-based approaches, which generally cannot detect hover events without the addition of supplemental sensors. Furthermore, this allows RI approaches to detect physical objects, whether through shape cues or fiducials (e.g. [32]). While RI is commonly used to support touch sensing on flat surfaces, it has also been applied to a variety of non-planar surfaces and even to deformable surfaces.

The HoloWall comprises a glass wall and a rear-projection sheet on which a projector displays imagery [103]. Alongside the projector are an array of IR LEDs and a camera. Users can interact with the other side of the wall via touch. The system is capable of recognizing two-dimensional

barcodes printed onto objects. As IR light is reflected by objects within 30 cm of the wall, the authors also propose recognizing and responding to user proximity in addition to touch.

TouchLight uses two cameras in an RI configuration to support touch on a holographic film material applied to a flat non-diffuse display surface [153]. The system relies on various image processing routines to reconcile the images from the two cameras in order to determine if contact with the display surface has occurred. The display surface also has an attached microphone to detect when users tap on it; the response to such an event depends on where it occurs. Because the display surface does not have a diffuser, the cameras could potentially perform facial recognition on users or allow for other input techniques. For instance, the physical display surface and the virtual plane on which interactions occur can be spatially decoupled so that hovers are accepted as input instead of touches.

Benko et al. proposed several techniques they called *Dual Finger Selections* to enable more precise and accurate interactions on touchscreens [8]. Their multi-touch tabletop uses a standard RDI setup. All but one of their selection techniques provided a cursor offset from the user's touch position to prevent occlusion. However, the single technique that did not provide an offset was both the most preferred and highest performing method. Known as *Dual Finger Stretch*, it allows users to magnify part of the user interface with one hand while making a precise selection with the other.

RDI-based touch sensing has been applied to more complicated surfaces than walls and tabletops, such as spherical displays. One known as Sphere supports multi-touch input from multiple users on a diffuse spherical display surface [7]. An IR camera and a visible-light projector share the same optical axis through the use of a cold mirror. After IR camera imagery is processed to detect finger or hand contact, it must be mapped onto the spherical surface in order to relate these touch events to appropriate locations in the projection imagery. Additionally, visual content projected

onto the sphere is pre-distorted so that it appears undistorted on the sphere. The authors provide interesting details regarding content creation and input handling on spherical surfaces. For example, to interpret touch events such as dragging an object, Sphere treats movements as quaternions, which requires care when designing intuitive user interfaces. Objects dragged a sufficient distance are also automatically oriented, as otherwise they may appear “upside down” or in an unnatural orientation. Unlike in scenarios with shared walls or tabletops, users can only see approximately half of the display at once, allowing for an easy separation between the workspaces of multiple users.

While some touch-sensing interfaces reduce contact information down to single points or gestures, ShapeTouch considers the entire contact area to allow for rich interactions on a rear illumination tabletop surface [27]. The associated touch sensing system also computes optical flow fields between input frames, combining them with shape contact regions. Contact area is used to compute the amount of “virtual force” users exert on the table. Based on the flow vectors, forces are classified as pressing, colliding, or friction. The amount and type of force affect the results of various gesture interactions. For instance, users can drag and rotate an object freely, or they can exert a pressing force on part of the object to anchor it in place, essentially choosing the origin of rotation.

To address some of the shortcomings of their FTIR Mouse, Villar et al. also proposed an RDI-based multi-touch mouse, called Orb Mouse [143]. The mouse has a hemispherical, diffuse surface, and touch sensing is achieved through a camera and four IR LEDs housed inside. Similar to touch detection on Sphere, interpreting touch on the hemispherical shape of the mouse requires undistorting the camera imagery. Though the RDI paradigm allows the mouse to take on a curved shaped and supports touch input across the entire surface, the authors found that the camera imagery contained more noise than their FTIR Mouse.

Similarly, the Side Mouse by Villar et al. has a largely RI-based design, comprising a camera and an IR-laser illuminator that projects a “sheet” of light outwards [143]. Unlike with the other multi-touch mice they proposed, interaction with this mouse is performed directly on the table surface: fingers that touch the table break the IR beam and can be sensed by the internal camera. Given careful positioning of the beam, the mouse is not susceptible to interference from fingers hovering just over the table. Users rest their palm on the relatively small base of the mouse, which uses an ordinary mouse sensor to detect mouse movement. To trigger mouse clicks, users press down on the mouse itself with their palm, which activates buttons inside the mouse. The interaction area is somewhat unconstrained and not limited to the device itself, potentially allowing for richer bimanual interactions; as a consequence, the Side Mouse may detect unwanted objects located in front of it.

Valkov et al. investigated the manner in which users touch 3D stereoscopic objects on a 2D RDI-based multi-touch wall [142]. In such paradigms, each eye receives a specific customized image, which can lead to challenges when users try to select a 3D object by touching a 2D surface. To allow for view-dependent rendering of the 3D content, they used an optical tracking system to track the heads of users. They found that users generally touched the surface between the two eye projection images, closer to the dominant eye projection image. Bruder et al. considered a similar question on an RDI-based multi-touch table, which they augmented with motion tracking cameras to support free-space 3D “touch” in a human-subject study [23]. In a 3D touch condition, participants placed their index fingertip at the center of 3D target spheres displayed stereoscopically; in a 2D touch condition, they were asked to move their finger “through” the 3D target sphere until reaching the 2D touch surface. In the 2D touch condition, subjects either touched the center of the two eye projections or touched one of the eye projections itself (either the dominant or non-dominant eye). Their results further suggest that this 2D touch technique is an appropriate substitute for 3D touch when users select objects within 10 cm from the display surface. In a sim-

ilar Fitts' Law experiment, they found that the performance of this 2D touch technique decreases faster than performance for the free-space 3D touch technique as the height of the virtual objects increases [24].

As an additional application of RI-based touch sensing to non-planar surfaces, Stevenson et al. created a deformable multi-touch surface that can be dynamically inflated or deflated using air pumps [133]. The touch surface is made of rubber latex, suspended on a cylindrical frame that houses the projector and an infrared camera. IR LEDs are placed along the top of the cylinder to support touch sensing. Using both internal and external pumps, the rubber latex material can be inflated or deflated, ranging in shape from a flat circle to a convex or concave hemispherical surface. Because the material is compliant, finger touches can cause further deformations; the authors suggest examining the size of the finger blobs in the camera imagery as an additional dimension of touch input. Similarly, using a combination of latex and spandex, Bacim et al. created two deformable surfaces: flat and hemispherical [6]. The surfaces support touch sensing in a standard RDI setup. In addition to comparing the two surface shapes, they also examined how visual and tactile feedback affect the accuracy of users in deforming the surfaces to a prescribed distance. Visual feedback outperformed tactile feedback, allowing for sub-millimeter precision. Additionally, both selecting targets and accurately creating deformations was more challenging on the hemispherical surface.

Bolton et al. compared user performance on competitive and cooperative tasks on both spherical and flat multi-touch RDI surfaces [17]. To ensure that participants were not influenced by eye contact and facial expressions, they purposefully designed their spherical surface to be large enough to prevent participants from seeing one another. They also experimented with two flat wall surfaces: one with no obstruction and one with a physical divider between the two users' workspaces. Participants were provided several UI-based methods of "peeking" at their partner's workspaces. While task performance on the unobstructed flat display was significantly different than performance on

the spherical surface, they found no significant differences in performance between the divided flat and the spherical display, suggesting that performance differences are due to the separation of users and their workspaces and not due to the display format.

PAPILLON is a technique based on 3D printed optics that supports both display and touch sensing on small curved surfaces [21]. The source of the image, which could be a projector, LCD screen, or similar, is decoupled from the display surface itself: the image passes through bundles of 3D light pipes that can be arranged in the desired shape. As the light pipes are bidirectional, the surface also supports RDI-based touch sensing; fingers within roughly 10 mm from the surface are visible. They experimented with creating interactive eyes for small toy characters. While they suggest that PAPILLON is suitable for arbitrary eye shapes, they do not discuss size limitations.

2.1.2.3 Other

To augment either FTIR or diffused-illumination touch sensing systems, Dang et al. devised a heuristic-based approach that determines the position and orientation of a user's fingers in order to map these fingers to specific hands [38]. In a human-subject study featuring a Microsoft Surface, they were able to successfully map fingers to hands in 97.5% of the recorded frames using RDI techniques.

GelForce comprises a rectangular silicone rubber medium with two layers of suspended spherical markers [81]. Below, a camera images the markers. As users touch the planar surface, the markers are displaced; camera imagery of this displacement is used for touch sensing. The amount of displacement can be used as a measure of pressure. The silicone medium need not be planar.

The Visual TouchPad uses downward-facing cameras oriented toward a planar surface, relying on stereo hand-tracking to detect touches on the surface [99]. As there are no occlusions between

the cameras and the user's hands, unlike in RI-based approaches, detecting and interpreting hand gestures over the surface is simpler in this paradigm. As the system relies on homography-based matching between the touch surface and the display surface, it is only applicable to planar interfaces.

CHAPTER 3: METHODOLOGY

In general, the overall goal of any system supporting touch as an input modality can be stated as **the conversion from *user touch input* to *system rendering output***. This process begins with the acquisition and subsequent processing of input data that is used to determine the presence, location, and nature of touch input—*sensing touch*—and ends with the initiation of semantically relevant responses to these touches provided in an appropriate modality to the user—*responding to touch*. Our proposed methodology addresses both of these challenges.

Following the traditional rear-diffused illumination (RDI) paradigm, our physical prototypes (described in Chapter 4) comprise infrared cameras to detect touch input for interaction with dynamic virtual content provided by projectors and speakers. When a touch surface can be mathematically parameterized, the conversion of two-dimensional camera imagery to coordinates on the virtual content is straightforward, as the mapping between them can be obtained via simple analytic equations. However, our proposed method is designed to be a generalizable approach for supporting tightly integrated touch sensing on rear-projection surfaces that are non-parametric, where such analytic equations are not available [69, 71, 72]. It is capable of detecting and localizing multiple simultaneous finger touch events across such surfaces and establishes a tight loop between the touch sensing and rendering systems, leading to highly responsive interactions.

There are several contributions we present over typical RDI implementations:

- **Non-parametric surfaces:** Our method supports camera-based touch sensing and response on a large set of non-parametric surfaces. Touch input is permitted at any location on the surface that at least two cameras are capable of reliably imaging—that is, without extreme distortions due to oblique camera angles. This precludes, for instance, objects with compli-

cated self-occlusions. However, this is not a major limitation, as such objects do not support rear-projection imagery for the same reason. In addition, this method allows for touch input on parametric surfaces.

- **Registration of rendered content:** The same mechanisms used to achieve touch sensing support the registration of projected virtual imagery onto these surfaces.
- **Touch/hover classification:** Through the use of multiple cameras, our method can filter near-contact events—called “hovers”—from touches. With simpler surfaces, such as planes, intensity thresholding may suffice; however, on non-parametric surfaces, achieving uniform rear-infrared lighting is generally a more challenging problem. We present and evaluate two algorithms for performing touch/hover classification in Section 3.2.2. Both consider the fundamental difference between touches and hovers: by definition, touches occur *on* the surface, and hovers occur *off* of it. The second algorithm, a plane sweep approach (Section 3.2.2.2), is further capable of estimating hover-surface distances, supporting interactions above the surface.
- **Multiple projectors:** Our method is capable of handling multiple projectors, allowing for touch sensing with registered projected imagery across larger surfaces.
- **Semantic response:** Through a separate semantic content engine, our method has a dedicated mechanism for assigning semantic meaning to touches in the context of dynamic interactions with virtual content. Semantic touch responses can be presented to the user in a variety of modalities, including visual and audio.
- **Evaluation:** In Chapter 6, we provide internal consistency metrics that represent the expected accuracy of touch sensing across an entire touch-sensitive surface. Moreover, we evaluate the localization accuracy of user touch input and the classification accuracy of touch and hover input.

In this chapter, we provide a detailed description of our proposed methodology for achieving touch sensing and semantic response on non-parametric rear-projection surfaces. We begin with an overview that highlights the fundamental components of this method. In particular, we present the design and construction of a *device-content relational lookup table* that stores relationships between the optical devices (the camera and projectors), the touch-sensitive surface, and the virtual content. Next, we focus on two algorithms for distinguishing touch from near-contact hover events by utilizing the lookup table relationships. We describe mechanisms for defining semantic meanings within the virtual content to touch input and for augmenting this lookup table to support the tight registration between touch detection and rendering output across several output modalities. While the overall discussion is general and thus independent of a particular touch-sensitive surface, we include figures of specific implementations where useful. Finally, we briefly discuss an extension to the overall approach that supports registered virtual imagery provided by optical see-through head-mounted displays as opposed to rear-mounted projectors.

3.1 Overview

Figure 3.1 shows a high-level overview of our method, demonstrating the overall flow between user touch input and system rendering output on a touch-sensitive physical-virtual surface. This is facilitated by three decoupled system components:

1. The **touch sensing system** comprises several infrared (IR) cameras and light sources. The camera imagery is analyzed to *detect touch input* on the touch-sensitive surface. Additionally, the touch sensing system processes detected touches to extract relevant information used to generate and display output to the user, which is sent to the semantic content engine.

2. The **semantic content engine** is a standalone component of the system that receives touch input from users and determines appropriate stimulus output in the context of a particular interaction scenario. Specifically, the semantic content engine is responsible for *assigning semantic meaning to touches*, through the specification of the locations on the virtual model that can be affected by touch and the exact responses that touches trigger in these regions. Depending on the needs of the scenario, a semantic content engine can range from simple projector-space-based interactions—such as the direct manipulation of projected imagery colors through touch—to sophisticated three-dimensional graphical models augmented with animations, audio effects, and other capabilities. The mapping between detected touches and semantic output is created directly in the two-dimensional coordinate space of the projectors in the former case and directly in the three-dimensional coordinate space of the virtual content in the latter case. In addition to touch-triggered responses, the virtual content can also be controlled by a human-in-the loop operator or artificial intelligence.

Furthermore, the semantic content engine maintains the current mode or state of the interaction, which may affect how touch input is processed. For instance, a touch at a particular location may prompt an animation in one semantic state or an audio response in another, and touch interactivity may depend on the current user interface object selected by the user. As with the other interactive capabilities, semantic state can be updated through touch, human control, or other such processes.

3. Finally, the **rendering system** is responsible for displaying the virtual content to the user by *realizing semantic touch output* as determined by the semantic content engine. It is composed of projectors to display registered imagery on the physical-virtual surface, speakers to play audio output, and similar equipment for other output modalities.

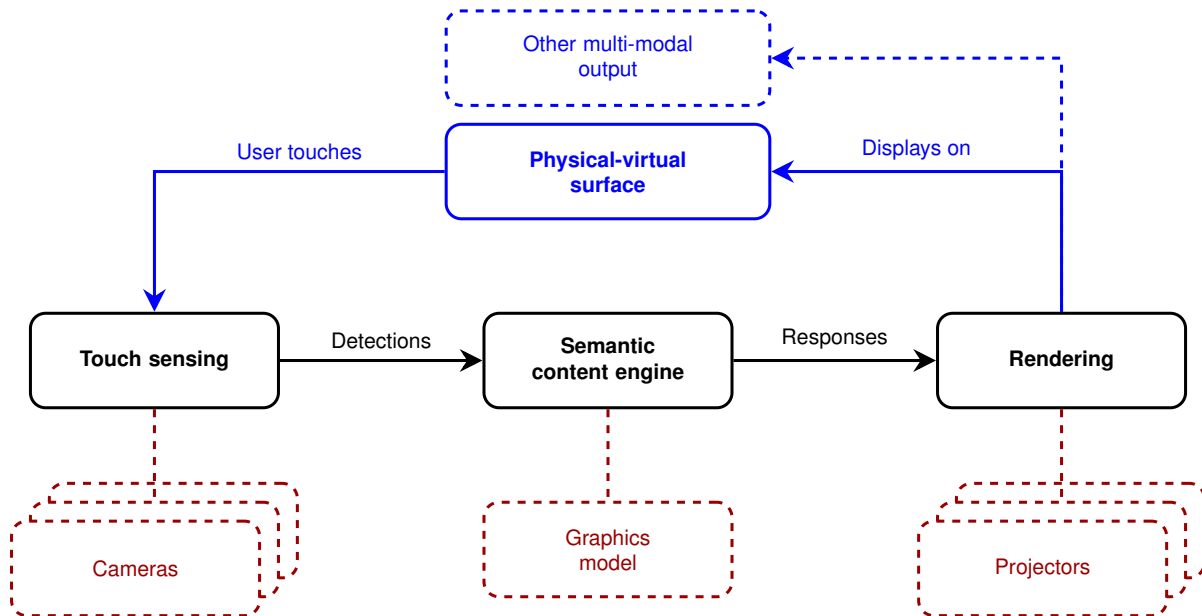


Figure 3.1: High-level overview of the proposed method. The **touch sensing system** continuously processes camera imagery to detect user touch input on a physical-virtual surface. Detected touches are sent to the **semantic content engine**, which determines appropriate responses in the context of a stateful 3D virtual model. Multi-modal output to the user is provided by the **rendering system**, including virtual imagery projected onto the surface, audio from speakers, and potentially other output devices.

Decoupling functionality into separate dedicated system components provides several advantages. First, it allows for the independent, parallel processing of touch input and selection of rendering output, reducing system latency. While the touch sensing and corresponding system output routines are designed to be general, a particular interaction scenario comprises specific virtual content and touch capabilities, so this design provides increased modularity and flexibility in terms of supporting a desired scenario without imposing restrictions on touch input detection. In certain situations, no graphical model is required. For example, a touch-controlled painting application allowing users to select a desired color via a palette and draw directly on a non-parametric surface can function without an associated graphical model of the surface. In this case, the semantic content engine must only consider which touch inputs change color and which produce 3D painted

contours on the touch surface; such interactions only depend on the two-dimensional coordinate spaces of the projectors. In contrast, displaying an anatomical model of a human body on a correspondingly shaped surface with supported touch interactions must operate on an associated 3D virtual model. However, the touch sensing system remains agnostic to the scenario—instead, it is solely responsible for detecting touches on the surface and transmitting them to the semantic content engine, which subsequently maps touches to appropriate output.

The touch sensing system, semantic content engine, and rendering system are linked together via a relational lookup table architecture, described below.

3.1.1 *Lookup Table*

The proposed method is designed to support touch sensing on surfaces for which the relationship between camera images of user touches and the virtual content cannot be directly described by parametric functions. In order to map touches detected in two-dimensional camera imagery to their corresponding positions on the three-dimensional virtual content, we construct a **device-content relational lookup table** that links these coordinate spaces. This reduces coordinate mapping to efficient $O(1)$ indexing operations—for instance, a particular camera pixel (x, y) can be used as an index to retrieve the corresponding 3D point (X, Y, Z) on the virtual model via a direct lookup. Essentially, the lookup table performs the role of the parametric functions that are not available for general non-parametric surfaces. However, lookup tables are still suitable for parametric surfaces; in fact, the equations describing a parametric surface could be used to populate the lookup table, which is advantageous when computing these equations is more costly than lookup table retrievals. Instead, to maintain the generalizability of the method, we populate our lookup table through the observation and processing of camera-projector correspondences in a *preprocessing phase*, described in Section 3.2.

Moreover, the lookup table can be augmented with additional information that facilitates rapid dynamic responses to touch input by directly encoding associations between camera coordinates and touch-triggered animations, audio effects, or other semantic responses. In this way, mapping touch input to semantic output can also be achieved through $O(1)$ indexing operations.

This paradigm frontloads execution time through an initial preprocessing and associated data storage phase in order to facilitate highly responsive touch sensing and response at runtime. Once the lookup table is constructed, conversions of a coordinate from one space to another and mappings from touch input to semantic responses are achieved through trivial lookup operations, limiting the amount of required runtime processing. This is analogous to image and video compression as a means of reducing file size: while encoding (i.e. preprocessing) can be a time-expensive operation, decoding (i.e. runtime processing) is comparably efficient.

In summary, encoding coordinate space relationships and other information in a lookup table provides three principal advantages in the context of tightly integrated touch sensing and response:

1. Lookup table accesses are *efficient*, requiring only $O(1)$ indexing operations.
2. Lookup table entries provide coordinate space correspondences for *non-parametric surfaces*, which may not be available otherwise.
3. Lookup table entries can be further augmented to directly encode *semantic responses* as efficient $O(1)$ indexing operations, further reducing computation at runtime.

As a means of linking the touch sensing system, semantic content engine, and rendering system together, the lookup table relates coordinates and semantic information among four fundamental types of coordinate spaces: the coordinate spaces of the cameras, those of the projectors, that of the physical surface, and that of the virtual content displayed on the surface. Conceptually, it is a

collective function of the cameras, projectors, and the touch-sensitive surface. More specifically, the lookup table is influenced by the number and resolutions of the cameras and projectors and by the relationship between the positions of the cameras, projectors, and surface. To facilitate the discussion of these coordinate spaces, we adopt a consistent naming convention: each coordinate space is assigned a three-letter abbreviation followed by a number indicating its dimensionality—2 for two-dimensional spaces and 3 for three-dimensional ones.

The two-dimensional coordinate spaces of the camera imagery are collectively denoted CAM_2 . Detected touches are localized as 2D (x, y) coordinates (denoted \mathbf{x}) within each camera’s coordinate frame. Given \mathcal{C} infrared cameras, there are \mathcal{C} such 2D coordinate spaces. Similarly, the two-dimensional coordinate spaces of the projected imagery are denoted PRO_2 . Ultimately, the visual stimulus presented to the user is a set of 2D images projected onto the touch-sensitive surface. Through the lookup table, 2D (x, y) camera coordinates can be converted to 2D (u, v) projector coordinates (denoted \mathbf{u}), allowing graphical updates directly within the projected imagery. Given \mathcal{P} projectors, there are \mathcal{P} such 2D coordinate spaces.

The three-dimensional coordinate space of the graphical content displayed to users is denoted GFX_3 . The graphics mesh G is composed of vertices \mathbf{V}^G of the form $(X, Y, Z)^G$. Other points in GFX_3 that exist on the faces connecting these vertices are denoted $\mathbf{X}^{\text{GFX}_3}$. (The distinction between *mesh vertices* and *mesh face points* will later be important for facilitating certain graphical touch responses, as discussed in Section 3.3.) Touches detected within the camera imagery can be converted via the lookup table to 3D graphical model points to trigger appropriate responses, such as activating animations and sound effects.

Additionally, there is an internal three-dimensional coordinate space containing a model of the touch surface as scanned by the cameras during a calibration procedure, representing its relationship to the joint extrinsic calibration of the cameras and projectors. This space is denoted TCH_3 ,

with the 3D scan of the surface S composed of points \mathbf{V}^S of the form $(X, Y, Z)^S$. Depending on the interaction scenario, it may be useful to expose this coordinate space to the rendering system. Our system does not assume that the graphical model G and the touch surface S are identical or even similar. One benefit of maintaining this distinction is that all graphical modeling and region-based semantic mappings between touches and responses can be done on a 3D graphical model that is designed to be easily topologized and animated, and the resulting model can be used on any number of non-matching touch surfaces.

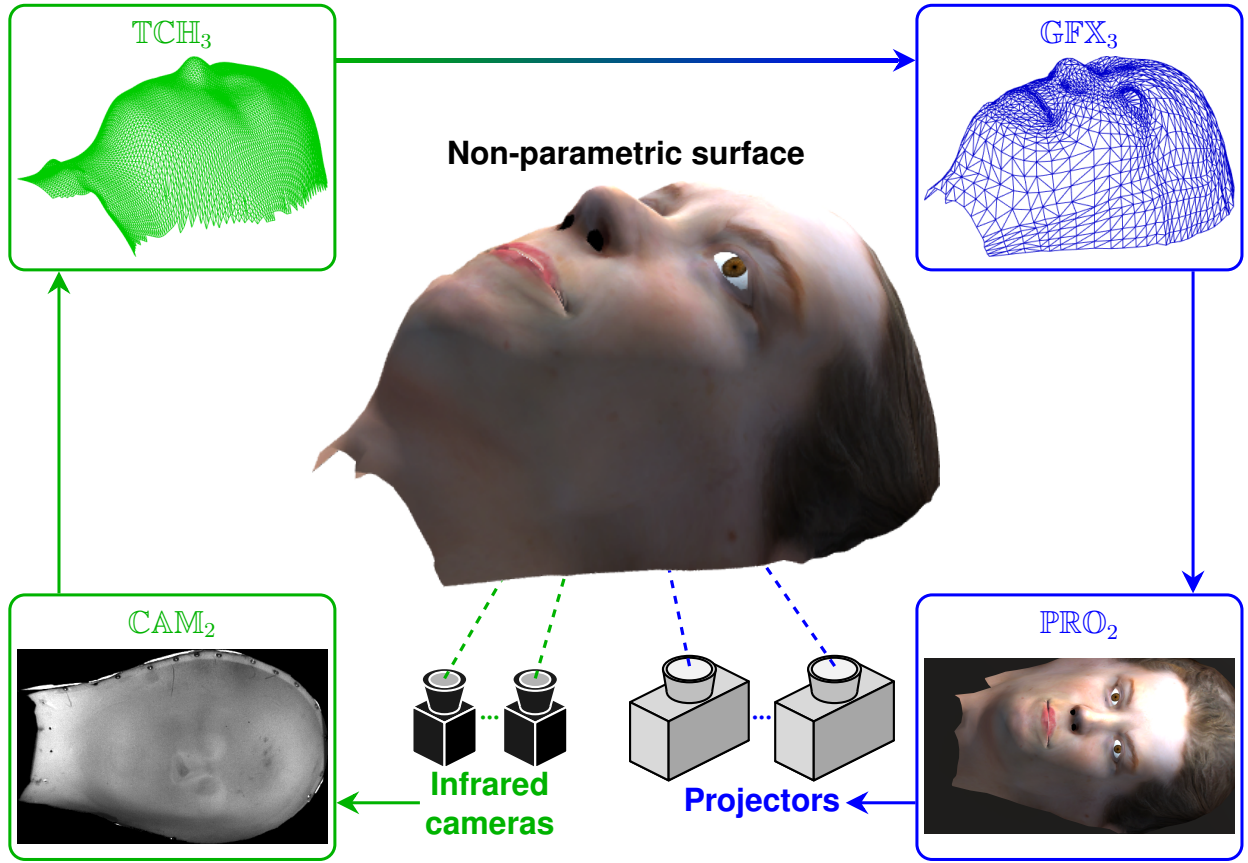


Figure 3.2: Relationships among the coordinate spaces used to achieve touch sensing and response, using imagery of a physical-virtual head surface to aid in the discussion. Imagery from the IR cameras (in CAM_2) is used for touch detection. Detected touches can be converted to 3D points on the scanned touch surface S (in TCH_3), 3D points on the 3D graphical model G (in GFX_3), and 2D projector pixels (in PRO_2) via the lookup table in order to trigger appropriate output.

Figure 3.2 shows the relationships among these various coordinate spaces, shown in the context of a physical-virtual head surface. Here, we show the internal touch surface coordinate space (\mathbb{TCH}_3). The lookup table, created during the *preprocessing phase*, stores correspondences among these spaces. Hence, when a touch is detected in the 2D camera imagery at *runtime*, it can be instantly converted to its equivalent location in the space of one of the projectors or on the 3D model, as necessary, in order to trigger appropriate responses.

Table 3.1 shows an example correspondence lookup table. Each row contains a single correspondence among all of the camera coordinate spaces, all of the projector coordinate spaces, the graphical model G in \mathbb{GFX}_3 , and the touch surface scan S in \mathbb{TCH}_3 . For example, the i th correspondence relates the camera coordinate $(x_i, y_i)^{C_1}$ in camera C_1 to $(x_i, y_i)^{C_2}$ in C_2 , to projector coordinate $(u_i, v_i)^{P_1}$ in projector P_1 , to $(X_i, Y_i, Z_i)^{\mathbb{GFX}_3}$ on the graphical model G , to $(X_i, Y_i, Z_i)^{\mathbb{TCH}_3}$ on the scan S , and so on. Certain entries may be blank, representing that a given coordinate in one space does not have a correspondence in other; for instance, due to the physical placement of cameras, camera C_1 may image a different portion of the physical-virtual surface than camera C_2 , so coordinates in the former might not have a correspondence in the latter. These blank entries still provide useful information and so are retained in the table. The construction of the lookup table, described next, begins with a collection of sparse correspondence observations, but the final table contains correspondences among *all* camera and projector coordinates.

Table 3.1: Example lookup table with correspondences relating coordinates in the cameras, projectors, graphics model, and touch surface. Each row contains a single correspondence among the spaces.

Row	Touch Sensing: Cameras			Rendering: Projectors			Semantic Content Engine	
	C_1	\dots	C_C	P_1	\dots	P_P	Graphics model	Touch surface
1	$(x_1, y_1)^{C_1}$	\dots	$(x_1, y_1)^{C_C}$	$(u_1, v_1)^{P_1}$	\dots	$(u_1, v_1)^{P_P}$	$(X_1, Y_1, Z_1)^{\mathbb{GFX}_3}$	$(X_1, Y_1, Z_1)^{\mathbb{TCH}_3}$
2	$(x_2, y_2)^{C_1}$	\dots	$(x_2, y_2)^{C_C}$	$(u_2, v_2)^{P_1}$	\dots	$(u_2, v_2)^{P_P}$	$(X_2, Y_2, Z_2)^{\mathbb{GFX}_3}$	$(X_2, Y_2, Z_2)^{\mathbb{TCH}_3}$
\vdots	\vdots	\ddots	\vdots	\vdots	\ddots	\vdots	\vdots	\vdots
n	$(x_n, y_n)^{C_1}$	\dots	$(x_n, y_n)^{C_C}$	$(u_n, v_n)^{P_1}$	\dots	$(u_n, v_n)^{P_P}$	$(X_n, Y_n, Z_n)^{\mathbb{GFX}_3}$	$(X_n, Y_n, Z_n)^{\mathbb{TCH}_3}$

Note that the correspondences in the table are *induced by and bound to the touch-sensitive surface*. In other words, the existence of a correspondence between coordinate $(x_i, y_i)^{C_1}$ in camera C_1 and coordinate $(x_i, y_i)^{C_2}$ in camera C_2 indicates that these two cameras image the same touch surface point $(X_i, Y_i, Z_i)^{\text{TCH}_3}$ at these respective pixels. Conceptually, each camera and projector coordinate describes a three-dimensional ray through the respective image plane that passes through the surface; the correspondences in the lookup table reflect these ray-surface intersections. This representation has important consequences for distinguishing touches—which occur *on* the surface—from hovers—which occur *off* of it. In particular, correspondences among camera imagery of a potential touch or hover event that exhibit a high degree of consistency or similarity indicate a common set of ray-surface intersections, which provides evidence of a touch; however, inconsistent correspondences indicate disjoint sets of ray-surface intersections, which provides evidence of a hover.

3.2 Sensing Touch

As described previously, the touch sensing system operates independently from the semantic content engine and the rendering system. All three components are together linked by the lookup table. Here, we describe the initial construction of this lookup table, restricting our focus to the subset of the correspondences that facilitates core touch sensing functionality. This includes relationships between coordinates in the cameras, projectors, graphics model, and touch surface scan (as in Table 3.1); for limited applications, this data might be all that is necessary. Additional augmentations to the lookup table that support certain kinds of semantic touch interactivity—such as audio and animations within the 3D virtual model—are separately presented in Section 3.3 as incremental updates to the lookup table.

The lookup table is created as part of the *preprocessing phase*. Later, at *runtime*, it is used both to detect user touch input and to process that input for transmission to the semantic content engine and ultimately the rendering system to display output to the user.

3.2.1 *Preprocessing Phase*

At a high level, the *preprocessing phase* is primarily concerned with learning the geometric relationships between the cameras, projectors, touch-sensitive surface, and graphical model to facilitate construction of the device-content relational lookup table. It begins with a calibration routine to determine the positions, orientations, and intrinsic parameters of the optical devices. Initial camera calibration matrices are estimated through image captures of calibration pattern objects. Following this, a projector-based feature scanning procedure is performed that provides information linking some subset of camera pixels to their corresponding pixels in the projectors. An initial 3D point cloud representing the touch surface is then estimated. These 3D points and their associated projected pixels are used for an initial calibration of the projectors. The calibration routine ends with a global optimization of the touch surface point cloud and the camera and projector calibration matrices. Dense correspondences relating all camera and projector pixels to the touch surface and to the graphics mesh are computed from these calibration matrices and stored in the lookup table for use in touch detection at runtime.

Before describing this process in more detail, we first review the nomenclature used throughout the following discussion:

- **Cameras and projectors.** There are \mathcal{C} cameras used for touch detection and \mathcal{P} projectors that display virtual content on the touch-sensitive surface. A general camera is denoted C ,

and the entire collection of \mathcal{C} cameras is represented by the set $\{C_1, C_2, \dots, C_C\}$. Similarly, the \mathcal{P} projectors are labeled $\{P_1, P_2, \dots, P_P\}$.

- **Camera pixels.** A camera C has an image plane composed of pixels \mathbf{x}^C with two-dimensional coordinates $(x, y)^C$ in \mathbb{CAM}_2 . The width and height of the camera image are given by w^C and h^C , respectively. The entire collection of $k = w^C h^C$ pixels in the image plane of C is represented by the set $\{\mathbf{x}_1^C, \mathbf{x}_2^C, \dots, \mathbf{x}_k^C\}$, denoted more compactly as $\{\mathbf{x}_i^C\}_{i=1}^k$. When clear, we will further use the shorthand $\{\mathbf{x}_i^C\}$ to refer to this collection of k camera pixels across the image plane of C .
- **Projector pixels.** Likewise, a projector P has an image plane comprising a set of pixels \mathbf{u}^P with the 2D coordinates $(u, v)^P$ in \mathbb{PRO}_2 . The shorthand notation $\{\mathbf{u}_i^P\}$ represents the entire set of pixels in the projector's image plane with resolution $w^P \times h^P$.
- **Touch surface mesh.** The three-dimensional touch surface mesh is denoted S , which contains 3D vertices $\mathbf{V}^S = (X, Y, Z)^S$ in the calibration space \mathbb{TCH}_3 . The mesh has associated edge connectivity among its vertices that defines faces. Points on this mesh that are not actual vertices are denoted $\mathbf{X}^{\mathbb{TCH}_3} = (X, Y, Z)^{\mathbb{TCH}_3}$.
- **Graphics mesh.** Similarly, the three-dimensional graphics model is denoted G , which is a set of 3D vertices $\mathbf{V}^G = (X, Y, Z)^G$ in the graphics space \mathbb{GFX}_3 and associated edge connectivity. Points on the faces of this mesh are labeled $\mathbf{X}^{\mathbb{GFX}_3} = (X, Y, Z)^{\mathbb{GFX}_3}$. The calibration space \mathbb{TCH}_3 and the graphics space \mathbb{GFX}_3 are not the same.
- **Correspondences.** The subscripts of each pixel or three-dimensional coordinate are identical for corresponding coordinates: that is, camera pixel \mathbf{x}_i^C in camera C corresponds to projector pixel \mathbf{u}_i^P in projector P , to touch mesh point $(X_i, Y_i, Z_i)^{\mathbb{TCH}_3}$, to graphics mesh point $(X_i, Y_i, Z_i)^{\mathbb{GFX}_3}$, and so on. A different camera pixel \mathbf{x}_j^C has a different set of associated corresponding points in the other coordinate spaces, each written with the same subscript j .

We refer to the set of correspondences with the subscript j as the j th correspondence, which is located in the j th row of the lookup table. The i th and j th correspondences represent two distinct touch input locations on the touch surface S .

Next, we describe the incremental construction of the lookup table during the preprocessing phase in more detail, with the steps numbered from PP1 to PP8. For each appropriate step, we include a general example of the lookup table available. While the discussion below is largely general by design, we include specific implementation notes along with figures and descriptions of the application of this process to our physical-virtual patient head surface when useful. To simplify the discussion, we present the lookup table in a theoretical singular form, where each table row contains correspondences among all dimensions. However, in practice, our implementation opts instead to construct a series of specialized tables, each with one specific lookup coordinate (e.g. camera pixels) and one specific subset of the possible output (e.g. corresponding projector pixels). Later, in Chapter 4, we provide motivation for these implementation decisions and describe our software implementation for realizing touch sensing and response on several specific physical surfaces.

PP1) Obtain initial estimates of the camera calibration matrices.

First, we calibrate the IR cameras. Standard calibration approaches based on image captures of checkerboard or other patterns are suitable [158]. In accordance with these approaches, the patterns are sequentially placed throughout the volume containing the physical-virtual surface at various positions and orientations. After individually calibrating each camera to obtain an initial estimate of its intrinsic matrix, we then calibrate each stereo pair of cameras to obtain estimates of their extrinsic matrices. We choose one camera to serve as the origin of the calibration coordinate space TCH_3 , aggregating the extrinsic estimates from all stereo

pairs containing this particular camera; the 3D coordinates of the touch surface S in \mathbb{TCH}_3 , created in step PP6, are represented with respect to this origin.

Once these intrinsic and extrinsic matrix estimates are obtained, we globally optimize them by performing bundle adjustment across all cameras and observed calibration points. *Bundle adjustment* refers to the process of jointly refining camera pose, intrinsic calibration, and the 3D reconstruction of observed camera points [139]. The results of this optimization are used as the initial estimates of the intrinsic and extrinsic camera calibration matrices.

PP2) Observe sets of bidirectional projector-camera and camera-camera correspondences.

To reliably distinguish between touches and hovers, we wish to leverage information from multiple camera viewpoints of a potential contact event. To facilitate this, we desire a means of determining how pixels in a given camera’s image of the touch-sensitive surface correspond to pixels in the other cameras. Additionally, in order to respond to detected touch events, the system requires information regarding how camera pixels relate to both projector pixels and to 3D coordinates on the graphical model. In normal computer vision applications involving multiple camera viewpoints of the same object, correspondences among the many views are obtained via a feature matching routine. However, rear-projection surfaces tend to be both smooth and uniform in color, lacking in obvious features that can be easily detected.

Instead, we project a set of “manual” features onto the surface at known projector coordinates, which we refer to as the *feature scan*. These features could be black and white stripes, circles, or other shapes—the only requirement is that they are visible and can be reliably segmented in the camera imagery. Each projected feature is detected and localized in camera space by all cameras that can image it; the fact that a particular camera is unable to image a given projector feature is also useful information. The localized positions of detected features in each camera’s imagery represent corresponding surface points among the cameras. Thus, at this stage, the collection of camera images of projector features provides:

- **projector-camera correspondences:** bidirectional mappings between projector pixels and camera pixels (i.e. between the locations of *projected features* and *a camera's observations* of them)
- **camera-camera correspondences:** bidirectional mappings between pixels in one camera and corresponding pixels in another camera (i.e. between the *two cameras' observations* of the same projected feature)

While this set could be exhaustive—i.e. mapping all possible projector pixels to their camera pixel correspondences—this is temporally expensive. For example, a projector with a resolution of 1920×1080 pixels contains over two million pixels. Instead, we collect a sparse set of projector-camera correspondences, which we will later supplement by estimating unobserved correspondences in step PP8. Through the use of coded pattern images [126], it is possible to detect and localize many features with a comparatively smaller number of projections.

At this point, the lookup table contains a sparse set of bidirectional projector-camera and camera-camera correspondences, as in Table 3.2. The lookup table contains only k rows, based on the number of projected features, where k is insufficient to ensure that every camera and projector pixel is present.

Table 3.2: Lookup table after observing a sparse set of projector-camera and camera-camera correspondences (step PP2). Many projector and camera pixels are not present.

Row	Touch sensing: Cameras			Rendering: Projectors		
	C_1	\dots	C_C	P_1	\dots	P_P
1	$(x_1, y_1)^{C_1}$	\dots	$(x_1, y_1)^{C_C}$	$(u_1, v_1)^{P_1}$	\dots	$(u_1, v_1)^{P_P}$
2	$(x_2, y_2)^{C_1}$	\dots	$(x_2, y_2)^{C_C}$	$(u_2, v_2)^{P_1}$	\dots	$(u_2, v_2)^{P_P}$
\vdots	\vdots	\ddots	\vdots	\vdots	\ddots	\vdots
k	$(x_k, y_k)^{C_1}$	\dots	$(x_k, y_k)^{C_C}$	$(u_k, v_k)^{P_1}$	\dots	$(u_k, v_k)^{P_P}$

Implementation note—multiple projectors: In setups with multiple projectors, constructing this initial form of the lookup table (Table 3.2) is less straightforward, namely with regard to finding pixel correspondences among multiple projectors for a given feature. When a feature is projected by a particular projector, it can be imaged by the entire collection of cameras to find correspondences. However, the remaining projectors are not imaging devices and thus cannot directly determine their own respective corresponding pixels. As described in more detail in Chapter 4, our practical implementation of the lookup table is actually partitioned into several sub-tables that each focus on conversions between specific subsets of the coordinate spaces. For example, instead of a single table relating all camera pixels to their correspondences in all projectors, there exist sub-tables responsible for storing correspondences between pixels of one camera and pixels of one projector. We will continue to represent the lookup table in its theoretical singular form for the purposes of the present discussion.

PP3) Estimate an initial 3D touch surface point cloud in TCH_3 using the sparse set of correspondences, and augment the lookup table to store correspondences from camera and projector pixels to this point cloud.

With the initial camera extrinsic matrix estimates from step PP1, the sparse set of camera-camera correspondences from step PP2 can be triangulated within the calibration coordinate space TCH_3 , forming a sparse 3D point cloud that represents the touch surface. This allows us to begin augmenting the current lookup table with 3D information, as we now know how projector pixels and camera pixels map to 3D coordinates on the touch surface, denoted $\mathbf{X}^{\text{TCH}_3}$. To triangulate points, we use the optimal formulation presented by Hartley and Zisserman that minimizes geometric error through epipolar constraints [64].

An example of the current lookup table at this stage is shown in Table 3.3. As shorthand, we let the set $\{\mathbf{x}_j^{\{C_i\}}\}$ refer to the j th correspondence across all cameras from $i = 1$ to \mathcal{C} , the

number of cameras: that is,

$$\{\mathbf{x}_j^{\{C_i\}}\} = \{\mathbf{x}_j^{C_i}\}_{i=1}^C = \{\mathbf{x}_j^{C_1}, \mathbf{x}_j^{C_2}, \dots, \mathbf{x}_j^{C_C}\}$$

As in the previous step, there are only k rows in the current table.

Table 3.3: Lookup table after triangulating sparse camera-camera correspondences (step PP3). Many projector and camera pixels are not present.

Row	Cameras $\{C_i\}$	Projectors $\{P_i\}$	Scan
1	$\{\mathbf{x}_1^{\{C_i\}}\}$	$\{\mathbf{u}_1^{\{P_i\}}\}$	$\mathbf{X}_1^{\text{TCH}_3}$
2	$\{\mathbf{x}_2^{\{C_i\}}\}$	$\{\mathbf{u}_2^{\{P_i\}}\}$	$\mathbf{X}_2^{\text{TCH}_3}$
\vdots	\vdots	\vdots	\vdots
k	$\{\mathbf{x}_k^{\{C_i\}}\}$	$\{\mathbf{u}_k^{\{P_i\}}\}$	$\mathbf{X}_k^{\text{TCH}_3}$

PP4) Obtain an initial estimate for the projector calibration matrices.

The correspondences between projector pixels and 3D coordinates on the estimated touch surface point cloud provide information that can be used to calibrate the projectors. For a particular projector, given a set of projector pixels $\mathbf{u}_i = (u_i, v_i)$ that correspond to 3D points $\mathbf{X}_i^{\text{TCH}_3} = (X_i, Y_i, Z_i)^{\text{TCH}_3}$, we can use the Direct Linear Transformation algorithm to compute the maximum-likelihood estimate (MLE) of the projection matrix \mathbf{P} satisfying $\mathbf{u}_i = \mathbf{P}\mathbf{X}_i^{\text{TCH}_3}$ for all i [64]. To do this, we create two linearly independent equations from each correspondence of the form

$$\begin{bmatrix} \mathbf{0}^T & -(\mathbf{X}_i^{\text{TCH}_3})^T & v_i(\mathbf{X}_i^{\text{TCH}_3})^T \\ (\mathbf{X}_i^{\text{TCH}_3})^T & \mathbf{0}^T & -u_i(\mathbf{X}_i^{\text{TCH}_3})^T \end{bmatrix} \begin{pmatrix} \mathbf{P}^1 \\ \mathbf{P}^2 \\ \mathbf{P}^3 \end{pmatrix} = \mathbf{A}p = \mathbf{0}$$

where \mathbf{P}^{iT} is the i th row of \mathbf{P} (represented as a column vector by \mathbf{P}^i). Given k correspondences, the matrix \mathbf{A} has $2k$ rows and 12 columns. The set of equations can be represented by the matrix formula $\mathbf{A}p = \mathbf{0}$, where p is a 12×1 column vector that can be reshaped to form the 3×4 projection matrix \mathbf{P} . The solution to this equation is over-determined, so we estimate it by minimization; this can be achieved by computing the singular vector of \mathbf{A} corresponding to the smallest singular value [64]. From the projection matrix \mathbf{P} , the intrinsic and extrinsic matrices for the projector can be obtained via RQ-decomposition [64]. This process is repeated to compute initial estimates of the intrinsic and extrinsic calibration matrices for each of the \mathcal{P} projectors.

PP5) Perform bundle adjustment to jointly optimize the camera calibration matrices, the projector calibration matrices, and the 3D point cloud of the touch surface.

At this stage, we have computed initial estimates for the camera and projector calibration matrices along with an initial point cloud of the touch surface. These can together be globally refined through a final application of bundle adjustment. The refined 3D point cloud will be used in step PP6 to construct the touch surface S . In step PP8, the refined camera and projector calibration matrices will be used to supplement the lookup table with estimates of unobserved correspondences—projector pixels with no associated features in the feature scan. Additionally, the final projector calibration matrices will be modeled as virtual cameras in 3D rendering engines, such as Unity [141] or OpenGL [131]; a specific Unity implementation is described in the Appendix. This provides the registration of the virtual content to the touch-sensitive surface such that imagery of the content appears in the correct positions when projected onto the surface.

Implementation note—distortion correction: Bundle adjustment routines that optimize the calibration matrices of cameras and projectors are generally capable of correcting optical distortion. This is typically desired, as it produces calibration matrices with lower repro-

jection errors that better model the projection of 3D points in the environment to 2D pixels on camera and projector image planes, and it can yield more accurate 3D reconstructions. Accounting for distortion correction within the lookup table architecture requires a few modifications. Specifically, to prevent the need to undistort incoming camera imagery for every frame—which would increase latency linearly in terms of the number of cameras—we encode *observed* (i.e. distorted) pixels as lookup indices. However, in certain cases, we might want to retrieve *ideal* (i.e. distortion-corrected) pixels. In practice, this can be achieved by storing both observed and ideal pixels in the lookup table. We will return to this discussion in Chapter 4.

PP6) Create the 3D touch surface mesh S in TCH_3 , and augment the lookup table to store correspondences from camera and projector pixels to vertices on this mesh.

The bundle adjustment of step PP5 produces a refined 3D point cloud representing the touch surface S in the calibration coordinate space TCH_3 . However, this consists only of a collection of vertices $\mathbf{V}^S = (X, Y, Z)^S$ with no edge connectivity information. In order to consider correspondences from camera and projector pixels to arbitrary points on this mesh rather than only vertices, we must determine how the vertices are connected to form mesh faces.

In setups with only a single projector, connecting vertices to faces is straightforward. The triangulated features (step PP2) used to construct the initial estimate of the point cloud (step PP3) arose from specific projector pixels. If these pixels were chosen with a regular structure—such as a grid in projector space—the edge connectivity of this structure can be directly applied to the 3D point cloud. For example, suppose the feature scan includes projector feature (u_i, v_i) , its horizontally neighboring feature (u_j, v_j) , and its vertically neighboring feature (u_k, v_k) —i.e. these three pixels constitute a 2D triangular face in projector space. These three pixels correspond to the touch mesh vertices \mathbf{V}_i^S , \mathbf{V}_j^S , and \mathbf{V}_k^S ,

respectively, which can be connected with edges to form a 3D triangular face on the mesh. Otherwise, for arbitrary sets of discrete pixels, techniques such as Delaunay triangulation can be used to connect projector features and thus vertices of S with edges.

However, determining edge connectivity for a feature scan across multiple projectors is more complicated, as a feature arising from one projector may fall inside the structured-imposed faces from the features of another projector. Instead, a surface fitting method can be used to either approximate or interpolate the touch mesh S into a structure with known connectivity, such as a grid; this structure can then be used to create the set of edges connecting vertices of S .

At this stage, we can augment the lookup table with correspondence information relating camera and projector pixels to the vertices of the touch surface mesh S . An example is shown in Table 3.4. Again, only k rows are present. Here, the camera and projector pixel correspondences specifically refer to actual vertices of the mesh S —each projector pixel represents a feature in the feature scan with corresponding observations by each camera and a corresponding mesh vertex. In step PP8, the remaining camera and projector pixels will be linked to points on the faces of S —i.e. not vertices—which are now available through the mesh faces resulting from the computed edge connectivity.

Table 3.4: Lookup table after creating the touch surface mesh S (step PP6). Many projector and camera pixels are not present.

Row	Cameras $\{C_i\}$	Projectors $\{P_i\}$	Scan
1	$\{\mathbf{x}_1^{\{C_i\}}\}$	$\{\mathbf{u}_1^{\{P_i\}}\}$	\mathbf{V}_1^S
2	$\{\mathbf{x}_2^{\{C_i\}}\}$	$\{\mathbf{u}_2^{\{P_i\}}\}$	\mathbf{V}_2^S
\vdots	\vdots	\vdots	\vdots
k	$\{\mathbf{x}_k^{\{C_i\}}\}$	$\{\mathbf{u}_k^{\{P_i\}}\}$	\mathbf{V}_k^S

PP7) Align the 3D touch surface scan mesh S with the 3D graphical model G .

While the lookup table now supports conversions from a subset of camera and projector pixels to 3D coordinates on the scanned touch surface S , it lacks information about how these 3D coordinates relate to the 3D graphical model G that will ultimately be projected on the surface. For various reasons, it is useful to maintain a distinction between these two 3D objects, especially in cases where they do not exactly match. The touch surface S has a topology induced by the specific projector pixels used in the feature scan and the computed edge connectivity from step PP7; a simplified and retopologized version of S that more closely represents the structure of the desired virtual content may be more amenable to modeling, texturing, and animating. The origin of the calibration space \mathbb{TCH}_3 is the position of one of the cameras, which by design is located below and away from the actual physical surface. Instead, separating the two meshes allows G to exist in a dedicated coordinate space with an origin that can simplify the modeling process: for instance, one located close to the mesh with x -, y -, and z -axes that are aligned with the general volume of the 3D model. Additionally, enforcing independence of the model and the touch surface permits the modifications of one without affecting another; if cameras are moved, the feature scan must be repeated and so S must be recomputed, but no adjustments to G are necessary.

This separation incurs one cost: correspondences to points on the graphics mesh G must be separately computed. However, we can facilitate this by aligning it to the touch surface S ; in step PP8, we will use this alignment to compute the correspondences. In general, we assume that the two meshes are sufficiently similar that computational registration methods, such as the iterative closest point algorithm (e.g. [11]), are capable of aligning them. If the surfaces are substantially different and thus not suitable for such methods, manual alignment may be required. In some scenarios, the surfaces may be completely different—for instance, when a flat planar touch surface is used as an input interface for a 3D model. To handle

these, it is often easier to compute the relationship between the touch and graphical surfaces by modeling them together: that is, by importing the scanned mesh directly into the 3D modeling environment and positioning it with respect to the existing 3D graphical content to achieve the desired display on the surface.

The output from the automated or manual alignment is a transformation, comprising a rotation matrix R and translation matrix T , such that transforming the touch mesh S results in a mesh whose points are close to the points of the graphics mesh G . For automated alignment, the computational registration method minimizes the distances between the meshes.

PP8) Densely supplement the correspondences from camera and projector pixels to the 3D touch surface S and the 3D graphical model G .

With only the sparse set of observed projector features, there is an equivalently sparse set of mappings from camera and projector pixels to 3D coordinates on the touch surface S . As the goal of the lookup table is the encoding of the relationships of *all* camera and projector pixels to their equivalent positions on S and on the graphics mesh G , we desire a means of including non-observed correspondences and of obtaining correspondences to G . To accomplish both of these goals, we will use the camera and projection calibration matrices from step PP5 and the transformation that aligns S and G from step PP7 to estimate correspondences.

First, let us consider the touch mesh S . It exists in the calibration coordinate space TCH_3 along with the calibrated cameras and projectors, and the current form of the lookup table contains correspondences from a subset of camera and projector pixels to vertices on S . Mathematically, a camera's projection matrix represents the manner in which a three-dimensional world point is projected onto the camera's two-dimensional image plane. For example, given a camera matrix P , the projection of a touch mesh vertex V^S to a pixel x of the camera [64] is given by

$$P V^S = x \quad (3.1)$$

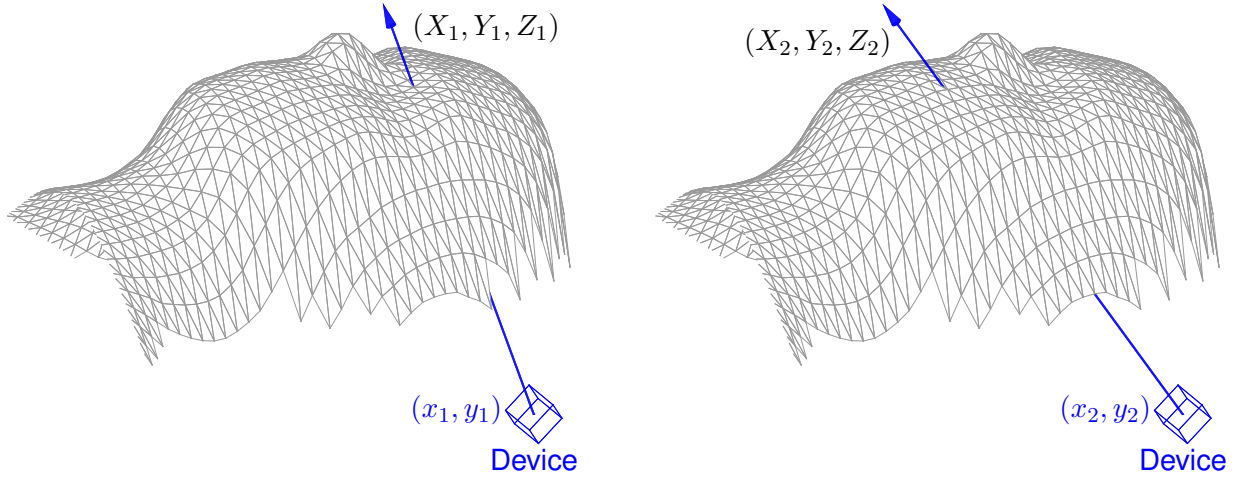


Figure 3.3: Supplementing unobserved correspondences with back-projected rays. All device (camera and projector) pixels are back-projected to three-dimensional rays; the points where these rays intersect the touch surface are stored as lookup table correspondences for the respective pixels.

Currently, the lookup table contains correspondences arising from actual *observations*: it stores \mathbf{x} as the camera pixel corresponding to \mathbf{V}^S because the projection of a feature at that 3D location was observed at that camera pixel. To estimate correspondences at a pixel with no such observations, we can back-project the pixel to a three-dimensional ray and compute its point of intersection on the touch mesh S . These intersections generally occur on the faces of the mesh, which are available via the edge connectivity computed in step PP6. Let \mathbf{M} be the left 3×3 sub-matrix of the camera projection matrix \mathbf{P} —i.e. $\mathbf{P} = [\mathbf{M} \mid \mathbf{p}_4]$, where \mathbf{p}_4 is the fourth column of \mathbf{P} . The back-projection of a pixel \mathbf{x} to a ray [64] is given by

$$\mathbf{X}(\mu) = \begin{pmatrix} \mathbf{M}^{-1}(\mu\mathbf{x} - \mathbf{p}_4) \\ 1 \end{pmatrix} \quad (3.2)$$

Thus, for a camera pixel with no observations, we need only choose a positive value of μ so that the ray starting at the camera’s position and passing through $\mathbf{X}(\mu)$ intersects the

touch mesh S . If $\mathbf{X}^{\text{TCH}_3}$ represents this intersection point, then the correspondence between the camera pixel \mathbf{x} and $\mathbf{X}^{\text{TCH}_3}$ can be added to the lookup table. This step is repeated for *all* camera and projector pixels so that a dense set of correspondences can be encoded in the lookup table. In particular, the original observations are *replaced* by these estimates. Figure 3.3 shows an example of this process.

Next, we wish to repeat this process for the graphics mesh G . While the touch mesh S , the cameras, and the projectors exist in the calibration coordinate space TCH_3 , G is located in GFX_3 , a separate coordinate space specifically for graphical modeling. Using the alignment transformation (rotation matrix \mathbf{R} and translation matrix \mathbf{T} from PP7), we transform the graphics mesh from GFX_3 to TCH_3 , back-project rays from the calibrated camera and projector pixels, and compute the intersection points on the transformed graphics mesh. Before storing the intersection points in the lookup table at the corresponding pixels, we transform them back to GFX_3 .

Table 3.5: Dense lookup table after back-projecting all camera and projector pixels to 3D rays and computing their intersections on S and G (step PP8). All projector and camera pixels are present.

Row	Cameras $\{C_i\}$	Projectors $\{P_i\}$	Scan	Graphics
1	$\{\mathbf{x}_1^{\{C_i\}}\}$	$\{\mathbf{u}_1^{\{P_i\}}\}$	$\mathbf{X}_1^{\text{TCH}_3}$	$\mathbf{X}_1^{\text{GFX}_3}$
2	$\{\mathbf{x}_2^{\{C_i\}}\}$	$\{\mathbf{u}_2^{\{P_i\}}\}$	$\mathbf{X}_2^{\text{TCH}_3}$	$\mathbf{X}_2^{\text{GFX}_3}$
\vdots	\vdots	\vdots	\vdots	
n	$\{\mathbf{x}_n^{\{C_i\}}\}$	$\{\mathbf{u}_n^{\{P_i\}}\}$	$\mathbf{X}_n^{\text{TCH}_3}$	$\mathbf{X}_n^{\text{GFX}_3}$

At this stage, the lookup table contains dense correspondences relating all camera and projector pixels to their intersection points on the touch mesh S and the graphics mesh G . An example is shown in Table 3.5; unlike previous example lookup tables with k rows, this iteration has n rows, where n is sufficiently high to ensure that all camera and projector pixels are present. In Section 3.3, we introduce mechanisms for linking the camera and projector

pixels to semantic behaviors on the touch surface during this same step of the preprocessing phase, which will further expand the lookup table.

Implementation note: If each camera and projector pixel is separately back-projected to a ray so that 3D intersection points on the touch mesh S and graphics mesh G can be computed, we must consider how to aggregate correspondences. That is, if camera pixel \mathbf{x} and projector pixel \mathbf{u} each have the correspondence $\mathbf{X}^{\text{TCH}_3}$, they should all exist in the same lookup table row. However, in practice, it is unlikely that *integer-valued* pixels from a camera and a projector, which are used as lookup table indices, correspond to identical three-dimensional points on S . Instead, we forward-project each mesh intersection point arising from some back-projected camera or projector pixel onto the image planes of the remaining cameras and projectors, and we separately maintain these forward-projections in sub-tables that store correspondences among a specific pair of optical devices. Figure 3.4 shows an example of this process. We return to this discussion in Chapter 4.

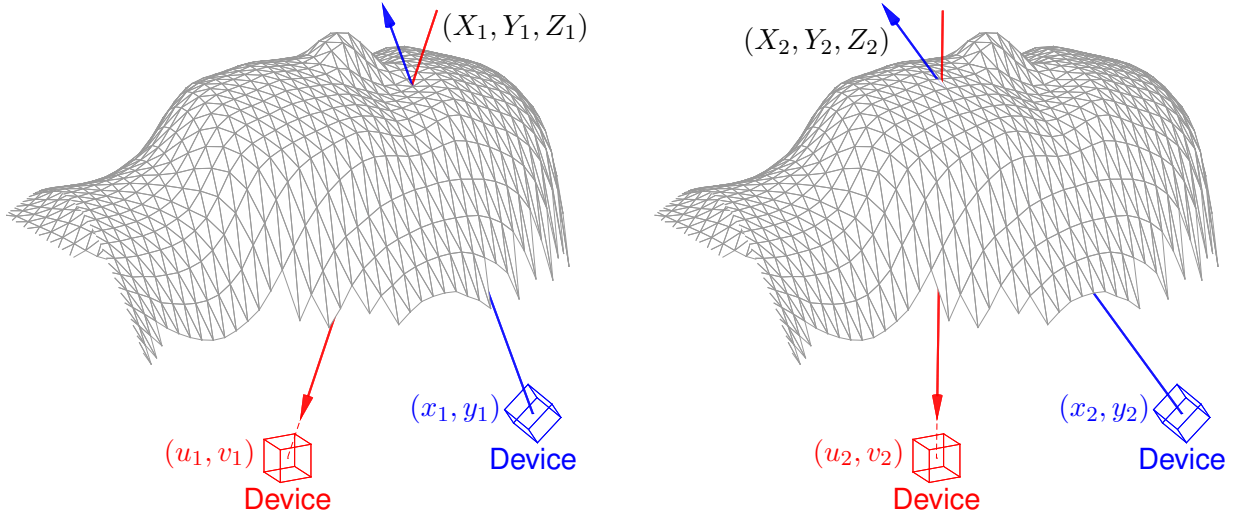


Figure 3.4: Supplementing unobserved correspondences with forward-projected intersections. All pixels on the image plane of one device (camera or projector) are back-projected to rays that intersect the touch surface, and these intersection points are forward-projected onto pixels on the image plane of another device. This provides correspondences between the pixels of the two devices.

3.2.1.1 Summary

The final result of this preprocessing phase is a lookup table suitable for sensing touch input, as shown in Table 3.5. Each row contains a set of corresponding points in all coordinate spaces. For instance, in row i , pixel $\mathbf{x}_i^{C_1}$ in camera C_1 corresponds to pixel $\mathbf{x}_i^{C_2}$ in camera C_2 , to pixel $\mathbf{u}_i^{P_1}$ in projector P_1 , to the point $\mathbf{X}_i^{\text{TCH}_3}$ on the touch mesh S , to the point $\mathbf{X}_i^{\text{GFX}_3}$ on the graphics mesh G , and so on. The correspondences are fully dense: all camera and projector pixels are present. Note that some pixels may not have correspondences in other coordinate spaces; for example, a particular projector pixel may not be visible to a particular camera. The lookup table captures this, storing a null value in the appropriate position.

The lookup table allows for constant-time conversions between coordinate spaces. A given coordinate can be queried in the table, and its equivalent coordinates in the other spaces can be determined as an $O(1)$ lookup. Thus, when touches are detected, appropriate responses can be triggered rapidly.

In its theoretical form, the table has n rows, where n represents the number of rows required so that all pixels across all cameras and all projectors are present; the value of n depends entirely on the exact configuration of cameras, projectors, and the touch surface. In practice, it is simpler to split this table into a collection of tables, each focused on a particular camera or projector. As an example, one sub-table can represent only the correspondences from camera C to projector P . The number of rows in each such table corresponds directly to the number of pixels in the optical device—i.e. the product of the width and height of the camera or projector image. This allows the sub-table to be stored directly as a matrix, allowing for efficient indexing operations and coordinate conversion retrievals in practice. We discuss these implementation aspects in more detail in Chapter 4.

Next, we present two algorithms for detecting user touch input at runtime via this lookup table. The lookup table can be further augmented with additional semantic information about the three-dimensional content displayed on the touch-sensitive surface to produce appropriate touch-triggered responses, such as animations and sound effects. By precomputing and encoding this information directly into the table, the amount of computation required at runtime to initiate these responses is reduced. We will return to this discussion in Section 3.3.

3.2.2 Runtime Touch Detection

Given the fully populated device-content relational lookup table from the preprocessing phase described in Section 3.2, the touch sensing system is next responsible for continuously analyzing camera imagery at runtime to detect and process user touch input so that an appropriate response can be selected. To this end, the touch sensing system seeks to address two primary problems at runtime:

- **Determining *if* a touch has occurred.** The principal challenge is reliably distinguishing between actual touches and near-contact proximity events (called “hovers”), which can have similar characteristics within the camera imagery. We refer to this process more generally as *touch/hover classification*.
- **Determining *where* a touch has occurred.** Following the successful detection of a touch event, the touch sensing system uses correspondences in the lookup table to link this touch to its 3D locations on the touch surface mesh S and on the graphics mesh G so that a proper semantic response can be triggered. We refer to this process more generally as *touch localization*.

Both of these problems can be solved by utilizing the correspondence information encoded within the lookup table. The dependent problem of determining *how to respond* to a detected and localized touch is handled separately by the semantic content engine, discussed in Section 3.3.

There are three possible classifications for potential events observed in the camera imagery at a particular time step:

1. A **touch event** represents contact between the user’s hand or fingers and the touch-sensitive surface.
2. A **hover event** represents proximity—but *not* contact—between the user’s hand or fingers and the touch-sensitive surface.
3. Otherwise, **no event** has occurred. This classification can arise due to noise in the camera imagery.

Each camera image can feature multiple events of each type.

Prior to runtime, initial background image models of the surface—imagery with no touches or hovers—are collected and stored for each camera. Additionally, binary region of interest (ROI) image masks are created for each projector-camera pair that represent the locations of valid camera image pixels—that is, pixels which can actually indicate touch input. First, each projector projects solid white images onto the surface, and each camera captures imagery. A pixel in one of these camera images that exceeds a threshold is set to 1 in the ROI masks, indicating both that the given camera pixel represents a specific location on the surface and that some corresponding projector pixel represents this surface location. Material that attenuates projected imagery—such as black felt—can be applied to the physical setup to prevent undesired projections on other objects or surfaces in the working volume. The remaining mask values are set to 0: either these pixels do not

represent a location on the surface, or they represent locations on the surface onto which the given projector cannot project. For a given camera, all projector-camera pair ROI masks are combined into a single camera ROI mask, representing all pixels within that camera that could correspond to a detectable touch that have a corresponding pixel in at least one projector.

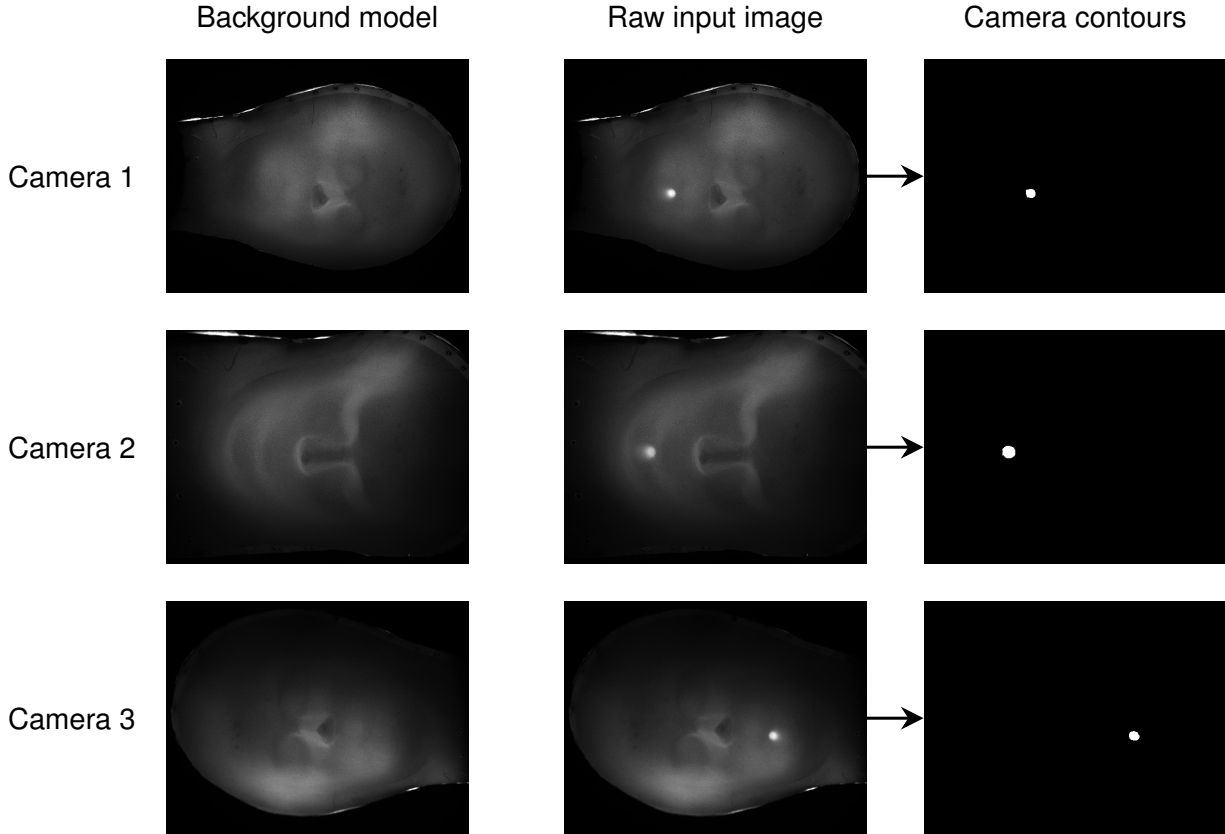


Figure 3.5: IR image processing. Initial background image models (first column) are subtracted from incoming camera images (second column). The result is thresholded and segmented into camera contours (third column) to be processed to determine the presence of touch input.

At runtime, incoming camera imagery is preprocessed to retain only those pixels that might correspond to a touch or hover event. First, the respective background model is removed from each camera image, a process called *background subtraction* [117]. In general, the camera images are expected to feature salt-and-pepper noise, which can be reduced from the background-subtracted

images using a morphological opening filter. Hysteresis thresholding is then applied to segment the image into candidate touch pixels. Finally, the resulting segmented image for each camera is filtered down via element-wise multiplication with the respective combined camera ROI mask. The pixels that survive this preprocessing are segmented into *camera contours*, distinct collections of pixels that represent potential touches or hovers. Figure 3.5 shows an example of this process.

Directly classifying these camera contours as touches or hovers poses several challenges. The entire collection of contours across all cameras together represents zero or more touches, zero or more hovers, and zero or more regions corresponding to no event (e.g. from remaining image noise); each such event may be observed as a separate contour in one or more cameras. Thus, the first goal is to group camera contours into subsets that each represent a single *candidate event*, with each camera contributing at most one contour to a particular candidate event. The group of contours belonging to each candidate event can then be collectively classified as a touch, a hover, or no event.

These camera contours arise from IR light that is reflected off of a user’s hand or fingers and transmitted through the touch-sensitive surface. Compared to traditional multi-view computer vision tasks, these contours provide relative few features—largely related to contour shape, contour geometry, and monochrome intensity. Due to the relationship between camera positions, IR light positions, and the touch-sensitive surface, contours corresponding to a particular touch event can have drastically different geometry and appearance across cameras, which can prevent successful feature matching. Moreover, camera imagery of actual touches is often highly similar to imagery of hovers.

Instead, we utilize the correspondence information in the lookup table to convert camera contours into alternative representations that allow for more direct processing. These representations appeal to the fundamental difference between touches and hovers: touches exist *on* the surface while

hovers exist *off* of it. The information obtained throughout the system preprocessing phase—and encoded in the lookup table—that relates the cameras, projectors, and the surface can be used to differentiate between touch and hover events. Furthermore, the lookup table also stores information that can be used to localize detected touches: Figure 3.6 shows the conversion of camera contours to corresponding 3D contours on the graphics mesh G .

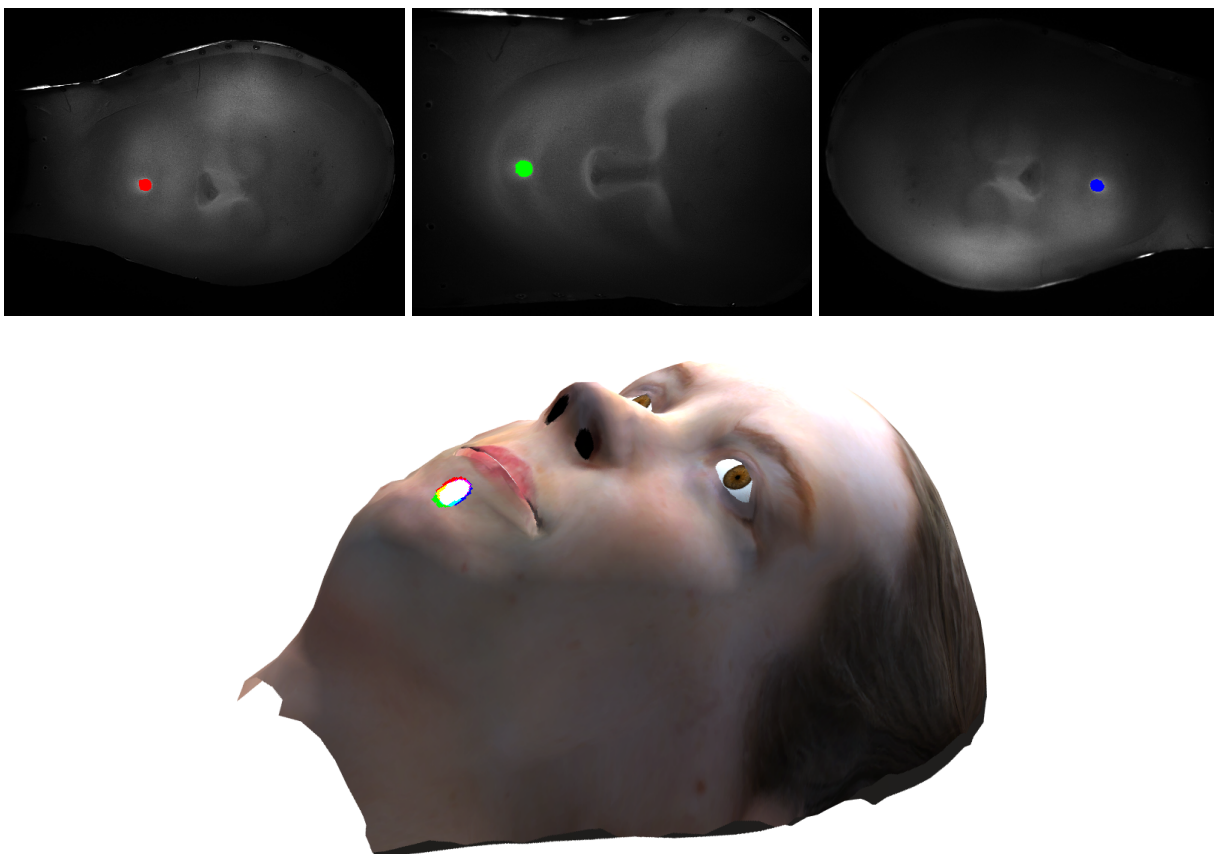


Figure 3.6: The conversion of camera contours (first row) to their corresponding positions on the graphical mesh G (second row), performed via the lookup table. In this visualization, the red, green, and blue contours from the three cameras are combined in RGB space on top of G —that is, white pixels indicate the overlap of the red, green, and blue contours.

