

# YE OLDE GRIMOIRE OF COMPONENTS FOR AUGMENTED REALITY SYSTEMS

WITH EXAMPLES USING UNITY SOFTWARE

SHAWN BANGAY

Ye Olde Grimoire of Components for Augmented Reality Systems

With examples using Unity software

Shaun Bangay

Version 0.8

Copyright ©2022 Deakin University, Geelong, Australia

These materials are not sponsored by or affiliated with Unity Technologies or its affiliates. “Unity” is a trademark or registered trademark of Unity Technologies or its affiliates in the U.S. and elsewhere.

ISBN: 9781070315539

# *Contents*

<b>1 Design for Enhanced Realities</b>	<b>19</b>
1.1 Project . . . . .	19
1.2 An Enhanced Reality Design Strategy . . . . .	20
1.3 Enhanced Reality Design . . . . .	24
1.3.1 Nature of reality . . . . .	24
1.3.2 Location (setting) . . . . .	28
1.3.3 Objects . . . . .	32
1.3.4 Interaction and Feedback . . . . .	38
1.3.5 Concepts explored . . . . .	40
1.3.6 Participant engagement . . . . .	43
1.3.7 Design of experience . . . . .	47
1.3.8 Equipment . . . . .	51
1.4 The Component for Enhanced Reality Design	55
<b>2 Project Management for Creative Technology</b>	<b>57</b>
2.1 Project . . . . .	57
2.2 Agile approaches . . . . .	57
2.2.1 Documentation . . . . .	61
2.3 Building Enhanced Reality Applications . . . . .	65
2.3.1 Building the foundation . . . . .	65
2.3.2 Agile working practices . . . . .	67
2.3.3 Team communication . . . . .	68
2.3.4 Task tracking . . . . .	71
2.3.5 Version control . . . . .	72
2.3.6 Testing . . . . .	75
<b>3 Coordinated Realities</b>	<b>77</b>
3.1 Project . . . . .	77

3.2	Coordinating Reality . . . . .	78
3.3	Mutual Presence . . . . .	78
3.4	Matchmaking . . . . .	80
3.5	Conversation . . . . .	80
3.6	Coordinating participants . . . . .	81
3.7	Cloud Coordination . . . . .	81
<b>4</b>	<b>Visual Tracking</b>	<b>83</b>
4.1	Project . . . . .	83
4.2	Visual Tracking . . . . .	83
4.3	Feature points . . . . .	85
4.4	Finding feature points . . . . .	87
4.4.1	Corner detection . . . . .	88
4.4.2	Feature detection . . . . .	89
4.4.3	Feature Matching . . . . .	90
4.5	Marker Tracking . . . . .	91
4.6	Location Tracking . . . . .	91
<b>5</b>	<b>Location based Mobile Augmented Reality</b>	<b>95</b>
5.1	Project . . . . .	95
5.2	Location (Setting) . . . . .	96
5.3	GPS concepts . . . . .	96
5.4	Working with maps . . . . .	97
5.4.1	Map representations . . . . .	98
<b>6</b>	<b>Sensor systems</b>	<b>99</b>
6.1	Project . . . . .	99
6.2	Sensing . . . . .	100
6.3	Sampling . . . . .	100
6.4	Sensors . . . . .	102
6.4.1	Orientation sensing . . . . .	102
6.5	Signal sensing . . . . .	103
6.6	Scene structure . . . . .	103
<b>7</b>	<b>Interaction</b>	<b>105</b>
7.1	Project . . . . .	105
7.2	Interaction and Feedback . . . . .	105
7.3	Gesture Recognition . . . . .	106
7.4	Visual Recognition . . . . .	106

<b>8 Capture of Physical Content</b>	<b>109</b>
8.1 Project . . . . .	109
8.2 Location Registration . . . . .	110
8.2.1 Transformations . . . . .	111
8.2.2 Achieving Registration . . . . .	112
8.2.3 Point cloud alignment . . . . .	114
8.3 Scanners . . . . .	115
8.3.1 Structured Light . . . . .	116
8.3.2 Phase shift . . . . .	118
8.3.3 Phase Unwrapping . . . . .	120
8.3.4 Coordinate Transformation . . . . .	123
8.4 Scene capture and stitching . . . . .	126
8.4.1 Feature identification . . . . .	128
8.4.2 Correspondence matching . . . . .	131
<b>9 Presentation</b>	<b>133</b>
9.1 Project . . . . .	133
9.2 Presentation . . . . .	133
9.3 Augmented Worlds . . . . .	134
9.4 Scene presentation . . . . .	134
<b>10 Augmentation across Modalities</b>	<b>137</b>
10.1 Project . . . . .	137
10.2 Spatial sound . . . . .	137
10.3 Speech to text . . . . .	138
10.4 Haptics . . . . .	138
<b>11 Extending Legacy Technologies</b>	<b>139</b>
11.1 Project . . . . .	139
11.2 Web Technology . . . . .	139
11.3 WebRTC . . . . .	140
11.4 WebVR . . . . .	141
11.5 WebAR . . . . .	141
<b>12 Usability</b>	<b>143</b>
12.1 Project . . . . .	143
12.2 Usability . . . . .	143
12.3 User experience evaluation . . . . .	144
12.3.1 Preparation for usability evaluation .	145

12.3.2	Conducting the evaluation . . . . .	147
12.3.3	Analyzing the results . . . . .	148
<b>13</b>	<b>Production and Deployment</b>	<b>149</b>
13.1	Project . . . . .	149
13.2	Publishing . . . . .	149
<b>14</b>	<b>Assignments for Enhanced Reality</b>	<b>151</b>
14.1	Understanding requirements . . . . .	151
14.2	Design Sprint . . . . .	155
14.3	Component Innovation . . . . .	160
14.4	Component Customization . . . . .	161
14.5	User Experience Evaluation . . . . .	161
14.6	Pitch . . . . .	163
14.7	Demonstration . . . . .	164
<b>15</b>	<b>Design Components</b>	<b>169</b>
15.1	■ A systematic approach to designing an enhanced reality application . . . . .	169
<b>16</b>	<b>Nature of Reality Components</b>	<b>179</b>
16.1	■ Creating an enhanced reality experience on a mobile platform . . . . .	179
16.2	■ Setting up a virtual reality experience on an Oculus Quest mobile headset . . . . .	192
16.3	■ Deploying an enhanced reality experience on a Daydream . . . . .	199
16.4	■ Presenting 360°panoramic content on a headset . . . . .	205
16.5	■ Presenting experiences using audio rendering . . . . .	211
16.6	■ Providing virtual reality experiences using web technologies . . . . .	216
16.7	■ Presenting augmented reality experiences using web technologies . . . . .	221
<b>17</b>	<b>Location (Setting) Components</b>	<b>225</b>

17.1	■■■ Tracking your location globally using GPS	226
17.2	■■■ Accessing and presenting your setting using map tile services . . . . .	230
17.3	■■■ Reconstructing your setting as a terrain mesh	237
17.4	■■■ Mapping your physical location as a virtual point cloud . . . . .	246
17.5	■■■ Capturing your environment as a 3D mesh with the Kinect . . . . .	261
17.6	■■■ Capturing and presenting scenes using 360° cameras . . . . .	271
17.7	■■■ Capturing cylindrical panoramas of scenes using Cardboard Camera and a mobile phone	278
17.8	■■■ Converting complex virtual scenes to stereo panoramic images using Unity . . . . .	279
17.9	■■■ Converting complex virtual scenes to stereo panoramic images using Blender . . . . .	282
17.10	■■■ Transitioning between realities using portals . . . . .	287
17.11	■■■ Aligning overlapping realities . . . . .	297
<b>18</b>	<b>Objects for Interaction Components</b>	<b>299</b>
18.1	■■■ Object control during testing and debugging . . . . .	299
18.2	■■■ Tracking the user and objects to allow interaction . . . . .	304
18.3	■■■ Sensing the orientation of a device . . . . .	308
18.4	■■■ Synthetic buttons and dials for use in an augmented reality user interface . . . . .	313
18.5	■■■ Identifying and recognising physical objects	319
18.6	■■■ Capturing the structure of objects in the scene using structured light . . . . .	336
18.7	■■■ Aligning multiple point clouds to reconstruct the appearance of objects . . . . .	354

<b>19 Interaction and Feedback Components</b>	<b>379</b>
19.1  Marker tracking with Vuforia . . . . .	379
19.2  Marker tracking with AR Core . . . . .	383
19.3  A handheld controller that can be dynamically adapted to the needs of the experience . . . . .	387
19.4  Interaction using gesture recognition with the Leap Motion . . . . .	400
19.5  Interaction using gesture recognition with the Kinect . . . . .	412
19.6  Managing interaction using body tracking on a general purpose device . . . . .	416
19.7  Tracking facial expression and head pose . . . . .	438
19.8  Supporting interaction using speech recognition . . . . .	458
<b>20 Concepts Explored Components</b>	<b>475</b>
20.1  Frequency domain signal processing applied to create a heart rate monitor . . . . .	475
20.2  Sensory synesthesia and haptic rendering applied to create a vibration display . . . . .	482
<b>21 Participant Engagement Components</b>	<b>497</b>
21.1  Achieving mutual presence for multi-participant experiences by using Photon . . . . .	498
21.2  Coordinating multiple worlds and groups of participants using lobbies and rooms . . . . .	507
21.3  Allowing participants to talk to one another using voice communication . . . . .	526
21.4  Creating a multi-participant experience with video communication using web technologies . . . . .	532
21.5  Customizing the data shared in a multi-participant experience . . . . .	549

21.6	■■■ Enabling persistent augmented reality experiences and shared location recognition using cloud anchors . . . . .	570
21.7	■■■ Legacy multi-participant collaboration using the now obsolete UNet . . . . .	583
<b>22</b>	<b>Design of Experience Components</b>	<b>599</b>
<b>23</b>	<b>Production Components</b>	<b>601</b>
23.1	■■■ Coordinating the efforts of a team of developers using version control . . . . .	601
23.2	■■■ Detecting issues early by writing unit tests	605
23.3	■■■ Publishing enhanced reality experiences for the Daydream . . . . .	612
<b>24</b>	<b>Resources</b>	<b>619</b>
<b>25</b>	<b>Bibliography</b>	<b>621</b>



# *List of Components*

Exercise 14.1.1 : Applying the design component . . . . .	151
Exercise 14.2.1 : Project management using a design sprint. . . . .	155
Exercise 14.3.1 : Innovate. . . . .	160
Exercise 14.4.1 : Contribute. . . . .	161
Exercise 14.5.1 : User experience evaluation. . . . .	161
Exercise 14.6.1 : Pitch. . . . .	163
Exercise 14.7.1 : Demonstration. . . . .	164
Component 15.1.1: Designing an enhanced reality application. . . . .	169
Component 16.1.1: Creating a mobile application with Unity software. . . . .	179
Component 16.2.1: Building applications for mobile virtual reality headsets. . . . .	192
Component 16.3.1: Daydream presentation. . . . .	199
Component 16.4.1: Display of stereoscopic panoramas. . . . .	205
Component 16.5.1: Spatial sound component. . . . .	211
Component 16.6.1: WebVR and A-Frame. . . . .	216
Component 16.7.1: WebAR. . . . .	221
Component 17.1.1: GPS based location. . . . .	226
Component 17.2.1: Retrieving maps. . . . .	230
Component 17.3.1: Reconstructing Terrain Maps	237

Component 17.4.1: Scene tracking with AR Core.	246
Component 17.5.1: Scene capture with Kinect.	261
Component 17.6.1: Scene capture with 360° cameras.	271
Component 17.7.1: Scene capture with Cardboard camera.	278
Component 17.8.1: Augmented scenes with Unity.	279
Component 17.9.1: Augmented scenes with Blender.	282
Component 17.10.1 Portal technology.	287
Component 17.11.1 Aligning worlds	297
Component 18.1.1: Prototyping location tracking component.	299
Component 18.2.1: Location tracking component.	304
Component 18.3.1: Gyroscope based orientation control.	308
Component 18.4.1: Buttons and dials.	313
Component 18.5.1: Visual recognition.	319
Component 18.6.1: Using phase shift structured light to capture objects and scenes	336
Component 18.7.1: Assembling objects from partial scans	354
Component 19.1.1: Marker tracking with Vuforia.	379
Component 19.2.1: Marker tracking with AR Core.	383
Component 19.3.1: Flexible controller.	387
Component 19.4.1: Gesture recognition with Leap Motion.	400
Component 19.5.1: Gesture recognition with the Kinect.	412
Component 19.6.1: Body Tracking.	416
Component 19.7.1: Face Tracking.	438
Component 19.8.1: Speech recognition.	458

Component 20.1.1: <b>Heart rate monitor.</b> . . . . .	475
Component 20.2.1: <b>Vibration display.</b> . . . . .	482
Component 21.1.1: <b>Mutual presence with Photon.</b> . . . . .	498
Component 21.2.1: <b>A Sense of Distributed Presence</b> . . . . .	507
Component 21.3.1: <b>Conversations with Photon Chat</b> . . . . .	526
Component 21.4.1: <b>Video conferencing.</b> . . . . .	532
Component 21.5.1: <b>Shared location.</b> . . . . .	549
Component 21.6.1: <b>Cloud anchors.</b> . . . . .	570
Component 21.7.1: <b>Mutual presence.</b> . . . . .	583
Component 23.1.1: <b>Version control for enhanced reality projects.</b> . . . . .	601
Component 23.2.1: <b>Writing unit tests.</b> . . . . .	605
Component 23.3.1: <b>Google Daydream publication.</b> . . . . .	612

### *About This Book*

A particular category of software applications consist almost exclusively of operations relating to creating, reading, updating and deleting data records. These C.R.U.D. applications represent such generic and mechanically generated functionality that often other opportunities to effectively use the data are overlooked. A similar phenomenon occurs with augmented reality applications where existing tool-kits encourage the form of application that involve only 'Display Object. Appreciate'. The user is expected to jump through several hoops such as installing applications, registering accounts, and taking photographs just to be able to scan a particular marker. The reward is limited to being able to passively view some information or a 3D object overlaid on that marker. Likewise other D.o.A. augmented reality applications, including the 'Dissect Object. Assess' educational tools, fail to achieve the full value inherent in augmented and virtual reality technologies.

The approach emphasized in this book identifies and incorporates multiple dimensions of enhanced reality technology in each application. This process starts at the design stage, ensuring that opportunities relevant to each dimension are explored. Each opportunity is then realized in the form of one or more augmented reality components. Such components are reusable elements common to different classes of augmented reality and mobile virtual reality applications. The following chapters present and explain the use of each of these components through a worked example.

The initial chapters define the design space for such enhanced reality applications, presented in the form of a design framework developed jointly with Sophie McKenzie at Deakin University. Our emphasis is on identifying all the opportunities appropriate to a particular problem context that can be addressed through a considered and coherent selection of appropriate augmented reality technologies. In particular, our philosophy emphasizes the

conceptual issues, setting and social interaction aspects of the experience and leaves the actual selection of particular hardware elements to the end.

For convenience, the following terms are used as described:

- Physical reality: The ‘real’ world representing the reality we experience without technology enhancement. While we can split hairs about whether we already live in the matrix or how someone might experience the world using hallucinogenic substances, in general the affordances of this reality are already well-defined and the interaction strategies are well defined.
- Virtual and augmented reality are used to define regions in a spectrum of enhanced realities (carefully avoiding referring to them as points within this space since that term has already triggered connotations in one paper reviewer<sup>1</sup>). As discussed in chapter 1, these terms respectively offer much more than the typical use of stereotypical head-mounted display, and view through a mobile phone-camera experiences that are commonly associated with them.
- Enhanced reality is the term introduced in this book to encompass the various forms of technology<sup>2</sup> mediated experiences that integrate the various physical, virtual and abstract elements found in existing virtual and augmented reality environments while still allowing scope for further imaginative designs.

Similarly the design, development and deployment processes for enhanced reality applications are also presented separately from the technical details related to individual components. Agile approaches that are adapted to both design and development are presented in early chapters, while aspects of user experience evaluation and application publishing are discussed at the end of the book.

The core chapters then focus on individual software components that can be instantiated to achieve the de-

<sup>1</sup> An example of stress relief in a recipient of anonymous peer review.

<sup>2</sup> Noting the use of technology in the form of an applied science, rather than electronic hardware.

signed products and applications. A project focused approach has been adopted for these, for several reasons. Initial drafts of this material were starting to fall into the trap of many textbooks where vast amounts of material needed to be digested before getting to any exciting application areas. The range of concepts required for building all of these components is extensive and are best explored on demand when motivated by a particular goal or outcome. Rapid developments in enhanced reality technologies ensure the need to continually update knowledge; both the fundamental background understanding and the practical issues relating to implementing individual components.

These components fall into a range of categories including:

- Those supporting interaction and communication between multiple participants in shared realities. Interaction between multiple participants and system components is assumed to be a fundamentally desirable property of augmented reality applications, and mechanisms to achieve this are demonstrated through many of the example components.
- Those based on tracking visible features in the physical world. This can be used to anchor (register) virtual content to physical locations but equally well provides insight into the motion of the viewpoint relative to the surrounding environment.
- Those exploiting the freedom of movement achievable with mobile devices. This involves not only being able to track and coordinate participants over large areas but also how to scale the area of activity to regions not previously encountered.
- Those that sense and perceive the physical environment to enrich the user experience. This includes opportunities for interaction between participants, objects and the location itself based on how the actions of each can be interpreted and presented.

- Those that capture and present the physical environment so that it can be combined with a virtual overlay. The distinction between physical and virtual disappears when physical structures can be immediately digitized, and when there is little difference in the ways that the physical and virtual elements are perceived.
- Those that use alternative modalities (other than visual) to communicate aspects of the physical and virtual world and includes opportunities to enhance and extend the abilities of participants.
- Those that represent interfaces between new opportunities and existing platforms. New technologies are not adopted in isolation and enhanced reality components can be meaningful extensions to other widely used applications.

Components are gathered together in the second part of this book in the interests of separating concept and practice. Individual components, and any references to them, are identified with the symbol: .



# 1

## *Design for Enhanced Realities*

### *1.1 Project*

The initial project represents a common scenario for a designer of enhanced reality experiences. A potential client wants to commission you to build an enhanced reality experience for them. Your client may be new to this category of experience and may have recently been introduced to the concept by participating in one particularly engaging experience. They want you to build one just like it, only for their particular context. Alternatively your client may be widely familiar with the field, and have participated in the development of similar experiences previously.

In the first instance we want to support the client through a responsible design process that aims to identify their needs and goals, and advise them on the range of opportunities that exist within enhanced realities. For the latter case, we need to ensure that all the dimensions of an enhanced reality experience are discussed during the design phase and that any assumptions are questioned and verified before any development happens.

The question we need to answer is: How do we go about designing an enhanced reality experience that supports the needs of our client? A good answer should satisfy the following needs:

1. The design strategy should provide insight into the needs of the client. It is the designer's role to extract

clarity on the needs. The client is not likely to be in a position to clarify these in advance.

2. The design strategy should not constrain the solution based on the skills and facilities available to the development team. The client is likely to specify platform before function based on their previous experience with enhanced reality systems. A good design process should be able to focus on the function and apply the appropriate technology once this has been established.
3. The design strategy should be holistic. The strategy should consider all aspects of an enhanced reality experience. An enhanced reality experience design can often elegantly reuse one component of the experience to satisfy several goals at once. Designs where further features are added on halfway through the implementation process are rarely as efficient.

This chapter describes one such design strategy. Other valid strategies may exist and you are encouraged to continue to refine the strategy that you use based on your own experiences.

## *1.2 An Enhanced Reality Design Strategy*

Virtual reality refers to synthetic content that replaces sensory input originating from a physical reality. Augmented reality refers to situations where additional content is added to that present in the physical reality. Despite a common trend to regard virtual and augmented reality as representing opposing extremes in alternative realities, they are not actually mutually exclusive and can coexist in any given application. This book tends to use the terms loosely, to encourage innovation and flexibility in design and to avoid the typical dark patterns prevalent in the use of each term:

*Dark Pattern: Virtual Reality:* This pattern assumes that a virtual reality application requires the use of a head-mounted display (*hardware fallacy*) and that the aspects

of the physical world revealed during the experience represent a failure of the application (motion sickness, cable hazards, collisions with obstacles).

*Dark Pattern: Augmented Reality:* This pattern assumes that an augmented reality application involves showing some 3D object (*content fallacy*, *interaction fallacy*) when the camera of a smart phone (*hardware fallacy*) is facing some distinctive marker (*scope fallacy*) in the physical world.

These common failings result from selecting a particular implementation platform (hardware or software) first and fitting application requirements into the constraints associated with the chosen platform. Lack of imagination occurs due to over-rigorous adherence to the apparent distinction between sensory replacement versus augmentation. Opportunities for interaction are often constrained by perceived limits, and assumptions about affordances. Content tends to be primarily visual, explicit and concrete.

The dark patterns represent perceptions common to lay-people who could make use of these technologies and who may commission the design and development of such applications. The augmented reality application designer needs to be able to understand the true requirements of their clients, must be able to explore all the alternatives and should encourage their clients to effectively employ strategies that provide best value in achieving their goals.

The remainder of this chapter presents a design process representing one such approach for exploring the full potential of an alternate reality application design. Details of this approach are provided in McKenzie et al. [2018] and further works still in press. The various categories that distinguish different enhanced reality applications from one another have been identified. Each of the categories should be presented to the clients during an initial design workshop. The opportunities within each cate-

gory can be explored through discussion using examples from existing applications and contemporary research studies. The client and designers can then make decisions with respect to the nature of the intended experience with confidence that all opportunities have been considered. Innovation can be encouraged by exploring how alternative approaches would translate to the client's particular context.

The categories covered consist of:

*Nature of reality:* Are you building a virtual reality application, an augmented reality experience, some mixture of these, or something completely different?

*Location (setting):* Where does the experience take place? Is it at a particular location in the physical world, or could it take place anywhere in the physical world? How much of the setting is represented with synthetic content?

*Objects for interaction:* What are the elements of the environment that the participants can interact with? What form do these take? Interaction might involve moving physical objects around, or perhaps the dynamic elements of the setting are all passive and virtual.

*Interaction and feedback:* What form of feedback is provided, both from the participant to the application (such as actions and behaviour that is sensed), as well as stimuli provided by the application to the participants? Typically physical actions of the participants are provided to the application, which then responds with an information representation. Innovation with respect to interaction strategies might challenge this status quo.

*Concepts explored:* What is the purpose of the application? These are normally intended to communicate a particular message; for example an educational application might teach a particular topic. What concepts need to be presented? These can range from concrete spatial

relationships or involve more abstract ideas that can be challenging to represent.

*Participant engagement:* Interaction occurs not only between participant and application, but also between all the participants using the experience. How are these interactions mediated, particularly when each party may not be present in the same space or at the same time?

*Design of experience:* How do you engage the participants in the experience? Providing content is all well and good, but on its own is not sufficient incentive to select this experience over any other. Opportunities to motivate participation relate to what inspires participation such as achieving a challenge or being part of a developing story.

Discussing each of these points ensures that major facets of an enhanced reality experience are explicitly covered. Even for applications with more limited scope, it is better to cover the topic and agree to provide only a minimal solution in the design than to assume that all stakeholders implicitly understand this. Technologies can then be selected to best fit the desired functionality. The components in the later chapters of this book provide examples of such technologies.

This approach is not intended to be prescriptive. Terminology is deliberately described with scope for reinterpretation so that opportunities yet to be envisaged can be incorporated. Other application design and specification strategies could also be adopted. The goal with this chapter is to ensure a base-line design process that is better suited to enhanced reality applications than the standard requirements elicitation processes currently used by software engineers.

## 1.3 Enhanced Reality Design

### 1.3.1 Nature of reality

A good first question to ask the client is whether a virtual or augmented reality solution is necessary at all. It may be possible to achieve their goals while making use of approaches that remain fully embedded in physical reality. Synthetic content can then be introduced either through sensory substitution (virtual reality) or sensory enhancement (augmented reality). There may also be situations in which it is not necessary to manipulate the senses externally, but rather to address the perception through manipulation of imagination. This could be achieved by providing a written narrative from which participants generate their own interpretation of potentially abstract concepts.

An enhanced reality application has three components: the physical world in which the experience actually takes place, the computer and associated hardware capable of interacting with the participant, and the human participants each equipped with a mind capable of manipulating their perception. The actual experience can then be modified by processes produced by any or all of these. Actions in the physical world represent a form of physical mediation of the experience. Virtual elements are produced through computer mediation. Abstract elements are mediated through the human participant.

There are thus three dimensions to consider that relevant to the form of reality employed by any particular application:

*Physical:* This dimension defines a spectrum of experiences that ranges from purely physical realities at one extreme to those completely intangible realities with no physical embodiment.

*Virtual:* Applications can range from those providing synthetic computer mediated content as a virtual or augmented reality to the opposite extreme where the

content is perceived as real.

*Abstract:* Human mediated content includes purely abstract concepts that may not have an established physical form (feelings and emotions) and that are easily produced internally in responses to signs and signals provided. The opposite end of the spectrum requires minimal interpretation and is where concepts are particular concrete and have a well-defined representation established in a traditional reality.

These dimensions can be conveniently represented by three orthogonal axes in 3D, constrained to a unit cube as illustrated in Figure 1.3.1a. More conveniently, a planar slice across that cube provides a triangular section (shown in Figure 1.3.1a) that can be labelled with the extremes of each of the three dimensions, and redrawn in Figure 1.3.1b in a less cluttered form. Any point within the triangle represents a mixture of the three dimensions corresponding to the triangle vertices, with the contribution from each dimension corresponding to its proximity to the associated vertex. The labels on the midpoints of edges are chosen to not only be the converse of the opposing vertex, but to incorporate aspects of the other two adjacent vertices so that they make sense in this representation as well.

The remaining categories use a similar representation although the names of the dimensions may vary. The designer and client should agree on the classification (position within the triangle) corresponding to the proposed application. The center point represents an equal mixture of all dimensions and can be a good starting point. However where the client wants something specific then the design process should result in agreement about where this would fall within the triangle diagram.

Disagreement about the interpretation of the various dimensions is expected; the value of this design exercise is not that of perfectly and exactly classifying the context but rather forcing consideration of alternatives to achieve

the most desirable outcome. Agreeing to not incorporate a particular dimension is also acceptable provided this decision is explicitly and deliberately made. The role of the enhanced reality designer is to act as the domain expert by describing scenarios corresponding to nominated points in the space defined by the triangle diagram.

The following examples represent examples of particular choices regarding the nature of reality. Coordinates provided represent relative proximity to each of the three primary dimensions. Each is a valid design that is distinct according to how the application incorporates mediation from the physical environment, from virtual synthetic elements and through the cooperation of the human participants.

1. Exercise games are often strongly based in the physical world, where participants employ their bodies to run around the local area, or to undertake various forms of physical activity. The game 'Beat the Street' [Harris, 2018] monitors levels of walking, running or cycling by getting participants to register at RFID beacons along their path. The experience is almost completely mediated by the physical world. A minor abstract element is included to acknowledge the need for participants to be aware that tagging the RFID token at checkpoints contributes to the rewards that they receive for their activity. Coordinates: (P: 0.9, V: 0, A: 0.1)

Other experiences that see their reality as part of the physical world are the pervasive games that are played out in urban settings such as cities. Participants receive instructions on a mobile device but play out the game by visiting physical locations, finding physical objects or interacting directly with other players. They may document evidence of their progress through taking photographs of locations or scanning markers at the locations. There is still a significant level of technology mediation taking place that will be recognized under other categories but the nature of reality involved in this context is primarily physical. Coordinates: (P: 1, V:

o, A: o).

2. Live action virtual reality games play out in the physical world, but the players wear a head mounted display and almost completely perceive a virtual environment [Silva et al., 2016]. The virtual world is sensitive to what happens in the physical world; hazards in the physical world are re-interpreted as barriers presented in the virtual world. The nature of the reality perceived by the participant is thus largely computer generated, although there are other stimuli from the physical world that may received with other senses, resulting from the perception of physical movement while walking and sounds overhead. Coordinates: (P: 0.3, V: 0.7, A: o).

The game 'AR Worms' requires players wear a head-mounted display attached to a camera. A virtual landscape is then overlaid onto a physical tabletop and game play is confined to this region. The synthetic content dominates in this particular application with a minor contribution from physical reality in the form of interaction controls based on the player's physical position and direction of gaze. The application uses terrain and play elements that are real despite being fictional. Coordinates: (P: 0.1, V: 0.9, A: o).

3. An experience that uses a web site to develop environmental awareness [Lee et al., 2013] requires significant mental effort to develop a model of the reality involved. Significant parts of the experience require an imaginative construction of a mental model of the experience, anticipating the consequences of actions and presenting and sharing information over the social network included on the web site. As such, the nature of the reality presented by this experience is largely abstract although it is potentially as engaging as any of the other experiences described. Part of the experience does also include investigating environmental elements in the physical world, and interacting with media provided

on the site. Coordinates: (P: 0.1, V: 0.2, A: 0.7).

Imaginative elements are a key component of traditional entertainment experiences such as story telling and related performances. Games such as Tetris represent an abstract reality with particular rules and representations that allow manipulation of concepts embedded in a space that is not confined to physically realistic rules. Coordinates: (P: 0.1, V: 0.3, A: 0.6).

Further examples of applications representing different design choices regarding the nature of reality are identified in Figure 1.3.1, with references to the description of each design example listed in Table 1.1. Where the referenced source describes a range of different experiences, then the relevant properties of all of these are included in the ratings provided.

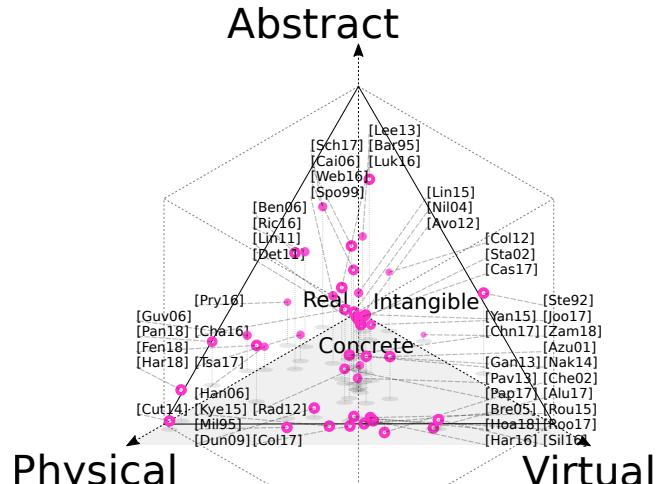
### 1.3.2 *Location (setting)*

The environment represented within the application is separated conceptually into the elements comprising the setting (fixed scenery, landscape, and buildings), and then also to the individual objects (see section 1.3.3) that the participants will interact with. The setting is typically handled differently to the objects with many conventional augmented reality applications being set in the physical world but provide interaction through additional virtual objects.

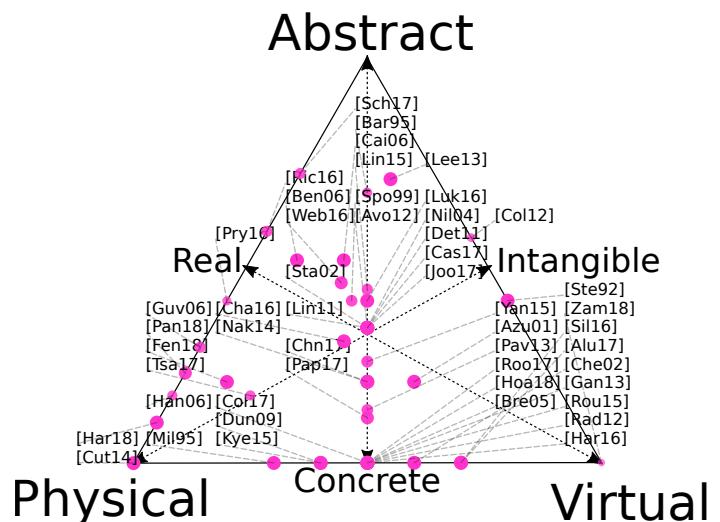
This section of the design process describes the dimensions related to the setting. The setting is typically static and does not move or change regularly in response to user interaction. Rather its value is in providing context for the application. It can also contribute by representing information, allowing participants to gain insight into various types of data through their exploration of the setting. Defining the setting involves making decisions about the way the participants perceive the location in which the experience occurs.

Describing the setting uses the physical-virtual-abstract

Figure 1.3.1: Nature of Reality is a design choice.



(a) Design choices presented using all three axes.



(b) Design choices presented relative to the three vertex points.

Table 1.1: Nature of reality design examples.

Key	Source	Abstract	Physical	Virtual
[Azu01]	Azuma et al. [2001]	0.2	0.3	0.5
[Ste92]	Steuer [1992]	0.4	0.0	0.6
[Bar95]	Barfield et al. [1995]	0.4	0.1	0.1
[Alu17]	Aluri [2017]	0.0	0.3	0.7
[Avou12]	Avouris and Yianoutsou [2012]	0.3	0.3	0.3
[Ben06]	Benford et al. [2006]	0.5	0.4	0.1
[Bre05]	Brederode et al. [2005]	0.0	0.5	0.5
[Cai06]	Caillois [2006]	0.5	0.3	0.2
[Chn17]	Ch'ng et al. [2017]	0.2	0.4	0.4
[Chat16]	Chatzidimitris et al. [2016]	0.3	0.4	0.3
[Cheo02]	Cheok et al. [2002]	0.0	0.4	0.6
[Col12]	Cole et al. [2012]	0.2	0.0	0.2
[Col17]	Collins et al. [2017]	0.0	0.5	0.5
[Cut14]	Cutter et al. [2014]	0.0	1.0	0.0
[Det11]	Deterding et al. [2011]	0.3	0.3	0.3
[Dun09]	Dunleavy et al. [2009]	0.0	0.6	0.4
[Fen18]	Fenu and Pittarello [2018]	0.2	0.7	0.1
[Gan13]	Ganapathy [2013]	0.0	0.2	0.2
[Har16]	Harley et al. [2016]	0.0	0.5	0.5
[Har18]	Harris [2018]	0.1	0.9	0.0
[Hoat18]	Hoang and Cox [2018]	0.0	0.5	0.5
[Joo07]	Joo-Nagata et al. [2007]	0.3	0.3	0.3
[Kye15]	Kysela and Storkova [2015]	0.0	0.6	0.4
[Lee13]	Lee et al. [2013]	0.7	0.1	0.2
[Lin15]	Lin and Chen [2015]	0.3	0.2	0.2
[Lin11]	Lindgren and Moshell [2011]	0.3	0.3	0.3
[Luk16]	Lukyanenko [2016]	0.4	0.3	0.3
[Mil95]	Milgram et al. [1995]	0.0	0.7	0.3
[Nak14]	Nakevska et al. [2014]	0.2	0.4	0.4
[Nil04]	Nilsen et al. [2004]	0.3	0.3	0.3
[Pan18]	Pang et al. [2018]	0.2	0.7	0.0
[Pap17]	Papathanasiou-Zuhrt et al. [2017]	0.1	0.4	0.4
[Pav13]	Pavlik and Bridges [2013]	0.1	0.3	0.3
[Pry16]	Pryss et al. [2016]	0.2	0.3	0.0
[Rad12]	Radu [2012]	0.0	0.3	0.3
[Ric16]	Richardson [2016]	0.4	0.3	0.0
[Roo17]	Roo and Hatchet [2017]	0.0	0.5	0.5
[Rou15]	Rouse et al. [2015]	0.0	0.5	0.5
[Sch17]	Schneider et al. [2017]	0.5	0.2	0.0
[Sil16]	Silva et al. [2016]	0.0	0.3	0.7
[Sta02]	Stapleton et al. [2002]	0.3	0.3	0.3
[Tsa17]	Tsai et al. [2017]	0.1	0.4	0.1
[Web16]	Weber [2016]	0.4	0.3	0.2
[Yan15]	Yan et al. [2015]	0.2	0.3	0.3
[Zam18]	Zamora-Musa et al. [2018]	0.0	0.0	0.3
[Han06]	Hansen [2006]	0.1	0.5	0.0
[Guvo06]	Guven [2006]	0.2	0.5	0.0
[Spo99]	Spoerer [1999]	0.3	0.2	0.2
[Cas17]	Castaneda et al. [2018]	0.3	0.3	0.3

dimensions introduced in section 1.3.1. Physical settings make use of the physical world to provide the background for the experience. This is commonly used in location based augmented reality applications, where participants either visit specific locations such as historical sites that are relevant to the application, or take advantage of the space around where the application is not dependent on any particular site. Virtual locations involve synthetic environments that may completely replace the physical world, as intended by typical immersive virtual reality applications which transport the participants to a completely new world. The degree of abstraction relates to the extent to which the setting would be achievable without a technology intervention. Representations of mathematical data sets, for example, would often be highly abstract involving mental interpretation while tours of solar systems and journeys at microscopic scale through the human body are often artistic reinterpretations of the actual setting, and would thus also involve a level of abstraction.