We present two specific algorithms for touch detection, chronologically following our own historical development. The first, called **projection space touch sensing**, operates within the coordinate

spaces of the projectors that ultimately display virtual imagery on the touch-sensitive surface. The second, called **plane sweep touch sensing**, instead more directly considers the three-dimensional relationships between the cameras and the surface through the construction of several parallel planes at the location of a potential touch. Where clear, we abbreviate these two algorithms as simply “projection space” and “plane sweep,” respectively. In general, the two methods involve merging the camera imagery into one or more unified coordinate spaces using the correspondence information from the lookup table and determining how much *overlap* exists among the contours from each camera in these spaces; high amounts of overlap indicate touches, while low amounts indicate hovers. In the respective sections, we provide motivations for and detailed descriptions of these two touch sensing algorithms, explaining the methods by which they *group camera contours* into candidate events; *classify candidate events* as touches, hovers, or no event; and *localize detected touches* in the three-dimensional coordinates spaces of the touch mesh S and the graphics mesh G . Finally, we present a brief summary comparing the two approaches.

Suppose that \mathcal{C} cameras capture images of an actual touch or hover event (i.e. not noise or erroneous contours), but it is unknown whether they correspond specifically to a touch or to a hover. Let E denote this event, and let the contour of camera C_i corresponding to this event be denoted E^{C_i} . Each such contour comprises the pixels $\{\mathbf{x}_1^{C_i}, \mathbf{x}_2^{C_i}, \dots, \mathbf{x}_n^{C_i}\}$. A summary is shown in Figure 3.7. Throughout the descriptions of the touch sensing algorithms, we will refer to the classification of this hypothetical event E . Initially, the discussions will be restricted to a single touch or hover event such that exactly one contour observed by each camera is already known to correspond to E . Once this baseline detection strategy is established for each algorithm, we will then describe how multiple touches and hovers can be detected—i.e. how to group the camera contours into distinct events for individual classification.

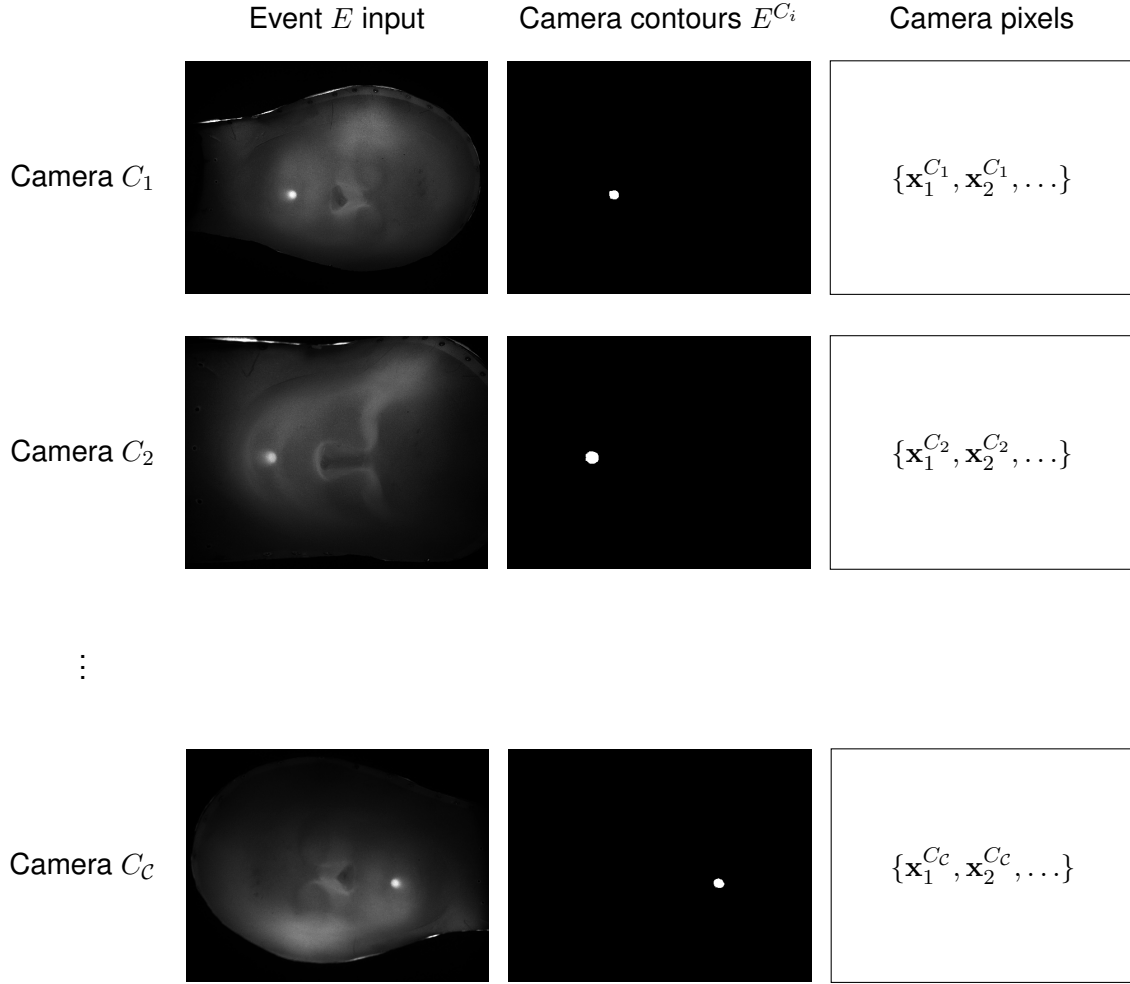


Figure 3.7: A potential touch or hover event E , comprising C camera images that each contain an associated contour E^{C_i} with pixels $\{\mathbf{x}_1^{C_i}, \mathbf{x}_2^{C_i}, \dots, \}$.

3.2.2.1 Projection Space Touch Detection

In the **projection space touch/hover classification algorithm**, the determination of whether camera imagery contains a touch or hover is performed by processing contours in \mathbb{PRO}_2 , the coordinate space of the projectors. There are a number of reasons for operating within projection space. The projectors are responsible for displaying virtual content on the touch-sensitive surface, and certain types of graphical responses—such as providing visual feedback of a touch to the user—occur

entirely within projection space. Similarly, the projectors serve as an intermediary between the cameras and the surface, as it is through triangulated projected features that the preprocessing phase establishes links between camera pixels and three-dimensional coordinates on the surface; as a result, information relating camera pixels to their corresponding projector pixels is readily available via the lookup table. In addition, contour computations in two-dimensional space can be performed efficiently, satisfying the design goal of rapid touch detection and response.

Via the lookup table, camera contours can be converted into corresponding *camera-to-projector contours* (or simply *projector contours*). Conceptually, a camera-to-projector contour represents the same “information” regarding a touch or hover in projector space that the camera contour represents in camera space: if the projector were an imaging sensor instead of a display device, it would have imaged the event at the location of this projector contour. Furthermore, we can develop intuition that suggests that these camera-to-projector contours should demonstrate a high degree of consistency for touches and a low degree of consistency for hovers. Consider a simple two-dimensional world consisting of a planar touch-sensitive surface, a camera, and a projector (Figure 3.8). A point that contacts the surface is imaged by the camera as a single pixel. Using a lookup table created in this simplified environment, this camera pixel can be converted to a corresponding projector pixel, shown exaggerated in Figure 3.8. Note that the camera pixel can be back-projected to a ray: any point on that ray is imaged at the same camera pixel whether it contacts the surface or not. Regardless of the point’s position on that ray—and whether it represents a touch or a hover—because it is imaged at the same camera pixel, it will be converted to *the same projector pixel* by the lookup table.

Rather than interpreting this result in terms of the camera pixel, we can instead consider the intersection of the back-projected ray and the surface, as it is this intersection point that is converted to projector space. Now, let us add a second camera to this simplified world. As the event object moves closer to the surface, eventually contacting it, the back-projected ray intersections converge.

Consequently, the camera-to-projector conversions will converge (Figure 3.9). These same concepts apply in 3D space.

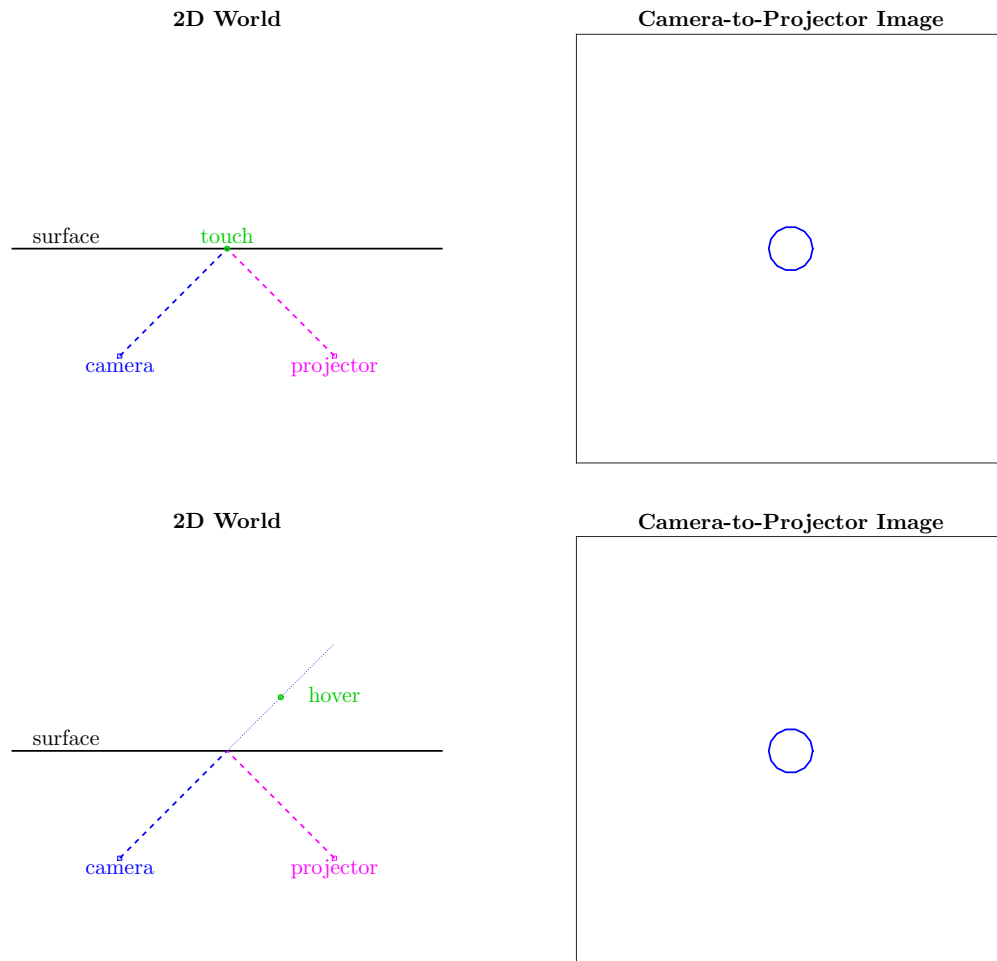


Figure 3.8: The camera-to-projector conversions of a touch and a hover in a simplified 2D world. A hover existing along the same back-projected ray as the touch is imaged by the camera at the same pixel, and so the camera-to-projector image is equivalent.

Figure 3.10 illustrates the process of converting camera imagery of a touch to projector space contours and combining the contours so that convergence can be analyzed. The contours from the three cameras are visualized as red, green, and blue, respectively; in the fourth row of Figure 3.10, these colored camera contours are shown combined in projector space, where white pixels indicate

overlap among all three contours. Likewise, the convergence of camera-to-projector contours for a touch and for a hover are compared in Figure 3.11. In projector space, the difference in convergence is prominent: touches exhibit much greater overlap than hovers. However, this distinction is not readily apparent in the raw camera imagery.

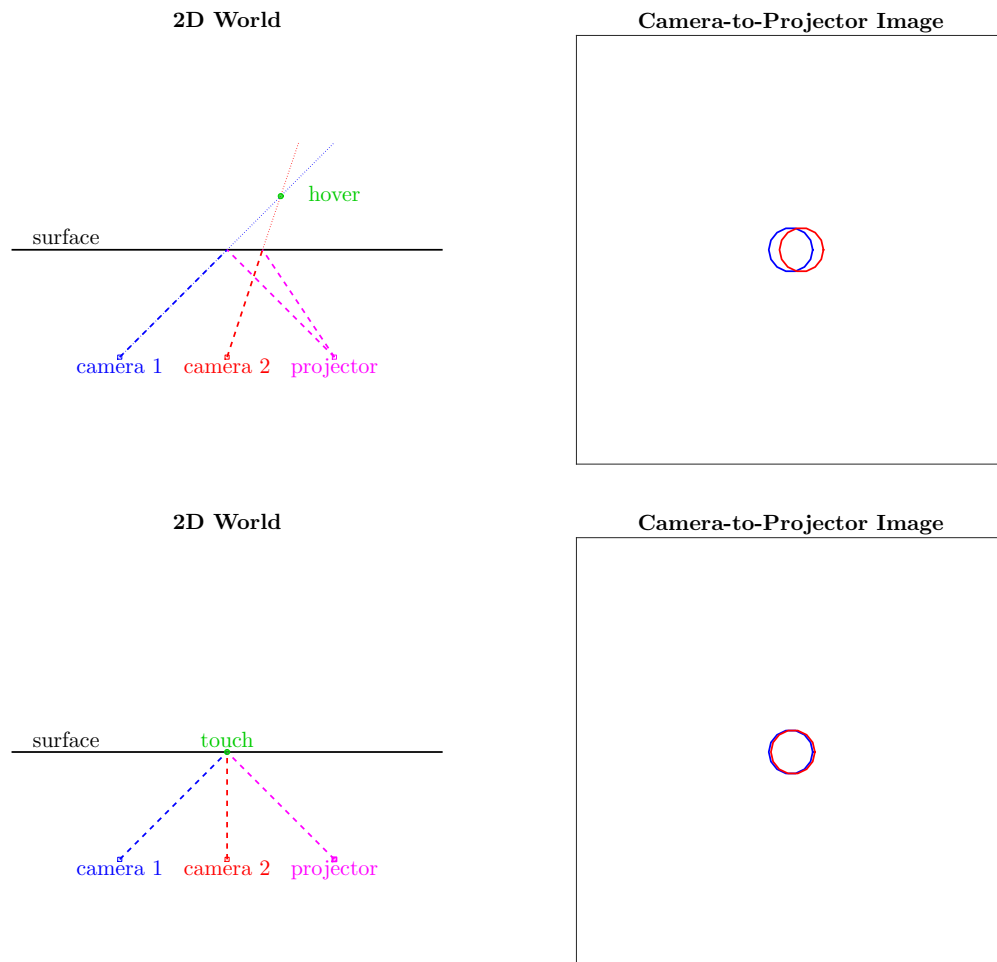


Figure 3.9: Multiple camera-to-projector conversions in a simplified 2D world. As the hover comes closer to the surface, the intersection points of the back-projected camera pixels to rays and the surface converge, so the camera-to-projector conversions converge.

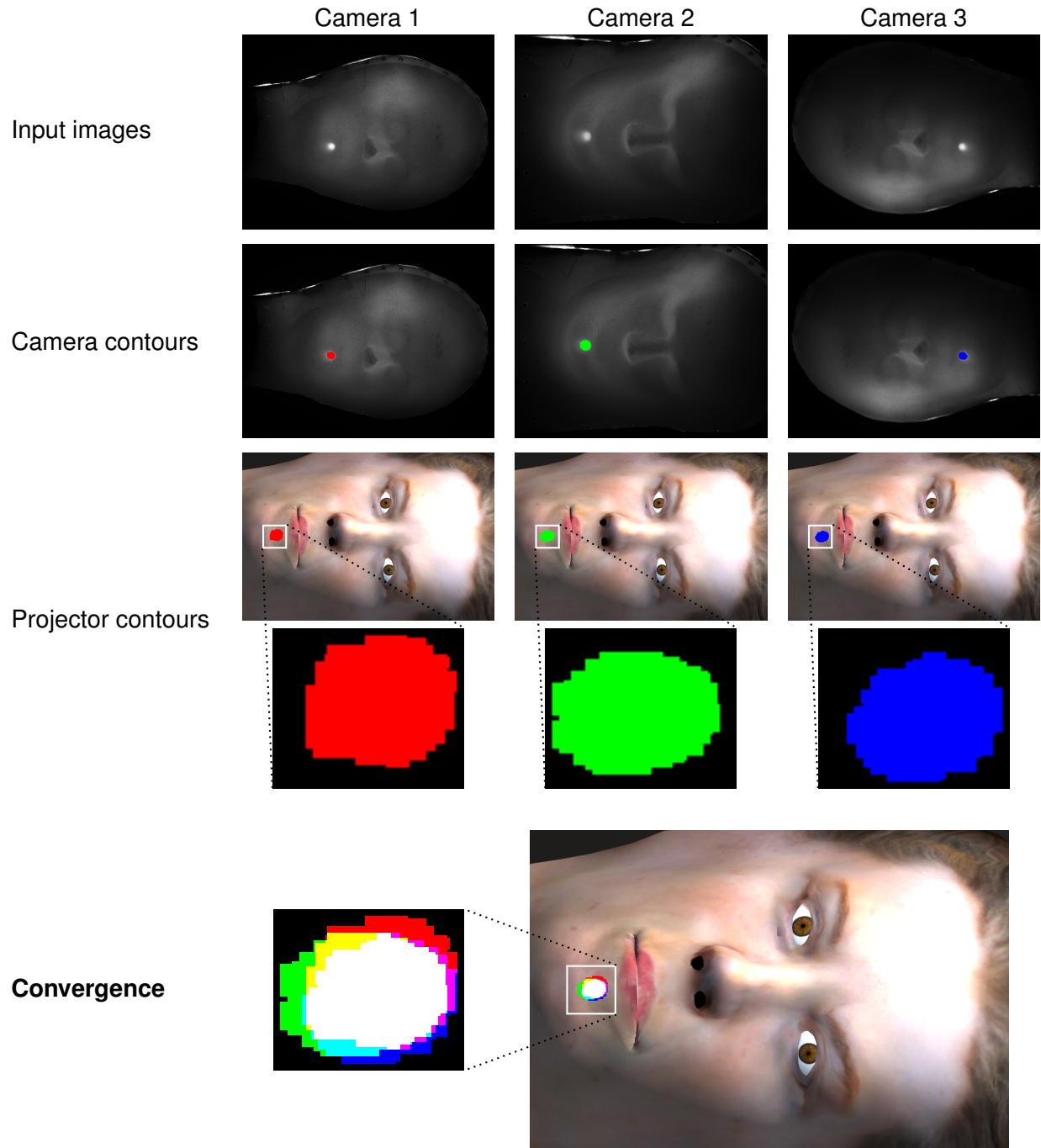


Figure 3.10: Touch convergence in PRO_2 . First, camera input images of a touch (first row) are processed to find camera contours (second row). Via the lookup table, the camera contours are converted to projector contours (third row). For touches, projector contours show a high degree of overlap (fourth row).

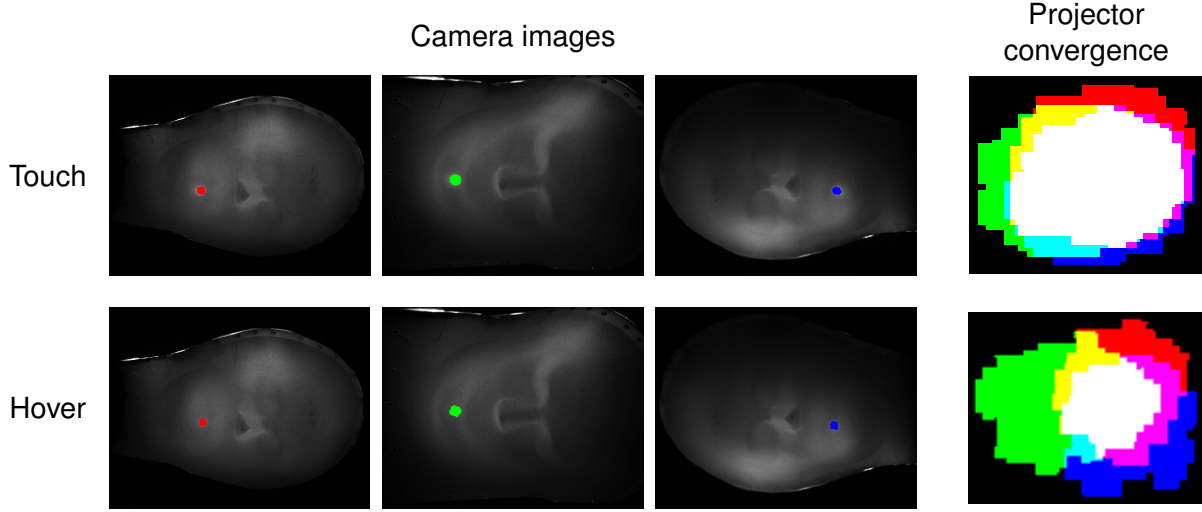


Figure 3.11: Convergence of camera-to-projector contours of a touch (first row) and a hover (second row). Touches exhibit much greater overlap than hovers in projector space.

3.2.2.1.1 Algorithm

Suppose that \mathcal{C} cameras capture images of an actual touch or hover event (i.e. not noise or erroneous contours), but it is unknown whether they correspond to a touch or to a hover. Let E denote this event, and let $E^{C_i} = \{\mathbf{x}_1^{C_i}, \mathbf{x}_2^{C_i}, \dots, \mathbf{x}_n^{C_i}\}$ denote the pixels in the camera contour of camera C_i , where $1 \leq i \leq \mathcal{C}$ (Figure 3.7). Via the lookup table, each camera pixel $\mathbf{x}_j^{C_i}$ can be converted to a 3D point $\mathbf{X}_{C_i,j}^{\text{TCH}_3}$ on the touch surface S ; let $X^{C_i} = \{\mathbf{X}_{C_i,1}^{\text{TCH}_3}, \mathbf{X}_{C_i,2}^{\text{TCH}_3}, \dots, \mathbf{X}_{C_i,n}^{\text{TCH}_3}\}$ be the set of 3D points on S from camera C_i 's contour. Furthermore, for the present discussion, we assume that the lookup table contains perfect correspondence information. If E represents a touch, then the actual 3D points corresponding to the user's finger are located *on* the touch surface S , which implies that the sets $\{X^{C_1}, X^{C_2}, \dots, X^{C_c}\}$ of the 3D correspondences of observed camera pixels comprise 3D points *on the same 3D region of* S . If E instead represents a hover, then these 3D user input points are located *off* the surface. However, by definition, all of the camera-to-3D points $\mathbf{X}_{C_i,j}^{\text{TCH}_3}$ are constrained to the surface, and so the sets X^{C_i} for $1 \leq i \leq \mathcal{C}$ will comprise 3D points *on*

different (but potentially overlapping) 3D regions of S . This is exactly equivalent to the behavior shown in Figures 3.8 and 3.9.

To simplify determining whether and to what extent the sets X^{C_i} overlap, let us instead consider the conversions of E^{C_i} from camera space to projector space: that is, let $U_{C_i}^P = \{\mathbf{u}_1^P, \mathbf{u}_2^P, \dots, \mathbf{u}_n^P\}$ be the projector contour pixels in projector P corresponding to the camera contour pixels of camera C_i . Since the camera pixel $\mathbf{x}_j^{C_i}$ corresponds to the 3D touch surface point $\mathbf{X}_{C_i,j}^{\text{TCH}_3}$, and $\mathbf{x}_j^{C_i}$ corresponds to projector pixel \mathbf{u}_j^P , it follows that \mathbf{u}_j^P corresponds to $\mathbf{X}_{C_i,j}^{\text{TCH}_3}$. The same conclusion about the contour overlap for touch events versus hovers applies here, but the advantage is that this holds in a two-dimensional coordinate space (PRO_2), where such computations are more direct. In other words, if E represents a touch, the sets $\{U_{C_1}^P, U_{C_2}^P, \dots, U_{C_C}^P\}$ will comprise *the same 2D regions in the coordinate space of projector P* , whereas if E represents a hover, these sets will comprise *different (but potentially overlapping) 2D regions*. This holds for the entire set of projectors $\{P_1, P_2, \dots, P_P\}$.

Compared to the analysis of the 3D regions X^{C_i} , this is a much simpler operation that can be performed directly on the pixels of images. To compute the amount of overlap, we create the *projector response mask* R^P with the same dimensions as the projector P , where $R_{u,v}^P$ is set to be the number of camera-to-projector contours containing pixel $(u, v)^P$. More formally,

$$R_{u,v}^P = \sum_{i=1}^C \begin{cases} 1 & \text{if } (u, v)^P \in U_{C_i}^P \\ 0 & \text{if } (u, v)^P \notin U_{C_i}^P \end{cases} \quad (3.3)$$

If camera C_i contains some pixel $(x, y)^{C_i}$ that corresponds to projector P pixel $(u, v)^P$, so that $(u, v)^P \in U_{C_i}^P$, we say that camera C_i *contributes* $(u, v)^P$. Thus, informally, the projector response mask R^P encodes the number of cameras that contributed each projector pixel through its camera-to-projector contour. An example is shown in Figure 3.12.

Based on this formulation, $R_{u,v}^P \geq 1$ for every projector pixel for which at least one camera-to-projector contour contains the pixel $(u, v)^P$, and $R_{u,v}^P = 0$ if no camera-to-projector contour contains this pixel. Thus, we can represent the aforementioned conclusion regarding touch versus hover imagery as the following: E contains a touch if $R_{u,v}^P \neq 0 \implies R_{u,v}^P = \mathcal{C}$ for all pixels $(u, v)^P$. This means that all camera-to-projector contours are identical, which implies that the corresponding 3D points on the touch surface S exist on the same 3D region of S . If this statement does not hold—that is, $\exists (u, v)^P \mid (R_{u,v}^P \geq 1 \wedge R_{u,v}^P < \mathcal{C})$ —then these 3D points occupy distinct (but potentially overlapping) regions of S and thus represent a hover.

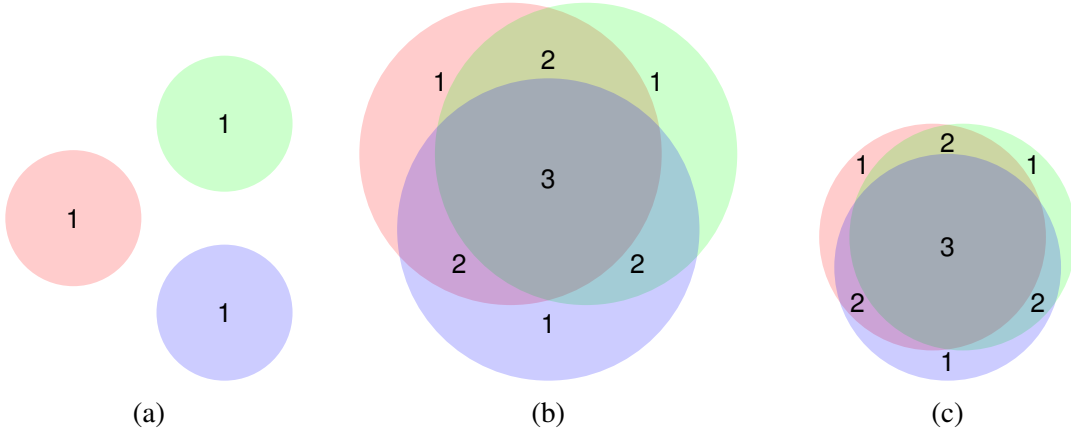


Figure 3.12: Example projector response masks, consisting of potentially overlapping camera-to-projector contours. In each example, the number of cameras contributing to each contour is shown. (a) The three camera-to-projector contours show no overlap, suggesting a hover. (b) The contours show some degree of overlap. (c) The contours show a high degree of overlap, suggesting a touch.

In practice, we must loosen this requirement. First, due to the relationships between the cameras, a projector P , and the touch-sensitive surface, it may be the case that not all \mathcal{C} cameras have a correspondence at pixel $(u, v)^P$. Instead, we consider the number of cameras that are known to have correspondences at $(u, v)^P$, denoted $N((u, v)^P)$, where $N((u, v)^P) \leq \mathcal{C}$. This value can be computed directly from the lookup table, which encodes missing correspondences as null entries. Now, we state that E contains a touch if $R_{u,v}^P \neq 0 \implies R_{u,v}^P = N((u, v)^P)$. Informally, this indicates that *all possible cameras* contributed pixel $(u, v)^P$.

Furthermore, we cannot assume that the lookup tables contain perfect correspondences: in practice, the camera-to-projector contours of an actual touch may exhibit a small amount of inconsistency, being composed of several close overlapping regions instead of a single consistent one, due to inaccuracies in calibration, lookup table construction, or infrared imagery segmentation. However, we can still expect camera-to-projector contours from hovers to show a higher degree of dissimilarity. Instead of asserting strict equality among the contours, we employ a weighted scoring scheme that considers how they overlap. If a large number of cameras—but not all—contributed a large number of projector response mask pixels—but not all—this is still evidence of a touch event. However, if only one or two cameras out of a possible four contributed each projector pixel, this suggests a hover. Thus, we consider the ratio of projector response mask pixels contributed by i cameras to the number of pixels contributed by at least one camera (i.e. the area of the projector response mask contour in pixels).

More formally, we create a weight vector $w = \{w_1, w_2, \dots, w_C\}$, where weight w_i applies to regions of the projector response mask R^P for which i cameras contained a contour pixel with a correspondence in projector P . Let $A(R^P)$ refer to the number of nonzero pixels in R^P , and let $A(R^P, i)$ be the number of pixels in the response mask equal to i , meaning i cameras contributed. Finally, let $N(R^P) \subseteq \{1, 2, \dots, C\}$ be the set of nonzero values in the response mask—that is, $i \in N(R^P) \implies \exists (u, v)^P \mid R_{u,v}^P = i$. Thus, the maximum number of cameras that could contribute to the projector response mask is given by $\max N(R^P)$. The *multi-camera agreement score* $S(R^P)$ is defined as

$$S(R^P) = \begin{cases} 0 & N(R^P) = \emptyset \\ \frac{\sum_i^{N(R^P)} w_i \frac{A(R^P, i)}{A(R^P)}}{\sum_i w_i} & N(R^P) \neq \emptyset \end{cases} \quad (3.4)$$

Informally, this represents:

- the *weighted sum* of the ratio between *the number of pixels contributed by i cameras* (represented by $A(\mathbb{R}^P, i)$) to *the number of pixels contributed by at least one camera* ($A(\mathbb{R}^P)$),
- divided by the sum of the weights.

This provides a confidence score in the range $[0, 1]$ for a projector response mask. In the event that all \mathcal{C} camera-to-projector contours are identical, then $N(\mathbb{R}^P) = \{\mathcal{C}\}$, $A(\mathbb{R}^P, \mathcal{C}) = A(\mathbb{R}^P)$, and the multi-camera agreement is equal to 1. If no cameras contributed any pixels to the projector response mask, the score is set to 0. As the size of regions for which the projector response mask is less than $\max N(\mathbb{R}^P)$ increases, the multi-camera agreement decreases as dictated by the weight values. For example, if $w_1 < w_2 < \dots < w_{\mathcal{C}}$, then pixels for which the projector response mask is equal to 2 contribute less to the score than those for which the projector response mask is equal to 3. This places a penalty on regions for which only a small number of cameras contributed projector response mask pixels and gives higher weight to regions with a larger number of contributing cameras. Thus, higher scores provide higher confidence of an observation of a touch, while lower scores suggest a hover.

3.2.2.1.2 Touch Localization

To localize a detected touch, we utilize lookup table correspondences relating projector pixels to 3D points on the touch surface S and the graphics mesh G . If desired, all projector response mask pixels—each of which arose from an observed camera pixel, thus providing information about the touch event—can be converted to their corresponding points on S and G . In many cases, it suffices to assign a single 3D point to the touch. To do so, we can compute the centroid of the region in P to which the maximum number of cameras contributed ($\max N(\mathbb{R}^P)$) and find its corresponding 3D

points via the lookup table. This region is chosen since it exhibits the highest degree of consistency among the camera observations: more cameras observed evidence of a touch in this region than in the others.

3.2.2.1.3 Multi-Touch Detection

The above discussion assumes that each camera contains up to exactly one contour already known to belong to the candidate event E . A useful property of this formulation is that it directly lends itself to the grouping of camera contours into distinct sets that each represent a separate input event—and therefore to the scoring and detection of multiple touches. Rather than scoring the entire projector response mask R^P , we begin by segmenting it into distinct contours. Suppose R^P can be segmented into the contours $\{R_1^P, R_2^P, \dots, R_r^P\}$ such that each contour has only nonzero values (i.e. at least one camera contributed each projector pixel) and moreover has at least one value greater than 1 (i.e. at least one pixel was contributed by two or more cameras). Each such projector response mask contour arises from camera contours that exhibit some degree of convergence within projector space, thus constituting a grouping of camera contours into a candidate event.

To detect multiple touches, we simply replace the multi-camera agreement scoring function and other related functions to operate on a single projector response mask contour R_j^P . Let $A(R_j^P)$ be the number of nonzero pixels in R_j^P , $A(R_j^P, i)$ be the number of pixels i cameras contributed, and $N(R_j^P) \subseteq \{1, \dots, C\}$ be the set of nonzero values in R_j^P . We can compute the score of each segmented contour exactly as before:

$$S(R_j^P) = \frac{\sum_i^{N(R_j^P)} w_i \frac{A(R_j^P, i)}{A(R_j^P)}}{\sum_i w_i} \quad \forall j \quad (3.5)$$

Since each projector response mask contour \mathbb{R}_j^P has only nonzero values, we need not handle the case when $N(\mathbb{R}_j^P) = \emptyset$.

Similarly, each contour in the projector response mask can be individually localized with corresponding 3D points on the touch surface S and graphics mesh G .

3.2.2.1.4 Multiple Projectors

There is one final consideration: handling the response masks of multiple projectors. Specifically, we wish to combine the confidence scores and localized detections from the response masks of multiple projectors that correspond to the *same input event* and to separate detections computed from the response masks of multiple projectors that represent *distinct input events*. To achieve this, we create a disjoint set data structure containing an initial set for each detection arising from the response masks of all projectors. Next, we compute pairwise 3D distances between each detection's corresponding position on the touch surface S . All pairs of detections whose distance is below a threshold are retained, as they are assumed to belong to the same input event. Subsequently, we take the unions of the sets corresponding to these retained pairs, producing disjoint groups of detections with similar 3D positions across all projectors. For groups that aggregate detections of the same input event across multiple projectors, the final assigned confidence score and 3D localizations on S and the graphics mesh G are averages across the scores and localizations of the constituent projectors, respectively.

3.2.2.2 Plane Sweep Touch Detection

In spite of the motivations for performing touch/hover classification directly in projection space, the aforementioned algorithm does have a few shortcomings. Its effectiveness is impacted by the

configuration of projectors used in the system—in particular, the number of projectors and their positions relative to the surface and to the cameras. In terms of performance considerations, as the number of projectors \mathcal{P} in the system increases, the complexity of the algorithm increases. Contours within each camera are converted to their corresponding projector contours for each projector; following contour scoring, the multi-camera agreement scores across all projectors must be aggregated to produce the final set of detections. Performing the camera-to-projector conversions requires the computation, storage, and indexing of camera-to-projector lookup table correspondences, each of which incurs additional costs for additional projectors.

Furthermore, though the projection space algorithm supports rapid processing by limiting analysis and computation to two-dimensional coordinate spaces, it loses some three-dimensional context that may improve classification results. For example, it does not directly utilize the known geometric relationships among the cameras and projectors, instead collapsing observed touch information into the two-dimensional coordinate spaces of the projectors. As a result, while the projection space algorithm can classify touches and hovers through the multi-camera agreement confidence score, it is unable to estimate the distance of a hover relative to the surface. Moreover, the interpretation of a given confidence score—whether it represents a touch or a hover—inherently depends on its location in projector coordinate space.

First, let us closely consider the manner in which the physical configuration of the projectors can impact the projection space touch sensing method. The position and orientation of a given projector relative to the surface greatly affects the shape, size, and position of camera-to-projector contours. In fact, even given a constant projector position, different configurations of cameras can produce the same camera-to-projector contours for different hover events. Figure 3.13 shows two configurations of cameras and hovers that yield the same camera-to-projector imagery in a simplified two-dimensional environment on a touch surface (the black line). Cameras a_1 and a_2 image hover a at the black dots in the graph, which can be forward-projected onto the projector's

image plane to produce the camera-to-projector images. However, cameras b_1 and b_2 image a different hover b —farther from the surface—than hover a . As such, between the two configurations, there is ambiguity in the camera-to-projector imagery, which alone is insufficient to distinguish between these two hovers. As the projection space method does not consider the positions of the cameras, it would therefore produce *the same confidence scores* for these two scenarios, despite the fact that hover b is located much farther from the surface than hover a . While the example in Figure 3.13 considers a two-dimensional environment, this same ambiguity is present in three-dimensional ones: for instance, if a sufficiently large object hovers at position b and a smaller object hovers at position a , they can produce the same imagery in camera pairs (a_1, a_2) and (b_1, b_2) , thus producing the same camera-to-projector imagery.

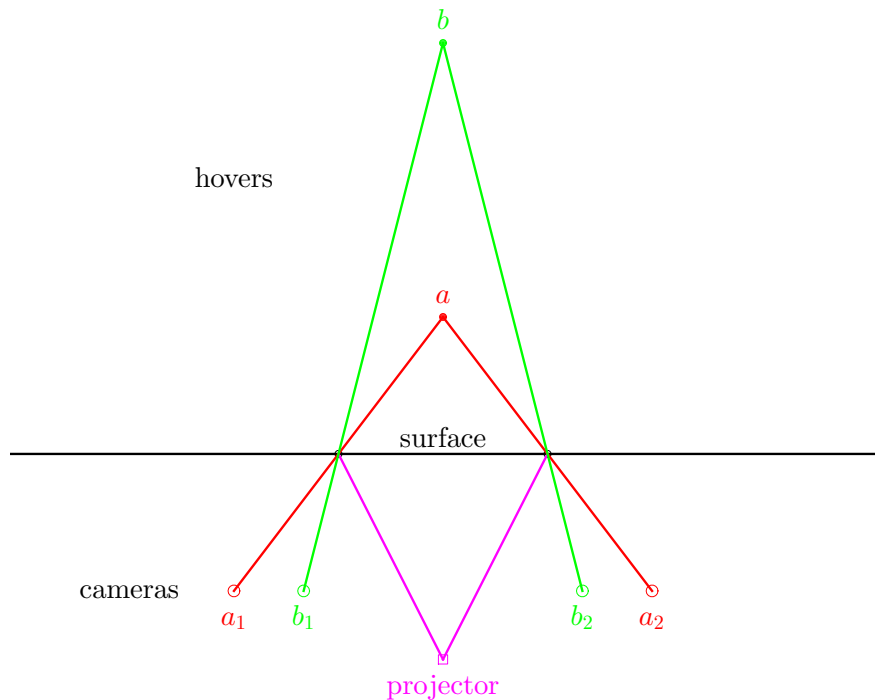
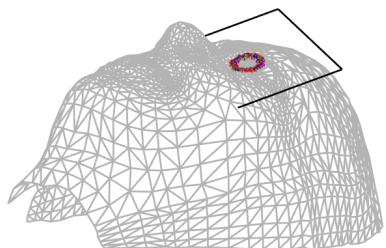


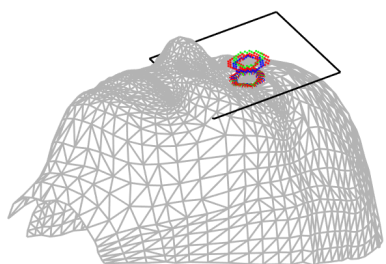
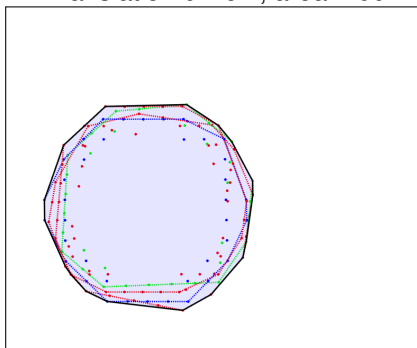
Figure 3.13: Ambiguity in projector space touch/hover classification. Two distinct hover events (a and b) produce the same camera-to-projector imagery for two configurations of cameras (pairs (a_1, a_2) and (b_1, b_2) , respectively).

Simulated Touch

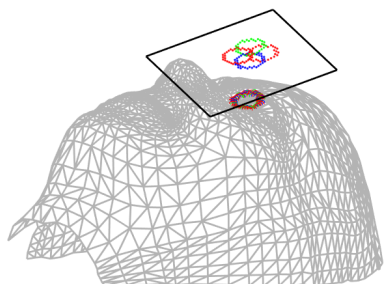
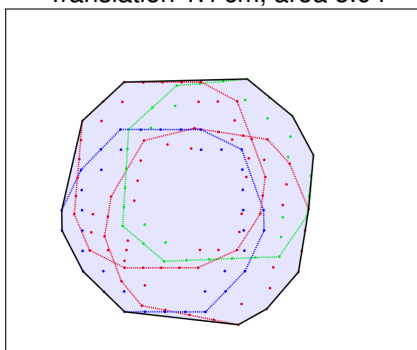


Plane Sweep

Translation 0.1 cm, area 4.00



Translation 1.1 cm, area 5.94



Translation 3.1 cm, area 10.32

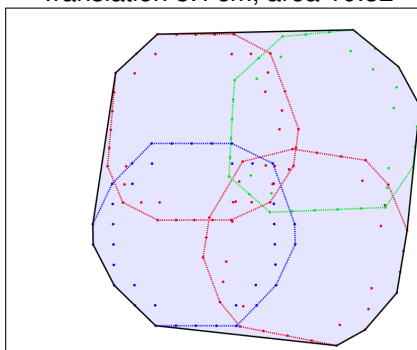
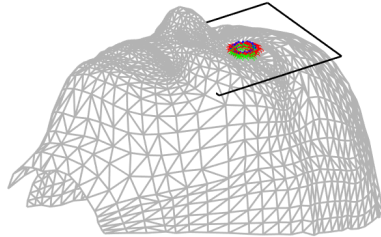


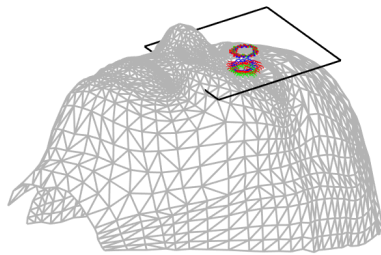
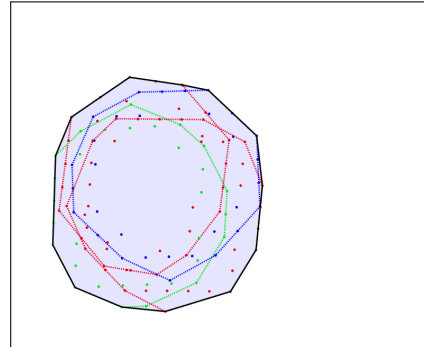
Figure 3.14: Plane sweep for a simulated touch. Each row shows a plane located at a progressively larger offset from the surface (left) and the plane projections of the camera contours (right). The plane projections of the camera contours have the smallest union area at a plane located close to the surface (first row), suggesting a touch.

Simulated Hover

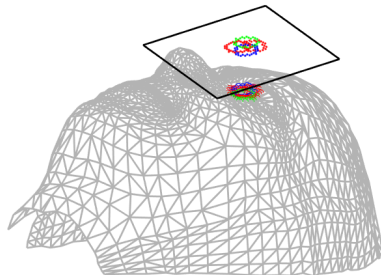
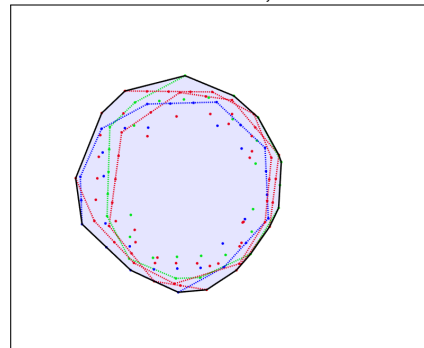
Plane Sweep



Translation 0.1 cm, area 3.45



Translation 1.1 cm, area 2.84



Translation 3.1 cm, area 5.96

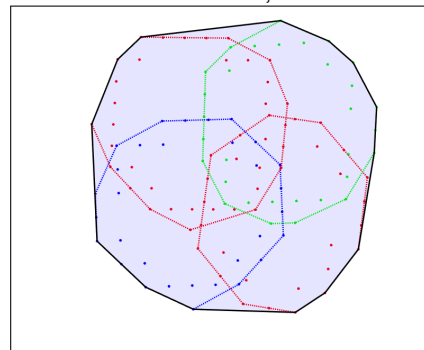


Figure 3.15: Plane sweep for a simulated hover. Each row shows a plane located at a progressively larger offset from the surface (left) and the plane projections of the camera contours (right). The plane projections of the camera contours have the smallest union area at a plane located away from the surface (second row), suggesting a hover.

3.2.2.2.1 Algorithm

To address these limitations, we propose a **plane sweep touch/hover classification algorithm** based on the construction of a series of parallel planes oriented within the calibration coordinate space \mathbb{TCH}_3 . Like the projection space classification algorithm, it involves overlap computations in two-dimensional coordinate spaces, which can be performed efficiently. However, the plane sweep algorithm is advantageous in that it more directly utilizes 3D information about relative camera and surface placement, which can improve the classification of touches and hovers. Moreover, it is completely independent of both the number and placement of projectors, instead relying exclusively on the correspondences between camera pixels and 3D touch mesh coordinates encoded in the lookup table. As a result, increasing the number of projectors does not impact the performance of this algorithm. Data collected across this series of planes is used to distinguish touches from hovers, and as an added benefit, the algorithm can estimate the position of a hover relative to the touch-sensitive surface.

At a high level, the algorithm computes 3D rays corresponding to the camera pixels of a potential touch or hover event and determines where the rays converge in space; if convergence is sufficiently close to the surface, the event is classified as a touch, while convergence sufficiently far from the surface results in a hover classification. Convergence computations are performed directly in the set of planes in \mathbb{TCH}_3 . An initial plane is oriented and positioned starting near the touch surface S , located based on the region of the surface corresponding to the camera pixels of the observed touch or hover event. Several parallel planes are subsequently constructed, each progressively farther from the surface. The 3D camera pixel rays intersect each of these planes, producing a series of 2D contours. For each given plane, the area of the union of the ray intersections across all cameras reflects a degree of similarity. As the planes approach the actual 3D intersections of the camera rays—that is, the actual location of the touch or hover event in 3D space—these planar

projections will converge, and the union area decreases. Therefore, if the plane corresponding to the minimum union area is sufficiently close to the touch surface, it signifies a touch event; if it is instead located sufficiently far from the surface, it signifies a hover. Figures 3.14 and 3.15 show simulated plane sweep results for a touch and hover, respectively.

The distance between the initial plane, located corresponding to the observed camera pixels on the touch surface, and the plane with the minimum projection union area is taken as the confidence score for the plane sweep algorithm. We refer to this plane as the *minimum union area plane*, and in general we state that the plane sweep algorithm *assigns* this plane (along with its distance) to a potential touch or hover event.

As before, let E denote a potential touch or hover event to be classified, and let the pixels in the camera contour of camera C_i be denoted $E^{C_i} = \{\mathbf{x}_1^{C_i}, \mathbf{x}_2^{C_i}, \dots, \mathbf{x}_n^{C_i}\}$ (Figure 3.7). First, we back-project each camera contour pixel to a 3D ray. For a given camera pixel, the $\text{CAM}_2\text{-to-TCH}_3$ correspondences in the lookup table contain the intersection point of the back-projection of that pixel to a ray and the touch surface S : that is, each pixel \mathbf{x}^{C_i} in camera C_i intersects the touch surface S at some point $(X, Y, Z)^S$. Let $I^{C_i} = \{(X_1, Y_1, Z_1)^S, (X_2, Y_2, Z_2)^S, \dots, (X_n, Y_n, Z_n)^S\}$ be the intersection points of these rays and S . Thus, the back-projection of camera pixel $\mathbf{x}_j^{C_i}$ to a 3D ray can be recovered via the lookup table by constructing a ray whose initial point is the position of C_i and that passes through $(X_j, Y_j, Z_j)^S$. Let R^{C_i} refer to these 3D rays for the contour pixels of camera C_i . Moreover, let $I = \{I^{C_1}, I^{C_2}, \dots, I^{C_c}\}$ be the collective set of intersection points of the rays for all cameras and the touch surface S , and let $R = \{R^{C_1}, R^{C_2}, \dots, R^{C_c}\}$ be the corresponding 3D rays. Conceptually, the true location of the event E is the location where the rays R converge in 3D space.

Next, we construct the initial plane P_0 in the series of planes $\{P_0, P_1, \dots, P_k\}$. The set I of the $\text{CAM}_2\text{-to-TCH}_3$ correspondences constitutes a reasonable starting point to perform the plane

sweep: if E is a touch event, it occurs *on* the touch surface S —exactly on the points in I . If E is instead a hover, it occurs at some location *off* the touch surface S , translated farther along the set of rays R . Thus, we begin with a best-fit plane to the points in I :

- The origin \bar{P}_0 of this initial plane P_0 is given by the centroid of the points in I .
- Let the matrix I_s be the intersection points with the mean \bar{P}_0 subtracted such that they are centered at the origin. Let $U\Sigma V^T$ be the singular value decomposition of I_s . The eigenvector associated with the minimum singular value—i.e. the third column of U —provides the normal for plane P_0 , which we denote N .
- As we wish to construct parallel planes moving away from the surface, the normal should be oriented along the same general direction as the back-projected rays; if the chosen eigenvector points in the opposite direction—i.e. toward the cameras—we reverse it.
- The initial plane P_0 is translated along its normal N until all intersection points in I are located “below” it with respect to the camera.
- Finally, we compute an orthogonal basis for the plane. First, we compute the cross product of the normal vector and any other vector not collinear to it, forming one axis d_1 . The second axis d_2 is formed by computing the cross product of the normal vector and the first axis d_1 .

The subsequent planes $\{P_1, \dots, P_k\}$ are then formed by translating the initial plane P_0 along its normal N by prescribed amounts. Each plane P_j has an associated origin \bar{P}_j , also computed by translating the initial plane origin \bar{P}_0 along the normal N by these amounts.

The mathematical *plane sweep* consists of the computation of the intersections of the camera rays R and each of the planes in $\{P_0, P_1, \dots, P_k\}$. The plane P_j —translated some distance t_j from the initial plane P_0 along the normal vector N —for which the planar projection points have the greatest

convergence provides an estimate for the location of the event E . For a measure of convergence, we consider the area of the union of the planar projection points: the union area will be minimized where they show the greatest convergence.

To compute the planar intersections, we rely on the standard vector notations of planes and lines. A plane containing some point P and having normal vector N consists of all points x such that

$$(x - P) \cdot N = 0$$

where \cdot indicates the dot product. Likewise, a line with direction R containing some point l can be parameterized as

$$x = sR + l$$

with the real parameter s . The intersection of this line and plane is thus given by

$$(sR + l - P) \cdot N = 0$$

which can be solved for the parameter s as follows:

$$(sR \cdot N) + ((l - P) \cdot N) = 0$$

$$(sR \cdot N) = (P - l) \cdot N$$

$$s = \frac{(P - l) \cdot N}{R \cdot N}$$

For plane P_j and a ray from camera C , we take the plane origin \bar{P}_j as the point on the plane and the position of camera C as the point on the line (with direction R). Thus, the intersection of this

line with the plane P_j is given by the set of points

$$x = sR + C \quad (3.6)$$

where the parameter s is given by

$$s = \frac{(\bar{P}_j - \mathbf{C}) \cdot N}{R \cdot N} \quad (3.7)$$

and \mathbf{C} refers to the position of camera C .

Currently, these points x exist in three-dimensional space. To facilitate area computations, we project them down to two-dimensional coordinates (p_x, p_y) , given by the scalar projections along the two orthonormal plane axes d_1 and d_2 :

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} d_1 \cdot (x - \bar{P}_j) \\ d_2 \cdot (x - \bar{P}_j) \end{bmatrix} \quad (3.8)$$

where the plane origin \bar{P}_j is subtracted so that it projects down to the origin $(0, 0)$ of the two-dimensional plane.

Using these two-dimensional planar projections, we can now investigate union areas. Note that a particular camera's planar projection area monotonically increases across each subsequent plane as we traverse along the normal vector. Conceptually, this is because the three-dimensional volume of the rays corresponding to a set of camera pixels contains more uncertainty as the distance from the camera increases, as shown in Figure 3.16. For a set of cameras with rays exhibiting *no overlap* in the planar projections—which in practice can result from hovers far from the surface—this means that the the overall *disjoint union area* among the planar projections monotonically increases. In other words, the plane with the minimum disjoint union area will be the one closest to the surface, incorrectly indicating a touch event. Instead, we consider the area of the *convex*

hulls of the planar projections, which exhibit the desired behavior: regardless of the degree to which the planar projections across a set of cameras overlap, the *convex hull union area* decreases as we approach the true location of the event E and increases as we translate away from it in either direction. This concept is illustrated in Figure 3.17.

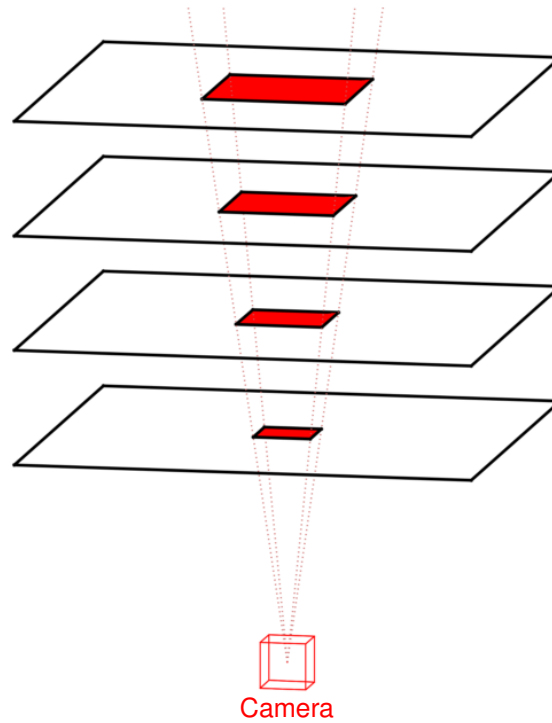


Figure 3.16: The uncertainty about the size of an object imaged by a camera increases with distance from the camera. As a result, the planar projection areas corresponding to this object increase as the distance from the plane to the camera increases.

3.2.2.2.2 Touch Localization

In the projection space algorithm, localizing a touch simply involves lookup table retrievals—namely the correspondence between a projector pixel and a 3D point on the touch surface S . However, for the plane sweep algorithm, the information used to classify touches and hovers instead

exists on a separately constructed plane in the calibration space \mathbb{TCH}_3 , which is not encoded in the lookup table. We compute a ray located at the initial plane origin and oriented in the opposite direction as the plane sweep normal (i.e. towards S), and the location where this ray intersects S is chosen as the localization point.

Moreover, the graphics mesh G exists in the graphics space \mathbb{GFX}_3 , which is separate from \mathbb{TCH}_3 . Using the alignment between \mathbb{TCH}_3 and \mathbb{GFX}_3 computed during step PP7 of the lookup table preprocessing phase, the initial plane can be transformed to an equivalent position relative to G . We repeat the above process, computing the intersection point of a ray from the transformed plane's origin oriented in the reverse normal direction and G to obtain the localization.

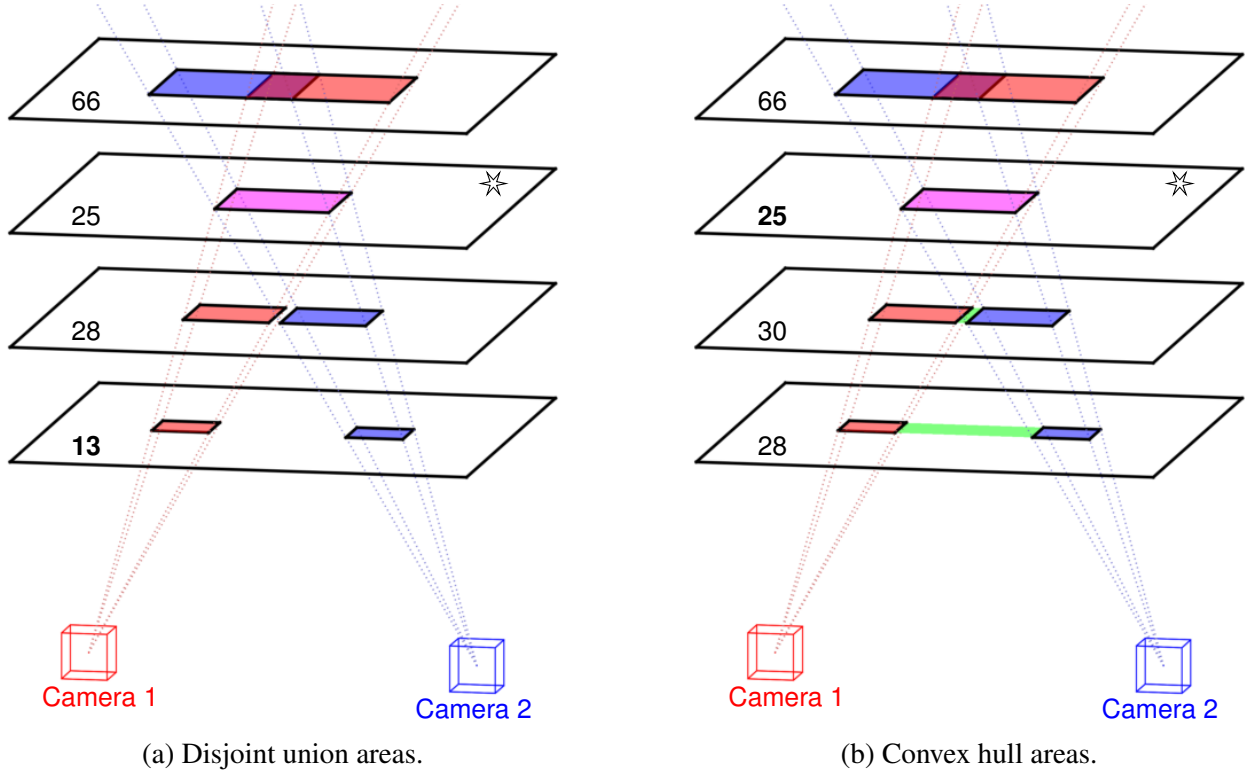


Figure 3.17: Planar projection area computations for a simulated event (located on the starred plane). (a) The disjoint union area is the smallest for the closest plane, which would erroneously suggest the event occurred close to the surface. (b) The convex hull area, which includes the green regions, is minimized at the plane corresponding to the event, which is the desired behavior.

3.2.2.2.3 *Multi-Touch Detection*

Next, we address the challenge of extending this approach to multiple touch detection. As before, the first step requires the grouping of contours across each camera into candidate events so that each candidate can be separately processed. We begin by creating a disjoint set data structure containing all contours observed by all cameras. For each contour, we compute the centroid and store its corresponding 3D position on the touch mesh S —obtained from the lookup table—into a disjoint set. Next, we compute pairwise distances between these positions, taking the unions of sets whose distances are below a threshold. Each disjoint set, comprising up to one contour from each camera, thus establishes a candidate event that can be processed using the aforementioned plane sweep formulation.

3.2.2.3 *Comparison of Approaches*

Here, we present a summary of the differences between the two touch/hover classification methods. Both the projection space and plane sweep algorithms involve area computations in two-dimensional space: in the space of the projectors and in the space of computed 2D planes, respectively. The source of 3D context used for touch detection is indirect for the projection space algorithm, being a function of how a touch or hover event in 3D space is observed as camera pixels and subsequently converted to projection space. Instead, the plane sweep algorithm more directly utilizes these 3D relationships, operating over 3D rays corresponding to these camera pixels. As a result, the plane sweep algorithm is able to estimate the distance between hovers and the surface.

In some ways, the projection space algorithm can be interpreted as a limited and specialized case of the plane sweep algorithm, for which only a small number of predefined planes is considered—one per projector. Additionally, rather than comparing contour areas across planes to better estimate

the actual position of the observed event, the projection space algorithm classifies the event as a touch or hover based on these areas within only a single plane for each projector.

Another notable advantage of the plane sweep algorithm over the projection space algorithm is that it is not dependent on the number or positions of projectors. In addition to facilitating improved detection and localization results, as will be discussed in Chapter 6, this also minimizes the number of lookup table correspondences that is required to relate touches to their 3D positions on the touch mesh in the calibration space TCH_3 and on the graphics mesh in the graphics space GFX_3 ; as a result, the plane sweep algorithm localizes and classifies touches significantly faster than the projection space algorithm. Specifically, the projection space algorithm uses sets of camera-to-projector, projector-to- TCH_3 , and projector-to- GFX_3 correspondences, which increase in number as the number of projectors increases. Given \mathcal{C} cameras with resolution $w^C \times h^C$ and \mathcal{P} projectors with resolution $w^P \times h^P$, this amounts to

$$\mathcal{C}w^Ch^C + 2\mathcal{P}w^Ph^P$$

lookup table entries.¹ In practice, the resolution of the projectors tends to be significantly higher than the resolution of the cameras, so the dependency on projector-to- TCH_3 and projector-to- GFX_3 correspondences is expensive. In contrast, the plane sweep algorithm requires only the camera-to- TCH_3 and camera-to- GFX_3 relationships, totaling only

$$2\mathcal{C}w^Ch^C$$

correspondences.

¹This discussion follows the use of sub-tables, described in more detail in Chapter 4: instead of a single lookup table encoding all correspondences, each sub-table stores correspondences between a specific subset, such as between pixels in a particular camera and pixels in a particular projector. The number of indices in each sub-table is therefore equal to the dimensions of the camera or projector images. This representation provides a more straightforward implementation compared to the theoretical singular form.

Additionally, there are correspondences related to semantic content responses, which will be described in Section 3.3. For now, we simply represent these as some set of R correspondences from pixels to semantic regions, where the projection space algorithm requires such correspondences from projector space and the plane sweep requires them only from camera space. Thus, the projection space algorithm requires

$$\mathcal{C}w^Ch^C + (2 + R)\mathcal{P}w^Ph^P$$

correspondences, while the plane sweep requires only

$$(2 + R)\mathcal{C}w^Ch^C$$

correspondences.

These comparisons are summarized in Table 3.6.

Table 3.6: Comparison between projection space and plane sweep touch detection algorithms.

	Projection Space	Plane Sweep
Computation coordinate space	2D (projection space)	2D (computed planes)
Source of 3D context	Indirect	Direct
Dependency on projectors	Yes	No
Hover distance estimation	No	Yes
Lookup tables	Camera-to-projector Projector-to- TCH_3 Projector-to- GFX_3 Projector-to-semantic-behavior	Camera-to- TCH_3 Camera-to- GFX_3 Camera-to-semantic-behavior
Lookup table correspondences	$\mathcal{C}w^Ch^C + (2 + R)\mathcal{P}w^Ph^P$	$(2 + R)\mathcal{C}w^Ch^C$

3.2.2.4 Touch Labeling

The projection space and plane sweep touch sensing algorithms address the problems of determining *if* and *where* a touch has occurred. To support dynamic, interactive touch experiences, we must also consider the problem of *labeling* touches, so that multiple touches that start and end at different time steps maintain specific information over time. Such information is important to the semantic content engine, which makes decisions about updates to the virtual content based on touch input; these updates may feature time-dependent components, such as updating an animation in response to continuous touch input beginning in specific regions.

The two touch sensing algorithms make no guarantee as to the order in which detected touches are returned. In practice, the ordering depends internally on the ordering of camera or projector contours during the segmentation process. Instead, we apply a final common runtime method to link the touches detected at the current time step t to those detected in the previous time step $t - 1$. For simplicity, we will refer to touches detected at time step t as *current touches* and touches detected at the preceding time step $t - 1$ as *previous touches*. We distinguish between the *index* of a touch as returned by one of the detection algorithms and the *label* of a touch, which is a persistent, unique identifier; the focus of the present discussion is the assignment of these labels. For each time step, as we process touches, we maintain a list called `assigned`, where `assigned(l) = true` once label l has been assigned to a current touch. The label l may only be assigned to a single touch at each time step.

First, we consider the case when one or more touches are detected on the current time step t . If no touches were detected on the previous time step $t - 1$, then no label matching across the two time steps is required, and the current labels can be taken directly from the indices produced by the detection algorithms. Otherwise, we compute the pairwise distances between the localizations of the current and previous touches on the touch mesh S , retaining pairs with distances below a

threshold. If current touch i is sufficiently close to previous touch j and j 's label has not been assigned, we perform the following operations to merge them:

- Flag current touch i as being an existing touch—i.e. not a new touch beginning on time step t .
- Previous touch j was tracked for some number of frames x : set current touch i 's frame tracked counter to $x + 1$.
- Likewise, previous touch j had information associated with semantic behavior: update current touch i 's initial configuration to match.
- Finally, set the label of touch i to be the label of touch j , and update `assigned` to store `true` for this label.

If no touches are detected on the current time step t , but touches were detected on the previous time step $t - 1$, we place the previous touches in a special *lost* state. Touches in this state are maintained for a small number of frames; if a similar touch appears in a similar position within this frame window, the touch is reverted to a normal *active* state, and its previous configuration information (label, number of tracked frames, semantic behavior information, etc.) is retained for subsequent updates. All lost previous touches are placed in the set of current touches, provided they are still within this window of frames, and they are removed once the window expires. Lost touches are flagged accordingly so that they do not trigger responses.

At this point, current touches that match previous touches have been assigned matching labels. However, current touches with no previous matches—i.e. new touches—have not. For each such touch, we assign the first unused label, found by looping through the `assigned` list and returning the first label l for which `assigned(l) = false`.

3.3 Responding to Touch

Once a touch has been detected and localized by the touch sensing system, the semantic content engine is responsible for determining the appropriate semantic response, which is ultimately realized by the rendering system. In the preceding discussion, we described the construction of a relational lookup table capable of achieving touch detection and localization through the use of encoded correspondences among coordinate spaces. The lookup table can be further augmented with information that simplifies the process of mapping touch input to semantic output. First, we discuss these augmentations and provide details about updates to the two touch sensing algorithms to support them. This is followed by a description of the *touch messages*, which are created by the touch sensing system and transmitted to the semantic content engine, that contain this semantic information. Finally, we briefly cover the role of the rendering system.

3.3.1 *Semantic Content Engine*

The *semantic content engine* consists of two major components: the virtual content displayed to users (e.g. through projected imagery and audio output) and a mechanism for assigning semantic meaning to user touch input. In general, the virtual content consists of a three-dimensional model representing the physical touch-sensitive surface; in earlier lookup table discussions, we referred to this as the graphics mesh G residing in the graphics space \mathbb{GFX}_3 . The touch mesh S in calibration space \mathbb{TCH}_3 can be used as an accurate starting point for creating G . However, it is typically a dense set of quads that is not suitable for texturing or animating that should be simplified and retopologized first to prevent animation distortions. The graphical model has an associated set of animations, sound effects, and other such output that can be activated in a variety of ways. Here, we focus on enabling such interactivity through user touch input.

For limited applications, such virtual models may not be necessary. As an example, allowing the user to interactively paint a surface does not require a separate graphical model, and in this case the semantic content engine is simply responsible for converting touch input to locations on the projected output.

To increase the types of interactivity available to users, we allow for the virtual content to be stateful, such that the selected output arising from a particular input depends on the content's state. For instance, a user touching a particular region might initiate an animation in one state but a sound effect in another. As such, for the semantic content engine to determine the appropriate response to touch input, it requires knowledge about the touch's location on the virtual model and about the current state. During the modeling phase, we partition the geometric faces of the graphical model G into semantic regions to which names are assigned. Each such region can then be associated with a variety of semantic behavior.

There are two primary forms of touch-triggered responses: *discrete* updates that are triggered upon contact and *continuous* updates that dynamically change as a result of continuous touch input (such as touch-and-drag operations). Discrete updates include predefined animations and sound effects, while continuous updates involve graphical changes to either the pixels in projector space or the actual vertices of the graphical model G . To reduce the amount of computation required by the semantic content engine to initiate these responses, we can further augment the lookup table such that accepted touches can be mapped either directly to semantic output or to data that facilitates the selection of the response. Below, we describe each of these touch-triggered responses in greater detail and present modifications to the lookup table to support this mapping. These incremental updates to the lookup table are performed during step PP8 of the preprocessing phase, when the relationships between the camera and projector pixels and the graphics mesh G are encoded.

- **Animations** are discrete events during which vertices on the graphical model G and/or colors in projection space are transformed in a scripted sequence that begins at the moment of touch input. Each semantic region defined in the space of the virtual content can be associated with a specific animation to be played upon touch input in that region. Such animations could range from user interface aspects, such as indicating the location of a user's touch, to more complicated output. For example, touch input on a physical surface representing the human circulatory system could initiate animations representing blood flow throughout the body. In practice, it is necessary to add a small timeout after an animation begins playing in order to prevent it from restarting again before it has finished due to repeated touch input in that region.

Table 3.7: Lookup table augmented with semantic regions for animations. Touches occurring at a particular camera pixel \mathbf{x}_i are linked to the animation region index stored in \mathbf{R}_i . If $\mathbf{R}_i = 0$, no animation is associated with the i th correspondence.

Row	Cameras $\{C_i\}$	Projectors $\{P_i\}$	Scan	Graphics	Animation Region
1	$\{\mathbf{x}_1^{\{C_i\}}\}$	$\{\mathbf{u}_1^{\{P_i\}}\}$	$\mathbf{X}_1^{\text{TCH}_3}$	$\mathbf{X}_1^{\text{GFX}_3}$	\mathbf{R}_1
2	$\{\mathbf{x}_2^{\{C_i\}}\}$	$\{\mathbf{u}_2^{\{P_i\}}\}$	$\mathbf{X}_2^{\text{TCH}_3}$	$\mathbf{X}_2^{\text{GFX}_3}$	\mathbf{R}_2
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
n	$\{\mathbf{x}_n^{\{C_i\}}\}$	$\{\mathbf{u}_n^{\{P_i\}}\}$	$\mathbf{X}_n^{\text{TCH}_3}$	$\mathbf{X}_n^{\text{GFX}_3}$	\mathbf{R}_n

The information necessary to map touch in a region to the identity of a specific animation to trigger can be directly stored in the lookup table. Each semantic *animation region* is composed of an index number and a set of 3D coordinates on the graphical model G . In step PP8 of the preprocessing phase, the lookup table is densely supplemented with correspondences from camera and projector pixels to points on G by back-projecting all such pixels to 3D rays and finding their points of intersection on G . A camera or projector pixel back-projected to a 3D ray that intersects a particular animation region can thus be linked to it

in the lookup table through the same process. For example, let \mathbf{R}_i be the animation region index of the coordinates in the i th correspondence on the lookup table, where $\mathbf{R}_i = 0$ if the given correspondence is not related to any such animation region. An example of a lookup table augmented to support animation regions is shown in Table 3.7.

- **Sound effects** are discrete events during which prerecorded sound files are initiated at the moment of touch input. They are implemented exactly as animations, with the graphical model G partitioned into predetermined *sound regions*. However, the semantic regions defined for sound effects need not be the same as those defined for animations. As a given 3D coordinate on the virtual content could be associated with both an animation and a sound effect, the lookup table must maintain these region assignments separately. This can be achieved by simply extending the number of region sets: one denoted \mathbf{R}^{Anim} for animations and one denoted $\mathbf{R}^{\text{Sound}}$ for sound effects. This can be generalized as the set of \mathcal{R} regions $\{\mathbf{R}^1, \mathbf{R}^2, \dots, \mathbf{R}^{\mathcal{R}}\}$, which can similarly be added directly to the lookup table such that the i th correspondence now contains links to these \mathcal{R} regions. An example is shown in Table 3.8.

Table 3.8: Lookup table augmented with generic semantic region indices $\{\mathbf{R}^1, \mathbf{R}^2, \dots, \mathbf{R}^{\mathcal{R}}\}$. The i th correspondence is linked to semantic region index \mathbf{R}_i^j for each set of regions \mathbf{R}^j .