The following examples represent particular choices regarding the location (setting):

1. The game of AR Worms [Nilsen et al., 2004] is played on a computer generated 3D landscape that is projected into the region above a table set up in the room. A head mounted display is used to display the setting of the experience superimposed on a live camera feed. Despite the physical room being visible in the background, the game experience is confined to the virtual landscape and so the coordinates assigned are (P: o, V: 1, A: o). Other variations on table top games also mentioned [Nilsen et al., 2004] include table top war games using physical models of the setting which would have a significant physical component to the coordinates. Dungeons and Dragons are a comparable experience that may use only paper maps and role playing and so require mental effort in presenting a representation of the setting. This would correspond to an abstract location.

2. Tourist experiences enhance visit to physical locations by providing additional content such as direction signage or local information. Recreations of historical sites use the actual location [Papathanasiou-Zuhrt et al., 2017] and enhance this with media such as video clips or sound effects, and role-playing activities that help to re-imagine the physical location as it was in the past. Coordinates: (P: 0.3, V: 0.2, A: 0.5).
3. A choreography experience [Yan et al., 2015] allows dancers to monitor their own practice. The dancers wear a head mounted display that shows them a view of themselves from an external camera. The location in this case is presented through technology but is an unaltered representation of the physical setting. Coordinates: (P: 1, V: 0, A: 0).

Further examples of applications representing different design choices regarding the setting are identified in Figure 1.3.2, with references to the description of each design example listed in Table 1.2. Where the referenced source describes a range of different experiences, then the relevant properties of all of these are included in the ratings provided.

### 1.3.3 *Objects*

The term ‘objects’ as used in an enhanced reality application refers to the individual assets which populate the location and that are typically dynamic (not fixed in place). The focus in making design decisions around the use and representation of objects is how these support interaction in the application. Static fixed objects are regarded as part of the setting, discussed in section 1.3.2.

As with location (setting) in section 1.3.2, the objects can vary according to the physical, virtual and abstract dimensions. Physical objects exist in the physical world but can be tracked or sensed so that interaction with these objects has an effect on the application. Physical objects have the benefit of a tangible existence subject to all the

Figure 1.3.2: Location (Setting) is a design choice.

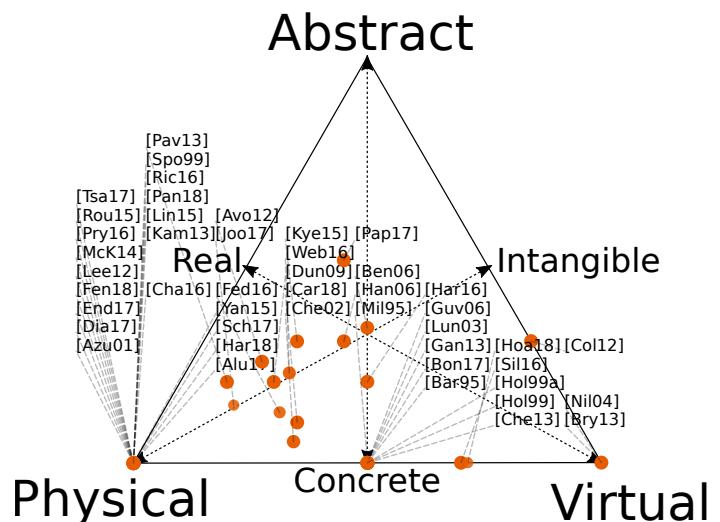
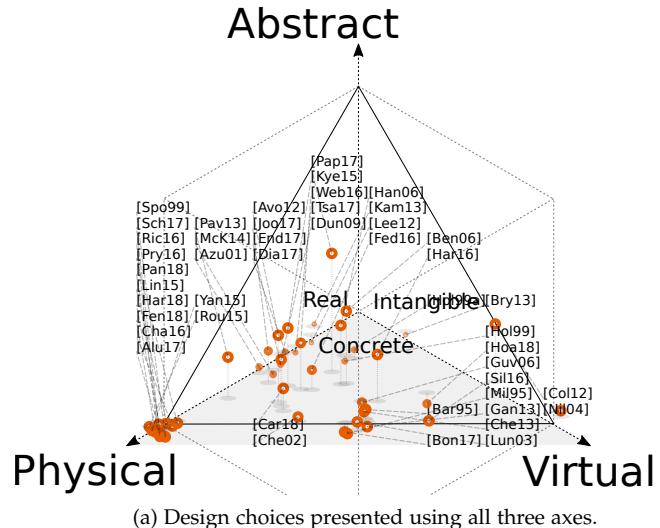


Table 1.2: Location (setting) design examples.

Key	Source	Abstract	Physical	Virtual
[Azu01]	Azuma et al. [2001]	0.0	0.5	0.0
[Bar95]	Barfield et al. [1995]	0.0	0.5	0.5
[Alu17]	Aluri [2017]	0.0	1.0	0.0
[Av012]	Avouris and Yiannoutsou [2012]	0.2	0.6	0.1
[Ben06]	Benford et al. [2006]	0.3	0.3	0.3
[Bon17]	Bonfert et al. [2017]	0.0	0.5	0.5
[Bry13]	Brynjolfsson et al. [2013]	0.0	0.0	0.2
[Car18]	Carlson et al. [2018]	0.1	0.6	0.3
[Chat16]	Chatzidimitris et al. [2016]	0.0	1.0	0.0
[Che13]	Chen et al. [2013]	0.0	0.5	0.5
[Cheo2]	Cheok et al. [2002]	0.1	0.6	0.3
[Col12]	Cole et al. [2012]	0.3	0.0	0.7
[Dia17]	Diaconu et al. [2018]	0.0	0.4	0.0
[Dun09]	Dunleavy et al. [2009]	0.2	0.6	0.2
[End17]	Endsley et al. [2017]	0.0	0.4	0.0
[Fed16]	Fedorov et al. [2016]	0.1	0.5	0.2
[Fen18]	Fenu and Pittarello [2018]	0.0	1.0	0.0
[Gan13]	Ganapathy [2013]	0.0	0.5	0.5
[Har16]	Harley et al. [2016]	0.2	0.4	0.4
[Har18]	Harris [2018]	0.0	1.0	0.0
[Hoa18]	Hoang and Cox [2018]	0.0	0.2	0.5
[Joo17]	Joo-Nagata et al. [2017]	0.1	0.5	0.1
[Kam13]	Kamarainen et al. [2013]	0.0	0.2	0.0
[Kye15]	Kysela and Storkova [2015]	0.3	0.5	0.2
[Lee12]	Lee [2012]	0.0	0.2	0.0
[Lin15]	Lin and Chen [2015]	0.0	1.0	0.0
[Lun03]	Lundgren and Bjork [2003]	0.0	0.5	0.5
[McK14]	McKenzie et al. [2014]	0.0	0.5	0.0
[Mil95]	Milgram et al. [1995]	0.0	0.5	0.5
[Nil04]	Nilsen et al. [2004]	0.0	0.0	1.0
[Pan18]	Pang et al. [2018]	0.0	1.0	0.0
[Pap17]	Papathanasiou-Zuhrt et al. [2017]	0.5	0.3	0.2
[Pav13]	Pavlik and Bridges [2013]	0.2	0.7	0.1
[Pry16]	Pryss et al. [2016]	0.0	1.0	0.0
[Ric16]	Richardson [2016]	0.0	1.0	0.0
[Rou15]	Rouse et al. [2015]	0.0	1.0	0.0
[Sch17]	Schneider et al. [2017]	0.0	1.0	0.0
[Sil16]	Silva et al. [2016]	0.0	0.3	0.7
[Tsa17]	Tsai et al. [2017]	0.0	0.3	0.0
[Web16]	Weber [2016]	0.2	0.5	0.2
[Yan15]	Yan et al. [2015]	0.0	1.0	0.0
[Han06]	Hansen [2006]	0.3	0.4	0.3
[Hol99]	Hollerer et al. [1999a]	0.0	0.2	0.2
[Guv06]	Guvenc [2006]	0.0	0.4	0.4
[Hol99a]	Hollerer et al. [1999b]	0.0	0.2	0.2
[Spo99]	Spoerer [1999]	0.0	1.0	0.0

(conventional) laws of physics which can be difficult to recreate accurately in a virtual world. Virtual objects are synthetic elements generated, manipulated and presented via a computer mediated overlay. Interaction with these will often need to be managed via some physical controller that translates a physical participant's actions into events that affect the virtual content. Abstract objects may lack an explicitly presentable form. They can often be represented indirectly, and inferred from values represented through a numeric readout, or through cues that enhance imagination, such as sound effects that might suggest the presence of a ghost. Objects, particularly those qualifying as abstract, will exhibit persistence and exist for extended periods throughout the application.

The following examples represent particular choices regarding objects:

1. Shift-Life uses physical objects such as watering cans [Ch'ng et al., 2017] as ways of controlling a virtual ecosystem simulation by pouring liquids which then affect pH and humidity. The object has both a tangible physical representation but also exists as the equivalent virtual object its interactive liquid pouring functionality is assumed to occur only in the virtual dimension. Both aspects are explicit and concrete. Coordinates: (P: 0.5, V: 0.5, A: 0).
2. An audio only version of Pacman has no visible content [Chatzidimitris et al., 2016]. Game objects such as ghosts and cookies are represented using sound cues whose production is mediated by the computer running the application, but whose interpretation still requires further processing by the participant. Some imagination is also required to fully perceive the experience. Coordinates: (P: 0, V: 0.7, A: 0.3). Game based activities work with abstract representations of objects that are never presented directly.
3. A location based game teaches language skills [Richardson, 2016] by completing tasks and solving puzzles

while exploring a city. The interactable elements are the media (videos, images) that appear when triggered by visible markers at key locations. These don't represent concrete objects themselves but need to be translated by the participants into solutions that lead to the next steps in the game. Coordinates: (P: 0, V: 0.2, A: 0.6).

Further abstraction can be achieved through even more indirect representations. Concepts such as points may be the only way to keep track of the virtual treasure found in a game where visiting key way points unlocks this form of reward [McKenzie et al., 2014].

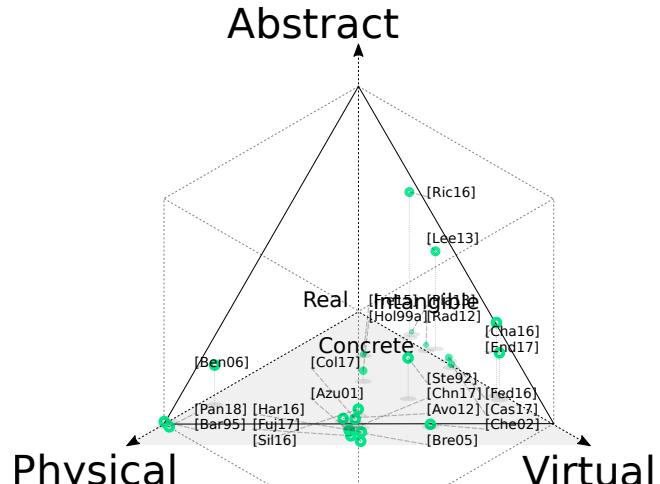
4. A museum tour that uses robots as tour guides [Pang et al., 2018] is providing information to the participants but the actual object that is interacted with is the robot itself. Coordinates: (P: 1, V: 0, A: 0).

Further examples of applications representing different design choices regarding interactive objects are identified in Figure 1.3.3, with references to the description of each design example listed in Table 1.3. Where the referenced source describes a range of different experiences, then the relevant properties of all of these are included in the ratings provided.

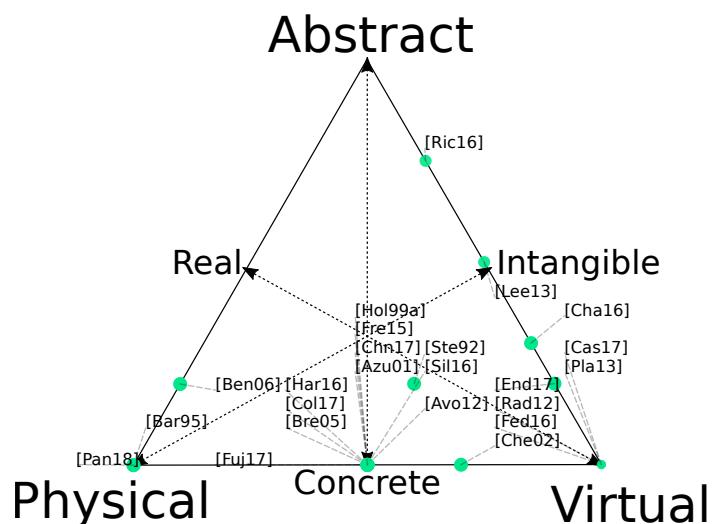
Table 1.3: Interaction with Objects design examples.

Key	Source	Abstract	Physical	Virtual
[Azu01]	Azuma et al. [2001]	0.0	0.5	0.5
[Ste92]	Steuer [1992]	0.2	0.3	0.5
[Bar95]	Barfield et al. [1995]	0.0	1.0	0.0
[Av012]	Avouris and Yianoutsou [2012]	0.0	0.5	0.5
[Ben06]	Benford et al. [2006]	0.2	0.8	0.0
[Bre05]	Brederode et al. [2005]	0.0	0.5	0.5
[Chn17]	Ch'ng et al. [2017]	0.0	0.5	0.5
[Cha16]	Chatzidimitris et al. [2016]	0.3	0.0	0.7
[Cheo02]	Cheok et al. [2002]	0.0	0.3	0.7
[Col17]	Collins et al. [2017]	0.0	0.5	0.5
[End17]	Endsley et al. [2017]	0.2	0.0	0.8
[Fed16]	Fedorov et al. [2016]	0.0	0.0	0.5
[Fre15]	Freschi et al. [2015]	0.0	0.2	0.2
[Fuj17]	Fujinawa et al. [2017]	0.0	0.5	0.5
[Har16]	Harley et al. [2016]	0.0	0.5	0.5
[Lee13]	Lee et al. [2013]	0.4	0.0	0.4
[Pan18]	Pang et al. [2018]	0.0	1.0	0.0
[Pla13]	Planinc et al. [2013]	0.0	0.0	0.2
[Rad12]	Radu [2012]	0.0	0.0	0.3
[Ric16]	Richardson [2016]	0.6	0.0	0.2
[Sil16]	Silva et al. [2016]	0.0	0.5	0.5
[Hol99a]	Hollerer et al. [1999b]	0.0	0.3	0.3
[Cas17]	Castaneda et al. [2018]	0.0	0.0	0.5

Figure 1.3.3: Interaction with Objects  
is a design choice.



(a) Design choices presented using all three axes.



(b) Design choices presented relative to the three vertex points.

### 1.3.4 *Interaction and Feedback*

For convenience in this section the term interaction is considered as the actions that user takes with respect to objects, location and other participants, with feedback being the resulting reaction of the application. This dimension thus encompasses both the nature of information provided by the participant to the application (interaction) and the information provided by the application to the participant (feedback). These are often handled independently with a typical virtual or augmented reality application interpreting physical actions of the participant, but providing the response in the form of a computer mediated image or audio response. Both directions can thus be considered independently under this category.

While interaction and feedback may take multiple forms with respect to mechanism and modality based on particular equipment technologies, the design stage should rather focus on the opportunities that are most appropriate to the application context. The physical-virtual-abstract triangle of dimensions can also be used as a tool to trigger ideas in this category.

Physical interaction and feedback occur when actions and responses are based on a physical mechanism. This would include physically performing the action of picking up an object, and receiving a physical response such as a tactile sensation as a result. Virtual interaction and feedback involves information transmission and display of response, such as manipulating an object just by pointing at it, and seeing a image representing that object. Abstract interaction and feedback may involve sensing mood, and responding by invoking an imaginative or emotional response. Some biofeedback systems may extend into the abstract dimension. You are welcome to debate whether the image on the screen of a mobile device derived directed from the camera is a form of physical or virtual feedback, as it has elements of both. You might be guided by how that image is intended for use in the application and assign it an appropriate position on the physical-

virtual-abstract spectrum based on this. The reasoning process used in making this decision is the value that this exercise has with respect to the design process.

The following examples represent particular choices regarding interaction and feedback:

1. An astronomy simulation casts the participants as an asteroid in the solar system [Lindgren and Moshell, 2011], translating their own movements into an element that affects the outcome of the dynamic interaction between planetary bodies. The interaction is regarded as reasonably limited because the motion only triggers the launch of an asteroid. Further motion is then determined by the planetary simulation. This is, however, appropriate to the content of the application where the goal is to explore the astronomical outcomes determined by these starting conditions. Coordinates: (P: 0.2, V: 0, A: 0).
2. Some enhanced reality experiences, such as the exergame ‘Beat the Street’ [Harris, 2018] requires players to physically tag their position as they visit different location. However the feedback provided is the relevant point here, as it is presented in the form of statistics provided through a web portal. This is considered to be feedback that is both virtual (collated and presented through software) but also with an abstract aspect in that the figures still need to be interpreted by the participant. Coordinates: (P: 0.1, V: 0.6, A: 0.3).  
Virtual feedback is often used in museum experiences where triggering an interaction results in a response that is presented virtually. This can either be a more concrete representation such as a virtual dinosaur that walks around the space or, more commonly, an information response such as a written or narrated response which would require mental processing. Coordinates: (P: 0, V: 0.6, A: 0.4).
3. While many augmented reality applications present explicit virtual views overlaying historical views on top of

contemporary structures, one approach [Harley et al., 2016] mentions the use of open-ended comparative questions where participants need to discover changes. This extra level of reasoning can be regarded as targeting understanding of abstract information included in the feedback. Coordinates: (P: 0, V: 0, A: 0.4)

Location based games allow players to track one another, even when others are only represented abstractly as dots on a map [Lundgren and Bjork, 2003]. Coordinates: (P: 0.1, V: 0.1, A: 0.8).

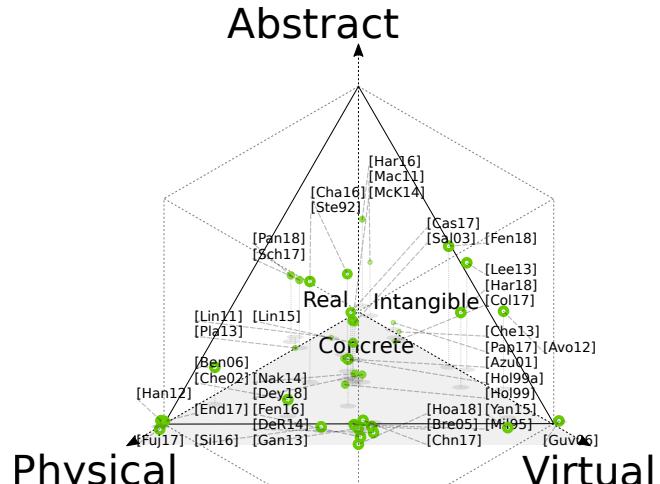
Table 1.4: Interaction and feedback design examples.

Key	Source	Abstract	Physical	Virtual
[Azu01]	Azuma et al. [2001]	0.2	0.4	0.4
[Ste92]	Steuer [1992]	0.4	0.4	0.2
[Avo12]	Avouris and Yiannoutsou [2012]	0.3	0.0	0.7
[Ben06]	Benford et al. [2006]	0.2	0.8	0.0
[Bre05]	Brederode et al. [2005]	0.0	0.5	0.5
[Chn17]	Ch'ng et al. [2017]	0.0	0.5	0.5
[Cha16]	Chatzidimitris et al. [2016]	0.4	0.3	0.2
[Che13]	Chen et al. [2013]	0.0	0.0	0.2
[Che02]	Cheok et al. [2002]	0.1	0.6	0.3
[Col17]	Collins et al. [2017]	0.0	0.0	0.2
[DeR14]	de Ribaupierre et al. [2014]	0.0	0.5	0.5
[End17]	Endsley et al. [2017]	0.0	1.0	0.0
[Fen18]	Fenu and Pittarello [2018]	0.5	0.0	0.5
[Fen16]	Fernandez-Cervantes et al. [2016]	0.0	0.5	0.5
[Fuj17]	Fujinawa et al. [2017]	0.0	1.0	0.0
[Gan13]	Ganapathy [2013]	0.0	0.5	0.5
[Han12]	Hansen [2012]	0.0	1.0	0.0
[Har16]	Harley et al. [2016]	0.4	0.0	0.0
[Har18]	Harris [2018]	0.3	0.1	0.6
[Hoa18]	Hoang and Cox [2018]	0.0	0.5	0.5
[Lee13]	Lee et al. [2013]	0.5	0.0	0.5
[Lin15]	Lin and Chen [2015]	0.2	0.3	0.3
[Lin11]	Lindgren and Moshell [2011]	0.0	0.2	0.0
[Mac11]	Macvean [2011]	0.2	0.0	0.0
[McK14]	McKenzie et al. [2014]	0.3	0.3	0.3
[Mil95]	Milgram et al. [1995]	0.0	0.5	0.5
[Nak14]	Nakevska et al. [2014]	0.0	0.3	0.3
[Pan18]	Pang et al. [2018]	0.3	0.3	0.0
[Pap17]	Papathanasiou-Zuhrt et al. [2017]	0.0	0.0	0.2
[Pla13]	Planinc et al. [2013]	0.0	0.3	0.0
[Sal03]	Salen and Zimmerman [2003]	0.2	0.2	0.2
[Sch17]	Schneider et al. [2017]	0.3	0.3	0.0
[Sil16]	Silva et al. [2016]	0.0	0.6	0.4
[Van15]	Yan et al. [2015]	0.0	0.1	0.9
[Hol99]	Hollerer et al. [1999a]	0.0	0.3	0.3
[Guvo06]	Guven [2006]	0.0	0.0	1.0
[Hol99a]	Hollerer et al. [1999b]	0.0	0.2	0.2
[Dey18]	Dey et al. [2018]	0.0	0.3	0.3
[Cas17]	Castaneda et al. [2018]	0.3	0.3	0.3

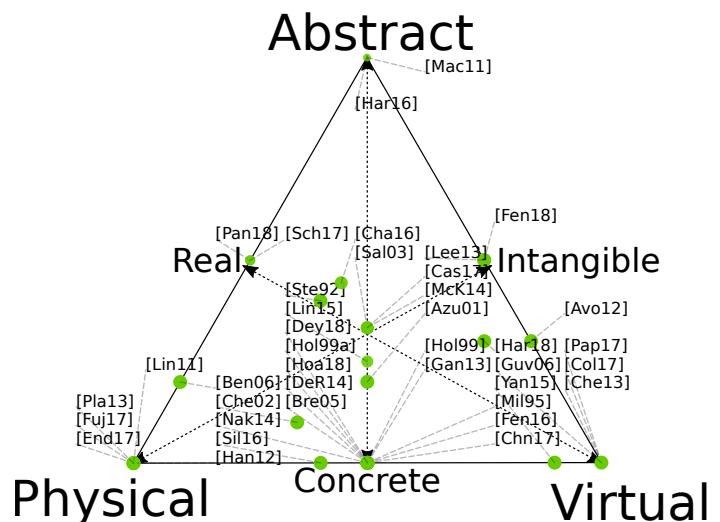
### 1.3.5 Concepts explored

Presenting the concepts associated with the application context is the category most likely to be of significant interest to the client, and open to opportunities for innova-

Figure 1.3.4: Interaction and feedback strategy is a design choice.



(a) Design choices presented using all three axes.



(b) Design choices presented relative to the three vertex points.

tion. Virtual and augmented reality applications are ideal for presenting spatial concepts such as size, shape and positioning of elements. This is applicable when learning to operate new pieces of equipment and developing the muscle memory related to where to find particular controls. It can be a great deal more challenging to present other forms of educational content such as the much reviled 'facts' or the more acceptable issues of abstract reasoning, critical thinking, problem solving or integrated skills such as communication, team-work or appreciation of diverse and creative insights.

Concepts are aligned variously to the physical, virtual and abstract dimensions used in other categories. Physically related concepts may involve working in a particular environment, such as a factory floor, and may be presented using an accurate simulation of the area and hazards involved. Virtual concepts involve spatial data that may not be readily available in the physical world but can be presented relative to the location. Wifi signal strength is a virtual concept that can be presented as a spatial overlay. Abstract concepts don't have a clear spatial component and include the idea of place attachment (learning to like a particular location) which would be a desirable outcome for a tourism application.

The following examples represent particular choices regarding concepts explored:

1. Many augmented reality applications use concrete concepts; mixtures of physical and virtual elements. One prediction for future shopping experiences combines a physical showroom environment with products presented as virtual models [Brynjolfsson et al., 2013]. This provides benefits of the context of the physical space with access to sales staff, while allowing the products to be experienced or dynamically customized without needing to maintain a large stock. Coordinates: (P: 0.5, V: 0.5, A: 0).

Physically oriented concepts include learning to operate or repair a piece of equipment [Radu, 2012] which

may range from a virtual reality activity learning operating procedures for an oil rig, or having an augmented guide to repairs cars and aircraft. Coordinates: (P: 0.5, V: 0.5, A: 0).

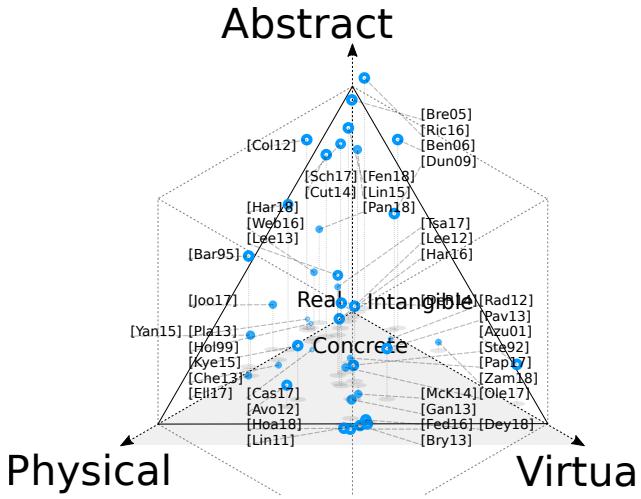
2. Virtual concepts are not easily created physically but have spatial representation. Examples include annotations added to television broadcasts of sporting events [Azuma et al., 2001]. Labels added to identify players, or markings on the field to indicate goal lines are spatial representations of information. Coordinates: (P: 0.0, V: 0.8, A: 0.2).
3. Abstract concepts require the most innovation to display. A literary museum could present physical and virtual representations of an author's writing but elects to rather communicate the ideas expressed by the author [Fenu and Pittarello, 2018]. The experience provides an opportunity to engage emotionally by presenting information about the context of the writing. Coordinates: (P: 0.05, V: 0.9, A: 0.05).

Other abstract concepts can range from triggering historical insights into an existing physical location (Coordinates: (P: 0.5, V: 0.1, A: 0.4)) to developing emotional ties with a location by enhancing brand awareness or place attachment [Xu et al., 2016, 2017] through activities that expose participants to associations of positive experiences and reward. Coordinates: (P: 0.1, V: 0.1, A: 0.8).

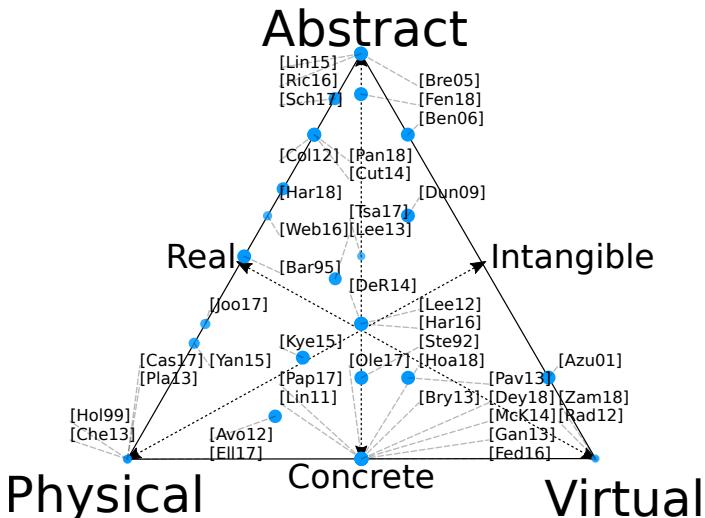
#### 1.3.6 *Participant engagement*

Participants in an alternative reality interact with location (setting), objects and with one another. The ways in which participants engage with one another, and with the experience, is another category to consider. This category uses a different set of dimensions to that presented in section 1.3.1:

*Copresence:* This dimension ranges from copresence where



(a) Design choices presented using all three axes.



(b) Design choices presented relative to the three vertex points.

Figure 1.3.5: Concepts communicated by the experience are a design choice.

Key	Source	Abstract	Physical	Virtual
[Azu01]	Azuma et al. [2001]	0.2	0.0	0.8
[Ste92]	Steuer [1992]	0.2	0.4	0.4
[Bar95]	Barfield et al. [1995]	0.5	0.5	0.0
[Avou12]	Avouris and Yiannoutsou [2012]	0.1	0.6	0.2
[Ben06]	Benford et al. [2006]	0.8	0.0	0.2
[Brec05]	Brederode et al. [2005]	1.0	0.0	0.0
[Bry13]	Brynjolfsson et al. [2013]	0.0	0.5	0.5
[Che13]	Chen et al. [2013]	0.0	0.4	0.0
[Col12]	Cole et al. [2012]	0.8	0.2	0.0
[Cut14]	Cutter et al. [2014]	0.8	0.2	0.0
[DeR14]	de Ribaupierre et al. [2014]	0.3	0.3	0.3
[Duno09]	Dunleavy et al. [2009]	0.6	0.1	0.3
[Ell17]	Ellmers et al. [2017]	0.0	0.5	0.0
[Fed16]	Fedorov et al. [2016]	0.0	0.5	0.5
[Fenu18]	Fenu and Pittarello [2018]	0.9	0.1	0.1
[Gan13]	Ganapathy [2013]	0.0	0.4	0.4
[Har16]	Harley et al. [2016]	0.3	0.3	0.3
[Har18]	Harris [2018]	0.6	0.3	0.0
[Hoa18]	Hoang and Cox [2018]	0.0	0.5	0.5
[Joo17]	Joo-Nagata et al. [2017]	0.2	0.4	0.0
[Kye15]	Kysela and Storkova [2015]	0.2	0.5	0.2
[Lee13]	Lee et al. [2013]	0.4	0.3	0.2
[Lee12]	Lee [2012]	0.3	0.3	0.3
[Lin15]	Lin and Chen [2015]	0.7	0.0	0.0
[Lin11]	Lindgren and Moshell [2011]	0.0	0.5	0.5
[McK14]	McKenzie et al. [2014]	0.0	0.3	0.3
[Ole17]	Oleksy and Wnuk [2017]	0.0	0.3	0.3
[Pan18]	Pang et al. [2018]	0.4	0.1	0.0
[Pap17]	Papathanasiou-Zuhrt et al. [2017]	0.0	0.2	0.2
[Pav13]	Pavlik and Bridges [2013]	0.2	0.3	0.5
[Pla13]	Planinc et al. [2013]	0.0	0.2	0.0
[Rad12]	Radu [2012]	0.0	0.0	0.2
[Ric16]	Richardson [2016]	1.0	0.0	0.0
[Sch17]	Schneider et al. [2017]	0.8	0.1	0.0
[Tsai17]	Tsai et al. [2017]	0.2	0.1	0.1
[Web16]	Weber [2016]	0.3	0.2	0.0
[Yan15]	Yan et al. [2015]	0.2	0.5	0.0
[Zam18]	Zamora-Musa et al. [2018]	0.0	0.0	0.4
[Hol99]	Hollerer et al. [1999a]	0.0	0.2	0.0
[Dey18]	Dey et al. [2018]	0.0	0.5	0.5
[Cas17]	Castaneda et al. [2018]	0.0	0.2	0.0

Table 1.5: Concepts communicated through design examples.

each participant's engagement is focused on, and mediated through, the experience itself to multi-participant where users engage with one another, for example by cooperating in order to achieve a given outcome.

*Physical Presence:* Communication with others can arise when all participants are in the same physical location and engagement primarily uses direct physical mechanisms. The converse of this is deliberate mediation through the experience in the virtual world to ensure participants are aware of the actions of others.

*Telepresence:* Virtual interaction ensures that communication between participants is provided, mediated and potentially modified by the application. The opposite end of the scale represents situated engagement: the case where participants can physically perceive the actions of others.

Popular location based games that involve visiting particular sites and collecting items are often classed as situated because players all visit the same physical location and can see others doing the same, but each player's progression in the game is unaffected by others. Telepresence focused engagement occurs in many online desktop games where players are geographically separated but communicate and work towards a common goal via the computer mediated communication over the network.

Social interaction is a significant mechanism for building a community around a particular activity or application. Social interactions also have the benefit of enriching the experience without necessarily requiring significant amounts of additional content to be developed.

The following examples represent particular choices regarding participant engagement:

1. An architectural walkthrough that involves a virtual building overlaid onto a physical mock-up of the house [Hoang and Cox, 2018] focuses on copresence. Multiple participants can move around the space at any

time but, other than seeing each other, do not interact in any way that affects the experience. Coordinates: (C: 0.9, P: 0.1, T: 0).

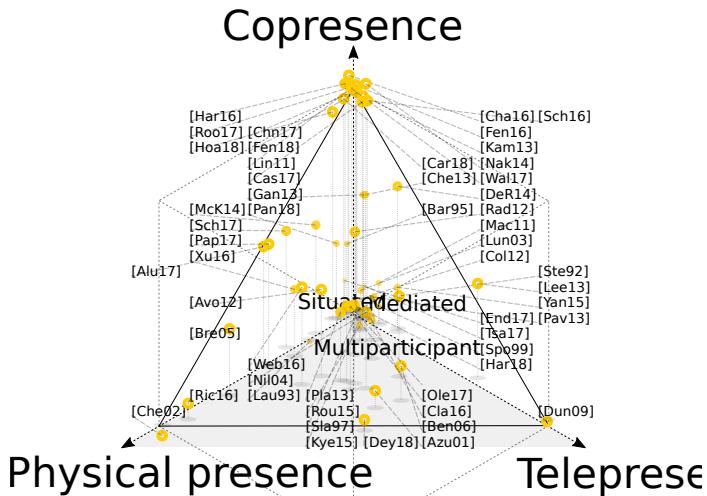
Such experiences have the participant responsible for their own outcomes, despite sharing a space where they can see others participating in the same experience. Location based games involving visiting a location and collecting the local speciality also fall into this category.