Row	Cameras $\{C_i\}$	Projectors $\{P_i\}$	Scan	Graphics	Semantic Regions
1	$\{\mathbf{x}_1^{\{C_i\}}\}$	$\{\mathbf{u}_1^{\{P_i\}}\}$	$\mathbf{X}_1^{\text{TCH}_3}$	$\mathbf{X}_1^{\text{GFX}_3}$	$\{\mathbf{R}_1^1, \mathbf{R}_1^2, \dots, \mathbf{R}_1^{\mathcal{R}}\}$
2	$\{\mathbf{x}_2^{\{C_i\}}\}$	$\{\mathbf{u}_2^{\{P_i\}}\}$	$\mathbf{X}_2^{\text{TCH}_3}$	$\mathbf{X}_2^{\text{GFX}_3}$	$\{\mathbf{R}_2^1, \mathbf{R}_2^2, \dots, \mathbf{R}_2^{\mathcal{R}}\}$
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
n	$\{\mathbf{x}_n^{\{C_i\}}\}$	$\{\mathbf{u}_n^{\{P_i\}}\}$	$\mathbf{X}_n^{\text{TCH}_3}$	$\mathbf{X}_n^{\text{GFX}_3}$	$\{\mathbf{R}_n^1, \mathbf{R}_n^2, \dots, \mathbf{R}_n^{\mathcal{R}}\}$

- Other discrete region-based events can be implemented similarly. Note that animations and sound effects can be combined. For example, an interactive model of a human could verbally identify the body part at the location of user touch via lip-synced audio output.
- **Pixel-level updates** are continuous responses during which color changes are made directly within the images projected onto the touch-sensitive surface. For instance, when a user touches the surface, he or she might receive feedback in the form of a projected white contour surrounding his or her finger. While it is possible to achieve this effect by augmenting the graphical model G directly—for instance, by adding a white sphere at the touch location—it is sometimes preferable to make direct changes to projector space. Pixel-level updates are dynamic in response to continued touch: as the user continues to touch the surface, the feedback contour is updated. The lookup table as previously defined directly facilitates pixel-level updates, allowing for instant conversions of touch locations in camera space to their corresponding positions in projector space.

Such updates typically require transmission of the entire projector contour, which represents the region of the surface being touched by the user directly in projector space. For example, in a painting application, users essentially adjust the actual projected imagery displayed on the surface by touch, achieved by modifying the colors of the projector contour pixels.

In the context of human patient simulation, another example of this type of touch-triggered update is the *capillary refill test* (or *blanch test*) [84]. When pressure is applied to the skin, blood flow is prevented at the point of contact, causing the skin to appear white. Within a few seconds after pressure is released, blood flow returns, and the skin regains its normal color. Similar to the painting application example, this type of change involves the modification of colors of the entire touch contour in projector space.

- **Vertex-level updates** are continuous responses consisting of a transformation of a specific set of graphical model vertices between a starting and ending configuration. In graphical

modeling, these are commonly referred to as **blendshapes**. Examples include the motion of a virtual human’s mouth or eyes as they open and close. Each blendshape has an associated activation percentage, ranging from 0% to 100%, that controls the amount of transformation for the entire set of affected vertices; each vertex moves linearly from its starting to ending position based on this percentage. Internally, each blendshape exists as a separate mesh comprising the exact same vertex ordering and edge connectivity as the graphics mesh G , with the specific blendshape vertices located at their ending positions.

Like pixel-level updates, blendshapes are continuously updated by touch-and-drag operations. For instance, the motion of a virtual human’s mouth could be controlled directly by touch, with the user tugging to move the lips up and down as desired. To permit greater control, each blendshape has two associated regions: a *start* region and an *update* region. Touches that occur in a start region initiate the corresponding blendshape, which then begins updating in response to continuing contact in either of the start or update regions. However, an initial touch within an update region will not initiate a blendshape. The start and update regions are defined and stored just as the semantic regions: that is, they are included in the set of regions $\{\mathbf{R}^1, \mathbf{R}^2, \dots, \mathbf{R}^{\mathcal{R}}\}$.

When a touch event is detected, its corresponding position on the graphics mesh G is available via the lookup table. This 3D position follows a particular 3D line in the graphics space \mathbf{GFX}_3 as the blendshape activation percentage increases, traveling from its starting to ending position; we will refer to this as the *blendshape line*. When subsequent touches in the start or update region are detected, the vector projection of the touch’s position onto this 3D line is computed, from which the appropriate blendshape activation percentage follows.

Thus, we wish to augment the lookup table with information permitting the computation of blendshape lines. Each correspondence in the lookup table stores an associated point on some face of G , denoted $\mathbf{X}^{\mathbf{GFX}_3}$. However, the blendshapes are defined in terms of actual

vertices of G . Thus, we add the *index of the closest vertex* defined on G to $\mathbf{X}^{\text{GFX}_3}$, which we denote $\hat{\mathbf{V}}^G$, to the lookup table. When touch occurs at a starting blendshape region, the touch sensing engine retrieves the index of closest vertex of G . The line connecting the starting and ending point of this vertex provides a reasonable approximation to this region's blendshape line for the user's touch location. Table 3.9 shows the augmented lookup table.

When a touch is detected by the touch sensing system, all relevant data is packaged into a touch message (described in Section 3.3.2) and transmitted to the semantic content engine for processing. Given that the majority of the data is precomputed in the lookup table and transmitted directly, the work required by the semantic content engine at runtime is minimal. The simulation may occupy one of many different states that affect how touches are processed; for instance, touch-triggered audio responses may be disabled in certain modes. With the range of possible responses available, the semantic content engine simply selects the correct one based on the current mode. As an example, for vertex-level updates, this involves selecting the appropriate blendshape and then computing and updating its activation percentage; for sound effects, this involves triggering the appropriate audio clip and initiating a timeout.

Table 3.9: Lookup table augmented with the closest vertex indices of the graphical mesh G . Using these indices, the line along which a vertex travels through a blendshape can be computed.

Row	Cameras $\{C_i\}$	Projectors $\{P_i\}$	Scan	Graphics	Semantic Regions
1	$\{\mathbf{x}_1^{\{C_i\}}\}$	$\{\mathbf{u}_1^{\{P_i\}}\}$	$\mathbf{X}_1^{\text{TCH}_3}$	$\mathbf{X}_1^{\text{GFX}_3}, \hat{\mathbf{V}}_1^G$	$\{\mathbf{R}_1^1, \mathbf{R}_1^2, \dots, \mathbf{R}_1^{\mathcal{R}}\}$
2	$\{\mathbf{x}_2^{\{C_i\}}\}$	$\{\mathbf{u}_2^{\{P_i\}}\}$	$\mathbf{X}_2^{\text{TCH}_3}$	$\mathbf{X}_2^{\text{GFX}_3}, \hat{\mathbf{V}}_2^G$	$\{\mathbf{R}_2^1, \mathbf{R}_2^2, \dots, \mathbf{R}_2^{\mathcal{R}}\}$
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
n	$\{\mathbf{x}_n^{\{C_i\}}\}$	$\{\mathbf{u}_n^{\{P_i\}}\}$	$\mathbf{X}_n^{\text{TCH}_3}$	$\mathbf{X}_n^{\text{GFX}_3}, \hat{\mathbf{V}}_n^G$	$\{\mathbf{R}_n^1, \mathbf{R}_n^2, \dots, \mathbf{R}_n^{\mathcal{R}}\}$

3.3.1.1 Updates to Touch/Hover Classification Algorithms

In addition to augmenting the lookup table, we must further augment the two touch/hover classification algorithms in order to access the additional information required to achieve semantic responses.

3.3.1.1.1 Projection Space Touch Detection

The projection space touch detection algorithm classifies touches and hovers in the coordinate space of the projectors. During this process, camera contours of an observed touch or hover event are converted to projector contours, and ultimately a single projector pixel is chosen to compute correspondences (such as the 3D point on the graphics mesh G corresponding to the touch input). To handle discrete region-based events, such as animations and sound effects, we can directly use the region indices corresponding to this projector pixel in the lookup table. For instance, a touch detected at projector pixel \mathbf{u}_i has an associated region index \mathbf{R}_i^j for some set of semantic regions \mathbf{R}^j , as in the lookup table shown in Table 3.9. Blendshape updates are incorporated exactly as the discrete region-based events: the projector pixel used to compute correspondences for a given touch is further used as a lookup index to retrieve the closest vertex index on the graphics mesh G .

Finally, pixel-level updates exist in projection space and therefore are directly available via the projector contours. That is, a pixel-level update modifies the colors of the projected imagery at the point of contact, which is exactly represented by the projector contours.

3.3.1.1.2 *Plane Sweep Touch Detection*

In contrast, the plane sweep touch detection algorithm classifies touches and hovers using a series of planar coordinate spaces constructed at the location of a potential touch event. Consequently, the manner in which a localized touch is mapped to semantic graphical output is more complex than for the projection space touch detection algorithm. A touch is localized by computing the intersection of the normal of these planes to the touch surface S . By forward-projecting this intersection point onto each camera's image plane, we can retrieve correspondences via the lookup table using each camera's forward-projected pixel as a lookup table index—for instance, to find the corresponding region indices for discrete region-based events. It is possible that the lookup operations for each camera's forward-projection retrieve different region indices; in this case, the algorithm uses the consensus among the retrieved region indices. Blendshape updates are incorporated analogously.

Finally, pixel-level updates are handled in a similar manner as the projection space algorithm, with the camera contours of a potential touch being converted to equivalent contours in projector space via the lookup table. However, the plane sweep algorithm differs in that this conversion is not performed during the normal course of touch detection; thus, supporting pixel-level updates requires a small amount of additional computation that is not required for touch/hover classification.

3.3.2 *Touch Messages*

When the touch sensing system detects a touch via either algorithm, it packages a variety of information regarding it into a *touch message* that is transmitted to the semantic content engine. Each touch message contains the information necessary for the semantic content engine to determine the appropriate output to display to the user; information used by the detection algorithms, such as camera pixels, is not transmitted. The exact response may depend on a variety of factors, such

as the current state of the simulation. As the touch sensing system and semantic content engine are decoupled, touch messages are sent over the network. The general form of a touch message is shown in Figure 3.18. Message sections are listed in brackets.

```

Number of touches  $n$ 
Touch 1
  [Basic information]
    Number of frames tracked  $t$ 
    Status (new touch, existing touch, lost touch)
  [Localization]
    Touch surface  $S$  coordinate  $(X,Y,Z)^{TCH_3}$ 
    Graphics mesh  $G$  coordinate  $(X,Y,Z)^{GFX_3}$ 
  [Semantic response]
    Closest  $G$  vertex  $\hat{V}^G$  at start frame 1
    Closest  $G$  vertex  $\hat{V}^G$  at current frame  $t$ 
    Blendshape index at start frame 1
    Blendshape index at current frame  $t$ 
    Semantic region indices: animation, sound effect, ...
    Projector contour pixels (for each projector)
Touch 2
  ...
...
Touch  $n$ 
  ...

```

Figure 3.18: The general format of a touch message, containing the information needed by the semantic content engine to determine the appropriate semantic response for touch input.

3.3.3 Rendering System

The *rendering system* is responsible for providing output to users. This output can be of a variety of modalities, including visual and audio. To compute the appropriate image to be displayed, virtual cameras are created in the space of the graphical content that correspond to the physical projectors:

that is, they are positioned and oriented with respect to the graphical model equivalently to the physical projectors relative to the physical touch surface. Additionally, each projector’s intrinsic calibration parameters are applied to the corresponding virtual camera. As a result, the manner in which physical projector pixels map to locations on the touch surface is replicated by the virtual cameras, which project analogous locations on the graphical model to analogous pixels in screen space. Therefore, the images rendered by the virtual cameras can be projected by the physical projectors, producing registered imagery of the virtual content on the physical touch surface.

While it is straightforward to apply the position and orientation information from a projector’s extrinsic matrix to a virtual camera, the application of the intrinsic parameters requires further computation: specifically, the construction of a 4×4 3D computer graphics projection matrix that models the behavior of the typical 3×4 computer vision projection matrices produced through the calibration described in Section 3.2.1. An example of this procedure for Unity virtual cameras [141] is described in the Appendix.

3.4 Head-Mounted Display Touch

Our proposed method is also capable of integration with augmented reality (AR) head-mounted displays (HMDs), with virtual imagery overlaid onto the physical touch surface instead of being provided by a rear-mounted projector [70]. In particular, we focus on the Microsoft HoloLens, a state-of-the-art AR HMD [105]. One major benefit of this approach as an input paradigm is that virtual imagery provided by the HMD is not bound to the surface—it can appear anywhere in the user’s environment. This comes at the cost of additional complexity for two primary reasons: the use of IR light in HMD environment-mapping procedures and the alignment of virtual imagery to physical objects.

3.4.1 IR Environment Mapping

As part of its environment-mapping routines, the HoloLens employs a time-of-flight sensor that projects IR light. Unfortunately, these IR projections occur in the same part of the spectrum as the IR light utilized for touch sensing in the proposed approach. This can produce interference with the touch sensing system, in general appearing as high-intensity flashes in the camera imagery across one or two frames. To support the HoloLens as a source of visual content, the touch sensing system analyzes the IR camera imagery for large intensity spikes, discarding any frames that are substantially different from the background models.

3.4.2 Physical-Virtual Alignment

Due to the partial IR transparency of our rear-projection surfaces, the HoloLens is not able to reliably reconstruct a 3D model of them, preventing successful environment-mapping-based alignment of virtual imagery. Instead, one can facilitate alignment by placing physical markers in the environment [106, 120]. This requires the ability to accurately detect and localize the markers and also assumes that the physical relationship between the desired virtual content and the physical markers is accurately established, which increases setup complexity. Instead, we developed a gaze-based method performed directly in the HoloLens—which reports the user’s head position and eye gaze direction within its own internal coordinate space—similar to calibration approaches such as the single point active alignment method [140].

Suppose the virtual content is composed of a set of 3D vertices V within the coordinate space of the HMD. The corresponding 3D coordinates on the physical surface will be denoted P . To align the virtual model to the physical surface, the registration method seeks a transformation from V to

P :

$$\mathbf{R}V + \mathbf{T} = P$$

where \mathbf{R} is a rotation matrix and \mathbf{T} is a translation matrix that together transform the virtual model V to the physical coordinates P . However, the coordinates P are themselves unknown. Using gaze vectors oriented toward reference points on the physical object, we can estimate a subset of P that allows us to estimate an appropriate transform. Let $\{C_i^V\} \subseteq V$ denote a subset of 3D model reference vertices which correspond to this subset of vertices $\{C_i^P\} \subseteq P$ on the physical object. That is,

$$\mathbf{R}C_i^V + \mathbf{T} = C_i^P \quad \forall i$$

Via the lookup table, we can project a visual target C_i^P onto the surface corresponding to each control point C_i^V . While wearing the HMD, we carefully direct our gaze toward each projected control point from two or more distinct orientations. Using the head position data reported by the HMD, we can construct a set of gaze vectors G_i that pass through the control point. Collecting additional gaze vectors produces additional information about the possible position of the actual physical surface point C_i^P . The 3D gaze vectors are not likely to intersect, but we can approximate the position of C_i^P by computing the closest point to the vectors G_i , which we denote X_i .

Now, we have a collection of known virtual control points $\{C_i^V\}$ and a collection of estimated corresponding points $\{X_i\}$ on the physical surface. These points are related by the transformation

$$\mathbf{R}C_i^V + \mathbf{T} = X_i \quad \forall i$$

Since the X_i are estimates of possibly inaccurate gaze vectors, this equation will not generally have an exact solution. Instead, we compute a rotation matrix \mathbf{R} and translation matrix \mathbf{T} that together

minimize the expression

$$\sum_i (\mathbb{R}C_i^V + \mathbb{T}) - X_i$$

using computational optimization techniques. This transformation can then be applied directly to the virtual environment displayed in the HoloLens, which registers the virtual content onto the physical touch surface.

CHAPTER 4: REALIZATIONS

In this chapter, we present our specific realizations of the general method for touch sensing and integrated semantic response on non-parametric rear-projection surfaces described in Chapter 3. First, we cover the overall software architecture, providing practical implementation details for each system component: the touch sensing system, the semantic content engine, and the rendering system. In many cases, we developed performance optimizations over straightforward, naive implementations to reduce execution time throughout the entire pipeline. Additionally, we motivate and explain various aspects that differ in practice from the theoretical framework described previously.

Following this, we discuss the design and construction of two physical prototype touch sensing systems and introduce several rear-projection surfaces tested on each one using this software architecture. Later, in Chapter 5, we will present the graphical content developed for these rear-projection surfaces and demonstrate a variety of touch interactions on each one. Chapter 6 concerns system evaluations of these touch-sensitive surfaces.

4.1 Software

Following the generalizable nature of the proposed methodology, we designed our software architecture to be modular and extensible; some design choices were informed by the specific hardware and software libraries we used. Parameters that are specific to a physical implementation or to a particular surface are instead abstracted to configuration files, maintaining overall software consistency and facilitating extensions to other desired touch-sensitive surfaces or other hardware setups. For example, the coordinates of the feature scan projections, the semantic region definitions, and

the touch detection algorithm parameters are all stored in text-based configuration files. In particular, none of the code places restrictions on the number of cameras, the number of projectors, or the physical size of the touch-sensitive surface.

Overall, we used the following libraries and development environments:

- In general, all live image processing is performed using the C++ interface of the **OpenCV library** [20]. Likewise, we use OpenCV to display certain kinds of output via the projectors, including the feature scan projections from the preprocessing phase and simple projection-space-based semantic content engines.
- Camera acquisition uses the **FlyCapture SDK** [52], also in C++. We designed a wrapper to integrate OpenCV and FlyCapture, which primarily handles the conversion of internal FlyCapture image formats to the OpenCV matrix object and is responsible for managing the set of cameras.
- We used the **Boost C++ libraries** [18] to handle a variety of supplementary routines, such as networking utilities and disjoint set data structures.
- Separately, the majority of the preprocessing phase code used to incrementally create the lookup table was written in **MATLAB** [101].
- The semantic content engines relating to three-dimensional animated virtual models were written in C# for the **Unity** game engine [141].
- All C++ and MATLAB code was developed on Linux systems. To support cross-platform availability, we used the **CMake** build automation system to compile the C++ code [31], while MATLAB code is already cross-platform. The Unity environments were created on Windows systems, with the builds targeting Linux environments.

Other libraries and toolboxes used for individual steps are presented below, along with our own implementation details.

Our practical implementation has one major difference compared to the theoretical framework presented in Chapter 3: whereas the lookup table was described as a single table containing all camera and projector pixels, their correspondences in other coordinate spaces, and other relevant semantic information, we instead create a series of **sub-tables** relating specific subsets of this data. For example, the conversion of a camera (x, y) pixel to a projector (u, v) coordinate is facilitated by two lookup tables, with one returning the u -coordinate and one returning the v -coordinate. There are four primary advantages to enforcing this separation:

- Internally, each lookup table is stored directly as an OpenCV matrix object whose rows and column indices correspond to the desired lookup index and whose elements correspond to the desired lookup table retrieval. As an example, for a camera-to-projector- u lookup table, the internal OpenCV matrix has the same dimensions as the camera, and a camera pixel can be used directly as an index to obtain the corresponding projector u -coordinate.
- In practice, it is unlikely that a set of pixels across multiple devices that correspond will all have *integer* coordinate values, which are required for the lookup. Thus, for each specific table, we can restrict the indices to integer-valued coordinates and only compute correspondences for them.
- Each touch sensing algorithm requires different subsets of the data; for instance, the projection space algorithm requires correspondences between projector pixels and 3D coordinates on the touch surface S , but the plane sweep algorithm is independent of the projectors and therefore does not. By separating the lookup table into sub-tables, we can implement each algorithm such that it only loads the specific relational data necessary for it to function.

- Finally, separating the tables simplifies the aggregation of data of certain types. In particular, finding correspondences between the pixels of one projector to the pixels of another is challenging, since projectors are image display devices rather than image sensors. Neither the preprocessing phase nor the two touch detection algorithms require projector-to-projector correspondences, so they can be omitted.

First, we describe our practical implementation of the preprocessing phase used to generate lookup tables for a rear-projection surface. Afterwards, we present the runtime phase implementation, which processes live camera imagery for touches and determines appropriate semantic responses.

4.1.1 Preprocessing Phase Implementation

Throughout this discussion, we label each implementation step from PP1 to PP8, following the labeling established in the general preprocessing phase description from Chapter 3.

PP1) During the **initial camera calibration** of the preprocessing phase, we capture dozens of images of calibration patterns at various positions and orientations throughout the general working volume of the desired touch-sensitive surface (Figure 4.1). We ensure that at least one such capture has the calibration pattern oriented to match the hypothetical base plane on which the touch surface physically rests; later, in the implementation of preprocessing phase step PP5, we will transform the initial estimate of the touch mesh to rest on this base plane in the calibration space TCH_3 , which will simplify the construction of the final touch mesh S . We obtained the best calibration results using an asymmetric circle grid calibration pattern. Asymmetric patterns are especially useful given the large working volume covered by the collections of cameras, typically at different orientations, as they allow for straightforward detection of the calibration pattern orientation. The calibration itself is performed using the

circle grid detection and subsequent processing routines provided by OpenCV; each camera is first calibrated individually to obtain intrinsic matrix estimates, which are subsequently used as initial estimates for the stereo calibration of every pair of cameras.

To ensure coverage across the working volume, our camera capture utility displays statistics about the number of usable pattern images captured for each individual camera and for each stereo pair of cameras. Not all stereo pairs are useful—for instance, a pair of cameras sufficiently far apart that there exists no overlap in their fields of view. Our capture utility can be customized to highlight these statistics for specific predefined pairs of cameras.

For each physical setup, one camera is specified as the origin of the calibration space \mathbb{TCH}_3 ; stereo calibration results for each stereo pair containing this camera are aggregated such that all camera positions and orientations are represented with respect to the position of the origin camera. To further improve calibration results, we perform bundle adjustment using the `sba` (sparse bundle adjustment) package [97], which globally optimizes the intrinsic and extrinsic matrices of all cameras and a point cloud of the detected calibration pattern points obtained via triangulation. At this stage, we do not perform any distortion correction. For visualization and later processing, we create MATLAB structures to store the intrinsic and extrinsic calibration results.

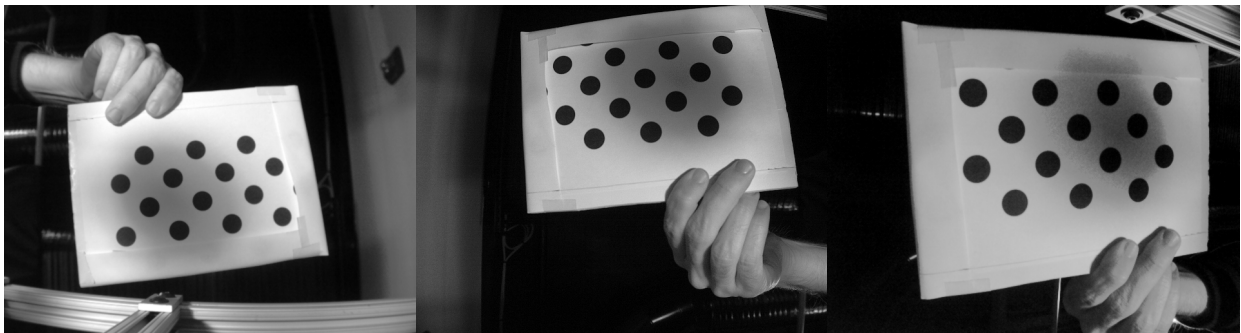


Figure 4.1: Calibration captures of an asymmetric circle grid pattern, shown across three cameras. The asymmetry in the pattern allows for each camera to uniquely determine the pattern’s orientation.

PP2) The **feature scan**—the collection of bidirectional projector-camera and camera-camera correspondences—is implemented in OpenCV, with a grid of features specified in a configuration file for projection on the surface. We experimented with a variety of projected features, including alternating horizontal and vertical lines, circles, and checkerboard patterns; in particular, we investigated structured coded light patterns [126], which encode a large number of features within a single projected pattern. However, while we had success with this approach specifically for the physical-virtual head surface [72], the reliability of decoding such patterns varied greatly across our different rear-projection surfaces due to light interreflections. Instead, to ensure consistency among the results across all tested surfaces, we use individual white circle features, sequentially projected across a grid of specified coordinates in the space of each projector.

Camera captures of the projected features are subsequently segmented in MATLAB, and feature coordinates in projector space are linked to the centroids of the segmented features in camera space in a simple list structure. Each list entry contains a single projector-camera correspondence with the following data:

- the index of the projector that produced the feature,
- the center pixel of the projector feature,
- the index of the camera that observed the feature, *and*
- the centroid of the segmented camera observation of the feature.

Thus, given \mathcal{C} cameras, each projected feature results in between 0 and \mathcal{C} entries in this list structure, depending on how many cameras observed it.

During the feature scan, we also project several test patterns onto the surface, including one comprising all white pixels; these captures are used to create the binary region of interest image masks for the projector-camera pairs.

The correspondence list structure roughly represents the example lookup table shown in Table 3.2. Projector-camera correspondences arise naturally from the individual entries of the structure, and camera-camera correspondences can be obtained by aggregating the projector-camera correspondences for a given projector feature. That is, if projector feature (u, v) is observed as pixel $(x, y)^{C_i}$ in C_i and as pixel $(x, y)^{C_j}$ in C_j , then $(x, y)^{C_i}$ and $(x, y)^{C_j}$ are corresponding points. The list structure is continuously augmented and transformed throughout the following steps, ultimately forming a series of individual lookup tables relating the various coordinate spaces and semantic content behavior.

PP3) Next, we **estimate the initial touch surface S** by triangulating the camera-camera correspondences obtained via the feature scan into a 3D point cloud in the calibration space TCH_3 . As described in the implementation step of PP2, for each projected feature, we aggregate all camera-camera correspondences and subsequently triangulate the correspondences for each pair of cameras. Since certain camera pairs may be too far apart for reliable triangulation, we establish a configuration option to select specific pairs to process, if desired. Each triangulation represents an estimated 3D position for the projected feature on the physical surface, based on observations from a particular pair of cameras. Ultimately, we retain the 3D centroid of these points to create the point cloud estimate of the touch surface S , which we add to the current correspondence list structure. Each correspondence list entry now contains:

- the index of the projector that produced the feature,
- the center pixel of the projector feature,
- the indices of all cameras that observed the feature,
- the centroids of the segmented camera observations of the feature,
- the set of 3D triangulated points for each pair of cameras, *and*

- the 3D centroid of these triangulations.

Thus, the current correspondence list structure roughly represents the example lookup table shown in Table 3.3.

PP4) We implemented maximum-likelihood estimation of projection matrices [64] in MATLAB in order to compute **initial projector calibration matrix estimates**, following the formulation presented in preprocessing phase step PP4. This begins with the Direct Linear Transformation algorithm, which computes an initial estimate of each projector’s projection matrix. We then iteratively refine these estimates by minimizing the geometric error—the sum of the distances between each projected feature and the forward-projection of the corresponding touch surface point cloud coordinate onto the projector’s image plane—with the Levenberg-Marquardt algorithm [92] provided by the non-linear least-squares solver of the MATLAB Optimization Toolbox [102]. For each projector, this results in a projection matrix P , which is decomposed into intrinsic and extrinsic matrices through RQ decomposition (implementation from Peter Kovesi’s MATLAB functions [87]).

PP5) Next, we again use `sba` [97] in the **final bundle adjustment phase** to jointly optimize the camera calibration matrices, projector calibration matrices, and the 3D touch surface S in TCH_3 . For this bundle adjustment, we enable distortion correction routines. As mentioned in Chapter 3, supporting distortion correction throughout the lookup table requires a few more modifications over the abstract version presented. The actual pixels observed in camera imagery (with distortion) are referred to as *observed* pixels, whereas pixels with these distortions corrected are called *ideal* pixels. Undistorting the live imagery from all cameras would incur additional latency; instead, we continue to use observed pixels as lookup table indices. Later, in the implementation of step PP8, we will describe instances for which the ideal pixels are desired. In particular, at this stage, the optimization of the final 3D touch surface S coordinates by `sba` utilizes the ideal distortion-corrected pixels. Additionally, along

with the refined calibration matrices, each camera and projector now has a set of distortion coefficients. Both `sba` and OpenCV support the distortion model used by Bouguet, which employs three coefficients for radial distortion and two for tangential distortion [19].

The maximum-likelihood estimation procedure from step PP4 operates over an entire 3×4 projection matrix and as such places no constraints on calibration matrix entries, such as the skew parameter. However, it is generally desirable to constrain the estimated calibration matrices to have zero skew, as this reflects typical cameras [64]. While `sba` can enforce a zero-skew constraint, in practice, we found that this generally decreased the quality of the calibration, even though overall reprojection errors were minimized. Instead, we developed an iterative approach. On the first iteration, we run `sba` as normal, allowing it to vary all elements of the cameras' and projectors' intrinsic matrices, including the skew parameters. Then, over the remaining iterations, we linearly decrease the skew value in each camera's and projector's intrinsic matrix towards zero and again perform bundle adjustment via `sba`. On the final iteration, we force the skew values to be zero and subsequently run `sba` with the zero-skew constraint active. In practice, we use 10 iterations for all surfaces.

Base plane: As described in the implementation of step PP1, the calibration coordinate space TCH_3 has one of the cameras chosen as its origin, and the 3D touch surface mesh S is positioned and oriented relative to this origin. To simplify both visualization and later processing of S , we transform it such that it rests on the xy -plane, with the vertical axis of the physical surface oriented along the z -axis of TCH_3 . This transformation is facilitated by a calibration pattern capture specifically taken to represent the base plane on which the physical surface is placed. The procedure is as follows:

- First, we retrieve the detected calibration pattern points of the base plane capture and triangulate them in TCH_3 using the refined camera calibration matrices.

- Next, we compute the best-fit plane for these points. We transform the plane so that it occupies the xy -plane of \mathbb{TCH}_3 , with the normal pointing in the positive z -axis.
- This same transformation is then applied to S so that it occupies the xy -plane.

PP6) Given the final refined 3D touch surface S point cloud, we now **create the 3D touch surface mesh S** by connecting its vertices to form edges. To accomplish this, we use `gridfit`, a surface-fitting function that smoothly approximates a surface over a 2D grid [42]. In other words, given a 3D point cloud and a set of query grid x - and y -coordinates, `gridfit` returns a function $z(x, y)$ that approximates the point cloud. The edge connectivity induced by the grid can be used directly in S —that is, we create edges between x -neighbors $z(x_i, y)$ and $z(x_{i+1}, y)$ and between y -neighbors $z(x, y_j)$ and $z(x, y_{j+1})$, forming a set of triangular faces. The transformation of the touch mesh point cloud to the xy -plane of the calibration space \mathbb{TCH}_3 in step PP5 simplifies the approximation of the point cloud as a function over a grid of x - and y -coordinates. The final touch mesh S is composed of the vertex estimates $z(x, y)$ and this associated edge connectivity.

At this stage, we can represent the lookup table of the general form of Table 3.4, which relates camera pixels, projector pixels, and 3D vertices on the touch surface mesh S . However, we will ultimately delay the actual construction of the set of sub-tables to the implementation of step PP8.

PP7) To **align the 3D touch surface scan S and the 3D graphical model G** , we use an iterative closest point algorithm [9] that estimates a rotation matrix R and translation matrix T between S and G .

As described later, several of our semantic content engines are created in Unity. Essentially, these comprise a virtual scene in Unity that matches a given physical setup, with virtual projectors (cameras in Unity) oriented toward the graphics mesh G analogously to the physical

projectors oriented toward the physical touch surface. Thus, we require a means for modeling the physical projectors as cameras in Unity, which we describe in more detail in the Appendix.

PP8) Finally, we compute **dense correspondences from camera and projector pixels to the touch surface S , to the graphical model G , and to semantic regions**. As described in Chapter 3, the estimation of dense correspondences is facilitated by the back-projection of all camera and projector pixels to three-dimensional rays; we compute the intersection points between these rays and various meshes— S , G , and the set of semantic regions encoding touch-triggered behaviors. Other than S , which is located in the calibration space \mathbb{TCH}_3 , these meshes all exist in the graphics space \mathbb{GFX}_3 . Thus, we begin by transforming all meshes to \mathbb{TCH}_3 via the transformation between S and G computed in step PP7.

Next, we loop over each camera and projector, which we will refer to generally as an *optical device* or simply as a *device*. A naive implementation might back-project all device pixels to 3D rays and evaluate intersection points for all such rays across each mesh. This approach is computationally inefficient for two main reasons:

- For a given device, not all pixels will back-project to rays that intersect a particular surface. Any computation time expended in evaluating mesh intersections for such pixels is therefore undesirable.
- Likewise, a device pixel that *does* intersect a mesh will do so only at a specific mesh face (or perhaps faces). However, mesh intersection algorithms must generally consider potential intersections on all mesh faces, and so a substantial amount of computation time is unnecessarily spent. This inefficiency is especially pronounced when considering meshes with tens of thousands of faces, which must each be evaluated for a large number of back-projected pixel rays.

Thus, as a means of substantially decreasing preprocessing time, we compute a small set of *candidate mesh faces* for each device pixel, representing the faces where an intersection between the associated back-projected ray and the mesh must occur. Given this set, we can drastically limit our search for intersections. Internally, the set of candidate faces is stored in a matrix structure whose size matches the dimensions of the optical device and whose individual elements are arbitrary-length lists of face indices.

The calibration data computed throughout the preprocessing phase directly facilitates the construction of a set of candidate mesh faces for a given pixel. Specifically, using the standard equation for the projection of a vertex to a pixel (Equation 3.1), we can project every vertex of a mesh M onto a device image plane \mathbb{P} . A given vertex v of M belongs to one or more faces $\{f_1, f_2, \dots, f_n\}$ of M , and so this immediately implies that these faces are candidates for back-projected ray intersections of the associated pixel. This addresses the two aforementioned inefficiencies:

- Device pixels that *will not* back-project to rays that intersect the surface will have an empty set of candidate mesh faces and therefore will not be processed.
- Device pixels that *will* back-project to surface-intersecting rays will be associated with a small set of candidate faces, so the entire set of mesh faces is not evaluated.

However, this only applies to device image plane pixels to which a particular mesh vertex projects, which may not cover all device pixels. Thus, for each mesh face f , we find the device pixels to which all vertices on f project and compute a rectangular bounding box over this range, including a small buffer to account for any potential calibration errors. We then append f to the set of candidate faces for each pixel in this region.

The back-projection of pixels to rays follows Equation 3.2, which we implemented as a single matrix equation in MATLAB that operates over an entire set of valid pixels. We check

for intersections among the associated set of candidate faces using the line-mesh intersection routine provided by the MATLAB package `geom3d` [91], which we modified to include some performance optimizations and to allow for the parallelized computation of intersections across the entire set of rays. If an intersection for the ray formed by back-projecting pixel (x, y) is found at face index f , we store the 3D intersection point (X, Y, Z) and face index f in the respective lookup tables at the index (x, y) ; the intersection points for graphics space meshes are transformed back from calibration space \mathbb{TCH}_3 to \mathbb{GFX}_3 before storage in the tables. As described previously, these are internally represented by individual lookup sub-tables (i.e. pixel-to- X , pixel-to- Y , pixel-to- Z , and pixel-to-face-index sub-tables).

Furthermore, we compute the index of the closest mesh vertex to the point of intersection. These indices are stored in an associated lookup table, used for certain graphical updates (described in Section 3.3). For semantic regions, we additionally store the corresponding region identification number. The start and update regions for animations are defined such that the update region is a proper superset of the start region. Therefore, when storing region identification numbers in the lookup tables, we assign higher precedence to start regions than to update regions.

Distortion correction: The incorporation of distortion correction routines introduces some complexity into the aforementioned approach, as back-projecting pixels to rays would then use a device calibration matrix \mathbb{P} that models *ideal* (i.e. distortion-corrected) pixels. As a result, using ideal pixels as lookup table indices would subsequently require the distortion correction of all incoming camera imagery at runtime, which would add computation proportional to the number of cameras. Instead, we ensure that all lookup table indexing and retrieval operations involving device pixels operate over observed pixels, and we encode the three-dimensional correspondences for a given observed camera pixel that arise from its associated ideal pixel. In this sense, the distortion correction results are *encoded* into the

lookup tables, requiring no special handling for the touch sensing algorithms described in Chapter 3. However, a few modifications to the above method for computing dense correspondences are required.

First, we project all mesh vertices onto the current device’s image plane. The standard projection model of Equation 3.1 produces a set of ideal pixels. Additionally, we apply the distortion coefficients obtained from the bundle adjustment to these pixels to estimate the observed pixels that would yield these ideal pixels upon distortion correction—in other words, we manually distort the projected ideal pixels. We create a rectangular region of interest (ROI) corresponding to the overall bounding box of these pixels, with some additional padding to account for any calibration errors. The estimated set of observed pixels in this region of interest is denoted \hat{O} . Likewise, this same region of projected ideal pixels forms the set I . The two sets are ordered and indexed identically: that is, the i th pixel of \hat{O} is the estimated observed pixel corresponding to the i th ideal pixel of I .

As before, when computing sets of candidate faces, we loop over all the faces of the current mesh. For each face, we create a rectangular region of interest (with some padding) out of its ideal pixel projections on the device’s image plane. For each pixel in this region of interest, we find its index in the ideal pixel ROI I ; using this same index into the observed pixel ROI \hat{O} produces the observed pixel estimate (x, y) . We then store the current face in the set of candidates for coordinate (x, y) . The rest of the process proceeds as described previously.

Dense camera-to-projector and projector-to-camera correspondences: Finally, we take the correspondences from observed camera and projector pixels to three-dimensional points on the touch surface S and forward-project them onto the image planes of each of the optical devices to estimate dense camera-to-projector and projector-to-camera correspondences. As discussed above, this forward-projection uses a calibration matrix that models ideal (undis-

torted) pixels; since we desire to use observed (distorted) pixels as lookup table indices, we manually distort the forward-projected pixels.

Summary: This entire process results in the computation of data represented in the theoretical lookup table shown in Table 3.9. Overall, correspondences are available from:

- observed camera pixels to observed projector pixels,
- observed projector pixels to observed camera pixels,
- observed device (camera/projector) pixels to 3D coordinates on the touch surface S ,
- observed device pixels to 3D coordinates on the graphics mesh G and to the indices of the closest mesh vertices, *and*
- observed device pixels to indices for each semantic region.

A set of internal lookup sub-tables is created for each class of correspondences, using pixel coordinates in the respective optical device as lookup indices. Each sub-table returns a single output value; for example, there are three sub-tables for each pixel-to-3D-coordinate correspondence: one each for the X -, Y -, and Z -coordinates. Collectively, this set of sub-tables encodes all of the data needed for touch detection and semantic response as described throughout Chapter 3.

4.1.1.1 *Special Case: Plane*

In spite of their geometric simplicity, planar surfaces actually pose a problem for the above implementation: the case when all scene points exist on a single plane is a *degenerate* configuration [64], and computing the maximum-likelihood estimates for the projector calibration matrices fails. This prevents several steps in the above pipeline, including the final bundle adjustment phase (step PP5) and the computation of dense correspondences via the camera and projector calibration matri-

ces (step PP8). To support planar surfaces, we instead interpolate the camera-to-projector and projector-to-camera correspondences obtained from the feature scan (step PP2), along with the camera-to-3D and projector-to-3D correspondences from the initial estimate of the touch surface (step PP3). Interpolation provides a simple model of optical distortion, since it is performed directly on observed pixels; however, the accuracy of this model is limited by the number of features projected. While such interpolation schemes are applicable for non-planar surfaces, they fail to incorporate a significant portion of the three-dimensional context regarding the optical devices and the touch surface modeled by the full preprocessing phase implementation described previously.

4.1.2 Runtime Implementation

Our runtime implementation consists of three primary components: the processing and segmenting of camera imagery, the touch/hover classification algorithms, and the semantic content engines. With the exception of the Unity-based semantic content engines, all of the runtime functionality is implemented in C++, with OpenCV providing various image manipulation routines.

4.1.2.1 Processing Camera Imagery

Live camera imagery is analyzed and segmented into potential touch contours using the image processing functionality of OpenCV. For incoming imagery from a given camera, we first perform background subtraction using the associated background model, retaining only the pixels of the region of interest mask. In practice, the images captured by the cameras and the resulting background-subtracted images are somewhat noisy, so we use morphological noise removal operations. To filter out background pixels, we use our own implementation of *hysteresis thresholding*, which retains both pixels above a high threshold and pixels above a low threshold that are connected to them (popularized as a thresholding technique in the Canny edge detector [26]). Using

OpenCV’s contour processing routines, we segment the thresholded imagery into candidate contours. An example of this process is shown in Figure 3.5. The collection of camera contours across the entire set of cameras is subsequently used as input to the touch/hover classification algorithms.

4.1.2.2 Touch/Hover Classification Algorithms

Next, we cover our implementation of the two touch/hover classification algorithms in C++ and OpenCV.

4.1.2.2.1 Base Touch Detector

We created a generic base touch detector class that handles certain components common to both the projection space and plane sweep touch detection algorithms, such as handling cameras and projectors, region of interest masks, background image models, and network communication. Additionally, this class has helper structures used to interface with the various lookup tables. Historical information about a touch—such as the assignment of a consistent label and the number of tracked frames (see Section 3.2.2.4)—is also maintained by this class, as this information is independent of the specific detection algorithm used. This class is designed to be easily extensible; our implementations of the two touch detection algorithms inherit from it, and other potential lookup-table-based algorithms could likewise extend this base class.

Depending on the desired semantic interactivity and on whether the touch surface is planar or non-planar, the touch detectors must consider one of three possible lookup table types:

1. **Full** lookup tables support the entire set of graphical responses as described previously, with Unity-based semantic content engines.

2. **Limited** lookup tables support touch detection and response when three-dimensional graphical content is not desired—that is, when interactions occur entirely within projection space. For limited lookup tables, the touch detectors need not consider localization on a graphics mesh G or any semantic regions (as discussed in Section 3.3.1).
3. **Interpolated** lookup tables support planar touch interactions. Planar surfaces require separate lookup table construction, as described in Section 4.1.1.1.

Each algorithm uses the same basic structure, starting with the segmentation of camera images, the detection of touches, the assignment of a label to each touch, and the transmission of touch messages to the semantic content engine. Our implementations of the two algorithms largely follow the theoretical presentation of Section 3.2.2; below, we highlight certain interesting aspects of our implementation, including some specific functionality and performance optimizations.

4.1.2.2 *Projection Space Touch Detector*

In practice, the conversion of camera imagery to projector space dominates the runtime requirements of this touch detection algorithm. This is especially true as the number of cameras and projectors increases. As such, we incorporated a few optimizations to reduce computation time.

Using OpenCV, it is possible to transform the pixels of a source image to a destination image using a predefined matrix of pixel mappings. Our earliest implementation of the projection space touch detection algorithm relied on this *remap* functionality. However, it is somewhat inefficient in this case, as it remaps the entire camera images, even though only a small number of pixels represent a potential touch event. Moreover, the remap operation functions in reverse, filling the pixels of the destination images by indexing into mappings into the source images; this leads to additional computation time, as the destination projector images have significantly higher resolutions than

the source camera images. Instead, we convert only the candidate touch contours themselves, performed via sequential lookup table conversions of each contour pixel in camera space to its corresponding pixel in projector space.

Furthermore, we improved the efficiency of many of the projection space operations by performing them on specific rectangular regions of interest rather than the entire projection space images. For example, projector response masks can be naively computed by thresholding and summing each full camera-to-projector image. However, by computing the bounding boxes of each camera-to-projector contour, we can reduce execution time by performing these threshold and sum operations only in regions with nonzero pixels. Likewise, when evaluating the number of cameras that contributed to a response mask contour, we can limit processing to the overall contour bounding box. In practice, these regions of interest are on the order of a few hundred pixels—significantly smaller than the entire 1920×1080 pixel projector images. Such bounding box processing is directly available in OpenCV, requiring only a few bounding box computations and resulting in a substantial speedup.

4.1.2.2.3 Plane Sweep Touch Detector

Our implementation for back-projecting camera pixels to rays and determining their intersections on a set of planes closely matches the theoretical presentation of Section 3.2.2.2. The singular value decomposition used for the initial best fit plane, the convex hull operations, and the planar area calculations are all facilitated by OpenCV routines. To compute intersections between the plane normal vectors and the touch and graphics meshes (S and G , respectively), we implemented the Möller-Trumbore ray-triangle intersection algorithm in C++ [107].

In general, our initial, straightforward implementation of the plane sweep touch detector was already significantly faster than even our optimized projection space touch detector implementation.

As one optimization, when computing mesh intersections, we use the camera-to- S and camera-to- G lookup sub-table correspondences to reduce the search to a set of candidate mesh faces, similar to the discussion of our implementation of preprocessing phase step PP8.

4.1.2.2.4 Sending Touch Messages

Transmitting information about a detected touch from the touch sensing system to a semantic content engine depends on the type of semantic content. For simpler projection-based content engines implemented in C++ and OpenCV, the internal detection structure—containing the coordinates of the touch in projection space and on the touch mesh S —already contains all necessary information to determine a touch response, so these structures are passed around as function arguments as needed.

However, for more sophisticated semantic content engines created in Unity, we use the UDP-based networking capabilities provided by the Boost C++ libraries [18] to send touch messages from our C++/OpenCV touch detector implementations to Unity. Internally, the touch messages (as described in Section 3.3.2) are created as strings storing the relevant information about the detection. As these messages are independent of the specific touch detection algorithm used, the base detector class handles the creation and transmission of touch messages.

4.1.2.2.5 Lookup Table Accesses

Finally, we wrote a small library comprising functions to load the lookup tables into OpenCV matrix structures to allow for indexing operations. This library also includes various routines that process lookup table data, such as converting camera contours to projector space. Each touch

detection algorithm implementation uses a subset of these loading and indexing routines specific to the lookup sub-tables it requires to function.

4.1.2.3 Semantic Content Engines

Next, we describe the implementation of various semantic content engines capable of assigning responses to touch input. These can be organized into three primary categories: projector-space-based, hover-based, and Unity-based. While all such semantic content engines and interaction types could be integrated into a single application, we will discuss each separately.

4.1.2.3.1 Projector-Space-Based Applications

Interactions for *projector-space-based applications* occur entirely within projector space—for instance, allowing users to directly affect the colors or positions of objects projected onto the surface. This paradigm is suitable for simple user interfaces presented on a touch-sensitive surface in which interface elements defined in projector space can be selected and manipulated by touch. For such applications, the only relevant data from the touch sensing algorithms is the projector contour corresponding to a detected touch. The semantic content engine is responsible for comparing the touch contour positions to the interface element positions to determine if one has been touched. For example, in a paint application comprising touch-based color palette and canvas regions, the semantic content engine determines the user's color selection based on touch input in the color palette region; subsequently, it applies the chosen color to the projector space touch contour based on touch input in the canvas region.

4.1.2.3.2 *Hover-Based Interactions*

In addition to distinguishing between touches and hovers, the plane sweep algorithm produces an estimate of the distance from a hover to the surface; the projection space algorithm is not capable of performing this estimation. This allows for *hover-based interactions* in which the user's proximity to the surface provides input to the system. For example, the semantic content engine might modify the color or position of an interface element or play a sound based on the surface-hover distance. For such interactions, the semantic content engine utilizes the distance of the plane with the minimum union area to the surface as an estimate of proximity.

4.1.2.3.3 *Unity Environments*

For interactions with a three-dimensional graphical model, we also designed semantic content engines based on *Unity scenes* [141], which serve as the graphics space \mathbb{GFX}_3 . These scenes comprise the following components:

- The graphics model G is imported and augmented with various mechanisms for controlling animations, sound effects, and other interactive capabilities.
- Virtual cameras are placed in the scene to simulate the physical projectors—that is, with the same intrinsic and extrinsic calibration parameters. The images rendered by these virtual cameras are sent to the rendering system, which projects them onto the physical surface using the projectors. This is facilitated by the conversion of projector calibration data to Unity camera projection matrices, described in the Appendix.
- Additionally, each scene has an object responsible for receiving and interpreting touch messages, discussed later.

Here, we focus on implementations of the various graphical responses presented in Section 3.3.

- **Semantic regions:** The graphics model G in Unity has associated sets of semantic-region-based behaviors, such as specific animations and sound effects, to trigger upon touch input in the appropriate region. Each received touch message includes an identification number for each type of region; thus, in Unity, the only computation required corresponds to the initiation of the behavior at the given index.
- **Pixel-level updates:** Such updates modify the colors of pixels rendered by the Unity cameras. The graphics model G is textured with a particular base texture B . In addition, we create a touch texture T to apply to G exclusively at the location of detected touch input. During rendering, the relative opacity of the touch texture T is controlled by a mask M : the final rendered texture is given by $B + (M \odot T)$, where the operation \odot represents the Hadamard product (i.e. element-wise matrix multiplication). When M is the zero matrix, the base texture B is applied to G ; when M is a matrix whose elements are all 1, the touch texture T is instead applied. Mask values in between 0 or 1 apply T with a corresponding amount of opacity over B . For instance, given a touch texture T composed of entirely red pixels and a mask M with values all equal to 1, this operation will modify the texture of G to be red at the specific pixels corresponding to touch input.

However, detected touches are localized in the space of a particular projector—that is, the *screen space* of the corresponding virtual Unity camera simulating this projector—and not in the *texture space* of the graphics mesh G . As such, for \mathcal{P} projectors, we need a corresponding set of \mathcal{P} masks $\{M_1, M_2, \dots, M_{\mathcal{P}}\}$, and we apply these masks to the rendered output rather than to the texture images themselves. We use a custom shader that applies this texture masking operation directly in screen space independently for each virtual camera, since the projector

space pixels corresponding to a touch are only appropriate for that specific projector. The render loop for each Unity camera C is modified as follows:

- Prior to rendering, camera C applies its respective mask M_C , constructed based on detected touch input localized in the space of the projector C represents.
- Virtual camera C renders an image, which reflects the corresponding modified texture of G .
- Following rendering, camera C applies a zero mask so that the base texture B is applied to G . This prevents interference for the remaining virtual cameras, which would otherwise incorrectly apply the touch texture T at the screen space pixels corresponding to the projector C represents.

During continuous touch input, each mask M_C updates dynamically. If desired, once touch input ceases, M_C can be set to linearly approach zero over time so that the touch texture slowly fades, gradually restoring the base texture. This allows for effects such as the simulation of capillary refill, in which blood flow is temporarily restricted due to the application of pressure, resulting in a white appearance; blood flow resumes once the pressure is removed, indicated by the return of normal color over a few seconds.

The touch masks are created as images with the same resolution as the projector, with the pixels corresponding to detected touch contours set to 1. Rather than sending either full touch masks or full projector contours in the network touch messages, we implemented a standard scanline-based flood fill algorithm in the touch sensing system, which constructs and transmits a list of mask rows in the form of (y, x_1, x_2) triplets. In Unity, all mask pixels between each row—from (x_1, y) to (x_2, y) —are filled.

- **Vertex-level updates:** Such updates are implemented as blendshapes in Unity. As with semantic regions, each blendshape in Unity has an associated identification number; touches

detected in corresponding regions are sent to the semantic content engine in messages containing these identification numbers. The computation of the blendshape activation percentage, described in Section 3.3.1, is performed directly in Unity once a touch in a blendshape region is received, using the lookup table correspondences to the nearest vertex on the graphics mesh G . Each blendshape has two associated regions—a *start* and an *update* region. In Unity, we use the number of tracked frames for a given touch to determine whether it is a new or existing touch; only new touches in a start region initiate a blendshape, while subsequent touch input in either the corresponding start or update region will update it.

The Unity-based semantic content engines also store semantic states, which can be used to change how touch input is mapped to output behavior. For instance, in the context of virtual patient simulators, these states can correspond to the patient’s condition, which may affect how they respond to user touches. Due to the decoupled design, the touch sensing system remains agnostic to this state, transmitting all detected touch input for the semantic content engine to process accordingly.

4.1.2.3.4 Receiving Touch Messages

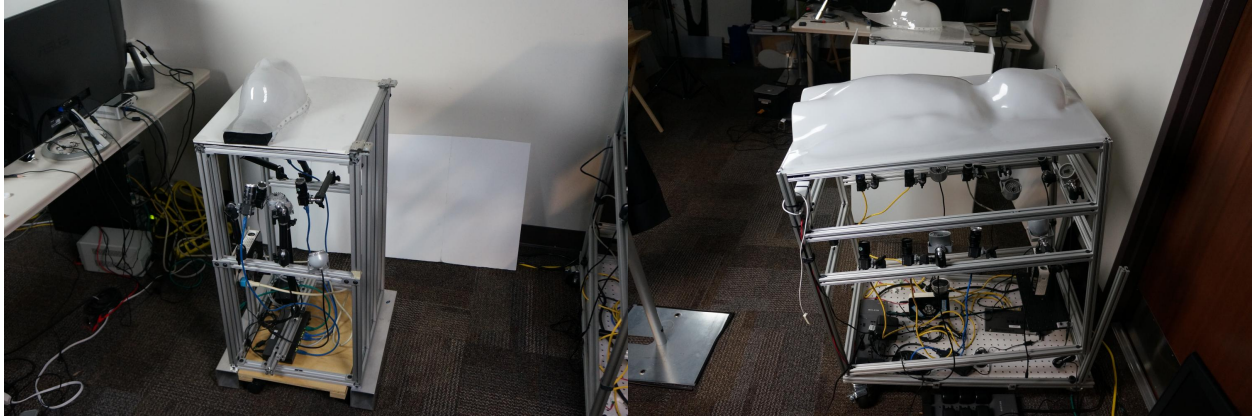
As described previously, there are two primary mechanisms for receiving the touch messages transmitted by the touch sensing system. Basic semantic content engines implemented directly in C++/OpenCV, such as those for projector-space-based applications and hover-based interactions, simply pass touch detection structures as function arguments.

As the Unity-based semantic content engines operate independently, we rely on UDP networking utilities in the Boost C++ libraries for transmission [18]. Accordingly, we implemented an asynchronous function in Unity that polls for UDP messages as a non-blocking operation. When a touch message is received, it is split into its constituent components: the touch’s label, position in

projector space, etc. (see Section 3.3.2). These components are then passed to various functions in the semantic content engine for processing. The asynchronous receive function cannot directly use certain Unity functionality, such as initiating touch-triggered animations, since it does not operate on the main thread. Instead, we create a queue structure holding such tasks to perform, which are then sequentially executed in the main Unity update thread.

4.2 Hardware

Next, we describe two physical touch sensing prototypes, shown in Figure 4.2. Our initial development of the overall touch sensing method was carried out on Prototype Rig I (Figure 4.2a), which supports touch-sensitive rear-projection surfaces around 20×30 centimeters in size using imagery from one projector. Specifically, we focused on touch interactions on a physical-virtual head surface [69,70,71,72], including for use in patient simulation scenarios [33,57,146]. We also explored the application of touch sensing to other rear-projection surfaces, described in Section 4.2.2. As we refined our algorithms and overall software architecture, we continued further development on the larger Prototype Rig II (Figure 4.2b), capable of supporting touch sensing on larger rear-projection surfaces around 50×70 centimeters in size, which we present in Section 4.2.3. One of the core goals when shifting development to Prototype Rig II was extending the method to handle multiple projectors, and we will later show touch interactions on a child-shaped surface with imagery provided by two projectors in Chapter 5. Additionally, this second prototype was designed to support human-subject studies related to healthcare training [35, 36]. We begin with a discussion of the hardware elements common to both of the physical prototypes.



(a) Prototype Rig I.

(b) Prototype Rig II.

Figure 4.2: Our two prototype rigs supporting touch sensing on various rear-projection surfaces. Each has a collection of rear-mounted infrared light sources, infrared cameras, and projectors.

4.2.1 *Common Elements*

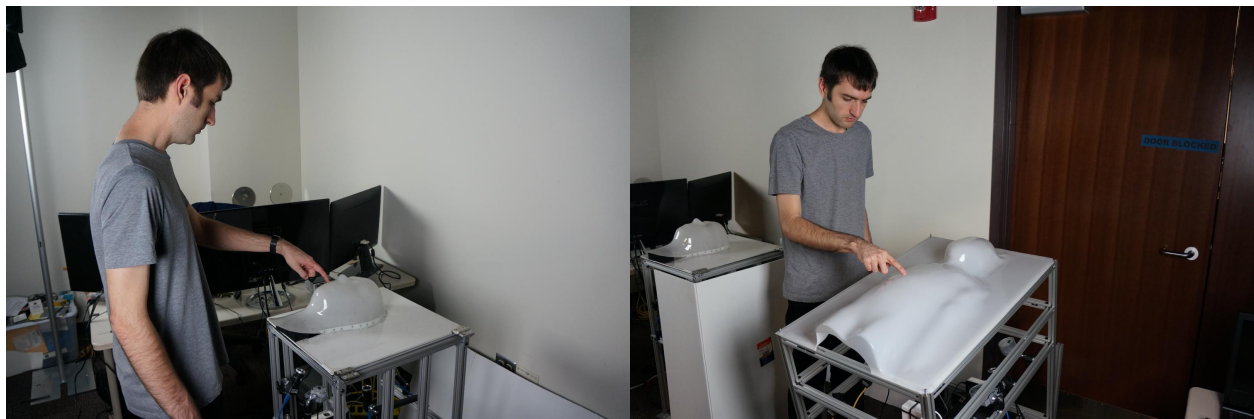
For both physical prototypes, we designed metal frames using 80/20, a modular extruded aluminum framing system [1]. The desired touch-sensitive rear-projection surface rests on top of the frame, positioned at a height suitable for interaction by standing users (Figure 4.3). Cameras, lights, projectors, and other equipment can be mounted to the sides of the frames. Additionally, we attached wheels to the bottom of each frame to allow for mobility.

For image acquisition, we use a collection of Point Grey Blackfly (BFLY-PGE-13E4M) GigE cameras¹, powered by Power-over-Ethernet switches. Each camera has a maximum supported resolution of 1280×1024 pixels, though we set them to 640×512 pixels to reduce bandwidth requirements. These cameras belong to two categories, depending on their function:

¹Point Grey has since been acquired by FLIR; they continue to sell products in the Blackfly line [15].

- The imagery from *touch sensing cameras* is analyzed to determine the presence and location of user touch input. As we rely on the infrared spectrum for this processing, these cameras must be able to image infrared light.
- *Reconstruction cameras* aid in the calibration (preprocessing phase step PP1) and reconstruction of the touch surface (step PP2, the projected feature scan of the physical touch surface). These cameras are not used for touch detection and therefore need not be able to sense infrared light.

Maintaining this distinction allows for the use of additional cameras to improve surface reconstruction results without the necessity for infrared light sensing. The touch sensing cameras are additionally used in the calibration and surface reconstruction process, as they are capable of sensing visible light. Prototype Rig I uses four touch sensing cameras, while Prototype Rig II uses three touch sensing and four reconstruction cameras.



(a) Prototype Rig I.

(b) Prototype Rig II.

Figure 4.3: User interaction with the two prototype touch sensing rigs.

For touch sensing, we add removable infrared (IR) filters (wavelength 780 nm) to eliminate interference from light in the visible spectrum; the filters are not used during the calibration procedure so that visible light for calibration pattern captures and the feature scan can be detected. IR light is provided by commercial off-the-shelf IR illuminators (wavelength 850 nm). The specific model we use has a sensor that dims IR output in the presence of sufficient visible light, which we physically block so that IR light is always provided. In terms of rendering output, virtual imagery is provided by AAXA P300 pico projectors [2] at a resolution of 1920×1080 pixels, and audio is played through standard computer speakers. Prototype Rig I and Prototype Rig II use one and two rear-mounted projectors for imagery display, respectively.

4.2.2 *Prototype Rig I: Small*

Our first prototype rig is designed to support touch-sensitive rear-projection surfaces around 20×30 centimeters in size, with imagery provided by a single projector. Up to four IR cameras and four light sources are used for touch detection, depending on the surface.

In particular, we explore touch sensing on the following three rear-projection surfaces using this prototype rig, summarized in Figure 4.4.

1. The **plane surface** is a flat rear-projection surface made of white acrylic that allows for projected imagery and IR light transmission (Figure 4.4a).
2. The **bowl surface** is a roughly hemispherical rear-projection surface made from a clear, plastic bowl (Figure 4.4b). We coated the bowl with a semi-transparent frosted glass spray paint, which we experimentally determined supports both rear-projection imagery and IR light transmission.



(a) The plane surface.



(b) The bowl surface.



(c) The head surface.

Figure 4.4: Touch-sensitive surfaces on Prototype Rig I. Each supports the transmission of IR light and allows for rear-projected imagery, permitting integrated touch sensing and response.

3. The **head surface** is a head-shaped rear-projection surface (Figure 4.4c) created by People-VisionFX [115]. The surface is coated with a proprietary material that produces extremely sharp projection images and permits the transmission of IR light.

Prototype Rig I was specifically designed with camera and projector positions to support touch sensing and display on the head surface. However, the setup is sufficiently general that it can be applied to the plane and bowl surfaces with minor tweaking to software camera parameters and to IR light positions. In particular, we use the same initial intrinsic and extrinsic camera calibration for all three surfaces, since the camera positions are constant; the projector calibration and final bundle adjustment depend on the surfaces, and so we perform these steps for each one independently. Each surface therefore has its own associated set of lookup sub-tables, created using the general software architecture presented in Section 4.1.

4.2.3 *Prototype Rig II: Large*

Our second prototype rig (Figure 4.5) is designed to support larger touch-sensitive rear-projection surfaces up to 50×70 centimeters in size, with imagery provided by two projectors. Seven cameras are mounted to the rig, including three IR cameras for touch sensing and four cameras for calibration and surface reconstruction. Six IR light sources are used for touch detection. Additionally, this prototype is outfitted with hardware components intended for more realistic physical-virtual patient simulation, particularly in the context of human-subject studies [35,36]. This includes five space heaters to simulate high patient temperatures and audio-haptic devices for the simulation of pulse from pre-recorded MP3 files.

Our main focus for Prototype Rig II was the application of touch sensing to a **child-shaped** rear-projection surface, created from a similar white acrylic as the plane surface on Prototype Rig I.

Like the other surfaces, it supports projected imagery and allows for the transmission of IR light. However, it is considerably larger than these surfaces.



Figure 4.5: The touch-sensitive child surface on Prototype Rig II.

CHAPTER 5: DEMONSTRATIONS

In this chapter, we present demonstrations of the general touch sensing and semantic response methodology of Chapter 3 on the four physical-virtual surfaces introduced in Chapter 4. For each, we begin with results from the construction of the lookup table in the preprocessing phase, including numerical calibration results and visualizations of the calibration coordinate spaces. Additionally, we provide more details on the semantic content engines developed for each surface, with accompanying illustrative examples of actual touch interactions. As explained in Chapter 4, the same software architecture is used to realize all of the following results.

First, we cover the two simpler surfaces: the *plane* and the *bowl*. For these two surfaces, the semantic content engines do not have associated 3D graphical models; instead, they operate directly in the space of the projected imagery in the context of simple touch-based user interfaces, such as a surface painting application and an object drag and drop application. Additionally, we investigated off-surface hover interactions for the plane to create a proximity-based musical instrument, based on the hover distance estimations provided by the plane sweep algorithm. The latter two surfaces—the physical-virtual *head* and *child*—are intended for use in healthcare simulation scenarios, and so each has a sophisticated virtual patient model with textures, animations, sound effects, and other interactive components.

5.1 Plane

To demonstrate touch interactivity on simple parametric surfaces, we first considered a planar surface, shown in Figure 5.1. Though we refer to it as a single surface to simplify the discussion, it consists of two planar objects: one sturdy piece of plexiglass for support and one thin white acrylic

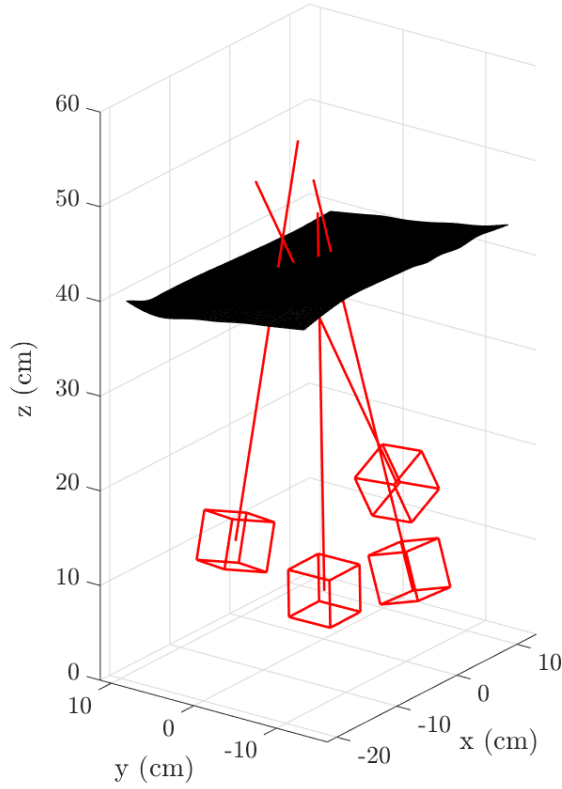
sheet that supports projected imagery. The combination of materials still permits the transmission of IR light and is therefore suitable for touch sensing. The plane rests on a wooden support structure that elevates it to a comfortable position for use, corresponding to the focus of the single projector in Prototype Rig I.

5.1.1 Preprocessing Phase

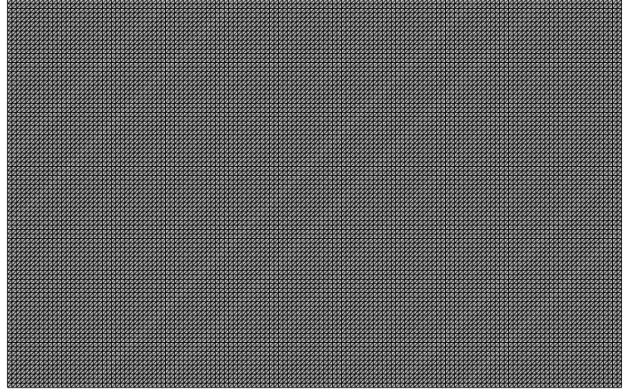
As discussed in Section 4.1.1.1, the planar surface poses a degenerate condition and therefore prevents the projector calibration and final bundle adjustment steps of the preprocessing pipeline. Accordingly, we present only the initial bundle-adjusted camera calibration results. The four cameras of Prototype Rig I were calibrated with reprojection errors of 0.255, 0.190, 0.291, and 0.254 pixels, respectively (mean 0.247 pixels). Figure 5.2a shows the calibration space \mathbb{TCH}_3 for the plane. A total of 189 features were projected during the feature scan. The touch mesh S , comprising 11919 vertices and 23392 faces, is shown in Figure 5.2b.



Figure 5.1: The touch-sensitive plane.



(a) Plane calibration space \mathcal{TCH}_3 .



(b) Plane touch mesh S .

Figure 5.2: Plane surface preprocessing phase. In the calibration space \mathcal{TCH}_3 (a), the four cameras of Prototype Rig I are represented by the red cubes.

5.1.2 Semantic Content Engines

For the planar surface, we created two simple semantic content engines that operate directly in projector space to demonstrate touch activity. The first is a painting application, with a color palette placed in a specific region within the projector’s imagery. Each provided color can be selected by touch, and the currently chosen color is highlighted with a white border. Subsequent touch-and-drag operations outside of the palette region “paint” the projected imagery with the chosen color.

Color selection and paint operations are both facilitated by the lookup table correspondences, with the user's touch as imaged by the cameras being converted to a contour in projector space. If the user touches the region corresponding to the palette, the semantic content engine determines if the touch contour centroid has occurred within any of the palette colors, updating the currently chosen color as appropriate. Paint operations occur outside the palette region: the entire touch contour in projector space is filled with the selected color. Several examples of this painting application are shown in Figure 5.3.

Additionally, we created a related application enabling drag-and-drop-based movement of colored objects in projector space, shown in Figure 5.4. The underlying principle is nearly the same: once a touch has been detected, the semantic content engine determines if it occurs within any of the colored objects. If so, the object becomes *active*—indicated by a white border—and subsequent dragging motions on the surface move the object to the point of contact. Once the user releases his or her touch, the object becomes inactive.

Finally, we developed a specialized semantic content engine supporting *hover-based* interactions with the plane surface via the hover distance estimations provided by the plane sweep touch sensing algorithm. This application is inspired by the *theremin*, an electronic musical instrument that emits sounds whose pitch and volume are determined by the proximity of the user's hands to two antennas [132]. In our theremin-like instrument, the estimated hover distance is translated to the frequency of a note output by a sound synthesizer, such that larger hover distances correspond to notes with higher pitches. Examples are shown in Figure 5.5, for which the user controlled the instrument's pitch with a wooden rod; the current hover distance estimation and a graph of the output frequency over time are projected onto the surface. Here, the plane sweep algorithm evaluated 150 planes, each spaced 1 mm apart, supporting high-resolution hover distance estimation and thus high-resolution note frequencies.

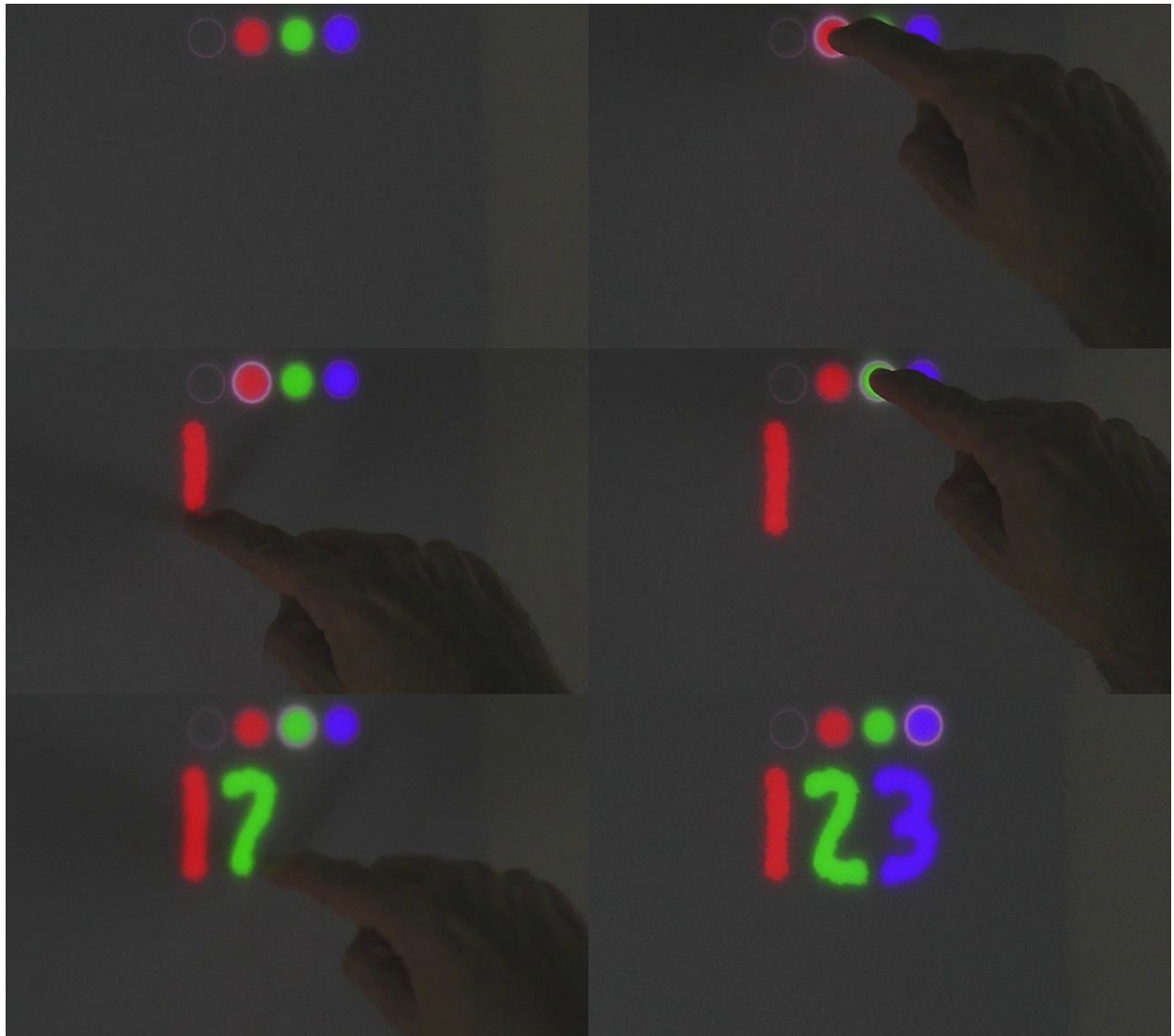


Figure 5.3: A touch-based painting application on the plane surface. The user can select a color from a provided color palette by touch; the selected color is highlighted with a white border. In the figure, time progresses from left to right and from top to bottom.

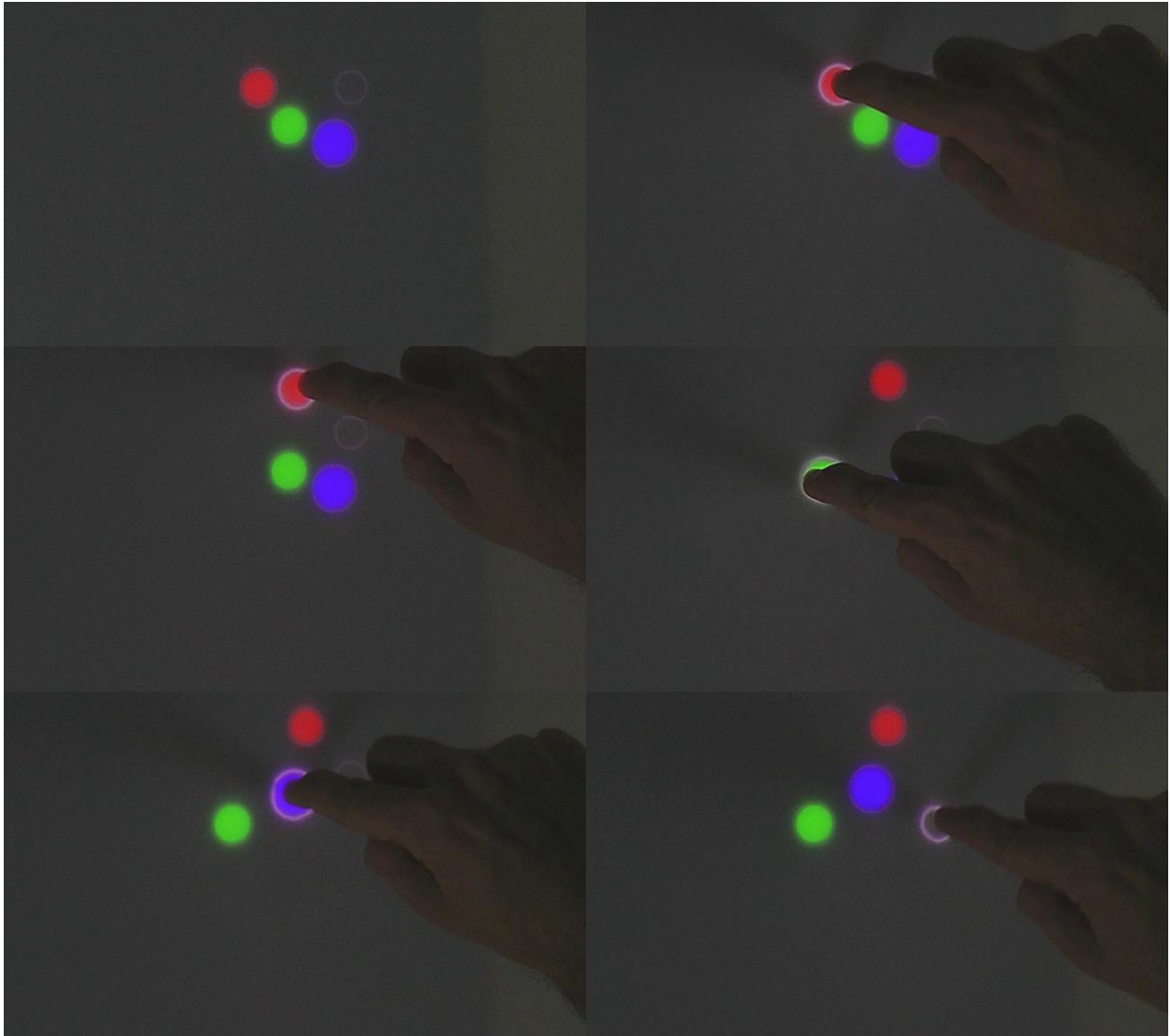
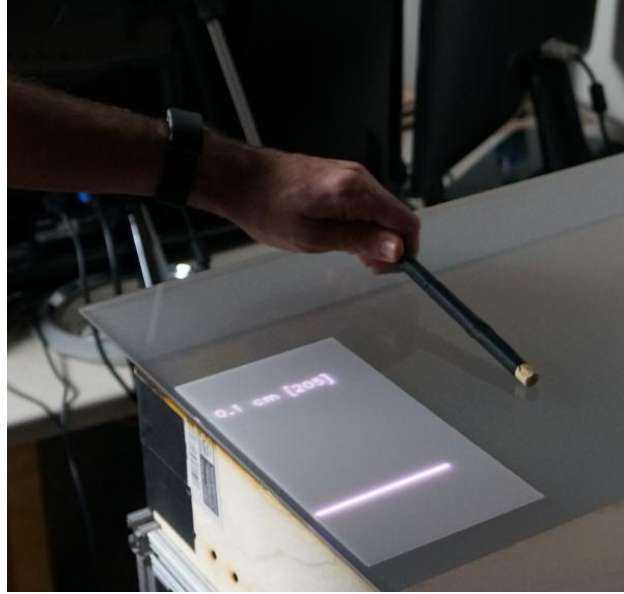
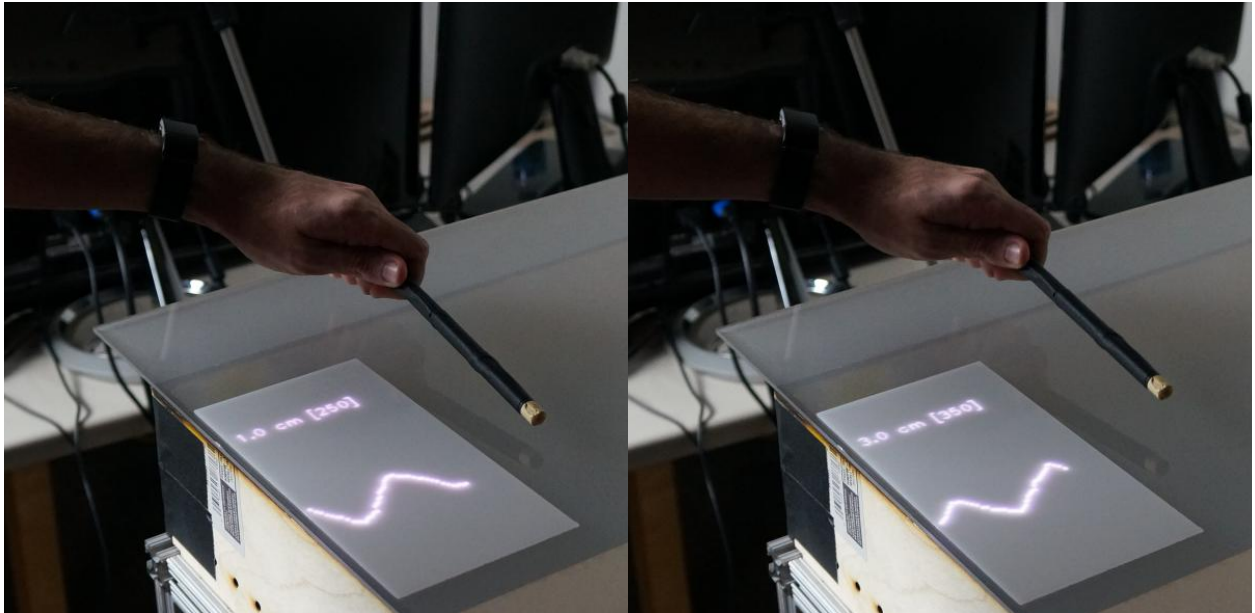


Figure 5.4: A touch-based drag and drop interface for the plane surface. The user can move each individual colored circle by touching it and dragging it across the surface. The currently selected object is highlighted with a white border. Time progresses from left to right and from top to bottom.



(a) Estimated hover distance: 0.1 cm



(b) Estimated hover distance: 1 cm

(c) Estimated hover distance: 3 cm

Figure 5.5: A theremin-like musical instrument interface based on hover distance from the plane surface, estimated by the plane sweep algorithm. As the wand moves farther from the surface, the emitted tone increases in pitch. The projected graph shows the current height estimate and the output frequency over time.



Figure 5.6: The touch-sensitive bowl surface.

5.2 Bowl

Next, we demonstrate touch interactivity on a roughly hemisphere-shaped surface: a clear plastic bowl spray-painted with frosted glass spray paint which allows both for projected imagery to form on the surface and for the transmission of IR light for touch detection. Like the plane, it is a simple, intuitive surface, and we experimented with similar touch-based applications. Figure 5.6 shows the general setup, with imagery provided by the single projector of Prototype Rig I. The bowl is placed upside down, with its flat bottom facing the user. It rests on a flat support surface, firmly mounted to Prototype Rig I.

5.2.1 Preprocessing Phase

During the feature scan, 2165 features were projected onto the bowl and triangulated in the calibration space \mathbb{TCH}_3 using an initial camera calibration. From these triangulated points, we computed a maximum-likelihood estimate for the projector’s projection matrix. After the final bundle adjustment, the four calibrated cameras of Prototype Rig I had reprojection errors of 0.147, 0.156,

0.193, and 0.172 pixels, respectively (mean 0.167 pixels), and the projector had a reprojection error of 0.094 pixels. The calibration space \mathbb{TCH}_3 and the final calibrated cameras and projectors are visualized in Figure 5.7a. The final touch mesh S (Figure 5.7b) has 7516 vertices and 14700 faces.

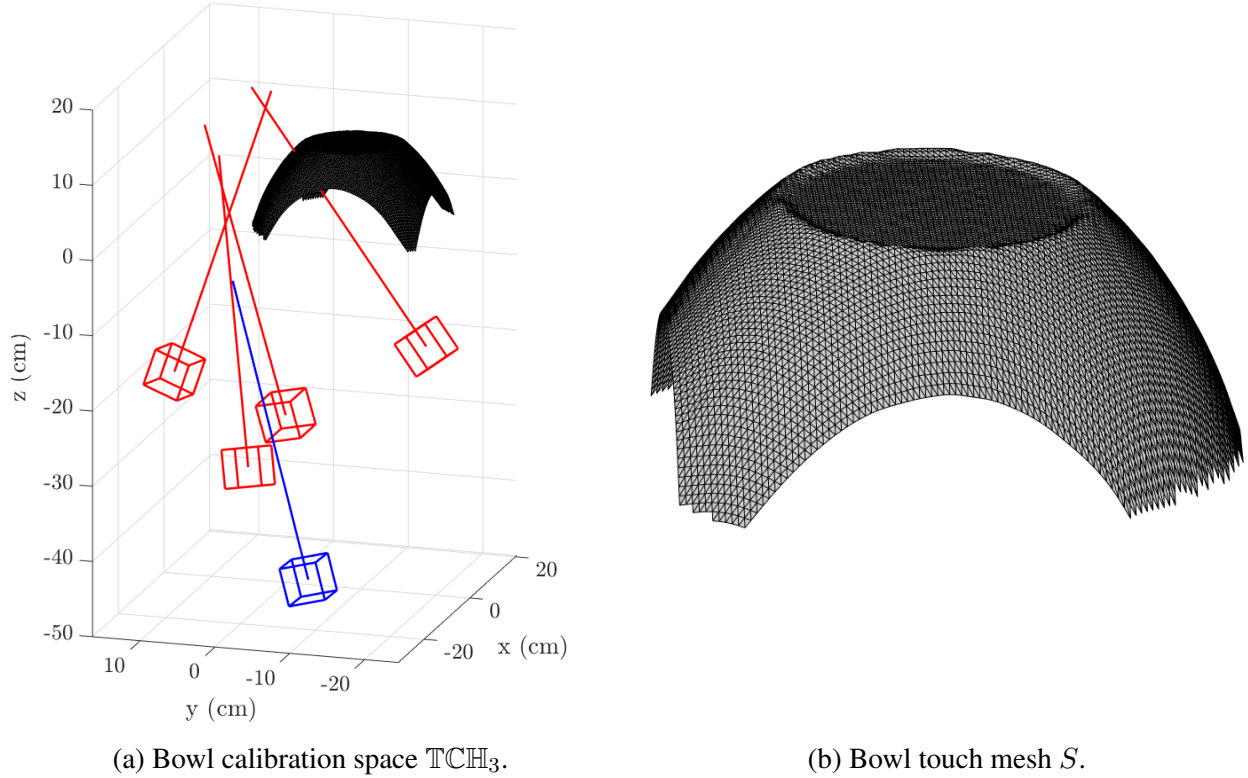


Figure 5.7: Bowl surface preprocessing phase. In the calibration space \mathbb{TCH}_3 (a), the four cameras and one projector of Prototype Rig I are represented by the red and blue cubes, respectively.

5.2.2 Semantic Content Engines

First, we used the same simple touch-based semantic content engines developed for the plane surface: the painting application and the drag-and-drop-based movement application. For the painting application, shown in Figure 5.8, the only difference from the plane surface implementation is the

location of the color palette in projector space—the flat region of the bowl. The second object movement interactivity application is shown on the bowl in Figure 5.9. In both cases, interactions are possible between the flat and curved portions of the bowl—for instance, the user can begin painting in one region and cross over to the other. As touches and interface elements both exist directly in projection space, no changes are necessary in the semantic content engines to support the bowl surface as compared to the implementation for the plane surface.

Appealing to the parametric nature of the bowl, we created a final semantic content engine based on contour lines. The bowl is divided into a series of concentric circles, beginning in the flat portion and proceeding throughout the curved sections. When the user touches the bowl, the corresponding concentric circle is highlighted, as shown in Figure 5.10. Each “circle” is defined in the three-dimensional space of the triangulated feature scan as a set of connected vertices. Prior to runtime, the semantic content engine forward-projects each contour circle onto the projector’s image plane, storing their locations in projector space; the images in projection space are distorted ellipses that appear as circular contours when projected onto the bowl. Thus, when touches occur, the semantic content engine simply determines which contour contains the user’s touch and highlights it.

5.3 Head

The preceding two surfaces are relatively simple, with correspondingly simple semantic content engines. Next, we investigate touch sensing on a more complicated non-parametric surface: a human head-shaped plastic shell, which we refer to as the *physical-virtual patient head* (or simply as the *head* for short). The surface itself is composed of a proprietary material that supports sharp projected imagery and is based on an accurate, high-frequency model of a human head, created by PeopleVisionFX [115]. Figure 5.11 shows the head surface, with touch sensing and projected imagery supported by Prototype Rig I.

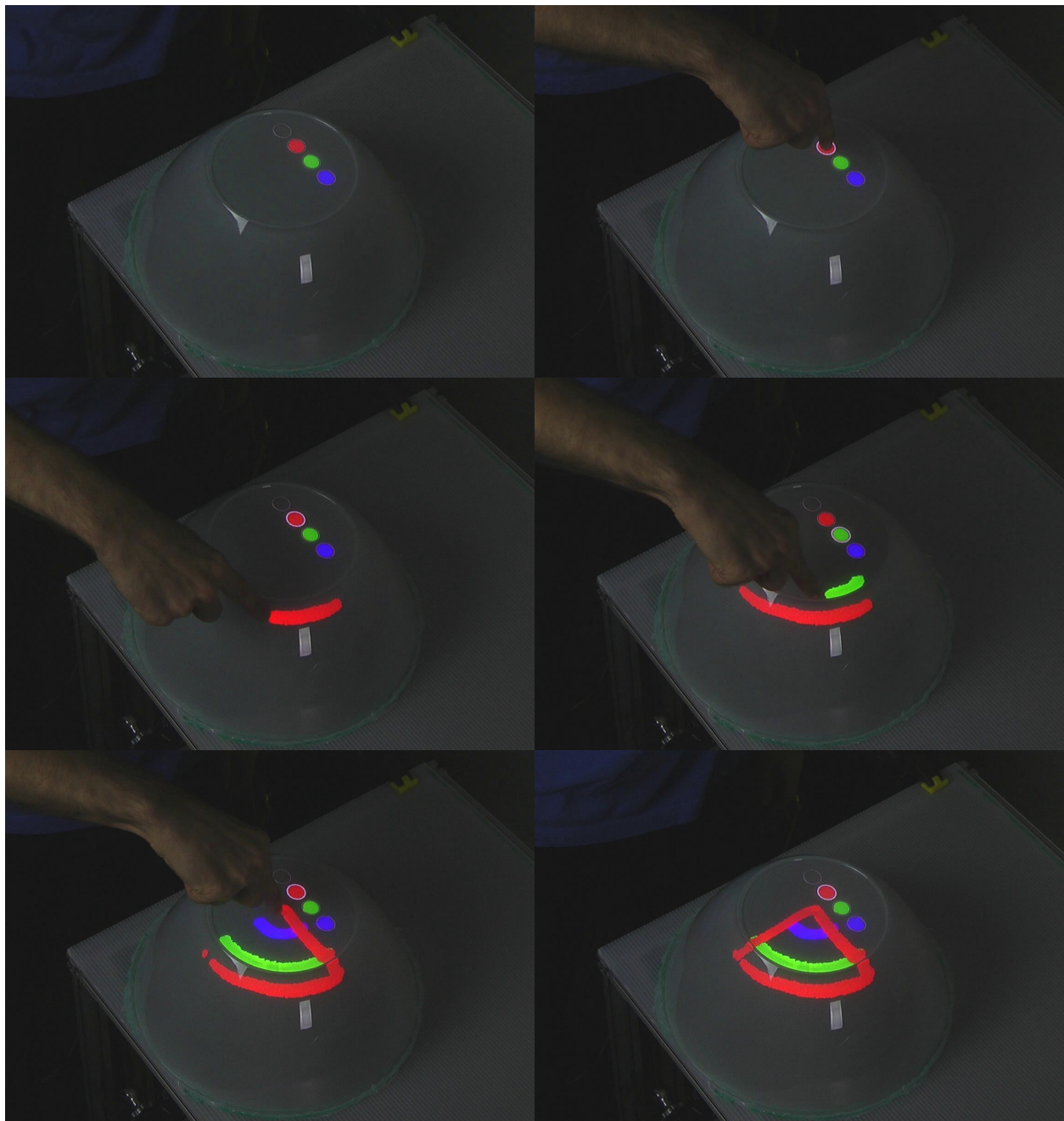


Figure 5.8: A touch-based painting application on the bowl surface, with user-selectable colors, similar to the application shown on the plane surface. Time progresses from left to right and from top to bottom.

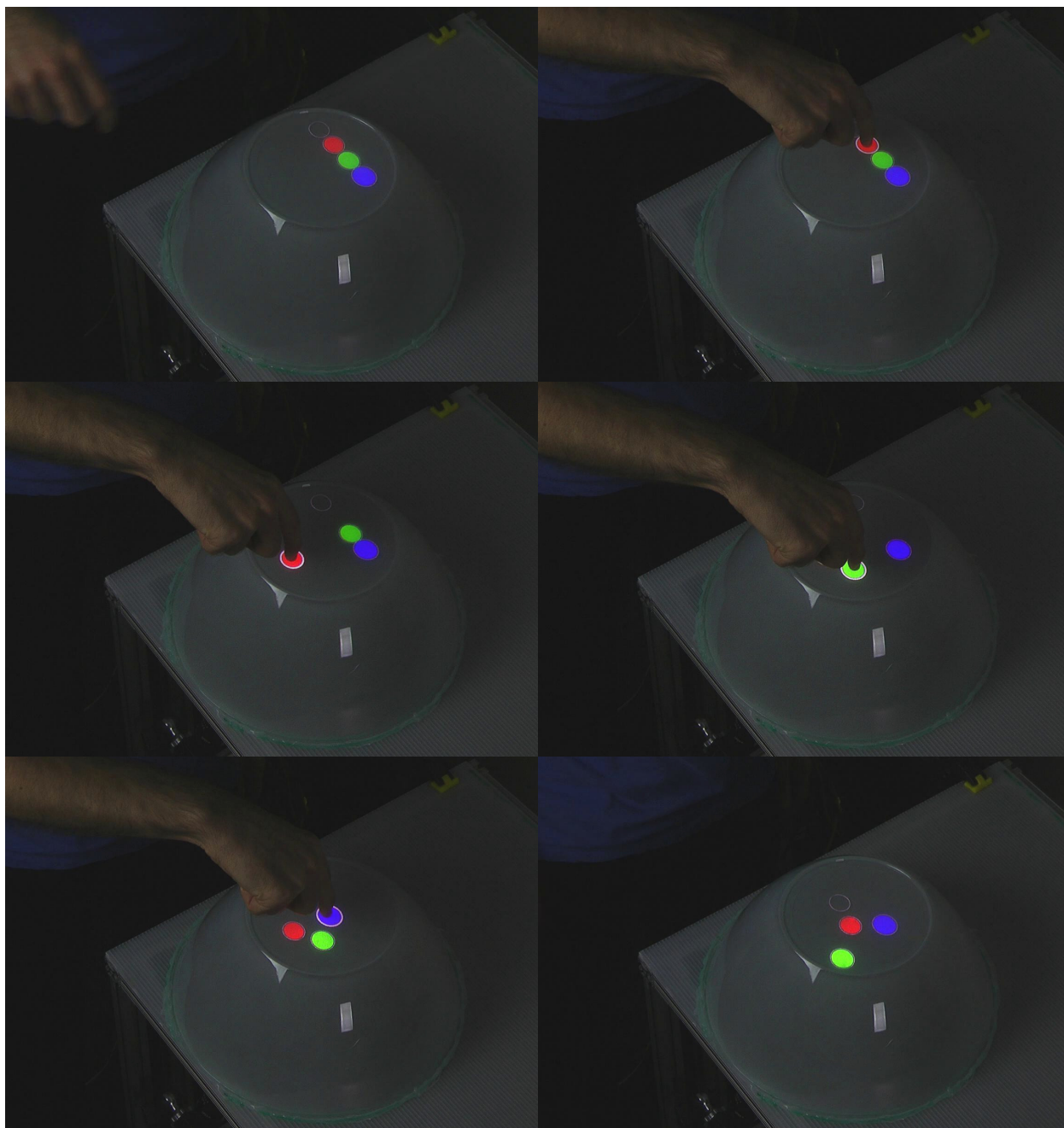


Figure 5.9: A touch-based drag and drop interface for the bowl surface, similar to the one shown for the plane surface. Time progresses from left to right and from top to bottom.

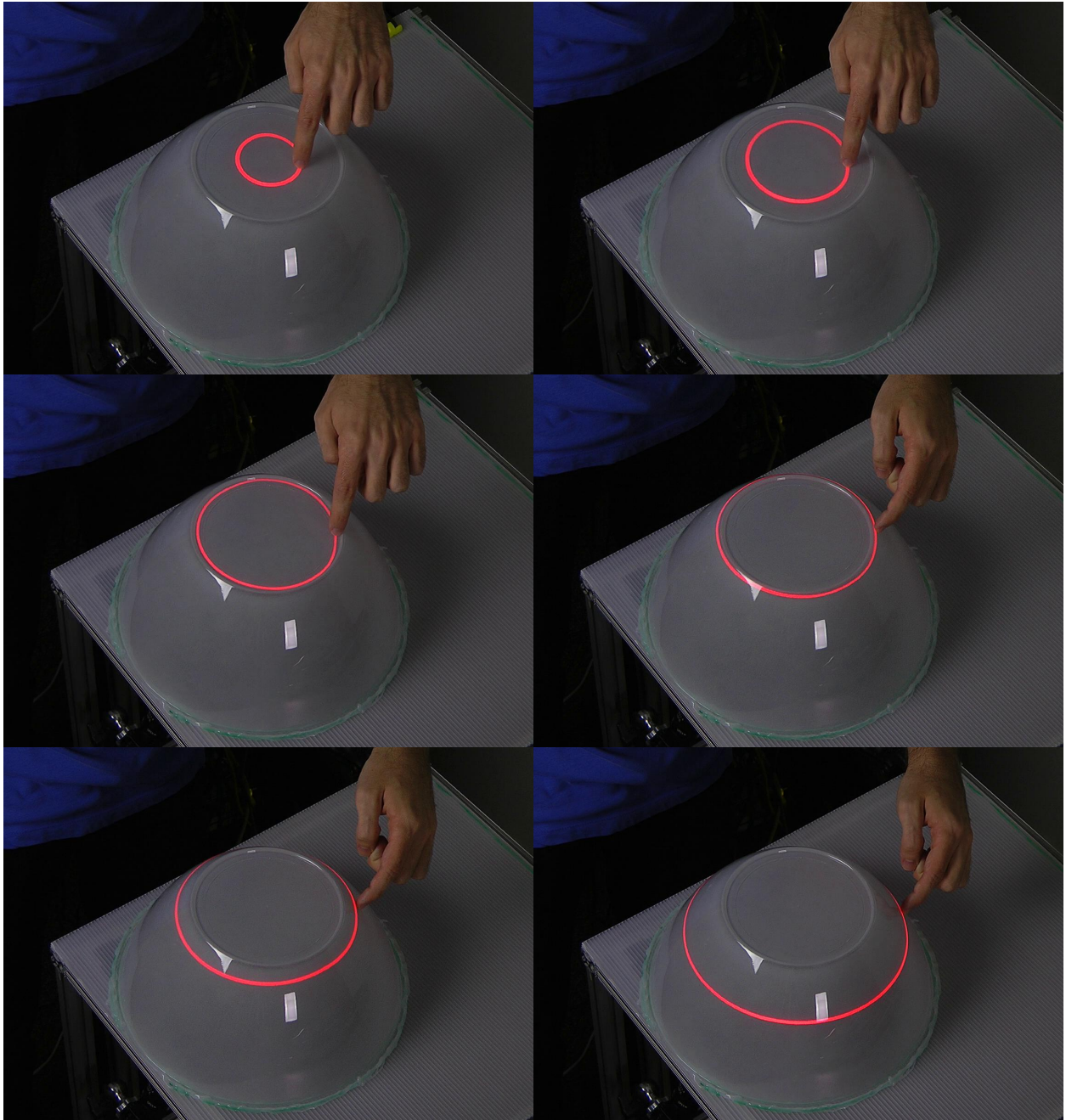


Figure 5.10: Touch-initiated contour lines on the bowl surface. As the touch location varies, the semantic content engine determines the appropriate contour line to project.



Figure 5.11: The touch-sensitive physical-virtual head surface.

Motivated by the benefits of physical-virtual patients in healthcare training, we developed more extensive semantic content for the head surface. This includes a sophisticated Unity environment containing a textured three-dimensional head model with corresponding animations and other behaviors; in one specific simulation mode, the virtual patient exhibits certain visual, audio, and touch-related symptoms of a stroke (Figure 5.12). Thus, the application of touch sensing to the head surface represents the entire proposed methodology of Chapter 3, including lookup table correspondences to graphics mesh coordinates and to touch-triggered animations and sound effects.

This section replicates and extends portions of two published peer-reviewed papers:

- “Touch sensing on non-parametric rear-projection surfaces: A physical-virtual head for hands-on healthcare training,” published in the proceedings of IEEE Virtual Reality 2015, by Jason Hochreiter, Salam Daher, Arjun Nagendran, Laura Gonzalez, and Greg Welch [71].
- “Optical touch sensing on nonparametric rear-projection surfaces for interactive physical-virtual experiences,” published in volume 25 of Presence (2016), by Jason Hochreiter, Salam Daher, Arjun Nagendran, Laura Gonzalez, and Greg Welch [72].

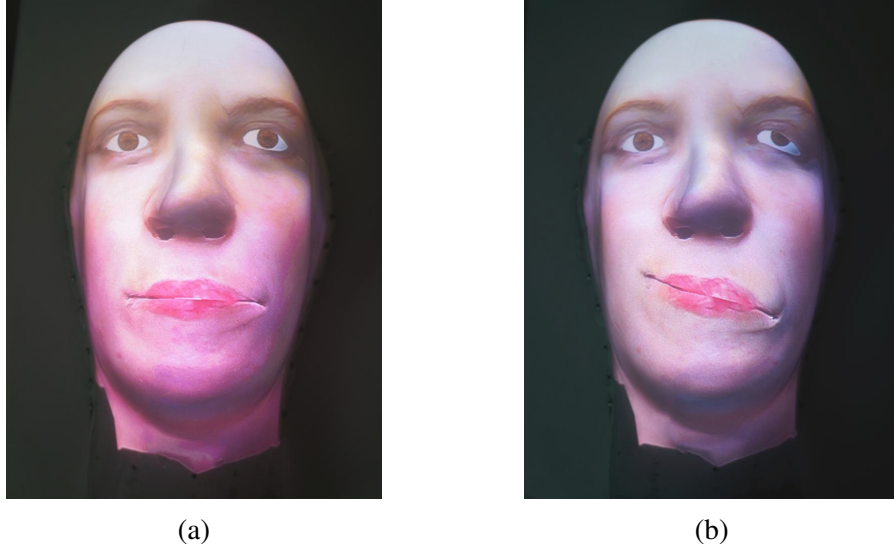
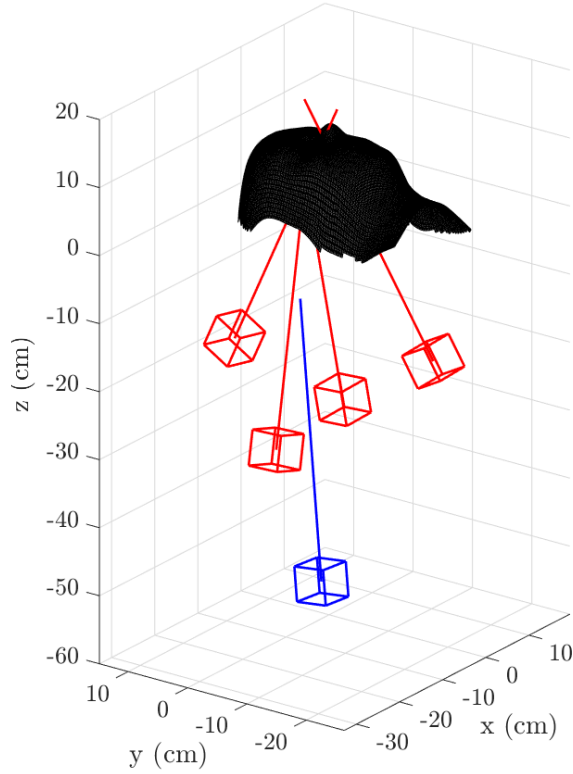


Figure 5.12: Simulated patients on the head surface. (a) Normal patient. (b) Patient showing signs of stroke, such as drooping eye and lips.

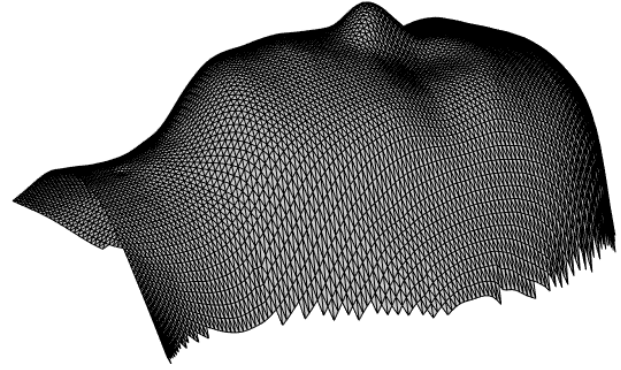
For this section, the word “we” refers to these colleagues. In particular, Salam Daher created the three-dimensional models, textures, and animations for the two virtual patients.

5.3.1 *Preprocessing Phase*

During the preprocessing phase, a total of 2319 features were projected onto the head surface and triangulated to form a 3D point cloud in the calibration space \mathbb{TCH}_3 . The initial calibration for the four cameras and the maximum-likelihood estimate for the projector’s calibration were then refined via bundle adjustment, resulting in average camera reprojection errors of 0.165, 0.227, 0.300, and 0.208 pixels, respectively (overall average 0.225 pixels), and a reprojection error of 0.141 pixels for the projector. These calibration results are visualized in Figure 5.13a. The final touch mesh S , shown in Figure 5.13b, has 7794 vertices and 15202 faces.



(a) Head calibration space TCH_3 .



(b) Head touch mesh S .

Figure 5.13: Head surface preprocessing phase. In the calibration space TCH_3 (a), the four cameras and one projector of Prototype Rig I are represented by the red and blue cubes, respectively.

5.3.2 Semantic Content Engine

The projector-space-based semantic content engines of the plane and bowl surface are still capable of supporting similar touch interfaces on the head surface (surface painting and object drag-and-drop interactions). However, to demonstrate capabilities useful in healthcare training scenarios, we developed a more sophisticated semantic content engine: a virtual Unity [141] environment containing a three-dimensional model of a patient with various behaviors appropriate to a health assessment scenario, including touch-triggered interactivity.

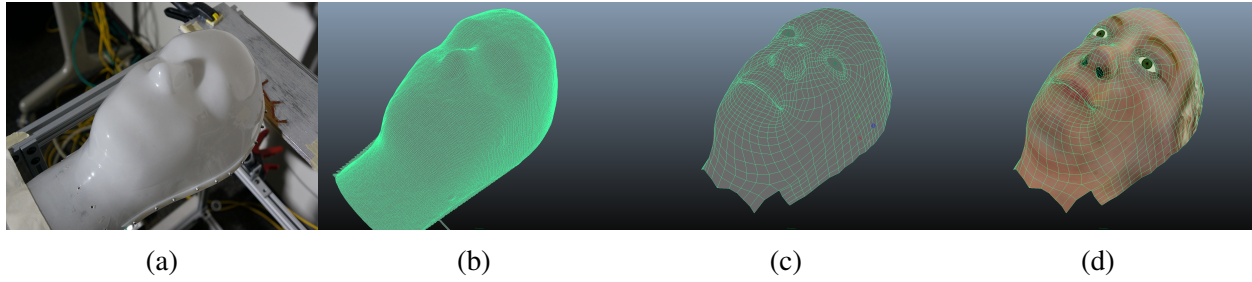


Figure 5.14: General process for modeling, texturing, and animating the physical-virtual head. (a) Physical head surface. (b) Dense touch surface mesh, created from the feature scan of the pre-processing phase. (c) Clean mesh with suitable topology for modeling, texturing, and animation. (d) Textured mesh.

5.3.2.1 3D Model

Figure 5.14 shows the general process for creating three-dimensional imagery for projection on the head surface (Figure 5.14a). The feature scan of the lookup table preprocessing phase produces a dense 3D surface mesh (Figure 5.14b), providing a reasonable starting point for a graphical model. However, the mesh is composed of a dense set of faces, arising from the grid structure of the feature scan, which is not amenable to texturing or animating. Thus, we simplify this mesh, reducing its density and retopologizing it to better accommodate the structure of a human head to reduce artifacts during animations (Figure 5.14c). Using a 3D head scan of one of our colleagues, we textured this simplified mesh (Figure 5.14d), adding geometry for the eyeballs and the inner mouth in Maya [5] for animations in these regions. In particular, the final graphical model supports the following animations:

- Upper and lower lip tug
- Upper and lower eyelid tug
- Eye open and close

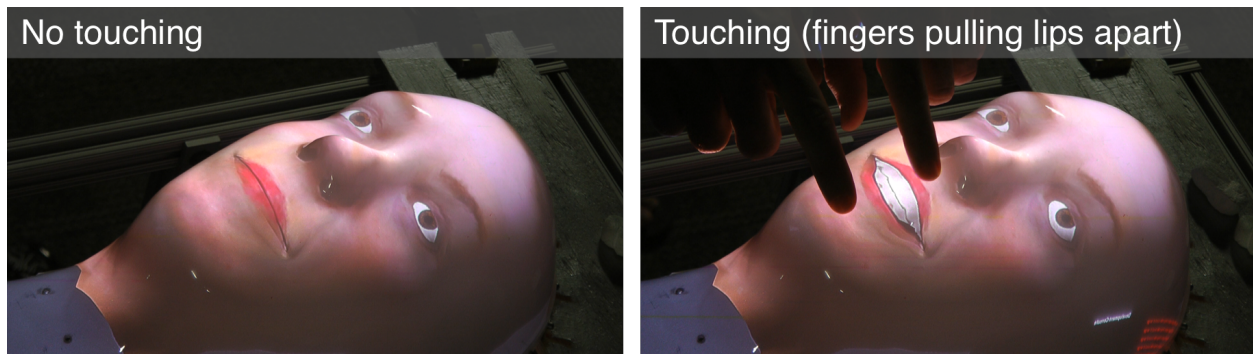


Figure 5.15: A touch-sensitive physical-virtual patient simulator. Left: no touch. Right: pulling the lips apart via touch to examine the patient’s teeth and gums.

- Pupil dilation
- Pupil movement
- Eyebrow raise and lower
- Tongue movement
- Mouth movements for lip syncing
- Emotions and facial expressions, such as happiness and surprise

While these animations were selected to provide a wide variety of typical patient behavior, our primary focus in this discussion is on the touch-triggered interactions: for instance, connecting the “upper lip tug” animation to the touch sensing system so that a healthcare practitioner can examine the patient’s upper gums by touch, as shown in Figure 5.15. As described in Section 3.3, each touch-based interaction has associated regions defined directly in the space of the 3D model so that touch input in specific locations can be linked to the appropriate behaviors in the lookup table.

The final animated model is then imported into a Unity scene with a virtual representation of the physical setup of Prototype Rig I. A virtual camera is created based on the physical projector cal-

ibration, oriented equivalently toward the graphical model, as described in the Appendix. Images rendered by the virtual camera are then sent directly to the rendering system to be projected onto the physical head surface.

5.3.2.2 *Patient Simulation: Healthy and Stroke*

The physical-virtual head is capable of portraying many of the visual and auditory signs and symptoms of a stroke, which we have separately evaluated in formative user studies (Chapter 7). We modified the original healthy patient model to create a special simulation mode representing a stroke patient, exhibiting signs and symptoms such as facial droop, slurred speech, facial asymmetry, and inability to sense touch in certain locations. The simulated patient can be dynamically switched between the two modes: *normal* and *neurological event*. We recorded a variety of verbal responses reflecting common answers and statements a patient undergoing a neurological assessment might provide. Each verbal response has two versions: one with a neutral voice and one with a slurred voice for the healthy and stroke patient modes, respectively. For simulated patient assessment, the verbal responses can be triggered manually, as in a Wizard of Oz paradigm [37]. Examples of manual verbal responses include:

- Patient history: e.g. her name, date of birth, health
- Current symptoms: e.g. “I have a headache,” “tingling,” and “I can’t feel anything there”
- Basic general responses: e.g. “yes,” “no,” and “I don’t remember”

Some of the semantic behavior is state-based: for instance, a touch in the normal mode might result in an audio response, but the same touch in the neurological event mode might have no response due to the patient’s inability to sense touch in certain places (i.e. *localized numbness*).

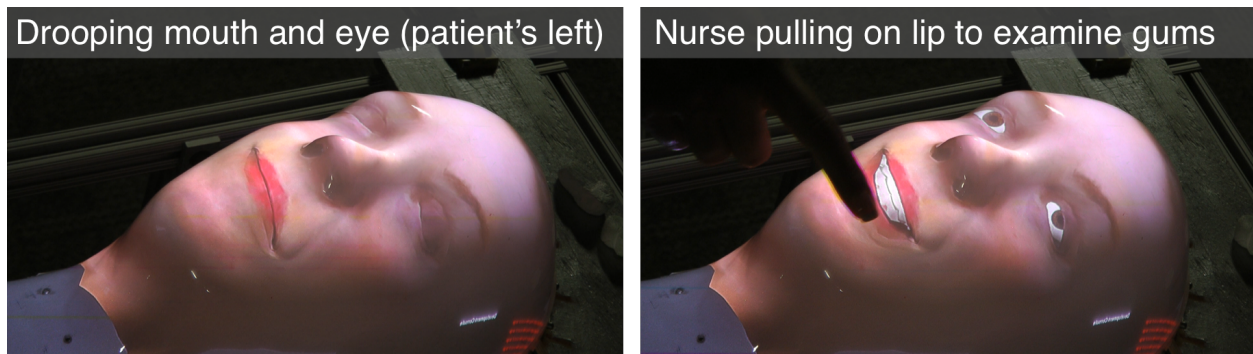


Figure 5.16: A touch-sensitive physical-virtual stroke patient simulator. The patient is exhibiting facial drooping and asymmetry on the left side; while the system can detect touch in the affected regions, the patient is unable to perceive touch there and therefore does not respond. Left: no touch. Right: pulling the lips apart via touch to examine the patient’s teeth, gums, and smile asymmetry.

Healthcare practitioners who suspect a patient might be experiencing a stroke might touch the patient’s head at certain locations and ask her to indicate whether she felt the touch. The semantic content engine supports a similar test. Touch input on predefined regions—such as the cheeks, mouth, and forehead—either automatically triggers an audio recording verbally identifying its location (e.g. “chin” and “left cheek”) or has no effect, depending on the neurological state of the patient. Touch-triggered animations, such as opening and closing the patient’s eyes and lips, work as expected in both modes, even when the patient’s left mouth and eye are drooping (Figure 5.16).

5.3.2.3 *Touch Interactions*

Many of the aforementioned visual and audio behaviors of the two patient models can be activated by touch. Figure 5.17 shows an example of capillary refill on the head. The user’s touch is converted to projector space by the lookup tables and transmitted to the semantic content engine, which then modifies the graphical model’s texture at the corresponding position. This is facilitated by a separate *blanched* texture, with a mask that encodes the relative weight of the blanched texture

to apply, as described in Section 4.1.2.3. During actual contact, the blanch texture is displayed with full opacity; once touch ends, the blanched opacity linearly decreases to zero over time, simulating the return of blood flow to this region. The blanch texture used to demonstrate capillary refill in Figure 5.17 is somewhat exaggerated to illustrate the effect.

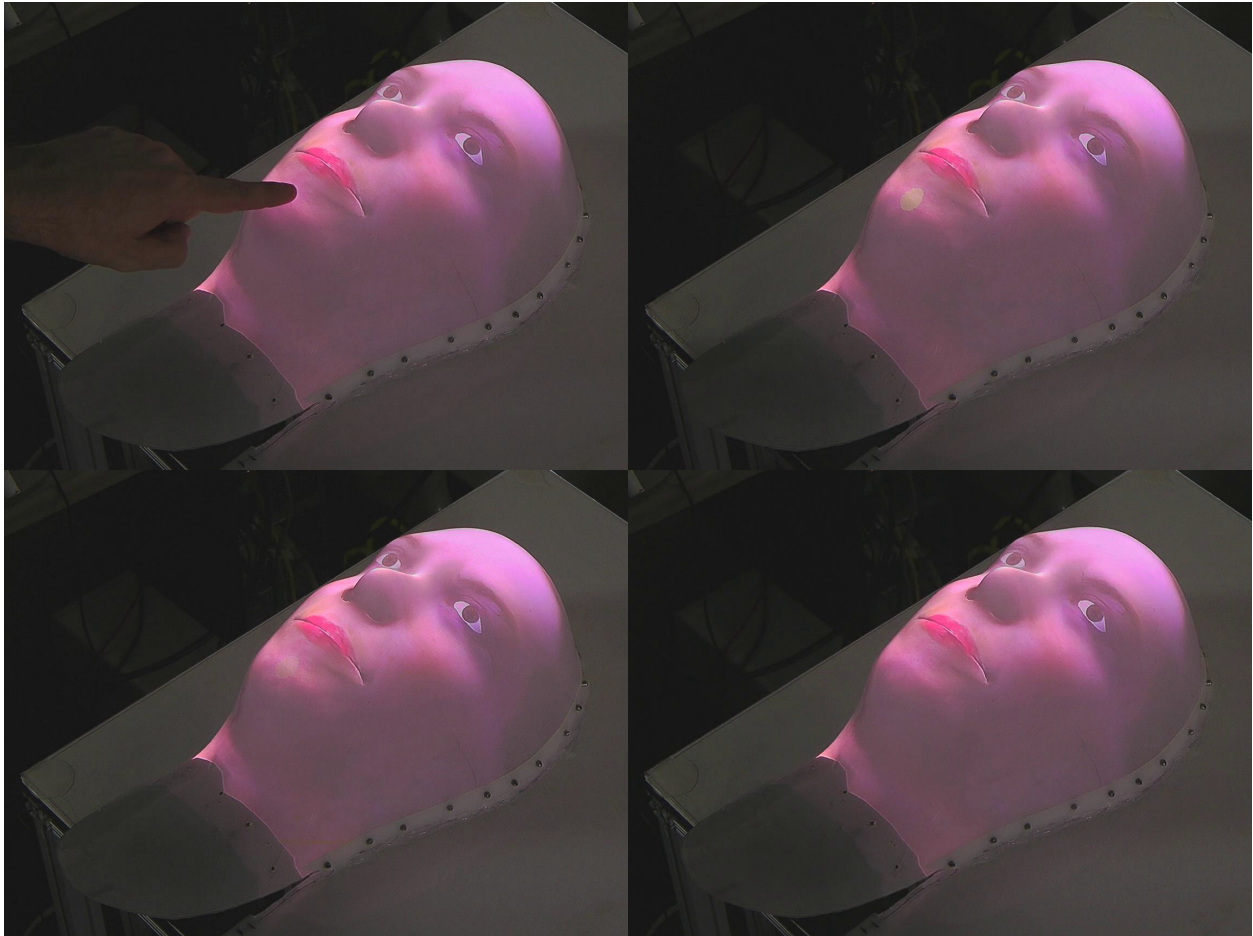


Figure 5.17: Touch-initiated capillary refill on the physical-virtual head surface, exaggerated for easier visibility. The texture changes at the point of contact, indicating blood flow has temporarily stopped. As time passes (from left to right and from top to bottom), the texture reverts to its original state, simulating the return of blood flow.

Touch-triggered blendshapes are demonstrated in Figure 5.18, including raising and lowering the patient's upper and lower lips and her upper and lower eyelids. The bottom row of the figure

shows examples of multiple blendshapes activating as a result of two detected touches. Moreover, the stroke patient is capable of exhibiting the same blendshapes, as shown in Figure 5.19.

During a neurological assessment, a healthcare practitioner may wish to test for sensory perception on various parts of the patient's face. To simulate this test, the patient is capable of verbally identifying the location of touch input (Figure 5.20). In the context of a stroke scenario, the patient may be unable to perceive touch, indicated by a lack of a verbal response to touch in the affected areas.

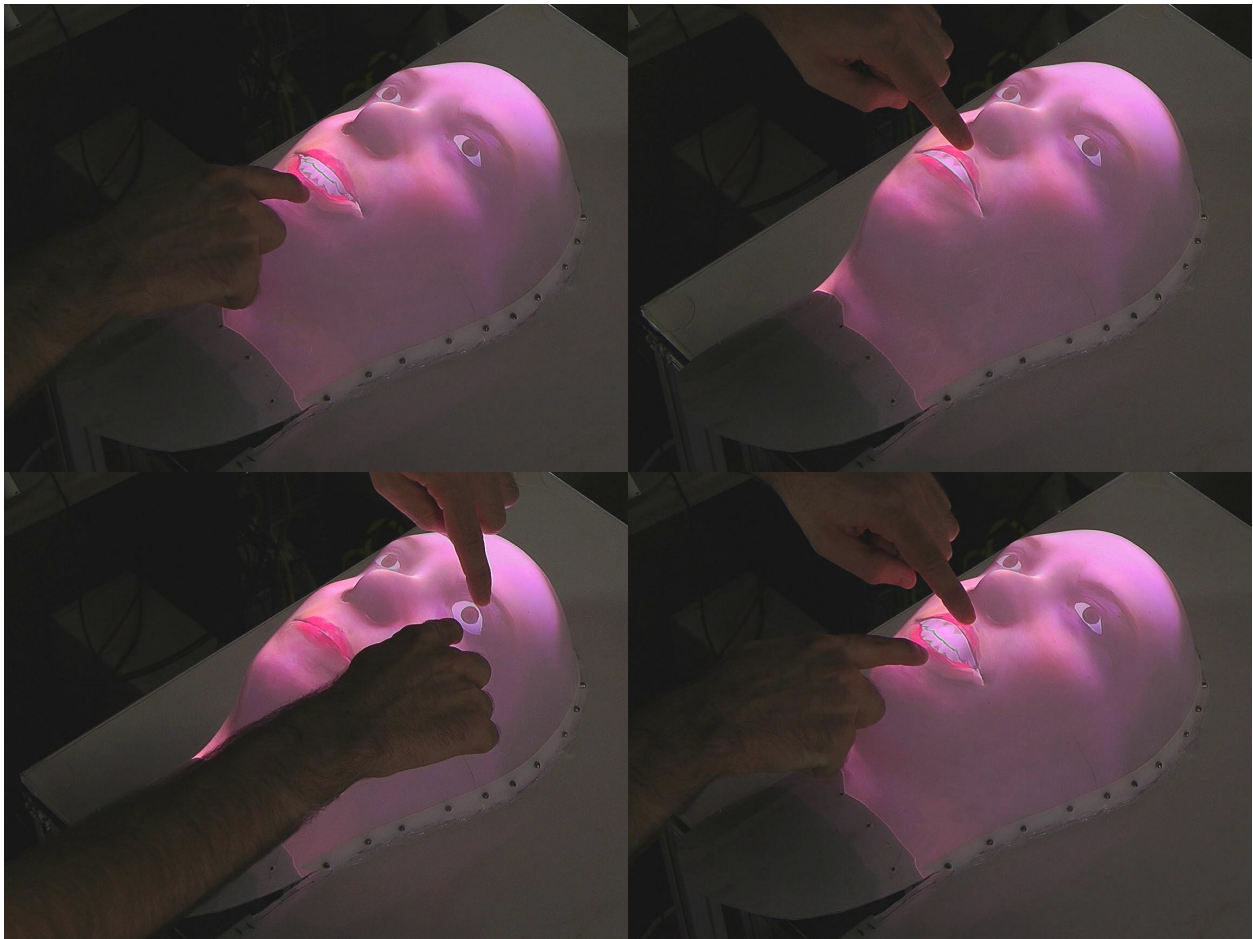


Figure 5.18: Touch-triggered blendshapes on the head surface. The bottom row shows examples of multiple simultaneous blendshapes updating due to multiple detected touch events.

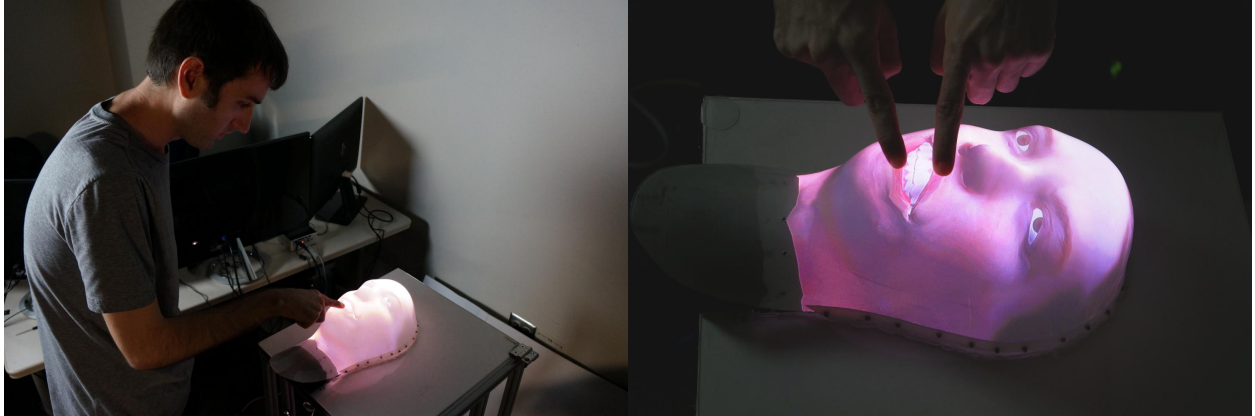


Figure 5.19: Touch-triggered blendshapes on the stroke patient function similarly to those on the normal patient.

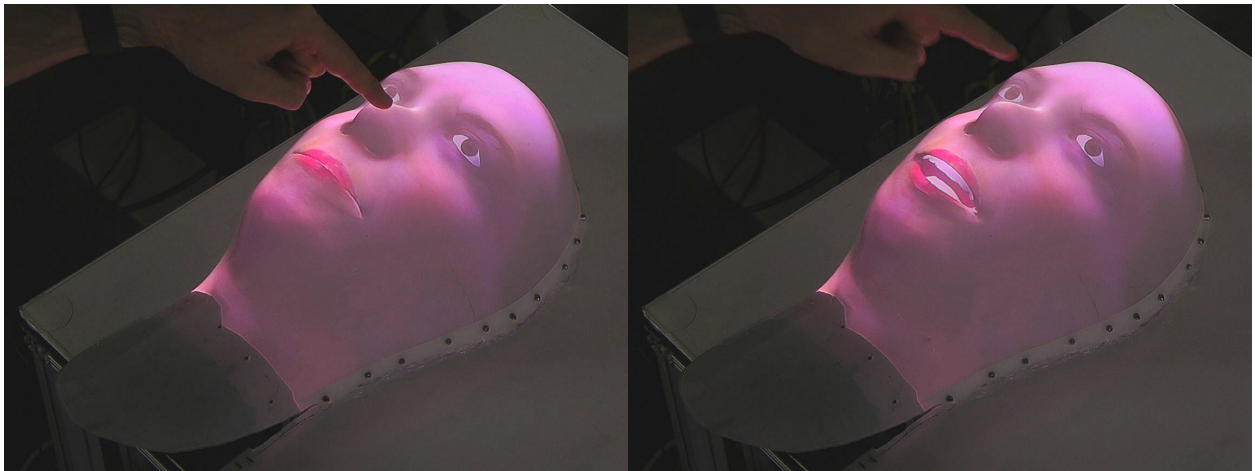


Figure 5.20: Touch-triggered audio on the head surface. The physical-virtual patient verbally identifies the location of a touch.

5.4 Child

Finally, we consider touch interactions on the most complicated of the four surfaces we investigated: a physical-virtual shell in the shape of a young child from the top of the legs to the head. Being a larger surface, it requires two projectors to display virtual imagery and so is supported

by Prototype Rig II. One projector primarily covers the child’s body, while the second covers his head. The physical-virtual child surface is shown in Figure 5.21. As with the physical-virtual patient head, we developed touch-triggered semantic content for the child rig suitable for patient simulation. The modeling, texturing, and animating of the child patient was again performed by Salam Daher; we have separately explored initial human-subject studies of this surface and content [35,36]. In this section, we focus on the extension of the automated touch sensing and response methodology of Chapter 3 to the child surface and virtual content.



Figure 5.21: The touch-sensitive physical-virtual child surface.

5.4.1 *Preprocessing Phase*

For the feature scan, a total of 3822 features were projected onto the child surface, with 2090 features from the body projector and 1732 from the head projector. These features were collectively triangulated to form an initial 3D point cloud representing the child surface. The final bundle adjustment (Figure 5.22a) produced reprojection errors of 0.264, 0.267, 0.233, 0.180, 0.204, 0.187, and 0.167 pixels for the seven cameras of Prototype Rig II, with an overall average camera reprojection error of 0.215 pixels. Moreover, the body and head projectors were calibrated with

reprojection errors of 0.200 and 0.082 pixels, respectively (average 0.141 pixels). The touch mesh S (Figure 5.22b) has 14570 vertices and 28644 faces.

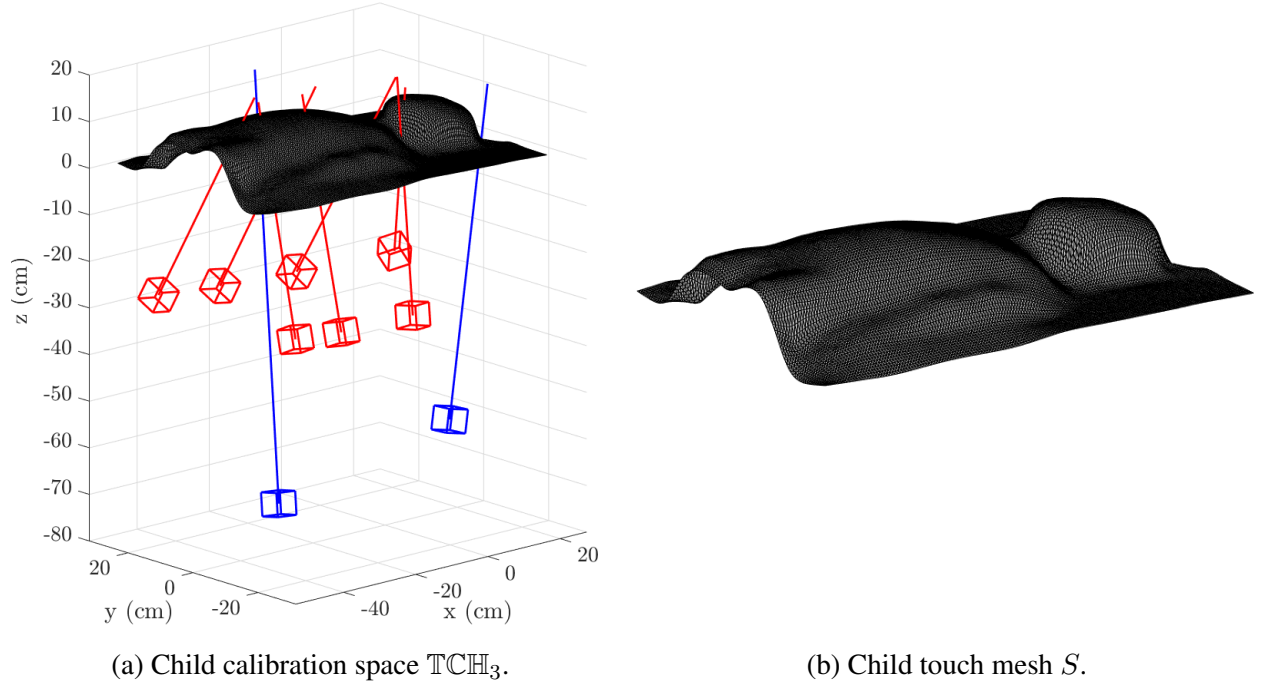


Figure 5.22: Child surface preprocessing phase. In the calibration space TCH_3 (a), the seven cameras and two projectors of Prototype Rig II are represented by the red and blue cubes, respectively.

5.4.2 Semantic Content Engine

As with the physical-virtual head surface, the child surface has a similar Unity-based semantic content engine consisting of an animated healthy child. The primary difference compared to the head's semantic content engine is in the use of two projectors, requiring the creation of two virtual cameras in Unity. With respect to touch-triggered semantic behavior, this has the greatest impact on the capillary refill effect, demonstrated in Figure 5.23. Each virtual camera in Unity must sep-

arately maintain its own blanch opacity mask, updated in response to the detected touch contours specific to the respective physical projector. Following rendering, each camera must reset the child model to use the original base texture, so that the blanch updates resulting from touch detections in the space of one projector do not affect the other projector.

Finally, Figure 5.24 shows touch-triggered blendshapes on the child surface, including multi-touch interactions. Functionally, these interactions operate identically to the blendshape activation method on the head surface, and a similar set of blendshapes was created corresponding to the child's lips and eyes.



Figure 5.23: Touch-initiated capillary refill on the physical-virtual child surface, shown exaggerated for easier visibility. This interaction simulates the temporary restriction and subsequent return of blood flow at the point of contact. Time progresses from left to right and from top to bottom.



Figure 5.24: Touch-triggered blendshapes on the child surface. The bottom row shows examples of multiple simultaneous blendshapes updating due to multiple detected touch events.

CHAPTER 6: SYSTEM EVALUATION

In this chapter, we present an overall evaluation of the proposed touch sensing methodology. First, we cover various system evaluation metrics as they pertain to both of the touch sensing algorithms of Chapter 3, including accuracy, modeling error, and performance metrics. Next, we present results for these metrics on the rear-projection surfaces described in Chapter 4. We conclude with a summary comparison of the results achieved by the two algorithms.

6.1 System Evaluation Metrics

To characterize the accuracy and performance of the overall touch sensing methodology, we considered the following system evaluation metrics:

- **Touch/hover classification accuracy:** How accurately does the system distinguish images of touch events from images of hovers?
- **Touch localization accuracy:** How close is a detected touch to the user’s intended touch position?
- **Modeling error:** How accurately does the system model the observed projector-camera correspondences?
- **Runtime performance:** How much time is required to detect touches from camera imagery?

Furthermore, each of these metrics can be compared for the two touch detection algorithms (projection space and plane sweep).

Assessing the true classification and localization accuracy of the system in terms of ground truth measurements is not straightforward. One possible avenue involves the use of a separate, validated means of locating the user’s finger in three-dimensional space with respect to a particular physical-virtual surface. However, this poses several challenges. Many 3D tracking systems, such as OptiTrack [114], rely on infrared light to track markers or other objects throughout an environment; as such, they are not appropriate for validating interactions supported by our prototype rigs, as this light would interfere with the IR light used for touch sensing. This approach would also require the temporal and spatial synchronization of data between the touch sensing and separate tracking systems, which may introduce additional sources of error specific to this synchronization. Moreover, the external tracking solution needs a mechanism to correctly determine if a given sample contains a touch or hover; if such a system existed, it would by definition solve the general challenge of touch sensing.

Instead, as a means of evaluating each of the above metrics, we developed an internal *guided target acquisition* process across two forms of input: **projected targets**¹ and **touch/hover targets**. Both involve the projection of circular features on the physical-virtual surface, chosen from prescribed points in the imagery of the projectors. Captured camera imagery of the projected targets can be processed through the detection algorithms: like touches, projected targets are located *on* the surface, so the two detection algorithms should classify them strongly as touches (i.e. with high confidence scores). Furthermore, the data encoded in the lookup table can be analyzed to determine how consistently the localizations of these projected targets match their observed positions during the preprocessing phase, which reflects how accurately the lookup table models these observations. Separately, an expert user carefully captures touches and hovers at specific projected feature locations to create a set of touch/hover targets, which are then classified and localized using the detection algorithms.

¹Throughout this section, we use the terms *projected target* and *projector target* interchangeably.

In terms of touch/hover classification, the two detection algorithms differ in their reported confidence scores. For the projection space algorithm, the multi-camera agreement score ranges from 0 to 1, with higher scores indicating the algorithm is more confident that the sample contains a touch. The plane sweep algorithm, which considers a series of parallel planes starting at the surface and extending along a normal vector, reports the distance along that normal vector corresponding to the plane having the minimum union area; lower distances indicate the minimum union area plane is closer to the surface, thus reflecting higher touch confidence. As touch events are located *on* the surface, we expect high confidence scores from both detection algorithms: high multi-camera agreement score and low minimum union plane distance for the projection space and plane sweep algorithms, respectively. Likewise, we expect low confidence scores (low multi-camera agreement score and high minimum union plane distance) for hovers, which occur *off* the surface.

Each touch sensing algorithm localizes detected touches on the physical surface, producing a 3D point on the touch surface S corresponding to the user’s touch location. While a validated ground truth coordinate representing the user’s touch is not available, we can compare the localization to the data encoded in the lookup table—specifically, the correspondence between the centroid pixel of the *projected target the user touched* and its estimated position on S . We define **target-detection distance** to be the distance between

- the 3D point on the touch surface S stored as the lookup table correspondence for a projector pixel (i.e. the *target*) and
- the 3D point retrieved when processing live imagery (of a projector or touch target) containing that projector pixel (i.e. the *detection*).

To evaluate various aspects of the system, we consider both projector-target-detection and touch-target-detection distances.

Lower projector-target-detection distances reflect higher consistency in the lookup table, indicating that it accurately models the observed projector-camera correspondences of the feature scan. In theory, this consistency additionally extends to the accuracy of touch input localizations, analyzed via the touch target data. It is important to note that touch-target-detection distances are inherently dependent on the ability of the expert user to touch the targets accurately. As such, these distances are influenced by a variety of possible user-centric error sources, including the user’s visual interpretation of the positions of the targets across the surface and the user’s physical dexterity in both orienting and positioning his or her finger to touch the intended positions. However, a set of touch-target-detection distances provides a reasonable representation of how close user touches are to intended targets *in practice*, which is a valuable overall system metric that reflects the level of accuracy achievable in real touch applications.

Touch target data is carefully collected from an expert user over two sessions: one that focuses on only touch events and one that focuses only on hovers. In both cases, the expert user is presented with a set of visual targets projected onto the surface, where each target contains a small crosshair to assist in visually locating the center. For touch data, the expert user is presented with a set of targets projected on the surface, and he is instructed to touch each one sequentially as carefully and precisely as possible. Once his finger is in place, he presses a key on a provided keyboard to capture imagery from the cameras. Thus, all captures are therefore known to contain touch events. For hover data, an expert user is again presented with these visual targets, but this time he carefully places his finger above the surface without touching it, again using the keyboard to initiate camera capture. In the event that he inadvertently touches the surface during data collection, the current target can be reset so that hover data can be recaptured. Given these two datasets, we can therefore report touch/hover classification accuracy results simply by determining how the touch sensing algorithms classify each of the touch samples and each of the hover samples.

To summarize, we propose the following procedures for evaluating the aforementioned metrics:

- **Touch/hover classification accuracy:** We process the datasets of touch and hover events using both the projection space and plane sweep algorithms and analyze the reported confidence scores for each.
- **Touch localization accuracy:** We localize touch events from the touch dataset in 3D space with both algorithms and compare these positions to the lookup table correspondences.
- **Modeling error:** We process projected targets with both algorithms and analyze the reported confidence scores and target-detection distances.
- **Runtime performance:** During localization and classification of projector and touch targets, we calculate and record the amount of required processing time.

Below, we present these metric evaluations for three of the four touch-sensitive surfaces presented in Chapter 4: the bowl, the head, and the child; as the plane poses a degenerate configuration that prevents successful bundle adjustment, we omit it from the present discussion (see Section 4.1.1.1). These analyses are performed using both the projection space and plane sweep touch detection algorithms. For the projected target results, we process the data collected during the respective feature scan for each surface, which reflects how accurately the created lookup tables model the scan observations. The visual targets presented to the expert user in the touch and hover target data collection phases are drawn from smaller grids in projector space. We present a graph of the projected- and touch-target-detection distances and confidence scores over each set of samples, indexed in the order they were captured. Using the projector coordinates of the targets, we display these same results graphically in the space of the projected imagery. Finally, we texture the touch surface meshes for each surface with these projector space result images, allowing for direct interpretation in a three-dimensional context.

For touch/hover classification, we provide classification accuracy independently for touches and hovers and for the overall combined dataset based on a range of confidence score thresholds. The two touch detection algorithms differ in this thresholding process. In the projection space algorithm, given a multi-camera agreement score threshold s , samples with scores greater than or equal to s are classified as touches, while those with scores less than s are classified as hovers. The plane sweep algorithm classifies samples whose minimum union area plane is at a distance of some threshold t or closer to the touch mesh as touches; samples with minimum union area planes at distances greater than t are classified as hovers. *Trivial hovers* are those that are removed before even being processed by the two algorithms, such as from hovering fingers that are sufficiently far from the surface that they reflect no detectable infrared light to the cameras. In the touch/hover classification results, we exclude these trivial cases, as our primary interest is in distinguishing actual touches from close hovers, which often appear nearly identical in the infrared camera imagery.

Additionally, we report the average runtime of each algorithm. In our physical prototypes (Chapter 4), our cameras capture imagery at 30 frames per second—approximately 33.33 milliseconds; thus, we can consider these touch sensing algorithms as achieving *real-time performance* if they can classify a set of camera images within 33.33 milliseconds, at which point the next set of images is available. Since samples with no usable image data—for instance, when a projected target is outside the bounds of the surface—are discarded and not processed for touches, they do not incur execution time for various parts of the overall pipeline, and so we only report average runtime across the set of usable samples. To more directly compare the execution times of the two touch sensing algorithms, we separately account for the time needed to capture and process camera images, which is constant for both, and the time each algorithm requires *specifically for touch detection* given processed images. Furthermore, since projected target data tends to be “cleaner”—with light only provided by the specific target projected—segmenting the associated camera imagery tends to require less time than for touch data. Each algorithm performs its entire pipeline, including deter-

mining semantic region correspondences. As such, these results provide a realistic representation of runtime requirements in practical use cases featuring interactions with touch-triggered semantic responses.

We conclude each section with a summarized results comparison between the two detection algorithms. Finally, Section 6.5 includes an overall algorithm comparison across the three surfaces.

6.2 Bowl

Projector target and touch target results for the bowl surface are described in Section 6.2.1 and Section 6.2.2, respectively.

6.2.1 Projector Targets

The feature scan for the bowl surface consisted of approximately 2200 projected targets.

6.2.1.1 Projection Space

The projection space touch detection algorithm successfully processed 2079 of the projected targets at an average overall speed of 8.85 milliseconds, with detection alone requiring 2.79 milliseconds.

Projected-target-detection distance: Figure 6.1 shows the overall projected-target-detection distance results from the projection space algorithm. A graph of these distances across the set of feature scan samples is shown in Figure 6.1a; the average target-detection distance is 0.0226 cm. Figure 6.1b provides a graphical visualization of these results in the space of the projector on Prototype Rig I, and Figure 6.1c presents them in the 3D context of the bowl touch mesh S . Here, the

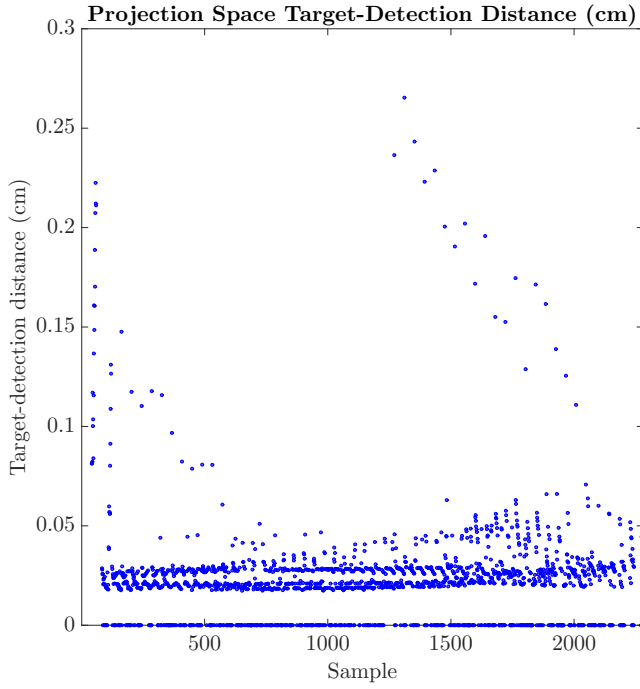
color blue corresponds to coordinates in projector space that are outside the bounds of the bowl surface. As shown in these visualizations, the largest distances are located on the bounds of the projected imagery.

Projected target confidence score: Overall confidence score results are shown in Figure 6.2, taken from the multi-camera agreement scores computed by the projection space algorithm. Scores across the projected targets are shown in Figure 6.2a: the mean score is 0.688, indicating high confidence of a touch. The confidence scores are shown graphically in projector space in Figure 6.2b and textured on the 3D bowl touch mesh S in Figure 6.2c. The targets with relatively high localization errors on the projection image perimeter have comparably low confidence scores. Moreover, detected targets on the slightly raised circular portion on which the bowl rests (the “lip”) have similarly lower confidence scores.

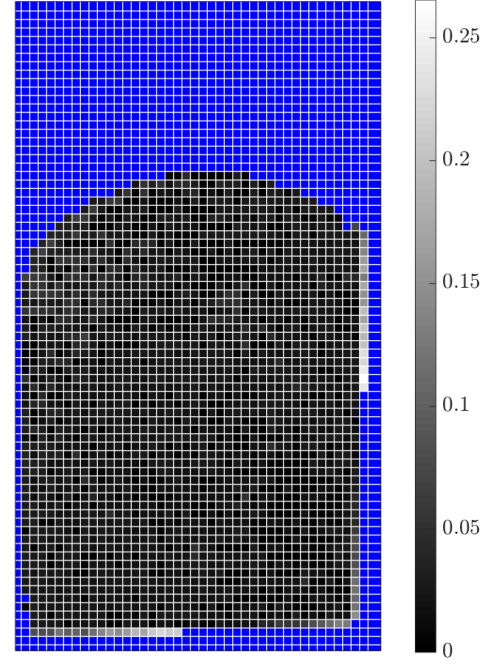
6.2.1.2 *Plane Sweep*

Out of the set of projected targets, the plane sweep algorithm successfully processed 2162. The overall average processing time was 6.37 milliseconds; the actual detection routines executed in 0.50 milliseconds.

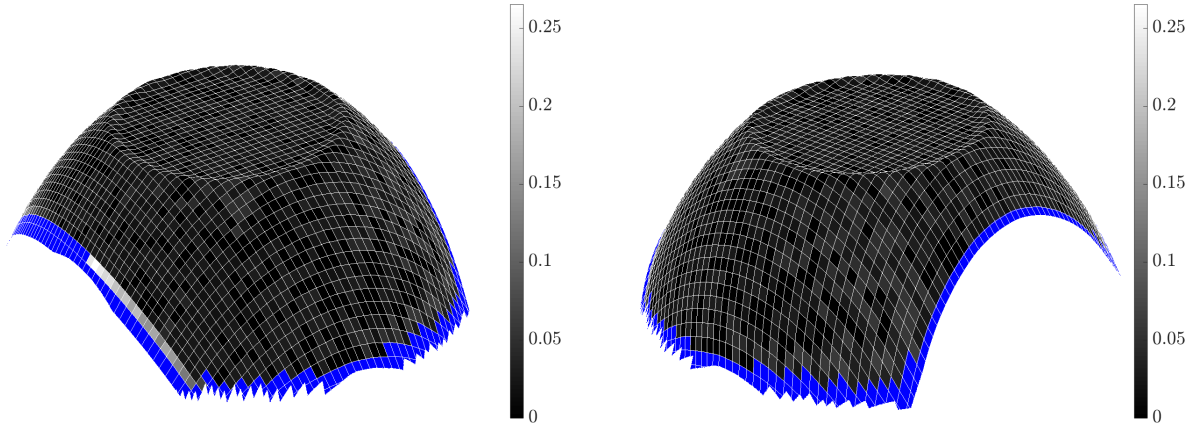
Projected-target-detection distance: Figure 6.3 presents the overall projected-target-detection distance results produced by the plane sweep algorithm. A graph of the distances across the set of projected targets is shown in Figure 6.3a; the mean distance is 0.0088 cm. Figure 6.3b shows these results directly in projector space, while Figure 6.3c visualizes them on the bowl touch mesh S . The projected-target-detection distances are extremely consistent and small across the surface, with the highest errors corresponding to the lip of the bowl. However, even at these locations, the distances between the projected targets and the detections are less than 1 mm.



(a) Graph over feature scan samples.

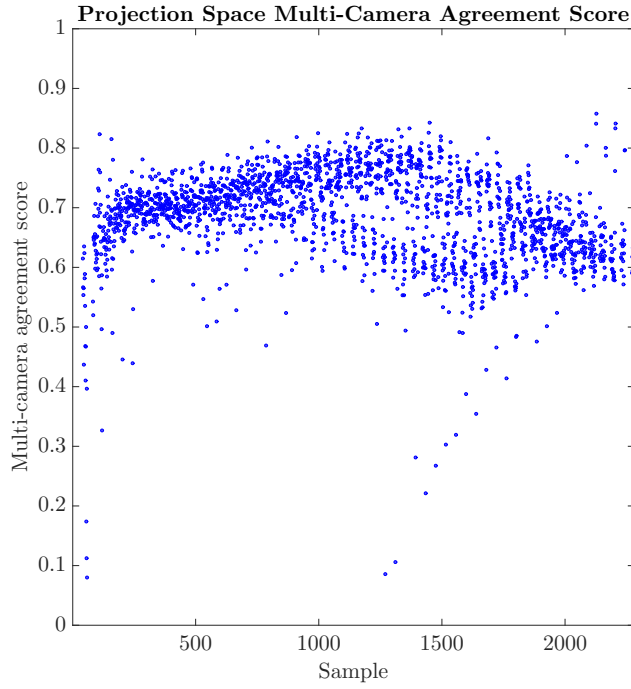


(b) Results shown in projection space.