2. Some games limit the number of devices to one per team, forcing participants to interact physically to achieve the goal [Schneider et al., 2017]. In this case participants talk directly to one another while also engaging with the simulation provided by the application. The physical presence can be used to encourage collaborative behaviours. Coordinates: (C: 0.5, P: 0.3, T: 0).
3. An ‘Alien Contact’ experience is played by groups all inhabiting the same large physical space [Dunleavy et al., 2009]. However each team member has a different role and access to different information so all engagement is by telepresence mediated through the experience. Coordinates: (C: 0, P: 0, T: 1).

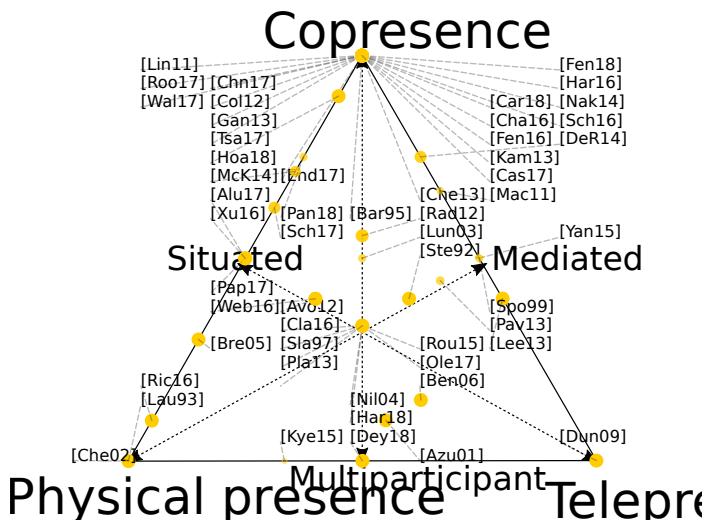
Mediated interaction can involve direct communication mediated through the application, or indirect awareness of others through shared information about how many points each participant has, or how much they are contributing towards a common goal [Planinc et al., 2013], even when they might be playing together in the same physical room. Coordinates: (C: 0.3, P: 0.3, T: 0.3).

### *1.3.7 Design of experience*

The nature of the experience in an alternative reality application should also be carefully considered with respect to the design. Applications will often support particular operations or mechanics but the aggregation of all of these



(a) Design choices presented using all three axes.



(b) Design choices presented relative to the three vertex points.

Figure 1.3.6: Participant engagement is a design choice.

Key	Source	Copresence	Physical presence	Telepresence
[Azu01]	Azuma et al. [2001]	0.1	0.4	0.5
[Ste92]	Steuer [1992]	0.4	0.2	0.4
[Bar95]	Barfield et al. [1995]	0.3	0.0	0.0
[Alu17]	Aluri [2017]	0.5	0.5	0.0
[Av012]	Avouris and Yiannoutsou [2012]	0.4	0.4	0.2
[Sla97]	Slater and Wilbur [1997]	0.1	0.1	0.1
[Ben06]	Benford et al. [2006]	0.1	0.3	0.6
[Bre05]	Brederode et al. [2005]	0.3	0.7	0.0
[Car18]	Carlson et al. [2018]	1.0	0.0	0.0
[Chn17]	Ch'ng et al. [2017]	1.0	0.0	0.0
[Cha16]	Chatzidimitris et al. [2016]	1.0	0.0	0.0
[Che13]	Chen et al. [2013]	0.5	0.0	0.0
[Che02]	Cheok et al. [2002]	0.0	1.0	0.0
[Cla16]	Clark and Clark [2016]	0.2	0.2	0.2
[Col12]	Cole et al. [2012]	0.1	0.0	0.0
[DeR14]	de Ribaupierre et al. [2014]	0.6	0.0	0.2
[Dun09]	Dunleavy et al. [2009]	0.0	0.0	1.0
[End17]	Endsley et al. [2017]	0.1	0.0	0.0
[Fen18]	Fenu and Pittarello [2018]	1.0	0.0	0.0
[Fem16]	Fernandez-Cervantes et al. [2016]	1.0	0.0	0.0
[Gan13]	Ganapathy [2013]	0.5	0.0	0.0
[Har16]	Harley et al. [2016]	1.0	0.0	0.0
[Har18]	Harris [2018]	0.3	0.3	0.3
[Hoa18]	Hoang and Cox [2018]	0.9	0.1	0.0
[Kam13]	Kamarainen et al. [2013]	1.0	0.0	0.0
[Kye15]	Kysela and Storkova [2015]	0.0	0.1	0.1
[Lau93]	Laurel [2013]	0.0	0.2	0.0
[Lee13]	Lee et al. [2013]	0.4	0.0	0.6
[Lin11]	Lindgren and Moshell [2011]	1.0	0.0	0.0
[Lun03]	Lundgren and Bjork [2003]	0.2	0.1	0.1
[Mac11]	Macvean [2011]	0.2	0.0	0.1
[McK14]	McKenzie et al. [2014]	0.5	0.2	0.0
[Nak14]	Nakevska et al. [2014]	1.0	0.0	0.0
[Nil04]	Nilsen et al. [2004]	0.3	0.3	0.3
[Ole17]	Oleksey and Wnuk [2017]	0.3	0.3	0.3
[Pan18]	Pang et al. [2018]	0.3	0.1	0.0
[Pap17]	Papathanasiou-Zuhrt et al. [2017]	0.5	0.5	0.0
[Pav13]	Pavlik and Bridges [2013]	0.2	0.1	0.2
[Pla13]	Planinc et al. [2013]	0.3	0.3	0.3
[Rad12]	Radu [2012]	0.5	0.2	0.2
[Ric16]	Richardson [2016]	0.1	0.9	0.0
[Roo17]	Roo and Hachet [2017]	1.0	0.0	0.0
[Rou15]	Rouse et al. [2015]	0.1	0.1	0.1
[Sch17]	Schneider et al. [2017]	0.5	0.3	0.0
[Sch16]	Schoneveld et al. [2016]	1.0	0.0	0.0
[Tsai17]	Tsai et al. [2017]	0.1	0.0	0.0
[Wal17]	Walk et al. [2017]	1.0	0.0	0.0
[Web16]	Weber [2016]	0.4	0.4	0.2
[Xu16]	Xu et al. [2016]	0.3	0.3	0.0
[Yan15]	Yan et al. [2015]	0.2	0.0	0.2
[Spo99]	Spohrer [1999]	0.1	0.0	0.1
[Dey18]	Dey et al. [2018]	0.0	0.5	0.5
[Cas17]	Castaneda et al. [2018]	1.0	0.0	0.0

Table 1.6: Participant engagement design examples.

is still insufficient to provide an experience to the users. Rather the elements need to be planned and integrated into a coherent user experience; one that would support the intended work flow. Exemplary applications tend to incorporate a mixture of three different dimensions:

*Narrative:* The narrative relates to the unfolding story and occurs where the design team have a predefined story, or set of story arcs, that guide the user through a series of stages. The opposite end of this spectrum is playful improvisation where activities are dynamic and only responsive to events that occur during use of the application.

*Performance:* Users of the application can be encouraged to role play, or otherwise introduce an imaginative overlay supported by the application. Performance incorporates users and bystanders as part of the experience reducing the need to manage all aspects purely through the developed content. The contrast to this is where participants just get to *experience* the design, and where there are strict rules in how they may contribute to the experience.

*Ludology:* Some applications are explicitly defined as games with associated rules and goals that govern player behaviours. Other applications gamify by incorporate aspects of game structures in non-game related contexts in order to employ various forms of motivation and behaviour shaping. Participants can potentially exploit the rules to produce novel emergent outcomes. Ludic play constrains the actions of participants as opposed to a *scripted* approach where participants can only choose from outcomes that are explicitly mapped out.

Incorporating all of these dimensions in a single application appears to require significant complexity. However aspects of each can be used, often with minimal effort, as in many cases the users are being guided in how to effectively contribute to the experience. Harmonious blending

is desirable so that it doesn't look like the application was assembled from a set of unrelated building blocks.

The following examples represent particular choices regarding design of experience:

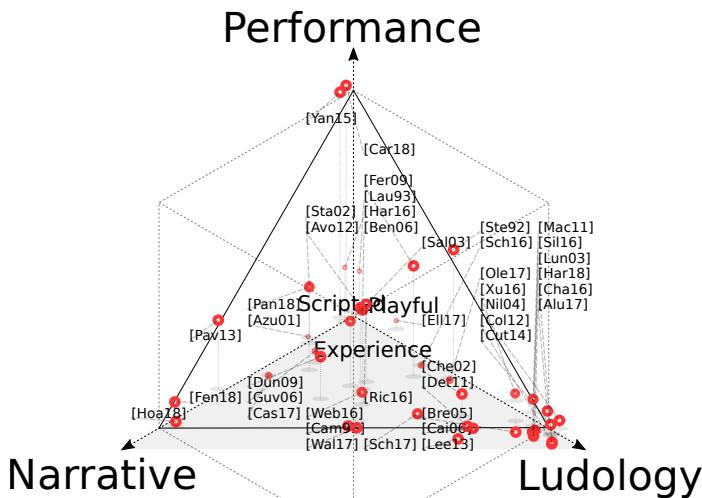
1. Augmented reality experiences designed for journalism concentrate on the unfolding story [Pavlik and Bridges, 2013]. An element of performance can be included as the reader becomes part of the narrative. Coordinates: (N: 0.7, P: 0.3, L: 0).

A story enfolds as tourists visit various sites in the medieval city of Rhodes encouraging them to continue their visit to unravel all the threads [Papathanasiou-Zuhrt et al., 2017]. Coordinates: (N: 0.8, P: 0.1, L: 0.1).

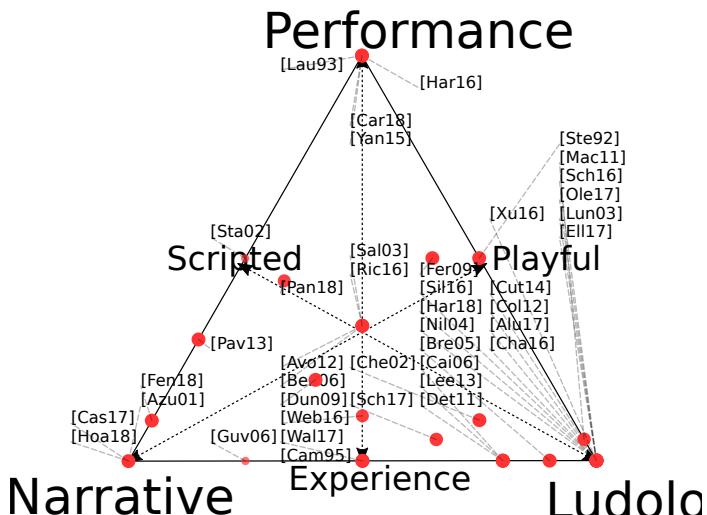
2. Game elements are frequently used as a form of gamification through leaderboards and financial rewards. More subtle behaviour shaping incorporates other game elements such the fun and challenge of solving a puzzle [Xu et al., 2016]. Game elements provide activities to market a location before arrival, but also can provide visitors with structured activities once they do arrive. Coordinates: (N: 0, P: 0, L: 0.8).
3. A street performance organized as a mixture of puzzle game and street theatre has participants exploring the city streets, looking for clues, and interacting with strangers (who may or may not be part of the game) [Benford et al., 2006]. The organizers must carefully manage the process to ensure the story unfolds while dealing with the unscripted and potentially chaotic elements. This experience, while challenging, represents a blend of the different dimensions. Coordinates: (N: 0.33, P: 0.33, L: 0.33).

### *1.3.8 Equipment*

The hardware required to implement an alternative reality application is deliberately left to last. While some design decisions may need to be refined in the light of currently



(a) Design choices presented using all three axes.



(b) Design choices presented relative to the three vertex points.

Figure 1.3.7: The style of the experience is a design choice.

Key	Source	Performance	Narrative	Ludology
[Azuo01]	Azuma et al. [2001]	0.0	0.2	0.0
[Ste92]	Steuer [1992]	0.5	0.0	0.5
[Alu17]	Aluri [2017]	0.0	0.0	1.0
[Av012]	Avouris and Yiannoutsou [2012]	0.3	0.3	0.3
[Ben06]	Benford et al. [2006]	0.3	0.3	0.3
[Bred05]	Bredereode et al. [2005]	0.0	0.1	0.9
[Cai06]	Caillois [2006]	0.0	0.2	0.8
[Cam95]	Cameron [1995]	0.0	0.5	0.5
[Car18]	Carlson et al. [2018]	1.0	0.0	0.0
[Chat16]	Chatzidimitris et al. [2016]	0.0	0.0	1.0
[Che02]	Cheok et al. [2002]	0.1	0.2	0.7
[Col12]	Cole et al. [2012]	0.0	0.0	1.0
[Cut14]	Cutter et al. [2014]	0.0	0.0	1.0
[Det11]	Deterding et al. [2011]	0.0	0.2	0.8
[Dun09]	Dunleavy et al. [2009]	0.2	0.5	0.3
[Ell17]	Ellmers et al. [2017]	0.0	0.0	0.2
[Fen18]	Fenu and Pittarello [2018]	0.1	0.9	0.0
[Fer09]	Fernandez-Vara [2009]	0.5	0.1	0.4
[Har16]	Harley et al. [2016]	0.2	0.0	0.0
[Har18]	Harris [2018]	0.0	0.0	1.0
[Hoat18]	Hoang and Cox [2018]	0.0	1.0	0.0
[Lau03]	Laurel [2013]	0.2	0.0	0.0
[Lee13]	Lee et al. [2013]	0.0	0.2	0.8
[Lun03]	Lundgren and Bjork [2003]	0.0	0.0	1.0
[Mac11]	Macewan [2011]	0.1	0.0	0.9
[Nil04]	Nilsen et al. [2004]	0.0	0.0	1.0
[Ole17]	Oleksy and Wnuk [2017]	0.0	0.0	0.5
[Pan18]	Pang et al. [2018]	0.4	0.4	0.1
[Pav13]	Pavlik and Bridges [2013]	0.3	0.7	0.0
[Ric16]	Richardson [2016]	0.3	0.3	0.3
[Sal03]	Salen and Zimmerman [2003]	0.3	0.3	0.3
[Sch17]	Schneider et al. [2017]	0.1	0.3	0.6
[Sch16]	Schoneveld et al. [2016]	0.0	0.0	0.4
[Sil16]	Silva et al. [2016]	0.0	0.0	1.0
[Sta02]	Stapleton et al. [2002]	0.2	0.2	0.0
[Wal17]	Walk et al. [2017]	0.0	0.5	0.5
[Web16]	Weber [2016]	0.1	0.4	0.4
[Xui16]	Xu et al. [2016]	0.0	0.0	0.8
[Yan15]	Yan et al. [2015]	1.0	0.0	0.0
[Guv06]	Guven [2006]	0.0	0.3	0.1
[Cas17]	Castaneda et al. [2018]	0.0	0.5	0.0

Table 1.7: Style of the experience illustrated through design examples.

feasible technologies there are a wide range of facilities that support virtual and augmented reality applications. Many of these are described as the technical components covered in later chapters. Creative adaptation and application of these technologies can be applied to achieve the design scenarios based on the previous categories. As is also discussed, participants are also able to contribute to the experience by invoking imagination and fantasy, if suitably motivated.

An exhaustive description of all augmented reality technologies is unlikely to be achievable given the rapid rate of innovation in this area. From a holistic view, however, the technologies available can be roughly categorized into four groupings:

*End-user:* These are technologies that are accessible to all by being incorporated into everyday products, including smart phones or web browsers. It also includes template applications, that allow non-specialists to quickly develop particular types of augmented reality application. In these cases development involves providing context specific content and the remainder of the application is automatically generated according to a previously created design. The benefits of working at this layer are that it is reasonable to assume that the target audience already has all the equipment provided, and that access to hardware is not a restriction in widely deploying the application.

*Expert developer:* Some technologies are available only through specialist development environments, or require significant and non-trivial content development. This may include having to map a particular environment in a specific way, or manage networking libraries and infrastructure to support multi-participant interaction.

*Augmented user:* The user needs specialist equipment which is not generally available. This includes custom wearable devices that may have been designed

specifically for the application, such as a form of haptic response vest.

*Expert operation:* There are scenarios where the application must be operated by trained staff. This may occur when the equipment used is complicated, bulky or hazardous, or when it may need to be calibrated or fitted to each individual user.

#### 1.4 The Component for Enhanced Reality Design

The previous section describes the areas to be considered while designing an enhanced reality application, and links the range of choices to examples of existing innovative designs. This information is particularly relevant to role of the enhanced reality expert within any design team. In a practical design session, there will be a range of stakeholders each with their own specific expertise who need to contribute to the design of the proposed experience.

 The component for enhanced reality experience design is provided in Component 15.1.1 on page 169.

This section describes how the design meeting could run, as a condensed overview for the remaining participants. The enhanced reality expert would still require all the background presented previously to guide the discussion and suggest opportunities as each category is covered.

This process can be used as the ideation stage of the design sprint, which is introduced in chapter 2.



## 2

# *Project Management for Creative Technology*

### *2.1 Project*

The scenario is as follows: you have completed your design after having completely engaged with the client, understood their needs, discussed how the dimensions of an enhanced reality experience apply in their context, and presented a description of the components of the experience that can be created and that clearly shows how the participants in the experience will have an effective and satisfying experience. The development team takes over. In this setting you need to take on the role of the project manager. This involves coordinating your team of developers, and defining a working process that will transform the design into the software and hardware components of an actual experience.

The question to be answered: What working practices will you agree on with your team?

### *2.2 Agile approaches*

Software development can be an isolating experience where specifications are provided to a coding team which are then expected to spin this into a functional application regardless of flaws in the documentation and changing

desires of stakeholders. Fortunately for virtual and augmented reality application development better work flows using agile strategies have been devised that both take into account the realities of the process, but also offer opportunities for the diverse roles and abilities of all team members to still contribute throughout the process. These roles include:

- Clients, users, and other stakeholders who stand to benefit from the completed application, and who are external to the development process but benefit the project by providing regular feedback on prototypes developed along the way.
- Project managers who ensure that interactions between various team members are facilitated, and that resources and facilities are available as required.
- Designers who consider the application requirements and devise creative and elegant combinations of functional components that meet these requirements. Designers must ensure that the design is feasible. User experience design ensures that the design, and the way it is expressed through the user interface, meets the needs of those who are going to be using the application. A good user interface guides the users efficiently through the common tasks that they need to complete. A bad user interface just exposes the underlying software components without considering how the application will be used.
- Developers are responsible for building the various components of the application and creatively solving the problems involved in adapting existing hardware, software elements and algorithmic processes to ensure that the component functions as required. They will often also be responsible for identifying the cause of any issues responsible for causing part of the application to misbehave, and may need to work with the design team to revise specifications where these are inappropriate.

- Regular reviews of the developing application involve testing of application prototypes by clients and potential users, with the intent of refining the set of features provided, improving the user experience, and ensuring that the application works in the context that it is designed for. Members of the development team are also responsible for testing smaller components of the application to ensure they behave as expected, and that issues do not arise when individually developed components need to interact with one another. Formal testing of the user experience with subjects with no affiliation to the project is also essential when creating extended reality applications.
- The completed application requires the skills of still further roles. The marketing team may start their work early in the application development life cycle, generating interest and then focusing on ensuring uptake once the application is published. Members of the development team are also involved in packaging the application for distribution, responding to error reports once the application is in the hands of its target audience, and monitoring the data analytics generated by the application to identify any adjustments that may be required.

The value in the agile approach is that it promotes working habits that adapt to each individual team, rather than mandating a one-size-fits-all approach.<sup>1</sup> Certain elements of team work are essential but the agile philosophy can be achieved by selecting from a number of strategies that incorporate these elements. The typical processes involve whole-of-team meetings to ensure everyone is aware of what all other members of the team are doing. For routine matters, these meetings are kept as short as possible to avoid wasting time. Documentation tends to be captured informally, on notes that can be re-prioritized, moved, altered, destroyed or created as required. These notes can be associated with individual team members to help track

<sup>1</sup> A through review of agile approaches can be found at: <https://www.atlassian.com/agile>

where effort is being applied. The shared note board then also provides feedback regarding the progress with the project to help manage deadlines.

This process is based on the steps of the design sprint as presented by Google Ventures.<sup>2</sup>

<sup>2</sup> <http://www.gv.com/sprint/>

*Monday:* The goal needs to be defined. This involves negotiation amongst the team of stakeholders but also informed decisions by undertaking research and consulting experts in the problem area. The goal should define the target to be reached at the end of the sprint.

*Tuesday:* Ideas are generated using whichever creative ideation strategy is agreed upon. These include approaches such as standard brainstorming, or attempting to remix old ideas. During the afternoon individuals will collate ideas towards achieving the goal and sketch out several potential solutions.

*Wednesday:* The various solutions are discussed and remixed. Voting strategies are used to achieve consensus on a preferred solution. Limitations are placed on new ideas to ensure that the process moves forward.

*Thursday:* A prototype is constructed. This will be limited in functionality and potentially mostly a mock up of the interface and experience. The team then does (internally) some test runs with the prototype.

*Friday:* The prototype is tested on potential customers, to assess their response to the concept. The outcomes are collated and interpreted to decide on the outcome of the sprint. This may be: complete failure (start sprint afresh next week), partial success (refine goal and focus on improvement in further sprints), or success (idea is promising and worth building and evaluating more detailed prototypes).

### 2.2.1 Documentation

The most useful documentation is that which actually gets read. Ideally all documents used are consistent, updated when decisions made, and are accessible to those that need them. This suggests that large bound paper tomes are significantly less valuable than online materials which are linked to the working processes which develop and use them.<sup>3</sup>

The information that is most useful during an agile design and development process includes:

1. Details of what the product has to be able to accomplish. Negotiation with clients and stakeholders can generate significant amounts of notes. These need to be summarized in a way that is accessible to the team. One strategy is to present a hierarchy of the goals that need to be accomplished. Higher levels of these goals represent the overall purpose of the project, which are then broken down into the smaller outcomes that are needed to accomplish these primary goals. At the finest level, each of the micro-goals is an outcome for a task.

Other ways of communicating these concepts includes providing a list of requirements (a different view of the goals), or presenting a sequence of user stories. Each story explains operations (or sequences of operations) that need to be achieved with the final product. Completing a story sequence is a convenient way of dividing up the development work into individual sprints. The prototype at the end of each sprint will be capable of supporting at least one extra story.

User stories are rapidly developing into a religion of their own. If you are that way inclined, then feel free to adopt the associated rules, regulations, acceptable phrasing and role restrictions. As with goals and requirements, the purpose of stories is ultimately to identify the sequence of tasks required to build the product.

As an alternative, consider a user story in the form of a

<sup>3</sup> What are you using to read this document?

narrative. Write one or two paragraphs telling the story of ‘a day in the life’ of one of the stakeholders engaging with the experience. The story should describe the steps that the user follows while using the product, and explain clearly how they achieve their goals in the process. These goals should correspond to the goals identified for the product. This form of user story can now serve a numbers of purposes:

- It is a form of design documentation. Your client can read the description and appreciate what facilities the experience provides and how these match with the outcomes they require.
- The steps of the description also specify the components to be created, and how these will interact. This is information needed by your development team.
- Stories can be prioritized. This provides insight into the way the project needs to be managed, and the order in which components need to be built and integrated.
- The other stakeholders that might be participants in the experience can review what they need to do, and what value the product provides for them. If their experience is particularly onerous then this suggest improvements to be made to the design of the user experience. This is also an opportunity to streamline the stories.
- Significantly, the stories may reveal opportunities to refine the experience. Many digital products are recreations of physical processes that also inherit inefficiencies and redundant processes. An enhanced reality experience that replaces an existing practice should not just be a mirror of its physical elements, but rather an opportunity to review the reasons for any existing process (can a different abstraction be applied) or to streamline the process by effectively using the capabilities of the virtual elements.

It does also help to be clear on what will **not** be created. Aspects that are out of the scope of the project should be explicitly listed, particularly if they have come up in conversation with the suggestion that they would be nice to have.

2. Architectural design documentation represents the software engineering elements of the project. This describes how the elements of the project are decomposed into logical units with particular interactions. Some software development projects treat this as a stage in the development pipeline and replace the goals with the structural documentation before proceeding on to further development. This is an effective way of ensuring that the usage experience for the software reflects the information transfer within the computing elements and the worst case user experience for all human stakeholders. The goals should drive the development process and the architecture (software components and their interactions) needs to support this. The architecture documentation is still necessary for programming tasks in order to produce well structured and robust modules. This should include a diagram of the components, showing their interactions, at the very minimum.
3. User interface mockups. This is an effective way of communicating with non-technical team members. In most cases people are concerned about their experience with the product as evidenced by walking through ways to achieve stories using the interface. Enhanced reality applications may also have novel forms of user interaction and mockups of the interface and devices involved can help communicate how these work. Making the technology elements achieve this interface then becomes the challenge for the design and development team.

Enhanced reality applications offer an opportunity relative to traditional digital interfaces. Interfaces need no longer be constrained to be collections of buttons and

menus, but have the opportunity to sense and interact in physical, virtual and abstract spaces. Challenge your design team to innovate to achieve minimal participant effort in the most common user stories.

4. Operational details that track tasks underway and help coordinate roles that interact. This is achieved through some form of dynamic document, such as a white-board with sticky notes (representing tasks) that can be moved around the board. Different regions of the board correspond to different stages of the task: unassigned, someone working on it, completed, or validated. Electronic versions of the concept supported distributed teams, and allow collection of statistics to excite middle management and terrorize team members.
5. The assets produced (software elements as well as other content) need to be tracked and managed. Version control systems ensure everything is in one common repository. Databases or spreadsheets help track and manage large collections of media content. A well organized directory structure ensures that individual assets are easy find, particularly if the entire team agrees to use a common convention. Further documentation, particularly regarding the emerging software architecture and any refinements, can be generated from comments embedded in program files. While there is some debate around this practice (try any search engine on the pros and cons of self documenting code and commenting practices), tools and associated conventions for writing comments are relatively common (<http://www.doxygen.nl/>). Such generated documentation is not intended for end-users but rather for use by other members of the development team, and to support later sprints and software maintenance.
6. Instruction manuals used to be a common way to communicate details of how to operate a particular product. Previously lengthy paper tomes shipped with a product, these increasingly transformed into electronic

documents shipped on disk and then to just a link to download. While the act of holding a manual may be reassuring to many older users, the practice of user documentation has moved on and current practice employs online forums to discuss and resolve problems. Search technology means that product users are more likely to expect an answer to their specific question than accept that they must read through the whole manual to perhaps glean some insight. This would suggest that a list of frequently answered questions, and a search engine indexed question and answer site is the better way of supporting users of your product.

## *2.3 Building Enhanced Reality Applications*

The design of an enhanced reality experience is followed by its implementation. This entails the use of a suitable hardware and software environment to actually develop the components of the application. Typically choice of the development environment would be dictated by the technologies chosen. In many cases a virtual and augmented reality engine provides a sufficiently generic platform to support a wide range of application designs.

### *2.3.1 Building the foundation*

The components in this book are demonstrated using Unity software. There is a certain amount of standard ‘housekeeping’ required for any new project. The components discussed in this section describe these steps, which are likely to form the initial stages of any project. The examples presented cover setting up a project to deploy to a mobile platform such as a smart phone used for a touch-screen based augmented reality context, or for a mobile virtual reality head mounted display.

A more comprehensive discussion of project management and development practices follows.

Two components are provided to demonstrate the pro-

cess of setting up an initial project which can be deployed and tested.  Component 16.1.1 on page 179 provides a component which can serve as the basis for a mobile augmented reality experience that would run on a smart phone.  Component 16.2.1 on page 192 is intended for virtual reality experiences that make use of a head mounted display and controllers.

Key points covered in these examples are:

- Configuring your engine to deploy to your target hardware. Particularly when this is a separate mobile device, the development process needs to ensure that the developer can rapidly develop and test on a desktop platform but also deploy on the mobile device to confirm that features specific to that environment are working as expected.
- The devices used for augmented and virtual reality experiences typically require third party components to allow the application to manage interaction and rendering as required by that platform. It is helpful to integrate these components into the development environment as soon as possible as they often impose constraints on how the development environment can be configured. Such constraints are often non-negotiable so it is worth being aware of them before you invest too much effort in the rest of the project.

Having a working prototype, regardless of how rudimentary it may start off as, is a valuable property to maintain as the project develops. Each developer can be productive without being blocked by having to wait for a separate component to be developed. Your client and other stakeholders (including the quality assurance team) can test the solution at any point, and identify issues that may arise while there is still time left to respond to them. This allows the development process to be agile and to prioritize producing a useful product over mechanically following instructions.

### 2.3.2 Agile working practices

Once the task level goals are identified, each task can be assigned to an individual representing the appropriate role to undertake that task. If the tasks are broken down properly then each task takes a small amount of time (a few hours to a few days) and has a clearly defined outcome so that there is no disagreement about whether the task is complete.

The agile development process breaks up the complete development process into a sequence of sprints. Each sprint results in a working prototype capable of achieving some (more) of the goals for the product. Tasks relevant to achieving these goals are the ones that are allocated during the sprint.

The sprint process starts with identifying which goals and tasks are appropriate for the sprint (the backlog). Each team member then tries to complete the tasks required during the sprint. At the end, there is a period to catch your breath, demonstrate and review the prototype with all stakeholders, before starting again. The existence of a working prototype after each sprint allows testing, and opportunities to identify potential issues and work to resolve these before too much effort has been invested.

Various forms of sprint exist, such as the scrum process which tries to avoid explicit management and encourages self-organization within the team. A scrum manager takes responsibility for any coordination required but otherwise focuses on ensuring all team members have what they need and fends off outside distractions. The entire team meets once a day to catch up on issues relevant to the team. Emphasis is placed on getting this over with as fast as possible (15 minutes total, held as a stand-up meeting to prevent anyone getting comfortable). Other approaches, such as kanban, involve working continuously on the tasks but having an integrated and working prototype at any time. In theory the product is potentially never finished but is always continually getting better.

### 2.3.3 Team communication

The different roles with the team need to ensure they are aware of what others are doing during the design and development of the product. Several communication scenarios are common:

*Project management and design roles:* During early stages of the project members of the design team must work with each other and other stakeholders to clearly identify the goals and requirements, to creatively design structures and architectures to achieve these, and communicate these to other team members. This utilizes design strategies such as those outlined in Chapter 1, and using design sprints (<http://www.gv.com/sprint/>). Typically these interactions involve communicating with others who have backgrounds and priorities that differ from your own. Concepts and jargon are often specific to different disciplines, and someone who appears to be using the right words may be saying something very different. There are communication strategies that can help with addressing these issues:

- Ask why. Do so repeatedly. Your partners may assume that your role is create what they ask for. In reality your job is to find out what they want, and to use your enhanced reality design and development expertise to help them consider the options that are available to them. Find out what their needs are (the goals for the project) rather than a list of features that must be provided. In many cases you may be asked to replicate some existing process that has always been used. Find out what the purpose of that process originally was - and then see if it can be achieved more efficiently and with greater satisfaction in other ways.
- Get the client to tell the story of how they intend to make use of the proposed product. This not only maps nicely to producing user stories, but again offers opportunities to suggest ways in which steps

could be simplified or streamlined. The user experience is optimized if you can reduce the number of steps required to achieve the same outcome (unless, for some reason, you're being paid by the number of features you add to the product). The ideal product is one that doesn't require any effort to use it at all. One strategy that does produce surprising results is to watch the client using their current solution. As an outsider with no involvement in the process it is easy to ask why, particularly when it comes to steps whose original purpose may have been lost, and which are now enacted solely by tradition. There is also an opportunity to spot ways in which enhanced reality concepts could be applied to enhance existing steps, or offer opportunities that were not feasible in the past.

Data can be collected by observing users of alternative solutions, or by adding a monitoring and analytics tool where appropriate. This is also helpful when the goal is to improve one part of an existing process, rather than design a new system from the ground up.

- Determine priorities. Some priorities may be defined by the need to build a working system and some foundation elements may be unavoidable. However the sprint prototypes need to start supporting user stories as soon as possible, and the highest priority ones should be targeted first. That way the product may be useful before all the features are even complete. Decide on priorities in conjunction with the other members of your team (including client and stakeholders). The most common tasks should have the most direct paths through the product experience and the interface. Less common tasks may then require a bit more effort to initiate.

This fits in with other good user experience design principles. Reveal the complexity of the system slowly. Expose the advanced features once the

user is familiar with the basics. This also provides a shallow learning curve which is important when products must introduce themselves to their users, as is the case for many online apps. Hinting as the next step supports this process, as does providing a progress indicator showing how much more effort is required before some reward or result is provided.

- Ensure you know who the stakeholders are. The person paying for the project is clearly one, but so are the people who will be using it. Consider also the other forms of user; the support staff, administration and maintenance team, supervisors and accountants. They may all have stories that need to be considered. In addition to user stories, consider developing personas for common groups of users. These provide a way of stereotyping these groups but in a way that makes their unique needs explicit.
- When you've produced your design documentation, architecture diagrams, user stories and interface mockups, share these with all the stakeholders. Explain back to them what you've understood and see if they agree. Listen for indicators that they really want something different. Concentrate on describing the experience they will have. The low-level technical details will be relevant only in those cases where you are dealing with stakeholders familiar with development processes.

*Development roles:* During the development process, team members communicate in the following ways:

- Stand-up meetings ensure that everybody is aware of who is working on what. This provides the opportunity for direct interaction between individual team members outside the meeting if a potential issue is identified.
- Team members communicate through the task tracking board. This ensures that two people don't start working on the same task independently (you

write your name on it when you move it to the “in progress” region). Points to note can be added to the card so that those continuing on with a later task can be aware of this information. The “completed” task section provides a way of monitoring progress of individuals and of the team with respect to the sprint time line.

- Team members may work directly with one another, or in small teams, on completing tasks. Pair programming is one strategy used where complementary skills or roles can combine to provide a better outcome than each member of the team working independently.

Use common sense as well. The intent with the agile process is that it adapts to the teams that use it, rather than vice versa (<https://agilemanifesto.org/>). If you see problem, then devise a strategy to deal with it.

#### 2.3.4 *Task tracking*

Task tracking is accomplished on a physical or virtual board, with each of the tasks contained on a small, movable note. Different regions of the board (usually columns) represent different states of the note.

Suggested regions to include:

- Goals: These are the overall goals of the current sprint. These notes never get moved, but rather serve as a reminder of what the purpose of the sprint is, and what the aggregation of the collected tasks will achieve at the end of the sprint.
- Pending: The backlog of tasks that still need to be started. Some notes in the pending list may have names associated with them, others may be available to anyone with the time or inclination to take them on. Tasks may be prioritized, and priorities in the pending category can be easily changed during the sprint. Changing

priorities of tasks in other categories is a lot more complicated.

- In progress: A task that has someone working on it. A convention such as: “before starting a task, move its card from pending to in progress, and write your name on it” provides a way of preventing duplication.
- Complete: The task is done, and any dependent tasks can now be started.
- Accepted: Some organizations may have procedures for double checking any completed tasks, for quality assurance purposes. Others may just require a project manager to transfer the task into this category so that they can keep track on progress within the sprint.