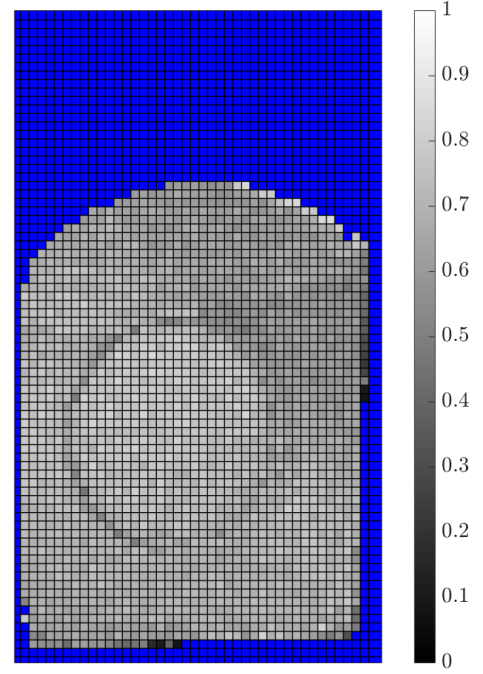


(c) Results shown in two views in 3D space on the bowl touch mesh S .

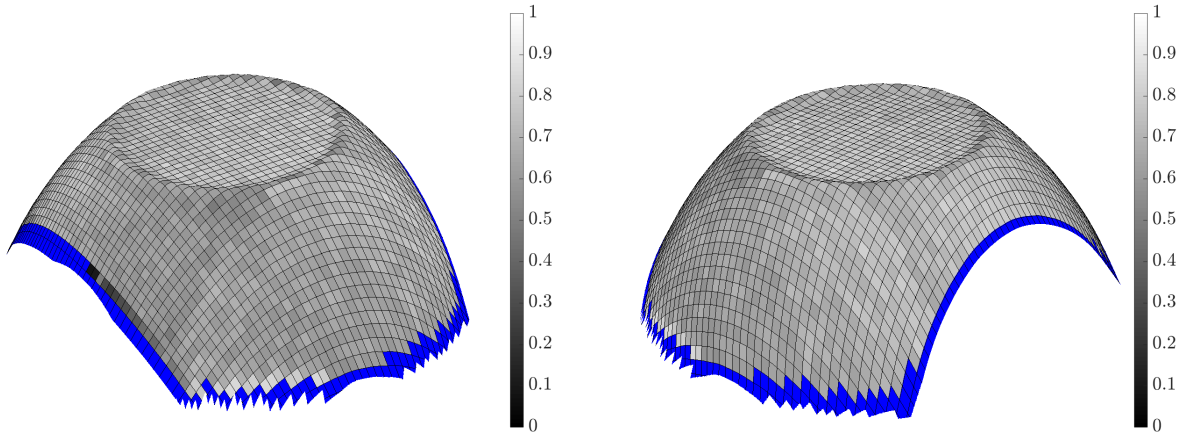
Figure 6.1: **Projection space algorithm** results for the **bowl surface**: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences.



(a) Graph over feature scan samples.

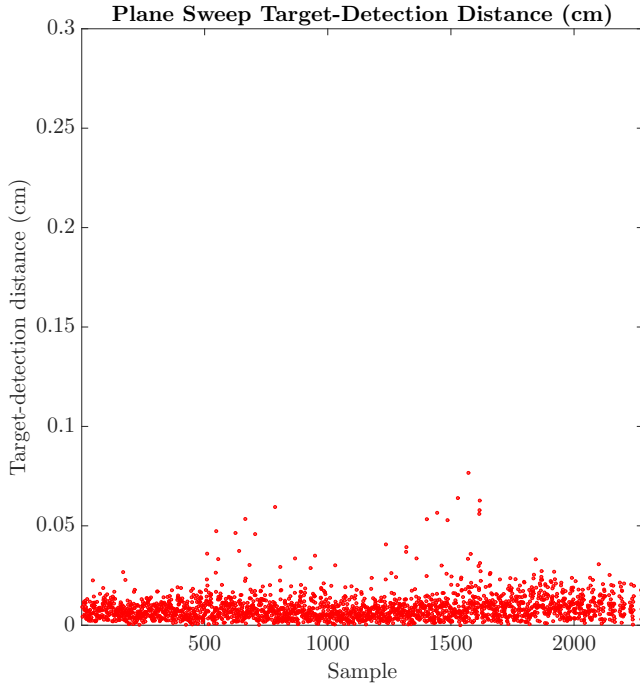


(b) Results shown in projection space.

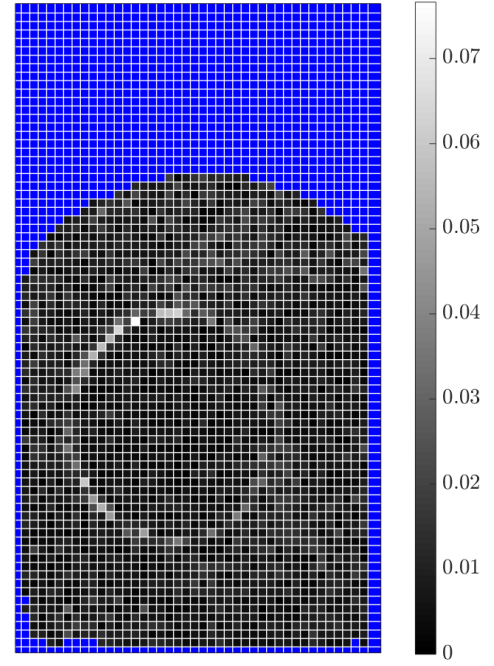


(c) Results shown in two views in 3D space on the bowl touch mesh S .

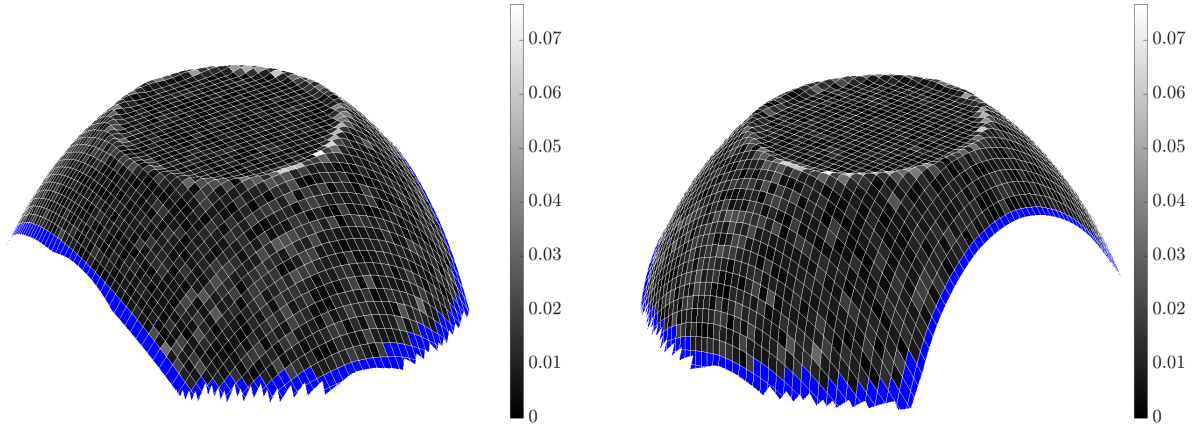
Figure 6.2: **Projection space algorithm** results for the **bowl surface**: multi-camera agreement score for **projected targets**.



(a) Graph over feature scan samples.

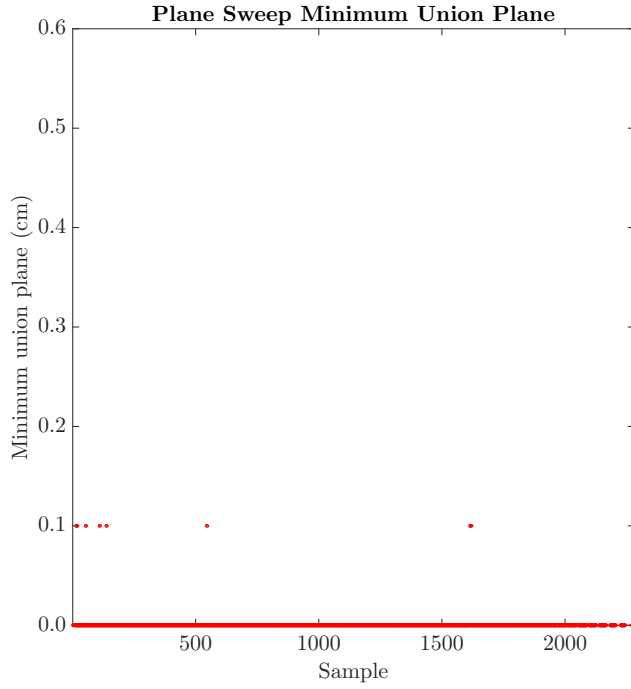


(b) Results shown in projection space.

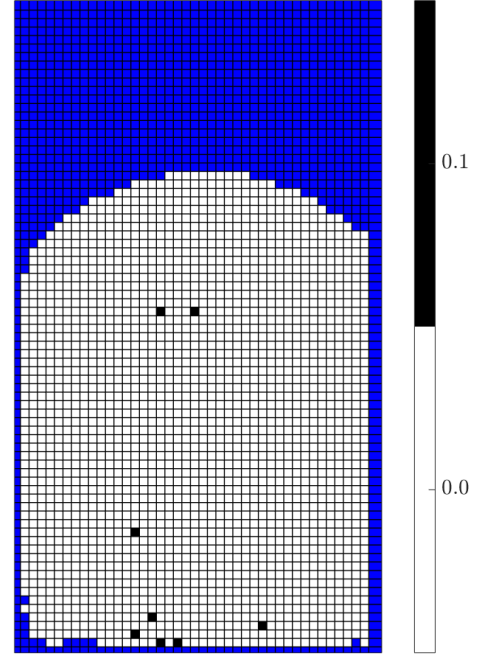


(c) Results shown in two views in 3D space on the bowl touch mesh S .

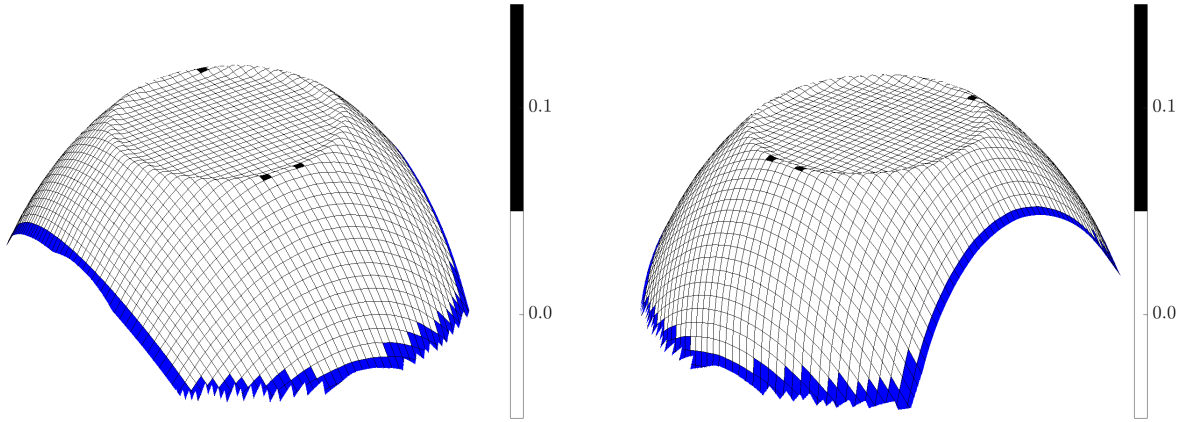
Figure 6.3: **Plane sweep algorithm** results for the **bowl surface**: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences.



(a) Graph over feature scan samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the bowl touch mesh S .

Figure 6.4: **Plane sweep algorithm** results for the **bowl surface**: minimum union plane (centimeters) for **projected targets**.

Projected target confidence score: The distances of the planes with the minimum union areas determined by the plane sweep algorithm are summarized in Figure 6.4. Figure 6.4a shows a graph across the set of feature scan targets, while Figure 6.4b and Figure 6.4c present them in two-dimensional projector space and the three-dimensional space of the bowl surface touch mesh S , respectively. Nearly every projected target was assigned to the initial plane located at the surface, indicating extremely high confidence of a touch. A small number of targets were instead assigned to the next plane translated 0.1 cm from the surface, also suggesting high touch confidence; in general, these correspond to regions on the lip of the bowl and to areas on the back of the bowl (i.e. the opposite of the side where the user stands for touch interactions).

6.2.1.3 Summary

Table 6.1 shows a summary of the projected target results for the bowl surface. For a direct comparison, Figure 6.5 plots the projected-target-detection distances across the set of feature scan targets.

In general, the plane sweep algorithm outperformed the projection space algorithm by all the considered metrics:

- The plane sweep algorithm successfully processed more of the feature scan targets.
- On average, the 3D distance between the projected targets and the localized detections on the bowl touch mesh S was approximately 40% that produced by the projection space algorithm.
- Overall, the plane sweep confidence scores were more consistent than the projection space scores, with the minimum union plane chosen to be the one closest to the surface for approximately 99.6% of projected targets. However, the projection space algorithm assigned relatively low confidence scores to a few dozen of the targets.

- Finally, the plane sweep algorithm processed samples significantly faster than the projection space algorithm. When considering total execution time, which includes the segmentation of camera imagery into contours, the plane sweep and projection space algorithms executed in 6.37 and 8.85 milliseconds, respectively. However, segmenting the imagery is a constant time required by both of the algorithms. The time required for detection alone reveals a much greater difference: 0.50 milliseconds for the plane sweep algorithm compared to 2.79 milliseconds for the projection space algorithm.

Table 6.1: **Projector target** results summary for the **bowl surface**.

Algorithm	Successful	Localization	Confidence Score	Runtime
Projection space	2079	0.0226 cm	0.688	Detect: 2.79 ms Total: 8.85 ms
Plane sweep	2162	0.0088 cm	0 cm: 99.63% 0.1 cm: 0.37%	Detect: 0.50 ms Total: 6.37 ms

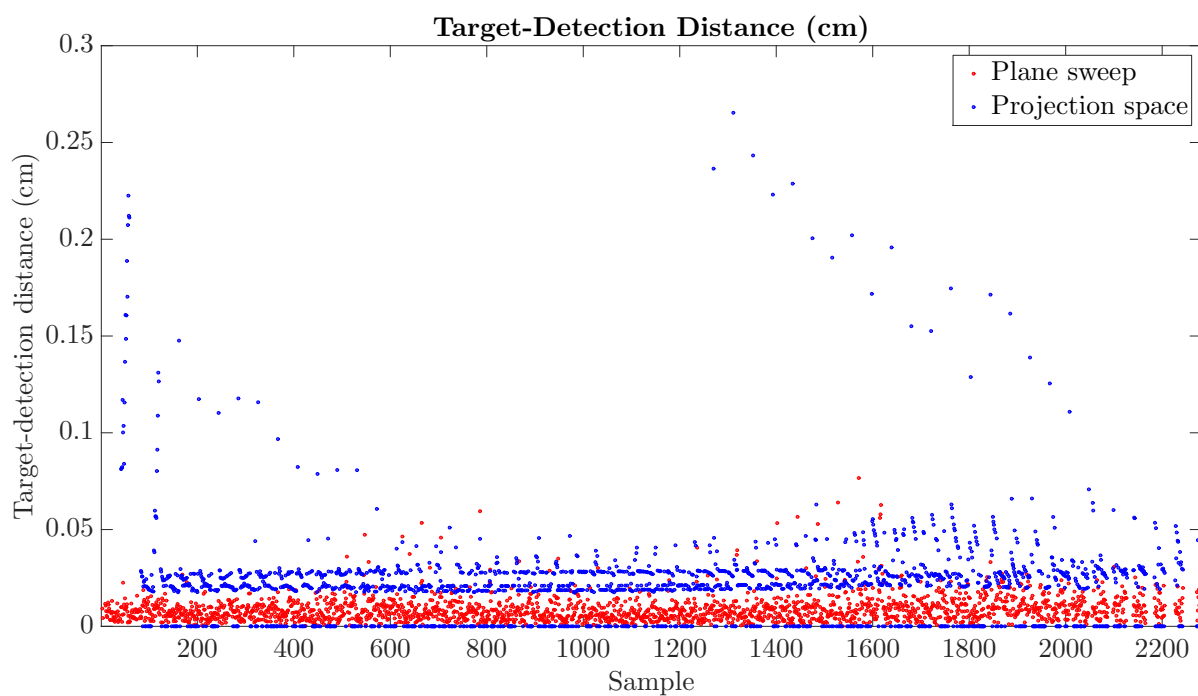


Figure 6.5: Comparison of **projection space** (blue) and **plane sweep** (red) algorithm results: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences for the **bowl surface**, shown across the samples of the feature scan. On average, the plane sweep distances are significantly lower.

6.2.2 Touch Targets

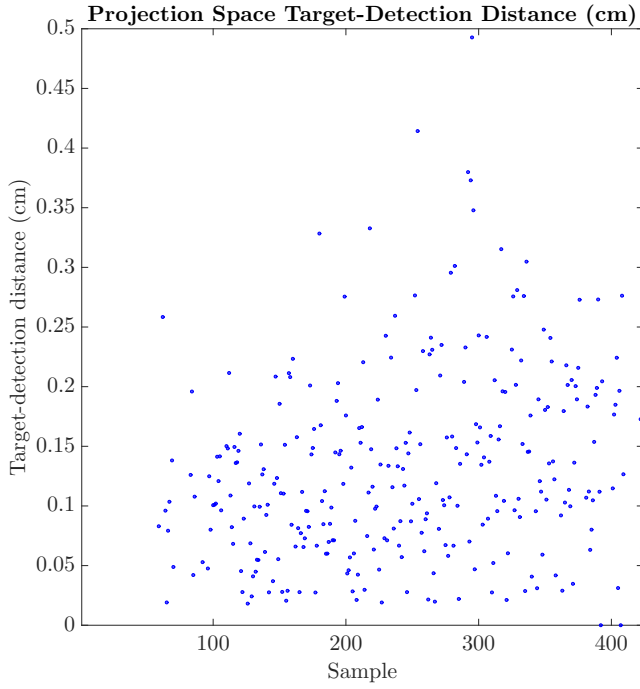
Approximately 300 touch samples arising from a grid of visual targets form the touch target dataset for the bowl surface.

6.2.2.1 Projection Space

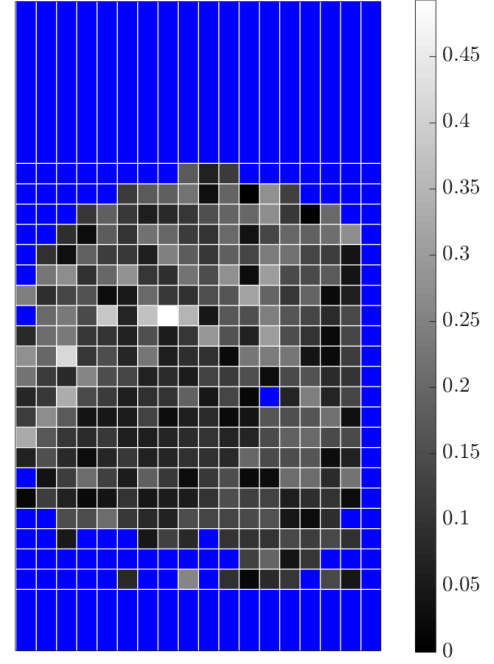
The projection space algorithm processed 293 of these touch targets, assigning to each a classification (either a touch or a hover) and a localized position in three-dimensional space. On average, the algorithm processed the targets in 9.90 milliseconds, with the detection routines operating in 2.83 milliseconds.

Touch-target-detection distance: A summary of the touch-target-distance results from the projection space algorithm is shown in Figure 6.6. Across the set of samples, these distances are scattered about a range from 0 to 0.5 cm, with the majority under 0.3 cm and an average of 0.1311 cm (Figure 6.6a). There is not much structure to the errors evident in the projection space image (Figure 6.6b) or in the 3D context of the touch mesh (Figure 6.6c); some of the larger errors are near the “lip” of the bowl, where the curved and flat portions meet.

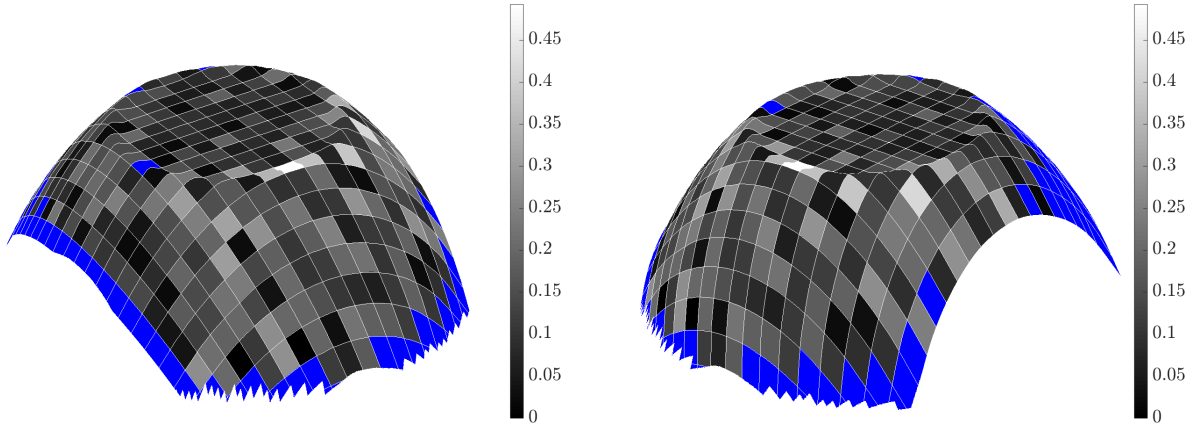
Touch target confidence score: Figure 6.7 shows the multi-camera agreement scores for the touch targets computed by the projection space algorithm. Compared to the target-detection distances, these scores are a little more consistent, generally clustered around values between 0.5 and 0.7 with an average of 0.614 (Figure 6.7a). Other than a few low confidence scores on the “lip,” the scores appear relatively uniform across the bowl surface (shown in projector space in Figure 6.7b and textured on the bowl touch mesh in Figure 6.7c).



(a) Graph over touch samples.

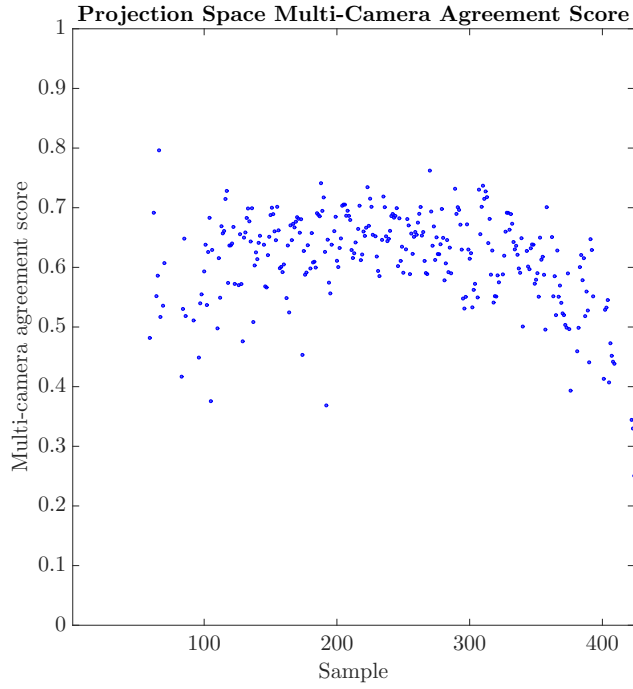


(b) Results shown in projection space.

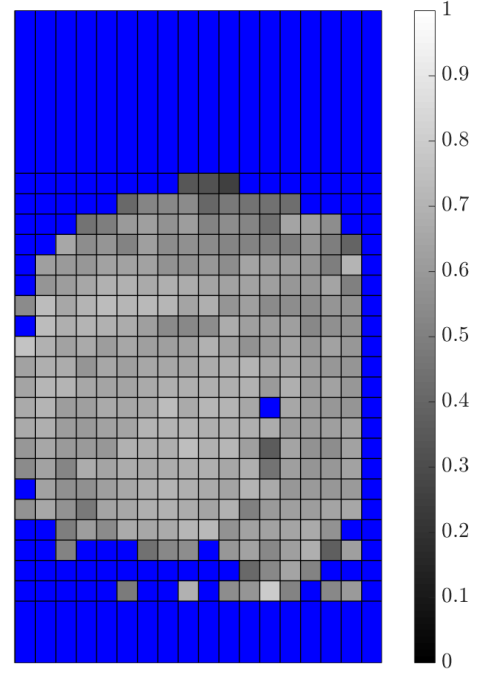


(c) Results shown in two views in 3D space on the bowl touch mesh S .

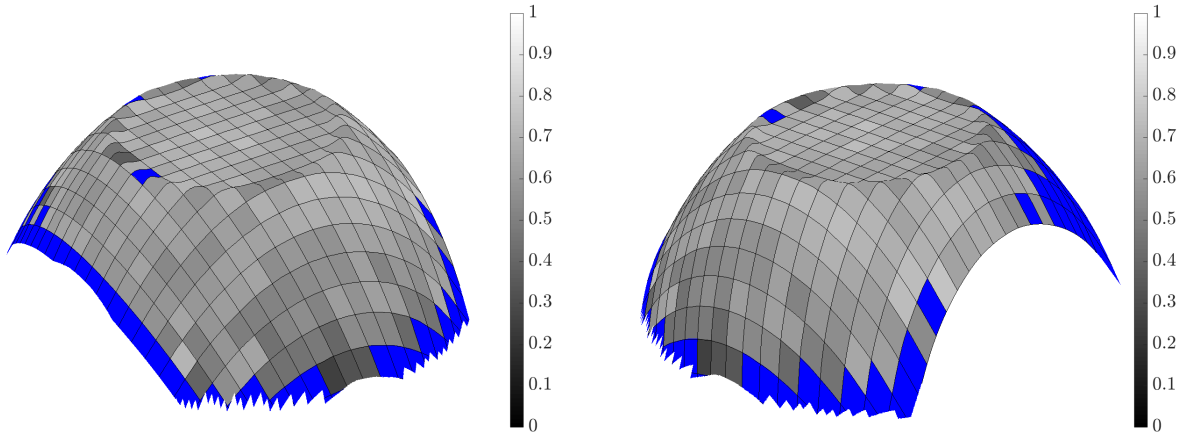
Figure 6.6: **Projection space algorithm** results for the **bowl surface**: distance (centimeters) between localized **touch targets** and the 3D lookup table correspondences.



(a) Graph over touch samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the bowl touch mesh S .

Figure 6.7: **Projection space algorithm** results for the **bowl surface**: multi-camera agreement score for **touch targets**.

Touch/hover classification: Figure 6.8 shows the touch/hover classification results for the projection space algorithm. Figure 6.8a plots the multi-camera agreement scores of the touch and hover samples; nearly all of the confidence scores for the hovers are below 0.3, and nearly all of the touches have confidence scores above this threshold. Total classification accuracy, along with separate touch and hover classification accuracies, are presented in Figure 6.8b across various multi-camera agreement score thresholds. The highest overall accuracy of 99.60% correct classifications (99.66% touches and 99.50% hovers correct) is achieved with a score threshold of 0.3.

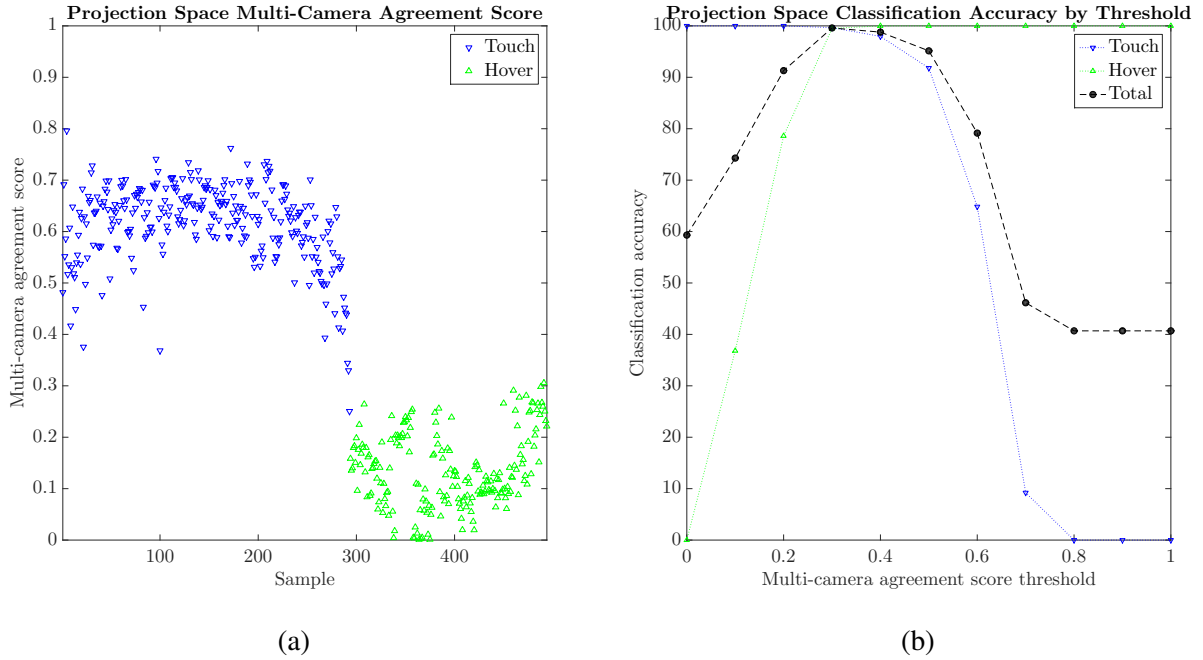


Figure 6.8: **Projection space algorithm** results for the **bowl surface: touch/hover classification**. (a) Confidence scores across touch and hover samples. (b) Overall touch/hover classification accuracy by confidence score threshold.

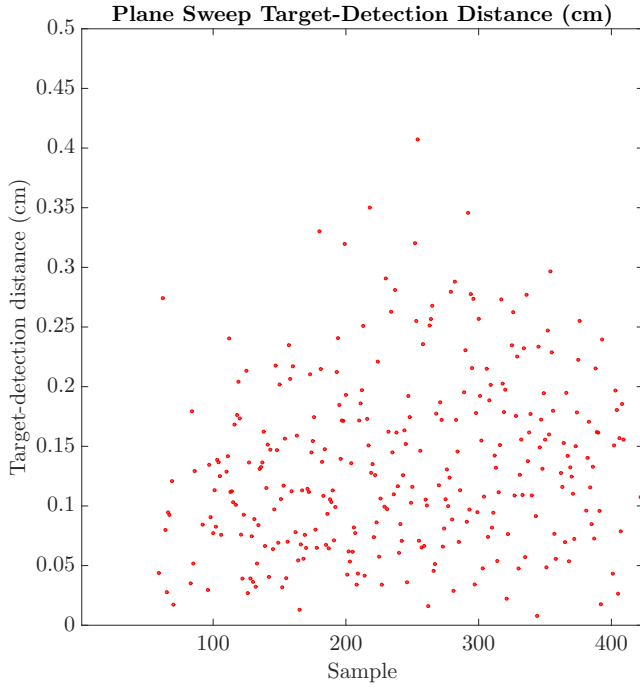
6.2.2.2 *Plane Sweep*

The plane sweep algorithm processed 296 of the touch targets in 7.63 milliseconds on average (0.63 milliseconds for the actual detection routines).

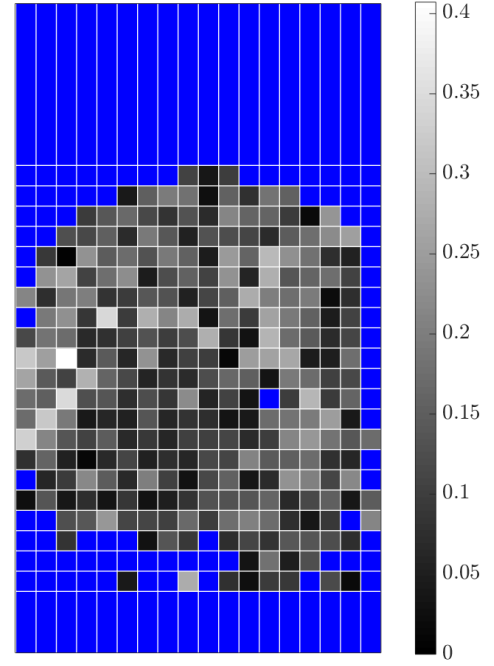
Touch-target-detection distance: Similarly to the projection space algorithm results, the touch-target-detection distances reported by the plane sweep algorithm are somewhat scattered, as shown in Figure 6.9. Here, the range of distances—from 0 to about 0.4 cm—is a little lower than that achieved by the projection space algorithm (Figure 6.9a). Figure 6.9b and Figure 6.9c present visualizations of these distances in projector space and in three-dimensional space, respectively.

Touch target confidence score: Figure 6.10 provides a summary of the plane sweep confidence scores—the distances between the planes with the minimum union area and the touch surface. The majority of the touch targets were assigned to planes with 0.2 cm of the surface, suggesting high touch confidence (Figure 6.10a). As seen in the projection space and the textured mesh result images— Figure 6.10b and Figure 6.10c, respectively—the curved parts of the bowl tend to be assigned to closer planes than the flat part. The algorithm is also less confident at parts of the perimeter.

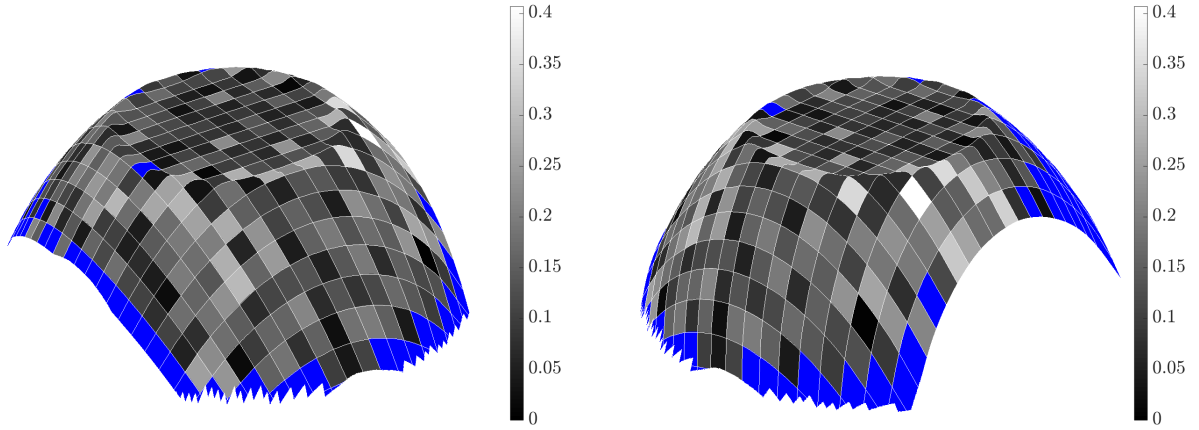
Touch/hover classification: Plane sweep touch/hover classification results are shown in Figure 6.11. Figure 6.11a plots the distance between the plane with minimum union area and the surface for each of the touch and hover samples, while Figure 6.11b presents overall and individual touch and hover classification accuracies using various thresholds for the minimum union plane. The majority of the touch samples are assigned to planes 0.3 cm from the surface and closer, and all but one of the hover samples are assigned to planes 0.3 cm from the surface and farther. In fact, all but two hovers are assigned to the farthest considered plane, located 0.6 cm from the surface.



(a) Graph over touch samples.

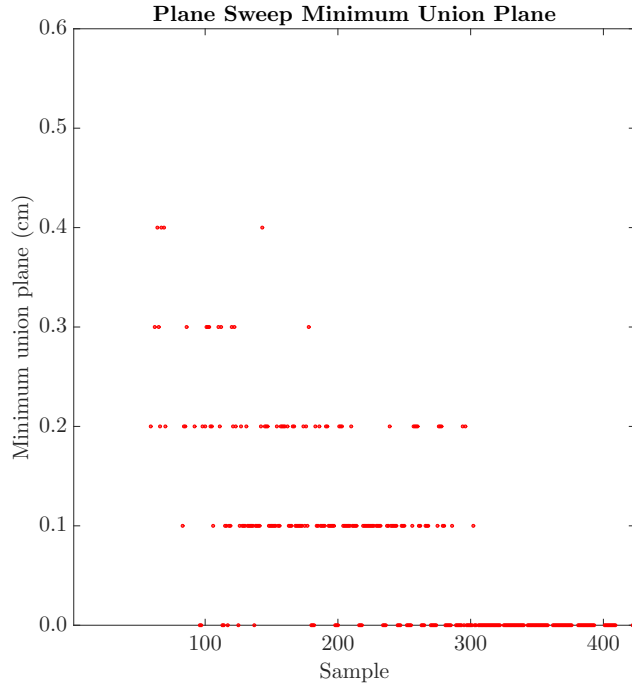


(b) Results shown in projection space.

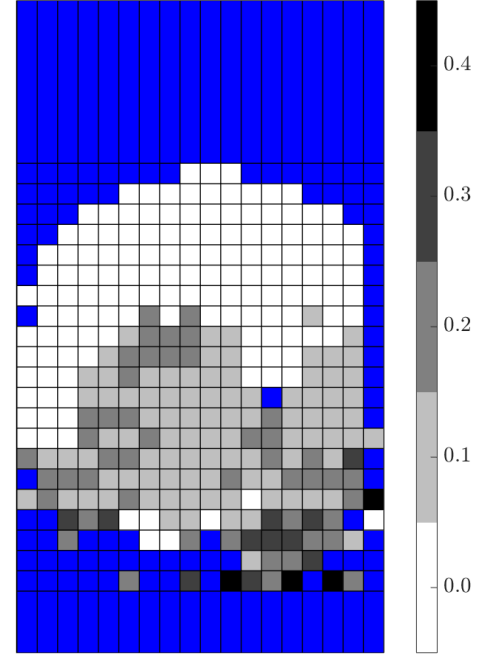


(c) Results shown in two views in 3D space on the bowl touch mesh S .

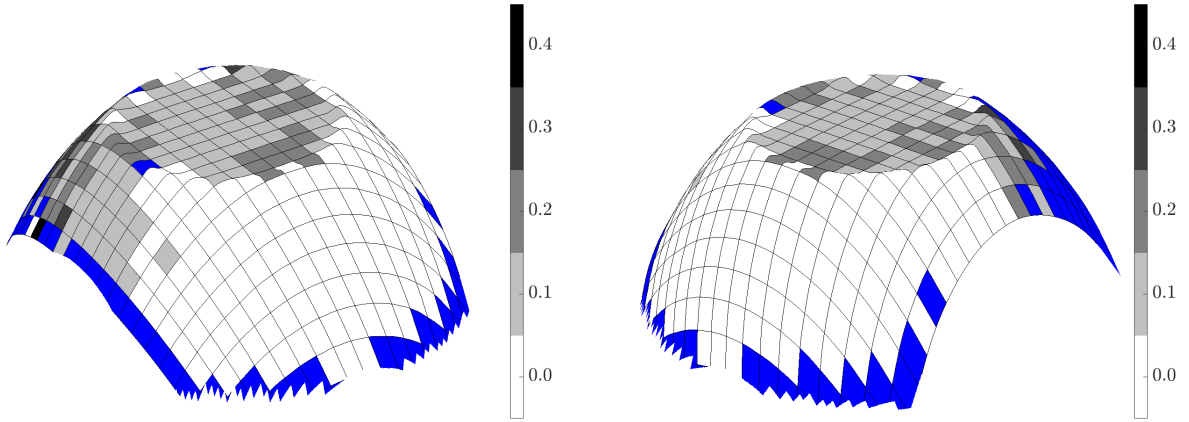
Figure 6.9: **Plane sweep algorithm** results for the **bowl surface**: distance (centimeters) between localized **touch targets** and the 3D lookup table correspondences.



(a) Graph over touch samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the bowl touch mesh S .

Figure 6.10: **Plane sweep algorithm** results for the **bowl surface**: minimum union plane (centimeters) for **touch targets**.

The best overall accuracy, 99.80% (100% of touches and 99.50% of hovers correctly classified), is achieved using a minimum union plane threshold of 0.4 cm.

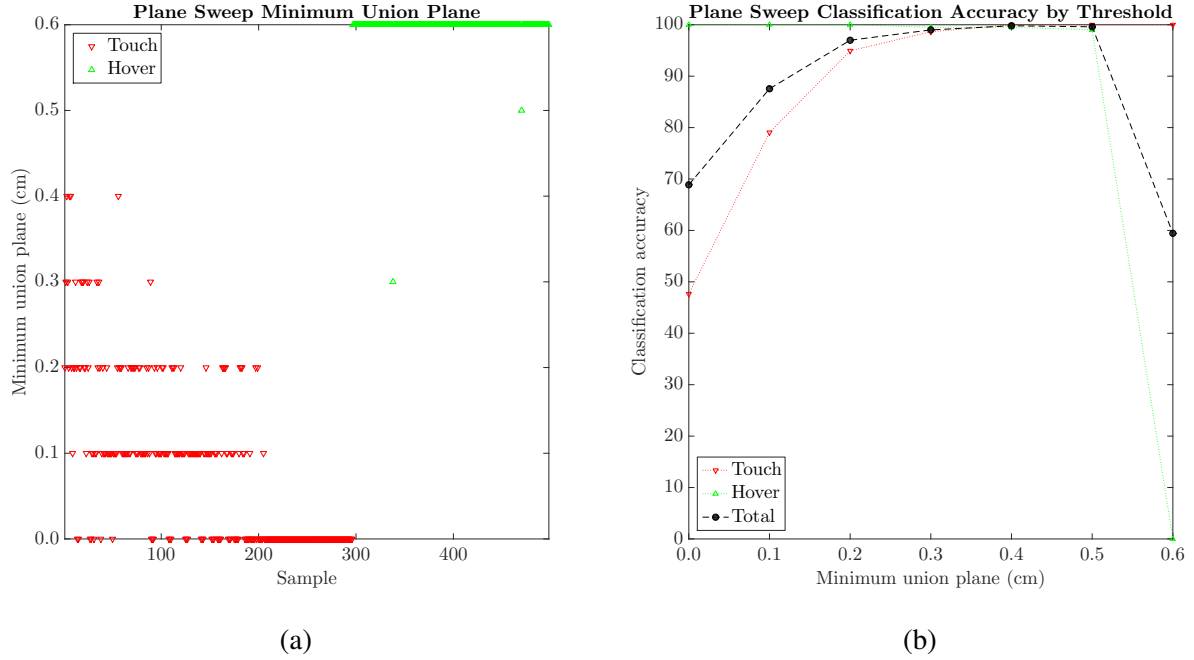


Figure 6.11: **Plane sweep algorithm** results for the **bowl surface: touch/hover classification**. (a) Confidence scores across touch and hover samples. (b) Overall touch/hover classification accuracy by confidence score threshold.

6.2.2.3 Summary

Table 6.2 shows a summary of the touch target results for the bowl surface, and Figure 6.12 plots the target-detection distances across the set of touch targets. Finally, Table 6.3 presents a summary comparison of touch/hover classification results.

The two algorithms performed comparably in terms of target-detection distances: each localized touches with an average distance of about 0.13 cm from the projected targets, with the projec-

tion space algorithm yielding slightly lower distances (approximately 0.003 cm). Likewise, each achieved touch/hover classification rates of over 99%. The plane sweep algorithm outperformed the projection space algorithm by two metrics. First, the plane sweep was able to successfully classify a few more of the touch targets than the projection space algorithm. In terms of runtime performance, the plane sweep algorithm processed the touch targets significantly faster than the projection space algorithm: 7.63 and 9.90 milliseconds overall (0.63 and 2.83 milliseconds for detection alone), respectively.

Table 6.2: **Touch target** results summary for the **bowl surface**.

Algorithm	Successful	Localization	Confidence Score	Runtime
Projection space	293	0.1311 cm	0.614	Detect: 2.83 ms Total: 9.90 ms
Plane sweep	296	0.1342 cm	0 cm: 47.64% 0.1 cm: 31.42% 0.2 cm: 15.88% 0.3 cm: 3.72% 0.4 cm: 1.35%	Detect: 0.63 ms Total: 7.63 ms

Table 6.3: **Touch/hover classification** results summary for the **bowl surface**.

Algorithm	Best Accuracy	Score Threshold
Projection space	99.60% (99.66% touch, 99.50% hover)	0.3
Plane sweep	99.80% (100% touch, 99.50% hover)	0.4 cm

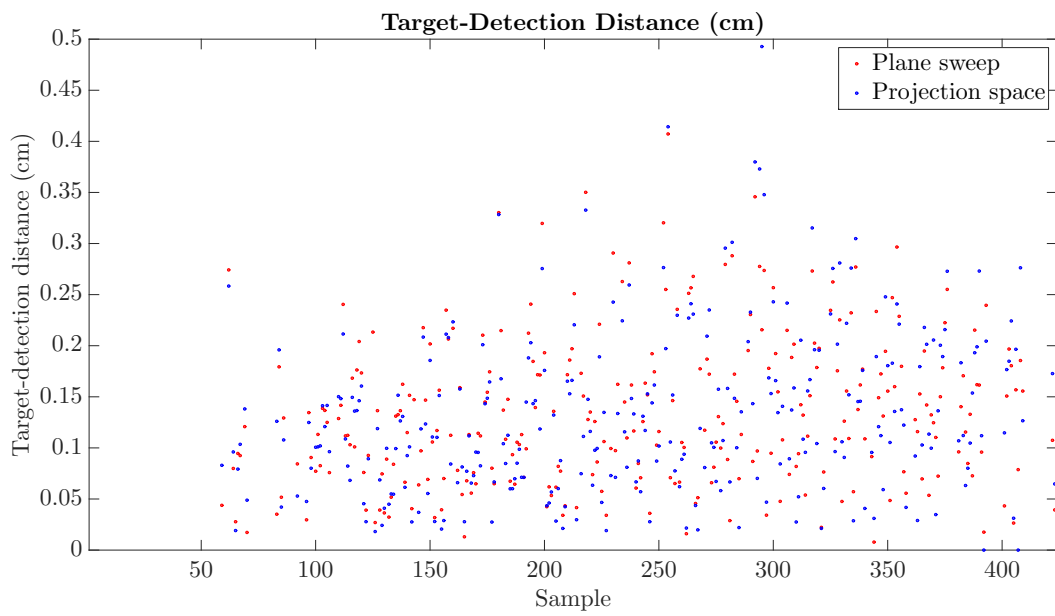


Figure 6.12: Comparison of **projection space** (blue) and **plane sweep** (red) algorithm results: distance (centimeters) between localized **touch targets** and the 3D lookup table correspondences for the **bowl surface**, shown across the set of touch samples. Both algorithms perform about the same on the touch samples.

6.3 Head

Next, we present projected target (Section 6.3.1) and touch target (Section 6.3.2) results for the head surface.

6.3.1 Projector Targets

For the head surface, the feature scan consisted of 3075 features, of which approximately 2300 were located on the surface and were visible to at least two cameras.

6.3.1.1 Projection Space

The projection space algorithm successfully classified 2267 of the feature scan samples at an average speed of 9.43 milliseconds, with detection alone executing in 3.38 milliseconds.

Projected-target-detection distance: Overall target-detection distance results are shown in Figure 6.13. On average, the distance between the localized detections and the projector targets in 3D space was 0.0256 cm. These distances are plotted across the set of feature scan samples in Figure 6.13a. Figure 6.13b presents the same data graphically in projector space, and Figure 6.13c visualizes these distances in the 3D context of the touch mesh S . Here, the color blue corresponds to coordinates in projection space that are outside the bounds of the head surface. These figures show that the samples with the highest errors correspond to the perimeter of the projectable area of the head surface.

Projected target confidence score: Overall confidence score results are shown in Figure 6.14. The mean multi-camera agreement score across the detected targets was 0.742. Feature scan sample scores are plotted in Figure 6.14a. These confidence scores are shown graphically in projector

space in Figure 6.14b and in the 3D context of the touch mesh S in Figure 6.14c. Similarly to the distance scores, the algorithm is a little less confident for targets on the perimeter of the lookup table data; additionally, targets on some of the curved regions of the surface—such as the nose and the sides—are less confidently classified.

6.3.1.2 *Plane Sweep*

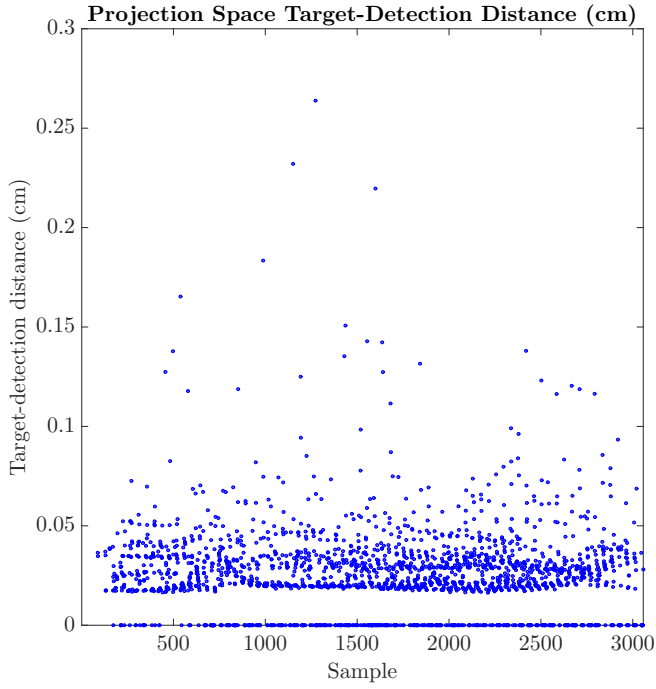
The plane sweep algorithm successfully classified 2266 of the feature scan samples at an average speed of 6.71 milliseconds, with detection alone executing in 0.60 milliseconds.

Projected-target-detection distance: Overall target-detection distance results are shown in Figure 6.15. On average, the distance between the localized detections and the projector targets in 3D space was 0.0132 cm, and the distances are generally less than 0.05 cm for the majority of the targets. These distances are shown across the set of feature scan samples in Figure 6.15a, in projector space in Figure 6.15b, and on the touch mesh S in Figure 6.15c. Here, the highest errors again correspond to targets on the perimeter of the lookup table data.

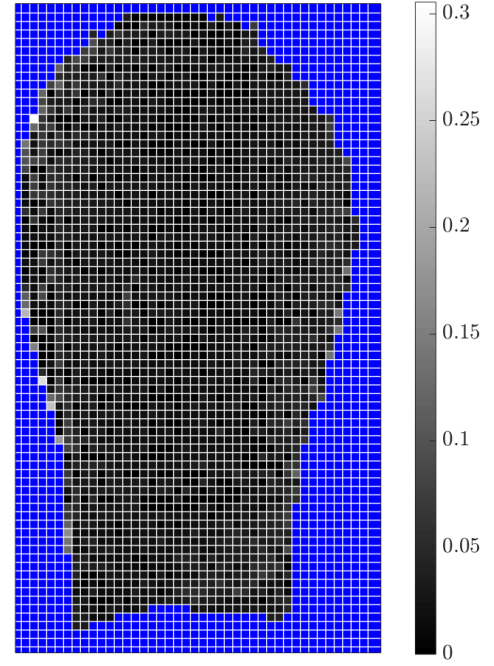
Projected target confidence score: Overall confidence score results are shown in Figure 6.16. Feature scan sample scores are plotted in Figure 6.16a. All detections were assigned to the plane closest to the surface, which indicates the highest possible confidence of a touch classification. These confidence scores are presented graphically in projector space in Figure 6.16b and in the 3D context of the touch mesh S in Figure 6.16c.

6.3.1.3 *Summary*

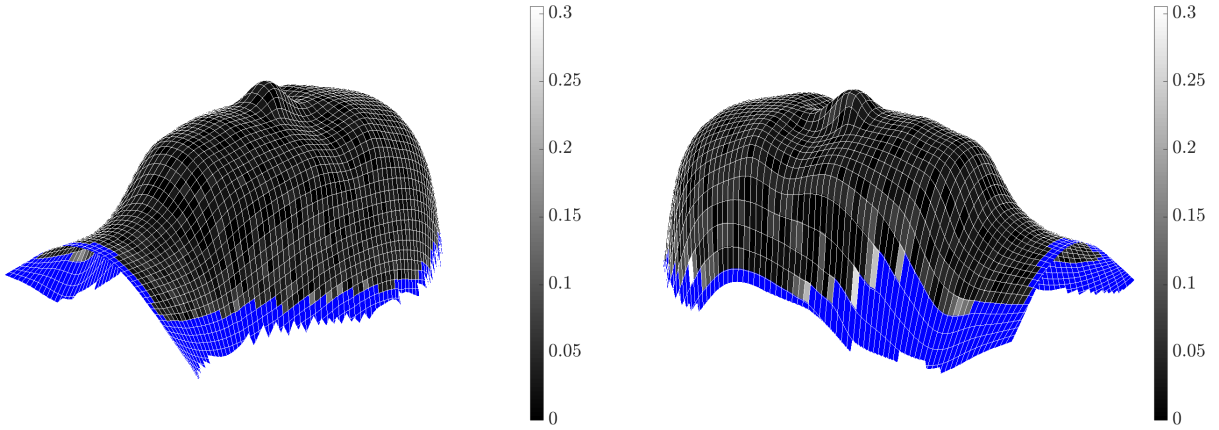
The feature scan projector target result summary for the head surface is shown in Table 6.4. A comparison graph between the projected target-localization distances is shown in Figure 6.17.



(a) Graph over feature scan samples.

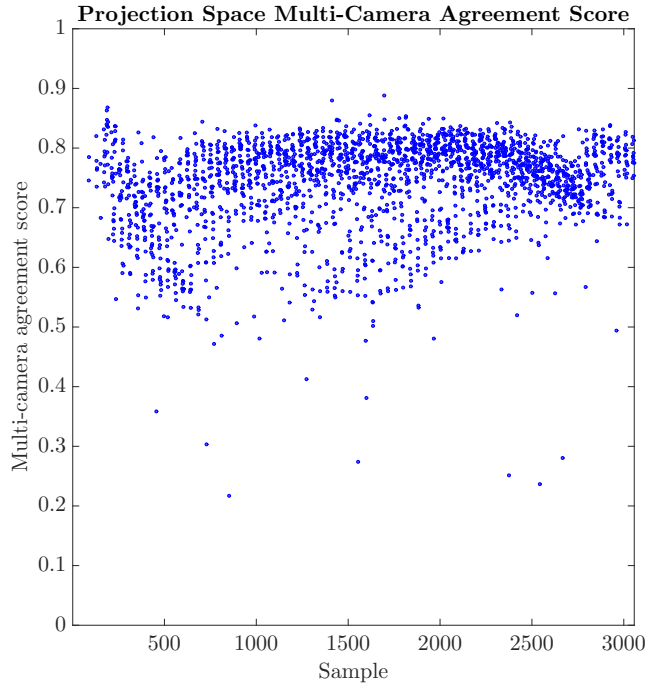


(b) Results shown in projection space.

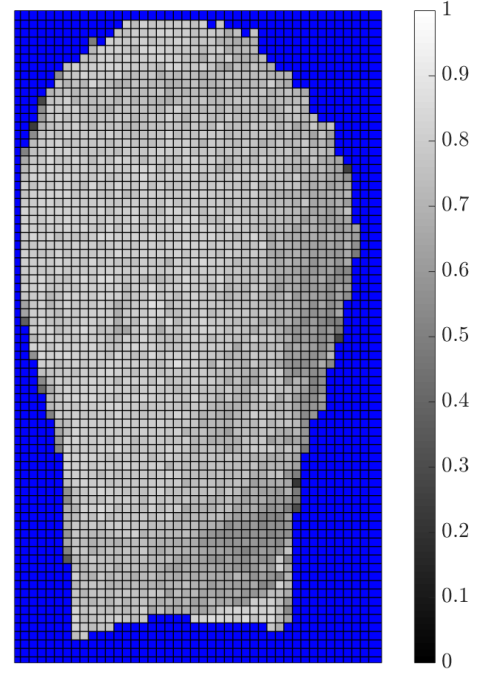


(c) Results shown in two views in 3D space on the head touch mesh S .

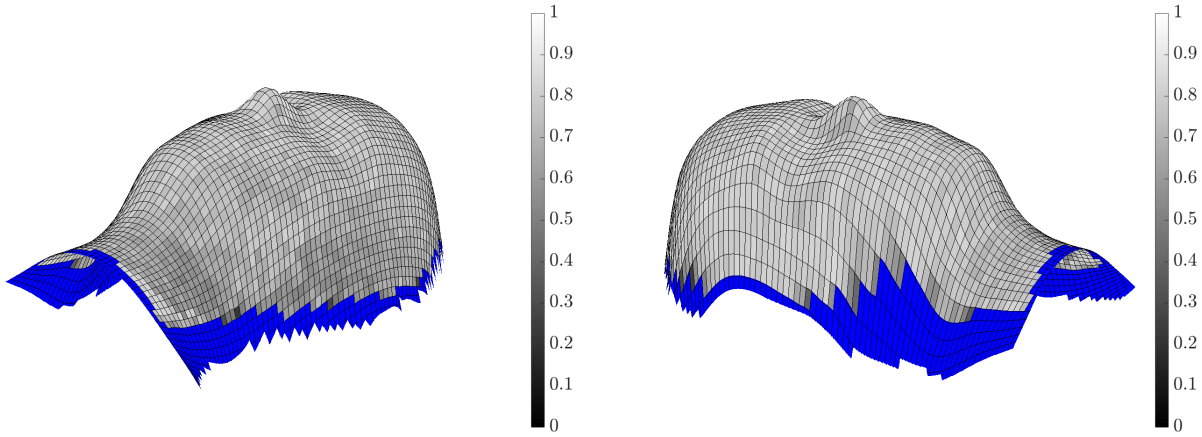
Figure 6.13: **Projection space algorithm** results for the **head surface**: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences.



(a) Graph over feature scan samples.

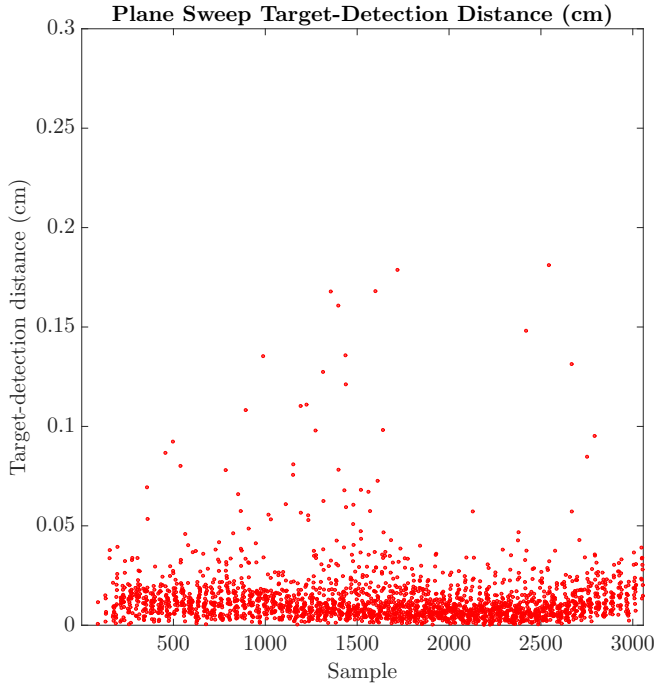


(b) Results shown in projection space.

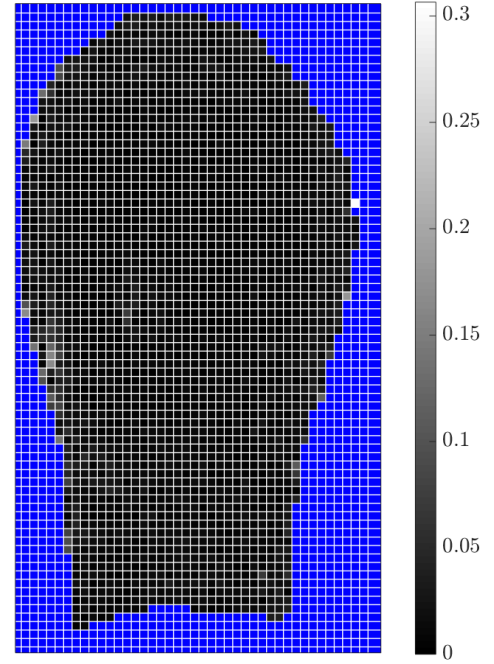


(c) Results shown in two views in 3D space on the head touch mesh S .

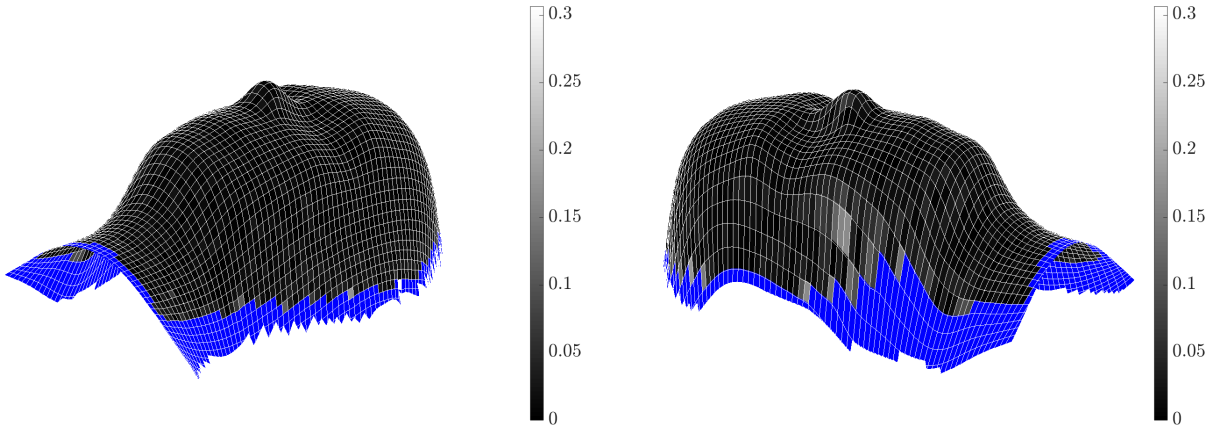
Figure 6.14: **Projection space algorithm** results for the **head surface**: multi-camera agreement score for **projected targets**.



(a) Graph over feature scan samples.

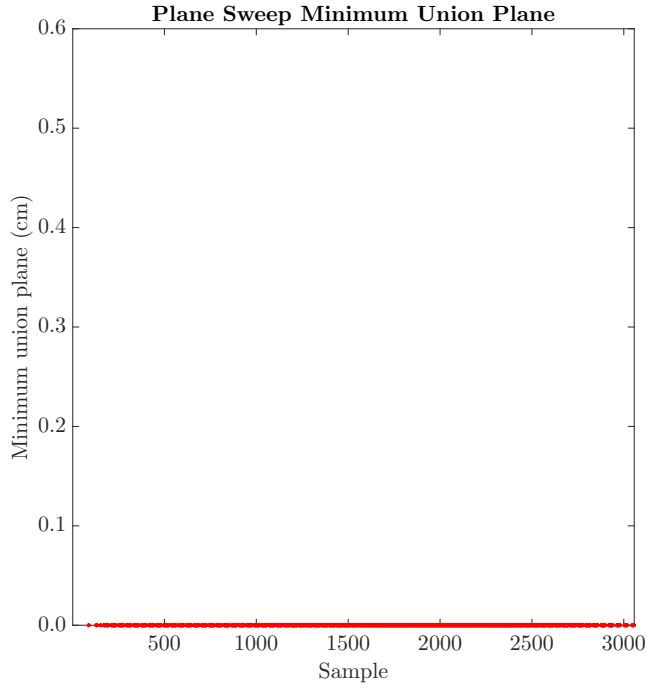


(b) Results shown in projection space.

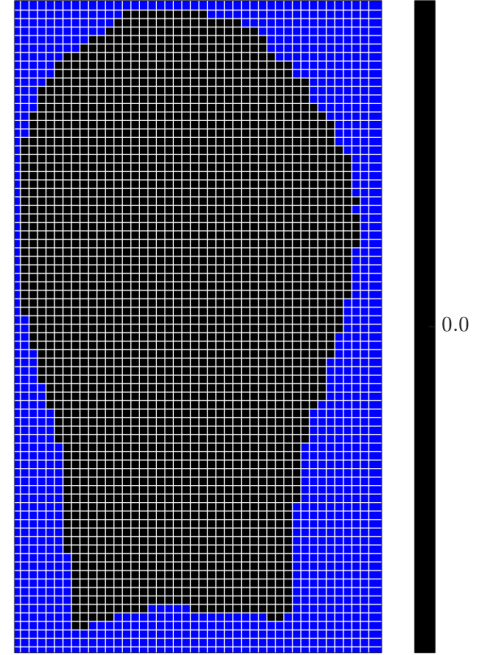


(c) Results shown in two views in 3D space on the head touch mesh S .

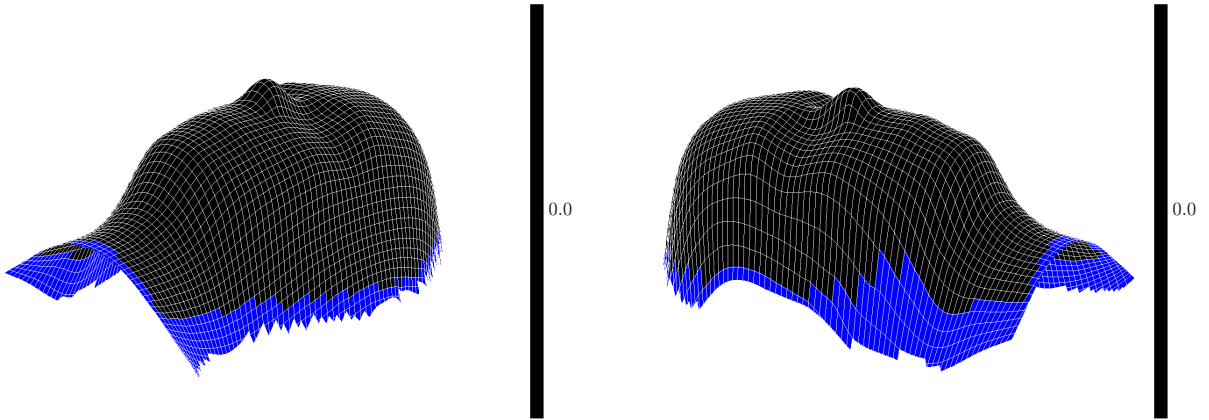
Figure 6.15: **Plane sweep algorithm** results for the **head surface**: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences.



(a) Graph over feature scan samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the head touch mesh S .

Figure 6.16: **Plane sweep algorithm** results for the **head surface**: minimum union plane (centimeters) for **projected targets**.

The projection space algorithm successfully classified one more target. However, the plane sweep algorithm outperformed the projection space algorithm by the remaining metrics:

- The average 3D distance between the targets and the localized detections on the touch mesh S was half that achieved by the projection space algorithm.
- While the confidence scores are not directly comparable, for the plane sweep algorithm, the minimum union plane indices were uniformly chosen to be the closest plane to the surface, indicating the highest possible confidence of a touch assignment. However, the multi-camera agreement scores for the projection space algorithm varied across the samples, with some having relatively low values; a reasonable threshold would misclassify some of the projected targets as hovers.
- Finally, the plane sweep algorithm outperformed the projection space algorithm in terms of execution time. When considering overall execution time, which includes segmenting camera imagery into contours, the two algorithms executed in 6.71 and 9.43 milliseconds, respectively. However, if we ignore segmentation time, which is common to both algorithms, the plane sweep significantly outperforms the projection space algorithm, detecting and localizing imagery in 0.60 milliseconds (compared to 3.38 milliseconds).

Table 6.4: **Projector target** results summary for the **head surface**.

Algorithm	Successful	Localization	Confidence Score	Runtime
Projection space	2267	0.0256 cm	0.742	Detect: 3.38 ms Total: 9.43 ms
Plane sweep	2266	0.0132 cm	0 cm: 100%	Detect: 0.60 ms Total: 6.71 ms

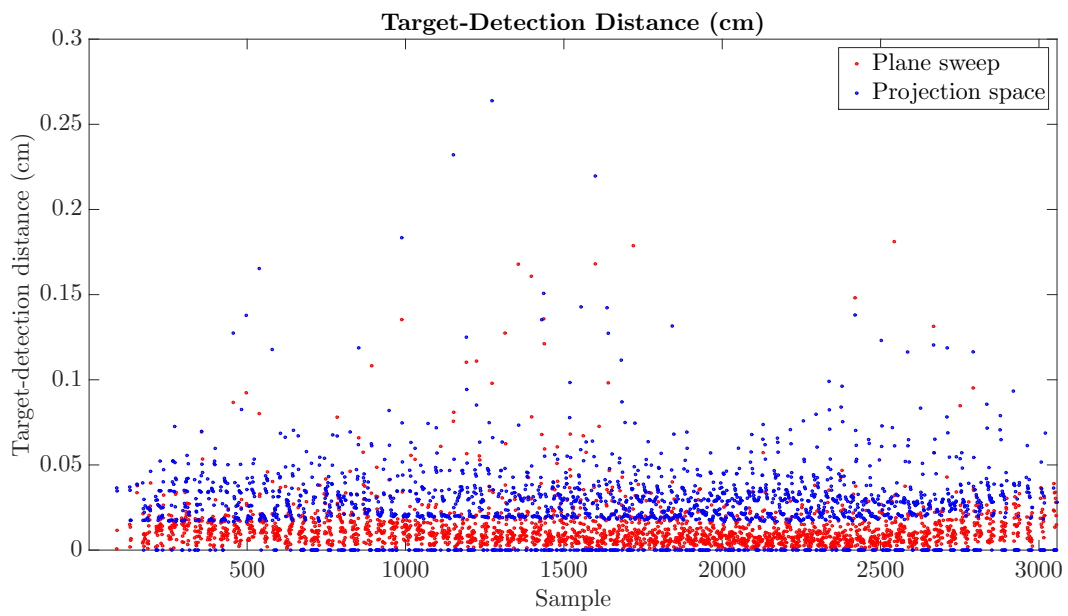


Figure 6.17: Comparison of **projection space** (blue) and **plane sweep** (red) algorithm results: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences for the **head surface**, shown across the samples of the feature scan. On average, the plane sweep distances are lower.

6.3.2 Touch Targets

For the head surface, the touch target dataset is composed of over 350 touch samples, captured across a grid of visual targets.

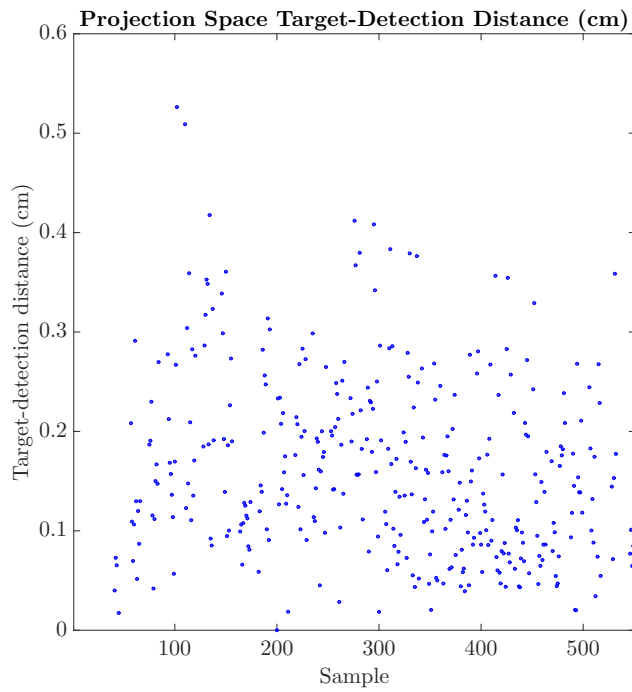
6.3.2.1 Projection Space

The projection space algorithm classified 359 touch targets at an overall average speed of 10.53 milliseconds; classification and localization alone executed in 3.31 milliseconds.

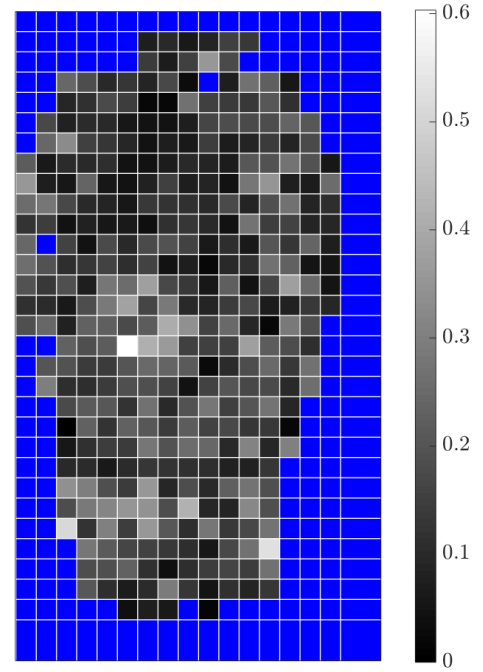
Touch-target-detection distance: The touch-target-distance results produced by the projection space algorithm are summarized in Figure 6.18. In general, these distances range from 0 to 0.6 cm, with an average of 0.1623 cm (Figure 6.18a). The largest errors tend to correspond to the curved portions of the nose and the neck of the head surface, as shown in the projection space image (Figure 6.18b) and in the 3D textured touch mesh image (Figure 6.18c).

Touch target confidence score: Figure 6.19 shows the touch target multi-camera agreement scores computed by the projection space algorithm. The majority are above a confidence score threshold of 0.4, with an average of 0.620 (Figure 6.19a). Some of the lower scores are clustered around areas of high curvature on the forehead regions, evident in the projector space results image (Figure 6.19b) and in the textured touch mesh image (Figure 6.19c).

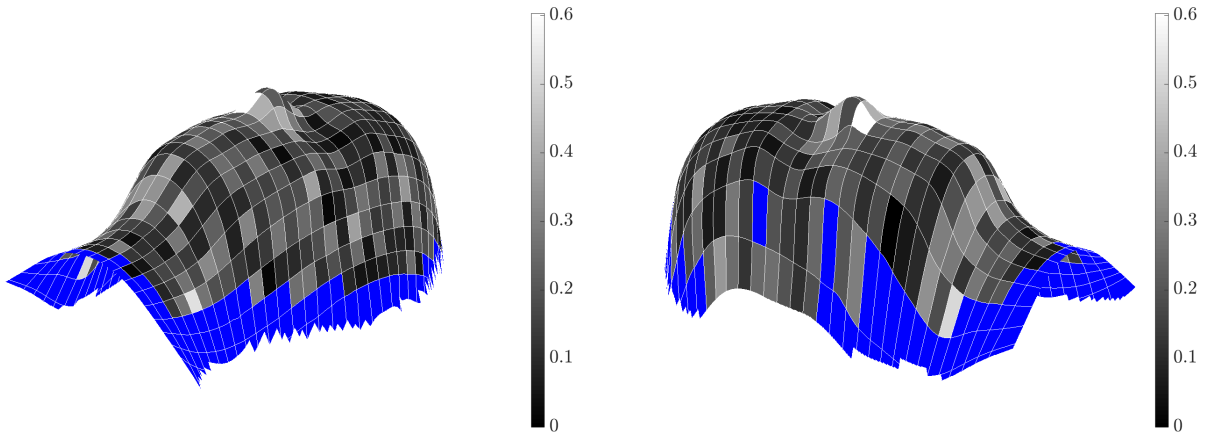
Touch/hover classification: Touch/hover classification results for the head surface computed by the projection space algorithm are shown in Figure 6.20. The multi-camera agreement confidence scores are plotted in Figure 6.20a over the set of touch and hover samples. Most of the touches have confidences above 0.4, and most of the hovers have confidences below this value. Combined touch/hover accuracy is presented in Figure 6.20b across a range of confidence score thresholds.



(a) Graph over touch samples.

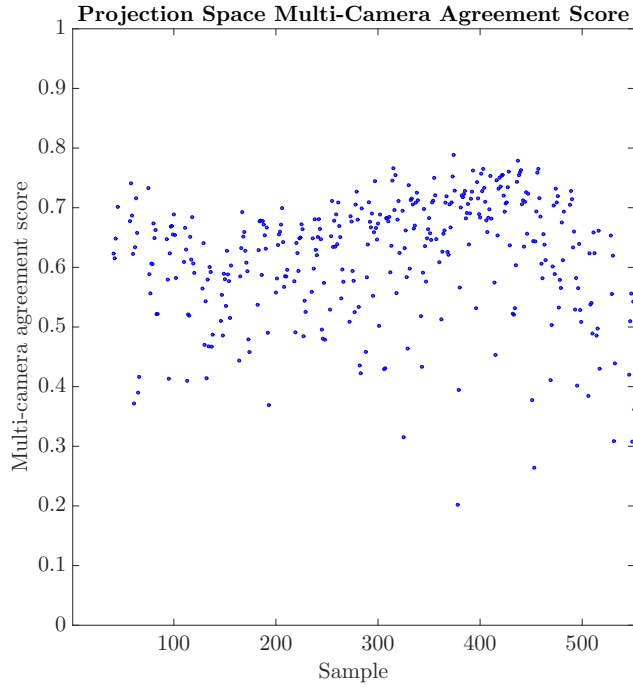


(b) Results shown in projection space.

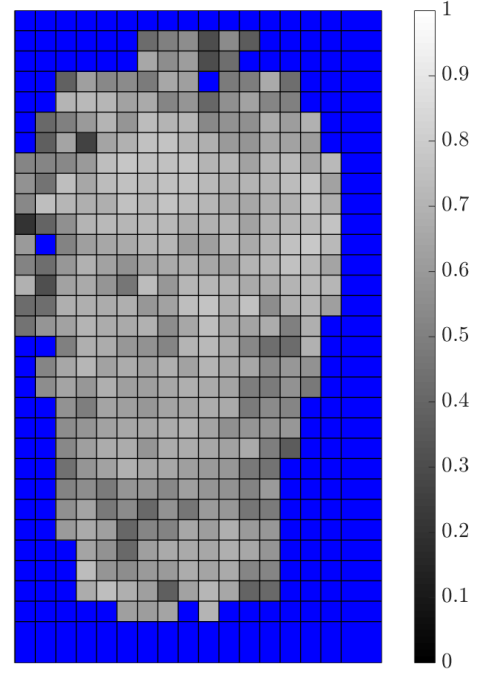


(c) Results shown in two views in 3D space on the head touch mesh S .

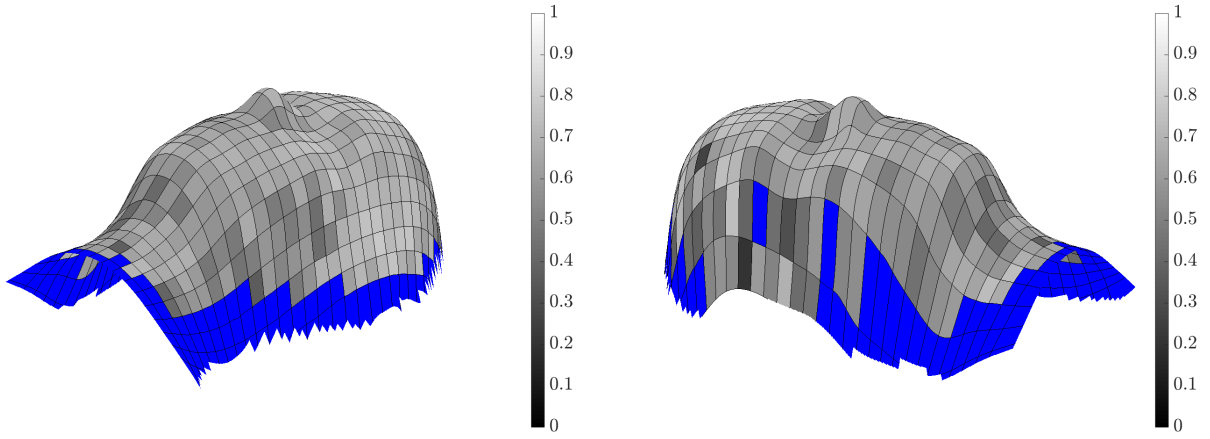
Figure 6.18: **Projection space algorithm** results for the **head surface**: distance (centimeters) between localized **touch targets** and the 3D lookup table correspondences.



(a) Graph over touch samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the head touch mesh S .

Figure 6.19: **Projection space algorithm** results for the **head surface**: multi-camera agreement score for **touch targets**.

Additionally, the individual classification accuracies for touches and for hovers are shown. With a threshold of 0.4, the projection space algorithm achieves its highest accuracy, correctly classifying 96.92% of all samples (96.66% of touches and 97.52% of hovers correct).

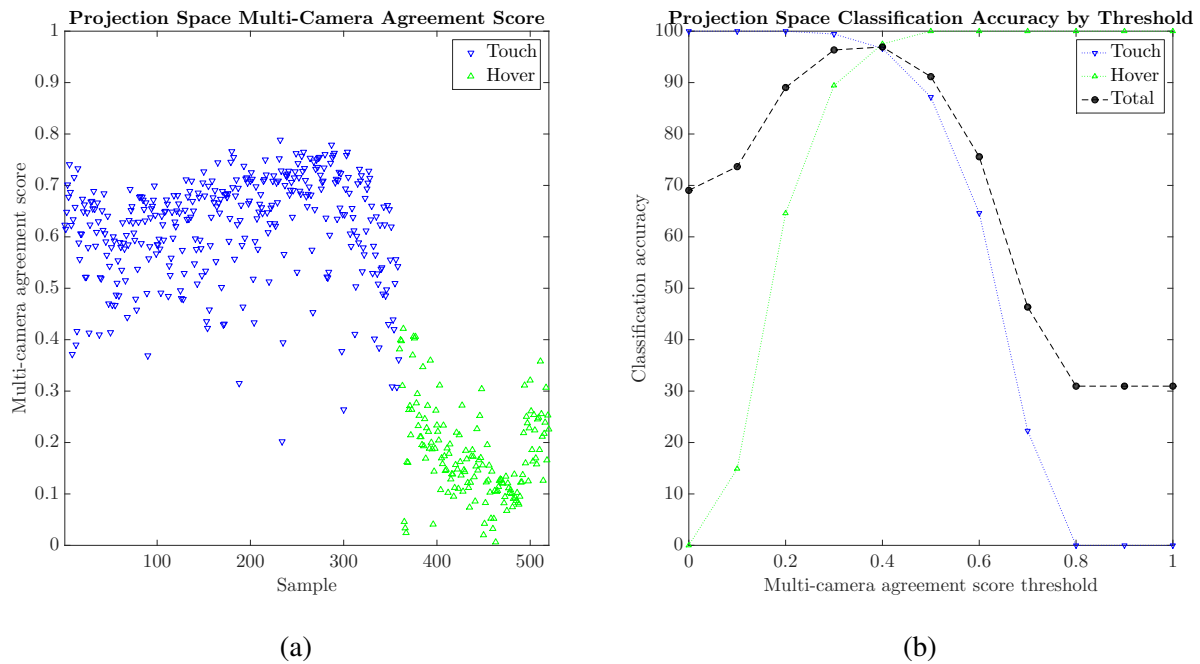


Figure 6.20: **Projection space algorithm** results for the **head surface: touch/hover classification**. (a) Confidence scores across touch and hover samples. (b) Overall touch/hover classification accuracy by confidence score threshold.

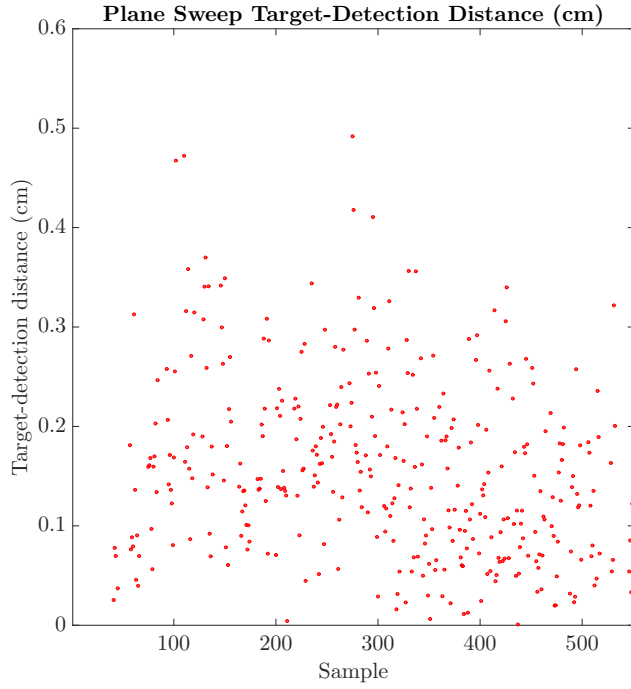
6.3.2.2 *Plane Sweep*

The plane sweep touch detection algorithm processed 359 of the touch targets. Overall, it operated in 7.81 milliseconds, with detection routines alone executing in 0.74 milliseconds.

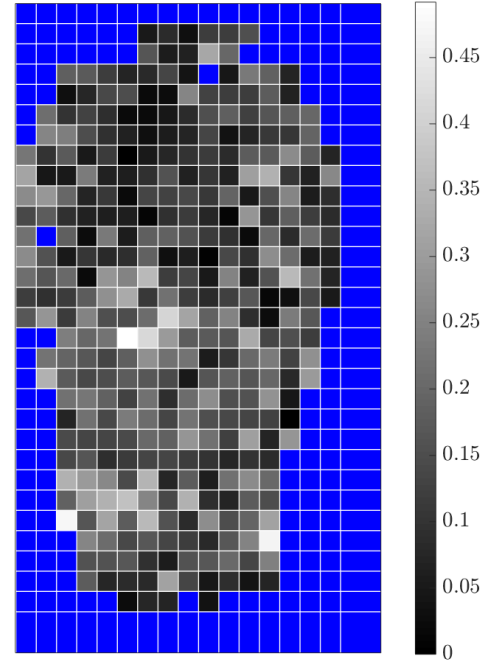
Touch-target-detection distance: The touch-target-detection distances achieved by the plane sweep algorithm are shown in Figure 6.21. Like the results reported by the projection space algorithm, they are scattered about a range from 0 to 0.5 cm (Figure 6.21a), with an average distance of 0.1560 cm. As seen in the projector space and 3D visualizations (Figure 6.21b and Figure 6.21c, respectively), the largest distances are near the nose and neck of the head surface.

Touch target confidence score: A summary of the plane sweep confidence scores is shown in Figure 6.22. All but two of the touch targets were assigned to planes within 0.2 cm of the surface, indicating extremely high touch confidence (Figure 6.22a). These two touch targets for which the algorithm was less confident are located on the neck of the head surface, as visualized in Figure 6.22b and Figure 6.22c.

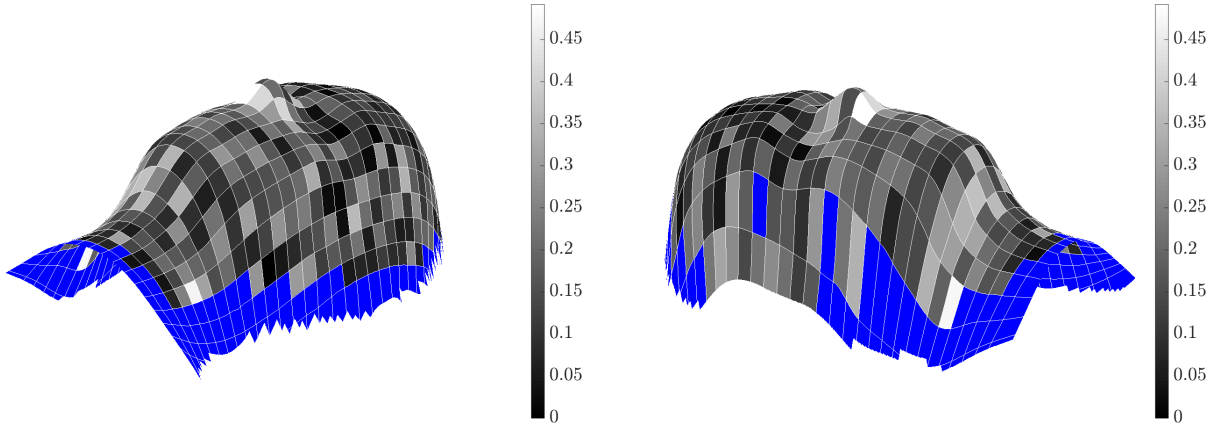
Touch/hover classification: Figure 6.23 shows the plane sweep touch/hover classification results. The confidence scores—the distances between the surface and the planes with minimum union area—are plotted across the touch and hover samples in Figure 6.23a. Nearly all of the hovers are assigned to the farthest plane, located 0.6 cm from the surface. Thus, the plane sweep is able to successfully distinguish between touches and hovers with extremely high accuracy, classifying 99.43% of the samples (99.44% of touches and 99.39% of hovers) correctly when using a plane threshold of 0.2 cm (Figure 6.23b).



(a) Graph over touch samples.

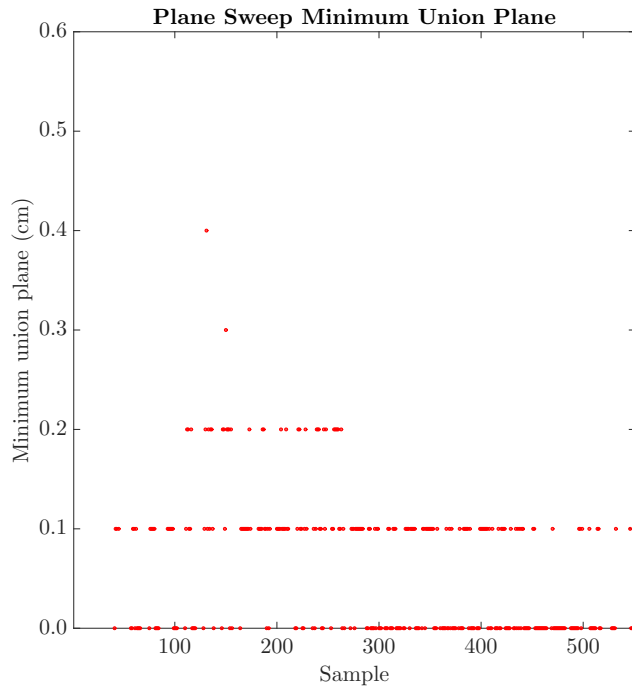


(b) Results shown in projection space.

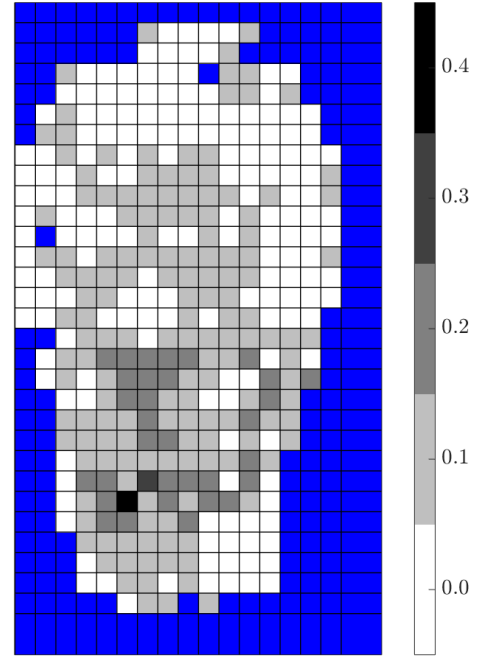


(c) Results shown in two views in 3D space on the head touch mesh S .

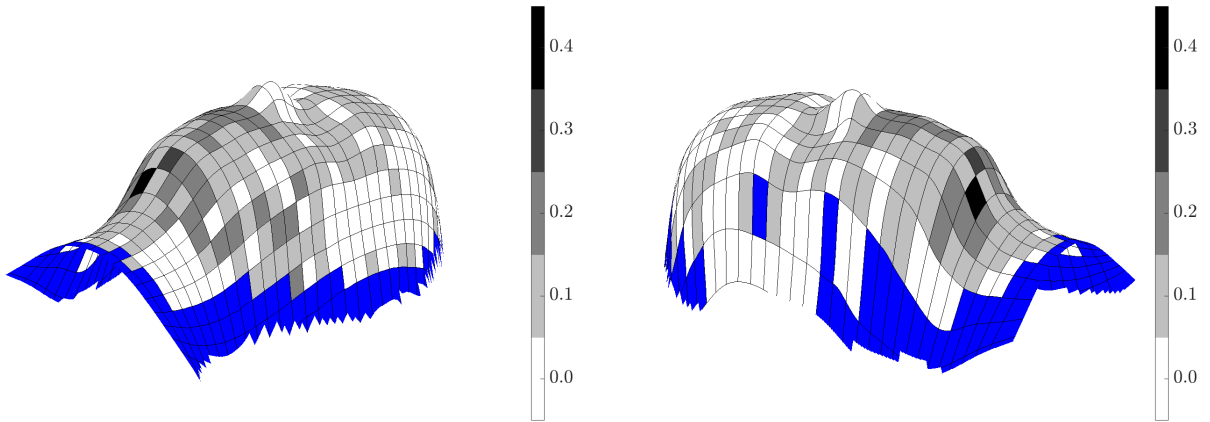
Figure 6.21: **Plane sweep algorithm** results for the **head surface**: distance (centimeters) between localized **touch targets** and the 3D lookup table correspondences.



(a) Graph over touch samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the head touch mesh S .

Figure 6.22: **Plane sweep algorithm** results for the **head surface**: minimum union plane (centimeters) for **touch targets**.

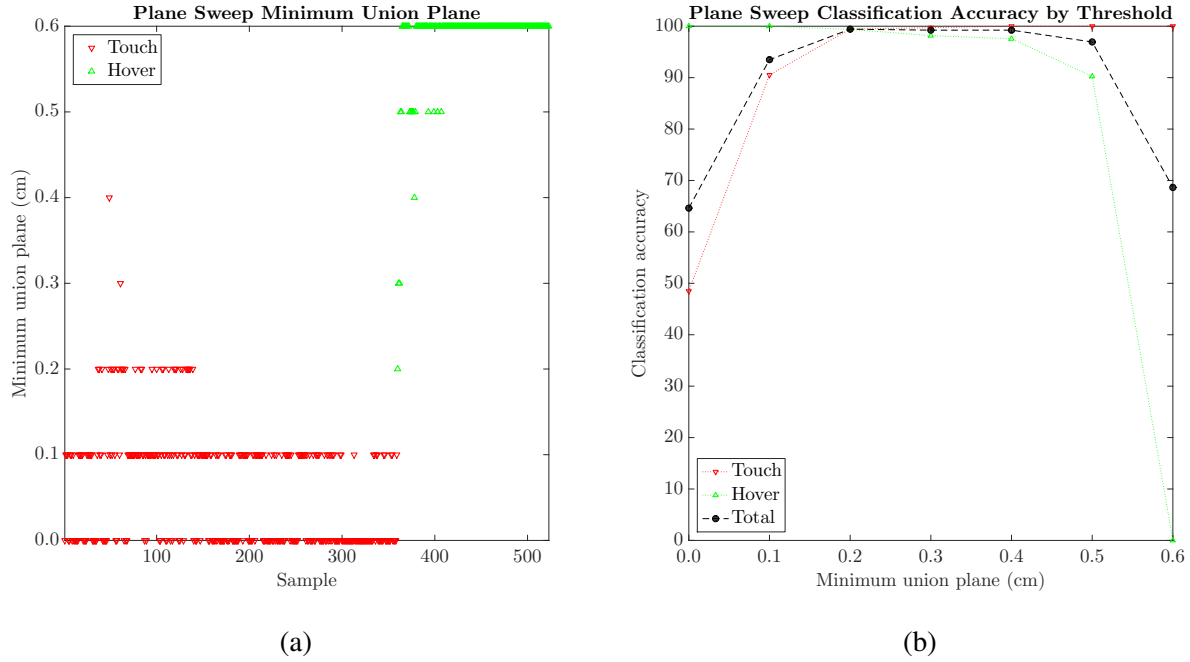


Figure 6.23: **Plane sweep algorithm** results for the **head surface: touch/hover classification**. (a) Confidence scores across touch and hover samples. (b) Overall touch/hover classification accuracy by confidence score threshold.

6.3.2.3 Summary

A summary of the touch target results for the head surface is shown in Table 6.5, and the touch-target-detection distances are plotted in Figure 6.24. Touch/hover classification results are summarized in Table 6.6.

The plane sweep algorithm slightly outperformed the projection space algorithm in terms of touch-target-detection distances, achieving an average distance of 0.1560 cm compared to 0.1623 cm. Confidence scores tended to be more indicative of touches for the plane sweep, which further outperformed the projection space algorithm in touch/hover classification accuracy (99.43% overall accuracy compared to 96.92%). Finally, as in previous results, the plane sweep algorithm required

less time to classify the touch targets than the projection space algorithm; the former executed in 7.81 milliseconds on average (0.74 milliseconds for detection), while the latter operated in 10.53 milliseconds (3.31 milliseconds for detection).

Table 6.5: **Touch target** results summary for the **head surface**.

Algorithm	Successful	Localization	Confidence Score	Runtime
Projection space	359	0.1623 cm	0.620	Detect: 3.31 ms Total: 10.53 ms
Plane sweep	359	0.1560 cm	0 cm: 48.47% 0.1 cm: 42.06% 0.2 cm: 8.91% 0.3 cm: 0.28% 0.4 cm: 0.28%	Detect: 0.74 ms Total: 7.81 ms

Table 6.6: **Touch/hover classification** results summary for the **head surface**.

Algorithm	Best Accuracy	Score Threshold
Projection space	96.92% (96.66% touch, 97.52% hover)	0.4
Plane sweep	99.43% (99.44% touch, 99.39% hover)	0.2 cm

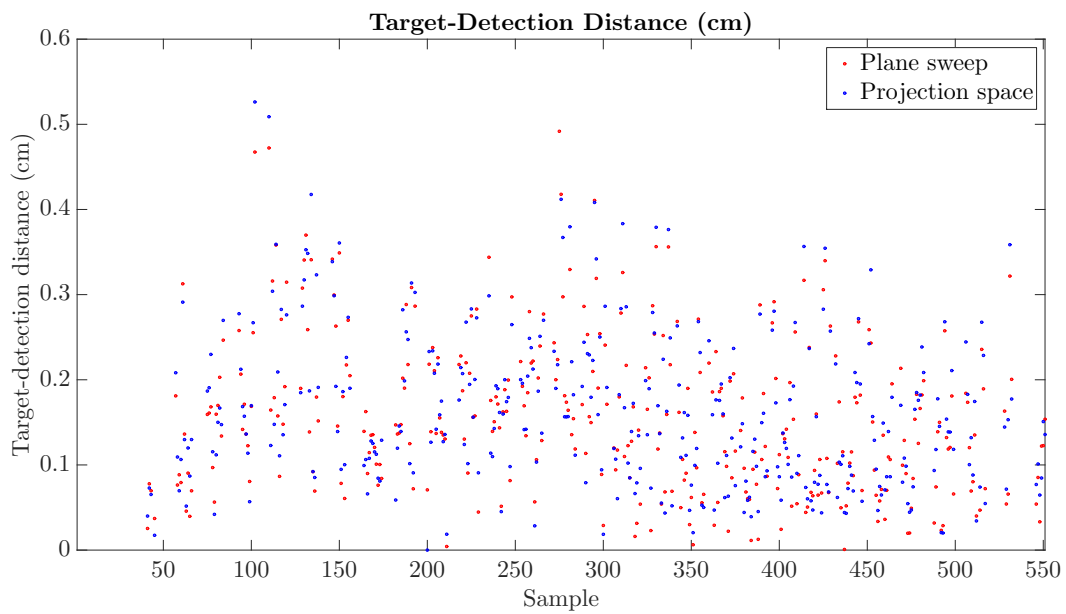


Figure 6.24: Comparison of **projection space** (blue) and **plane sweep** (red) algorithm results: distance (centimeters) between localized **touch targets** and the 3D lookup table correspondences for the **head surface**, shown across the set of touch samples. Both algorithms perform about the same on the touch samples, with the plane sweep achieving slightly lower distances on average.

6.4 Child

Unlike the bowl and head surfaces supported by Prototype Rig I, for which touch detection is facilitated through four infrared cameras, the setup for the child surface on Prototype Rig II uses three touch sensing cameras (oriented toward the head of the surface) and four additional cameras to aid in reconstruction (oriented toward the body). The latter cameras are unable to sense infrared light, and so we only report touch target results (Section 6.4.2) for the regions of the child surface the former cameras are capable of imaging. To supplement these results, we present localization and classification results for the projected targets across the entire surface (Section 6.4.1), which provides evidence that successful touch sensing is possible in these regions through the use of additional infrared-sensing cameras.

Furthermore, imagery for the child surface is provided by two projectors rather than one. The *body projector*, capable of covering most of the surface, is responsible for displaying imagery on the legs and torso regions of the child shell. Imagery for the child’s head is provided by a specific *head projector*. As such, we present projected target results for the two projectors separately, along with summary results across them both.

6.4.1 Projector Targets

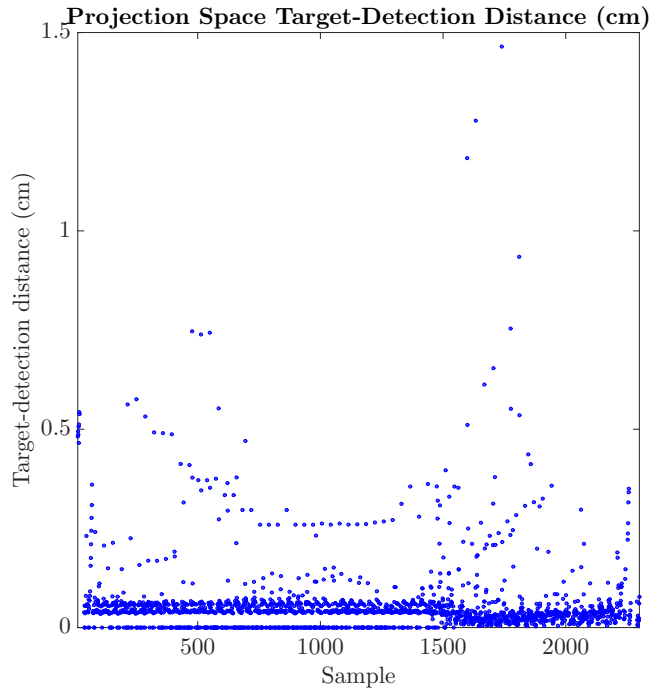
The feature scan for the child surface consisted of nearly 4000 total features projected onto the surface, with approximately half from each of the two projectors. Below, we present visualizations of projector target classification and localization results separately for the two projectors along with summary results across the entire set of projected targets.

6.4.1.1 *Projection Space*

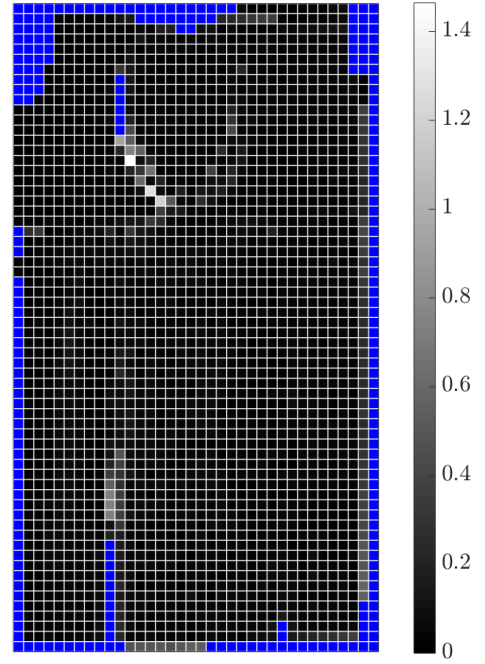
The projection space algorithm successfully classified 3899 of the projected feature samples across the two projectors. Processing executed in 20.41 milliseconds on average (5.99 milliseconds for detection alone).

Projected-target-detection distance: Figure 6.25 and Figure 6.26 show summary projected-target-detection distance results for the body and head projectors, respectively. The body projector covers a larger physical volume, requiring more cameras to image; while target-detection distances are relatively low in general (Figure 6.25a), there are a few large errors visible at certain curved parts of the child’s arm and face (shown in projector space in Figure 6.25b and textured on the touch mesh in Figure 6.25c). Several of the projected targets on the face are significantly distorted due to the orientation of the body projector relative to this region, and some targets in this region are not visible to at least two cameras. By comparison, the projected-target-detection results for the head projector are consistently low (Figure 6.26a). In this case, some of the projected features on one of the curved parts of the child’s face are not successfully localized, again due to an insufficient number of cameras capable of imaging these regions, as shown in Figure 6.26b and Figure 6.26c. The average projected-target-detection distance across the entire collection of features is 0.0473 cm.

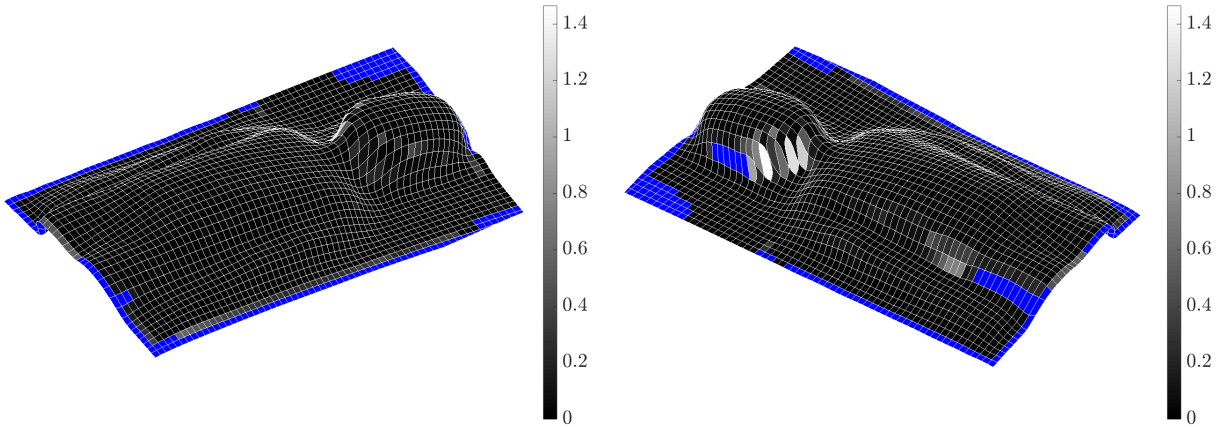
Projected target confidence score: Projection space algorithm confidence scores are summarized for the body and head projector targets in Figure 6.27 and Figure 6.28, respectively. Scores across both projectors are comparable, with most over 0.7 and a variety of lower scores spread out across the targets (Figure 6.27a and Figure 6.28a); the overall average confidence score for the entire set of targets from both projectors is 0.824. Targets with lower confidence scores are located in similar regions, such as the curved parts of the face and shoulders, as seen in the visualizations in projection space (Figure 6.27b and Figure 6.28b) and on the 3D mesh (Figure 6.27c and Figure 6.28c).



(a) Graph over feature scan samples.

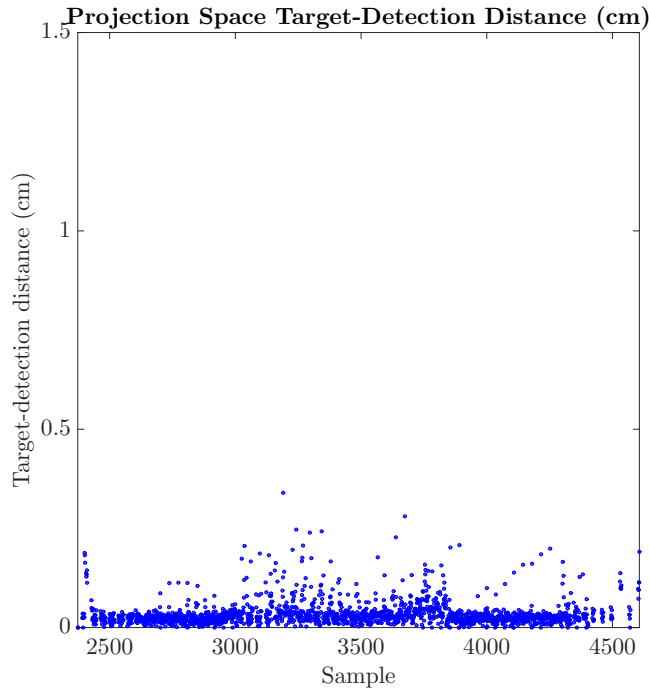


(b) Results shown in projection space.

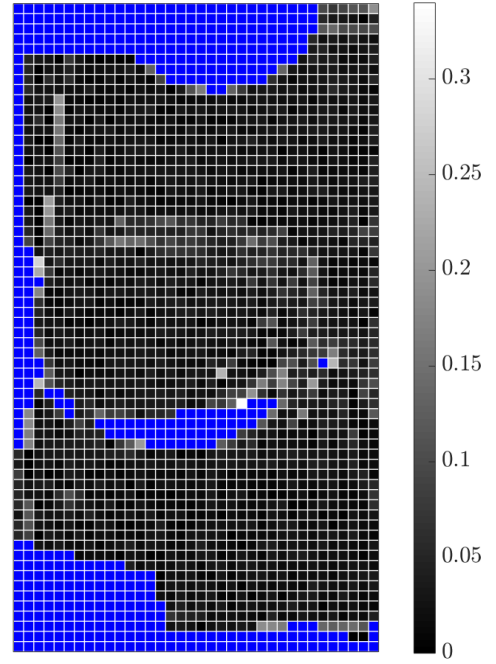


(c) Results shown in two views in 3D space on the child touch mesh S .

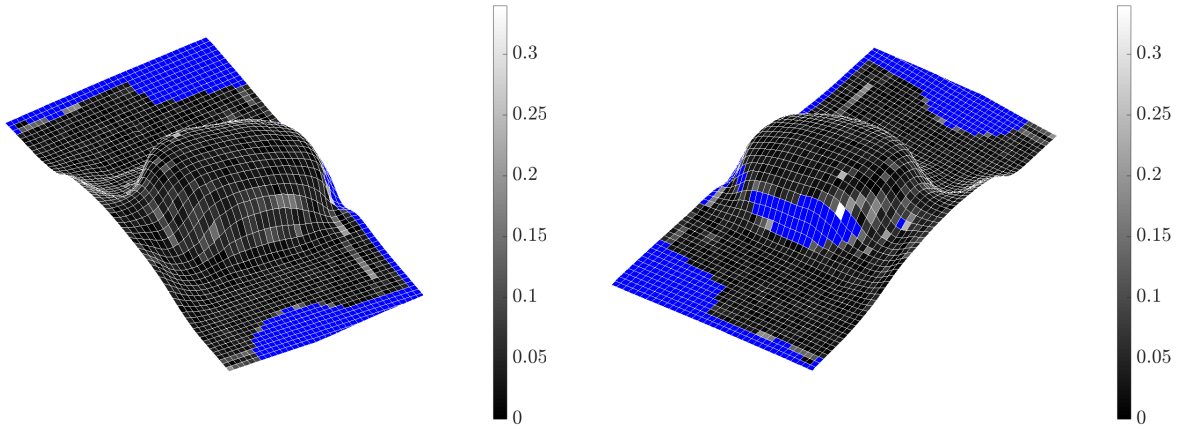
Figure 6.25: **Projection space algorithm** results for the **child surface**: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences. Results are shown for the *body projector*, which covers the majority of the child shell.



(a) Graph over feature scan samples.

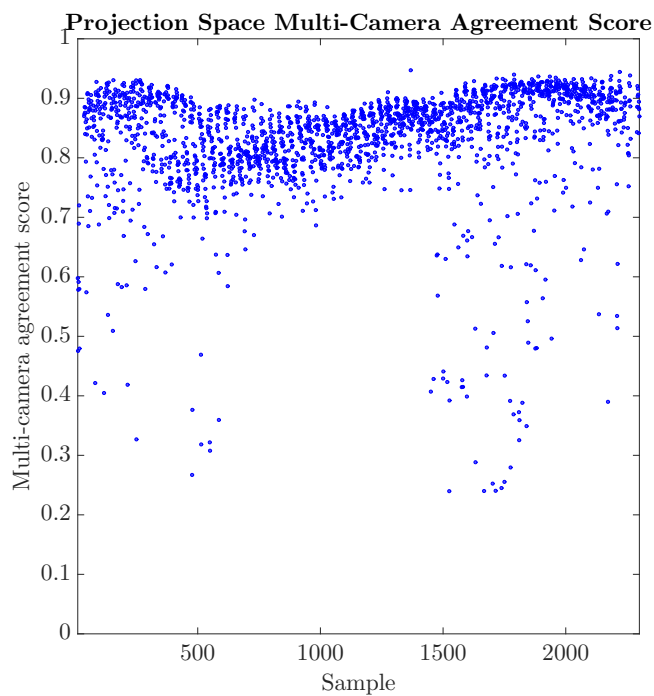


(b) Results shown in projection space.

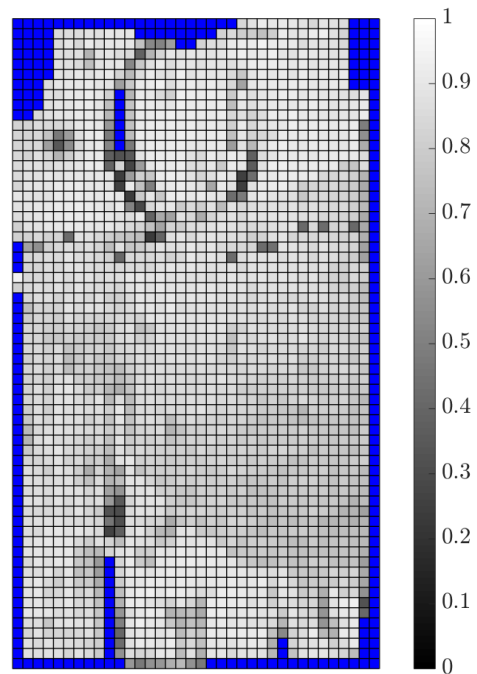


(c) Results shown in two views in 3D space on the child touch mesh S .

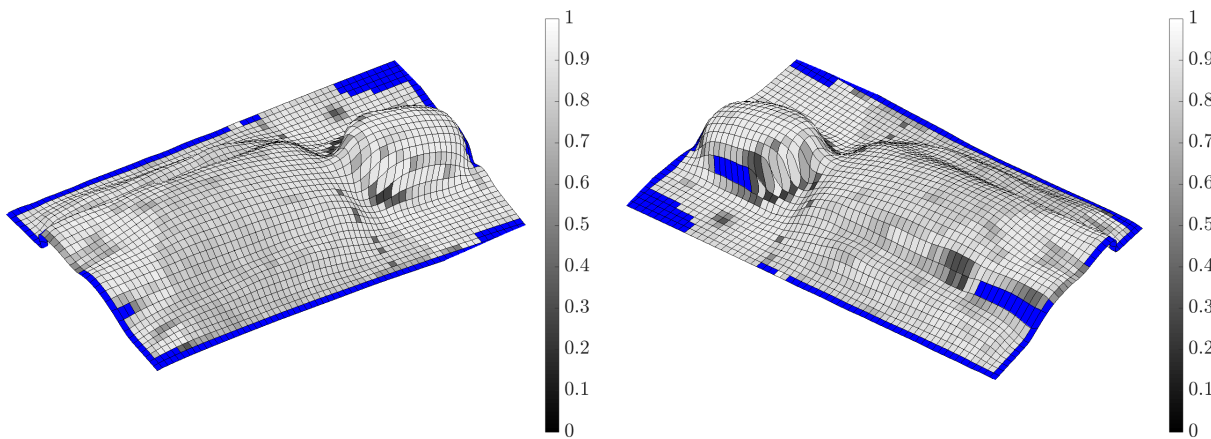
Figure 6.26: **Projection space algorithm** results for the **child surface**: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences. Results are shown for the *head projector*, which covers the head portion of the child shell.



(a) Graph over feature scan samples.

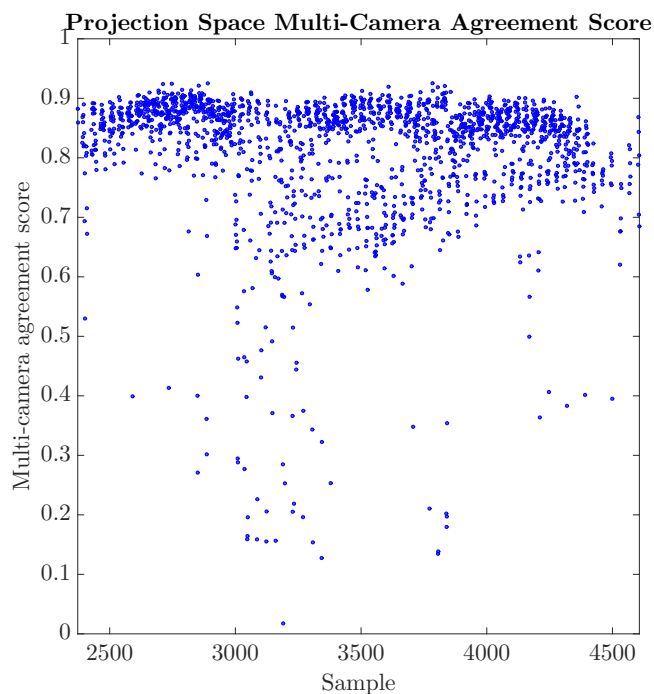


(b) Results shown in projection space.

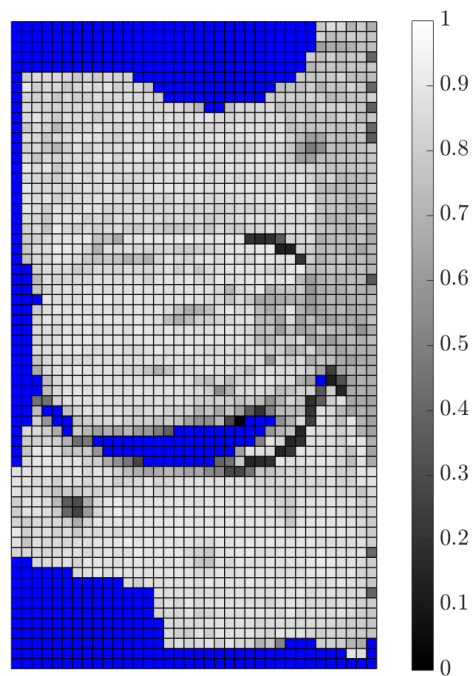


(c) Results shown in two views in 3D space on the child touch mesh S .

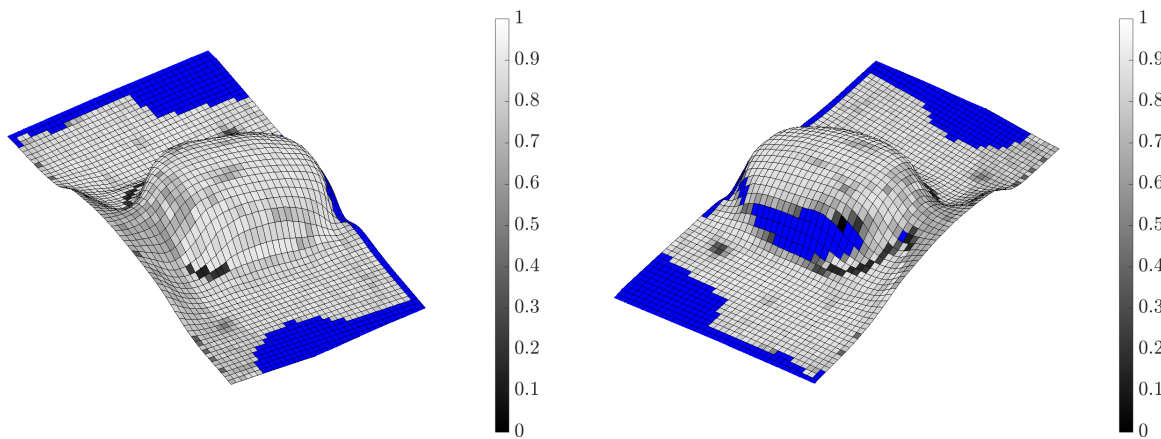
Figure 6.27: **Projection space algorithm** results for the **child surface**: multi-camera agreement score for **projected targets**. Results are shown for the *body projector*.



(a) Graph over feature scan samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the child touch mesh S .

Figure 6.28: **Projection space algorithm** results for the **child surface**: multi-camera agreement score for **projected targets**. Results are shown for the *head projector*.

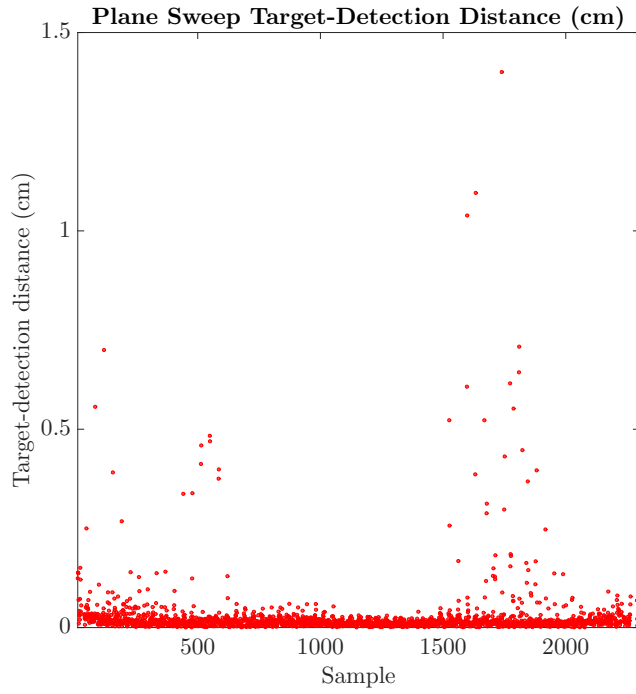
Furthermore, the projection space algorithm assigns lower confidence scores to targets projected on parts of one of the child's arms. In general, the targets with low confidence scores tend to have correspondingly higher target-detection distances.

6.4.1.2 Plane Sweep

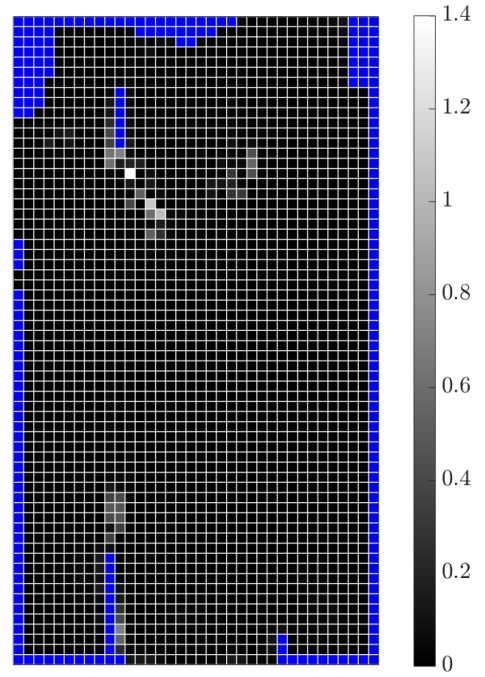
The plane sweep algorithm successfully classified 3910 of the feature scan samples displayed by the two projectors. On average, the entire pipeline operated in 9.52 milliseconds, and classification alone executed in 0.68 milliseconds.

Projected-target-detection distance: Plane sweep projected-target-detection distances are summarized in Figure 6.29 and Figure 6.30 for the body and head projectors of the child surface, respectively. The majority of the distances are close to 0, with an overall average of 0.0223 cm across the two projectors (Figure 6.29a and Figure 6.30a). As with the projection space algorithm results, there are a few targets with relatively high localization errors, generally corresponding to certain curved regions of the child surface such as the arm and head, as seen in the visualizations in projector coordinate space (Figure 6.29b and Figure 6.30b) and in 3D context on the touch mesh (Figure 6.29c and Figure 6.30c).

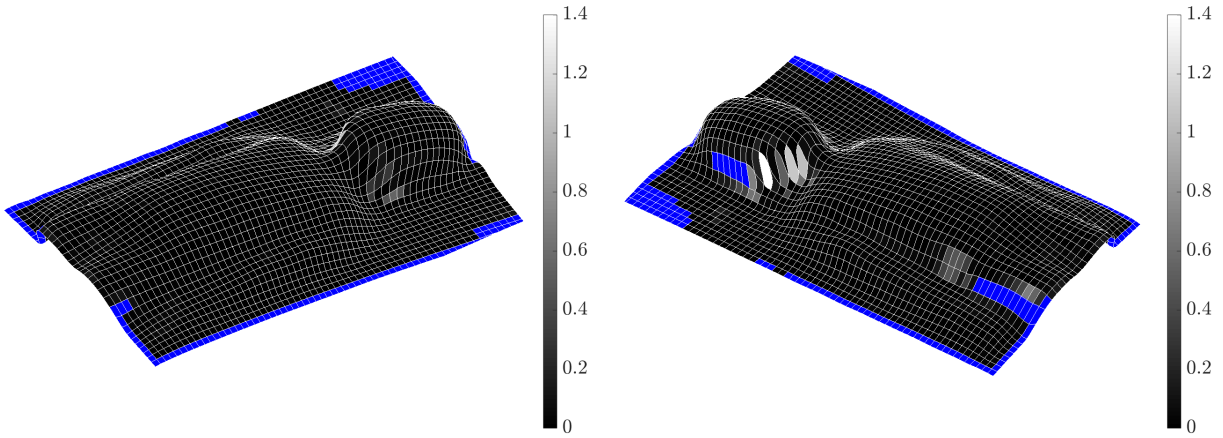
Projected target confidence score: Figure 6.31 and Figure 6.32 show the plane sweep confidence scores for the projected targets from the body and head projectors, respectively. Nearly all of the targets were assigned to the plane closest to the surface, indicating extremely high confidence of touch classifications (Figure 6.31a and Figure 6.32a). The remaining targets—assigned to planes within 0.2 cm of the surface and thus still classified with high confidence—are generally on the periphery of the shell or on the curved parts of the face. Figure 6.31b and Figure 6.32b show these results in the coordinate spaces of the two projectors, while Figure 6.31c and Figure 6.32c visualize them directly on the 3D child surface touch mesh.



(a) Graph over feature scan samples.

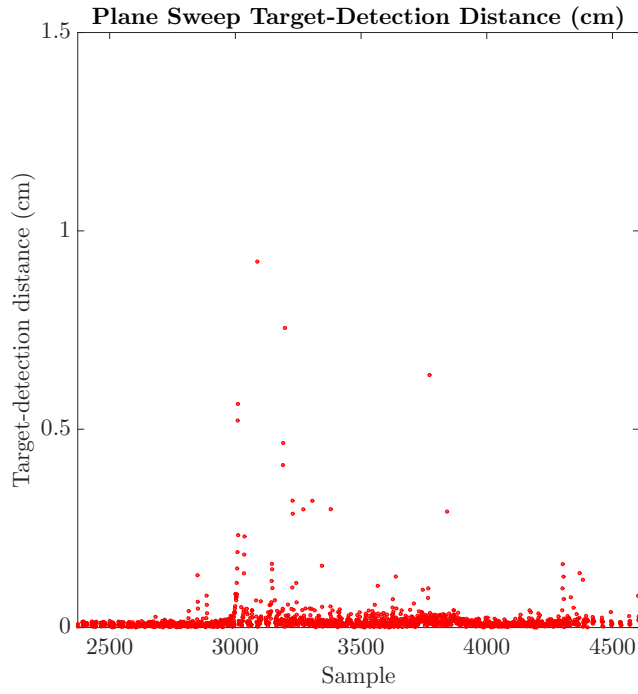


(b) Results shown in projection space.

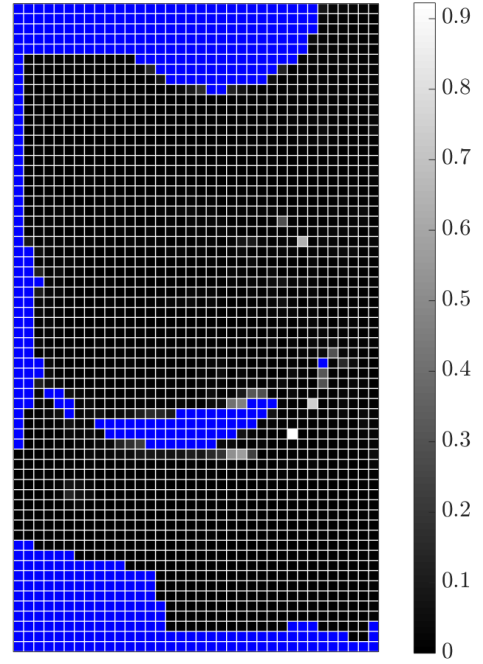


(c) Results shown in two views in 3D space on the child touch mesh S .

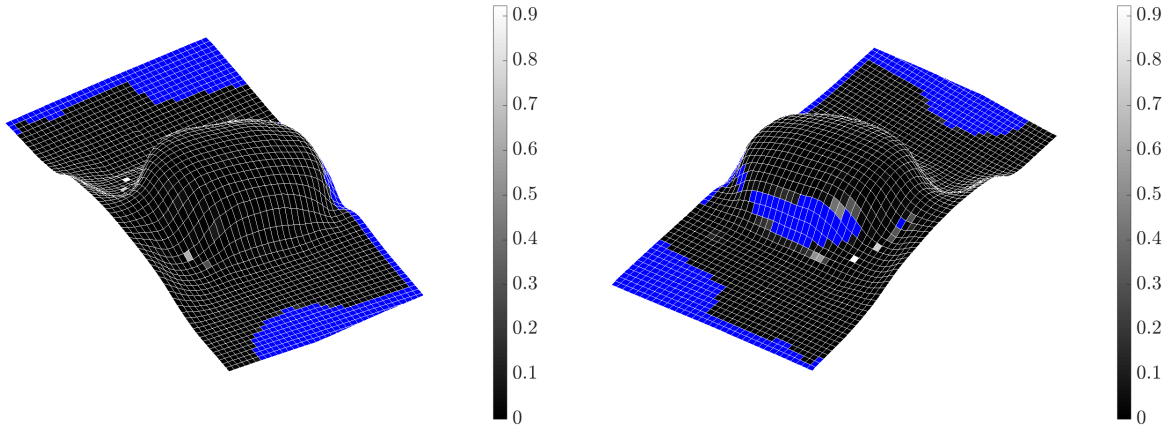
Figure 6.29: **Plane sweep algorithm** results for the **child surface**: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences. Results are shown for the *body projector*.



(a) Graph over feature scan samples.

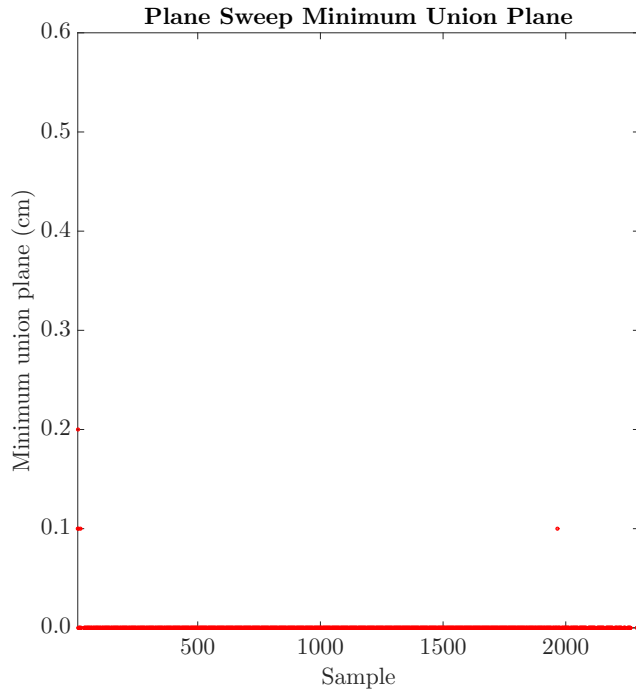


(b) Results shown in projection space.

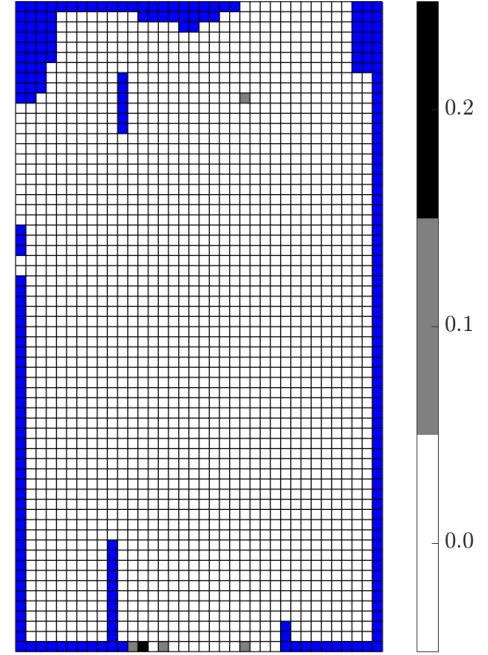


(c) Results shown in two views in 3D space on the child touch mesh S .

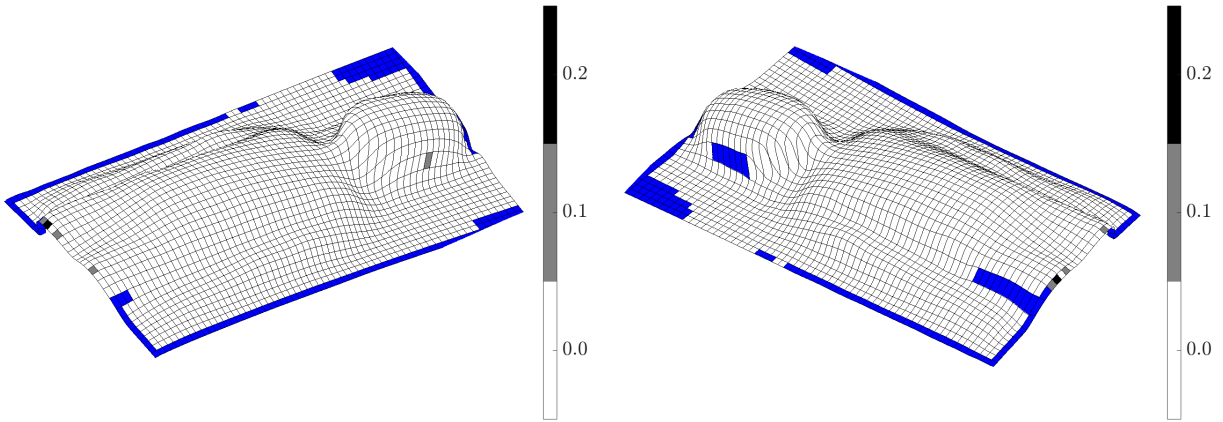
Figure 6.30: **Plane sweep algorithm** results for the **child surface**: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences. Results are shown for the *head projector*.



(a) Graph over feature scan samples.

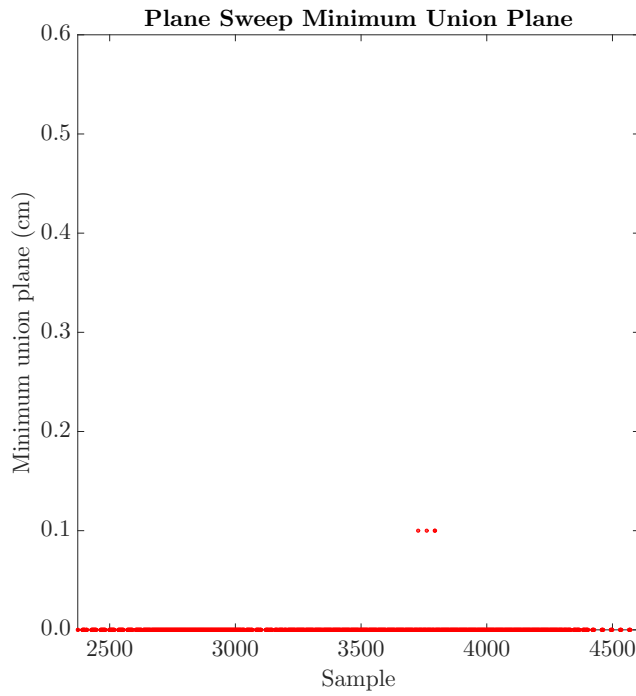


(b) Results shown in projection space.

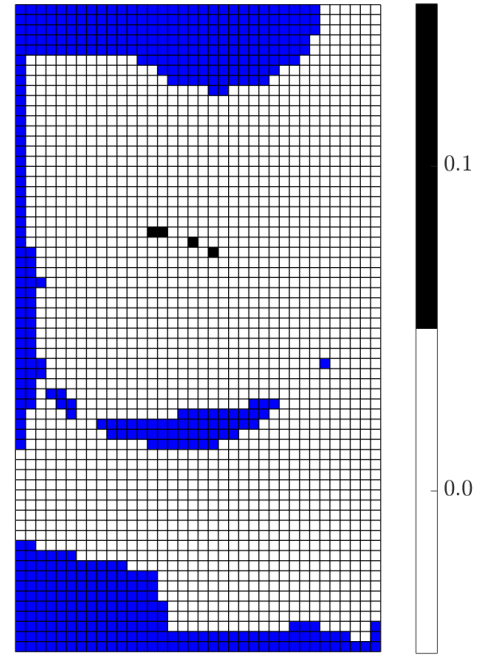


(c) Results shown in two views in 3D space on the child touch mesh S .

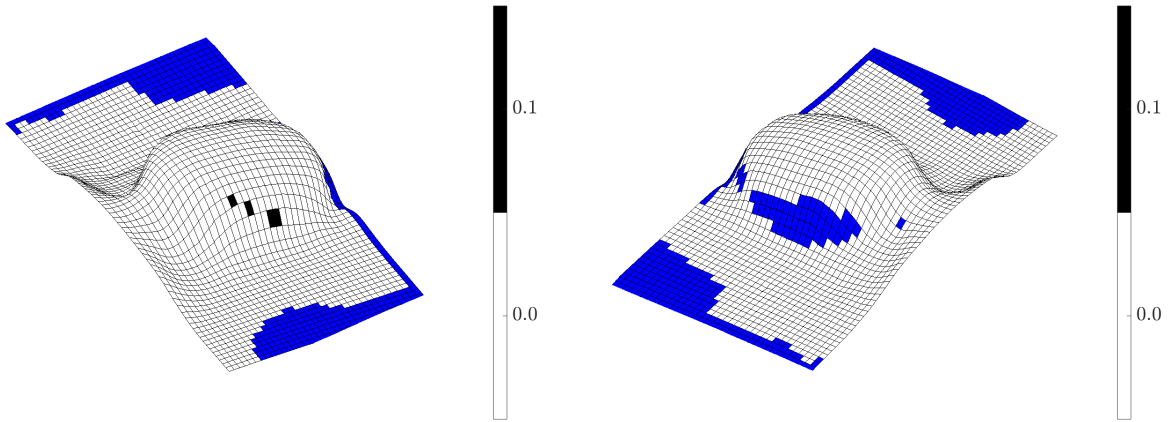
Figure 6.31: **Plane sweep algorithm** results for the **child surface**: minimum union plane (centimeters) for **projected targets**. Results are shown for the *body projector*.



(a) Graph over feature scan samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the child touch mesh S .

Figure 6.32: **Plane sweep algorithm** results for the **child surface**: minimum union plane (centimeters) for **projected targets**. Results are shown for the *head projector*.

Table 6.7: **Projector target** results summary for the **child surface**.

Algorithm	Successful	Localization	Confidence Score	Runtime
Projection space	3899	0.0473 cm	0.824	Detect: 5.99 ms Total: 20.41 ms
Plane sweep	3910	0.0223 cm	0 cm: 99.77% 0.1 cm: 0.20% 0.2 cm: 0.03%	Detect: 0.68 ms Total: 9.52 ms

6.4.1.3 Summary

Table 6.7 summarizes the results for the projected target evaluation on the child surface for the projection space and plane sweep algorithms. The projected-target-detection distances across the body and head projectors are shown in Figure 6.33.

As in many of the preceding results, the plane sweep algorithm outperformed the projection space algorithm by all of the considered metrics. The plane sweep algorithm successfully classified more projected targets, and the resulting target-detection distances are less than half of those produced by the projection space algorithm. In terms of confidence scores, the plane sweep classified almost 99.8% of the targets as touches with the highest possible confidence, whereas the projection space algorithm assigned relatively low classification scores to many of the targets.

Finally, the difference in execution time between the two algorithms is significant. As the child surface requires two projectors for image coverage, the runtime requirements for the projection space algorithm increase: potential camera contours must be converted to the coordinate spaces of both projectors, and the results from each projector must be subsequently merged. Likewise, both algorithms must consider seven total cameras on Prototype Rig II, compared to the four cameras for the preceding surfaces on Prototype Rig I. As a result, the projection space algorithm required 20.41 milliseconds for execution, with classification operating in 5.99 milliseconds. However, the

overall execution time of the plane sweep was only 9.52 milliseconds on average, with classification executing in 0.68 milliseconds.

Both algorithms experienced a few difficulties localizing targets on regions of high curvature on the child shell. Furthermore, some targets were missed due to insufficient camera coverage; only a single camera was capable of imaging targets in these regions, which does not provide the necessary 3D context for the two algorithms to localize the targets on the surface. These issues can be addressed through the use of additional cameras across the child surface.

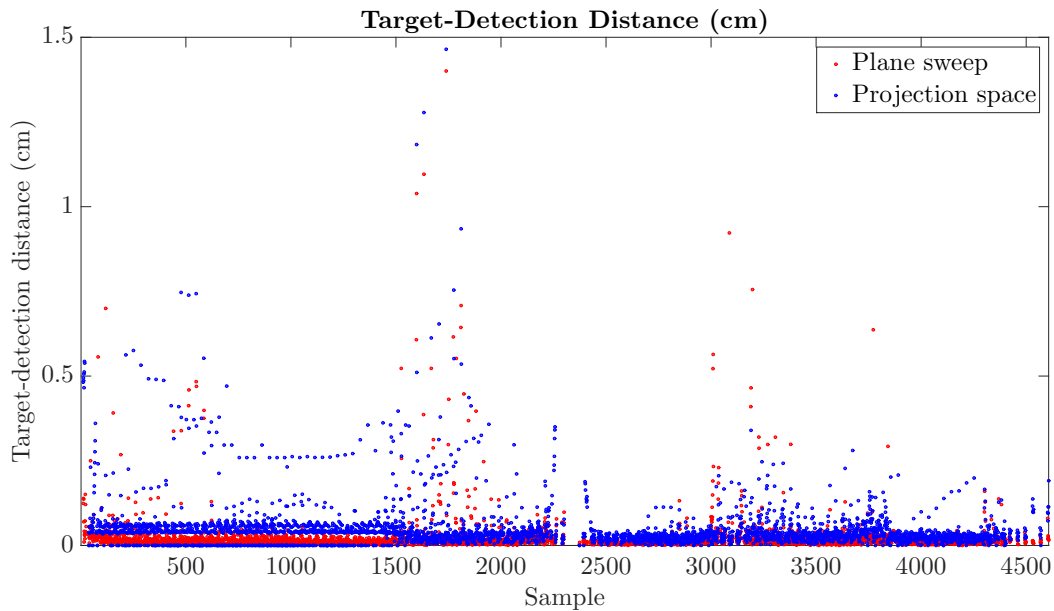


Figure 6.33: Comparison of **projection space (blue)** and **plane sweep (red)** algorithm results: distance (centimeters) between localized **projected targets** and the 3D lookup table correspondences for the **child surface**, shown across the samples of the feature scan over both the *body* and *head* projectors. On average, the plane sweep distances are roughly half the distances produced by the projection space algorithm.

6.4.2 Touch Targets

As described previously, touch sensing on the child surface on Prototype Rig II is supported by three touch sensing cameras. Four additional cameras that are unable to sense infrared light are used to aid in the calibration and surface reconstruction process. The three cameras used for touch detection are oriented toward the head of the surface, since the touch interactions we target are located there (as demonstrated in Section 5.4). Thus, the touch target results presented below arise from visual targets projected by only the head projector on Prototype Rig II, totaling around 240 captured touch samples.

6.4.2.1 Projection Space

Of the touch targets, the projection space algorithm successfully classified 239. On average, the algorithm required 15.87 milliseconds to completely process images, with detection alone operating in 5.26 milliseconds.

Touch-target-detection distance: Projection space algorithm touch-target-distance results are summarized in Figure 6.34. These results are similar to those obtained on the other surfaces, with an average distance of 0.1751 cm (Figure 6.34a). Some of the highest distances, on the order of 0.5 cm, accompany targets located on the periphery of the head of the child surface (Figure 6.34b and Figure 6.34c). As with the projected target results, some regions on the curved parts of the face are not successfully localized due to insufficient camera coverage.

Touch target confidence score: Confidence scores from the projection space algorithm for the touch targets are shown in Figure 6.35. The average multi-camera agreement score is 0.621 across the targets, though many are assigned confidences of less than 0.5 (Figure 6.35a). As seen in the visualizations in projector space and on the 3D touch mesh (Figure 6.35b and Figure 6.35c,

respectively), regions with low confidence detections generally correspond to the curved parts of the head of the child surface along with certain regions on the neck.

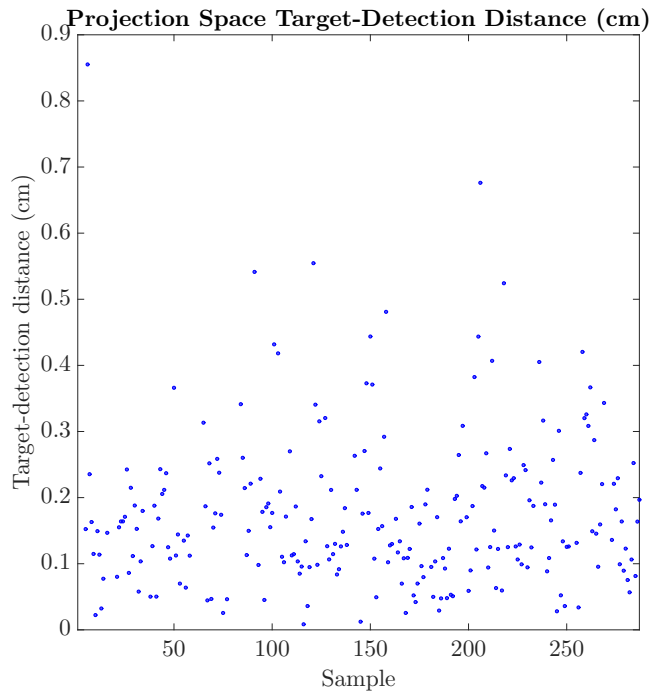
Touch/hover classification: Figure 6.36 shows the projection space algorithm touch/hover classification results for the child surface. In general, the hovers are classified with confidence scores ranging up to 0.4; however, many touches are assigned confidence scores in this range (Figure 6.36a). Overall touch/hover classification accuracy is plotted in Figure 6.36b, along with individual classification accuracy for touches and for hovers. The projection space algorithm misclassifies more of the touch and hover samples on the child surface than on the preceding ones. With a confidence score threshold of 0.4, it correctly classifies only 89.66% of the targets—84.94% of the touches and 97.32% of the hovers.