The tasks described in the notes need to have clear and testable outcomes associated with them. These could also be supported with unit tests where the task is related to the development of particular software function.

### 2.3.5 *Version control*

 Component 23.1.1 on page 601 outlines the approach required to set up a version control system to manage your project. Regardless of the specific version control system and host platform there are a number of principles common to working with version control.

A version control system consists of a repository of your project assets. This will include the source code files and often other assets such as the art work and configuration files required by the project. Version control systems are different to just making a backup copy of your project. Specifically they do two main things. They keep track of all the changes that have ever been made to each file. This makes it possible to recover previous versions should anything go wrong at any point. Secondly they allow multiple developers to co-operate on the same project. Each will keep their own copy of the repository, and will

synchronize their copy with the master version at regular intervals.

What is the minimum that each team member needs to know? Working practices within any team will typically be negotiated within the team. However there are some principles that could be considered universal.

You need to adapt your workflow to make use of the version control system. Firstly, whenever you start working on a task you need to update your local copy so that it is up to date with the master version. This ensures that you have the most recent changes contributed by other team members and are not developing solutions that are out of date with the rest of the team.

You then proceed to work on your local copy as you would normally do. Complete your task, and test that it works. Once you are satisfied then you can upload the changes back to the version control system.

These steps represent the bulk of your interaction with the version control system. Occasionally it may turn out you've changed the same file as someone else. Often the version control system will manage this but on occasion you may be required to merge the changes. The version control system will present the two alternatives (your changes compared to those of your team member). Use your knowledge of what has been done to make a decision as to which of the two should be in the final version of the file. Once the problem is resolved you will upload this to the version control system.

You do need to be aware that you are working with others during this process. Some common conventions are associated with polite behaviour.

Don't break the build. Adding an asset to a working project that stops the project working means that any other member of the development team is no longer able to build and test their own work. A typical example is including some program code that fails to compile. Such an error will render the project unusable until someone fixes this. This is a distraction for anyone else trying to

meet their own development deadlines so they will typically make you very aware of what you have done. As a general principle: make sure the project builds and runs with any of your changes before you update the version control system.

If someone else breaks the build then let them know politely. Mistakes can happen.

Do update the version control system with your changes on a regular basis. The longer you delay, the further the project will diverge from the version you are working with. If the project diverges too much then your work will become increasingly harder to integrate and at some point will risk being completely outdated before your share it.

Choose wisely which files you share with the version control system. Many development environments create temporary files to improve the efficiency of their operation. Including these in the version control system will slow down access for every team member. As a general principle don't use version control to manage any file that can be completely generated from other files (that are managed by the version control system). When you upload files to the version control system review the list presented and make sure those uploaded correspond to the changes you have made.

Changes committed to the version control system can, and should, have a meaningful comment included as part of the change. Your team may have conventions about what information needs to be included. This includes providing enough detail for other team members to understand what the effect of your update will be.

This overview represents the most common operations used in version control systems. Most support more advanced operations. For example, your team may have multiple branches representing different versions of the product. This can be used to allow critical updates to be made to published versions of the product, while still keeping a separate stream to support the development of newer versions.

### 2.3.6 *Testing*

Test driven development is a good habit to get into, particularly as you leave the shallows of novice programming practices and small assignment sized projects and move into the deeper waters of team based practice and larger projects consisting of interacting software components. This example explores unit tests; additional scripts that are written to call a function in your existing code with some sample data, and then verify that the results produced match what is expected. This does not eliminate the risk of bugs, but does provide a very satisfying experience when you see a bank of green lights pop up after tweaking a part of your project.

Unit tests are well suited to non-interactive applications where the majority of functions transform data from one format to another. Our focus is on interactive applications making use of live data, human input, interacting software components, and the peculiar ability of real-world data to immediately spot the flaw in any piece of code. Automated testing does not completely solve all of these issues but this example demonstrates how some elements of a process spanning several frames can be validated using the testing tools available.

The component under consideration is a function that causes the project to change from one scene to another. This is useful in a number of scenarios. A common situation is the change over from a title/instruction scene which is provided when the application starts, to another that contains the primary experience associated with the application. This might seem like a relatively simple operation that can be achieved in a single step, using a convenient engine provided function. However there are complicating factors such as the need to retain some content from one scene to another, or to manage a delayed scene change which allows the current scene to continue running until the next scene has completely loaded.

Consistent with recommended practice, this example develops the test cases first before writing the code that

will be tested. This satisfies two goals:

1. The tests are written according to the desired functionality, rather retro-fitted to known working cases in the code.
2. The tests must all fail at least once so they can be verified as working (both fail and pass code pathways are used), and they should all fail when calling the function they are testing before that function is written.

 Component 23.2.1 on page 605 demonstrates how unit tests can be included in enhanced reality application development processes.

# 3

## *Coordinated Realities*

### *3.1 Project*

Your client is focused on training nurses for surgical procedures. During the conversation that explored the goals it becomes apparent that the emphasis is not on the traditional surgical simulation and the skills associated with process and procedure. Instead the goal is to improve communication between all members of the surgical team. In particular nurses provide an important supporting role and need to be able to anticipate the needs of the surgeon which are often communicated non-verbally by monitoring the progress of the surgery and the issues that may arise at each stage. Additional communication occurs in the form of hand gestures.

The planned experience will be focused on the physical presence and telepresence region of the participant engagement triangle. The client anticipates that multiple participants will be present in the experience simultaneously to play the various roles. However participants will not necessarily be all present in the same physical location as this sort of training practice needs to take place when the opportunity is available. The experience will need to take place wherever the participants can find some free space, so an augmented reality experience on a smart phone has been identified as the format for the experience.

The question to be answered: What are the ways to achieve the communication goals for this experience?

### *3.2 Coordinating Reality*

Many contemporary virtual environments present isolated pocket universes. In general this is not consistent with the concepts of enhanced reality that integrate with the large and complex physical world. Many of the early virtual reality engines [Bangay et al., 1997] were engineered around multiverses of many connected worlds supporting the simultaneous interaction of many participants.

As a consequence it would be ideal if every enhanced reality application developed from the components provided in this book could automatically provide a minimal level of support for shared experiences and collaborative interactions. This chapter is provided for that purpose. Those eager to get onto the specifics of the individual enhanced reality components can skip this chapter.

This chapter describes how to set up projects that support generic awareness of other participants, to enable basic interactions between individuals. These are usually easier to build into the project at its inception rather than trying to retrofit them at the end.

### *3.3 Mutual Presence*

Mutual presence is achieved when all participants are sharing the same experience. This includes the physical presence resulting from other participants participating at the same location as well as the virtual presence achieved through the depiction of avatars of others who are sharing the same experience but from other locations. Presence is usually achieved through synchronous activities that take place at the same time but could also potentially be achieved more abstractly by conveying an awareness of others through effects that they have directly or indirectly on the shared state.

The individual components of enhanced reality systems tend to capture and present information about the setting and the user. A shared environment is achieved by ensuring that this information is shared amongst all participants. For example, tracking the pose and gestures of each user provides data that can be used to position and animate virtual representations of these users.

At time of writing, the accepted approach to performing networking with Unity software is a series of programming libraries known as UNet. This is in the process of being deprecated but its replacement is not yet established. As such,  component 21.7.1 on page 583 demonstrates how to set up a collaborative virtual environment using UNet and remains useful for short-term projects. Many of the principles described are applicable to other platforms as well so this example is retained to illustrate these concepts.

The lack of a current viable alternative in the Unity software engine is likely to encourage designers and developers to utilize third-party solutions to providing a networked virtual environment. One such capable platform is Photon,  with a comparable component described in component 21.1.1 on page 498, and this is the recommended approach to follow at present.

Mutual presence is typically demonstrated by shared the position and orientation of each participant with one another. This process can be readily extended to apply to the movement of other objects in the scene, but also to sharing other types of information. A text chat facility results from sharing strings of text that might be displayed on each user's avatar or on a shared message board. Other forms of information might include information sensed from the controllers being used, or to share documents or other forms of graphical asset.

### 3.4 Matchmaking

The networking examples deliberately avoid managing the connection establishment process in any detail. They use a quick and dirty approach of assigning all members to the same lobby and room. This does help avoid bloating those examples with additional detail. However in this section we concentrate on the process of connecting all participants to ensure that each group is added to their desired space.

 Component 21.2.1 on page 507 shows how to provide a visual representation of where each participant is located. While the representation used is simplified, it does provide a sense of distributed presence so that everyone has a sense of who is sharing their virtual surroundings.

This example did get out of hand. Originally it was intended to be a demonstration of the lobby and room facilities. However there was also an opportunity to explore a facility missing in many conferencing and online teaching tools around managing breakout rooms. In particular communication between rooms is usually limited. This example demonstrates how communication back to the lobby can be achieved so that an instructor can be notified when their presence in a particular room is required. At that point it became worthwhile to provide a persistent (for the life of the room), room based text chat facility supporting multiple participants in each room.

### 3.5 Conversation

Communication in virtual and augmented reality can make effective use of action information to communicate intent and even body language. However traditional forms of audio and video communication can also be relevant.  Component 21.3.1 on page 526 presents a component that allows a voice based chat to be included into an enhanced reality application. Video based com-

munication can be more challenging due to the need for efficiently managing a complex and time-critical information stream. Concepts relating to this process are part of  component 21.4.1 on page 532.

### 3.6 Coordinating participants

Multi-participant interaction is a key dimension to building enhanced reality applications. The facilities provided in the virtual and augmented reality engines support most of the common scenarios. There are occasions when more complex networking operations might be required and this is an opportunity to work directly with the underlying communication protocols.  Component 21.5.1 on page 549 demonstrates a component that uses these facilities to exchange some information specific to the experience's context.

### 3.7 Cloud Coordination

Several components demonstrate ways in which information is shared between mobile devices to achieve collaborations in the virtual environments. Component 21.5.1 ensures a shared understanding of the position and movements of other participants in non-co-located settings. The representation used is abstract as the experience involves large scale worlds and other participants are usually not directly visible in physical or virtual representations. Other examples (components 19.3.1 and 19.4.1) share information relating to individual actions in ways that can be perceived by other participants in the same environment.

In these cases, each participant communicates directly with one another even if it is mediated through a central server. In other cases we don't need participants to be aware of each other. However, the contributions of each participant can still be worth sharing. An augmented

reality experience may involve scanning a particular location and annotating key features. This information would improve the experience for the next person to visit that location. Cloud coordination involves uploading a record of one participant's engagement with a location so that future participants can benefit from it. It also provides a way of ensuring that properties of a location persist through successive visits.

 Component 21.6.1 on page 570 is a component that makes use of cloud anchors to track feature points at a location across multiple visits.

# 4

## *Visual Tracking*

### *4.1 Project*

Your client is building a table-top board game enhanced with virtual content. They have several expectations. The experience will take place on a table top (such as the family dining table). There will be cards involved and it would be desirable to have virtual content overlaid onto these. The cards will be active play pieces in that they will be moved around, and placed together to form hands or decks. The centrepiece of the table will be a large 3D structure such as a building or region of terrain. This needs to adapt dynamically to the progress of the game (i.e. be able to present progress in the game). Players should be able to walk around the table and view the game experience from all reasonable directions. The client assumes that the experience will be produced using a smart phone.

The question to be answered: What advice can you give the client to best achieve this goal?

### *4.2 Visual Tracking*

Augmented reality typically mixes a view of virtual content with physical elements. The two forms of reality need to interact as if they are aware of one another. Since physical objects lack awareness, the physical world has usually

been limited to showing synthetic content as an overlay, perceived through a boundary device such as the screen of a smartphone. Virtual objects can be aware of physical elements if the physical object is capable of communicating its information. Again this is achieved through the intervention of an external device, such as the camera of a smartphone.

Ideally we would like active physical objects capable of communicating their own state and responding to the information universe. Until then augmentation is provided through the sensing and actuation capabilities of those physical platforms that do possess such capabilities. This chapter deals with technologies to visually track objects in the physical world and the use of this information to manage interaction between physical and virtual spaces.

The typical process of sensing the location of a physical object involves:

- Locate feature points of the object in the 2D camera image.
- Work out the location of the feature points relative to the camera in 3D space.
- Using the known location of the camera relative to the world, work out the location of the feature points relative to the world.

Variations on this process are used in different scenarios. If the physical object is moving and the position of the camera is known then the process described above is used to track the position and trajectory of the physical object. If the object is in a known fixed location in the world, then the process instead is used to find the position and trajectory of the camera through the world. Knowledge about several adjacent feature points on the object is required to convert from 2D to 3D positions, and to infer information about the orientation of the object. This assumes that feature points on the object are in known and fixed positions on the physical object.

The first step in any of these processes is detection of feature points in the camera image.

### 4.3 *Feature points*

Any point used for visual tracking needs to be reliably identified in each of the photographs taken by the camera. It should be unique and able to be identified without ambiguity. Regular repeating patterns are thus an issue as it becomes hard to identify the position of a point uniquely, particularly when the camera may only see part of the pattern at any time. Plain surfaces of uniform colour are the worst case scenario being an infinitely tiny pattern element repeating endlessly, and lack any distinguishing feature.

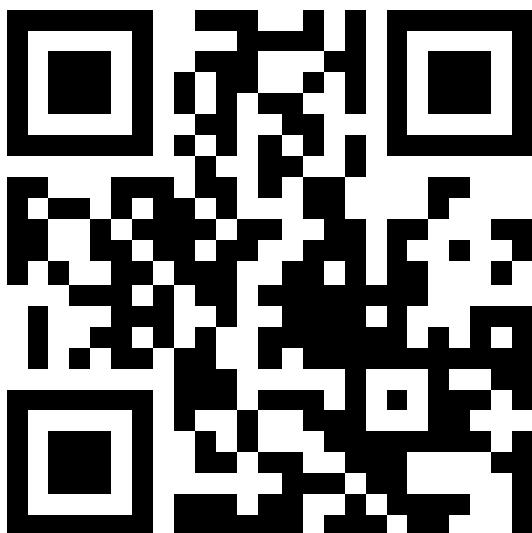
Thus we need to identify the feature points in an image: those points that are visually distinctive. In addition these points should continue to be good feature points even when viewed from different orientations and when lighting conditions change.

Marker based tracking for augmented reality ensures the presence of good feature points by explicitly adding them to the physical world by printing or manufacturing objects with distinctive patterns (fiducial markers) that can be stuck onto objects in the physical world. Control of the form of the pattern allows them to be created with high contrast colours, and to be encoded in ways that are easy to process and uniquely identify (such as barcodes, and higher dimensional representations of such). Since the process for identifying such markers uses strategies that differ from those used by the human visual system it is possible to create markers that mix elements that are visually pleasing to humans (such as pictures or logos) with the blocky patterns that are extracted using image processing. The code embedded in the structure of the marker is used to identify the different markers used to tag different objects or locations in the environment.

QR codes, such as the one shown in Figure 4.3.1 demon-

strate some of the key features of a synthetic marker. The black and white squares are high contrast and help distinguish the individual bits encoded into the image. These bits encode messages or URLs that are embedded in the code. The longer the message, the more bits are required and the smaller each of the individual elements must be. Other colour schemes are possible but may impact on legibility. The sequence of bits includes an error-correction code that allows the information to be recovered even if 7-30% of the image is corrupted, not visible, or obscured by artistic elements. The bits are encoded in such a way that large regions of solid colour are avoided making it easier to detect the boundaries of even the smaller squares. The three large squares in the corners, and the smaller square in the remaining corner ensure that the size and orientation of the code can be determined even when viewed at an angle.

Figure 4.3.1: Key features of a fiducial marker are visible in this QR code.



Markerless tracking exploits feature points that occur

naturally in the physical world. This avoids the need to prepare an environment for use in an augmented reality application but can have trouble with typical human habitations that have plain coloured walls, and regular patterns in floor tiles and carpet. Markerless tracking may in some cases learn about the structure of the environment during an initial scanning stage, but in other cases may require human intervention to capture images of the physical setting to train the system. The markers used can incorporate the 3D structure of physical objects, for example by capturing views from different angles or by approximating objects using simple shapes such as cubes.

#### *4.4 Finding feature points*

There are several techniques for processing images to identify feature points. The following description outlines one such approach. New proprietary systems are being created constantly which are usually silent on the specific refinements that they use. The description provided outlines some of the key concepts, and can be used by an augmented reality application developer to identify particular usage issues and opportunities.<sup>1</sup>

Particular characteristics of good feature points and their surrounding regions can be identified. For example, corners are often on boundaries between two regions with different visual patterns. Corners tend not to repeat in any particular direction the way regions on a edge do (as you move along the edge). Identification of a particular characteristic then leads to the next step of algorithmically measuring that characteristic for regions across the image.

Measures associated with several of these characteristics can then be combined into a feature descriptor. For example, corners might use a feature descriptor that contains values representing:

- The number of intersecting edges in the corner.
- The angle between each pair of adjacent intersecting

<sup>1</sup> Further details available at: [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_feature2d/py\\_table\\_of\\_contents\\_feature2d/py\\_table\\_of\\_contents\\_feature2d.html](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html)

edges.

- Main colour of the image in the region between each pair of intersecting edges.

Distinctive feature points are those that have unique values for their feature descriptor. The measures can also be chosen to be *invariant* under particular operations. In this case, the angle between adjacent edges is invariant (retains its value) under rotation, whereas a measure of the direction of an edge (relative to, for example, the upwards direction of the camera image) would change if the camera is rotated.

#### 4.4.1 Corner detection

A measure for detecting corners is described below to illustrate a strategy used by image processing algorithms. The image is transformed into another image space. In this case, the coordinates in this space  $(u, v)$  represent the extent to which the image region matches itself when shifted in the direction  $(u, v)$ .

If  $I(x, y)$  represents the original image region around a potential feature point, then the transformed image  $E(u, v)$  is:

$$E(u, v) = \sum_{x,y} [I(x + u, y + v) - I(x, y)]^2$$

Regions containing corners cannot be shifted in any direction without significantly changing the image. Thus regions that contain corners maximize the function  $E(u, v)$ . After some approximation, a maximum for  $E(u, v)$  occurs when the matrix  $M$  is maximized, where  $M$  is  $\begin{bmatrix} I_x I_x & I_x I_y \\ I_x I_y & I_y I_y \end{bmatrix}$ ,  $I_x$  is the overall change in pixel intensity in the  $x$  direction, and  $I_y$  is the change in pixel intensity in the  $y$  direction. Intuitively a right angled corner would occur where both  $I_x^2$  and  $I_y^2$  are large and about the same size (a vertical edge and a horizontal edge).

This can be generalized to edges at other angles by computing the eigenvalues  $\lambda_1$  and  $\lambda_2$  of matrix  $M$ . Corners occur when  $R = \lambda_1\lambda_2 - k(\lambda_1 + \lambda_2)$  is big, which is where both  $\lambda_1$  and  $\lambda_2$  are large and about the same size.

The Harris corner detection computes  $R$  for all regions in the image, and marks those where  $R$  is above a threshold level as being corners.

#### 4.4.2 Feature detection

Corner detection is invariant under rotation, thanks to the eigenvalue based measure. However many corner detectors will perform differently when the image is scaled (imagine the augmented reality view coming closer).

Sharp edges from a distance may be part of a gentle curve close up.

The Scale-Invariant Feature Transform (SIFT) is one of the original approaches to identify features in a way that does not depend on scale. This approach exploits strategies that are also commonly used in other areas of image processing.

- The image pyramid. An image can be represented by a stack of images, each half the size of the one below it. Feature comparisons across all levels of the pyramid ensure that any structure is considered at all possible sizes. Shrinking an image requires that at least 4 of the original pixels be collapsed into a single pixel of the reduced image. These large regions are often combined through averaging in proportion to a Gaussian (bell-shaped) distribution so that pixels closest to the center of the new point count the most.
- The averaging of the pixels can occur using different distributions. The Laplacian (second derivative of the Gaussian, approximated as the difference of two Gaussians of different width) essentially subtracts a blurred copy of the image from itself and gives a measure of the curvature of the image. This acts as a blob detector

by scoring highly for a central region of consistent intensity surrounded by a region of a different intensity.

- Feature points are those that have local extreme values of space and scale. Thus they are distinctive by being different to their neighbours at the current level of the pyramid and at the scales above and below.
- A histogram of gradients is created by calculating the gradient (direction of intensity change) for each pixel in the neighbourhood of a feature point. These are added to bins representing each 10 degree change in direction. The number of entries in each of the resulting 36 bins then becomes representative of the region. The highest value in the histogram is taken as a reference direction, so that any comparisons are first done by aligning histograms so their reference directions coincide. This ensures rotation invariance (matching neighbourhoods don't rely on the angle of the camera when the image is taken).
- Histograms for several neighbourhoods around the feature point are concatenated to make up the feature vector describing the point.

#### 4.4.3 *Feature Matching*

Feature detection is not the end of the process. Features will be detected in the original markers when they are initially developed. Another set of features will be detected in the camera image when the augmented reality application is used. These features need to be matched, to tell whether the camera is looking at the markers.

This is often a brute force approach that aims to ensure that features in one image match to the feature in the other image that has the closest feature vector. Further constraints imposed may include requirements that the solution represent a feasible (affine) transformation corresponding to rotation, translation and scaling operations applied to the object in the original image in order to

reach the configuration shown in the second image. This is a requirement if 3D spatial transformations are to be derived. Objects that deform or explode are beyond the scope of these strategies.

## 4.5 Marker Tracking

- Component 19.1.1 on page 379 provides a component that uses Vuforia to provide visual tracking of markers.
- Component 19.2.1 on page 383 provides a component that uses AR Core to provide visual tracking of markers. It is worth comparing these two packages and assessing their relative merits. Particular areas to consider are the robustness of the tracking (including when the marker moves in and out of visibility), how well each responds to movement of the marker images, and how well each supports multiple concurrent markers.

## 4.6 Location Tracking

If we know the position of at least one of the markers relative to the world, then we can then infer the location of the camera relative to the world. This allows the trajectory of the camera through the world to be used as a component of our enhanced reality applications.

Tracking of an individual fixed marker would only allow the camera trajectory to be monitored during the times that the marker is visible to the camera. However many tracking libraries make use of additional information to continue to provide camera location even when views of the marker have been lost. Strategies used include:

1. Using the gyroscope on the device to determine the orientation of the camera. If the camera is not moving, this provides information about the 3 coordinates needed to represent the rotation of the camera, thus providing values for 3 of the degrees of free-

dom (3DOF) permitted for location. This works well when the camera is attached to a fixed position but can swivel (such as the head of a seated participant).

2. Inertial navigation offers opportunities to combine movement sensors, such as accelerators, with rotational sensors such as gyroscopes, to dead reckon all 6 degrees of freedom in a location measure. These sensors are often capable of providing very frequent measurements, particularly more frequent than camera based trackers, and so make applications much more responsive to motion of the device. Downsides include drift in the readings due to accumulations of errors in acceleration and velocity measurements. These become exaggerated through the integration processes required to calculate position and orientation. Noise filtering process can improve accuracy but introduce delay which manifests as latency in applications using the data.
3. The feature points in the scene that persist across several sequential camera images can be used to stabilize values from inertial tracking measurements. Rapid movements of the camera, or featureless scenery reduces the benefit of this process.
4. Feature points that appear to all lie on the same surface are used for detection of ground planes (horizontal) and wall planes (vertical). This provides some insights that can help fill in missing data. For example, camera height can be estimated as being around head height for an average human, if no better information is available. Planar regions of the environment can also be mapped out and used to adapt content to context.
5. The process of tracking camera position relative to the world, while at the same time building up a map of the environment is termed simultaneous localization and mapping (SLAM). Toolkits that approach the problem from this perspective tend to consider the process as one of iterative refinement where additional

information can be used to improve on estimates made previously. This viewpoints contrasts slightly with that of an AR application developer who may focus instead on making best use of information available at any instant, and ensuring the user experience is consistent thereafter.

- Component 18.2.1 on page 304 provides a component that turns the smart phone into a device whose position and orientation in space can be tracked. Such a component can be awkward to use while developing on a desktop platform so ■ component 18.1.1 on page 299 demonstrates how such functionality can be simulated, purely for testing purposes prior to deployment to the device.



# 5

## *Location based Mobile Augmented Reality*

### *5.1 Project*

Your client wants to promote tourism on their particular stretch of the coastline. A key attraction to the area is a local arts festival where works of art are exhibited at key locations along the coast. The clients goals are two-fold: to help visitor find and visit each location while the festival is running, and to provide access to the art work and tourism experience during the rest of the year once the exhibits have been removed.

The art works are varied in form; some might be typical static artefacts such as paintings and sculptures while others can be extremely original. Examples of the latter include those with kinematic (moving) parts, or cloth and fabric constructions that respond to the weather. Others might have an intimate connection to the structure of the surrounding landscape, by aligning to key geographical elements. An additional challenge is that some of the works are augmented reality experiences in their own right.

The question to be answered: How do you integrate the setting into the experience?

## 5.2 *Location (Setting)*

Tracking relative to individual visual markers may not be feasible when operating applications designed to be experienced at large scales. Typical location based enhanced reality systems operating over large geographical areas require solutions that work at planetary scales, such as global positioning systems (GPS).

## 5.3 *GPS concepts*

There are currently a number of competing global positioning systems in operation and being planned. The original GPS is owned and operated by the United States government but is widely used in many countries across the world. Originally a lower accuracy service (selective availability) was provided with higher quality service being reserved for military applications. This limitation has been discontinued and equal access is available to all. However several other countries have launched their own services to guarantee control over mission critical applications. Some GPS receivers are capable of working with multiple services in an attempt to improve quality of tracking.

GPS systems rely on a constellation of satellites in low earth orbit. Each satellite continuously transmits accurate information about the current time and the position of the satellite. The position of a receiver can then be estimated from the information provided by multiple satellites. At least 4 satellites are required to solve for 4 degrees of freedom (3 position coordinates, and an accurate time measurement).

Each satellite is fast moving and orbits the planet every 12 hours. Satellites will drop in and out of visibility to a receiver as they drop below the horizon. The various orbits are arranged so that at least 4 (often more) are directly visible from any point on the globe. However tall obstacles (mountains, large buildings) can reduce access

to the GPS signal.

Accuracy of a GPS location is limited. Typically positions are within about 5 m of the true position. Various enhancements are available to improve these estimates. Newer satellites use more modern technologies and additional communication channels which allow greater precision. Ground stations at known locations can measure the effects of atmospheric disturbances and share these with nearby receivers (over other communication channels) to allow them to compensate. Other location based information, such as presence of nearby wireless or cellular network access points, can be used to improve position estimates. The latter strategies are particularly significant when it comes to tracking inside buildings which block satellite signals.

 Component 17.1.1 on page 226 retrieves the GPS location on a device and uses this to represent that position relative to a representation of the planet.

#### 5.4 Working with maps

Applications working with global location services need to be able to present the experience to participants. Issues of scale become significant. It is usually not feasible to represent all the content in the entire world in one scene, or even within the viewing window. Apart from the number of elements involved, most are also too far away to see easily.

One form of presentation that is reasonably familiar to most people is a map based layout. A top down view of the current region is scaled at a user controllable scale. Key locations and other participants are represented using some form of marker on the map.

Various online mapping services provide facilities whereby they take a given location and return a representation of the map around that location. Offline versions of some of these services also exist but require significant amounts of storage on the device being used. This also

translates to a significant amount of set up time when installing an offline version of such an experience.

A convenient map representation divides the world up into a number of tiles. Applications can then retrieve these tiles in the form of images.

#### 5.4.1 Map representations

A map tile is described with 3 coordinates:

- a zoom level. Level 0 represents the entire world in a single tile. Each successive zoom level doubles the number of tiles representing the world (actually 4 times, since doubles in the x and y directions).
- an x coordinate. This represents the horizontal (longitude) coordinate. It ranges from 0 to  $2^{\text{zoom}} - 1$ .
- the y coordinate. This represents the vertical (latitude) coordinate.

Many online map servers provide access to their map tiles directly through a URL which contains the zoom, x and y coordinates.

Given longitude and latitude coordinates, map tile coordinates can be found by:

$$\begin{aligned} x &= [1 + \text{longitude}/\pi] / 2 \\ y &= \frac{[1 - (\log(\tan(\text{latitude})) + \sec(\text{latitude}))/\pi]}{2} \\ \text{zoom} &= \text{zoom} \end{aligned}$$

Convenient functions to map between tile coordinates and geographic coordinates are available at: [https://wiki.openstreetmap.org/wiki/Slippy\\_map\\_tilenames](https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames).

Location (setting) can be represented using a series of image tiles from one of the mapping services as demonstrated in  component 17.2.1 on page 230. Some mapping services provide altitude data encoded in the map tile, which allows 3D models of the terrain structure to be included in the enhanced reality application.

 Component 17.3.1 on page 237 is a component that provides this facility.

# 6

## *Sensor systems*

### *6.1 Project*

Your clients are a consortium of organizations involved variously in: teaching primary school-aged children, developing and deploying enhanced reality teaching and learning experiences, and in providing teaching analytics that measure the effectiveness of learning and teaching. This group believes that enhanced reality experiences offer improved educational experiences and want to be able both identify opportunities for improvement but also find ways to measure this.

There are a range of requirements for this project that have already been identified:

1. The learning experience needs to be individualized.  
Each student learns at their own rate, and progress for each student needs to be monitored and shared with the teacher.
2. The teacher needs to be aware of how each student is progressing. Ideally this information needs to be available during class time so that feedback can be provided when and where it is required.
3. Students are in the early stages of primary school.  
Active learning is encouraged with opportunities for investigation, physical interactions and social engagement.

4. This is not a toy. Solutions developed need to apply to a range of lessons rather than be customized to a single one. A software solution that be easily added to third party applications, or a completely separate hardware/software solution would help meet this requirement.
5. Students will be engaging with enhanced reality experiences which offer ways of monitoring student engagement and interaction in ways that may not have been possible previously. However data collection should not be disruptive nor intrusive.

The questions to be answered: What opportunities are available to better understand how students are learning?

### 6.2 *Sensing*

One of the forms of augmentation inherent in augmented reality is the ability to sense and perceive the participant and their environment, and to be able to use this information to enhance the experience. This process involves being able to collect a signal from one or more sources, and then being able to process that signal in a way that makes sense of it.

A complete exploration of all the possibilities inherent in the field of signal processing is well beyond the abilities of this book. However there are a number of approaches in common use in virtual and augmented reality systems that are worth distilling into components that can be reused in different situations.

### 6.3 *Sampling*

In a perfect world, measurements taken from sensors would be represented as a continuous signal using unlimited precision. You can probably guess that every adjective used in the previous sentence does not apply. The quantities measured by the sensors attached to the digital sys-

tems used by enhanced reality hardware are both discrete and quantized.

They are discrete because they are measured at set time intervals, usually once a frame, rather than at every instant of time. This means that the measurements are actually *samples* of the continuous signal that may exist in the physical world. These samples are taken at discrete intervals and all information about how the signal behaves between these intervals is lost. We can expect that losing this information will have an effect on what we can do with the signal. The samples are usually taken at discrete *time* intervals but other forms of sampling are also possible. For example, a digital camera will sample the *space* of the room when it takes a photograph, with each image element taking a measurement of the colour or brightness of a different point. A video stream consists of a series of images taken at different time intervals, and so any pixel in a video stream is sampled with respect to both time and space.

Sensor measurements are quantized because they are represented digitally using a finite number of bits after having been transformed from an analog form. The values must be rounded off to the nearest value that can be represented in this form. This affects the precision with which an individual measurement can be represented. Once again, information from the original signal is lost.

Sampling and quantization represent losses due to sensing that assume that the sensor is actually able to measure the original signal correctly in the first place. However, not even that is true. Various forms of noise are likely to further contaminate the process. There may be other factors in the environment that contribute towards the signal being measured. For example, the light intensity signal from a fixed laser source (used in a positional tracking system, for example) will have further light contributions from the ambient light in the room, with additional changes from light reflected off the people in the room which changes as they move. The sensor itself

may add extra noise, by perhaps being more sensitive to light when the temperature is higher. Variations in temperature are then included as part of the measurements of the signal.

We would like to assume that these sources of noise produce effects that are small compared to the signal being measured. In this way, most of the noise might actually disappear when the signal is quantized. Additionally we assume that the noise is not correlated to the signal, and could be cancelled by averaging enough readings. Neither of these assumptions are true in all cases.

## 6.4 Sensors

### 6.4.1 Orientation sensing

A sensor provides access to a stream of sensor readings, usually at a steady rate.  Component 18.3.1 on page 308 demonstrates how to access one particular sensor typically found on platforms that support inertial navigation. The gyroscope is a sensor that measures the orientation of the device. This is typically used in an augmented reality context to ensure that the view of the virtual content is aligned with the direction in which the device is facing.

Smart phones are very flexible devices and contain a significant number of sensors. While typically used as a presentation device in an enhanced reality context, taking a sensor view of these devices reveals their potential as a source of data. A smart phone can be easily repurposed as a controller device. Inertial navigation and optical tracking allows their position and orientation to be sensed. The touch screen itself can be regarded as a source of buttons, sliders and other control information complemented by the ability to dynamically change the visual appearance at any time. The flexible controller component in  component 19.3.1 on page 387 is a reusable component that can be adapted to many projects. It makes use of coordination

components to relay the sensed information to other devices and thus allowing physical separation of the sensor platform from the presentation device.

## 6.5 *Signal sensing*

Sensor data can sometimes be used directly but often needs some form of processing to extract aspects of the information that is present in the data stream. For example, the accelerometer on a device captures all the forces that are applied to that device. Holding a smart phone in your hand means that the phone's accelerometer picks up all acceleration; including that of your body as you move around, any gestures you make with your hand such as waving, as well as any subtle movements resulting from the autonomic movements of your body due to breathing or your heart beating. Separating these different signals requires some signal processing.  Component 20.1.1 on page 475 demonstrates a heart rate monitor that filters out only the part of the signal corresponding to the typical frequency range for a human heart rate.

This component is intended to demonstrate a common signal processing operation rather than to provide a reliable heart beat sensor. However a plausible result can often be achieved if the subject lies on their back with the phone placed on their chest. Likewise, debugging the code and verifying the measurement can be accomplished by shaking the phone in your hand at a known frequency. A heart rate of 60 beats per minute corresponds to moving the phone up and then back down once per second.

## 6.6 *Scene structure*

The tracking facilities provided by augmented reality tool kits that offer visual tracking is another example of sophisticated sensing and signal processing operations. While it is convenient to use the information provided for the purpose it is intended, these processes can also

leak additional information that can be useful for other purposes. 3D feature point information is convenient for locating the position of the device and for identifying horizontal and vertical planes in the scene but can also be used to reconstruct the structure of the environment.

 Component 17.4.1 on page 246 demonstrates how information about the structure of the surrounding environment can be collated, even on devices that lack the depth cameras present on more recent devices.

# 7

## *Interaction*

### *7.1 Project*

Your client wants to produce yet another application to support online meetings and collaborations. They believe that an enhanced reality application should offer a better experience because of the opportunity to capture and present the subtle aspects of in-person communication that are lost in other forms of online interaction. It is clear from discussion that your client really wants to recreate the face-to-face meeting scenario as authentically as possible. As a responsible enhanced reality experience designer you feel that you should also make your client aware of opportunities for providing an experience that is even more effective than this baseline.

The question to be answered: What ways can you suggest to build a superior experience for supporting online collaborations?

### *7.2 Interaction and Feedback*

Interaction in an augmented reality context could be considered to be either augmented interaction, where natural interactions are enhanced through technologies (for example, recognition of gestures), or interaction augmentation where it is possible to interact in novel ways through new opportunities introduced by technological enhancements.

Similarly presentation opportunities can either be made better, or completely remade. Opportunities to sense inaccessible modalities fall into the latter category, such as the ability to feel wifi signal strength or observe nearly invisible movements.

This chapter covers opportunities employing augmentation technologies for use in enhancing interaction techniques or improving presentation of content. Further exploration of additional modalities of augmentation is covered in Chapter 10.

Traditional user interface elements can be recreated with a physical existence.  Component 18.4.1 on page 313 recreates some widgets using markers employing physical interaction and visual feedback.

### 7.3 *Gesture Recognition*

Gesture recognition is another strategy supporting physical interaction in enhanced reality. Some forms of gesture recognition make use of dedicated hardware to track hands or body.  Component 19.4.1 on page 400 uses the hand tracking provided by the Leap Motion to support gestures based on finger configurations. The full body tracking provided by the Kinect in  component 19.5.1 on page 412 allows a wider range of gestures.

Pose tracking is no longer the exclusive realm of dedicated hardware. Solutions exist that can run on generic desktop computers and even on mobile devices.  Component 19.6.1 on page 416 provides a component that tracks full body pose.

### 7.4 *Visual Recognition*

The most flexible interaction with a scene in an augmented reality context requires the ability to understand what is present in the scene. Visual recognition processes are becoming increasingly useful, particularly as they start

to be able to recognize a range of different classes of object.  Component 18.5.1 on page 319 provides access to this facility.



# 8

## *Capture of Physical Content*

### *8.1 Project*

Your client works for a museum. They have a large collection consisting of a diverse range of objects, varying in size, shape and surface. Display space is limited so large amounts of the materials available are inaccessible. Potential users include the general public who would be interesting in viewing the artefacts, but also researchers who need to work with accurate measurements of the properties of these objects.

This is a complex project and a single solution is not likely to be achievable. Instead your client would accept any incremental improvement in their situation. Areas that can be addressed include:

- Providing a way to view more artefacts within the existing physical space of the museum.
- Allowing a virtual version of the museum to house the overflow.
- Performing precise measurements of physical properties. One example is measuring the length of the beaks of bird specimens, to investigate a hypothesis that evolution is adapting species to a changing climate.
- Capturing detail in the artefacts, such as the structure of a fabric, brush strokes on a canvas, or the feathers

on a bird. This detail must then either be reproduced faithfully, or be available for further analysis.

The question to be answered: How can you use enhanced reality components to assist this client?

## 8.2 *Location Registration*

The museum display, described in section 8.1, represents a scenario where there are objects such as museum display artefacts placed at precise locations in the physical setting. Virtual elements may then be superimposed on these, or placed at other areas within the museum setting that are free from obstacles. A key requirement is to be able to register the virtual and physical elements and ensure that the spatial coordinates used to describe each remain consistently related to one another.

This is a subtly different problem from the spatial tracking problems described in Chapters 4 and 5. Spatial tracking involves being able to identify where you are within a location. A spatial tracking solution would define a coordinate system with an origin (usually the point at which tracking is initialized) and a set of axes typically defining an x, y and z direction. A good spatial tracking system would be accurate (distance measurements are calibrated and reliable) and consistent (returning to the same location reports the same coordinate values).

Location registration takes two coordinate systems and ensures that a coordinate reported by one solution can be mapped to a coordinate reported in the other (and vice versa). In the example context described above, the coordinate system for the virtual content needs to be registered to the coordinate system in the physical space. When each device used in the enhanced reality experience starts up, it will likely report its own coordinate system for the physical space depending on where, and at what orientation, it was when the application is started. Some drift in calibration may also occur with some tracking solutions. Thus having a way to re-register (re-align)

the two coordinate systems would allow this drift to be corrected during the experience.

### 8.2.1 Transformations

The mechanism to align two coordinate systems is a transformation. There are several categories of transformation:

*Translation:* These are used to move the origin of the one coordinate system until it is placed at the origin of the other coordinate system.

*Rotation:* These align the axes of one coordinate system until they face in the same direction as the axes of the other coordinate system. Several rotations around the origin by different amounts about various axes may be required.

*Scale:* Scale is used to adjust the relative lengths of each coordinate system axis. This can be useful when different measurement systems are used for each tracking system (e.g. one measures in centimetres, the other in metres), or when aligning a virtual world data set defined in microscopic (or macroscopic) scales. However, typically we would expect to change scale only under unusual circumstances.

There are several other types of transformation that are seldom used for location registration, but worth being aware of. In particular, reflections are one class of transformation to watch out for. A reflection (for example, a mirror image is a form of reflection) cannot be recreated by any sequence of rotations. Thus a development environment that supports transformations only in the form of translation, rotation and scale will have issues trying to perform location registration. Reflections can occur in cases where one coordinate system is defined as a 'left-handed' system and the other as a 'right-handed' system, since choice of handedness<sup>1</sup> is arbitrary and different organizations used different conventions. It is possible to

<sup>1</sup> Handedness of a coordinate system is defined by which hand has the thumb aligned with the z-axis, when the fingers curl from the x-axis direction to the y-axis direction.

achieve a reflection using negative scale values if that ever becomes an issue.

Typically we work in a three-dimensional Euclidean space. However, representing transformations efficiently is done using a four-dimensional projective space. While this has the benefit of adding a set of projection transformations to our repertoire, the principle benefit in practice is that any transformation can be represented using a  $4 \times 4$  matrix. Any two transformations can be combined (equivalent to performing one transformation and then the other) by multiplying two of these  $4 \times 4$  matrices, which produces another  $4 \times 4$  matrix. The consequence of this is that any sequence of transformations can be represented by a single  $4 \times 4$  matrix.

In conclusion, location registration can be achieved by identifying a single  $4 \times 4$  matrix which contains the translation and rotation operations to map a coordinate in one space to a coordinate in the other.

### 8.2.2 Achieving Registration

For the purposes of this discussion we assume the existence of two coordinate systems: the physical space being tracked by a device, and the virtual space containing a number of 3D virtual objects at defined coordinates. These virtual objects are already defined so that the two spaces will line up correctly if we just rotate and translate the tracked physical space coordinates in virtual space coordinates. We want a transformation that will map key known positions in the physical space to the coordinates of each of the virtual object.

There are several strategies for achieving registration:

1. Identify the origin and axis directions in the virtual space. Find the corresponding pose (position and orientation) in the physical space and make sure the device initializes with that pose. The required transformation is the identify transformation (no translation or rotation) and so life is simple. The limitations with

this approach is the ritual required to align the device during initialization. This often requires some sort of harness to hold the device correctly during this process and can be surprisingly difficult to get right. The software then has no support for coordinate registration meaning that loss of tracking at any point can require that the whole experience to be restarted.

2. During the enhanced reality experience select one of the virtual objects and transform it (in virtual space) to where it should be if it were correctly aligned with its physical location. The virtual space transformation from initial to final pose is then the transformation required to register virtual and physical coordinates. This solution is feasible, but is very sensitive to the final placement of the virtual object. Any small variation in orientation can cause massive position inaccuracies in virtual objects that are far away (the angular error is multiplied by the distance to achieve the positional error).
3. Repeat the previous process but align several virtual objects. Only position of the objects is required, which eliminates the error resulting from rotational inaccuracies. The set of initial positions, and the set of final positions are treated as two point clouds that need to be aligned. Point cloud alignment is even simplified as we know which initial and final positions correspond to one another. By aligning virtual objects on opposite extremes of the scene we can ensure good alignment for all of the intermediate points. The only downside is that at least 3 virtual objects need to be placed before the alignment transformation can be calculated. However, an arbitrary number of additional objects can be aligned to continuously improve the registration and to compensate for any misalignment in individual objects.

The point cloud alignment approach is described in more detail.

### 8.2.3 Point cloud alignment

The process for creating a transform which aligns two point clouds [Sorkine-Hornung and Rabinovich, 2017] involves identifying a translation transformation,  $T$ , and rotation transformation,  $R$ , that transforms points in one cloud,  $P$ , into their corresponding points in the other cloud,  $Q$ . In this case, we assume that each point cloud consists of the same number of points, and that the order in which points are listed in the cloud identifies corresponding points. The point cloud is a list of point positions, and entries at the same offset in the two lists represent the same point: one in virtual space and the other in the physical space. Points that do not have corresponding positions in both spaces can just be left out of the process, and not added to either list.

The translation,  $T$ , needs to translate cloud  $P$  by  $t = \bar{q} - R\bar{p}$ , where  $\bar{p}$  and  $\bar{q}$  are the centroids (or average positions) of all the points in  $P$  and  $Q$  respectively.

The rotation,  $R$ , requires several steps to calculate:

$$1. S = \begin{bmatrix} p_1 - \bar{p} & p_2 - \bar{p} & \cdots & p_n - \bar{p} \end{bmatrix} \begin{bmatrix} q_1 - \bar{q} \\ q_2 - \bar{q} \\ \vdots \\ q_n - \bar{q} \end{bmatrix}. \text{ The}$$

points  $p_i$  from  $P$  are represented as column vectors, and the  $q_i$  from  $Q$  are represented as row vectors, so this operation is actually the matrix product of a  $3 \times n$  matrix with a  $n \times 3$  matrix, to give  $S$  as a  $3 \times 3$  matrix.

2. The singular value decomposition operation is performed on matrix  $S$  to decompose it into three matrices that meet the relationship:  $S = U\Sigma V^T$ . Details of singular value decomposition are not directly relevant to this section and typically a generic matrix library that provides this functionality can be used for this step.
3. The transformation  $R$  is then calculated using:  $R =$

$$V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det(VU^T) \end{bmatrix} U^T.$$

This transformation is derived by minimizing the distance between points in  $Q$ , and the transformed points from  $P$ . Thus it is robust to minor errors in aligning individual points, and will even produce a better fit as more points are added.

Location registration is achieved by providing opportunities for the user to matching points in the virtual scene to known locations in the physical space. This can be done in many ways. A simple solution might be to have the user place their device on a series of fixed landmarks (points in the physical space) in the room during initialization, or during any calibration phase. These landmarks would correspond to known points already mapped out in the virtual space. The resulting transformation can then be applied to any position value produced by the tracking system (in physical space) before using the position in the application (where it is applied to the virtual space).

### 8.3 Scanners

Capture of 3D surfaces can be achieved by using multiple camera views; either from multiple cameras at different positions or a single camera that moves between photographs. The latter situation is effectively what we have used in chapter 4.

All of these systems involve recovering a 3D coordinate by using a pair of 2D coordinates, and triangulating based on further information regarding the pose and other properties of the cameras involved. Further permutations of the 3D scanning process are possible. For example, one of the cameras can be replaced with a projector that illuminates surface points in ways that identify the direction from the projector to that surface point. This range of scanning solutions is neatly categorized by Geng [2011] and summarized below.

It is worth considering that the goal of 3D scanning is to determine the  $(x, y, z)$  coordinates of a point in space. Given a pair of  $(u, v)$  coordinates in the form of pixel coordinates in a camera image, or two angular descriptors for the direction of a projected laser beam, it should be possible to determine four coordinate values. Thus we potentially have an extra degree of freedom. This may point to the problem being overconstrained (where the two camera images are not, in fact, completely independent of one another), or can suggest that there are ways of collecting information that would allow extra information to be captured. For example, we typically know both the coordinates of a feature point in a camera image, as well as its colour. There is potential for capturing further information about the 3D surfaces if we employ sensors effectively and creatively.

### 8.3.1 Structured Light

A key challenge with visual tracking is identifying feature points on the visible surfaces. Those feature points that are visible in multiple images represent fixed points in the scene that support triangulation, trilateration and tracking. Surfaces that lack visible features can still be scanned, provided we supply our own feature points. These can be projected onto a surface using a projector.

A benefit of projecting onto surfaces is that the pattern projected can be selected in ways that simplify feature point detection. As an extreme example, detecting a feature point would be easier if there was only one point projected, and even more so if we could turn it on and off at will. That single varying point would be readily detectable, particularly if we assumed the rest of the scene were static. Such an approach would be quite time consuming, however, as the process would need to be repeated for every potential feature point location. More efficient strategies include:

*Binary patterns:* Projecting a sequence of  $n$  black and

white stripe patterns, representing the successive bits in a binary sequence, allows any point on the visible surface to be classified as being on one of the  $2^n$  stripes. The use of black and white patterns provide high contrast, increasing visibility of the projected patterns. Features produced from sequential patterns do require that the scene does not change before all of the images have been projected.

*Gray patterns:* Black and white provides only two levels for patterns. Using various levels of gray (i.e.  $m$  different gray values) allows each digit projected to encode more information. The sequence of  $n$  patterns can thus encode numbers up to  $m^n$ . Shorter sequences of gray patterns can encode the same number of stripes as a longer sequence of a binary pattern. The limitation is that the process is more sensitive to variations in the light reflected off the surface.

Ideally the patterns projected should be chosen to maximize the differences between adjacent stripes.

*Phase shift:* This is described in detail in the next section, with details of how to use this in an enhanced reality component. Phase shift strategies are associated with the phase unwrapping problem, where the modulo  $2\pi$  operation inherent in the process produces ambiguities in the values recovered. Phase unwrapping strategies exist, but the problem can also be addressed by encoding additional information in the patterns that are projected. For example, the colour channels can be used to help label each iteration of a cyclic pattern that is projected onto the surface.

*Photometrics:* The brightness of a surface varies depending on the position of the light sources relative to this surface. By taking multiple images of the surface under different lighting conditions, the shape of the original surface can be recovered by reversing the process that causes this shading of the surface.

*Rainbow patterns:* The patterns projected on the surface can be coloured. This potentially increases the amount of information that can be displayed on the surface, and making it easier to find the points that correspond between the projected image and that seen by the camera.

*Segment patterns:* The projected images typically consist of stripes. A code can be embedded across the width of each stripe to identify it and help distinguish it from the other stripes visible.

*De Bruijn sequencing:* De Bruijn sequences are sequences of symbols, where any sequence of symbols of length  $k$  occurs at most once. This means there are no repeated patterns, even in the overlap between neighbouring blocks of length  $k$ . Thus if a  $k$ -sequence in the set is found, then it uniquely identifies that particular occurrence of the pattern. This is helpful in labelling stripes.

*Grid indexing:* Rather than using stripes to locate one dimension at a time, grid patterns can be projected. Codings such as those used for a pseudo-random binary array, a multivalued pseudo-random array, or coloured variations of these, ensure that any sub-block of a defined size is unique, allowing these to be accurately located once they are identified.

### 8.3.2 Phase shift

A sinusoidal pattern is projected onto the surface. Suppose that it makes a pattern on the surface,  $f(\mathbf{x})$  that varies with position coordinates  $\mathbf{x}$ . The intensity of light seen by the camera from a point on the surface is:

$$I = I_0 + I_P \cos(f(\mathbf{x}) + \phi)$$

where  $I_0$  is the base level of light reflected by the surface, and the contribution from the projector is given by  $I_P \sin(f(\mathbf{x}) + \phi)$ . The spatial variation in the pattern is determined by  $f$ , while the entire signal emitted from the

projector can be shifted by phase  $\phi$ , under the control of the projector operator. This phase represents an offset to the cyclic pattern and ranges across  $2\pi$  radians (or  $360^\circ$ ).

Suppose we project a sequence of three cyclic patterns, each shifted in turn by  $\frac{2}{3}\pi$  radians (or  $120^\circ$ ). For any point in the corresponding camera image, we record an intensity of:

$$I_A = I_0 + I_P \cos \left( f(x) - \frac{2}{3}\pi \right)$$

$$I_B = I_0 + I_P \cos(f(x))$$

$$I_C = I_0 + I_P \cos \left( f(x) + \frac{2}{3}\pi \right)$$

Expanding out the first term:

$$\begin{aligned} I_A &= I_0 + I_P \cos \left( f(x) - \frac{2}{3}\pi \right) \\ &= I_0 + I_P \left( \cos(f(x)) \cos \left( -\frac{2}{3}\pi \right) - \sin(f(x)) \sin \left( -\frac{2}{3}\pi \right) \right) \\ &= I_0 + I_P \left( -\frac{1}{2} \cos(f(x)) + \frac{\sqrt{3}}{2} \sin(f(x)) \right) \\ &= I_0 - \frac{1}{2} I_P \left( \cos(f(x)) - \sqrt{3} \sin(f(x)) \right) \end{aligned}$$

Similarly:

$$\begin{aligned} I_C &= I_0 + I_P \cos \left( f(x) + \frac{2}{3}\pi \right) \\ &= I_0 - \frac{1}{2} I_P \left( \cos(f(x)) + \sqrt{3} \sin(f(x)) \right) \end{aligned}$$

Thus:

$$I_A - I_C = \sqrt{3} I_P (\sin(f(x)))$$

and

$$2I_B - I_A - I_C = 3I_P (\cos(f(x)))$$

Then:

$$\frac{\sqrt{3}(I_A - I_C)}{2I_B - I_A - I_C} = \frac{\sin(f(x))}{\cos(f(x))} = \tan(f(x))$$

$$f(x) = \arctan\left(\frac{\sqrt{3}(I_A - I_C)}{2I_B - I_A - I_C}\right)$$

The term  $f(x)$ , often referred to as the 'phase', identifies the region of the cyclic pattern that is projected onto the surface. There are two further stages before this value can be used to recover the structure of the surface that it is projected on.

Firstly, the recovered phase is limited to a  $2\pi$  radians (or  $360^\circ$ ) range - it has been wrapped to this range. This would produce a stepped appearance in the final surface, so an additional process is required to unwrap the phase into a continuous signal that varies smoothly over a larger range.

Secondly, the phase value needs to be converted into actual depth values, or otherwise used to retrieve the 3D coordinates of the surface. While it may be possible to do this if you know the exact pose and properties of the camera and projector, a more common strategy is to estimate the transformation between phase coordinates and 3D surface coordinates by including a calibration phase where the phase at known 3D positions is measured. Typically a flat planar surface, or a reference object such as a physical cube of known dimensions, are used to create this mapping.

### 8.3.3 Phase Unwrapping

During the projection process, the continuous function  $f(x)$  is wrapped into the sinusoidal, or cyclically varying, function  $\cos(f(x) + \phi)$  and projected onto the 3D surface. Ideally this function  $f(x)$  would be continuous (or vary smoothly), and monotonic so that every value of  $f(x)$  corresponds to just one value of  $x$ . After the three phase shifted images are projected onto the surface, the value

of  $f(x)$  can be recovered. However, strictly speaking, the value recovered is not the original function, but rather a version of it modulo  $2\pi$  radians.

$$\begin{aligned} f'(x) &= \arctan \left( \frac{\sqrt{3}(I_A - I_C)}{2I_B - I_A - I_C} \right) \\ &= f(x) \mod 2\pi \end{aligned}$$

The phase unwrapping process attempts to recover the original  $f(x)$  from the wrapped version  $f'(x)$ . This can be done by identifying where the phase wraps between 0 and  $2\pi$  and either adding or subtracting  $2\pi$  to all the phase values in the neighbouring cycle. In practice, this is complicated by several factors. The image taken by the camera is not a perfect representation as it may have noise, quantization effects due to its limited intensity range, areas where the image is saturated or dark or pixel measurements may not vary linearly with intensity. Edges in the object being scanned can also introduce abrupt changes in phase (as they should) but these can potentially be confused with phase wraps. Areas of the surface shadowed from the projector contribute random phase values and this can confuse the phase unwrapping.

Robust phase wrapping needs to consider trends across the entire image, and to use trends observed to identify and resolve ambiguous cases locally. This can involve trying to minimize the total error across the entire image, or by identifying and building paths through the image and making sure these are consistent as they wrap back to their starting point. Errors in the phase unwrapping can cause issues during the calibration step. There are a range of solutions proposed for phase unwrapping. One of these, the algorithm provided by Herráez et al. [2002] is described in further detail below.

This algorithm [Herráez et al., 2002] builds paths through the phase values, aiming to make key decisions based on the most reliable values first. Reliability is determined locally in the camera image. If a pixel is similar in

value to its neighbours, then it is considered reliable.

In this case,  $x = (i, j)$  where  $(i, j)$  are the pixel coordinates in the camera image, and  $I(i, j)$  is the intensity value of the pixel at  $(i, j)$ . In theory we could use each of the red, green and blue colour channels but for the purposes of this algorithm we will just combine these into a single intensity value. Reliability in this case makes use of the difference of the differences between neighbouring pixels. This second order difference gives an indication of the curvature in the image intensity and is considered provide a more resilient measure of reliability (compared to the first order difference which represents the intensity gradient).

The first order difference  $\Delta_{\delta i, \delta j}(i, j) = \gamma(I(i, j) - I(i + \delta i, j + \delta j))$  is just the difference in intensity values between neighbouring pixels. Both  $\delta i$  and  $\delta j$  are either  $-1, 0$ , or  $1$  to refer to edge pixels in the  $3 \times 3$  block centered on the pixel of interest at  $(i, j)$ . The  $\gamma$  function ensures that the differences are always mapped back to the  $2\pi$  range.

The second order reliability combines differences in the first order differences in the horizontal, vertical and two diagonal directions:

$$H(i, j) = \Delta_{-1,0} + \Delta_{1,0}$$

$$V(i, j) = \Delta_{0,-1} + \Delta_{0,1}$$

$$D_1(i, j) = \Delta_{-1,-1} + \Delta_{1,1}$$

$$D_2(i, j) = \Delta_{-1,1} + \Delta_{1,-1}$$

$$R = \frac{1}{H^2 + V^2 + D_1^2 + D_2^2}$$

These second order differences are squared to ensure they are all positive, and the reliability  $R(i, j)$  is the reciprocal of their sum.

The pixels in the image are now represented as a graph, where each pixel is represented by a node, and edges exist between nodes if the corresponding pixels share a horizontal or vertical boundary. A weight is assigned to each edge and this is set to the sum of the reli-

abilities of the pixels corresponding to the nodes at each end. Kruskal's algorithm for finding a maximal spanning forest is then applied to the graph. Kruskal's algorithm builds multiple trees by adding edges (and their end nodes) in order of greatest reliability, provided the newly added edge does not already join two nodes in the same tree. A significant step in this process is applied when the new edge connects two trees in the forest. The phase offset at the edge where the trees connect is added to the existing phase,  $f'(i, j)$ , of all nodes (in the smaller of the two trees) so that the reconstructed phase function is continuous.

### 8.3.4 Coordinate Transformation

After the phase unwrapping, any pixel in the camera image can be described by a 3D coordinate:  $(i, j, f(i, j))$ . If we assume that this coordinate system results from a combination of translations, rotations, scaling transformations and perspective projections resulting from the relative poses of the projector and camera, and projections associated with the projector and the camera, then we expect that we could find a  $4 \times 4$  homogenous transformation that would relate  $(i, j, f(i, j))$  to the  $(x, y, z)$  coordinates on the surface of the object being scanned. Note that this assumption may not be completely valid but this context serves as a useful opportunity to describe how a transformation would be found.

We want to find a transformation  $A$  that transforms the 3D pixel coordinates into coordinates in the 3D space describing the surface of the scanned object. Since this transformation includes translations and projections, it is necessary to describe this using homogenous coordinates. For this, we add a fourth coordinate,  $w$ . Thus:

$$\begin{bmatrix} xw \\ yw \\ zw \\ w \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \begin{bmatrix} i \\ j \\ f(i,j) \\ 1 \end{bmatrix}$$

To convert from homogenous coordinates back to 3D coordinates, we have to divide by the  $w$  value. Hence this term is retained as a multiplicative factor in the surface coordinate representation on the left hand side. The pixel coordinate on the right hand side is a known quantity so we can set its  $w$  value directly to 1. It might be tempting to do the same to the  $w$  value on the left hand side, but that tends to eliminate the bottom row of the transformation matrix, and so we lose any opportunity to represent perspective projections.

We need to solve for the 16 values of  $a_{ij}$ , which are the components of our target transformation  $A$ . For a single pixel coordinate  $(i, j, f(i, j))$  and its corresponding 3D coordinate  $(x, y, z)$  we can multiply out this expression to give the equations:

$$xw = a_{11}i + a_{12}j + a_{13}f(i, j) + a_{14}$$

$$yw = a_{21}i + a_{22}j + a_{23}f(i, j) + a_{24}$$

$$zw = a_{31}i + a_{32}j + a_{33}f(i, j) + a_{34}$$

$$w = a_{41}i + a_{42}j + a_{43}f(i, j) + a_{44}$$

We have four equations with 17 unknowns - the  $a_{ij}$  and the value  $w$ . We can substitute to eliminate  $w$ :

$$x(a_{41}i + a_{42}j + a_{43}f(i, j) + a_{44}) = a_{11}i + a_{12}j + a_{13}f(i, j) + a_{14}$$

$$y(a_{41}i + a_{42}j + a_{43}f(i, j) + a_{44}) = a_{21}i + a_{22}j + a_{23}f(i, j) + a_{24}$$

$$z(a_{41}i + a_{42}j + a_{43}f(i, j) + a_{44}) = a_{31}i + a_{32}j + a_{33}f(i, j) + a_{34}$$

This is now three equations, with 16 unknown values. However, every extra set of matching pixel coordinates  $(i, j, f(i, j))$  and corresponding 3D coordinate  $(x, y, z)$  that

we produce from a calibration procedure generates three extra equations. We will need at least six of these points to determine  $A$  completely.

Since the calibration process is unlikely to yield perfectly exact values, we anticipate that there might be minor variances in some of the points measured during the calibration stage. We rephrase the problem from finding the perfect values for  $a_{ij}$  to one of finding the best values for  $a_{ij}$  that fit the given calibration points. This can then be solved using a least squares fit.

We write the problem in the form:  $Bv = 0$ , where:

$$v = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{14} \\ a_{21} \\ a_{22} \\ a_{23} \\ a_{24} \\ a_{31} \\ a_{32} \\ a_{33} \\ a_{34} \\ a_{41} \\ a_{42} \\ a_{43} \\ a_{44} \end{bmatrix}$$

contains the 16 values we need to find.

Assuming we have  $n$  calibration points, where for simplicity of notation we refer to  $p_i$  being the various  $(i, j, f(i, j))$  values and  $q_i$  being their corresponding  $(x, y, z)$ . The sets of three equations resulting from each calibration point collectively define the matrix  $B$  as:

$$B = \begin{bmatrix} p_{1x} & p_{1y} & p_{1z} & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{1x} & p_{1y} & p_{1z} & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{1x} \\ & & & & & & & \dots & \dots \\ p_{nx} & p_{ny} & p_{nz} & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & p_{nx} & p_{ny} & p_{nz} & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & p_{nx} \\ \dots & & & & & & & & \\ 0 & 0 & 0 & -q_{1x}p_{1x} & -q_{1x}p_{1y} & -q_{1x}p_{1z} & -q_{1x} \\ 0 & 0 & 0 & -q_{1y}p_{1x} & -q_{1y}p_{1y} & -q_{1y}p_{1z} & -q_{1y} \\ p_{1y} & p_{1z} & 1 & -q_{1z}p_{1x} & -q_{1z}p_{1y} & -q_{1z}p_{1z} & -q_{1z} \\ & & & & & & \\ 0 & 0 & 0 & -q_{nx}p_{nx} & -q_{nx}p_{ny} & -q_{nx}p_{nz} & -q_{nx} \\ 0 & 0 & 0 & -q_{ny}p_{nx} & -q_{ny}p_{ny} & -q_{ny}p_{nz} & -q_{ny} \\ p_{ny} & p_{nz} & 1 & -q_{nz}p_{nx} & -q_{nz}p_{ny} & -q_{nz}p_{nz} & -q_{nz} \end{bmatrix}$$

The solution to the homogenous least squares problem:  
 $Bv = 0$  can be found [Inkilä, 2005] by:

1. Decompose  $B$  using singular value decomposition into:  
 $B = U\Sigma V^T$ .
2. Then  $v$  is the transpose of the row in  $V^T$  that corresponds to the smallest value in  $\Sigma$ . This will usually be the last row, if  $\Sigma$  is ordered from largest to smallest.

All the points in the camera image can now be mapped to the 3D coordinates of the surface of the object. The steps of this process are demonstrated in  Component 18.6.1 on page 336.

#### 8.4 Scene capture and stitching

The previous sections have described processes that allow support alignment of point clouds with known configurations, and have also used structured light as a mechanism to acquire detailed point clouds from surfaces in the physical environment. These concepts can be directly

employed in the context of the project for this chapter by ensuring that virtual content can be precisely registered against physical locations, and can perform limited scanning of surfaces of small objects. A detailed reconstruction of the interior of a museum requires one further facility: the ability to connect fragmented point clouds from multiple individual scans into a coherent structure to represent large artefacts or entire rooms.

There are several strategies for scanning a single point cloud. Depth cameras, such as the Kinect, are already configured to use structured light or time-of-flight sensors to capture not only the colour of an individual pixel but also the three dimensional coordinates of that pixel. This can be used as a tool for quickly capturing physical elements that exist in the setting.  Component 17.5.1 on page 261 provides a component to access this information in the form of a point cloud that can be connected as a mesh using the known grid resulting from the pixel structure.

Assume that the input for the next process consists of several point clouds captured by such a device. The point cloud consist of an arbitrary number of vertices that are defined by their 3D position relative to a coordinate system defined for that point cloud only (i.e. there is no common coordinate system shared between clouds). Points may have further attributes such as colour or geometric relationships to other points, but we won't utilize those attributes in this section. The point clouds may overlap in that we expect each cloud to represent a scan of a physical surface, and the physical regions scanned would overlap. However, we cannot guarantee that exactly the same set of points from the physical surface are present in both overlapping clouds. Similarly, they may be scanned from different positions, orientations, and distances so even the scale of the 3D region containing the points may differ between two point clouds.

The goal is to be able to take overlapping two point clouds and transform one so that the overlapping points are aligned in 3D space with the corresponding points

from the other cloud. The process used for this is fast global registration [Zhou et al., 2016], which is related to the similar process used align image markers with regions in the camera view for augmented reality tracking (section 4.4.3).

#### 8.4.1 Feature identification

As with visual tracking (section 4.4), aligning point clouds requires identification of features. Here features provide a simplified representation of the geometry around each point of interest. A feature is represented with a feature description, which is usually just a list of numbers representing various geometric properties. Two features are considered similar, if they have similar values at corresponding positions in their respective lists.

In the case of a point cloud, each point can be treated as a point of interest. For any given point, the point and its relationship with its neighbours needs to be represented using the feature description. This description should be independent of the position, orientation and scale of the point cloud. An effective way to achieve this independence is to describe neighbour geometry in terms of the angles between neighbouring points. In the process used [Zhou et al., 2016, Rusu et al., 2009] this involves a few steps:

1. Calculation of normal vectors. Each point in a point cloud effectively exists in isolation. Typically these points are on the surface of a three dimensional object and so actually should have some relationship to their neighbours. The process of calculating normal vectors at each point is an attempt to reconstruct the shape of the surface that the points belong to. The direction of the normal vector at each point is perpendicular to the surface at that point.

The normal vector is useful as it start to provide a reference direction at each point, which can later be extended to create a local coordinate system at the point.

These coordinates are then useful for measuring any geometric properties that define the point cloud locally, such as the angular relationships to neighbouring points that will be used as distinctive features.

One strategy for calculating a normal vector at a point is to assume that the point and its immediate neighbours all lie on a plane through the point. Typically, with curved surfaces, this will not actually be the case but what we are after is an approximation of the local tangent plane on the surface at that point. Given that all points on a plane satisfy the plane equation:

$$(p - p_s) \cdot n = 0$$

where  $p$  is any point on the plane,  $p_s$  is the point in the point cloud that the plane must pass through and for which the normal vector  $n$  must be calculated. If we have a collection of neighbouring points,  $p_i$ , we can pose this as a homogenous matrix equation:

$$\begin{bmatrix} p_{1x} - p_{sx} & p_{1y} - p_{sy} & p_{1z} - p_{sz} \\ p_{2x} - p_{sx} & p_{2y} - p_{sy} & p_{2z} - p_{sz} \\ \dots \\ p_{nx} - p_{sx} & p_{ny} - p_{sy} & p_{nz} - p_{sz} \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = 0$$

As with any of homogenous matrix equation, this can be solved with established techniques including using singular value decomposition (see section 8.3.4).

2. Calculation of point features. The point features need to be a series of numeric descriptors that give the same measure regardless of the scale, rotation or position of the point cloud. They should also have the property that similar structures in the cloud should produce similar numeric values. A series of angles [Rusu et al., 2009] can be calculated to meet this property. Given a pair of neighbouring points,  $p_i$  and  $p_j$  with their corresponding normal vectors,  $n_i$  and  $n_j$ , we can define

a local coordinate system with axes of:  $u = n_i$ ,  $v = (p_j - p_i) \times u$  and  $w = u \times v$ . The following angular measurements can then be used as feature values:

$$\begin{aligned}\alpha &= v \cdot n_j \\ \phi &= u \cdot (\widehat{p_j - p_i}) \\ \tan \theta &= \frac{w \cdot n_j}{u \cdot n_j}\end{aligned}$$

In practice, the calculation of the normal vectors in the previous step are ambiguous. They can equally easily point towards the outside of the surface, or to the inside. A consistent direction is important in order to uniquely identify features. There are processes for ensuring consistency in normal vectors. These may include ensuring that neighbouring normals point in similar directions, and propagating this throughout the point cloud. Another approach uses the assumption that the point cloud is scanned from a particular viewpoint, and makes sure all normals face towards that viewpoint. Another strategy is to assume that the normal vector could point in either direction, and calculate the feature angles for both scenarios. Taking the largest value of the alternative feature measures would also produce usable features without having to make additional assumptions about the scene.

Since there are a variable number of neighbours within a set distance of any point in the cloud, a feature descriptor that just consists of the list of angles for each neighbour is not useful. Instead histograms are used. The angular range from 0 to  $2\pi$  (or 0 to  $360^\circ$ , if you prefer) is divided up into  $k$  partitions. Angles are assigned to one of these partitions. The feature descriptor is then found by counting the number of angles in each partition, divided by the total across all partitions. In practice, we use 3 histograms for each of  $\alpha$ ,  $\phi$  and  $\theta$ , giving a feature descriptor with  $3k$  elements.

3. A further refinement replaces the features with a

weighted average of the feature values in a region around each point.

### 8.4.2 Correspondence matching

Once each point in the point cloud has a feature descriptor that identifies its local geometric properties, the next step is to find matching points across two point clouds. Once corresponding points have been identified then a transformation can be found that moves these points into matching positions. Corresponding points are identified by finding a point in the other point cloud that has the most similar feature values.

A few extra steps are used to eliminate accidental matches.

1. If  $p$  and  $q$  are corresponding points from the two clouds,  $P$  and  $Q$  then the feature descriptor for  $p$  must be the closest match to the feature descriptor for  $q$  in features of cloud  $Q$ , but also the feature descriptor for  $q$  must be the closest match to the feature descriptor for  $p$  of all the features of cloud  $P$ . In summary, they must be the best option regardless of whether you search the feature descriptors of cloud  $P$  or cloud  $Q$ .
2. Given three sets of corresponding points  $p_i$  and  $q_i$ , for  $i$  in  $[1, 2, 3]$ , distance in either cloud should be approximately the same. Thus  $\frac{|p_i - p_j|}{|q_i - q_j|} \approx 1$ . This is one way of eliminating bad correspondences where triangles of points have very different shapes.

Some correspondences can be spurious - just accidentally similar. For example, regions of flat surface might correspond to any other region of a flat surface. The true test of a correspondence is when the corresponding points collectively all end up occupying the position of their partner once the one point cloud is transformed to align with the second. This constraint is a global one - we want as many of the corresponding points as possible to align once the correct transformation has been found.