6.4.2.2 *Plane Sweep*

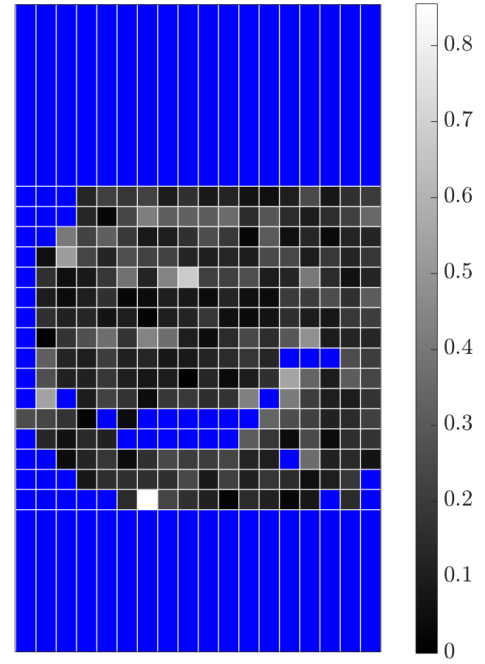
The plane sweep touch detection algorithm processed 238 of the touch targets on the child surface, executing in 10.53 milliseconds on average. Detection alone required 0.61 milliseconds.

Touch-target-detection distance: Plane sweep touch-target-detection distances are summarized in Figure 6.37. Compared to the projection space results, these distances occupy a smaller range, with an average of 0.1638 cm and a maximum of just over 0.6 cm (Figure 6.37a). However, the distances are lower for the periphery of the child surface head, with the highest errors generally clustered around the neck region. These distances are visualized in projection space in Figure 6.37b and textured on the 3D touch mesh in Figure 6.37c.

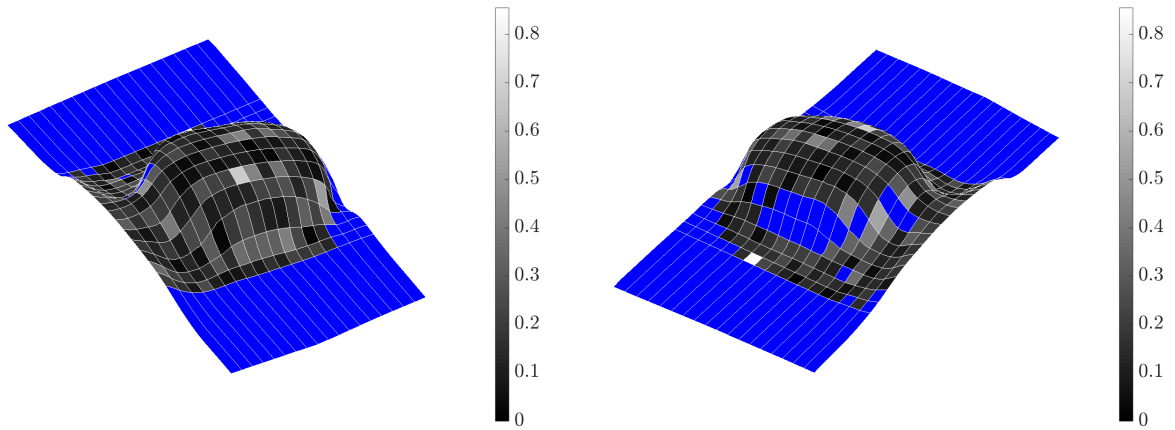
Touch target confidence score: Figure 6.38 presents the confidence scores assigned by the plane sweep algorithm to the set of touch targets on the child surface. Around 80% of these touch targets are assigned to the plane closest to the surface, indicating the highest confidence of a touch.



(a) Graph over touch samples.

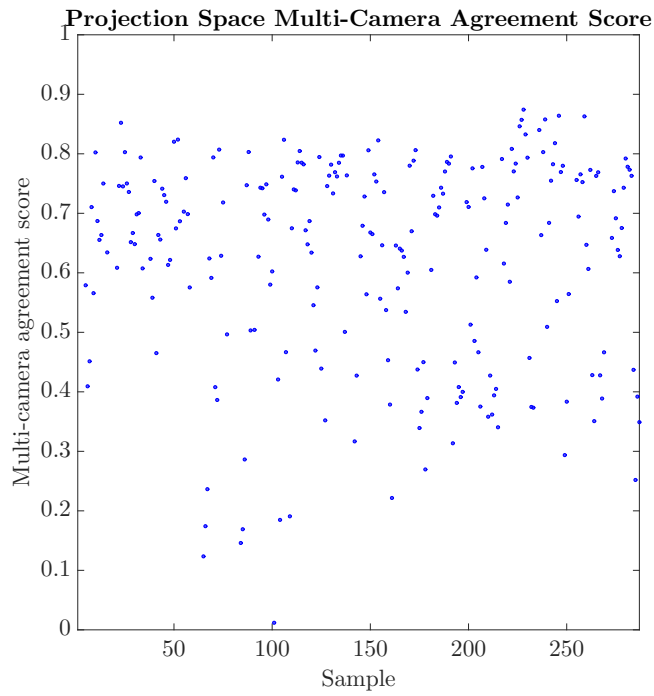


(b) Results shown in projection space.

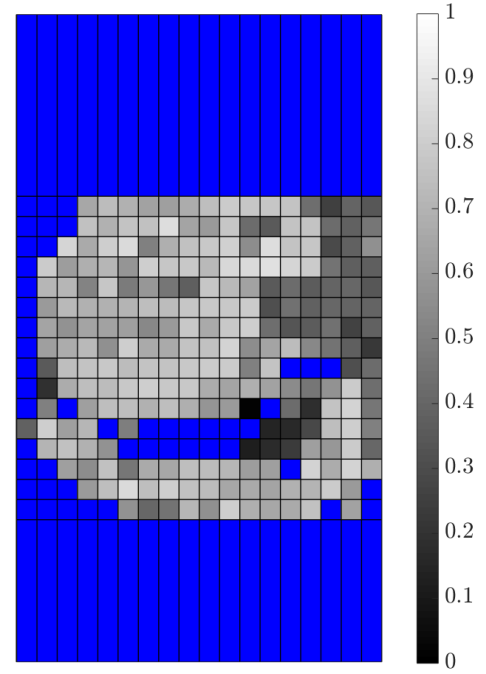


(c) Results shown in two views in 3D space on the child touch mesh S .

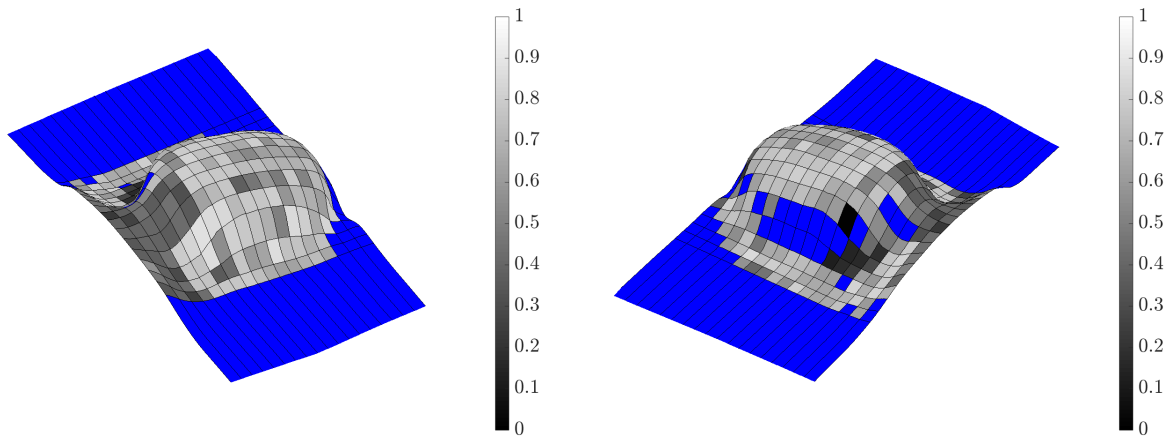
Figure 6.34: **Projection space algorithm** results for the **child surface**: distance (centimeters) between localized **touch targets** and the 3D lookup table correspondences.



(a) Graph over touch samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the child touch mesh S .

Figure 6.35: **Projection space algorithm** results for the **child surface**: multi-camera agreement score for **touch targets**.

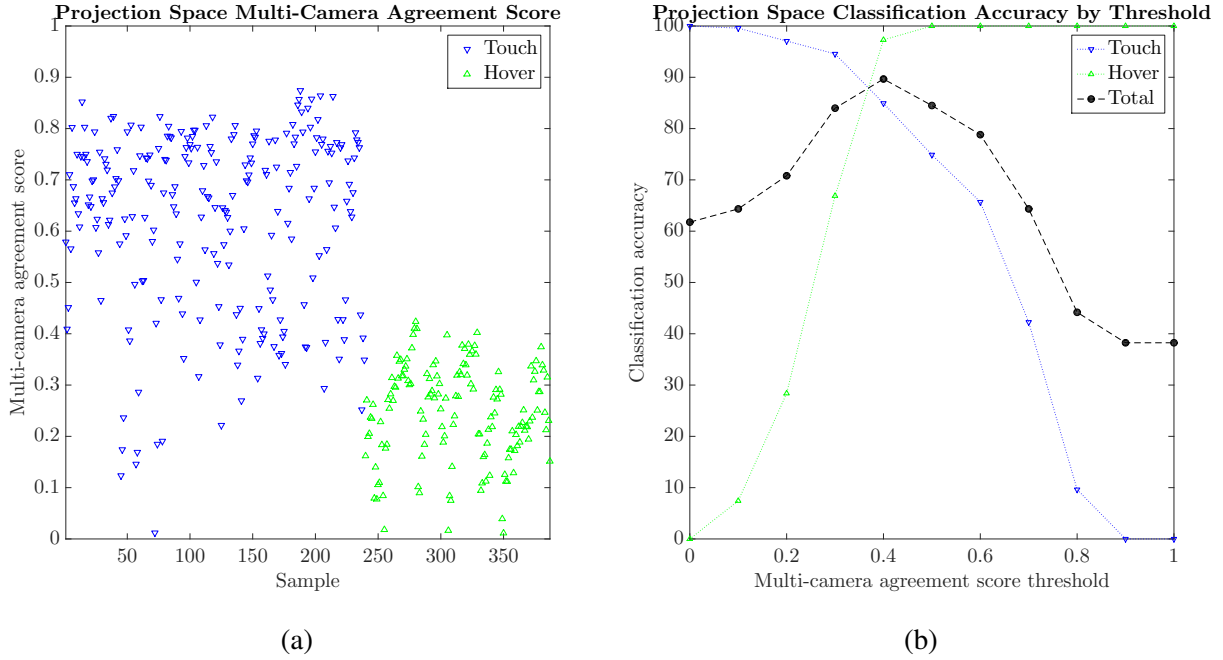
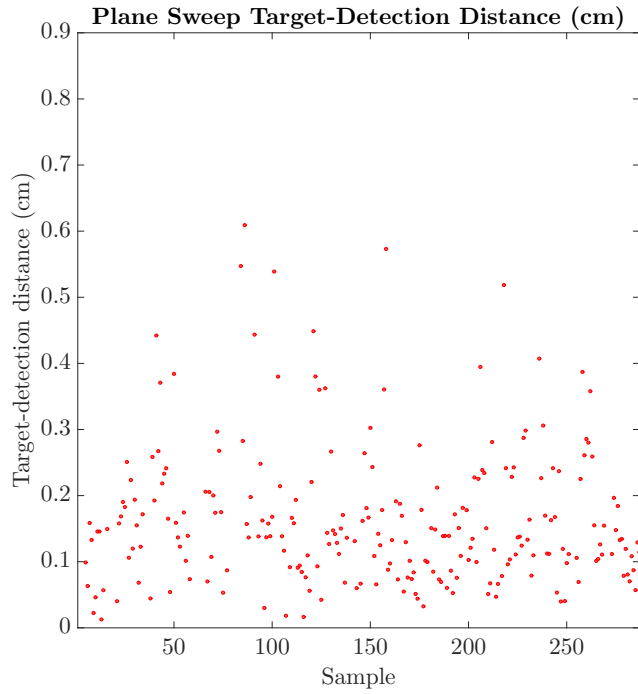


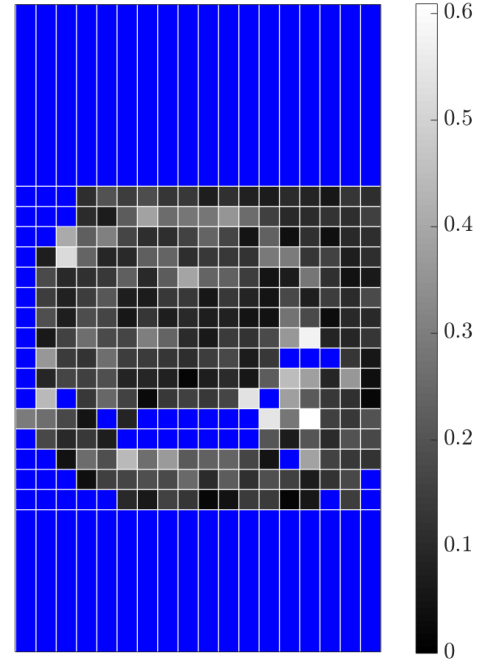
Figure 6.36: **Projection space algorithm** results for the **child surface: touch/hover classification**. (a) Confidence scores across touch and hover samples. (b) Overall touch/hover classification accuracy by confidence score threshold.

The remaining targets are assigned to planes within 0.2 cm of the surface, which similarly suggest high touch confidence (Figure 6.38a). Interestingly, the regions with the highest touch-target-detection distances are still strongly classified as touches. Instead, lower confidence is assigned to regions by the left eye of the child surface, as visualized in projection space (Figure 6.38b) and on the 3D mesh (Figure 6.38c).

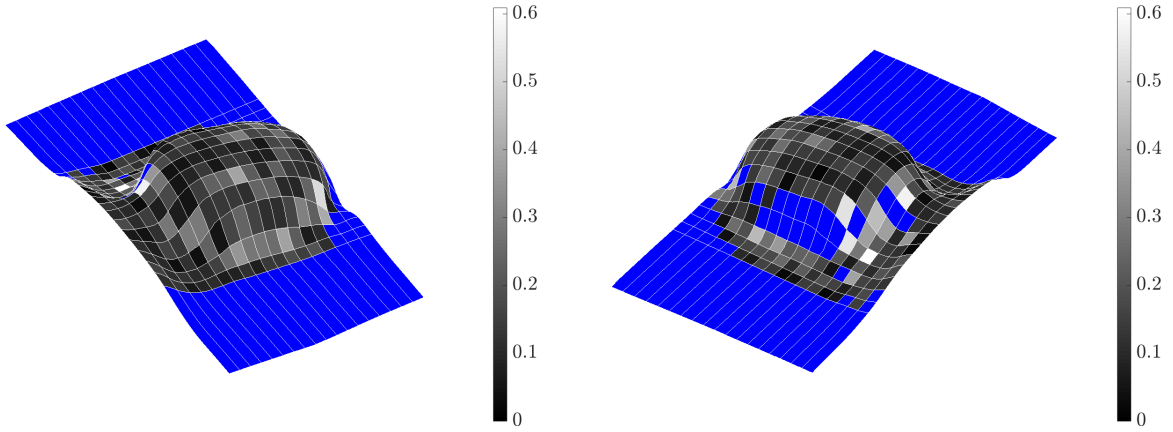
Touch/hover classification: Child surface touch/hover classification results for the plane sweep algorithm are plotted in Figure 6.39. While the touch targets are all classified as touches with high confidence—the plane with the minimum union area is close to the surface—several of the hovers are erroneously classified as touches (Figure 6.39a). However, in terms of classification accuracy, the plane sweep algorithm outperforms the projection space algorithm on the child surface.



(a) Graph over touch samples.

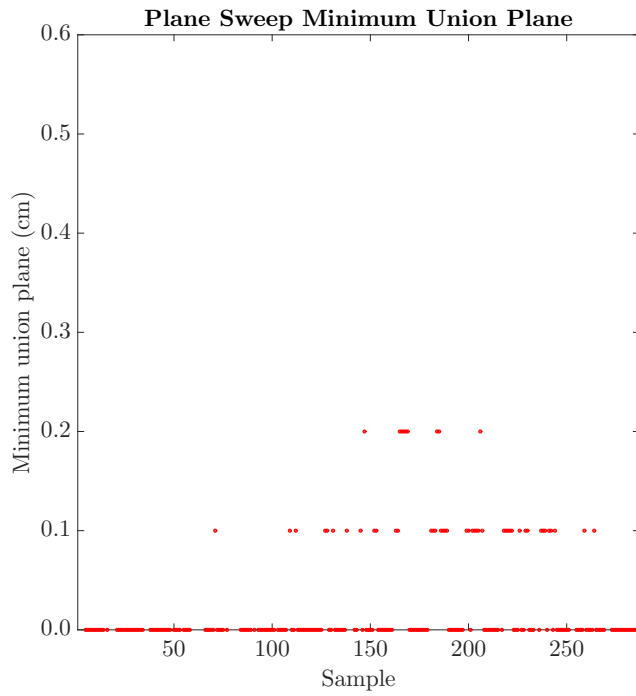


(b) Results shown in projection space.

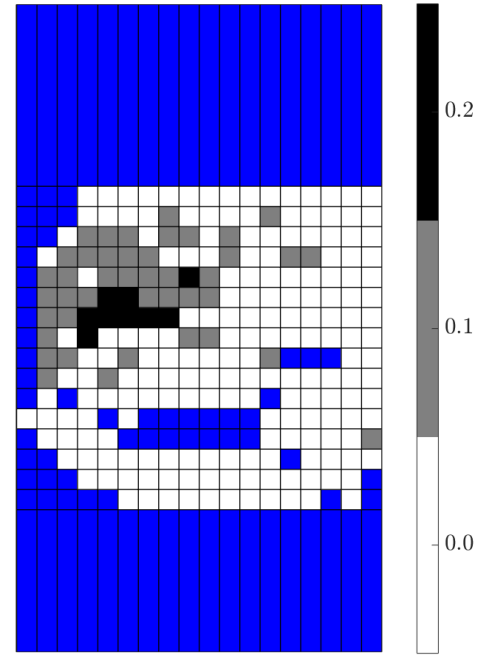


(c) Results shown in two views in 3D space on the child touch mesh S .

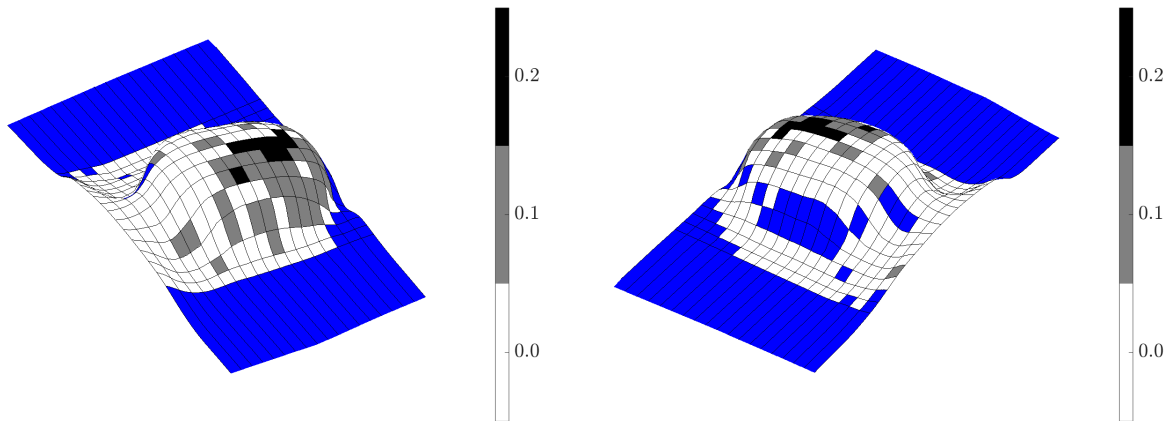
Figure 6.37: **Plane sweep algorithm** results for the **child surface**: distance (centimeters) between localized **touch targets** and the 3D lookup table correspondences.



(a) Graph over touch samples.



(b) Results shown in projection space.



(c) Results shown in two views in 3D space on the child touch mesh S .

Figure 6.38: **Plane sweep algorithm** results for the **child surface**: minimum union plane (centimeters) for **touch targets**.

Figure 6.39b presents the classification accuracy for the overall touch/hover dataset and for the individual touch and hover datasets using various thresholds for the minimum union plane. The plane sweep algorithm achieves its highest overall accuracy of 96.63% (100% of touches and 91.22% of hovers correctly classified) with a minimum union plane threshold of 0.2 cm.

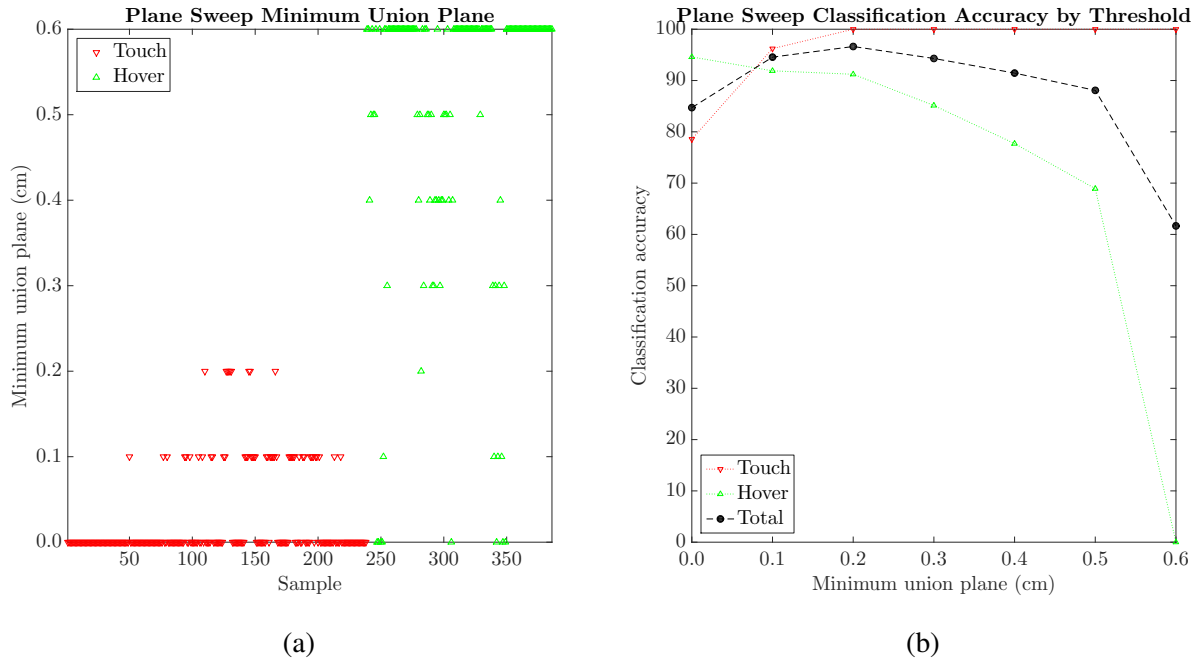


Figure 6.39: **Plane sweep algorithm** results for the **child surface: touch/hover classification**. (a) Confidence scores across touch and hover samples. (b) Overall touch/hover classification accuracy by confidence score threshold.

6.4.2.3 Summary

A summary of the touch target results on the child surface for both the projection space and planes sweep algorithms is shown in Table 6.8. Touch-target-detection distances are plotted in Figure 6.40. Finally, touch/hover classification results are summarized in Table 6.9.

Table 6.8: **Touch target** results summary for the **child surface**.

Algorithm	Successful	Localization	Confidence Score	Runtime
Projection space	239	0.1751 cm	0.621	Detect: 5.26 ms Total: 15.87 ms
Plane sweep	238	0.1638 cm	0 cm: 78.57% 0.1 cm: 17.65% 0.2 cm: 3.78%	Detect: 0.61 ms Total: 10.53 ms

The projection space and plane sweep algorithms correctly classified 239 and 238 of the touch targets, respectively. However, by the remaining metrics, the plane sweep algorithm achieved the best results. In particular, the plane sweep algorithm localized the touch targets with lower target-detection distances (0.1638 cm compared to 0.1751 cm) and classified the targets with higher touch confidence. As a result, it correctly classified around 96.6% of the touch and hover samples, while the projection space algorithm achieved an overall accuracy of about 89.7%. Additionally, the plane sweep algorithm executed significantly faster than the projection space algorithm. Overall, the plane sweep processed targets in 10.53 milliseconds on average (compared to 15.87 milliseconds for the projection space algorithm). The difference is even more significant when we consider the execution time specific to detection alone: 0.61 milliseconds for the plane sweep algorithm and 5.26 milliseconds for the projection space algorithm.

Table 6.9: **Touch/hover classification** results summary for the **child surface**.

Algorithm	Best Accuracy	Score Threshold
Projection space	89.66% (84.94% touch, 97.32% hover)	0.4
Plane sweep	96.63% (100% touch, 91.22% hover)	0.2 cm

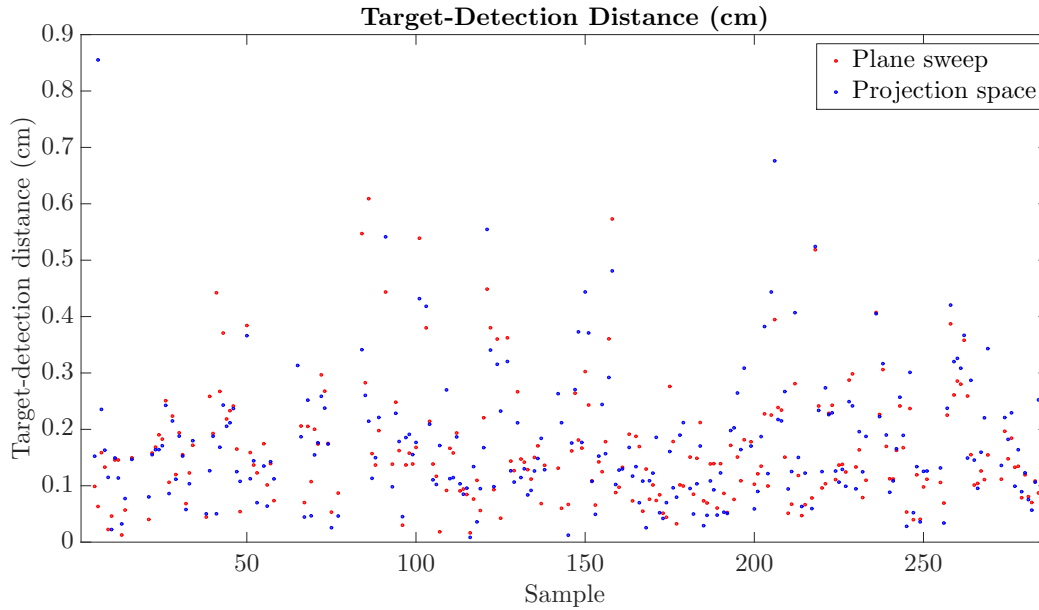


Figure 6.40: Comparison of **projection space** (blue) and **plane sweep** (red) algorithm results: distance (centimeters) between localized **touch targets** and the 3D lookup table correspondences for the **child surface**, shown across the set of touch samples. Both algorithms perform about the same on the touch samples, with the plane sweep algorithm achieving slightly lower distances.

6.5 Algorithm Comparison

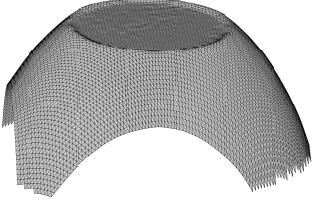
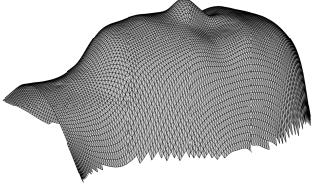
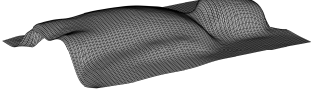
Table 6.10 compares the results achieved by the projection space and plane sweep algorithms on the bowl, head, and child surfaces. With the exception of the touch targets on the bowl surface, the plane sweep target-detection distances were lower than those from the projection space algorithm. In particular, the plane sweep projected-target-detection distances were generally half those of the projection space algorithm, indicating better modeling of the observed camera-projector correspondences from the feature scan in the preprocessing phase. Both algorithms localized detected user touches for all surfaces within less than 2 mm from the intended targets on average.

Touch/hover classification results were comparable for both algorithms on the bowl surface, with both classifying nearly all samples correctly. However, the plane sweep algorithm correctly clas-

sified more of the touch and hover targets on the head and child surfaces than the projection space algorithm. This is perhaps due to the more direct utilization of three-dimensional context by the plane sweep algorithm as compared to the projection space algorithm, which collapses images of a potential touch or hover event into one two-dimensional coordinate space per projector. The child surface features the most significant difference in the results: the plane sweep correctly classified approximately 96.6% of the touch and hover targets compared to only 89.7% for the projection space algorithm. The larger size and optical properties of the child surface contribute to these classification difficulties for both algorithms, specifically due to challenges in segmenting camera imagery into candidate event contours.

Finally, the plane sweep algorithm outperformed the projection space algorithm in terms of execution time across all evaluations. For the head and bowl surfaces, the plane sweep classified and localized targets five times faster than the projection space algorithm. The difference is more pronounced for the child surface. While both algorithms are impacted by the seven cameras of Prototype Rig II—compared to only four for the bowl and head surfaces on Prototype Rig I—the two projectors used for the child surface require additional computation for the projection space algorithm, which must now convert camera imagery to two projectors and merge results. However, the plane sweep algorithm operates independently of the number of projectors. As a result, the projection space algorithm execution time increases significantly for the child surface compared to the bowl and head surfaces, while the plane sweep algorithm execution time increases only a small amount. Still, it is important to note that both algorithms achieved real-time performance on all surfaces, processing camera imagery faster than it was captured.

Table 6.10: Summary of results for the projection space and plane sweep touch detection algorithms, evaluated on the bowl, head, and child surfaces. For projector and touch targets, we provide target-detection distances and overall execution times, with the execution time specific to the detection routine provided in parentheses. For touch/hover classification, we include the best overall accuracy and the confidence score threshold used to achieve it.

Surface	Data	Projection Space	Plane Sweep
 Bowl	Projector	0.0226 cm 8.85 (2.79) ms	0.0088 cm 6.37 (0.50) ms
	Touch	0.1311 cm 9.90 (2.83) ms	0.1342 cm 7.63 (0.63) ms
	Classification	Total: 99.60% Touch: 99.66% Hover: 99.50% Threshold: 0.3	Total: 99.80% Touch: 100% Hover: 99.50% Threshold: 0.4 cm
 Head	Projector	0.0256 cm 9.43 (3.38) ms	0.0132 cm 6.71 (0.60) ms
	Touch	0.1623 cm 10.53 (3.31) ms	0.1560 cm 7.81 (0.74) ms
	Classification	Total: 96.92% Touch: 96.66% Hover: 97.52% Threshold: 0.4	Total: 99.43% Touch: 99.44% Hover: 99.39% Threshold: 0.2 cm
 Child	Projector	0.0473 cm 20.41 (5.99) ms	0.0223 cm 9.52 (0.68) ms
	Touch	0.1751 cm 15.87 (5.26) ms	0.1638 cm 10.53 (0.61) ms
	Classification	Total: 89.66% Touch: 84.94% Hover: 97.32% Threshold: 0.4	Total: 96.63% Touch: 100% Hover: 91.22% Threshold: 0.2 cm

CHAPTER 7: USER STUDIES

In this chapter, we describe controlled user studies examining various aspects about our proposed touch sensing methodology. First, we cover a formative study designed to obtain qualitative feedback on the usefulness of the paradigm in a healthcare training scenario involving nursing student assessments of a touch-sensitive physical-virtual patient [33, 57, 146]. The second study concerns the effects of mismatches between the physical and visual perceptions of a touch interface—for instance, when virtual content is displayed on a physical surface with mismatched geometry [70].

7.1 Formative Study: Stroke Patient Simulation

This section replicates portions of the following published works:

- “Preliminary assessment of neurologic symptomatology using an interactive physical-virtual head with touch,” by Salam Daher, Laura Gonzalez, and Gregory Welch, presented at the International Meeting on Simulation in Healthcare (IMSH) 2016 [33].
- “Student nursing assessment of discrete neurology symptoms using an interactive physical head,” by Laura Gonzalez, Salam Daher, Jason Hochreiter, and Gregory Welch, presented at the International Nursing Association for Clinical Simulation and Learning (INACSL) 2016 [57].
- “Interactive rear-projection physical-virtual patient simulators,” by Gregory Welch, Salam Daher, Jason Hochreiter, and Laura Gonzalez, presented at Medicine Meets Virtual Reality (NextMed/MMVR) 2016 [146].

Early detection and treatment of stroke is of paramount importance to reduce damage to the brain, as certain treatments lose effectiveness within a few hours following the onset of symptoms [121]. As a tool to aid the early recognition of stroke, the American Stroke Association (ASA) suggests the mnemonic FAST [134]. The letters stand for the following warning signs:

- **F:** face drooping. A person experiencing a stroke may exhibit drooping on one side of the face, and they may be unable to perceive touch on the affected side. The ASA recommends checking for asymmetry in the person's smile.
- **A:** arm weakness. The person may experience weakness or numbness in one of the arms and have difficulty raising it.
- **S:** speech difficulty. This often presents as slurred speech or difficulty repeating simple phrases.
- **T:** time to call 9-1-1. Someone who exhibits these symptoms may be experiencing a stroke. The ASA recommends immediately calling emergency services and keeping track of when the person first began showing symptoms.

In addition to the mnemonic, the ASA lists several other potential warning signs of stroke. These include difficulty understanding speech, one-sided numbness, vision problems, dizziness, and sudden severe headache.

Stroke recognition training generally involves the use of task trainers and standardized patients. Standardized patients are capable of exhibiting some of the symptoms of stroke, such as slurred speech and decreased touch sensation, but they are not able to realistically simulate signs such as facial droop [82] and lid lag. Simulated stroke patients are not commonly used in healthcare training due to their inability to accurately portray certain symptoms, such as localized weakness, perhaps resulting from a general lack of simulators designed specifically for stroke assessment [55].

As many of the signs and symptoms of stroke are related to the face—for example, involving appearance, vocals, pain expression, and touch perception—our physical-virtual patient head provides a suitable platform for stroke simulation. We created virtual content corresponding to visual, auditory, and touch-related symptoms of a stroke and their complementary representations in a healthy patient, which we examined in a preliminary user study [33, 57, 146]. The content, described in more detail in Chapter 4, included patient dialog, with slurred speech for the stroke patient; various animations, including facial expressions and manipulations of the patient’s eyes and lips; and audio responses that indicated the healthy patient’s ability and the stroke patient’s inability to sense touch in certain locations. In a formative study, nursing students assessed this content across two conditions:

1. The *PVHT* condition features the *physical-virtual head* surface with automated *touch* sensing as described in Chapter 3. The head surface is situated near a simulated mannequin body.
2. The *MV* condition features a standard physical *mannequin* augmented with *virtual* imagery from a nearby computer monitor.

Both conditions presented the same scenario with signs and symptoms suggesting a stroke. In the PVHT condition, the virtual imagery was presented directly on the head surface, which was further capable of touch interactions (as shown in Section 5.3). This included the ability to examine the patient’s eyes, mouth, and sensation of touch. Meanwhile, in the MV condition, the virtual imagery was instead shown on a separate computer monitor. In both conditions, simulated vital signs were also displayed on the monitor. The two conditions are shown in Figure 7.1.

This study featured a between-subjects design. A total of 23 nursing students assessed the simulated patient presented by one of these two conditions. We observed their ability to recognize the

asymmetry of facial expressions, their assessment of the patient's perception of touch, the specific diagnostic questions they asked, and their evaluation of the patient's speech. After their assessments, students were prompted to answer the following free-response question: "Please comment on anything that would help us better understand your interaction with the patient—you can touch upon any topic you like, such as where you think such virtual humans could be beneficial, what they should look like, how they should behave, and so on."

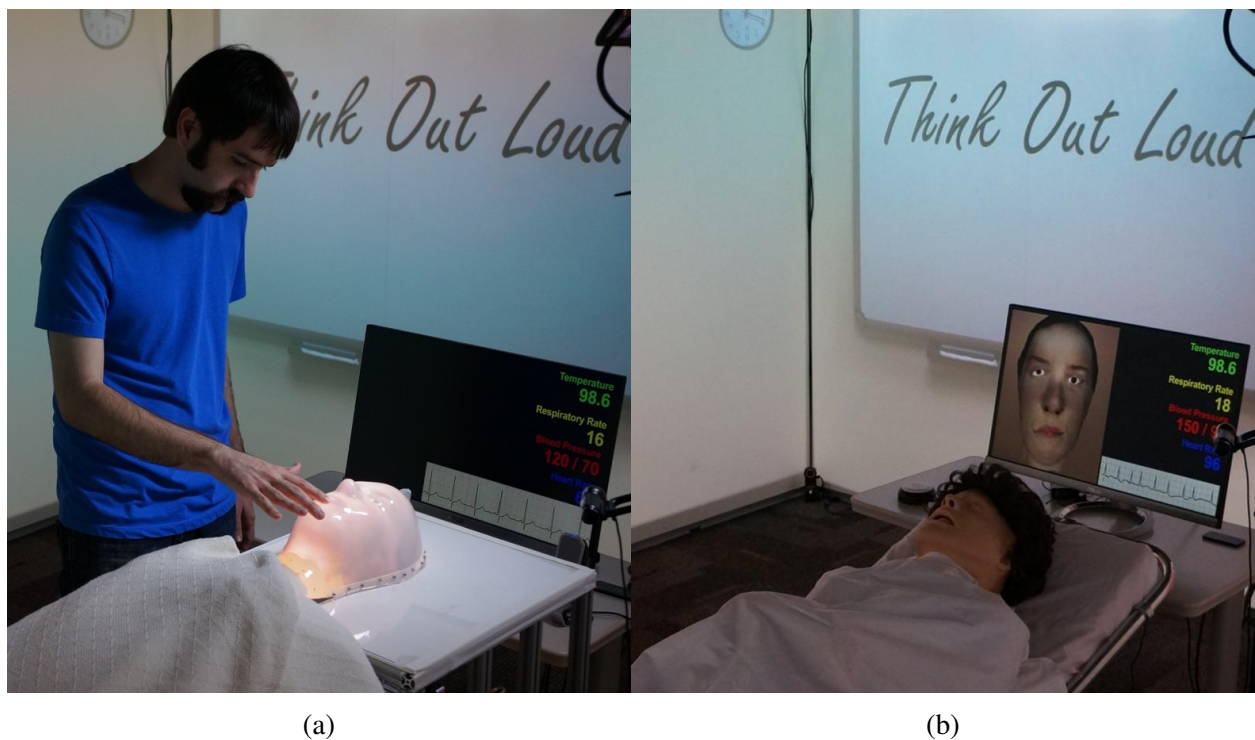


Figure 7.1: The two experimental conditions of the stroke study. (a) PVHT condition, with the physical-virtual touch-sensitive head surface. (b) MV condition, with a physical mannequin and virtual imagery on a separate display.

In subjective questionnaire results, the students who participated in the PVHT condition commented on the value of the physicality of the head, the realism of the voice in terms of clinical presentation, and the ability of the patient to communicate with them. Many expressed appre-

ciation for the touch interaction capabilities and noted that the verbal and non-verbal cues were tightly linked; in particular, one commented that this synchrony allowed her to perform most of the standard neurological and head assessment tasks she learned to perform. Several participants praised the realism of this presentation, noting that it supported treatment of the simulated patient “as if she were a real person” and that it was “easier to work with and much more realistic than any of the [other] mannequins.” A common sentiment among participants was the desire to extend the technology to a full-body system to allow for the simulation of symptoms and direct touch interaction for conditions that are not isolated to the head.

The nursing students who participated in the MV condition described the separate visual display as being a “great bonus” over standard mannequins with static appearances. In general, the participants rated the realism of the visuals highly, including simulated symptoms specific to stroke (such as right-sided sagging). Compared to the PVHT condition, the MV was perceived to be less able to communicate freely—that is, not speaking unless directly prompted. However, when asked “what would you have done differently if [the patient you assessed] was a human patient instead of a simulator?” more of the MV participants indicated that they would have called for a doctor or the nurse in charge of the healthcare facility, whereas the PVHT participants tended to focus on tests they would have performed on the rest of the patient’s body, if it were present.

7.2 Physical-Virtual Mismatches

When users interact with 3D content via touch input, they are affected by their physical and visual perceptions of the interface. The level to which the geometry of the touch interface matches the geometry of the displayed content can vary. Additionally, the source of virtual content can have performance implications. In this study, we examined how mismatches between physical and visual perceptions can affect cognitive load and performance in an AR touch task [70]. Across

four conditions, our study design varied the degree of *physical fidelity*—how well the physical and virtual content matched—and the *visual mechanism* used to display virtual imagery (Figure 7.2). Touch performance, cognitive load, usability, and subjective preferences were all influenced by the physical interface used to interact with the virtual content and the source of the imagery.



Figure 7.2: Participants interacted with four representations of a 3D virtual head that varied in physical form and visual display.

This section substantially replicates a peer-reviewed paper, “Cognitive and touch performance effects of mismatched 3D physical and visual perceptions,” published in the proceedings of IEEE Virtual Reality 2018, by Jason Hochreiter, Salam Daher, Gerd Bruder, and Greg Welch [70]. In the context of this dissertation research, the presented study focuses on touch interactions on the

physical-virtual plane and head surfaces on Prototype Rig I described in Chapter 4, facilitated by the projection space touch detection method described in Section 3.2.2.1. An additional interaction paradigm involving free space touching, separate from this algorithm, was specifically created to support one of the study conditions.

7.2.1 Introduction

This study concerned an AR touch task on a virtual human head model displayed to users in various physical-virtual representations. Participants were tasked with accurately touching specific targets on the virtual head across these representations; in one study phase, they had limited time with which to select from one of three targets based on size while completing a concurrent counting task. As touch performance metrics, we considered accuracy, response time, target selection, usability, cognitive load, and various subjective ratings. Our four physical-virtual representations considered two dimensions, each with two levels:

1. The *physical fidelity* of the display surface as a physical touch interface for the virtual content—either matching or non-matching
2. The *visual mechanism* by which the virtual content was displayed to users—either through projector-based spatial augmented reality (SAR) or an AR head-mounted display (HMD)

The experiment thus featured four corresponding conditions:

1. **SAR Head:** rear-projection spatial augmented reality (SAR) imagery displayed on a matching physical head-shaped surface
2. **SAR Plane:** rear-projection SAR imagery displayed on a flat surface

3. **HMD Head:** HMD-based AR imagery registered to a matching physical head-shaped surface
4. **HMD Hologram:** HMD-based AR imagery with no physical surface present

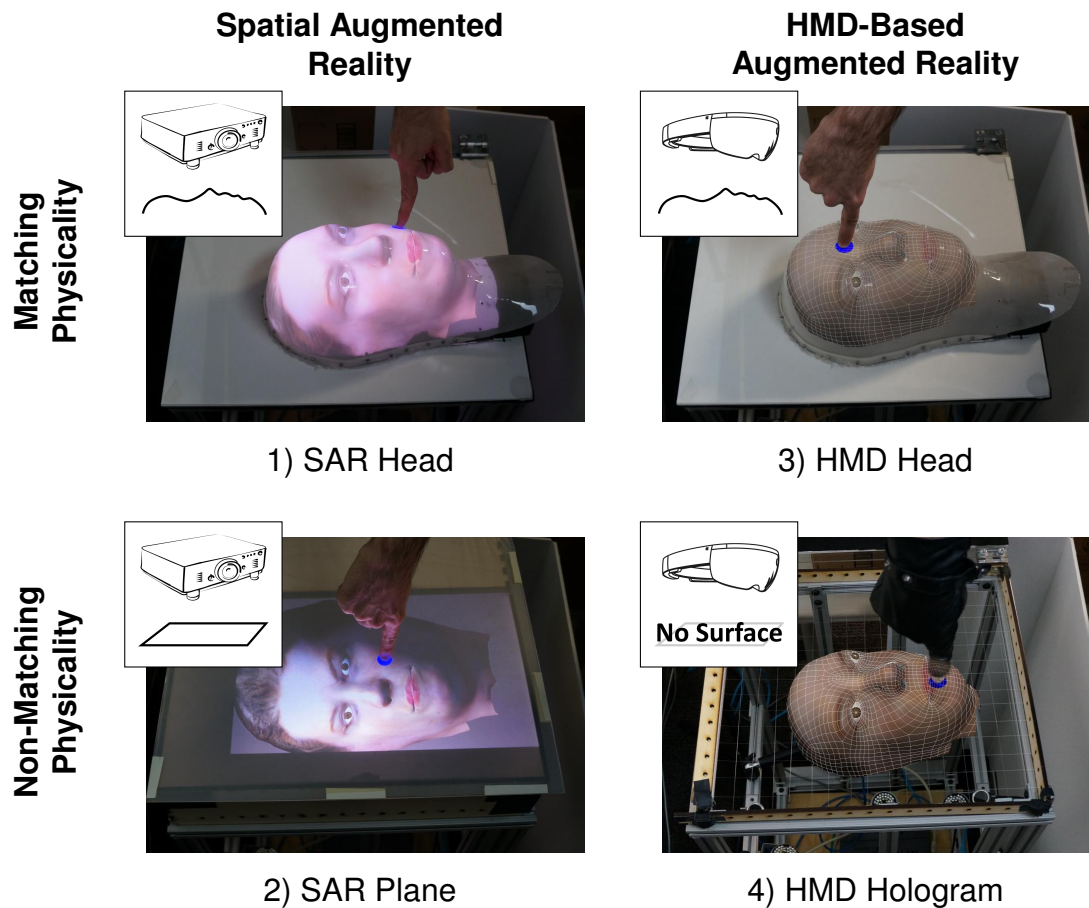


Figure 7.3: The four study conditions each differed in their physical and virtual representation of a 3D human head model. Across one dimension, the physical object with which users interacted either matched or did not match the geometry of the virtual object. Separately, the virtual object was displayed either using a projector or an HMD. For the HMD conditions, the imagery above is simulated.

These conditions are summarized in Figure 7.3. For the conditions involving physical touch surfaces—SAR Head, SAR Plane, and HMD Head—participants interacted with the virtual content directly by touch, using the touch sensing methodology described in Chapter 3. In the HMD

Hologram condition, no physical surface was present: participants instead placed their finger at the desired location and then indicated a “touch” by pressing a button on a separate input device (Figure 7.4). This approach, involving the localization of the user’s finger in midair, was created specifically for this study.

When designing these four conditions, we had three primary goals. First, we aimed to create reasonable, ecologically valid interfaces that reflect how these paradigms are used in AR touch tasks in practice. For example, our decision to use an external input device in the HMD Hologram condition follows from the typical inclusion of such interactive devices with current HMDs. This may increase the cognitive load experienced by users, but such load is inherent to these interfaces, and it is this load we wished to measure. Furthermore, we wanted to ensure consistency across the four conditions. While HMD-based free-space interaction methods could instead allow users to indicate a touch by leaving their fingers in place for several seconds or performing a gesture, this would lead to inconsistencies in response time measures. Finally, to specifically investigate the impacts of physical-virtual mismatches, we designed our conditions so that participant interactions followed naturally from their physical and visual perceptions of each representation. This includes limitations that are imposed by the conditions; for instance, visually searching for targets on a 3D surface may require examining it from multiple viewpoints, which is not the case for planar surfaces. We intentionally omitted visual, audio, or other aids for target search and selection, which would interfere with the perceptions of participants and could induce additional unintended cognitive load.

Here, we briefly summarize the major findings of the study:

- Touch performance was highest for the two SAR conditions. Touch errors, computed as the distance between displayed targets and the locations of participant touches, were comparable for the SAR Head and SAR Plane.

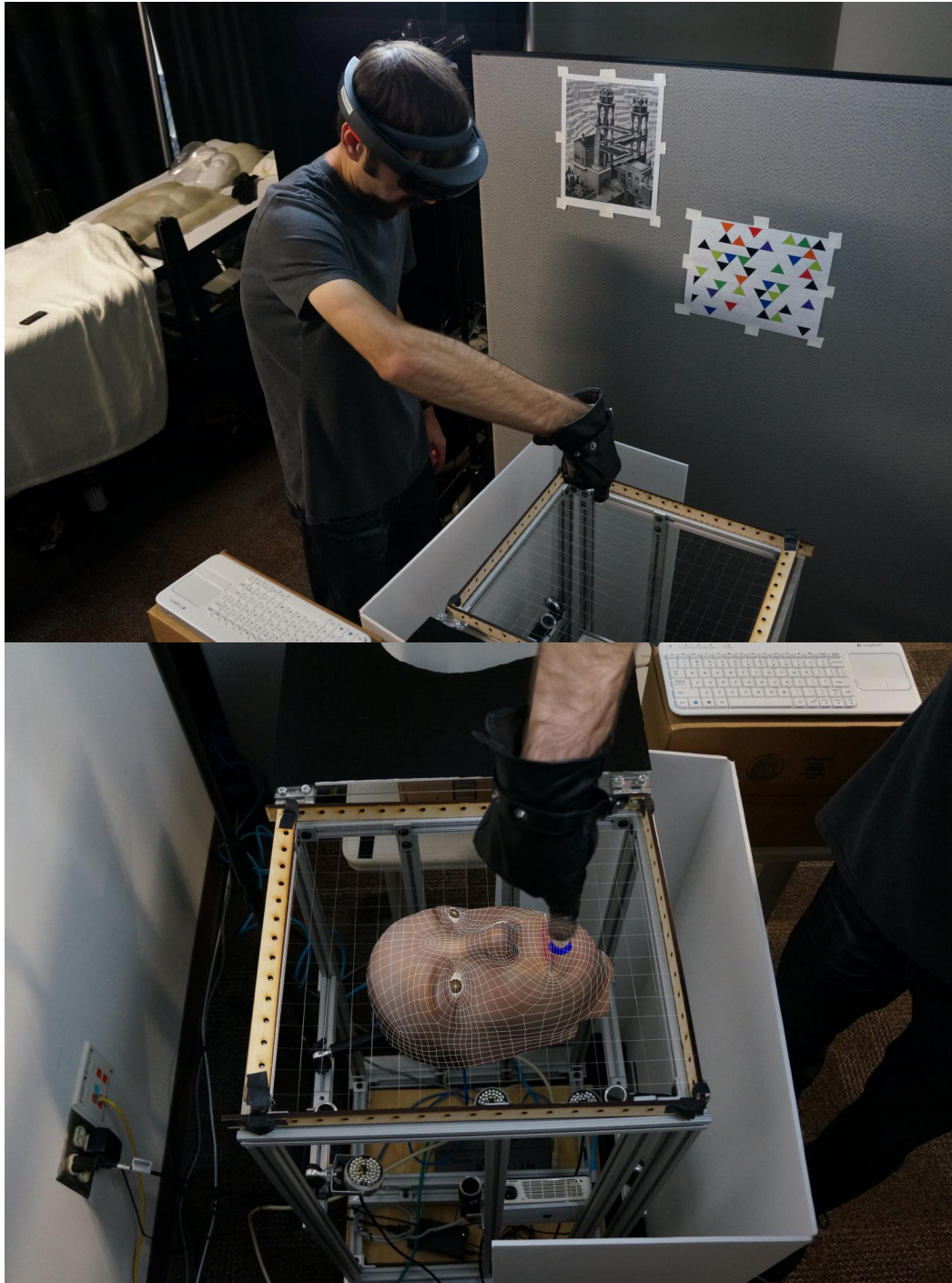


Figure 7.4: Midair touch condition. Top: participant wearing a HoloLens interacting with a virtual 3D head hologram with no physical surface present. Bottom: example (simulated) participant view, with virtual imagery provided by the HoloLens.

- Our participants were less accurate in selecting targets of specified sizes in the two HMD conditions than in the two SAR conditions.
- According to subjective questionnaire results, participants found the SAR Head to be the least demanding of the four conditions. The two SAR conditions were considered roughly equal in terms of usability.
- Participants found the SAR Head to be the easiest condition in terms of accurately touching the targets. The SAR Plane was considered to be the easiest in terms of visually locating targets by over 50% of participants, since all targets were visible from a single vantage point. However, the SAR Head was ranked as the easiest in this regard by almost 40% of users, despite requiring some physical effort to view certain locations.
- Participants overwhelmingly preferred interacting with the SAR Head compared to the other three conditions, finding it largely intuitive and user-friendly. The second-most preferred condition was the HMD Head; although it was considered to be more difficult and less intuitive than the SAR Plane, participants expressed appreciation for interacting with a matching physical object and for the novelty of the HoloLens. However, this novelty did not outweigh the difficulty of the HMD Hologram condition, which was the least preferred.

7.2.2 *Experimental Setup*

Our study platform (Figure 7.5), common to all conditions, uses Prototype Rig I and the physical-virtual plane and head surfaces (Chapter 4). To support touch sensing, we used four Point Grey Blackfly monochrome cameras (resolution 640×512 capturing at 30 frames per second) with removable 780 nm IR filters and three IR illuminators (850 nm). Imagery for the two SAR conditions is provided by an AAXA P300 pico projector (resolution 1920×1080). The cameras and projector are jointly calibrated, following the process outlined in the preprocessing phase of Chapter 3.

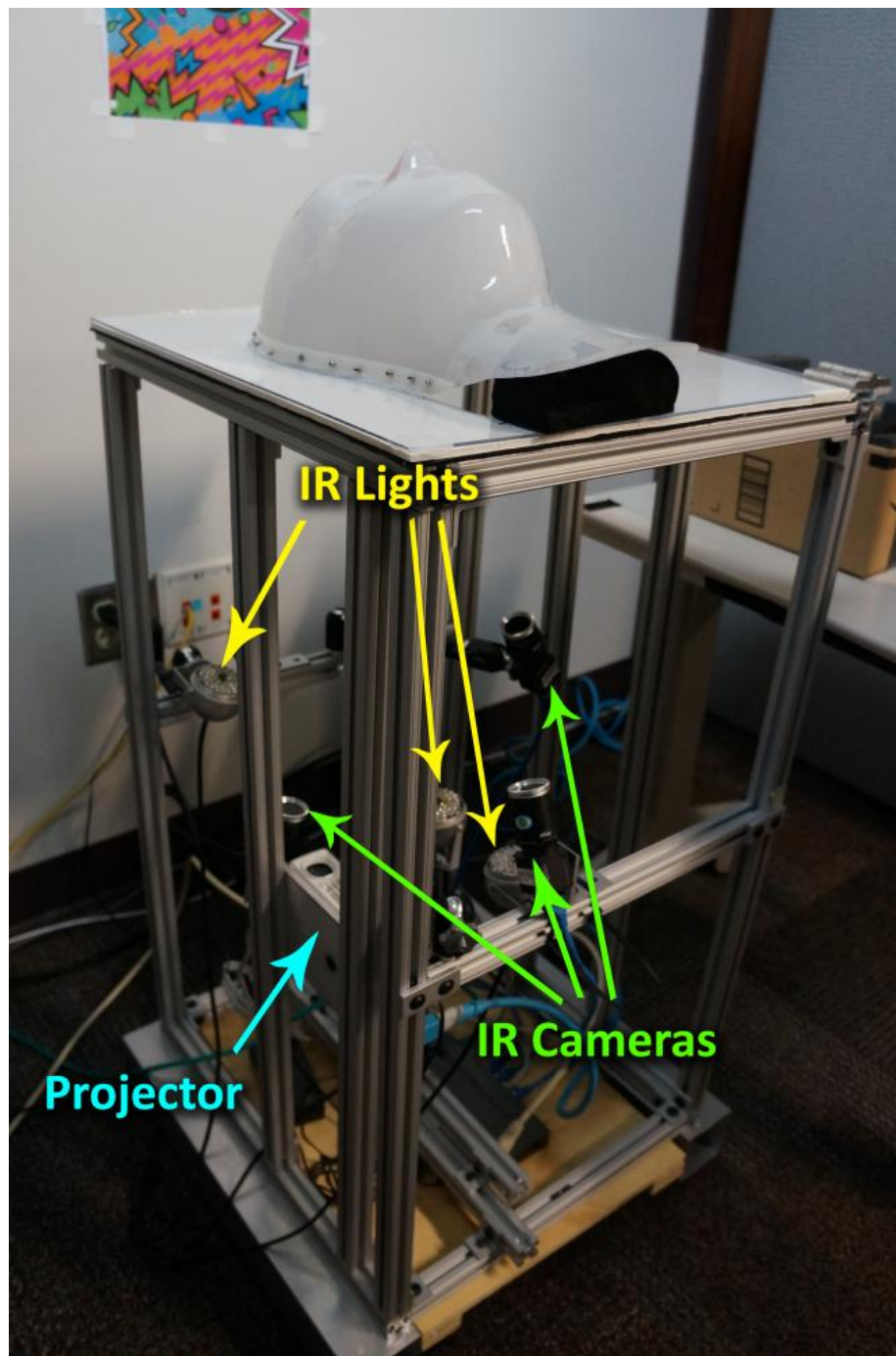


Figure 7.5: Study platform, following the general setup of Prototype Rig I from Chapter 4. Infrared lights and cameras are used for touch sensing, with registered graphics from a projector. In some conditions, virtual imagery is instead provided by a HoloLens. We also considered interactions on a plane surface and in midair.

For the HMD conditions, we used a Microsoft HoloLens to display virtual imagery of the head model. For the HMD Head condition, virtual imagery was aligned to the physical head surface using the physical-virtual alignment method discussed in Section 3.4. Once the alignment was performed, we created a spatial anchor in the HoloLens that preserved this position over time. The physical head was removed for the HMD Hologram condition; augmented imagery was presented in the same physical location the head previously occupied via the spatial anchor.

We made a few modifications to this platform to support the four experimental conditions. To allow for simple removal and replacement, we attached our physical head surface to the aluminum frame with hinges. This ensured that the head had a constant position. Similarly, we constructed a wooden support for our planar surface that attaches firmly to the top of the platform. As a safety mechanism, we placed a wooden guard with a wire grid on top of the frame during the HMD Hologram condition to prevent participants from accidentally touching internal equipment.

7.2.2.1 Touch Sensing

Touch input for the four conditions was achieved using two related camera-based methods.

1. The first method, used in the conditions featuring a physical rear-projection surface (SAR Head, SAR Plane, and HMD Head), employs the relational lookup table architecture outlined in Chapter 3, specifically using the projection space touch sensing algorithm of Section 3.2.2.1. Additionally, to support touch sensing alongside the use of the HoloLens in the HMD Head condition, we follow the special considerations discussed in Section 3.4.
2. The second method was designed specifically for the HMD Hologram condition to allow for free-space touching. Instead of processing camera imagery for potential touches on a physical surface, the touch sensing system localizes a user's finger in 3D space. When user

input is indicated through the separate input controller, this 3D position is used as the touch location. We asked participants to wear a special glove experimentally determined to not reflect IR light with a hole cut in the index fingertip to simplify localization.

Rather than creating a specific virtual model for the plane surface, we forward-projected the full three-dimensional head model onto it and modified the internal set of lookup tables accordingly. As a result, touches detected on the plane were immediately converted to their *corresponding positions on the 3D head model*. This allowed us to use the same semantic content engine for all conditions, with some additional functionality to support the HoloLens interactions.

7.2.2.2 Visual Stimuli

For the two SAR conditions, virtual imagery is provided by a projector. Following our previously described touch sensing setup, this requires calibrating the projector and determining its geometric relationship to the physical surfaces so that the appropriate projection image can be computed.

For the two HMD conditions, visuals are instead provided by a Microsoft HoloLens (Figure 7.6). In particular, for the HMD Head condition, the hologram is registered to the physical head surface; for the HMD Hologram condition, the physical head is removed, and the hologram is presented in the same physical location the head previously occupied. To support this, we used the physical-virtual alignment procedure discussed in Section 3.4. While developing this method, we determined that roughly sixty control points on the virtual model with two corresponding gaze vector captures resulted in reasonably close alignments. In practice, they required small manual translational adjustments on the order of 1 or 2 cm and minimal rotational tweaks. For the study, we performed this alignment procedure only one time, using the same transformation across all participants.

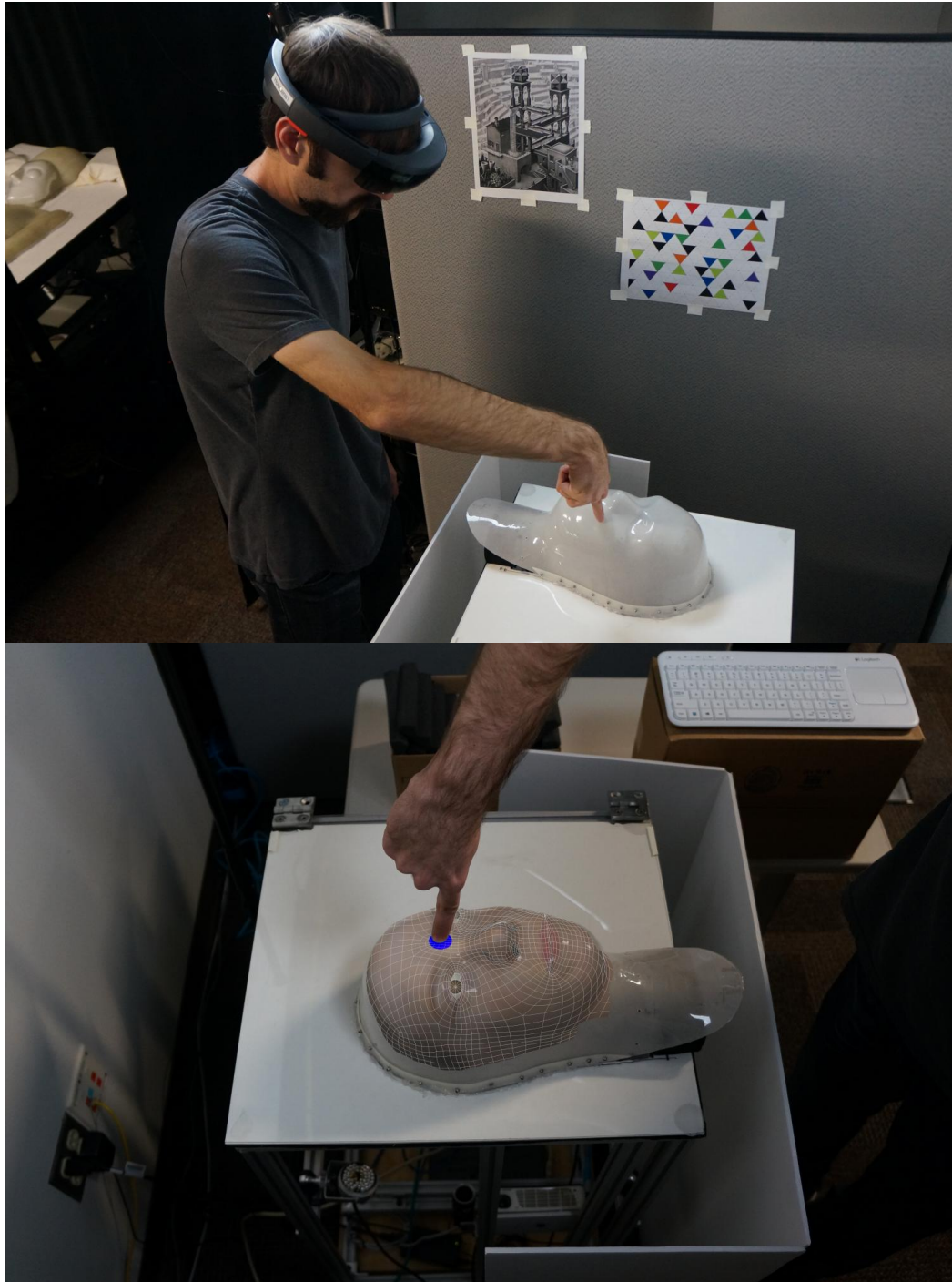


Figure 7.6: Alignment of a virtual HoloLens hologram to a physical object. Top: participant wearing a HoloLens interacting with a virtual 3D head hologram on a matching head-shaped surface. Bottom: example (simulated) participant view, with virtual imagery provided by the HoloLens.

7.2.2.3 *Computing*

The study setup closely follows the methodology presented throughout Chapter 3 and the implementation described in Chapter 4. Touch sensing and rendering were completely decoupled, with one computer processing camera imagery to detect touches and a second computer running a Unity server. Along with controlling the visual stimuli displayed to users, the Unity server also handled study state and data logging routines. The same Unity server managed all four study conditions, featuring the same semantic content engine; in particular, detected touches across all conditions were always linked to their corresponding positions on the same virtual head model. For the SAR conditions, the Unity server created appropriate projected imagery through a virtual camera with the physical projector’s calibration data applied to it. To support the two HMD conditions, the server interfaced with a HoloLens client containing the same virtual head model for display in the HMD. After performing the physical-virtual alignment procedure, we stored a spatial anchor so that the HoloLens would render the virtual head imagery on top of the physical head surface. The partial IR transparency of the head surface, along with reflective materials in the physical setup, prevented the HoloLens from obtaining reliable environmental maps, so we added several color images with asymmetric features to the study environment to help preserve the anchor over time.

7.2.3 *Experiment*

Next, we present an overview of our within-subjects experiment. Participants experienced all four conditions in counterbalanced order using a Latin square design. Each condition featured two phases, both focusing on accurate target selection via touch on a physical-virtual representation of a 3D human head model.

7.2.3.1 Participants

Twenty-four people (14 males), all students or professionals within our local university community, participated in the study. Their ages ranged from 18 to over 50. Eight of the participants wore glasses during the experiment, and no participants reported any visual or motor disorders. We asked participants to interact with the physical-virtual content using their dominant hand; twenty-two were right-handed. Of the two left-handed participants, one used the non-dominant hand due to a medical condition. Only three subjects had previously used a HoloLens, and fourteen total had some level of experience interacting with 3D content. Prior to the start of the study, we measured each participant's interpupillary distance (IPD), used by the HoloLens to accurately generate virtual content ($M = 6.12$ cm, $SD = 0.3$ cm).

7.2.3.2 User Tasks

First, we created a set of 39 vertices uniformly distributed across the 3D head model (Figure 7.7). Each vertex had three associated spheres of specific radii—small (5 mm), medium (7.5 mm), and large (10 mm)—which served as the basis for the visual targets. Rather than displaying the spheres directly, which would have resulted in different appearances across the four study conditions, we intersected them with the head geometry, producing consistent targets that were approximately circular in shape. These targets were displayed individually in the *Touch Accuracy Phase* and in sets of three during the *Cognitive Load Phase*. In both phases, participants were tasked with touching these targets. In the SAR Head, SAR Plane, and HMD Head conditions, the location of the first detected contact on the head or planar surface was retained as the participant's touch; for the HMD Hologram condition, the location of the participant's finger at the time he or she pressed the HoloLens clicker was accepted as the submitted touch.

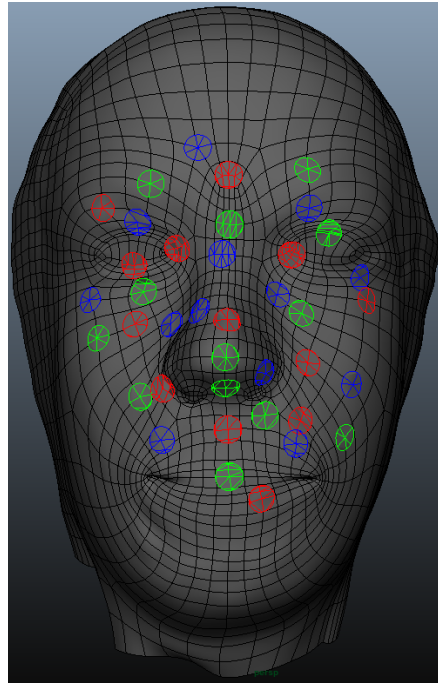


Figure 7.7: The set of 39 visual targets displayed to participants during the four study conditions, distributed across the 3D head model. For the *Touch Accuracy Phase*, the targets were divided into three groups: small (5 mm, shown above in red), medium (7.5 mm, green), and large (10 mm blue); all targets are shown with small sizes above for the purposes of visualization.

7.2.3.2.1 *Touch Accuracy Phase*

In the first study phase, participants were presented a sequential set of individual targets, and they were asked to touch the centers of these targets as carefully and precisely as possible with no time constraints. Here, we were interested in evaluating the touch performance of each participant across the four study conditions. The set of 39 targets was divided into three groups—13 small, 13 medium, and 13 large—again distributed uniformly across the 3D head model. Figure 7.7 shows these assignments by color: red for small, green for medium, and blue for large. Each condition had a preassigned, random sequence of these sized targets, and each individual target appeared twice. Upon touching each target, participants pressed the spacebar key of a separate keyboard,

using the same finger they used for touch selections. This advanced the study state to the next target. Furthermore, this ensured that all touches in the *Touch Accuracy Phase* started from a consistent position, allowing for response time comparisons. The keyboard was only used during this phase, and it was not used to indicate touch input.

There were two dependent variables representing participant touch performance in the *Touch Accuracy Phase*. The first, *distance from touch to target*, is the Euclidean distance between a displayed target and the location of the participant's touch. For the conditions involving a physical surface, the 3D touch location was obtained directly via the lookup table; for the HMD Hologram condition, the participant's finger was triangulated in 3D space. The *response time* reflected the amount of time that elapsed between the spacebar key press and the participant's next touch.

7.2.3.2.2 Cognitive Load Phase

In the second study phase, participants were presented with a sequential set of target triplets. Each triplet comprised one small, one medium, and one large target, each drawn from the original set of 39 targets (Figure 7.8). Participants were responsible for completing two simultaneous tasks during this study phase. The primary task concerned target size estimation, requiring the participants to select the medium-sized target by touch (Figure 7.8a). Each condition again had a predetermined, random sequence of target triplets, and all of the 39 targets were included as the correct answer for two trials.

Concurrently, participants completed a secondary task. Along with the predetermined sets of target triplets, each condition had an associated starting number around 500, which participants used in a mental arithmetic task. For some target triplets, the virtual human closed her eyes. For these trials, participants were required to subtract 7 from their current count and then verbally report the result (Figure 7.8b). If the virtual human's eyes were instead open during a trial, participants

were instructed to retain the current count and provide no verbal response. The eye behavior of the virtual human was only allowed to change upon the display of a new target triplet. In some cases, the virtual human's eyes remained closed for two consecutive triplets; participants were instructed to update the count and verbally respond twice in this situation. The eye behavior sequence of the virtual human was predetermined for each condition.

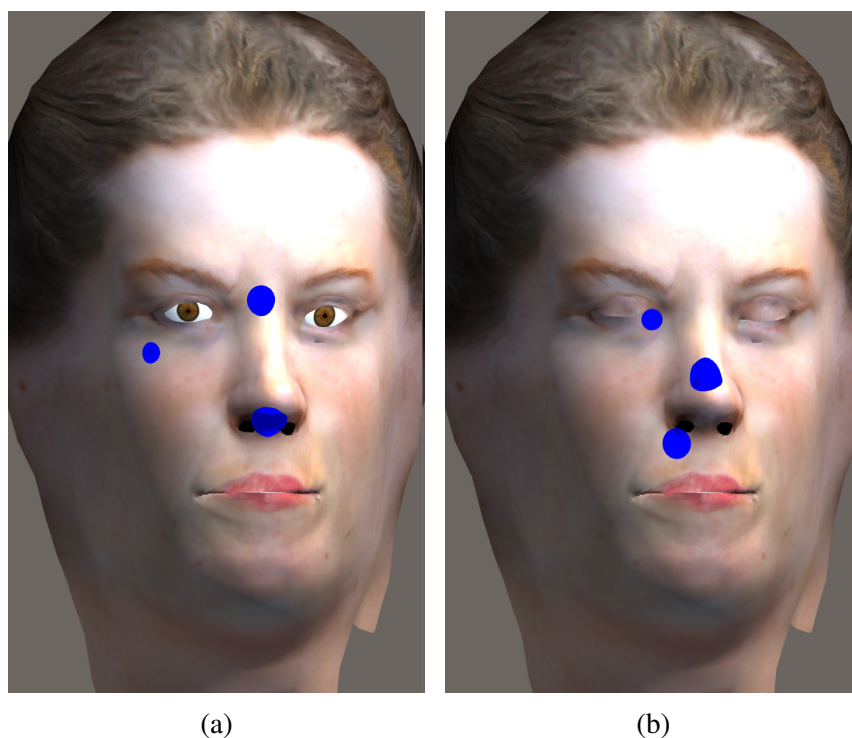


Figure 7.8: Example *Cognitive Load Phase* trials, for which a set of one small, one medium, and one large target is presented to participants, who must quickly select the medium-sized one. This phase also included a secondary counting task: given a starting number around 500, participants would decrement by 7 and provide a verbal update of their number for trials for which the virtual human's eyes were closed (b).

Unlike the *Touch Accuracy Phase*, the *Cognitive Load Phase* had a time limitation: target triplets were only shown for 4 seconds, after which the next triplet would automatically appear. Participants were asked to give priority to the primary size estimation task and only complete the

secondary verbal counting task after touching the medium-sized target from the current triplet. The dual-task paradigm in this phase was intended to provide a means of assessing the mental, physical, and temporal demands induced by each physical-virtual interface.

The *Cognitive Load Phase* also featured two dependent variables. The first, *selection of correct target*, considered the percentage of trials for which participants correctly touched the medium-sized target out of the target triplets. The *verbal counting task response* reflected the participant's accuracy in performing the secondary mental arithmetic task and providing verbal responses.

7.2.3.3 Study Procedure

After providing consent and completing a short demographic questionnaire, participants watched a 2-minute video providing simplified samples of the cognitive load tasks. This provided them an opportunity to practice the verbal counting task as prompted by the eye behavior of the virtual human. The video also presented target triplets every 4 seconds, just as in the *Cognitive Load Phase*. However, the targets were all the same size, requiring no size estimation or touching. This was to familiarize participants with the timing of the *Cognitive Load Phase* trials and to ensure they understood when verbal counts were and were not required.