Thus finding the correct transformation is a two step process, that can be repeated until it converges:

1. First identify valid corresponding points, as pairs of points with one from each cloud.
2. Identify the transformation that finds the best alignment across all corresponding pairs.

This optimization can be presented as trying to minimize the expression:

$$E = \sum_{p,q} \left( \frac{\mu}{\mu + |p - Tq|} \right)^2 |p - Tq| + \mu \left( \frac{\mu}{\mu + |p - Tq|} - 1 \right)^2$$

Here  $p$  and  $q$  represent the various pairs of corresponding points from the two point clouds. The term  $|p - Tq|$  needs to be minimized as it represents the distance between points,  $p$ , in the original point cloud, and the version  $Tq$  from the second point cloud after having been transformed by the current proposed transformation,  $T$ .

The term  $\left( \frac{\mu}{\mu + |p - Tq|} \right)$  is a function that returns 1 when  $|p - Tq|$  is close to 0, and drops off to zero for all other values. The factor  $\mu$  determines how quickly it drops off. This term is used to eliminate bad correspondences: points that have similar feature values but are not actually in corresponding positions in the two clouds. The optimization process starts with  $\mu$  fairly large which allows all the corresponding points to be considered, but reduces this on later iterations. This means that the bad correspondences will have less and less influence until only the points that produce a useful  $T$  value contribute to the value of  $E$ .

 Component 18.7.1 on page 354 provides an implementation of this process that can be combined with other point cloud scanning components.

For the interested reader, further details of the optimization process, and the extension to multiple point clouds, are available in the paper [Zhou et al., 2016].

# 9

## *Presentation*

### *9.1 Project*

Your client is the custodian of a local marine park. This is a protected region of the ocean including a small section of the coast. Visiting the region in person is not possible both because of restrictions on access to the park but also because the underwater environment is physically difficult to access. Fortunately they have a partnership with a university research team that is involved in mapping the area and is in a position to capture materials. The goal is to provide virtual access to this setting.

A key objective is to ensure that the local population appreciate the value of conserving this area. Since nobody receives any value directly from the park itself, this reward needs to be provided indirectly through an enhanced reality experience. Participants in such experiences need to be engaged, educated and provided with the benefits resulting from the presence of the park.

The question to be answered: What can you communicate to the target audience using an enhanced reality experience?

### *9.2 Presentation*

Inexpensive head mounted displays, particularly those that wrap around an existing mobile phone, are a conve-

nient way of deploying a virtual reality experience aimed at a non-specialist audience. Developing such applications needs to focus on the presentation, with interaction being a secondary concern. When controllers are available in such settings they typically provide a few controls that allow limited navigation through a set of fixed options.

■ Component 16.3.1 on page 199 is a typical component used to adapt content to the (now obsolete) Daydream headset.

### 9.3 Augmented Worlds

Portals are a convenient metaphor for blending the presentation of two different environments. In an augmented reality application they provide a mechanism to move between an augmented physical environment and an almost completely virtual setting. Portals can be created in a wide variety of ways. ■ Component 17.10.1 on page 287 presents one representation that is well suited to an augmented reality setting.

### 9.4 Scene presentation

Using a mobile device to present virtual reality will limit the amount of detail that can be included in a scene. One strategy to address this is to bake the various scene elements into a single asset that can be presented more efficiently. The principle takes various forms. The rich complexity of a natural scene can be ‘baked’ into a single panoramic photograph. These photographs can be captured in an instant with a dedicated 360° camera.

■ Component 17.6.1 on page 271 describes this process. A more accessible alternative is to take successive photographs and stitch these into a panorama. ■ Component 17.7.1 on page 278 presents a component that does this.

By default, photographs don’t capture the depth of

points and so the image appears to be attached to the surface of a sphere or cylinder when presented in a virtual reality headset. The depth effect can be recreated by taking photographs from viewpoints corresponding to the left and right eyes respectively.  Component 16.4.1 on page 205 describes how these can then be presented in a headset.

These photographs don't have to originate from the physical world. Sufficiently detailed 3D models can also be baked into a similar image based representation. This works particularly well for detailed but distant scenery where the static nature will not be noticed. As might be expected, the baked content is not well suited to representing interactive elements, and does not respond to movements of the viewpoint.  Component 17.8.1 on page 279 describes how to produce such images within Unity. This is particularly useful when capturing screenshots of existing experiences, which can then be used as a marketing asset for the experience when displayed in another headset.  Component 17.9.1 on page 282 uses a 3D modelling package to directly create panoramic images.



# **10**

## *Augmentation across Modalities*

### *10.1 Project*

Filled with goodwill to all humankind, you decide to be your own client this time and employ your now significant design and development skills towards helping those experiencing some form of sensory disability. Your goal is sensory substitution: using an enhanced reality experience to transform one sensory modality to another. A typical example would be a sensory substitution device to support a deaf person. This would make use of electronic sound sensors to detect sounds, and convert this to a visual modality; for example flashing a light when a particular sound occurs.

Sensory substitution does not have to be limited to human senses. A typical smart phone can perceive WiFi signal strength, for example. By turning this into an audible sound, it would provide its user with the ability to 'hear Wifi'.

The question to be answered: Given a sensory disability of your choice, how could you use enhanced reality to provide a superior quality of life?

### *10.2 Spatial sound*

Sound is modality that is capable of communicating spatial concepts.  Component 16.5.1 on page 211 provides

a component that positions sounds in space.

### 10.3 *Speech to text*

Understanding the meaning of sounds adds value to an enhanced reality experience. Speech is a form of sound that can be converted to a text based representation that is easier to extract meaning from.  Component 19.8.1 on page 458 presents one strategy for building a speech to text component.

### 10.4 *Haptics*

In addition to sensors, many smart phones contain actuators. These devices can act on the world. Typically this would be by displaying an image on the screen or producing a sound through speakers. Modern phones also include quite sophisticated devices to produce vibrations as a form of haptic response.  Component 20.2.1 on page 482 demonstrates how to employ this to produce a form of sensory substitution.

# **11**

## *Extending Legacy Technologies*

### *11.1 Project*

Your client is developing therapy for vision problems, such as amblyopia. As part of this process their patients need to be supported using a range of small applications intended in equals parts to encourage visual activity (i.e. exercise the eye muscles) and to entertain and distract the patient while undergoing other less comfortable aspects of the treatment. The major constraint is that the client is technology adverse. They are not in favour of significant extra ‘screen-time’ and have no desire to encourage their patients to play with any cool new toys.

As an enhanced reality enthusiast you know that their opinion will change once they get a chance to see what the possibilities are. In the meantime, however, you need to provide some engaging experiences that can work on existing platforms (i.e. a web browser) while providing a pathway for reuse of your work once the advantages of the technology are truly embraced.

The question to be answered: How can you build enhanced reality experiences for platforms past and future?

### *11.2 Web Technology*

Many of the facilities used to create enhanced reality applications are being built directly into web browsers. This

offers opportunities to develop applications in a single development environment: HTML and javascript, and to have these run across a range of platforms from desktop to mobile without modification. Particular support libraries such as WebVR enables access to dedicated hardware such as the head mounted displays and controllers.

This chapter diverges from the Unity software focus to explore opportunities with level 1 technologies; the hardware and software that are immediately available to end-users.

### 11.3 *WebRTC*

Direct communication between human participants is typically achieved with voice and video in existing conferencing applications. Managing interaction between avatars has already been achieved (Bangay [2019], components 21.5.1 and 19.3.1) but has involved maintaining shared state such as position and orientation of individual virtual elements. Voice and video are time-sensitive continuous streams containing large amounts of information that make use of significant software infrastructures to solve problems around:

- Compressing information to reduce the quantity of information and time it takes to send while still retaining the elements that are most significant to human communication.
- Ensuring that all communicating participants can agree on the particular protocols and formats to use. This includes issues around distributing and supporting the software implementations of these across multiple different platforms.
- Dealing with restrictions on network structure imposed by security settings and other trends in building large and small networks (such asymmetries introduced in establishing connections by network address translation gateways).

There are advantages to using existing solutions for incorporating voice and video into enhanced reality applications. Communication technologies such as WebRTC already incorporate solutions to the problems listed. They also help with the bootstrapping issue: getting multiple sites set up to participate all at once. Exploiting existing traditional videoconferencing clients allows new prototype applications to access an established pool of participants until the stage is reached where all participants are willing to embrace and convert to the enhanced reality experience.

 Component 21.4.1 on page 532 provides a component to set up a video conference within a web browser.

#### 11.4 *WebVR*

WebVR provides a browser based abstraction of virtual reality hardware. This allows content authored to assume some generic facilities, and for the application to run whether dedicated VR hardware is present or not.

 Component 16.6.1 on page 216 is a component that employs WebVR, but mediated through the A-Frame platform that supports authoring of experiences.

#### 11.5 *WebAR*

Many web based augmented reality frameworks exist. In general these are javascript based libraries that make use of external visual tracking solutions such as the AR Toolkit, AR Core or AR Kit. As with WebVR many of these require a browser that has the external library built into it.  Component 16.7.1 on page 221 demonstrates augmented reality marker tracking uses a compatible web browser.



# **12**

## *Usability*

### *12.1 Project*

At this point you are likely to have a number of prototype enhanced reality applications available to you. We do need to know if they meet the goals defined by our client. One goal that we would assume is common to all experiences is: the experience needs to make it easier (or more efficient) to complete your task compared to your previous approach. One way to find out would be to take a participant through the journey described in one of our user stories. Their ability to effectively use the experience would provide an indication of the value and usability of the prototype.

It is worth noting that this is not about whether the experience works. That is a different problem, associated with testing and debugging. We would assume you have already completed that stage before any user gets to engage with the application.

The question to be answered: How do we know if our experience has met its goals?

### *12.2 Usability*

Apart from the aspects of component testing discussed in component 23.2.1 that are specific to working with virtual reality engines the bulk of software testing processes for

enhanced reality applications are identical to techniques used in other software engineering contexts. Testing usually starts with unit tests before proceeding to consider how software units interact with one another (integration testing) before culminating in system testing where the completed application is tested against the tasks that it is intended to support.

Another aspect of testing that is often overlooked, and that is vital to enhanced reality applications, is usability testing. This considers not only whether the software operates correctly, but whether it achieves its goal of providing an effective experience for participants, users and stakeholders. This includes evaluating whether use of the application is well suited to the tasks that need to be completed using the application. A typical but unfortunately common failure case is where the use of the system takes longer and requires more effort than previous approaches to achieving the same goals.

### *12.3 User experience evaluation*

User experience evaluation measures the usability of the product. This is the final stage of the design sprint, started in Chapter 2. The aim of the evaluation is to confirm that the participants can efficiently and effectively achieve the stated goals of the product while using the product.

The overall process involves finding some willing experimental subjects to use the product. These participants are given some defined goals; tasks that have to be completed using the product. They are not given step-by-step instructions on what to do. Their attempts to achieve the goal are recorded unobtrusively, for example by recording the session or by asking the subjects to describe their reasoning processes. These records are then shared with the development team after the evaluation is complete. The team discusses these and identifies issues and potential solutions that can be implemented in subsequent versions of the product.

### 12.3.1 Preparation for usability evaluation

A usability evaluation is a form of experiment. Unlike research experiments the goal is not rigor but rather discovery. Usability evaluations can often identify potential issues using a handful of experimental subjects, whereas much larger groups are usually required to achieve significant findings in scientific research. Usability evaluations are also able to test with a few subjects, make changes, and repeat this process. The goal is to identify issues and improve the product.

A working prototype is required prior to starting a usability evaluation. This needs to represent a known and testable state of the product. There is clearly no point performing an evaluation with a broken version of the application which has already known defects. The version being tested should represent a particular version of the application (reproducible from the version control system) so that any issues identified can be matched with the corresponding software elements.

Define a set of tasks that the test subjects need to complete. These should not be a list of the actions that they need to take (such as a list of instructions) since you are evaluating the application and its interface, and are not assessing the subjects themselves. Rather define them in terms of particular goals: “Use the application to ...”. These tasks should correspond to the goals that the application was originally designed to achieve, or to sequences of actions that the users of the application are commonly expected to complete. Usability evaluation can also assess more esoteric aspects of the system but these are usually lower priority.

The number of tasks to be completed in a single session should be limited to avoid exhausting the subjects. Typically the test should last no longer than half an hour. This might allow one long task or several smaller steps.

Subjects need to be recruited. Good subjects are willing (either volunteers, or paid for their time if your organization is generous). They should not be members of your

development team, nor anyone with prior exposure to the application. They should ideally be likely users of the application. There may be reasons for being flexible on some of these guidelines, for example if the application is targeted at a small and specific group then using these people would help ensure that their needs are met while simultaneously building interest in the product from the target market. The key principle is to have subjects who will provide meaningful insight into the operation of the current version of the product.

Write the script for the test session. Ideally you want to ensure that each subject goes through the same process without variations due to different instructions or test conditions. Following the script is one way to ensure this. The script should include:

- Welcome and thank the subject.
- Ensure the subject is willing to be recorded.
- Outline the process, taking care to instruct the subject that talking aloud and explaining their thought processes is an important part of the investigation. Mention that you will be observing to assess the working of the product and that this is not a test of the subject's abilities in any way.
- Explain the tasks (usually one at a time, after the previous task is complete). Use neutral language as far as possible to avoid introducing bias. For example, an instruction such as "We want you to ... to show how accurate the system is" could encourage a sympathetic subject to exaggerate accuracy related actions to please the experimenter.

Issues may still occur during the evaluation where the subject will ask for help. Supply assistance as required while trying to minimize influencing the subject. Make a record of any assistance that is given. Bear in mind that applications deployed in online stores assume no (or minimal) user support. A realistic evaluation scenario would

then be one where the subjects are given no extra instructions outside those already available in the application.

Set up recording devices. Make these unobtrusive if possible to avoid intimidating the subjects. Screen recording software can be used. Video recordings are common and should be statically mounted in an unobtrusive spot. It helps to be able to capture what the subject is doing, but also their expressions, actions and reactions. Only one experimenter should be in the room with the subject. This person should be neutral, and ideally not a member of the development team. The development team may watch through a one-way window but should not be allowed to interact directly with the experimental subject to avoid contamination of the results.

Augmented reality experiments that utilize several devices or involve moving around can benefit from a 360 degree camera to record everyone at the same time. Logs of actions and sensor information from each of the devices help diagnose issues but also can keep track of the actions of each user. Each event in the log file should have an accurate time-stamp.

### *12.3.2 Conducting the evaluation*

Set up an appointment with the subjects so that each subject performs the evaluation separately and at different times. It is worth scheduling some slack in the times to avoid keeping subjects waiting if test sessions overrun. This is particularly crucial where subjects are volunteers.

Ensure that the application is set up and working before bringing in any subjects. It is worth running through a trial evaluation with another team member who role plays the subject to make sure that all logistical elements work as intended. Remove any other software from the devices to avoid confusion.

Make sure that your recording devices work, have sufficient space, and are fully charged.

Follow the script as previously prepared. Try to engage

with your subject around your common purpose: investigating how well the application is suited to its purpose.

The script may include some questions to ask once the evaluation is complete. It is also acceptable at this point to ask the subject about any issues noticed by the experimenter and where the subject may be able to assist by explaining their reasoning. Take care to make it clear that the goal is to improve the product rather than critique the subject.

### *12.3.3 Analyzing the results*

Once the subjects have left the building, meet with the design and development team to review the results. This might include watching the recordings together and identifying any issues that are captured. Developers who are deeply invested in their product might feel the need to criticize the behaviour of the subjects. This is normal. However it is important to direct these conversations towards the aspects of the product that are leading towards the unexpected or undesirable behaviours and to plan strategies that would address these issues.

Identify the issues common to several subjects. These suggest that the issues lies in the experience offered by the application.

Allow the team to identify positive aspects of the product identified by the evaluation. It is important that these be preserved. Elements that have improved since the last round of evaluations should also be noted, as it is likely that the strategies used there have been successful.

Decide on the actions the team will take in response to the insights resulting from the usability evaluation.

# 13

## *Production and Deployment*

### *13.1 Project*

Previous project goals have been phrased in terms of meeting the requirements of your client. It is now worth thinking about the nature of that client in practice. In some cases the client may want an experience built only for their own internal consumption. The client bears the cost of the development and the deployment process is typically well defined in that case. The alternative scenario is that your client is an entrepreneur, possibly even a subject expert that you have partnered with. In this case you still need to earn the money required to pay for your development costs. You need to get your product to market.

The question to be answered: How would you publish your experience to the wider market?

### *13.2 Publishing*

The process of developing enhanced reality applications has been covered in the preceding chapters from the stages of negotiating, ideation and design through development, integration, testing and evaluation of the various components involved. In practice the early prototypes developed tend to occupy only about 10% of the development effort with the remaining 90% being spent

on the finishing touches required to get the product ready for production and deployment.

Many virtual and augmented reality applications, particularly those developed for mobile devices, are deployed through third party online stores. Unlike other software, these are deployed in a context where they must install correctly on customer equipment, and function effectively from the start. The application has to assume responsibility for training the user in using the application effectively. Most stores offer refund options so applications that don't provide immediate value can be easily discarded.

 Component 23.3.1 on page 612 provides a description of the considerations involved in publishing an application to one of the virtual reality market places.

# 14

## *Assignments for Enhanced Reality*

This book is intended to be used in several different ways. When used as a text book, the remainder of this chapter suggests activities that can be undertaken for assessment purposes. Many of these activities include team work that practices skills used in an authentic design, development and production setting.

### *14.1 Understanding requirements*

#### **Exercise 14.1.1: Applying the design component**

*Applied  
Design*

This exercise describes an environment that has been mocked up to facilitate an enhanced reality application design session. This is built using Rec Room, a multi-player virtual reality platform. This facilitates participation in a single virtual location for the design team, and also provides an environment where aspects of a virtual reality application are front of mind throughout the entire process.

Ideally the process should involve discussion with a client. In the absence of a client, one team member can be given a description of the problem context and deputized to play the role of a simulated client. The example of a

typical context, and application goal, provided here is based on an authentic scenario popular amongst those who have worked through this material previously.

*Context:* ‘Providing freedom of movement to autistic individuals offers them greater independence and opportunities to be part of their communities. Public transport is a facility that can be used to meet this goal. Existing educational programs are able to prepare individuals to handle the routine aspects of the process and to cope with the process when everything runs according to schedule. However there are circumstances when this routine breaks down; for simple reasons such as a bus being late, or due to more complex conditions when a familiar location may be closed or changed due to renovations. The goal is to provide an enhanced reality application that will intervene in these situations.’

<sup>1</sup> There is, of course, the requirement of producing an enhanced reality application because of the focus of this document. Ideally this could be relaxed as well when applying the principles covered in the context of a multidisciplinary team.

The goal<sup>1</sup> is deliberately very broadly phrased. Many software development projects insist on specific and clearly defined requirements that must be provided by the project’s client. Constantly innovating fields such as those relating to enhanced reality need to support the clients in exploring what solutions are possible, or risk receiving the specifications for outdated technologies that limit opportunities to achieve the goal. Care must also be taken to avoid implicitly specifying a goal which matches expectations on the form of the solution. It doesn’t ask for yet another GPS directing finding application, with potentially a simplified interface to adapt it to the perceived needs of the target audience. It may not even be satisfied with a mobile application, or an electronic solution. Exploring the classification of the goal is intended to expand perceptions regarding the form of the solution.

1. Create an account on Rec Room. This tool is, as of time of writing, free to use and supported across a range of virtual reality devices. There is also a screen based interface for desktop users although this is not recommended; the design experience is likely to be too

constrained by the interface. Participants in the session should go through the account creation process well in advance of the time scheduled for the shared session as this process can be time consuming. User names can be assigned randomly, but it does help with managing sessions if you and your colleagues can be identified and distinguished based on the information in the handles used.

This exercise requires multiple participants to be effective. As an academic exercise, nominate a number of your peers to play the roles of the various stakeholders such as clients and other interested parties. When undertaking the process in earnest then all relevant stakeholders should be invited to the scheduled session.

Once you've created an account, exchange the details with the other people participating in the session. Rec Room offers the opportunity to send friend requests to others, which then provides an easy way to find your colleagues in the future. However it will take time for the request and acknowledgement process to be completed. It does help to have another communication channel (such as a messaging application) available to manage this process. Have this communication channel available during the session as well, to deal with the inevitable coordination required or issues resulting from technology failures.

2. Once logged into Rec Room you will need to enter the room created for this activity. The room designation is 'oldegrimDesign' and entering part of the name into the room search tool may help find it as the room naming process goes through periodic revisions.

The number of participants in any one room is limited. This is compensated for by creating multiple instances of each room. The downside is that each member of your group may be added to a different room. This is where the prior friend connections are required. One

member of the party can be designated as room owner, and should invite all the other members of the group to their room. Once everyone has managed to enter the same room, then the actual design process can begin.

3. It should be noted that rooms are transient. All changes made to the room will be lost when the last person exits the room. A record of the session can be made in a number of ways:
  - (a) On the mobile Oculus headsets, there is an option for recording a video of the session on the headsets home menu (under the Sharing option). This records audio from all other participants, but does not record the voice of the owner of the headset. Two participants will need to record the video if a full record of the session is required.
  - (b) Rec Room has a virtual camera that can take images of the scene. These should be used to capture images of note boards, or aspects of the scene layout that are required as part of the record of the session.
4. The room is set up with the triangle diagrams for each of the enhanced reality design categories. Some of these are pre-populated with examples of existing designs. Hold down the button to hear a brief description of that particular design. There are also coloured blocks in the scene that can be used to mark the position in the triangle that your team is planning to focus on.
5. Next to each triangle is a noteboard with coloured pens. Use this to make brief notes to remind the group of the points discussed, or of the decision made. As mentioned previously, use your camera to make a record of this note board before you leave the room. If you capture the neighbouring triangle with its headings in the image as well then you will also have a record of the category that those notes applies to.

## 14.2 Design Sprint

### Exercise 14.2.1: Project management using a design sprint.

This exercise involves forming a group with several colleagues, picking one of the project outlines provided, employing the enhanced reality classification to consider innovative design directions, completing several stages of the design sprint to generate a creative solution, and then preparing the initial task board to manage a single sprint that will generate, evaluate and publish a prototype. Actually building the prototype is not yet a requirement.

The purpose of this exercise is to practice processes related to the design of effective virtual and augmented reality applications. The design is not considered in isolation, but with respect to the processes required to implement, evaluate and deploy these applications. The documentation produced during the process represents one way in which such designs can be communicated to team members, clients and other stakeholders.

#### *Instructions*

1. Form groups of 3 or 4 members.
2. Select, or be assigned, one of the outlines below. This outline will provide the goal (the Monday morning stage of the design sprint). Remember that the purpose of the process is to design an enhanced reality application to achieve this goal. Constraints on the process include being able to implement this application using the techniques covered in this document, and the resources and equipment that are available.
3. As a group, complete the Monday afternoon activities of the design sprint. In particular, nominate one of your team members to role play the expert. That person, together with a handy internet search engine, can provide answers. Document the process by generating “How Might We” notes. All team members should keep a

copy of the notes board at the end of the session (take a photograph if necessary). Depending on time available, the design sprint processes may have to be conducted faster than normally recommended.

4. As a group, classify the goals and initial design according to the enhanced reality classification scheme. For each category, each group member should pick a distinctive coordinate and explain how the goal could be achieved with an application matching that coordinate. Suggestions should be well considered but this process is intended to trigger creativity and an element of humour is appropriate. Each individual should keep a record of their coordinates chosen for each category, and a brief description of how that would affect the nature of the experience.
5. As individuals, complete the Tuesday activities for the design sprint. The four-step sketch is the highest priority phase. Each individual should save the results of their work, to be contributed to the group documentation later.
6. As a group, complete the Wednesday activities for the design sprint. In particular, you need to end up with a clear plan describing the nature of your prototype. Each member of the group must keep a record of the final agreed structure.
7. Define the tasks that each member of the group must complete in order to implement, evaluate and publish the prototype. The tasks need to be well defined with a clear statement of what the output will be, and how to tell if the output meets the requirements (in form readily apparent to someone outside the development team).

Tasks that are covered by examples provided in this book include:

- Component 16.1.1 on page 179 creates an application, deploys it to a mobile device and sets up some

basic user interface elements for monitoring and debugging.

- Component 23.2.1 on page 605 builds unit tests to ensure that key components satisfy the requirements placed on them.
- Components 19.1.1 on page 379 and 19.2.1 on page 383 provide basic augmented reality functionality by placing 3D content onto visually tracked markers.
- Component 18.2.1 on page 304 shows how visual and inertial tracking can be used to locate a mobile device as it moves through an environment.
- Component 18.1.1 on page 299 demonstrates how multiple control schemes can be developed for applications that are deployed on different types of platforms. This approach is also useful for developing and testing in a desktop environment, even when the application is intended for a mobile device.
- Component 17.10.1 on page 287 provides a strategy for mixing physical and virtual spaces using portals.
- Component 18.4.1 on page 313 creates controls that can be used to manipulate virtual content using physical interactions.
- Component 17.1.1 on page 226 tracks the user on a global scale using GPS.
- Component 17.2.1 on page 230 access online maps to show content associated with the location.
- Component 21.5.1 on page 549 provides tools for coordinating the location information of multiple participants in a shared environment.
- Component 18.3.1 on page 308 uses sensors located on a mobile device to track the direction it is facing.
- Component 19.3.1 on page 387 turns the mobile device into a control tool for managing interaction in mixed realities.
- Component 20.1.1 on page 475 applies signal processing techniques to utilize an existing sensor to

extract a specific signal. This allows generic sensor platforms, such as mobile devices, to be used to achieve goals specific to a particular problem context.

- Component 17.4.1 on page 246 builds a three-dimensional map of the environment using information collected by a moving, tracked camera.
- Components 19.4.1 on page 400 and 19.5.1 on page 412 make use of gesture recognition to interact with virtual realities.
- Component 18.5.1 on page 319 integrates external libraries with an existing project. It also demonstrates how an artificial intelligence model, trained externally, can be used to gain an understanding of information received from sensors. In this case, the model recognizes particular types of object visible in the camera image.
- Component 16.3.1 on page 199 considers how virtual content, and content captured from the physical world, can be presented to the participants.
- Component 17.5.1 on page 261 uses a depth camera to capture geometry from the physical world for use in a virtual overlay.
- Component 16.4.1 on page 205 presents both content (virtual and captured in the physical world) as 3D content with unlimited levels of realism.
- Components 17.6.1 on page 271 and 17.7.1 on page 278 capture panoramic and 3D views of physical environments.
- Components 17.8.1 on page 279 and 17.9.1 on page 282 generate synthetic panoramas.
- Component 21.4.1 on page 532 provides a way of achieving voice and video communication between participants.
- Component 16.6.1 on page 216 demonstrates a web based solution to creating virtual environments with

no overhead for installation of the application, and which is readily available on multiple platforms and devices.

- Component 16.7.1 on page 221 extends the web centered philosophy to sensing the physical environment and overlaying virtual content.
  - Component 21.6.1 on page 570 shares location markers captured by one participants with others to support shared augmented reality experiences.
  - Component 21.7.1 on page 583 investigates strategies for mediating communications between participants.
  - Component 16.5.1 on page 211 uses spatial sound as a modality for augmenting environments.
  - Component 19.8.1 on page 458 exploits speech as a facility to interact with virtual reality applications.
  - Component 20.2.1 on page 482 demonstrates how a signal received in one modality can be translated and output in another, allowing augmentation of human senses.
8. As a group prepare a single document that explains the proposed solution. This should use the template below which collects the output from each of the preceding steps:
- (a) Front matter (a title, group members, outline of the goal).
  - (b) Initial opportunities. This should have a subsection for each individual where they present their own contribution to the process, and their personal record of the notes board. Should the group desire, they can collectively provide a summary at the top of this section.
  - (c) Classification. This should have a subsection for each individual where they present their classification suggestions. The group consensus on the classification actually used should be presented separately at the top of this section.

- (d) Individual ideas. This should have a subsection for each individual where they present their sketches.
- (e) Nature of the prototype. This is a collective effort from the entire group. Explain the nature of the planned prototype as clearly as possible. Suggested strategies are to define the features or components of the prototype. Alternatively describe how the stakeholders achieve their goal by providing several short “a day in the life” paragraphs describing the experience of someone using the prototype.
- (f) The task board, also a collective effort. Each task should outline:
  - i. What the task is.
  - ii. What the output will be.
  - iii. How to tell if the output is what is desired.
  - iv. Who (possibly several people) assigned to work on this task.
  - v. Any pre-requisites (other tasks that must be completed before this can start).
  - vi. How the output will be incorporated into the project (as a pre-requisite for another task).
  - vii. When the task will start and end (position in the timeline).

### 14.3 Component Innovation

**Exercise 14.3.1: Innovate.** *The purpose of this exercise is to apply concepts covered in this chapter, and to practice the skills demonstrated in the examples. This supports your ability to implement components of enhanced reality applications but also contributes to the experience you need to design effective experiences.*

*Achievement can be demonstrated at different levels. Basic learning is achieved by following the instructions and replicating the results. Higher forms are demonstrated by utilizing the principles involved to adapt the process to solve related but distinct problems.*

*Instructions*

1. Follow the instructions for one of the examples in this chapter. Once you understand the process (discuss with your teacher to clarify anything that may be confusing), define a particular context in which this technique can be of use. Repeat the process again, this time adapting the steps to build a prototype that would be appropriate for your particular context. The goal is to demonstrate your mastery of the technique which is best done by adapting or extending the process. Substitution of content is less likely to achieve this goal.

#### *14.4 Component Customization*

**Exercise 14.4.1: Contribute.** *This exercise is a continuation of an earlier task involving the development of a more significant enhanced reality application, and its associated project management and cooperative work behaviours. This allows you to practice processes associated with implementing and deploying such applications, but also involves sharing and communicating your efforts with your team members.*

*Instructions*

1. Complete a task on your group development project that is associated with the topics covered in this chapter. Once you've done this, commit your changes to the version control system, and update the record on the task tracking system (update the status of the task, and record details of what you've done). Update your local copy of the project from the version control system and make sure your changes integrate with any updates contributed by other members of your team.

#### *14.5 User Experience Evaluation*

**Exercise 14.5.1: User experience evaluation.** *Ideally you should be evaluating the user experience of your application*

*on an ongoing basis, right from the design stage (for example, as part of the validation of the prototype on the Friday of the design sprint). This exercise is an opportunity to work through key steps of this process with a relatively mature version of your prototype.*

*This provides an opportunity to refine your design based on feedback from representatives of your particular target audience. Feedback received from actually observing use of your application often provides great insights which, after reflection, can enhance the way you approach design and development of such systems in the future.*

### *Instructions*

1. For this exercise you will need:
  - (a) A written script defining what you will tell your participant, what you will ask them to do, and any questions you would like answered after they have completed the process.
  - (b) A willing participant. Ideally this should be a representative of your target audience, but someone who has not used (or discussed) the application before may be an adequate substitute. Other members of your team are not suitable.
  - (c) A workable, up-to-date version of your application. In early stages of development, some portions of the application may be substituted for paper prototypes, or a wizard-of-oz technique where a human actor will simulate some steps.
  - (d) Recording instruments (a video camera on a tripod is helpful), and permission from your participant to record them, and to use the recording.
2. Perform the user experience evaluation.
3. Report on the process. If the test went completely smoothly and the participant did everything expected then either you are a god of enhanced reality application development, or your script was too specific

(perhaps telling the subject how to do tasks, rather than what tasks to do). If the latter then start again.

Report on issues identified during the evaluation. This should be communicated in a form that will provide enlightenment to the remainder of your team. You should explain:

- (a) What the task was.
- (b) Outline the steps that you expected the subject to follow.
- (c) Show the recording, presenting what the subject actually did.
- (d) Explain why the subject's behaviour diverged from that expected. Points noted from the subject's talk aloud feedback are appropriate here, or any discussion with the participant during an end-of-test interview.
- (e) Suggest modifications that might resolve the situation. This might involve working on modifying behaviour of users so that they conform to your expectations, but opportunities to modify the application to support the desires of its users can also be considered.

## 14.6 Pitch

**Exercise 14.6.1: Pitch.** Your brief, should you choose to accept it, is to develop a pitch for your team based on the product design and development experience that you've gained from working through this material. Note that this is not aimed at just showing off your product (you get to do that in a separate exercise) but rather on presenting each member of the team as a potential candidate for employment, either individually or as part of a package set.

This exercise provides the opportunity to communicate your knowledge and skills related to the development of enhanced reality applications to potential clients and employers.

*Instructions*

1. As a team, prepare a 5 minute pitch that shows off the knowledge and skills gained and demonstrated in enhanced reality application development. Evidence must be provided using the product that the team has been working on.

The time limit is strict. Saying something efficiently and succinctly is harder than waffling on and on. This skill is developed through preparation and practice, so write down what you want to communicate, practice saying it, and trim as necessary to leave only relevant details. The pitch can be presented as a pre-recorded video, allowing opportunities to practice the presentation, edit it to maximize impact, and to include any additional media that support the pitch.

Each team member must be visibly present in the presentation.

#### 14.7 Demonstration

**Exercise 14.7.1: Demonstration.** *This demonstration provides the opportunity to show off the final outcome of your team development project. Consider the lessons around providing an effective demonstration and aim to make the most impact.*

*This is another opportunity to communicate to clients and potential customers. Ideally the demonstration also shows what you've learned with respect to designing effective enhanced reality applications, customized to your particular target audience, and your ability to implement a working product.*

*Instructions*

1. The demonstration should last about 10 minutes and involve the participation of each team member. It may be presented live, taking opportunity of audience participation, or pre-recorded.

You are strongly advised to avoid the common pitfall of working through each feature of the application in

tedious detail. Some operations, such as creating an account, are only interesting if you truly manage to stuff them up. Rather think about the audience and the journey they would have with the application under a typical scenario. Walk them through the narrative, highlighting the ways in which the application anticipates and addresses their needs. Show how your product makes life easier.

The demonstration must show the actual application in action.



# **Component Categories**



# 15

## *Design Components*

### **Contents**

---

15.1	 A systematic approach to designing an enhanced reality application . . . .	169
------	---	-----

---

15.1	 A systematic approach to designing an enhanced reality application
------	---

**Component 15.1.1: Designing an enhanced reality application.** *The worksheets on the following pages can be used to manage the design process. They are formatted for use as a series of cards that can be shared during discussions, for stakeholders to use as a reference, or to scribble on and annotate in order to capture ideas.*

*The intention during this phase is to explore all corners of the design space to suggest possibilities of what might be done, and to explicitly reject options that are not within scope. Start with the outline of the steps to follow, using your judgement to adapt the process to best suit the context of the application being designed, and the needs and characteristics of each person participating in this activity.*

Guiding principles and steps to follow:

1. Decide what the goal is: “An enhanced reality application to ...”.
2. The definitions of each category are not definitive.
  - (a) Utilize the ambiguity to debate the options.
  - (b) Explain the category.
  - (c) It also helps to give examples of how different regions of the space could apply to the given goal. Be careful to avoid influencing the design process unduly. Either use some of the examples from previous systems, or choose some potentially extreme and outlandish ideas relevant to the current goal.
3. Classify according to each category.
  - (a) Consider different regions and what that would translate to in terms of achieving the goal.
    - i. Generate a sticky note with this scenario outlined.
    - ii. Don’t worry about feasibility at this stage. This can be managed through later stages of a design sprint (see exercise 14.2.1).
    - iii. There is no need to force compliance with the category definition. Creative ideas relevant to other categories should be recorded. However, do encourage consideration of empty regions and prompt discussion around options in this area.
  - (b) Is the midpoint what is required (often: no)?
4. This process is one strategy for performing ideation activities as part of a design process, such as a design sprint. The sticky notes produced feed into that design process. Make use of the output of this activity in a way that is consistent with the complete design process.

Categories to be considered:

1. Setting

(a) Nature of reality

The form of experience that can be used to achieve the goal, or “Not everything has to be an app”.

(b) Location (setting)

The environment or setting in which the experience takes place.

2. Interactivity

(a) Objects

The form taken by the elements in the settings that the participants interact with, or through which interaction is achieved.

(b) Interaction and Feedback

The nature of communication from the participant to the experience, and from the experience to the participant.

(c) Concepts explored

The nature of the content that is used to achieve the goal.

3. User experience

(a) Participant association

The way in which participants in the experience interact with one another.

(b) Design of experience

The activities that are part of the experience.

4. Equipment

The hardware and software elements. This is deliberately left to last to avoid constraining design ideas by technology constraints.

Definitions for each category:

*Nature of reality:* (Physical-Virtual-Abstract)

*Physical:* physical reality with physical embodiment.

*Virtual:* synthetic content, usually with a well established representation.

*Abstract:* does not have an established form (mathematics, feelings and emotions).

*Location (setting):* (Physical-Virtual-Abstract)

*Physical:* make use of the physical world to provide the setting.

*Virtual:* locations involve synthetic environments that replace the physical world.

*Abstract:* the extent to which the setting would be achievable without a technology intervention.

*Objects:* (Physical-Virtual-Abstract)

*Physical:* exist in the physical world but can be tracked or sensed.

*Virtual:* synthetic elements generated, manipulated and presented via an overlay.

*Abstract:* lack an explicitly presentable form, represented indirectly.

*Interaction and Feedback:* (Physical-Virtual-Abstract)

*Physical:* actions and responses are based on a physical mechanism.

*Virtual:* involves information transmission and display of response.

*Abstract:* involve sensing mood, and responding by invoking an imaginative or emotional response.

Definitions for each category (continued):

*Concepts explored:* (Physical-Virtual-Abstract)

*Physical:* involve operating a particular piece of equipment, accurate simulation of device.

*Virtual:* spatial data not readily available in the physical world but can be presented relative to the location.

*Abstract:* no clear spatial component.

*Participant engagement:* (Copresence, Physical Presence, Telepresence)

*Copresence:* each participant engages with the experience.

*Physical Presence:* communication when all participants are in the same physical location, or when participants are directly aware of the actions of others.

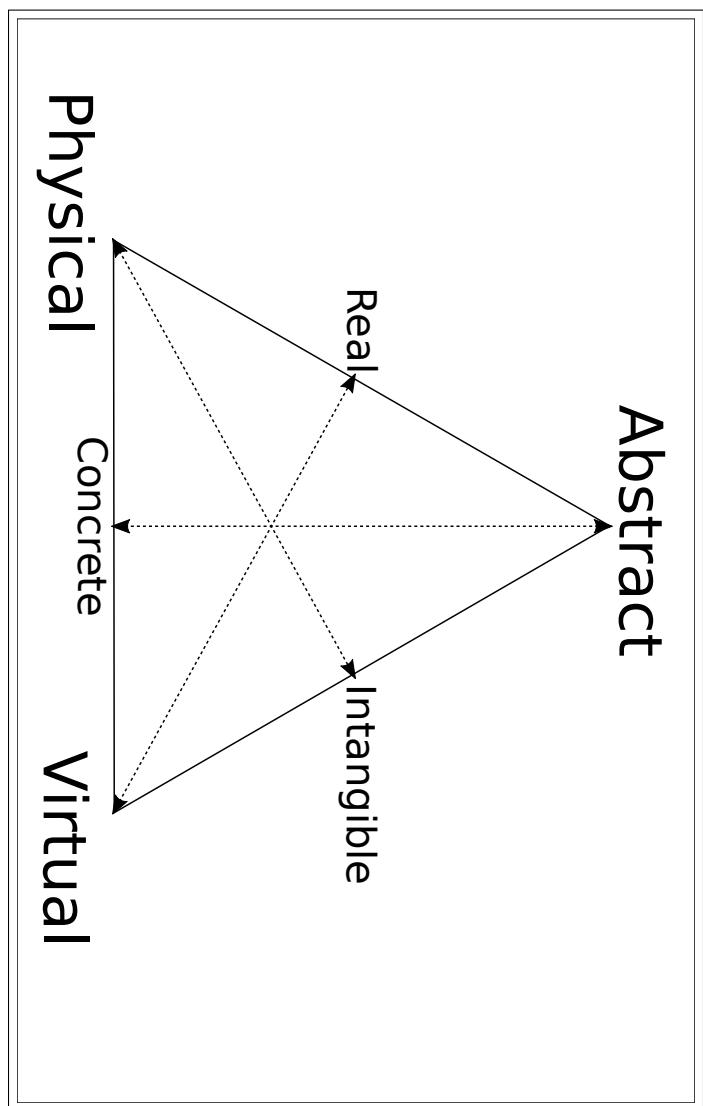
*Telepresence:* communication between participants is mediated by the application.

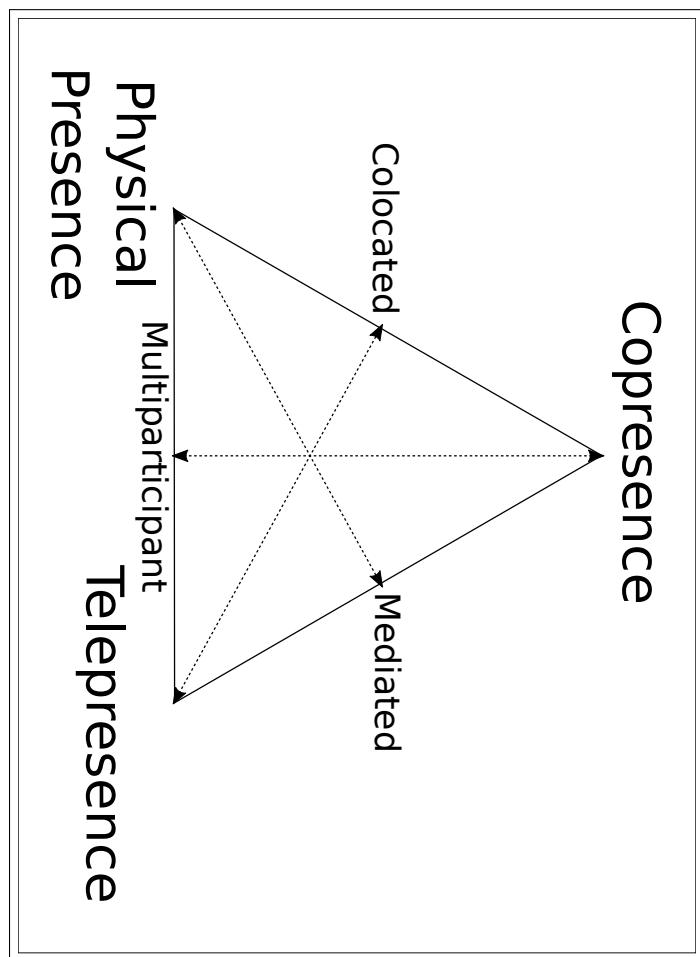
*Design of experience:* (Performance, Narrative, Ludology)

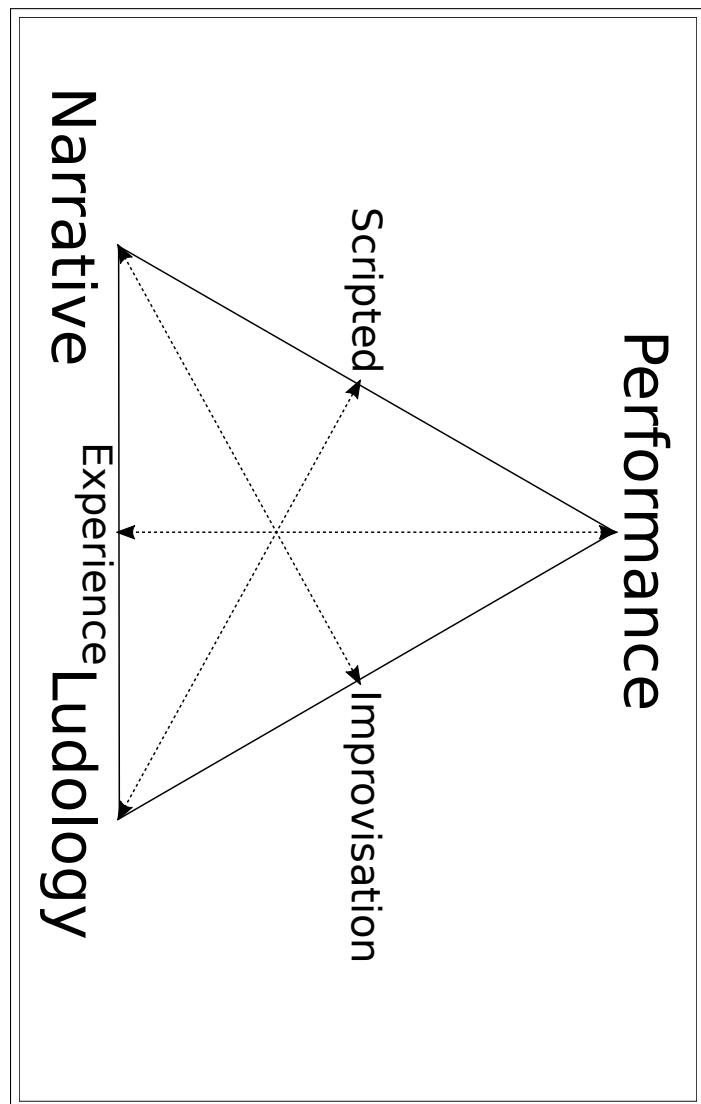
*Narrative:* predefined story, or set of story arcs, that guide the user through a series of stages.

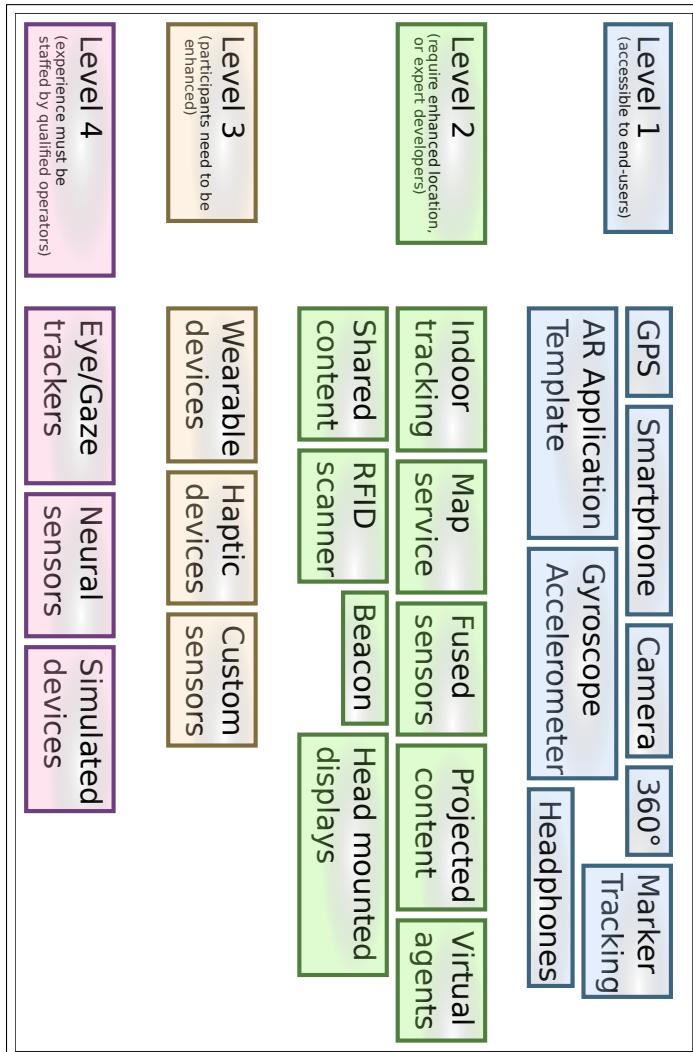
*Performance:* users role play, or otherwise use an imaginative overlay.

*Ludology:* game rules and goals govern player behaviour.











# 16

## *Nature of Reality Components*

### Contents

---

16.1	Creating an enhanced reality experience on a mobile platform . . . . .	179
16.2	Setting up a virtual reality experience on an Oculus Quest mobile headset .	192
16.3	Deploying an enhanced reality experience on a Daydream . . . . .	199
16.4	Presenting 360° panoramic content on a headset . . . . .	205
16.5	Presenting experiences using audio rendering . . . . .	211
16.6	Providing virtual reality experiences using web technologies . . . . .	216
16.7	Presenting augmented reality experiences using web technologies . . . . .	221

---

- 16.1 Creating an enhanced reality experience on a mobile platform

Component 16.1.1: Creating a mobile application with

Unity software.

Mobile  
Deployment

The project files associated with this example are available in the associated project archive, in the directory indicated in the

box above.

*The examples and exercises that follow make use of Unity software to create individual applications that are deployed to a mobile platform. These platforms include generic smartphones as readily available augmented reality platforms but also platforms that incorporate additional hardware, such as head mounted displays, to provide mobile virtual and augmented reality facilities in more convenient form factors.*

*This example runs through the process of creating and deploying such an application. This procedure will then be assumed for all subsequent examples. The specific application created will be rudimentary since it is intended as a skeleton upon which more advanced virtual and augmented reality technologies can be added in later examples. It does demonstrate a number of facilities which should not be included in any production applications, but are shown because they support the development process. These include touchscreen based user interface controls and content which are often not the most appropriate user interface strategy for an enhanced reality application. Access to particular sensor information and presentation of 3D content will also be managed by specialized components in later examples.*

*The instructions provided assume the presence of an Android device. Physical devices are most useful although some progress can be made with emulators providing virtual devices. Some components developed do not work on emulators because of differences in the computing architecture, or may work perfectly (when the real device fails because of sensor limitations when working in the physical world).*

1. Start up the Unity software and create a new project. The default project settings are usually adequate, but make sure that the 3D option is set when creating the project. The resulting interface appears similar to that in Figure 16.1.2.
2. Content can be added to the scene via the Create option under the Hierarchy tab (usually on the top, left hand side of the window). This scene is to be rudi-

Figure 16.1.1:  
Creating and  
deploying a mobile  
application.

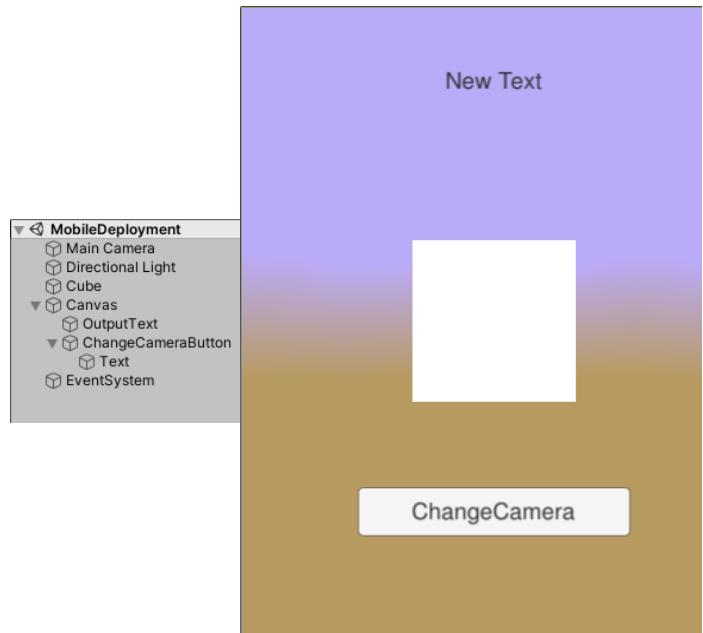
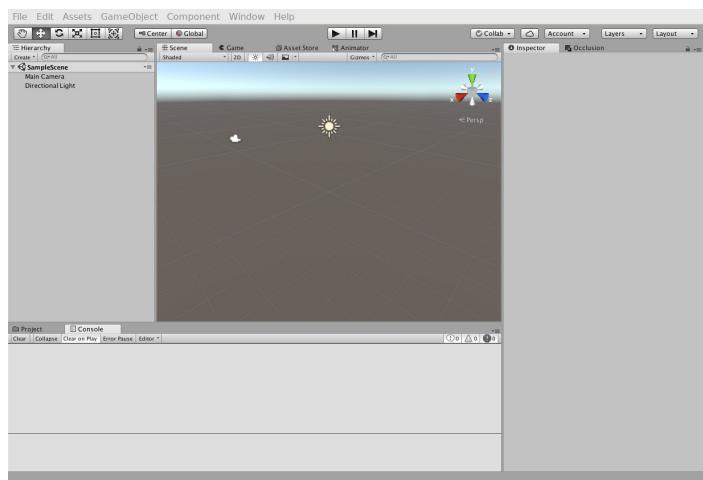


Figure 16.1.2: Unity  
software interface  
for a new project.



mentary, so select the create option and add a Cube (under the 3D Object category in the drop down list that appears).

Clicking on the Cube causes its properties to show in the Inspector pane (usually on the right hand side of the window). Adding some new content to each project in this way helps provide some visual reference when testing the process of initially deploying the application to a particular device. Making the content unique and distinctive helps prevents issues where deployment appear to succeed but actually fails silently because another application with the same identifier already exists on your test devices.

3. The sample scene currently consisting of a camera (named Main Camera), a light source (named Directional Light) and the newly created Cube. Clicking on each of these results in their properties displayed in the Inspector pane. The view of the scene, as registered by the Main Camera, is available by selecting the Game tab for the central window.

By default the Cube is positioned at the origin (0, 0, 0) and the Main Camera is situated 10 units backwards (at (0, 0, -10) where the convention is usually that the x-axis runs off to the right, the y-axis represents up, and the z-depth is forward depth). Since many enhanced reality applications are focused on the device as both a physical camera (using the camera hardware on the device), and as a software rendering camera (producing the view of the virtual 3D scene as is done by Main Camera) it is convenient to treat viewpoints as placed at or near the origin (with perhaps a vertical displacement relative to a ground plane or body, or a horizontal displacement to represent an offset for eyes relative to the center of the head).

Set the position of the Cube to (0, 0, 4) and the position of the Main Camera to (0, 0, 0).

4. This step outlines the primary requirements for de-

ploying to a mobile device. These need to be repeated for any new project, but generally only have to be set up once.

Use the File/Build Settings menu option to open the build settings window.

- (a) Select Android as the platform.
- (b) Access the player settings (often via a button in the Build Settings window, or under the Edit/Project Settings menu option). The player settings appear in the Inspector tab. Make the following changes:
  - i. Under Other Settings/Identification heading, set the package name. This is a globally unique identifier for your application so choose something unique to you and your application. Typically in the form of three words separated by full stops such as xr.company.product.
  - ii. The minimum API level will depend on what other components you are including in your application. Advanced components may require more recent versions of Android software libraries. Typically set this to a recent version that is still supported by the device you are using.
- (c) Under the build settings, the Run Device can usually be left at the default value which will use any device found. If you have multiple devices attached, you can also specify which one to use.
- (d) Select the Build and Run button in the build settings window. You may be prompted for:
  - i. The name of the file to contain the application. The default directory (part of the project you created) is fine so don't change this. Enter a suitable name.
  - ii. The location of various Android development libraries. Make sure these are installed (this is typically done by installing Android Studio <https://developer.android.com/studio/install>, and

then using this to download the Android SDK and other requirements). Provide the path to the folders requested.

- iii. Permission to download applications to the device. This involves settings and prompts on the actual Android device or phone itself. Make sure developer settings are enabled (<https://developer.android.com/studio/debug/dev-options>). In the settings (only visible once developer mode has been enabled) enable USB debugging. When Unity software first tries to download an application to the device it will pop up a request for authorization. Accept this quickly before it disappears and you have to rerun the build process to trigger it again.
  - (e) The build process may be slow on the first build as it packages all the pieces it requires but should eventually report Success in the Console tab. Any errors encountered will also be reported in the Console tab. Errors may need to be resolved by consulting online sources but typically result from missing or inappropriate versions of supporting software. The Android build process requires that the Unity software interact with a significant number of external packages which can require some effort to correctly configure.
  - (f) The application may start itself automatically, but should also be available under the device's list of applications with a Unity icon, and under the name of the project. Try it out - it should show a static view of a scene, with the cube visible.
5. Many more settings will have to be modified before the application can be released for publication in one of the online stores. Details of these requirements are available in later examples, such as Component 23.3.1.
6. The next step is to connect the camera feed so that it displays on the surface of the cube. This allows some

augmentation of the virtual setting with physical content. This unlike most other examples where we augmented the view of the physical setting with virtual content.

The Project tab (usually at the bottom of the window) contains the Assets folder with all the elements that are part of the project (but may not be in the current scene). There are conventions for managing this folder to simplify interchange of projects with other professionals. The first of these is a folder naming convention: different classes of asset are stored in appropriate folders. These folders will have names such as: Prefabs, Scripts, Materials, Scenes, Shaders, and Resources.

Create a folder (right-click on the assets area, select Create/Folder from the resulting menu). Rename this folder: Materials. Inside the Materials folder create a new material (Create/Material) and rename this as PhysicalCameraViewMaterial.

Create a folder in the assets area called Scripts. Inside the Scripts folder, create a new C# script (Create/C# Script) and name this PhysicalCameraTexture. Double click on the PhysicalCameraTexture script to open it in a text editor. Copy the following script into the text editor to replace what is there already.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PhysicalCameraTexture : MonoBehaviour {
6
7      public Material camTexMaterial;
8
9      private WebCamTexture webcamTexture;
10
11     void Start () {
12         webcamTexture = new WebCamTexture ();
13         camTexMaterial.mainTexture = webcamTexture;
14         webcamTexture.Play ();
15     }
16 }
```

---

This script sets the image from a physical camera at-

tached to the device as the texture for a material passed in as the variable `camTexMaterial`.

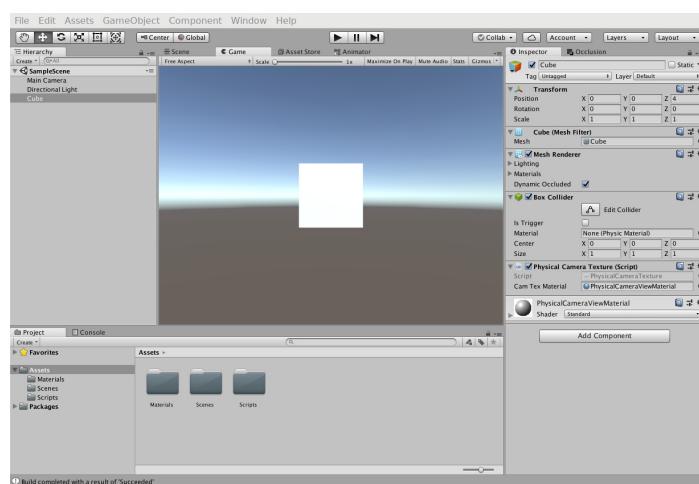
7. Add the newly created material to the Cube (drag it over and drop it on the Cube).

Add the newly created script to the Cube (also drag and drop).

Click on the Cube, so its properties are visible in the Inspector pane.

Drag the `PhysicalCameraViewMaterial` onto the Cam Tex Material field of the Physical Camera Texture (Script) property. The result should appear as shown in Figure 16.1.3.

Figure 16.1.3: The cube contains both a new material, an additional script component and the material is set as the target of the camera image in the Cam Tex Material property.



You should now be able to run the application (on the development machine if you have a camera attached, or via the build process performed earlier on the Android device). The image from the camera should appear (potentially upside down) on the cube. Delays in initializing the camera can prevent the image from appearing on some occasions. An enhancement would involve checking if the camera was playing in an `Update` function, and starting it if it was not.

8. The 3D nature of the cube is not particularly noticeable with the static, front facing view. Add in an additional script called RotateObject to the Scripts folder, and provide the following code:

Code Listing 16.1: RotateObject.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class RotateObject : MonoBehaviour {
6      void Update () {
7          transform.rotation *= Quaternion.AngleAxis
8              (0.5f, new Vector3 (0.3f, -0.5f, 0.1f));
9      }
}

```

Add this script to the Cube object as well so that the cube spins when the application runs.

9. The next step is to build a rudimentary user interface to interact with the application. This is primarily presented for debugging purposes. Applications deployed on mobile devices and that are sensitive to their environment cannot always be easily tested on a desktop development environment or through simulators and emulators. Quick user interfaces allow status information to be displayed to the developers during testing and simple controls allow configurations to be changed without having to rebuild and redeploy the application.

In this case, the example creates one control that displays internal status of the application, and another that allows a configuration to be changed. The application currently defaults to one particular camera on the device, and the interface provided displays the list of available cameras, and allows the user to cycle through them by pressing a button.

10. From the Hierarchy tab, create a UI/Text element. Rename this to OutputText. The first time a UI element is created, various supporting elements (a Canvas and

an EventSystem) are also added to the scene. The text element has various properties. Set the Pos Y to 100, the Width to 200 and the Height to 100 so that the text does not overlap with other content on the screen, and so that there is sufficient room to show several lines of text. Set the Alignment controls to center the text as well.

Create a UI/Button element. Rename this to ChangeCameraButton. Move this out of the center by setting Pos Y to -100. Expand the Button in the scene hierarchy to show the Text element that is part of it. Change the Text field in the Inspector to contain the words: “Change Camera”.

11. Edit the PhysicalCameraTexture script and add in the functions shown which show the available devices by writing them to a text object, and that also keep track of current device and change to the next.

Code Listing 16.2: PhysicalCameraTexture.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class PhysicalCameraTexture : MonoBehaviour {
7
8      public Material camTexMaterial;
9
10     public Text outputText;
11
12     private WebCamTexture webcamTexture;
13
14     private int currentCamera = 0;
15
16     private void showCameras ()
17     {
18         outputText.text = "";
19         foreach (WebCamDevice d in WebCamTexture.devices)
20         {
21             outputText.text += d.name + (d.name ==
22             webcamTexture.deviceName ? "*" : "") + "\n";
23         }
24
25     public void nextCamera ()
```

```

26     {
27         currentCamera = (currentCamera + 1) %
28             WebCamTexture.devices.Length;
29             // Change camera only works if the camera is
30             stopped.
31             webcamTexture.Stop ();
32             webcamTexture.deviceName =
33             WebCamTexture.devices[currentCamera].name;
34             webcamTexture.Play ();
35             showCameras ();
36     }
37
38     void Start () {
39         webcamTexture = new WebCamTexture ();
40
41         showCameras ();
42
43         camTexMaterial.mainTexture = webcamTexture;
44         webcamTexture.Play ();
45     }
46 }
```

12. Once the script is accepted by the Unity software (no error messages in the Console window), drag the OutputText object in the scene into the Output Text field of the Physical Camera Texture script on the Cube. The list of available cameras should now be shown in the OutputText object when the application is run.
13. Connect the ChangeCameraButton to the function provided to change the camera by following these steps:
  - (a) Select the ChangeCameraButton so its properties appear in the Inspector window. Press the + button under the On Click () list to add an extra event handler when the button is clicked.
  - (b) Press the circle next to the None (Object) to select the object that contains the component with the function we require. In this case, that is the Cube object from the Scene tab (not from the Assets tab).
  - (c) Select from the “No Function” drop down list to choose the nextCamera function on the Physical-CameraTexture component.

Now when the button is clicked, the nextCamera function is called.

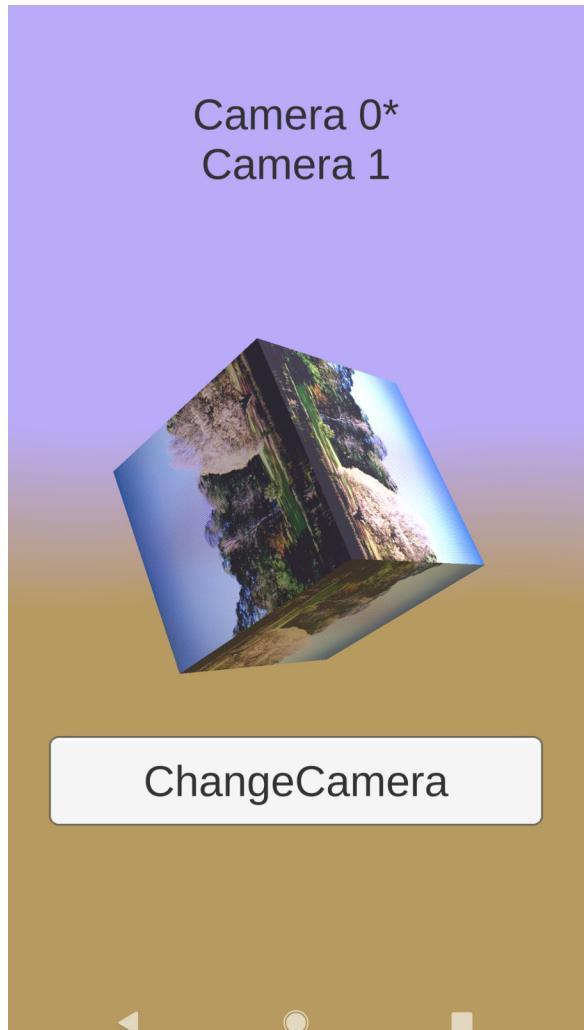
14. Testing on the device should show the list of cameras available, and allow a press of the button to change cameras.

This might be awkward on some devices because the interface scales to minuscule size on some devices. Ideally the interface should remain a consistent size and fit into the available screen space. The following steps suggest some principles that can be used for consistent layout.

- (a) The canvas element in the scene has a Canvas Scaler component accessible through the Inspector tab. Setting the UI scale mode to “Constant Physical Size” helps retain the scale under different devices.
- (b) The game view has a resolution option at the top of the window (typically set to the default value of “Free Aspect”). The effect of different screen sizes can be tested by changing this setting without needing to build for a defined platform.
- (c) The individual canvas elements (the text output or the button) can be anchored to edges of the screen. The anchor can be chosen from the options provided in the Rect Transform component of the UI element. Select the drop down menu on the layout diagram icon and select the appropriate anchor point. The OutputText can be anchored to the top-center, for example, and the ChangeCameraButton can be anchored to the bottom-center.

Deploy your application and make sure it works as desired. Remember to test the effect of changing from portrait to landscape orientation.

Figure 16.1.4:  
Mobile application  
deployed to device.



## 16.2 Setting up a virtual reality experience on an Oculus Quest mobile headset

The process described in the following example is intended for deploying applications to the Oculus Quest head mounted display and controllers. The process is conceptually similar for other mobile headsets, such as the Oculus Go or the (discontinued) Google Daydream, although specific details will differ.

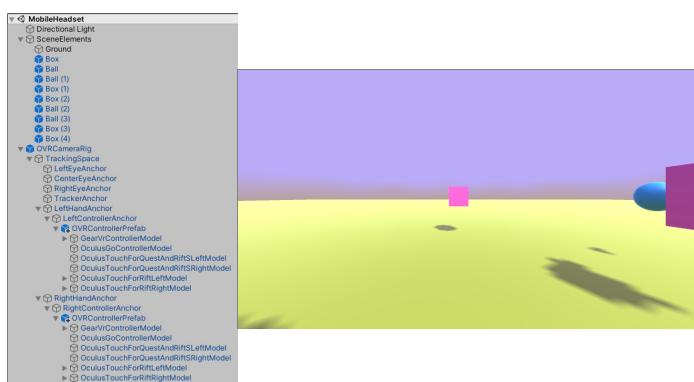
### Component 16.2.1: Building applications for mobile

#### virtual reality headsets.

Mobile  
Headset

*This example demonstrates the foundations of an application built for the Oculus Quest headset. This minimal functionality provides a scene, within which the participant is immersed. Use of the controllers is also demonstrated, to provide the ability to respond to the individual controls on the controllers. A location motion technique is also included although this is intended to support project development rather than be used as part of a production environment.*

Figure 16.2.1: Building applications for a mobile headset.



1. Start a new project using the Unity software. The first step involves building a scene representing the location (setting) of the application. In this case, some facilities of the Unity software are demonstrated which may be helpful to those without much prior exposure to the system.

The project tab, usually at the bottom of the window, contains the assets used by the project. By default this is nearly empty and likely contains only the asset representing the default initial scene. Projects should ideally be structured into a set of folders to organize the various assets used in the project. One common convention uses folders for: Scenes, Scripts, Prefabs and Materials, and any additional classes of asset. Some folder names have special significance to the Unity software and are required when using certain types of asset. The Resources folder often contains textures and models that may have to be accessed while the application is running, and the StreamingAssets folder is useful for files that may have been packaged for deployment to a mobile device.

As a starting point, ensure you have a Scenes folder, and that the current scene is in the folder. A scene by default contains a camera (representing the point from which the scene is viewed), and a light source. Typically the camera is positioned so that it has a view of the origin, where new objects are most likely to be created. The y-axis represents the up direction, while the z-axis is typically depth in the initial orientation. Most projects replace the camera with an augmented or virtual reality version representing the position of a smart phone, or a headset.

2. The Hierarchy pane within the Unity window shows the structure of the current scene. We also use this to add new objects to the scene. Use the Create button to add a 3D Object/Plane to the scene. This can be used to create an initial ground plane, which is an important visual reference in a virtual environment.

The Inspector pane within the Unity window displays the various properties of the object, related to any components that have been attached to the object. Make sure the plane is highlighted, and set its Transform/Scale property to 10, 1, 10 to make the ground large

enough to walk on.

Select the Plane and rename it (right click/Rename, or F2) to 'Ground'. Meaningful names for scene elements help you and your colleagues to understand the scene structure.

3. In the assets region (Project tab), create a new folder for Materials, and within that folder create a new material (right click, Create/Folder, and then right click-/Create/Material). The material will be used to colour the ground. Remember to name the folder and material appropriately (right click/Rename, or F2). With the material selected, use the Albedo box in the Inspector pane (white box next to the eyedropper icon) to select a distinctive colour for this.

Drag the material from the assets region and drop it onto the Ground object in the Hierarchy pane, to assign that component to the ground object. The view in the Game tab should change to show the colour change.

4. Use the create button in the Hierarchy pane to add some other objects to the scene, such as cubes and spheres. Create and assign materials to each of these.

For the cube, use the Add Component button at the bottom of the Inspector pane to add a Physics/Rigid-body component. This makes the object respond to physics based interactions, including a local gravity field if this is enabled.

Creating several of each of these can be tedious. It is easier to turn each into a template that can be reused (instantiated) multiple times. Create a Prefabs folder in the assets region, and drag one each of the cube and sphere objects into this. These are now prefabs (templates) that can be dragged back into the scene multiple times.

Change the position and rotation properties of the transform component (Inspector pane) for each object so they are situated at different places around the

scene. Try to make the scene distinctive so that a participant wearing the headset can see landmarks to get their bearing.

5. Save your project (save both the scene and the project, under the File menu). Test the application by pressing the play button in the top center of the window. The button turns blue while playing. Any changes made to the scene while playing are lost when the application stops (press the play button a second time). These temporary changes are useful for tweaking settings, but as a rule, only make changes when the application is stopped.

Ideally you should see movement in any objects subject to physics interaction once this project runs.

6. The project now needs to be configured to export to the Android platform used by the Oculus Quest. Open up the build settings using the File/Build Settings menu option. Select the Android option from the Platform list. If it is not present then you may still need to install this part of the Unity software. Confirm the change using the Switch Platform button.

Some further configuration settings are still required. These can be accessed via the Edit/Project Settings/-Player menu option, but are also conveniently accessible via the Player Settings button on the Build Settings window.

Under Other Settings/Package Name, provide a unique identifier for your project. This is typically of the form of three words, separated by dots. The Target API level may also need to be set if any warnings are received during later build stages.

Attach an android device (any will do at this stage), usually via a USB cable. Make sure developer mode is enabled, that USB debugging is enabled under the developer settings, and that you grant permission for your host device to connect to the android device when

the prompt pops up on the android device's screen. The exact mechanisms for this differ according to the device so you may need to look up how to enable developer mode, and where the developer settings are on the settings menu for your device.

For the Oculus Quest, developer mode is enabled through the host phone that you used to setup your Quest in the first place. You may be asked to create an organization while doing this. If so, log into your Oculus account, and follow the prompts to create an organization. The permissions dialog pops up in the headset while it is connected to the host computer (often when trying to deploy an application for the first time) and will need to be accepted by putting the headset on and granting permission.

If you have multiple devices connected to the host, select the appropriate one as the Run Device on the Build Settings. Press the Build and Run button to create the Android package and deploy it to the device. Verify that it runs as expected on that device.

7. This next step configures the project specifically for use with the Oculus Quest. Under the Project Settings, find the section that is part of the Player/XR Settings. Check the box labelled Virtual Reality Supported. In the list of Virtual Reality SDKs add the Oculus support. Check the VR Signing box to support the Quest.

Use the Unity Asset Store to find and download the Oculus Integration package.

Some settings may have to be altered:

- Under Project Settings, then Player/Other Settings, remove Vulkan from the list of Graphics APIs as it is not supported as at the time of writing.
- Also under Project Settings, then Player/Other Settings, set the Minimum API level to a value such as 25, corresponding to Android 7.1 Nougat.

The project should be able to deploy to the Quest headset in this state. A first person view is possible in the current state, but no interaction (or even representation of the controllers) is available.

8. Remove the Main Camera from the scene (select and delete it). From the assets region, find the Oculus/VR/Prefabs folder and add the OVRCameraRig to the scene. This will probably be created at the origin, but it is worth setting the y coordinate of the position to 1 just to get a familiar view in the Unity editor.

Unfold the components of the OVRCameraRig in the scene (by clicking on the arrow next to it). This shows the child components, such as the TrackingSpace. Keep unfolding until the Left and Right-ControllerAnchors are visible. These are reference objects that follow the location of the physical controllers.

Drop an OVRControllerPrefab onto each of them. This provides the 3D model that shows the shape of the controller. You may also want to configure each of these to correspond to the left and right controllers respectively.

Deploy the project in this state to have working head and hand tracking.

9. To get some functionality from the controllers we will add a script that moves the camera rig in the direction the controller is pointing while the trigger is pressed.

Create a Scripts folder in the assets area. Right click and select: Create/C# Script. Name this script: MoveInDirectionOf. The spelling and capitalization is important, as the program code we write needs to match this. Double click on the script to open it up in an editor, and replace the code with that given in Algorithm 16.3.

Code Listing 16.3: MoveInDirectionOf. The Oculus controller is used to move the object this script component is attached to.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class MoveInDirectionOf : MonoBehaviour
6  {
7      [Tooltip ("Controller input used to trigger movement")]
8      public OVRInput.RawButton button =
9          OVRInput.RawButton.LThumbstickUp;
10
11     [Tooltip ("Speed of movement.")]
12     public float speed = 10.0f;
13
14     [Tooltip ("Object used to determine direction of
15         travel")]
16     public GameObject pointer;
17
18     [Tooltip ("Constraint vector used to filter axes of
19         movement. The normal to the plane that movement is
20         projected to.")]
21     public Vector3 constrainDirection = new Vector3 (0, 1,
22         0);
23
24     void Update()
25     {
26         if (OVRInput.Get (button))
27         {
28             this.transform.position += speed *
29                 Vector3.ProjectOnPlane (pointer.transform.forward,
30                 constrainDirection);
31         }
32     }
33 }
```

Add two copies of this script to the OVRCameraRig (drag and drop it onto the OVRCameraRig in the Hierarchy pane twice). Set the button field to the L Index Trigger and R Index Trigger respectively. Set the pointer field to be the LeftControllerAnchor and RightControllerAnchor respectively.