Prior to each condition, participants completed a short training phase featuring 6 *Touch Accuracy Phase* practice trials and 25 *Cognitive Load Phase* practice trials on the upcoming physical-virtual interface. They were given a final opportunity to ask questions. Afterwards, the *Touch Accuracy Phase* began. Following this phase was a 60-second break period, during which the virtual imagery was disabled, allowing participants time to rest their arms, neck, and eyes. Ten seconds prior to the end of the break, participants were reminded of their starting number for the secondary counting task. The *Cognitive Load Phase* began immediately after the break. After the *Cognitive Load Phase*, participants completed subjective questionnaires. The NASA Task Load Index (TLX) [63]

provides a measurement of cognitive load, and the Simple Usability Scale (SUS) [22] concerns the perceived usability of each physical-virtual interface.

Following completion of the four conditions, participants were asked several subjective questions relating to their personal difficulty rankings of various aspects of the different interfaces and their preferences. During a brief followup interview, we asked other qualitative questions regarding their experiences.

7.2.4 Results

As an overview of the results, we found significant main effects of display condition on the following:

- *Touch Accuracy Phase:*
 - touch-target distance
 - response time
- *Cognitive Load Phase:*
 - medium-sized (correct) target selection
 - small-sized (incorrect) target selection
- Subjective responses:
 - NASA-TLX task load scores
 - SUS usability scores
 - subjective rankings of the easiest and hardest conditions in terms of both accurately touching and visually locating targets

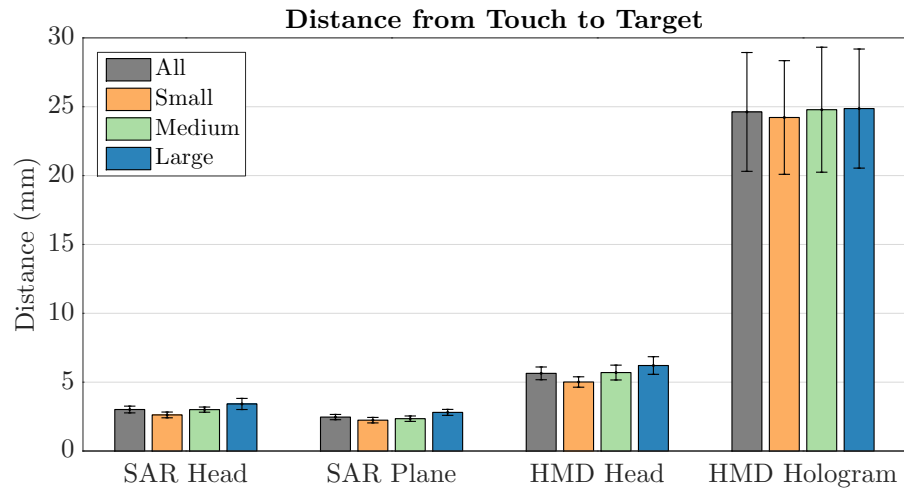
Below, we present these results in more detail. For our analysis, we used repeated-measures ANOVAs and Tukey multiple comparisons with Bonferroni correction at the 5% significance level. To confirm normality, we used Shapiro-Wilk tests at the 5% level and QQ plots. When Mauchly's test indicated the violation of the sphericity assumption, we corrected degrees of freedom using Greenhouse-Geisser estimates of sphericity. For the analysis of questionnaire responses, we used parametric statistical tests, following the ongoing discussion in the field of psychology suggesting that such tests are valid and potentially more expressive for such ordinal data analyses [85, 88].

7.2.4.1 Touch Performance

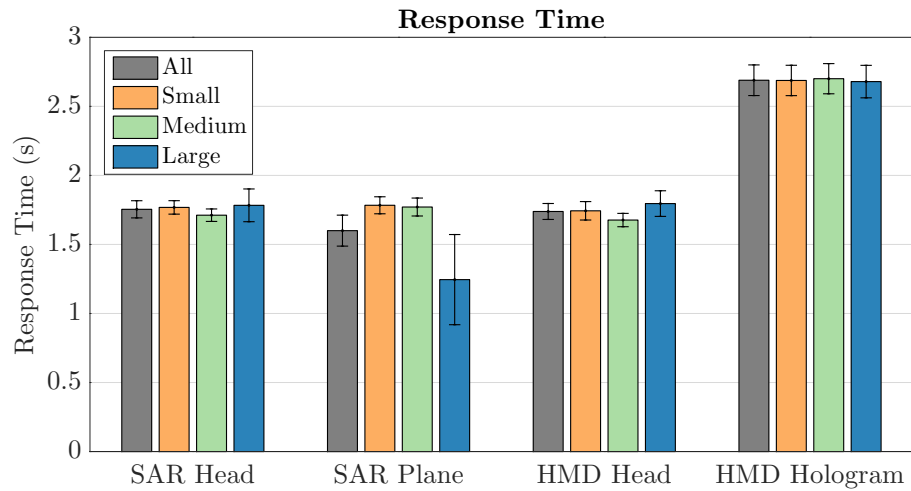
During the *Touch Accuracy Phase*, participants were tasked with touching individual targets as carefully and precisely as possible. We found significant main effects of display condition on both dependent measures: touch-target distance ($F(1.02, 23.50) = 25.16, p < 0.001, \eta_p^2 = 0.52$) and response time ($F(2.08, 47.90) = 32.45, p < 0.001, \eta_p^2 = 0.59$). Figures 7.9a and 7.9b show the average Euclidean distance between displayed targets and participant touch and the average touch time, respectively, for the four study conditions. Pairwise comparisons also revealed significant differences between all pairs of conditions in terms of touch-target distance (all $p < 0.05$). Additionally, according to pairwise comparisons, the HMD Hologram condition had significantly longer response times than the other three conditions (all $p < 0.001$).

7.2.4.2 Cognitive Load

During the *Cognitive Load Phase*, participants performed a size estimation touch task along with a concurrent mental arithmetic task. Figure 7.10 shows the average percentage of trials for which participants correctly selected the medium-sized target (Figure 7.10a) or incorrectly chose the small or large target (Figure 7.10b and Figure 7.10c, respectively).



(a)

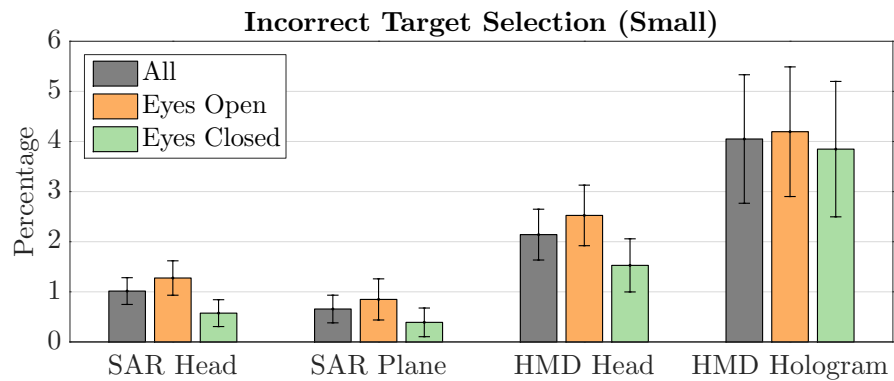


(b)

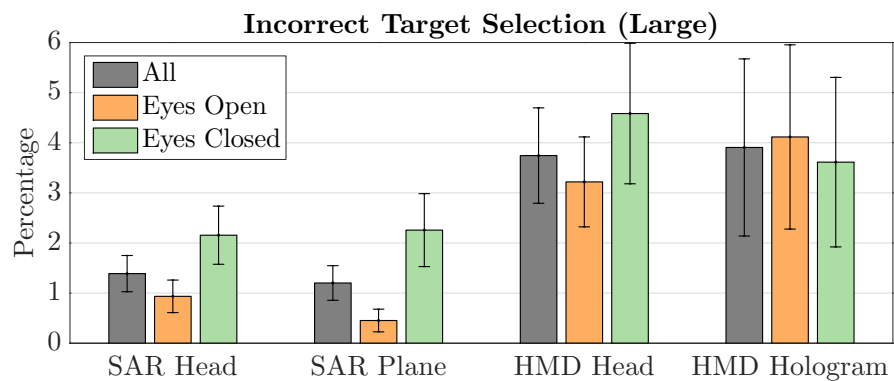
Figure 7.9: Results for the *Touch Accuracy Phase*, separated by study condition. The error bars show the standard error. (a) Average distance between displayed target and user touch. (b) Average time taken to touch a target. We found significant main effects of display condition on both touch-target distance and response time.



(a)



(b)



(c)

Figure 7.10: Results for the target selection task in the *Cognitive Load Phase*. (a) The percentage of trials for which participants correctly touched the medium-sized targets. (b) and (c) The percentage of trials for which participants incorrectly touched either the small- or large-sized targets, respectively. We found significant main effects of display condition on medium (correct) and small (incorrect) selections.

We found a significant main effect of display condition on the correct selection of medium targets ($F(1.40, 32.10) = 3.87, p = 0.045, \eta_p^2 = 0.14$) and on incorrect selections of small targets ($F(1.39, 31.99) = 4.83, p = 0.025, \eta_p^2 = 0.17$). However, we did not observe a significant main effect of display condition on incorrect selections of large targets ($F(1.39, 31.88) = 2.52, p = 0.113, \eta_p^2 = 0.10$). Pairwise comparisons revealed a significantly lower percentage of correct selections for the HMD Head condition than for the SAR Plane condition ($p = 0.019$) as well as trends suggesting a lower percentage of small selections for the SAR Plane than for the HMD Head ($p = 0.066$) and the HMD Hologram ($p = 0.062$).

Interestingly, we did not observe a significant main effect of display condition on either correct or incorrect verbal counting task responses.

7.2.4.3 Subjective Responses

Figures 7.11 and 7.12 show the results of the subjective questionnaires (concerning task load and usability), participant preferences, and participant rankings for the four conditions.

7.2.4.3.1 Task Load

Results for the NASA-TLX questionnaire are shown in Figure 7.11a. Lower values indicate decreased task load. We found a significant main effect of display condition on the NASA-TLX task load scores ($F(3, 69) = 3.40, p = 0.023, \eta_p^2 = 0.129$). In particular, pairwise comparisons revealed significantly higher average task load for the HMD Hologram condition than for the SAR Head condition ($p = 0.033$).

7.2.4.3.2 Usability

Results for the SUS questionnaire are shown in Figure 7.11b. Higher values indicate higher usability. We found a significant main effect of display condition on the SUS usability scores ($F(1.75, 40.22) = 9.95, p = 0.001, \eta_p^2 = 0.302$). Pairwise comparisons revealed significantly lower usability for the HMD Hologram condition compared to the SAR Head ($p = 0.003$) and SAR Plane ($p = 0.002$) conditions. Additionally, pairwise comparisons suggested trends for lower usability for the HMD Head condition compared to the SAR Head ($p = 0.074$) and the SAR Plane ($p = 0.067$) conditions.

7.2.4.3.3 Preferences

After completing the four study conditions, we asked participants subjective questions regarding their preferred conditions (Figure 7.12a). There was a statistically significant preference for the SAR Head condition over the other three ($z = 2.357, p = 0.018$). In particular, participants generally preferred interactions with a matching surface that provided natural physical touch feedback; separately, many participants commented about discomfort due to wearing the HMD, and all noticed the narrow field of view. Those participants who preferred the HMD Head and HMD Hologram conditions generally appreciated the novelty of the interface, but there was still a strong preference for interacting with a physical surface. Participants provided many recurring justifications for their preferences, including:

The SAR Head was easier to use “because [it was] physically there”

The HoloLens was “uncomfortable” and “very heavy”

The SAR Plane was easier to use because “[I am] used to seeing [imagery] on [a] screen”

7.2.4.3.4 *Rankings*

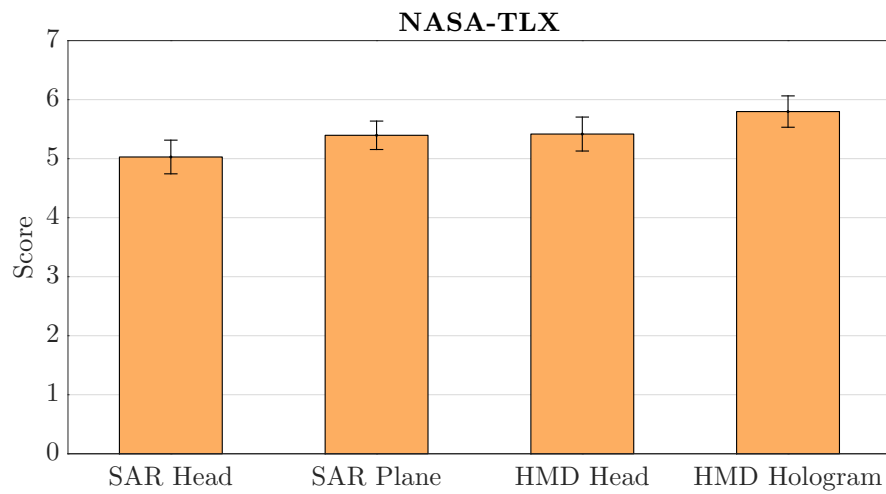
Additionally, we asked participants to rank the difficulty of the four conditions across two tasks: accurately touching and visually locating the targets (Figure 7.12b and Figure 7.12c, respectively). For analysis of these rankings, we calculated the exact Clopper-Pearson confidence interval [30, 51]. In terms of accurately touching targets, the SAR Head was considered to be the easiest ($z = 3.771$, $p < 0.001$) and the HMD Hologram was considered to be the hardest ($z = 5.185$, $p < 0.001$). Likewise, participants found the task of visually locating the targets much easier in the two SAR conditions ($z = 3.674$, $p < 0.001$) and more challenging in the HMD conditions ($z = 2.449$, $p = 0.014$).

7.2.5 *Discussion*

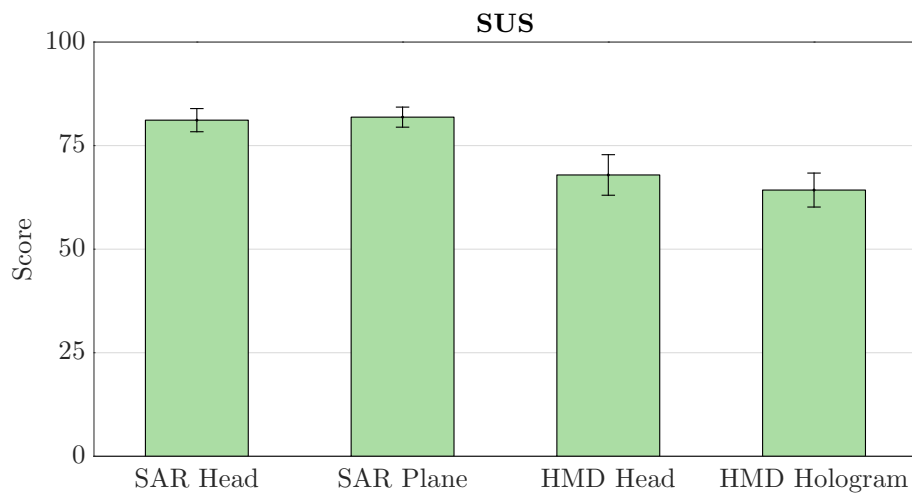
Overall, we observed a strong preference for the SAR Head condition. Questionnaire results and recurring subjective comments indicated that this condition was easier to use and more intuitive in the presented touch task. Because of familiarity with smart phones and touchscreens and the ability to see all targets from a single viewpoint, many participants found the SAR Plane to be the easiest condition in terms of visually locating the targets. Interestingly, participants expressed that it was easier to accurately touch the targets on the SAR Head, even compared to the SAR Plane. Touch-target distance was indeed the lowest for the two SAR conditions; even though the HMD Head condition featured touch interactions on the same head-shaped surface as the SAR Head condition, touch-target distances nearly doubled, perhaps due to the differences in visual display or to the increased cognitive load of the interface. The HMD Hologram featured the largest touch-target distances—roughly 2.5 cm on average. Touch response times were generally quite similar across the SAR Head, SAR Plane, and HMD Head conditions, while response times were noticeably longer for the HMD Hologram condition.

The two most preferred conditions, the SAR Head and HMD Head, featured a physical object whose geometry matched the displayed virtual content. Many participants specifically expressed the usefulness of interacting with the virtual head on a matching head-shaped object that provides physical feedback on touch. The two HMD conditions were generally perceived to be more difficult than the two SAR conditions; indeed, participants tended to select the wrong-sized targets in the *Cognitive Load Phase* slightly more frequently in these cases. Several participants mentioned that the use of an external input device to indicate “touches” during the HMD Hologram condition was a less enjoyable and less user-friendly experience than the direct touch paradigms. It is important to note that the preferences and results for the HMD conditions are affected by the attributes of the HoloLens, such as general comfort, ergonomics, field of view, and the inability for physical objects (particularly hands) to occlude the augmented imagery. Such issues are inherent to modern HMDs, and so we expect these results hold in general for similar AR touch tasks.

These results suggest general guidelines for practitioners interested in supporting similar AR touch tasks, depending on the available equipment and on the importance of usability factors and cognitive load requirements. The use of projected imagery for visual content over HMDs is preferable for training systems focusing on high touch accuracy, high usability, and low cognitive load; when possible, the geometry of the physical display surface should match the geometry of the virtual content. However, physical objects in such touch tasks are not always appropriate for projected imagery. In such cases, augmenting the object with an HMD leads to similar (but reduced) benefits compared to free-space interaction, which can be quite challenging and exhibit significantly lower user touch performance and accuracy.

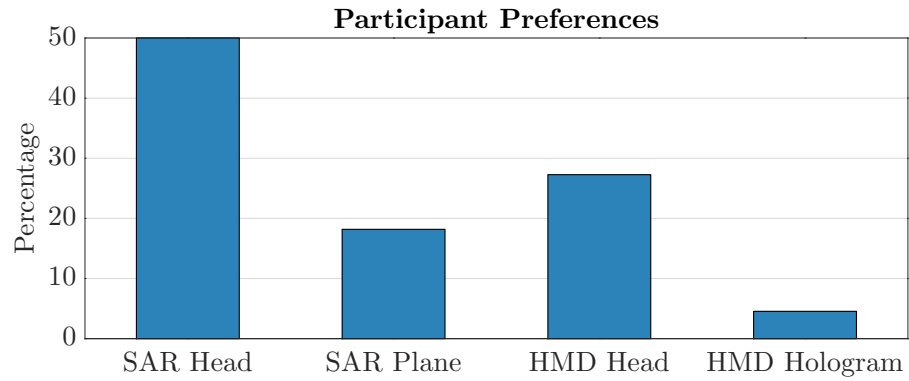


(a)

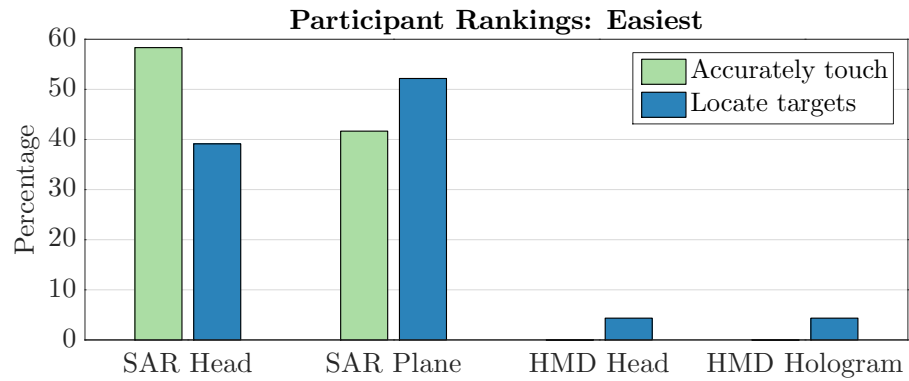


(b)

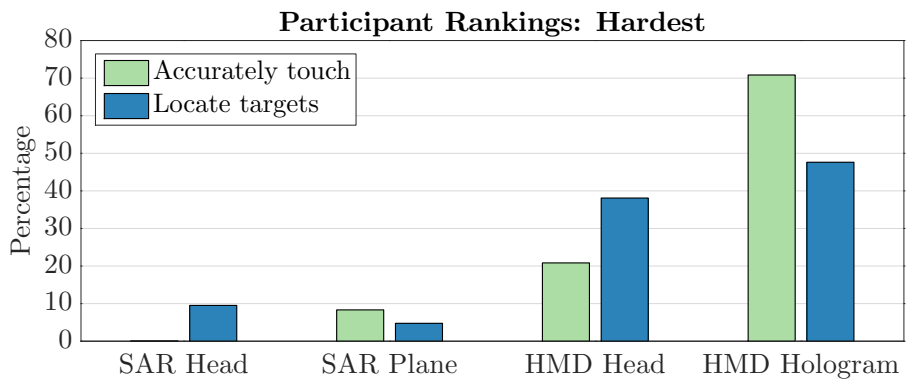
Figure 7.11: Results of the subjective questionnaires. (a) NASA Task Load Index (TLX): lower scores indicate lower mental, physical, and temporal demands. (b) Simple Usability Scale (SUS): higher scores indicate higher perceived usability. We found significant main effects of display condition on both scores. These results align with objective user performance across the four study conditions in terms of touch-target accuracy and response time.



(a)



(b)



(c)

Figure 7.12: Subjective rankings for the four experimental conditions. (a) Overall preferred condition. (b) and (c) Subjectively ranked easiest and hardest conditions, respectively, in terms of *accurately touching* and *visually locating* the targets. Participants preferred interacting with the head-shaped surface, since it affords physical feedback on touch, and they found the HoloLens uncomfortable and disliked the field of view limitations.

CHAPTER 8: CONCLUSIONS AND FUTURE WORK

This dissertation introduced a generalizable system for integrated touch detection and semantic response on non-parametric rear-projection surfaces. In Chapter 3, we described a theoretical framework based on a lookup table architecture storing relationships between optical devices (cameras and projectors), a physical touch surface, and the virtual content registered to and displayed on the surface. This includes two algorithms for detecting touch using the lookup table (Section 3.2), along with a related mechanism for associating detected touch input with semantic responses in the context of the virtual content (Section 3.3). Next, we presented a general software architecture capable of realizing this framework in Chapter 4; in addition, we discussed two physical prototype implementations supporting touch interactions on several rear-projection surfaces using this software architecture. Chapter 5 covered a variety of semantic content created for these surfaces, along with illustrative examples of touch-triggered behavior on them. These behaviors include dynamic imagery changes, such as touch-based painting and object manipulation; dynamic graphical model changes, such as raising and lowering a virtual human’s eyelids; and hover-based events, such as a musical interface that produces a note whose pitch is based on surface proximity.

Additionally, we evaluated a variety of system consistency metrics in Chapter 6, demonstrating that the lookup table accurately encodes device-content relationships; furthermore, in practice, our system localized datasets of user touch input within less than 2 mm of the intended targets with low latency across these surfaces. Finally, we investigated the use of this methodology in human-subject studies (Chapter 7), including an initial exploration of a touch-capable stroke patient simulator and an examination of the effects of mismatches between physical and visual perceptions in an augmented reality touch task. In the latter study, we showed that users exhibit improved touch performance, decreased cognitive load, and greater subjective enjoyment when interacting with physical surfaces with geometry that matches the virtual content, which our method facilitates.

The main contributions of this research over typical touch interfaces can be summarized as follows:

- the design and implementation of a relational lookup table architecture that supports touch sensing on non-parametric surfaces, extending beyond the simple geometric shapes traditionally used for touch surfaces (such as tabletops);
- the registration of virtual imagery to such physical touch surfaces to support semantically defined touch responses;
- a plane sweep approach for detecting hover input;
- the novel application of touch input and response to patient simulators; *and*
- a controlled investigation into the benefits of touch input on content-matched physical surfaces.

Here, we briefly consider potential future work in the scope of this research.

Automated or guided camera/projector placement: The placement of cameras and projectors in this paradigm requires some forethought and potentially some trial and error. While final system evaluation results can indicate insufficient camera coverage, this only occurs after the system has been fully calibrated. Instead, we suggest tools that provide some amount of guidance during the camera and projector placement process to improve overall system performance, a topic which has been explored in the context of surveillance applications [16] and image-based modeling [50]. This could potentially include machine learning methods that analytically model and assess alternative configurations.

Off-surface interactions: The plane sweep algorithm of Section 3.2.2.2 is able to estimate the distance between an object and the surface. While we presented an initial exploration of this paradigm in the form of a proximity-based musical instrument (Section 5.1), such interactions

could be explored in greater detail. For example, in the context of human patient simulation, neurological exams often require the patient to follow the healthcare provider's finger with his or her eyes [127], which could be facilitated as a hover interaction.

Continuous calibration: The preprocessing phase calibration routine, presented in Section 3.2, is performed a single time prior to system use. In the event that a camera, a projector, or the surface moves, the calibration process must be repeated. Instead, we expect that continuous online calibration techniques [39] can be employed to maintain a sufficient level of calibration accuracy over time—for instance, through static calibration markers on the physical rigs and through monitoring routines that detect when devices have moved.

Novel applications: The comparison of various physical-virtual interfaces in the second human-subject study presented in Chapter 7 highlights advantages of this touch sensing paradigm in terms of user performance and cognitive load. We anticipate further benefits afforded by this approach in facilitating the integrated visual and tactile understanding of the geometric relationships of complex virtual content, such as relative shapes, sizes, and positions. The ability to register these virtual components to matched-geometry physical touch surfaces with dynamic imagery capabilities could support a variety of novel three-dimensional touch-based applications, such as terrain maps, anatomical models, telepresence (e.g. [47, 119]), and artistic media.

Given the preliminary feedback from nursing students in the simulated stroke assessment (Section 7.1), we anticipate that the addition of touch input capabilities to physical-virtual patient simulators will provide significant benefits to healthcare training scenarios. These include improvements to presence and the ability for learners to perform more realistic diagnostic procedures compared to standard patient simulators. Likewise, we expect these benefits could translate naturally to intelligent virtual agents in general—for instance, through the use of touch input for the conveyance of emotions (e.g. [65, 66]) and cultural proxemics training [60, 144].

APPENDIX: UNITY PROJECTION MATRICES

Here, we describe the process for modeling a calibrated projector in the calibration coordinate space \mathbb{TCH}_3 as a virtual Unity camera in the graphics space \mathbb{GFX}_3 . The former is represented by a 3×4 calibration projection matrix \mathbf{P} , while the latter is represented by a 4×4 graphical projection matrix \mathbf{U} . Each models the forward-projection of a three-dimensional point in space onto a pixel of a two-dimensional image plane differently, and the primary purpose of this process is the computation of a graphical projection matrix that produces forward-projections equivalent to the calibration matrix. While our discussion here is specific to Unity, it relies on general graphical projection matrices and therefore could be implemented in other graphics engines (e.g. OpenGL [131]), potentially with some small tweaks. Once this process is complete, the imagery of the graphics mesh G computed by the virtual camera in Unity will be registered to the actual touch surface S when displayed by the projector. The overall framework utilizes various components of the general methodology described in Chapter 3, and the graphical projection matrices are computed as part of the lookup table preprocessing phase.

To simplify notation, we will refer to the calibration space \mathbb{TCH}_3 as \mathbb{T} and the graphics space \mathbb{GFX}_3 as \mathbb{G} . Moreover, we will denote the *local space* of a virtual camera as \mathbb{L} : in this space, the camera's position is the origin $[0, 0, 0]^T$, the x - and y -axes of its image plane are aligned to the world's x - and y -axes, and the camera's forward-vector is the local z -axis $[0, 0, 1]^T$. A transformation from coordinate space \mathbb{A} to \mathbb{B} is represented by the rotation matrix $\mathbf{R}_{\mathbb{A}}^{\mathbb{B}}$ and translation matrix $\mathbf{T}_{\mathbb{A}}^{\mathbb{B}}$.

The input to the process is the projection matrix $\mathbf{P}^{\mathbb{T}}$ of a given projector, located at $\mathbf{C}^{\mathbb{T}}$ in the calibration coordinate space \mathbb{T} . This 3×4 matrix $\mathbf{P}^{\mathbb{T}}$ can be decomposed to intrinsic and extrinsic calibration matrices, given by

$$\mathbf{P}^{\mathbb{T}} = \mathbf{K}[\mathbf{R}^{\mathbb{T}} \mid \mathbf{t}^{\mathbb{T}}]$$

where

$$\mathbf{t}^{\mathbb{T}} = -\mathbf{R}^{\mathbb{T}}\mathbf{C}^{\mathbb{T}} \tag{A.1}$$

We will make use of both of these formulations as necessary. It is assumed that we have computed a transformation between the calibration space \mathbb{T} and graphics space \mathbb{G} , consisting of a rotation matrix $\mathbf{R}_{\mathbb{T}}^{\mathbb{G}}$ and a translation matrix $\mathbf{T}_{\mathbb{T}}^{\mathbb{G}}$ (from preprocessing phase step PP7). First, we compute an equivalent calibration matrix in graphics space \mathbb{G}

$$\mathbf{P}^{\mathbb{G}} = \mathbf{K}[\mathbf{R}^{\mathbb{G}} \mid \mathbf{t}^{\mathbb{G}}]$$

such that $\mathbf{P}^{\mathbb{T}}$ and $\mathbf{P}^{\mathbb{G}}$ forward-project equivalent points in their respective coordinate spaces to the same pixels on their image planes. From $\mathbf{P}^{\mathbb{G}}$, we will compute a frustum that will be applied to the final Unity projection matrix \mathbf{U} . Below, we describe this process in more detail.

A note on coordinate handedness: The coordinate space in Unity is left-handed, whereas the coordinate conventions in standard computer vision calibration approaches are right-handed. As a result, when computing a 3D point or vector to apply to Unity, we convert from right-handed to left-handed coordinates by negating the x -values.

1. First, we **transform the projector calibration in the calibration space \mathbb{T} to an equivalent projection matrix $\mathbf{P}^{\mathbb{G}}$ in graphics space \mathbb{G}** using the transformation between these two coordinate spaces. The transformed projector position in \mathbb{G} is obtained directly from the transformation matrices:

$$\mathbf{C}^{\mathbb{G}} = \mathbf{R}_{\mathbb{T}}^{\mathbb{G}} \mathbf{C}^{\mathbb{T}} + \mathbf{T}_{\mathbb{T}}^{\mathbb{G}}$$

where $\mathbf{C}^{\mathbb{T}}$ is the original position of the projector in calibration space \mathbb{T} . The point $\mathbf{C}^{\mathbb{G}}$ (with negated x -coordinate to account for handedness) is directly used as the position of the virtual camera in Unity.

While the intrinsic calibration matrix \mathbf{K} of $\mathbf{P}^{\mathbb{G}}$ remains the same in both coordinate spaces, the extrinsic matrix changes. To compute it, let us consider a point $\mathbf{X}^{\mathbb{T}}$ in the calibration

space \mathbb{T} . Given the transformation, this point moves to

$$\mathbf{X}^{\mathbb{G}} = \mathbf{R}_{\mathbb{T}}^{\mathbb{G}} \mathbf{X}^{\mathbb{T}} + \mathbf{T}_{\mathbb{T}}^{\mathbb{G}}$$

in the graphics space \mathbb{G} . The goal is to construct a calibration matrix $\mathbf{P}^{\mathbb{G}}$ such that both points $\mathbf{X}^{\mathbb{T}}$ and $\mathbf{X}^{\mathbb{G}}$ are forward-projected onto the same pixel \mathbf{x} on the image planes of $\mathbf{P}^{\mathbb{T}}$ and $\mathbf{P}^{\mathbb{G}}$, respectively:

$$\mathbf{x} = \mathbf{P}^{\mathbb{T}} \mathbf{X}^{\mathbb{T}} = \mathbf{P}^{\mathbb{G}} \mathbf{X}^{\mathbb{G}}$$

Expanding, we have

$$\begin{aligned} \mathbf{K}[\mathbf{R}^{\mathbb{T}} \mid \mathbf{t}^{\mathbb{T}}] \mathbf{X}^{\mathbb{T}} &= \mathbf{K}[\mathbf{R}^{\mathbb{G}} \mid \mathbf{t}^{\mathbb{G}}] (\mathbf{R}_{\mathbb{T}}^{\mathbb{G}} \mathbf{X}^{\mathbb{T}} + \mathbf{T}_{\mathbb{T}}^{\mathbb{G}}) \\ [\mathbf{R}^{\mathbb{T}} \mid \mathbf{t}^{\mathbb{T}}] \mathbf{X}^{\mathbb{T}} &= [\mathbf{R}^{\mathbb{G}} \mid \mathbf{t}^{\mathbb{G}}] (\mathbf{R}_{\mathbb{T}}^{\mathbb{G}} \mathbf{X}^{\mathbb{T}} + \mathbf{T}_{\mathbb{T}}^{\mathbb{G}}) \end{aligned}$$

We wish to solve for the extrinsic parameters $\mathbf{R}^{\mathbb{G}}$ and $\mathbf{t}^{\mathbb{G}}$. We start by moving the registration transformation applied to $\mathbf{X}^{\mathbb{T}}$ into the extrinsic parameters so that we can see how they operate on $\mathbf{X}^{\mathbb{T}}$ directly:

$$[\mathbf{R}^{\mathbb{T}} \mid \mathbf{t}^{\mathbb{T}}] \mathbf{X}^{\mathbb{T}} = [\mathbf{R}^{\mathbb{G}} \mathbf{R}_{\mathbb{T}}^{\mathbb{G}} \mid \mathbf{t}^{\mathbb{G}} + \mathbf{R}^{\mathbb{G}} \mathbf{T}_{\mathbb{T}}^{\mathbb{G}}] \mathbf{X}^{\mathbb{T}}$$

We immediately have

$$\begin{aligned} \mathbf{R}^{\mathbb{T}} &= \mathbf{R}^{\mathbb{G}} \mathbf{R}_{\mathbb{T}}^{\mathbb{G}} \\ \mathbf{R}^{\mathbb{G}} &= \mathbf{R}^{\mathbb{T}} (\mathbf{R}_{\mathbb{T}}^{\mathbb{G}})^T \end{aligned}$$

Finally, we can compute

$$\mathbf{t}^{\mathbb{G}} = -\mathbf{R}^{\mathbb{G}}\mathbf{C}^{\mathbb{G}}$$

analogously from Equation (A.1). Thus, with $\mathbf{C}^{\mathbb{G}}$, $\mathbf{R}^{\mathbb{G}}$, and $\mathbf{t}^{\mathbb{G}}$ known and with \mathbb{K} fixed, we can compute the calibration projection matrix

$$\mathbf{P}^{\mathbb{G}} = \mathbb{K}[\mathbf{R}^{\mathbb{G}} \mid \mathbf{t}^{\mathbb{G}}]$$

in the graphics space \mathbb{G} equivalent to the original calibration projection matrix $\mathbf{P}^{\mathbb{T}}$ in the calibration space \mathbb{T} .

2. Next, we **compute the axes of the projection matrix $\mathbf{P}^{\mathbb{G}}$** , which we will use to set the orientation of the virtual camera in Unity. The y - and z -axes of the physical projector correspond to the up-vector and forward-vector of the virtual camera, respectively. The back-projection of a pixel $\mathbf{x} = [x, y, 1]^T$ to a ray in the graphics space \mathbb{G} is given by

$$\mathbf{B}^{\mathbb{G}}(\mathbf{x}, \mu) = \begin{bmatrix} (\mathbf{M}^{\mathbb{G}})^{-1}(\mu\mathbf{x} - \mathbf{p}_4^{\mathbb{G}}) \\ 1 \end{bmatrix}$$

where $\mathbf{P}^{\mathbb{G}}$ is written in block form as $[\mathbf{M}^{\mathbb{G}} \mid \mathbf{p}_4^{\mathbb{G}}]$ [64]. Since we are computing direction vectors, the exact value of the parameter μ , which determines the length of the back-projected ray, is not important; thus, we can assume $\mu = 1$ and remove it as a parameter for the pixel back-projection function $\mathbf{B}^{\mathbb{G}}(\mathbf{x})$. Using this function, we back-project the corner pixels of the projector's image plane to rays:

$$\{\mathbf{B}^{\mathbb{G}}([0, 0, 1]^T), \mathbf{B}^{\mathbb{G}}([w, 0, 1]^T), \mathbf{B}^{\mathbb{G}}([0, h, 1]^T), \mathbf{B}^{\mathbb{G}}([w, h, 1]^T)\}$$

where w and h are the width and height of the projector image, respectively.

The x - and y -axes correspond to edges of the projector's image plane. Thus, they can be computed as follows:

$$\begin{aligned}\mathbf{X}^{\mathbb{G}} &= \text{norm}(\mathbf{B}^{\mathbb{G}}([w, 0, 1]^T) - \mathbf{B}^{\mathbb{G}}([0, 0, 1]^T)) \\ \mathbf{Y}^{\mathbb{G}} &= \text{norm}(\mathbf{B}^{\mathbb{G}}([0, h, 1]^T) - \mathbf{B}^{\mathbb{G}}([0, 0, 1]^T))\end{aligned}$$

(While the x -axis is not used directly, it can be a useful debugging aid.)

The z -axis of the projector $\mathbf{P}^{\mathbb{G}}$ may not be aligned with the center of the image plane; in practice, this is true for our projectors, which use off-axis projection. Thus, for the direction of the z -axis, we compute the principal ray of $\mathbf{P}^{\mathbb{G}}$

$$\mathbf{Z}^{\mathbb{G}} = \text{norm}\left(\det(\mathbf{M}^{\mathbb{G}})(\mathbf{m}^{\mathbb{G}})^{3T}\right)$$

where $(\mathbf{m}^{\mathbb{G}})^{3T}$ is the third row of $\mathbf{M}^{\mathbb{G}}$ [64].

To set the virtual camera's orientation, we use Unity's `LookAt` function, which specifies the camera's forward- and up-vectors as points within graphics space \mathbb{G} . For the forward-vector point $\mathbf{F}^{\mathbb{G}}$, we simply add the camera's position $\mathbf{C}^{\mathbb{G}}$ to the principal ray of $\mathbf{P}^{\mathbb{G}}$:

$$\mathbf{F}^{\mathbb{G}} = \mathbf{C}^{\mathbb{G}} + \mathbf{Z}^{\mathbb{G}}$$

The up-vector $\mathbf{U}^{\mathbb{G}}$ is exactly equal to the y -axis of the projector's image plane:

$$\mathbf{U}^{\mathbb{G}} = \mathbf{Y}^{\mathbb{G}}$$

Both the forward- and up-vectors must have their x -components negated to account for Unity's left-handedness.

At this point, we could construct the frustum of the projector in the graphics space \mathbb{G} . However, the final projection matrix we need to apply to the virtual Unity camera must be in the *local space* of the camera, denoted \mathbb{L} .

3. Accordingly, we **transform the projector calibration matrix $\mathbf{P}^{\mathbb{G}}$ in graphics space \mathbb{G} to an equivalent matrix $\mathbf{P}^{\mathbb{L}}$ in local space \mathbb{L}** . In local space, the camera is positioned at the origin $[0, 0, 0]^T$. Likewise, the camera's forward-vector should correspond to the local space z -axis $[0, 0, 1]^T$, and the camera's up-vector should correspond to the local space y -axis $[0, 1, 0]^T$. To satisfy these constraints, the projection matrix $\mathbf{P}^{\mathbb{L}}$ should have the identity matrix for its rotation matrix and a zero-vector for its translation—that is, $\mathbf{R}^{\mathbb{L}} = \mathbf{I}$ and $\mathbf{t}^{\mathbb{L}} = \mathbf{0}$. Thus, we have

$$\mathbf{P}^{\mathbb{L}} = \mathbf{K}[\mathbf{I} \mid \mathbf{0}]$$

4. Finally, we **compute the frustum in local space**, from which we will compute the graphical projection matrix \mathbf{U} for the virtual camera in Unity. Following the standard format of the perspective projection matrix [131], the near plane is composed of the points

$$\{(l, b, n), (l, t, n), (r, b, n), (r, t, n)\}$$

From these points, we will construct a perspective projection matrix of the form

$$\mathbf{U} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

where n and f refer to the z -coordinates of the near and far clipping planes, respectively, in \mathbb{L} .¹ To find the x -coordinates l and r and the y -coordinates b and t , we back-project the corner pixels of the projector's image plane to 3D rays in local space \mathbb{L} . These rays must have z -coordinates equal to n . Let $\mathbf{B}^{\mathbb{L}}(\mathbf{x}, \mu)$ be the parameterized back-projection function in \mathbb{L} . We wish to compute μ_n such that $\mathbf{B}^{\mathbb{L}}(\mathbf{x}, \mu_n)$ has z -coordinate equal to n .

Let the x -coordinate of $\mathbf{B}^{\mathbb{L}}(\mathbf{x}, \mu)$ be given by $x^{\mathbb{L}}(\mathbf{x}, \mu)$, the y -coordinate by $y^{\mathbb{L}}(\mathbf{x}, \mu)$, and the z -coordinate by $z^{\mathbb{L}}(\mathbf{x}, \mu)$. In other words,

$$\mathbf{B}^{\mathbb{L}}(\mathbf{x}, \mu) = \begin{bmatrix} (\mathbf{M}^{\mathbb{L}})^{-1}(\mu\mathbf{x} - \mathbf{p}_4^{\mathbb{L}}) \\ 1 \end{bmatrix} = \begin{bmatrix} (\mathbf{m}_{\mathbb{L}}^{-1})^{1T} \cdot (\mu\mathbf{x} - \mathbf{p}_4^{\mathbb{L}}) \\ (\mathbf{m}_{\mathbb{L}}^{-1})^{2T} \cdot (\mu\mathbf{x} - \mathbf{p}_4^{\mathbb{L}}) \\ (\mathbf{m}_{\mathbb{L}}^{-1})^{3T} \cdot (\mu\mathbf{x} - \mathbf{p}_4^{\mathbb{L}}) \\ 1 \end{bmatrix} = \begin{bmatrix} x^{\mathbb{L}}(\mathbf{x}, \mu) \\ y^{\mathbb{L}}(\mathbf{x}, \mu) \\ z^{\mathbb{L}}(\mathbf{x}, \mu) \\ 1 \end{bmatrix}$$

where $(\mathbf{m}_{\mathbb{L}}^{-1})^{iT}$ is the i th row of $(\mathbf{M}^{\mathbb{L}})^{-1}$ and \cdot represents the dot product. It follows that

$$z^{\mathbb{L}}(\mathbf{x}, \mu) = (\mathbf{m}_{\mathbb{L}}^{-1})^{3T} \cdot (\mu\mathbf{x} - \mathbf{p}_4^{\mathbb{L}})$$

Note that $\mathbf{P}^{\mathbb{L}} = \mathbf{K}[\mathbf{I} \mid \mathbf{0}]$, and so $\mathbf{p}_4^{\mathbb{L}} = \mathbf{0}$:

$$z^{\mathbb{L}}(\mathbf{x}, \mu) = (\mathbf{m}_{\mathbb{L}}^{-1})^{3T} \cdot \mu\mathbf{x}$$

Solving for μ :

$$\mu = \frac{z^{\mathbb{L}}(\mathbf{x}, \mu)}{(\mathbf{m}_{\mathbb{L}}^{-1})^{3T} \cdot \mathbf{x}}$$

¹Note that camera points in Unity are located in the positive z -axis, and so we use n instead of $-n$ as the z -coordinate for these points.

Thus, the value for μ_n such that the z -coordinate of the back-projection is n is given by

$$\mu_n = \frac{n}{(\mathbf{m}_{\mathbb{L}}^{-1})^{3T} \cdot \mathbf{x}}$$

Since the projection matrix $\mathbb{P}^{\mathbb{L}}$ is located at the origin, we can use any pixel \mathbf{x} in the projector's image plane to compute μ_n . As we are computing the positions of the frustum, we will again use the corner pixels for these computations. We compute

$$\{\mathbf{B}^{\mathbb{L}}([0, 0, 1]^T, \mu_n), \mathbf{B}^{\mathbb{L}}([w, 0, 1]^T, \mu_n), \mathbf{B}^{\mathbb{L}}([0, h, 1]^T, \mu_n), \mathbf{B}^{\mathbb{L}}([w, h, 1]^T, \mu_n)\}$$

The left edge of the frustum is composed of the back-projections of the pixels along the left edge of the projector: $[0, 0, 1]^T$ and $[0, h, 1]^T$. To compute the x -coordinate l , we could take the x -coordinates of either back-projection; instead, to improve stability, we use their means. With an analogous construction for the right, bottom, and top edges of the frustum, we have:

$$\begin{aligned} l &= \frac{1}{2} \left(x^{\mathbb{L}}([0, 0, 1]^T, \mu_n) + x^{\mathbb{L}}([0, h, 1]^T, \mu_n) \right) \\ r &= \frac{1}{2} \left(x^{\mathbb{L}}([w, 0, 1]^T, \mu_n) + x^{\mathbb{L}}([w, h, 1]^T, \mu_n) \right) \\ b &= \frac{1}{2} \left(y^{\mathbb{L}}([0, 0, 1]^T, \mu_n) + y^{\mathbb{L}}([w, 0, 1]^T, \mu_n) \right) \\ t &= \frac{1}{2} \left(y^{\mathbb{L}}([0, h, 1]^T, \mu_n) + y^{\mathbb{L}}([w, h, 1]^T, \mu_n) \right) \end{aligned}$$

From these, we can compute the projection matrix for Unity. Since the Unity coordinate system is left-handed, we want to reverse the x -coordinates. This can be achieved by swapping l and r with $-r$ and $-l$, respectively, or by multiplying by an identity matrix with a -1 in

position (1, 1):

$$U = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

LIST OF REFERENCES

- [1] 80/20 Inc. - T-slotted aluminum framing system. <https://www.8020.net>. Accessed: 2018-07-06.
- [2] AAXA P300 pico projector. http://www.aaxatech.com/products/p300_pico_projector.htm, 2019. Accessed: 2019-06-01.
- [3] O. Ariza, P. Lubos, F. Steinicke, and G. Bruder. Ring-shaped haptic device with vibrotactile feedback patterns to support natural spatial interaction. In *Proceedings of the 25th International Conference on Artificial Reality and Telexistence and 20th Eurographics Symposium on Virtual Environments*, pages 175–181. Eurographics Association, 2015.
- [4] R. Arsenault and C. Ware. Eye-hand co-ordination with force feedback. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 408–414. ACM, 2000.
- [5] Autodesk. Maya — computer animation & modeling software — Autodesk. <https://www.autodesk.com/products/maya/overview>, 2018. Accessed: 2018-07-19.
- [6] F. Bacim, M. Sinclair, and H. Benko. Understanding touch selection accuracy on flat and hemispherical deformable surfaces. In *Proceedings of Graphics Interface*, pages 197–204. Canadian Information Processing Society, 2013.
- [7] H. Benko, A. D. Wilson, and R. Balakrishnan. Sphere: Multi-touch interactions on a spherical display. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, pages 77–86. ACM, 2008.

- [8] H. Benko, A. D. Wilson, and P. Baudisch. Precise selection techniques for multi-touch screens. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1263–1272. ACM, 2006.
- [9] P. Bergström. Iterative closest point method. MATLAB Central File Exchange, 2016. Accessed: 2019-05-12.
- [10] J. Beskow, C. E. Peters, G. Castellano, C. O’Sullivan, I. Leite, and S. Kopp, editors. *Intelligent Virtual Agents - 17th International Conference, IVA 2017, Stockholm, Sweden, August 27-30, 2017, Proceedings*, volume 10498 of *Lecture Notes in Computer Science*. Springer, 2017.
- [11] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, Feb 1992.
- [12] X. Bi, Y. Li, and S. Zhai. Fitts law: Modeling finger touch with Fitts’ law. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1363–1372. ACM, 2013.
- [13] O. Bimber and R. Raskar. *Spatial Augmented Reality: Merging Real and Virtual Worlds*. AK Peters/CRC Press, 2005.
- [14] F. Biocca, C. Harms, and J. K. Burgoon. Toward a more robust theory and measure of social presence: Review and suggested criteria. *Presence: Teleoperators and Virtual Environments*, 12(5):456–480, 2003.
- [15] Blackfly S GigE. <https://www.flir.com/products/blackfly-s-gige/>, 2019. Accessed: 2019-06-01.

- [16] R. Bodor, A. Drenner, P. Schrater, and N. Papanikolopoulos. Optimal camera placement for automated surveillance tasks. *Journal of Intelligent and Robotic Systems*, 50(3):257–295, Nov 2007.
- [17] J. Bolton, K. Kim, and R. Vertegaal. A comparison of competitive and cooperative task performance using spherical and flat displays. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pages 529–538. ACM, 2012.
- [18] Boost C++ libraries. <https://www.boost.org>, 2019. Accessed: 2019-05-12.
- [19] J.-Y. Bouquet. Camera calibration toolbox for Matlab. http://www.vision.caltech.edu/bouquetj/calib_doc/index.html, 2015. Accessed: 2019-5-22.
- [20] G. Bradski. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*, 2000.
- [21] E. Brockmeyer, I. Poupyrev, and S. Hudson. PAPILLON: Designing curved display surfaces with printed optics. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, pages 457–462. ACM, 2013.
- [22] J. Brooke. SUS: A ‘quick and dirty’ usability scale. In P. Jordan, B. Thomas, I. McClelland, and B. Weerdmeester, editors, *Usability Evaluation in Industry*, pages 189–194. Taylor & Francis, 1996.
- [23] G. Bruder, F. Steinicke, and W. Stuerzlinger. Touching the void revisited: Analyses of touch behavior on and above tabletop surfaces. In *IFIP Conference on Human-Computer Interaction*, pages 278–296. Springer, 2013.
- [24] G. Bruder, F. Steinicke, and W. Sturzlinger. To touch or not to touch?: Comparing 2D touch and 3D mid-air interaction on stereoscopic tabletop surfaces. In *Proceedings of the 1st Symposium on Spatial User Interaction*, pages 9–16. ACM, 2013.

- [25] W. Buxton, R. Hill, and P. Rowley. Issues and techniques in touch-sensitive tablet input. *ACM SIGGRAPH Computer Graphics*, 19(3):215–224, 1985.
- [26] J. Canny. A computational approach to edge detection. In *Readings in Computer Vision*, pages 184–203. Elsevier, 1987.
- [27] X. Cao, A. D. Wilson, R. Balakrishnan, K. Hinckley, and S. E. Hudson. ShapeTouch: Leveraging contact shape on interactive surfaces. In *3rd IEEE International Workshop on Horizontal Interactive Human Computer Systems*, pages 129–136. IEEE, 2008.
- [28] J. H. Chuah, A. Robb, C. White, A. Wendling, S. Lampotang, R. Kopper, and B. Lok. Increasing agent physicality to raise social presence and elicit realistic behavior. In *2012 IEEE Virtual Reality Short Papers and Posters (VRW)*, pages 19–22. IEEE, 2012.
- [29] J. H. Chuah, A. Robb, C. White, A. Wendling, S. Lampotang, R. Kopper, and B. Lok. Exploring agent physicality and social presence for medical team training. *Presence: Teleoperators and Virtual Environments*, 22(2):141–170, 2013.
- [30] C. J. Clopper and E. S. Pearson. The use of confidence or fiducial limits illustrated in the case of the binomial. *Biometrika*, pages 404–413, 1934.
- [31] CMake. <https://cmake.org/>, 2019. Accessed: 2019-05-12.
- [32] E. Costanza and J. Robinson. A region adjacency tree approach to the detection and design of fiducials. In P. Hall and P. Willis, editors, *Vision, Video, and Graphics (VVG) 2003*. The Eurographics Association, 2003.
- [33] S. Daher, L. Gonzalez, and G. Welch. Poster: Preliminary assessment of neurologic symptomatology using an interactive physical-virtual head with touch. In *17th International Meeting on Simulation in Healthcare (IMSH)*, 2016.

- [34] S. Daher, J. Hochreiter, N. Norouzi, L. Gonzalez, G. Bruder, and G. Welch. Physical-virtual agents for healthcare simulation. In *Proceedings of the 18th International Conference on Intelligent Virtual Agents*, pages 99–106. ACM, 2018.
- [35] S. Daher, J. Hochreiter, N. Norouzi, R. Schubert, G. Bruder, L. Gonzalez, M. Anderson, D. Diaz, J. Cendan, and G. Welch. Poster: Matching vs. non-matching visuals and shape for embodied virtual healthcare agents. *IEEE VR*, 2019.
- [36] S. Daher, J. Hochreiter, R. Schubert, L. Gonzalez, J. Cendan, M. Anderson, D. Diaz, and G. Welch. The physical-virtual patient simulator: A physical human form with virtual appearance and behavior. *Simulation in Healthcare*, 2019. *To appear*.
- [37] N. Dahlbäck, A. Jönsson, and L. Ahrenberg. Wizard of Oz studies—Why and how. *Knowledge-Based Systems*, 6(4):258–266, 1993.
- [38] C. T. Dang, M. Straub, and E. André. Hand distinction for multi-touch tabletop interaction. In *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, pages 101–108. ACM, 2009.
- [39] T. Dang, C. Hoffmann, and C. Stiller. Continuous stereo self-calibration by camera parameter tracking. *IEEE Transactions on Image Processing*, 18(7):1536–1550, 2009.
- [40] P. L. Davidson and J. Y. Han. Extending 2D object arrangement with pressure-sensitive layering cues. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, pages 87–90. ACM, 2008.
- [41] G. De Leo, L. A. Diggs, E. Radici, and T. W. Mastaglio. Measuring sense of presence and user characteristics to predict effective training in an online simulated virtual environment. *Simulation in Healthcare*, 9(1):1–6, 2014.

- [42] J. D’Errico. Surface fitting using gridfit. MATLAB Central File Exchange, 2016. Accessed: 2019-05-12.
- [43] P. Dietz and D. Leigh. DiamondTouch: A multi-user touch technology. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, pages 219–226. ACM, 2001.
- [44] K. Dohse, T. Dohse, J. D. Still, and D. J. Parkhurst. Enhancing multi-user interaction with multi-touch tabletop displays using hand tracking. In *First International Conference on Advances in Computer-Human Interaction*, pages 297–302. IEEE, 2008.
- [45] R. Downs. Using resistive touch screens for human/machine interface. *Analog Applications Journal Q*, 3:5–9, 2005.
- [46] F. Echtler, M. Huber, and G. Klinker. Shadow tracking on multi-touch tables. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 388–391. ACM, 2008.
- [47] J. Edelmann, P. Gerjets, P. Mock, A. Schilling, and W. Strasser. Face2Face - A system for multi-touch collaboration with telepresence. In *IEEE International Conference on Emerging Signal Processing Applications*, pages 159–162. IEEE, 2012.
- [48] F. N. Eventoff. Electronic pressure sensitive transducer apparatus, Feb. 2 1982. US Patent 4,314,227.
- [49] P. M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381, 1954.
- [50] S. Fleishman, D. Cohen-Or, and D. Lischinski. Automatic camera placement for image-based modeling. In *Proceedings of the Seventh Pacific Conference on Computer Graphics and Applications*, pages 12–20, Oct 1999.

- [51] J. L. Fleiss, B. Levin, and M. C. Paik. *Statistical Methods for Rates and Proportions*. John Wiley & Sons, 2013.
- [52] FlyCapture SDK. <https://www.flir.com/products/flycapture-sdk/>, 2019. Accessed: 2019-05-12.
- [53] I. Fujieda and H. Haga. Fingerprint input based on scattered-light detection. *Applied Optics*, 36(35):9152–9156, 1997.
- [54] M. Fulkerson. *The First Sense: A Philosophical Study of Human Touch*. MIT Press, 2014.
- [55] M. J. Garside, M. P. Rudd, and C. I. Price. Stroke and TIA assessment training: A new simulation-based approach to teaching acute stroke assessment. *Simulation in Healthcare*, 7(2):117–122, 2012.
- [56] E. Goffman. *Behavior in Public Places: Notes on the Social Organization of Gatherings*. Simon and Schuster, 1963.
- [57] L. Gonzalez, S. Daher, J. Hochreiter, and G. Welch. Student nursing assessment of discrete neurology symptoms using an interactive physical virtual head. *Presentation at International Nursing Association for Clinical Simulation and Learning*, 2016.
- [58] R. Greene. The drawing prism: A versatile graphic input device. *ACM SIGGRAPH Computer Graphics*, 19(3):103–110, 1985.
- [59] J. Gu and G. Lee. TouchString: A flexible linear multi-touch sensor for prototyping a freeform multi-touch surface. In *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology*, pages 75–76. ACM, 2011.
- [60] E. T. Hall. A system for the notation of proxemic behavior. *American Anthropologist*, 65(5):1003–1026, 1963.

- [61] J. Y. Han. Low-cost multi-touch sensing through frustrated total internal reflection. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, pages 115–118. ACM, 2005.
- [62] C. Harms and F. Biocca. Internal consistency and reliability of the networked minds measure of social presence. In *Seventh Annual International Presence Workshop: Presence 2004*, pages 246–251, 2004.
- [63] S. G. Hart and L. E. Staveland. Development of NASA-TLX (task load index): Results of empirical and theoretical research. In *Advances in Psychology*, volume 52, pages 139–183. Elsevier, 1988.
- [64] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.
- [65] M. J. Hertenstein, R. Holmes, M. McCullough, and D. Keltner. The communication of emotion via touch. *Emotion*, 9(4):566, 2009.
- [66] M. J. Hertenstein, D. Keltner, B. App, B. A. Bulleit, and A. R. Jaskolka. Touch communicates distinct emotions. *Emotion*, 6(3):528, 2006.
- [67] M. J. Hertenstein, J. M. Verkamp, A. M. Kerestes, and R. M. Holmes. The communicative functions of touch in humans, nonhuman primates, and rats: A review and synthesis of the empirical research. *Genetic, Social, and General Psychology Monographs*, 132(1):5–94, 2006.
- [68] W. D. Hillis. A high-resolution imaging touch sensor. *The International Journal of Robotics Research*, 1(2):33–44, 1982.

- [69] J. Hochreiter. Optical touch sensing on non-parametric rear-projection surfaces. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 805–806. IEEE, March 2018.
- [70] J. Hochreiter, S. Daher, G. Bruder, and G. Welch. Cognitive and touch performance effects of mismatched 3D physical and visual perceptions. In *IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 379–386. IEEE, March 2018.
- [71] J. Hochreiter, S. Daher, A. Nagendran, L. Gonzalez, and G. Welch. Touch sensing on non-parametric rear-projection surfaces: A physical-virtual head for hands-on healthcare training. In *IEEE Conference on Virtual Reality (VR)*, pages 69–74. IEEE, 2015.
- [72] J. Hochreiter, S. Daher, A. Nagendran, L. Gonzalez, and G. Welch. Optical touch sensing on nonparametric rear-projection surfaces for interactive physical-virtual experiences. *Presence: Teleoperators and Virtual Environments*, 25(1):33–46, 2016.
- [73] D. Holman and R. Vertegaal. TactileTape: Low-cost touch sensing on curved surfaces. In *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology*, pages 17–18. ACM, 2011.
- [74] S. A. Iacolina, A. Soro, and R. Scateni. Improving FTIR based multi-touch sensors with IR shadow tracking. In *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pages 241–246. ACM, 2011.
- [75] ISO 9241-9:2000 – Ergonomic requirements for office work with visual display terminals (VDTs) – Part 9: Requirements for non-keyboard input devices, 2000.
- [76] S. P. Jobs, S. Forstall, G. Christie, S. O. Lemay, S. Herz, M. Van Os, B. Ording, G. Novick, W. C. Westerman, I. Chaudhri, et al. Touch screen device, method, and graphical user interface for determining commands by applying heuristics, Jan. 20 2009. US Patent 7,479,949.

- [77] P. Joguet and G. Largillier. Devices and methods of controlling manipulation of virtual objects on a multi-contact tactile screen, Nov. 1 2011. US Patent 8,049,730.
- [78] K. Johnsen, D. Beck, and B. Lok. The impact of a mixed reality display configuration on user behavior with a virtual human. In *International Conference on Intelligent Virtual Agents*, pages 42–48. Springer, 2010.
- [79] R. G. Johnson and D. Fryberger. Touch actuatable data input panel assembly, June 27 1972. US Patent 3,673,327.
- [80] S. E. Jones and A. E. Yarbrough. A naturalistic study of the meanings of touch. *Communications Monographs*, 52(1):19–56, 1985.
- [81] K. Kamiyama, K. Vlack, T. Mizota, H. Kajimoto, K. Kawakami, and S. Tachi. Vision-based sensor for real-time measuring of surface traction fields. *IEEE Computer Graphics and Applications*, 25(1):68–75, 2005.
- [82] K. Karpa, C. Pinto, A. Possanza, J. Dos Santos, M. Snyder, A. Salvadia, D. Panchik, R. Myers, M. Fink, and A. Dunlap. Stroke simulation activity: A standardized patient case for interprofessional student learning. *MedEdPORTAL*, 14(10698):1–8, March 2018.
- [83] L. R. Kasday. Touch position sensitive surface, Nov. 20 1984. US Patent 4,484,179.
- [84] D. King, R. Morton, and C. Bevan. How to use capillary refill time. *Archives of Disease in Childhood - Education and Practice*, 99(3):111–116, 2014.
- [85] T. R. Knapp. Treating ordinal scales as interval scales: An attempt to resolve the controversy. *Nursing Research*, 39(2):121–123, 1990.
- [86] L. Kohli, M. C. Whitton, and F. P. Brooks. Redirected touching: Training and adaptation in warped virtual spaces. In *IEEE Symposium on 3D User Interfaces (3DUI)*, pages 79–86. IEEE, 2013.

- [87] P. D. Kovesi. MATLAB and Octave functions for computer vision and image processing. <http://www.peterkovesi.com/matlabfns>, 2019. Accessed: 2019-05-22.
- [88] W. M. Kuzon Jr., M. G. Urbanchek, and S. McCabe. The seven deadly sins of statistical analysis. *Annals of Plastic Surgery*, 37(3):265–272, 1996.
- [89] K. M. Lee, Y. Jung, J. Kim, and S. R. Kim. Are physically embodied social agents better than disembodied social agents?: The effects of physical embodiment, tactile interaction, and people’s loneliness in human–robot interaction. *International Journal of Human-Computer Studies*, 64(10):962–973, 2006.
- [90] S. Lee, W. Buxton, and K. C. Smith. A multi-touch three dimensional touch-sensitive tablet. *SIGCHI Bulletin*, 16(4):21–25, Apr. 1985.
- [91] D. Legland. geom3d: 3D geometry toolbox. MATLAB Central File Exchange, 2018. Accessed: 2019-5-25.
- [92] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly of Applied Mathematics*, 2(2):164–168, 1944.
- [93] A. I. Levine, S. DeMaria Jr., A. D. Schwartz, and A. J. Sim. *The Comprehensive Textbook of Healthcare Simulation*. Springer Science & Business Media, 2013.
- [94] B. Lok, J. H. Chuah, A. Robb, A. Cordar, S. Lampotang, A. Wendling, and C. White. Mixed-reality humans for team training. *IEEE Computer Graphics and Applications*, 34(3):72–75, 2014.
- [95] M. Lombard and T. Ditton. At the heart of it all: The concept of presence. *Journal of Computer-Mediated Communication*, 3(2), 1997.

- [96] J. Lopreiato, D. Downing, W. Gammon, L. Lioce, B. Sittner, V. Slot, A. Spain, et al. *Health-care Simulation Dictionary*. Agency for Healthcare Research and Quality, Rockville, MD, 2016.
- [97] M. I. Lourakis and A. A. Argyros. SBA: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):2, 2009.
- [98] C. F. Malacaria. A thin, flexible, matrix-based pressure sensor. *Sensors*, pages 102–104, 1998.
- [99] S. Malik and J. Laszlo. Visual touchpad: A two-handed gestural input device. In *Proceedings of the 6th International Conference on Multimodal Interfaces*, pages 289–296. ACM, 2004.
- [100] J. B. Mallos. Touch position sensitive surface, Aug. 24 1982. US Patent 4,346,376.
- [101] MATLAB. <https://www.mathworks.com/products/matlab.html>, 2019. Accessed: 2019-05-12.
- [102] MATLAB Optimization Toolbox. <https://www.mathworks.com/products/optimization.html>, 2019. Accessed: 2019-05-22.
- [103] N. Matsushita and J. Rekimoto. HoloWall: Designing a finger, hand, body, and object sensitive wall. In *Proceedings of the 10th Annual ACM Symposium on User Interface Software and Technology*, pages 209–210. ACM, 1997.
- [104] M. L. McLaughlin, G. Sukhatme, and J. Hespanha. *Touch in Virtual Environments: Haptics and the Design of Interactive Systems*. Prentice Hall PTR, 2001.
- [105] Microsoft HoloLens. <https://www.microsoft.com/en-us/hololens>, 2019. Accessed: 2019-05-30.

- [106] Mixed reality companion kit. <https://github.com/Microsoft/MixedRealityCompanionKit>. Accessed: 2017-09-07.
- [107] T. Möller and B. Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1):21–28, Oct. 1997.
- [108] A. Montagu. *Touching: The Human Significance of The Skin*. Columbia University Press New York, 1971.
- [109] R. Mueller. Direct television drawing and image manipulating system, Nov. 5 1974. US Patent 3,846,826.
- [110] A. W. Ng and A. H. Chan. Finger response times to visual, auditory and tactile modality stimuli. In *Proceedings of the International Multiconference of Engineers and Computer Scientists*, volume 2, pages 1449–1454, 2012.
- [111] A. Nguyen and A. Banic. 3DTouch: A wearable 3D input device for 3D applications. In *IEEE Virtual Reality (VR)*, pages 55–61. IEEE, 2015.
- [112] K. Nicol and E. M. Hennig. Apparatus for the time-dependent measurement of physical quantities, Jan. 9 1979. US Patent 4,134,063.
- [113] N. Norouzi, K. Kim, J. Hochreiter, M. Lee, S. Daher, G. Bruder, and G. Welch. A systematic survey of 15 years of user studies published in the intelligent virtual agents conference. In *Proceedings of the 18th International Conference on Intelligent Virtual Agents*, pages 17–22. ACM, 2018.
- [114] OptiTrack - Motion Capture Systems. <https://optitrack.com/>, 2019. Accessed: 2019-05-19.
- [115] PeopleVisionFX. <http://peoplevisionfx.com>, 2019. Accessed: 2019-05-29.

- [116] H. Perski and M. Morag. Dual function input device and method, July 13 2004. US Patent 6,762,752.
- [117] M. Piccardi. Background subtraction techniques: A review. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 3099–3104. IEEE, 2004.
- [118] I. Poupyrev, S. Maruyama, and J. Rekimoto. Ambient touch: Designing tactile interfaces for handheld devices. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology*, pages 51–60. ACM, 2002.
- [119] D. Prattichizzo, F. Chinello, C. Pacchierotti, and K. Minamizawa. RemoTouch: A system for remote touch experience. In *19th International Symposium in Robot and Human Interactive Communication*, pages 676–679. IEEE, 2010.
- [120] L. Qian, E. Azimi, P. Kazanzides, and N. Navab. Comprehensive tracker based display calibration for holographic optical see-through head-mounted display. *arXiv:1703.05834v1*, 2017.
- [121] Recognizing stroke early - Harvard Health. <https://www.health.harvard.edu/heart-health/recognizing-stroke-early>, January 2017. Accessed: 2018-07-07.
- [122] J. Rekimoto. SmartSkin: An infrastructure for freehand manipulation on interactive surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 113–120. ACM, 2002.
- [123] D. J. Rivera-Gutierrez, G. Welch, P. Lincoln, M. C. Whitton, J. Cendan, D. A. Chesnutt, H. Fuchs, and B. H. Lok. Shader lamps virtual patients: The physical manifestation of virtual patients. *Studies in Health Technology and Informatics*, 173:372–8, 2012.

- [124] I. Rosenberg and K. Perlin. The UnMousePad: An interpolating multi-touch force-sensing input pad. *ACM Transactions on Graphics (TOG)*, 28(3):65, 2009.
- [125] A. Roudaut, H. Pohl, and P. Baudisch. Touch input on curved surfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1011–1020. ACM, 2011.
- [126] J. Salvi, J. Pagès, and J. Batlle. Pattern codification strategies in structured light systems. *Pattern Recognition*, 37(4):827–849, 2004.
- [127] E. Sanders, R. de Keizer, and D. Zee. *Eye Movement Disorders*. Monographs in Ophthalmology. Springer Netherlands, 2012.
- [128] S. Schätzle, T. Ende, T. Wüsthoff, and C. Preusche. VibroTac: An ergonomic and versatile usable vibrotactile feedback device. In *19th International Symposium in Robot and Human Interactive Communication*, pages 670–675. IEEE, 2010.
- [129] J. Schöning, J. Hook, T. Bartindale, D. Schmidt, P. Oliver, F. Echtler, N. Motamedi, P. Brandl, and U. von Zadow. Building interactive multi-touch surfaces. In *Tabletops - Horizontal Interactive Displays*, pages 27–49. Springer, 2010.
- [130] J. Schöning, F. Steinicke, A. Krüger, K. Hinrichs, and D. Valkov. Bimanual interaction with interscopic multi-touch surfaces. In *IFIP Conference on Human-Computer Interaction*, pages 40–53. Springer, 2009.
- [131] M. Segal and K. Akeley. The OpenGL graphics system: A specification (version 1.1), 1999.
- [132] T. L. Ssergejewitsch. Method of and apparatus for the generation of sounds, Feb. 28 1928. US Patent 1,661,058.

- [133] A. Stevenson, C. Perez, and R. Vertegaal. An inflatable hemispherical multi-touch display. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, pages 289–292. ACM, 2011.
- [134] Stroke warning signs and symptoms. https://www.strokeassociation.org/STROKEORG/WarningSigns/Stroke-Warning-Signs-and-Symptoms_UCM_308528_SubHomePage.jsp. Accessed: 2018-07-07.
- [135] K. Tanie, K. Komoriya, M. Kaneko, S. Tachi, and A. Fujikawa. A high resolution tactile sensor. In *Proceedings of the 4th International Conference on Robot Vision and Sensory Controls*, pages 251–260, 1984.
- [136] R. J. Teather, D. Natapov, and M. Jenkin. Evaluating haptic feedback in virtual environments using ISO 9241–9. In *IEEE Virtual Reality Conference (VR)*, pages 307–308. IEEE, 2010.
- [137] R. J. Teather, A. Pavlovych, W. Stuerzlinger, and I. S. MacKenzie. Effects of tracking technology, latency, and spatial jitter on object movement. In *Proceedings of the IEEE Symposium on 3D User Interfaces (3DUI)*, pages 43–50. IEEE, 2009.
- [138] R. J. Teather and W. Stuerzlinger. Pointing at 3D targets in a stereo head-tracked virtual environment. In *Proceedings of the IEEE Symposium on 3D User Interfaces (3DUI)*, pages 87–94. IEEE, 2011.
- [139] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon. Bundle adjustment – A modern synthesis. In *International Workshop on Vision Algorithms*, pages 298–372. Springer, 1999.
- [140] M. Tuceryan, Y. Genc, and N. Navab. Single-point active alignment method (SPAAM) for optical see-through HMD calibration for augmented reality. *Presence: Teleoperators and Virtual Environments*, 11(3):259–276, 2002.

- [141] Unity. <https://unity3d.com>, 2018. Accessed: 2018-07-19.
- [142] D. Valkov, F. Steinicke, G. Bruder, and K. Hinrichs. 2D touching of 3D stereoscopic objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1353–1362. ACM, 2011.
- [143] N. Villar, S. Izadi, D. Rosenfeld, H. Benko, J. Helmes, J. Westhues, S. Hodges, E. Ofek, A. Butler, X. Cao, et al. Mouse 2.0: Multi-touch meets the mouse. In *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, pages 33–42. ACM, 2009.
- [144] O. M. Watson. *Proxemic Behavior: A Cross-Cultural Study*, volume 8 of *Approaches to Semiotics*. Mouton, 1970.
- [145] M. Weiss, S. Voelker, C. Sutter, and J. Borchers. BendDesk: Dragging across the curve. In *ACM International Conference on Interactive Tabletops and Surfaces*, pages 1–10. ACM, 2010.
- [146] G. Welch, S. Daher, J. Hochreiter, and L. Gonzalez. Poster: Interactive rear-projection physical-virtual patient simulators. In *22nd Medicine Meets Virtual Reality (NextMed / MMVR)*, Los Angeles, CA, USA, 2016.
- [147] D. Wessel, R. Avizienis, A. Freed, and M. Wright. A force sensitive multi-touch array supporting multiple 2-D musical control structures. In *Proceedings of the 7th International Conference on New Interfaces for Musical Expression (NIME)*, pages 41–45. ACM, 2007.
- [148] W. Westerman. *Hand tracking, finger identification, and chordic manipulation on a multi-touch surface*. PhD thesis, University of Delaware, 1999.
- [149] W. Westerman and J. G. Elias. Method and apparatus for integrating manual input, Nov. 27 2001. US Patent 6,323,846.

- [150] W. Westerman, J. G. Elias, and A. Hedge. Multi-touch: A new tactile 2-D gesture interface for human-computer interaction. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 45(6):632–636, 2001.
- [151] R. M. White. Tactile sensor employing a light conducting element and a resiliently deformable sheet, May 26 1987. US Patent 4,668,861.
- [152] F. N. Willis and H. K. Hamm. The use of interpersonal touch in securing compliance. *Journal of Nonverbal Behavior*, 5(1):49–55, 1980.
- [153] A. D. Wilson. TouchLight: An imaging touch screen and display for gesture-based interaction. In *Proceedings of the 6th International Conference on Multimodal Interfaces*, pages 69–76. ACM, 2004.
- [154] M. Wu and R. Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, pages 193–202. ACM, 2003.
- [155] W. Wyman. Method for optical comparison of skin friction-ridge patterns, Aug. 17 1965. US Patent 3,200,701.
- [156] S. Yohanan and K. E. MacLean. A tool to study affective touch. In *Extended Abstracts on Human Factors in Computing Systems*, pages 4153–4158. ACM, 2009.
- [157] S. Yohanan and K. E. MacLean. The role of affective touch in human-robot interaction: Human intent and expectations in touching the haptic creature. *International Journal of Social Robotics*, 4(2):163–180, 2012.
- [158] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, Nov 2000.