Deploy the application. Pulling the trigger on either controller should move the entire camera rig in the direction the controller is pointing. The script is set by default to leave the height constant, so it should only allow motion across the ground place.

10. As a refinement you may want to change the speed

property to make the movement faster or slower.

Also adding a box collider component (Physics/Box-Collider) to the OVRCControllerPrefabs can allow the controllers to interact with any objects in the scene that have a Rigidbody component, when you reach out and touch them. Setting the Collision Detection property in the Rigidbody components of the prefabs created previous may help with this.



Figure 16.2.2:  
Controller based  
interaction using a  
mobile headset.

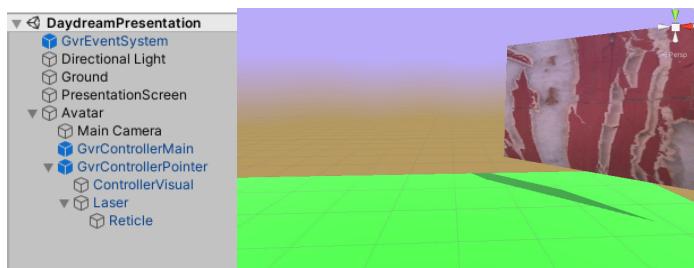
### 16.3 Deploying an enhanced reality experience on a Daydream

#### Component 16.3.1: Daydream presentation.

The Daydream headset and controller provide opportunities to present and interact with virtual content. This example demonstrates the process.

Daydream  
Presentation

Figure 16.3.1:  
A presentation  
space using the  
Daydream headset.



1. Start with a new project. Set this up to build on a mobile device.

Under the XR settings, enable Virtual Reality Supported. Add the Daydream to the list of supported SDKs.

Building and running the project at this stage demonstrates that much of the Daydream functionality is already present. The application requests the Daydream headset when the application starts, and performs the step of connecting the controller. Head movement is tracked, although this is harder to verify in the featureless scene present at this stage.

2. In the theme of presenting virtual worlds, in this case through a Daydream headset, this example will gloriously misinterpret the concept and develop an experience for emulating the mind-numbing presentations presented during meetings, seminars and lectures. A series of images are shown on a virtual screen. This does provide the opportunity to include the Daydream controller as an interactive part of the experience, to advance the slides and otherwise control the nature of the presentation.
3. Populate the scene with a ground plane and a presentation screen (a plane). Add a number of images to represent the different presentation slides. Create a material called PresentationSlideMaterial and add it to the presentation screen.

The camera needs to be set up carefully in the Daydream environment. The Daydream components en-

sure that the orientation of the camera is controlled using input from the device's gyroscope. Directly manipulating the camera's transform component is then inadvisable because it is likely to be overwritten almost immediately. Instead the solution is to create a parent object, representing the avatar of the participant and moving the user by manipulating the transform component of the avatar.

Create a new empty object named Avatar. Position this at the origin, at head height. Make the camera a child of this object and position it at the origin relative to the avatar (so also at head height).

4. Access to the Daydream controller requires the Google VR SDK for Unity. This package can be downloaded from <https://github.com/googlevr/gvr-unity-sdk/releases> and imported into the project. Try also installing the Google VR Android package using the package manager. Given the lack of support for the Daydream platform, it is likely that this process will only work for older versions of the Unity software.

Then make the following changes:

- (a) Set the camera near clipping plane property to 0.03, so that it doesn't cut off the view of the controller which may sometimes be less than 30 cm away (the default value).
- (b) From the Google VR SDK, add the GvrControllerMain (from GoogleVR/Prefabs/Controller) to the Avatar object. This is the manager object for the Daydream controller.
- (c) From the Google VR SDK, add the GvrControllerPointer (from GoogleVR/Prefabs/Controller) to the Avatar object. This ensures that the controller and camera both remain in position relative to another when the avatar is moved. The Controller Pointer prefab provides the visualization of the controller, but also models the movement of a virtual arm when

the controller is used. In this way the controller appears to move naturally despite it only being to track controller orientation and not position.

5. Handling events from the controller can be done in several ways. The approach shown here makes use of the integration between the Google VR SDK and the Unity software event system.
  - (a) Add a GvrEventSystem (from GoogleVR/Prefabs/EventSystem) to the scene.
  - (b) Add a GvrPointerPhysicsRaycaster component to the camera object, so that collisions between the controller pointer and objects in the scene generate events. Make sure the camera is still tagged as a Main Camera.
  - (c) Create a C# script called UpdatePresentation which contains some functions to manipulate the presentation object. Attach this to the presentation screen. This needs a set of slides. Import some suitable images and provide a list of these to the component. The code for controlling the slide show is provided in Algorithm 16.4.

Code Listing 16.4: `UpdatePresentation`. Slide show controls move backwards or forwards in the list of images provided as slides.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class UpdatePresentation : MonoBehaviour {
6
7      public List<Texture> slides;
8
9      private int currentSlide;
10
11     private void updateScreen ()
12     {
13         GetComponent <MeshRenderer> ().material.mainTexture
14         = slides[currentSlide];
15     }
16
17     public void Start ()
18     {
19         currentSlide = 0;

```

```

19     updateScreen ();
20 }
21
22 public void nextSlide ()
23 {
24     currentSlide = (currentSlide + slides.Count + 1) %
25     → slides.Count;
26     updateScreen ();
27 }
28
29 public void previousSlide ()
30 {
31     currentSlide = (currentSlide + slides.Count - 1) %
32     → slides.Count;
33     updateScreen ();
}

```

- (d) Add an event trigger component to the presentation screen object.

Create a new Pointer Click event and connect this to the nextSlide function on the UpdatePresentation component of the screen object.

You may need to set the Max Laser Distance field on the GvrLaserVisual component of the Laser object (child of the GvrControllerPointer) to a larger value so that the laser can reach the screen.

When the controller is aimed at the screen, and the touchpad on the controller is clicked, then the slide show moves to the next slide.

6. We can tweak the response to controller input to produce a more effective presentation clicker. The ability to move backwards and forwards through the slide set can be managed by swiping left and right on the touchpad. This action involves detecting when the touchpad is touched, recording the horizontal position, detecting when the touchpad is released and identifying a swipe by the change in horizontal position. The event system does not seem to produce events for touching the touchpad, so this has to be managed through a state machine instead.

The two states are: Touch pad being touched, and touch pad not being touched. Events causing transitions between states are the touch (and release) actions themselves. This state machine is implemented in Algorithm 16.5. Add this component to the presentation screen object as well.

Code Listing 16.5: SwipeAction. The state machine implementing the swipe gesture detection process on the Daydream controller touchpad.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class SwipeAction : MonoBehaviour {
6
7      // State machine states.
8      enum State { Touched, NotTouched };
9      private State state;
10
11     // Access the presentation to change slides.
12     private UpdatePresentation presentationControls;
13     // Access the controller to sense touchpad.
14     private GvrControllerInputDevice device;
15     // Remember horizontal position at start of swipe.
16     private float touchx;
17     // Determine length of movement to qualify as a swipe.
18     private float swipeThreshold = 0.3f;
19
20     void Start () {
21         state = State.NotTouched;
22         presentationControls = GetComponent<UpdatePresentation> ();
23         device = GvrControllerInput.GetDevice (GvrControllerHand.Dominant);
24     }
25
26     void Update () {
27         // Does not check for collisions with the screen, so
28         // works at any time.
29         switch (state)
30         {
31             case State.Touched:
32                 // Event touch up
33                 if (device.GetButtonUp
34                     (GvrControllerButton.TouchPadTouch))
35                 {
36                     state = State.NotTouched;
37                     if (device.TouchPos.x - touchx > swipeThreshold)
38                     {
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
317
318
319
319
320
321
322
323
324
325
326
327
327
328
329
329
330
331
332
333
334
335
336
337
337
338
339
339
340
341
342
343
344
345
346
347
347
348
349
349
350
351
352
353
354
355
356
357
357
358
359
359
360
361
362
363
364
365
366
367
367
368
369
369
370
371
372
373
374
375
376
377
377
378
379
379
380
381
382
383
384
385
386
387
387
388
389
389
390
391
392
393
394
395
396
397
397
398
399
399
400
401
402
403
404
405
406
407
407
408
409
409
410
411
412
413
414
415
416
416
417
418
418
419
419
420
421
422
423
424
425
426
426
427
428
428
429
429
430
431
432
433
434
435
436
436
437
438
438
439
439
440
441
442
443
444
445
446
446
447
448
448
449
449
450
451
452
453
454
455
456
456
457
458
458
459
459
460
461
462
463
464
465
466
466
467
468
468
469
469
470
471
472
473
474
475
476
476
477
478
478
479
479
480
481
482
483
484
485
486
486
487
488
488
489
489
490
491
492
493
494
495
496
496
497
498
498
499
499
500
501
502
503
504
505
506
506
507
508
508
509
509
510
511
512
513
514
515
515
516
517
517
518
518
519
519
520
521
522
523
524
525
525
526
527
527
528
528
529
529
530
531
532
533
534
535
535
536
537
537
538
538
539
539
540
541
542
543
544
545
545
546
547
547
548
548
549
549
550
551
552
553
554
555
555
556
557
557
558
558
559
559
560
561
562
563
564
565
565
566
567
567
568
568
569
569
570
571
572
573
574
575
575
576
577
577
578
578
579
579
580
581
582
583
584
585
585
586
587
587
588
588
589
589
590
591
592
593
594
595
595
596
597
597
598
598
599
599
600
601
602
603
604
604
605
606
606
607
607
608
608
609
609
610
611
612
613
614
614
615
616
616
617
617
618
618
619
619
620
621
622
623
624
624
625
626
626
627
627
628
628
629
629
630
631
632
633
634
634
635
636
636
637
637
638
638
639
639
640
641
642
643
644
644
645
646
646
647
647
648
648
649
649
650
651
652
653
654
654
655
656
656
657
657
658
658
659
659
660
661
662
663
664
664
665
666
666
667
667
668
668
669
669
670
671
672
673
674
674
675
676
676
677
677
678
678
679
679
680
681
682
683
684
684
685
686
686
687
687
688
688
689
689
690
691
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
701
702
703
703
704
705
705
706
706
707
707
708
708
709
709
710
711
712
713
713
714
715
715
716
716
717
717
718
718
719
719
720
721
722
723
723
724
725
725
726
726
727
727
728
728
729
729
730
731
732
733
733
734
735
735
736
736
737
737
738
738
739
739
740
741
742
743
743
744
745
745
746
746
747
747
748
748
749
749
750
751
752
753
753
754
755
755
756
756
757
757
758
758
759
759
760
761
762
763
763
764
765
765
766
766
767
767
768
768
769
769
770
771
772
773
773
774
775
775
776
776
777
777
778
778
779
779
780
781
782
783
783
784
785
785
786
786
787
787
788
788
789
789
790
791
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
801
802
802
803
804
804
805
805
806
806
807
807
808
808
809
809
810
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
1661
1661
1662
1662
1663
1663
1664
1664
1665
1665
1666
1666
1667
1667
1668
1668
1669
1669
1670
1671
1671
1672
1672
1673
1673
1674
1674
1675
1675
1676
1676
1677
1677
1678
1678
1679
1679
1680
1681
1681
1682
1682
1683
1683
1684
1684
1685
1685
1686
1686
1687
1687
1688
1688
```

```

37         presentationControls.nextSlide ();
38     }
39     if (device.TouchPos.x - touchx < -swipeThreshold)
40     {
41         presentationControls.previousSlide ();
42     }
43     // else just touched and released without moving.
44     }
45     break;
46 case State.NotTouched:
47     // Event touch down
48     if (device.GetButtonDown
→ (GvrControllerButton.TouchPadTouch))
49     {
50         touchx = device.TouchPos.x;
51         state = State.Touched;
52     }
53     break;
54 }
55 }
56 }
```

This variation doesn't check that the controller is pointing at the screen. This could be added as an extension.

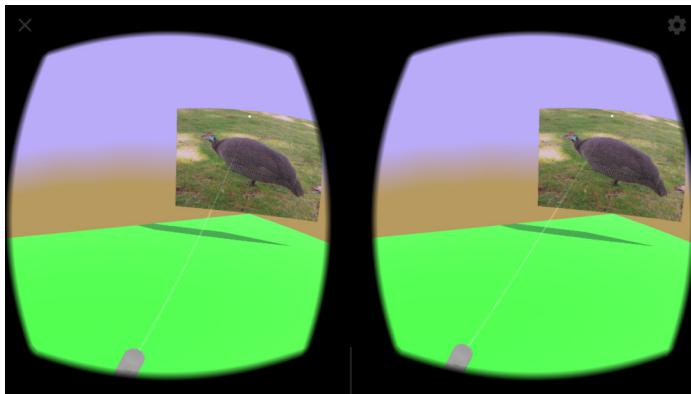


Figure 16.3.2: Controller and stereoscopic view using the Daydream headset.

## 16.4 Presenting 360° panoramic content on a headset

### Component 16.4.1: Display of stereoscopic panoramas.

Stereo
Panorama

*The following examples describe how to develop components that capture both real and synthetic environments in the form of stereoscopic panoramic imagery. More generally this is referred to as 3D 360 in reference to the opportunity for depth perception with the two image components of the stereoscopic image (3D) and the ability to look in any direction (360 degree view). This term does get abused to include both cylindrical panoramas where the rotational ability is limited to a horizontal plane (the viewer can only turn sideways) but also spherical panoramas that allow the viewer to look up and down as well. The more pedantic may refer to the latter as  $4\pi$  steradian images which is unlikely to catch on as a consumer product label.*

*Interestingly stereoscopic effects still assume that the two eyes occupy the same horizontal plane. Depth effects can drop off near the top and bottom of the view as these are often distorted when content is mapped to these regions. However tilting the head does tend to destroy the depth effect and is a handy test of whether a scene is being directly rendered using stereoscopic 3D or just achieving this by displaying content that was previously rendered or captured.*

*The process for creating stereoscopic panoramas is easier to understand and achieve if we have a way of verifying the output of the process. Thus this example will develop a viewer for the range of images that are produced. The core of this viewer is the same regardless of the particular format used.*

Figure 16.4.1: 3D environments presented using stereoscopic images.



1. A 3D 360 viewer requires two key components. The 3D aspect requires the ability to present two separate images to each eye. This is achieved using a headset; in this case a Daydream and a specific material shader

to select different content when rendering the left and right eye views. The 360 degree view results from mapping the image content onto either a cylinder or a sphere surrounding the user. Head rotation tracking provided by the Daydream allows the image presented to respond to the user's head movements.

The starting point is a fresh project. This needs to be set up as follows:

- (a) Set up as an android project in the build settings, provide an appropriate package name, and set an appropriate minimum API level (level 26 is suggested).
- (b) Under the XR settings, enable support for virtual reality and add the Daydream to the list of supported SDKs. This also ensures that the default camera now renders a left and right view. You may also need to install the Google VR Android package, using the package manager, to get Daydream applications to build and deploy on recent versions of the Unity software.

At this stage, the application should build, provide stereoscopic viewing in a Daydream headset and support head tracking. Given that the Daydream is not longer supported, you may want to substitute another head mounted display. The remaining instructions should work on any project that is already set up to support a head tracked headset.

2. Create a viewing surface for the image content. This can be either a cylinder or sphere depending on the nature of the images. The default objects provided by the Unity software are a reasonable starting point and can be replaced at a later point by manually modelled equivalents. More detailed models may be required if smaller polygons are needed to produce a smoother surface. Texture mapping may also be an issue that can

be resolved by carefully unwrapping the objects used as display surfaces.

For convenience ensure both camera and viewing surface object are placed at the origin. The viewing surface can then be scaled to a reasonable distance from the camera. The exact value is not too crucial but it should be sufficiently far away that any additional content (such as a controller) will not pass through it. A radius of 5 m is suggested. The vertical scale of a cylinder may need to be adjusted to fit the proportions of a simple panorama.

We want to disable the stereo rendering facilities provided when we enabled the Daydream VR mode. If we leave this on, then there will be a tendency for objects to appear on the surface on the object (sphere or cylinder) that we map the images to. One way to do this is to set the Stereo Separation property of the camera to 0. Both eyes end up at the same position and the only stereo effect is due to the difference between the left and right eye image content. Alternatively, if the scene does contain other content that needs to be rendered in stereo, increase the size of the viewing surfaces. Effects of eye displacement due to stereo separation in the camera reduce with distance so camera effects are minimized and the only stereo effect is due to the intrinsic difference between left and right images.

By default, the viewing surface is likely to be invisible when viewed from the camera's location. Materials are typically one sided and only the outside surface is rendered. This is addressed in the next step.

3. As part of the project create a new material called `StereoMaterial`, and a new shader called `StereoTexture`. The shader can be based on the `Unlit` shaders since the lighting effects on the viewing surface are already present in the captured image content. Ensure that the material makes use of the newly created shader. Add the material to the viewing surface object.

Edit the shader and make the following modifications:

- (a) Ensure that the inside surface is rendering by disabling backface culling.

```

1   ...
2   SubShader
3   {
4       Tags { "RenderType"="Opaque" }
5       LOD 100
6       Cull Off
7   ...

```

---

- (b) Modify the shader to accept two input textures, corresponding to the left and right images. For initial testing, add two visual distinct images to the project as textures, and supply them to the shader. Once the shader works the viewer can expect to see the left image in the left eye, and the right image in the right eye.
- (c) Adapt the vertex and fragment shaders to work with these images. In particular the fragment shader uses a Unity software interval variable to determine where it should select pixel colours from the left or the right texture images. The resulting shader is:

Code Listing 16.6: StereoTexture.

```

1 Shader "Unlit/StereoTexture"
2 {
3     Properties
4     {
5         LeftTex ("Texture", 2D) = "white" {}
6         RightTex ("Texture", 2D) = "white" {}
7     }
8     SubShader
9     {
10        Tags { "Queue"="Transparent"
11        "RenderType"="Transparent" }
12        LOD 100
13        Cull Off
14        ZWrite Off
15        Blend SrcAlpha OneMinusSrcAlpha
16
17        Pass
18        {
19            CGPROGRAM
#pragma vertex vert

```

```

20         #pragma fragment frag
21
22         #include "UnityCG.cginc"
23
24         struct appdata
25     {
26             float4 vertex : POSITION;
27             float2 uv : TEXCOORD0;
28         };
29
30         struct v2f
31     {
32             float4 objvertex : TEXCOORD1;
33             float2 uv : TEXCOORD0;
34             float4 vertex : SV_POSITION;
35         };
36
37         sampler2D LeftTex;
38         sampler2D RightTex;
39         float4 LeftTex_ST;
40         float4 RightTex_ST;
41
42         v2f vert (appdata v)
43     {
44             v2f o;
45             o.objvertex = v.vertex;
46             o.vertex = UnityObjectToClipPos(v.vertex);
47             o.uv = TRANSFORM_TEX(v.uv, LeftTex);
48             return o;
49         }
50
51         fixed4 frag (v2f i) : SV_Target
52     {
53         // fix mapping for a sphere
54         fixed2 uv;
55         float xz = sqrt(i.objvertex.x * i.objvertex.x +
56             i.objvertex.z * i.objvertex.z);
57         float latitude = atan2(i.objvertex.y, xz);
58         float longitude = atan2(i.objvertex.z,
59             i.objvertex.x);
60         uv.y = 0.5 + latitude / 3.14159;
61         uv.x = longitude / (2 * 3.14159);
62
63         // sample the texture
64         fixed4 col;
65         if (unity_StereoEyeIndex > 0)
66     {
67             col = tex2D(LeftTex, uv);
68         }
69         else
70     {
71             col = tex2D(RightTex, uv);
72         }

```

```

71     return col;
72 }
73 ENDCG
74 }
75 }
76 }
```

At this point, testing with the application running in the Daydream headset should show different images for each eye.

4. Texture quality can often be an issue, particularly on mobile devices. Some options to address this include:
  - (a) Under the Quality settings (one of the project settings menus) in the Unity software, select the default quality level for the mobile device (usually Medium). Switch on Anti Aliasing. The level chosen may need to adapt to specific requirements of your application, but the higher the better. Also ensure Anisotropic textures are enabled.
  - (b) Aniso Level can also be modified for each individual texture instead. This setting controls the way the texture is filtered (how neighbouring texels are combined when multiple texels need to be mapped onto a single pixel). Higher quality filtering improves the clarity of the texture. However it cannot compensate for an image that simply doesn't have the required detail.

## 16.5 Presenting experiences using audio rendering

### Component 16.5.1: Spatial sound component.

**Spatial Sound**

*This component requires a location tracking component, such as those developed in section 4. The spatial sound component developed here modifies the audio output in response to the movement of the user.*

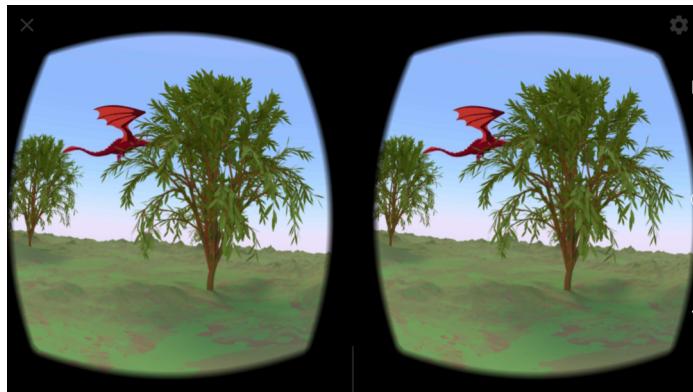


Figure 16.4.2:  
Stereoscopic images  
presented on  
devices that can  
display separate  
views to the left  
and right eyes  
respectively.

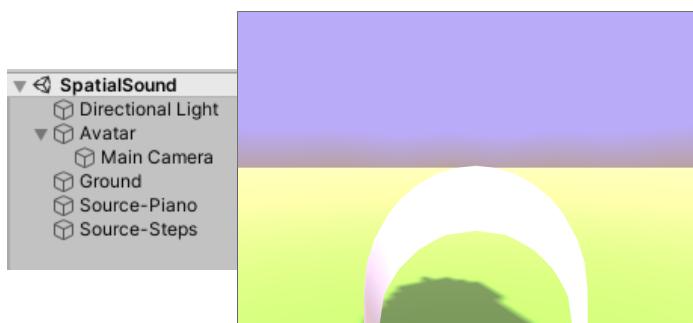


Figure 16.5.1: A  
test-bed for proto-  
typing concepts in  
spatial sound.

1. The scene set up is relatively simple. An avatar with a location tracking component is added to the scene. The AudioListener component is usually a part of the Camera object so this must be added as a child of the avatar in order for any built-in spatial effects to work. A number of objects with Audio Source components are also scattered around the scene. Set these sound sources to play from the start of the simulation, and to loop continuously. Achieving 3D spatialized sound with this component is relative simple: move the spatial blend slider all the way from 2D across to 3D. You should then be able to perceive a number of sound effects related to the position and movement of the avatar.
2. The 2D setting retains the original channels in the source (such as the left and right stereo channels) which may be adequate for static listeners but fails to take movement of the listener into account. The 3D setting combines all channels into one, but then modifies the sound to provide effects reflecting the spatial positioning of source and listener.
3. Volume drops off with distance to the listener. The loudness of sound is related to the energy contained in the vibrations. This energy propagates outwards from the source over the surface of a sphere, with surface area  $4\pi r^2$ , where  $r$  is the radius of the sphere. Thus the volume drops off with distance from the source, in proportion to  $\frac{1}{r^2}$ . This is quite a rapid decline with distance and so many virtual reality engines offer more controlled declines - either  $\frac{1}{r}$ , a constant ("linear") decline until volume reaches 0 at a set distance, or some customized variation usually to retain a certain volume level within a set zone around the user.
4. The Doppler effect is due to the relative movement of the source and listener. When the source is approaching the listener (similarly for when the listener approaches the source) the peaks of the sound waves

are closer together because the source is moving in the same direction as the emitted sound. This shorter wavelength is equivalent to an increase in frequency which is manifest as an increase in the pitch of the sound. As the source moves past the listener, the increase in pitch abruptly changes to a decrease as each new sound wave is stretched out as the source moves in the opposite direction to the previous peak.

Assuming that the relative velocity of sound source and listener is  $v$ , and the unit vector  $\hat{d}$  represents the direction between source and listener, the relative speed in the direction from source to listener is given by:

$$s = v \cdot \hat{d}$$

This speed is positive if the source and listener are approaching one another, and negative if they are heading in opposite directions. The resulting change of pitch can then be modelled by an expression such as:

5.

$$f_{doppler} = \left( \frac{c + s}{c - s} \right) f$$

Here  $c$  represents the speed of sound in the medium (such as air).

6. The component developed in Algorithm 16.7 is intended to demonstrate how specific elements of audio spatialization can be achieved, in case only particular effects are required, or in scenarios where existing audio spatialization is not available on the intended target platforms.

Code Listing 16.7: `AudioSpatializer`. Audio spatialization effects such as attenuation and Doppler shift are demonstrated in this component.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class AudioSpatializer : MonoBehaviour {

```

```
6   public List< AudioSource > sources;
7
8   public float dropoffDistanceConstant = 0.8f;
9
10  public float attenuationFactor = 1.5f; // Should be 2.0f;
11
12  public float speedOfSound = 330.0f;
13
14
15  void Update () {
16      foreach ( AudioSource source in sources )
17      {
18          // Volume effects.
19          GameObject sourceObject = source.gameObject;
20          float distance = Vector3.Distance
21              ( sourceObject.transform.position, transform.position );
22          source.volume = 1.0f / Mathf.Pow
23              ( dropoffDistanceConstant * distance,
24              attenuationFactor );
25
26          // Doppler effects.
27          // Assume sound sources are stationary.
28          Vector3 sourceVelocity = new Vector3 ( 0, 0, 0 );
29          Vector3 myVelocity = GetComponent
30              < PrototypeLocationTracking > ().getVelocity ();
31          Vector3 relativeVelocity = myVelocity -
32              sourceVelocity;
33          Vector3 directionBetweenMeAndSource =
34              Vector3.Normalize ( sourceObject.transform.position -
35                  transform.position );
36          float relativeSpeed = Vector3.Dot
37              ( directionBetweenMeAndSource, relativeVelocity );
38
39          source.pitch = ( speedOfSound + relativeSpeed ) /
40              ( speedOfSound - relativeSpeed );
41
42      }
43  }
```

Make sure all engine spatializers are turned off when using this component (for example, by having the spatial blend slider on 2D). Turn the spatial blend slider to 3D to take advantage of the spatial sound facilities provided by the Unity software. These will provide many of the same effects as demonstrated in the previous steps.



Figure 16.5.2:  
Spatial sound  
is affected by  
the position and  
movement of the  
sound source and  
listener.

## 16.6 Providing virtual reality experiences using web technologies

### Component 16.6.1: WebVR and A-Frame.

 Web VR

*There are several outcomes that are achieved by this example demonstrating facilities available via WebVR and A-Frame.*

*WebVR provides ways to integrate content display in a browser with virtual reality hardware. In this case, the content is displayed on a Daydream headset, and interaction is provided using the Daydream controller.*

*A-Frame provides additional markup tags that describe the scene structure.*

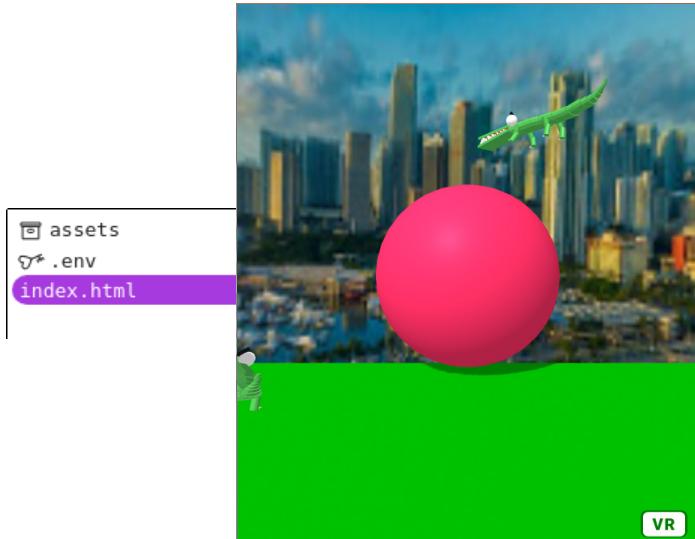
1. Start a new project on [glitch.com](https://glitch.com). Edit the index.html file to provide a minimal scene.

```

1  <!DOCTYPE html>
2  <html lang="en">
3    <head>
4      <title>Daydream VR experience using WebVR and A-
          Frame</title>
5      <script src="https://aframe.io/releases/0.9.0/
          aframe.min.js"></script>
6    </head>
7    <body>
8      <a-scene>
9        <a-sphere id="coloursphere" position="0 1.25
          -5" radius="1.25" color="#EF2D5E" shadow
        ></a-sphere>

```

Figure 16.6.1: Web VR applications using A-Frame.



```

10      <a-plane position="0 0 0" rotation="-90 0 0"
11          width="5" height="5" color="#00AA00"
12          shadow></a-plane>
13      <a-camera position="0 1.5 0" rotation="0 0 0"
14          width="5" height="5" color="#00AA00"
15          shadow></a-camera>
16      </a-scene>
17  </body>
18 </html>
```

---

Open this scene in a browser on the mobile device used with the Daydream headset. The 3D scene is visible in the browser window and responds to changes in the orientation of the phone.

Now insert the device into the Daydream headset. This should invoke the Daydream application, check controller connection and launch into the stereoscopic version of the browser (i.e. fully immersive, not just a floating window). If not, you may have to navigate to: chrome://flags/#enable-webvr (if using the chrome browser) and enable this functionality.

This step validates that the 3D scene can be created and effectively viewed as a virtual reality experience using the Daydream headset.

2. The Daydream controller is required to provide input to the application. This may also need to be enabled in the browser: `chrome://flags/#enable-gamepad-extensions` (from the chrome browser).

Including the daydream controls element in the scene makes a virtual representation of the controller appear, which tracks the rotation of the controller, and adds some positional movement through an arm model.

---

```
1      <a-entity daydream-controls></a-entity>
```

---

3. Responses to interaction are achieved by attaching javascript event handlers to events generated in the virtual environment. In this case, the component responsible for setting event handlers is specified as a property of the controller object.

---

```
1      <a-entity daydream-controls daydream-listener
2          ></a-entity>
```

---

The daydream-listener component then creates event listeners for buttons on the daydream controller. The touchpad is the control used, since the other buttons are dedicated to either managing the Daydream system or the web browser interaction. Events can be based on the clicking or touching of the touchpad, or can make use of the axismove event produced by the generic tracked controls component that the individual controller derive from.

```
1      <script>
2          AFRAME.registerComponent("daydream-listener",
3              {
4                  init: function() {
5                      this.el.addEventListener("axismove", (e)
6                          => {
7                              var colourSphere = document.
8                                 querySelector('#coloursphere');
9                              colourSphere.setAttribute ("color", "rgb
10                                 (" + parseInt((1 + e.detail.axis[0])
11                                 *127) + "," + parseInt((1+e.detail.
12                                 axis[1])*127) + ",0)");
13                          })
14                  }
15              })
```

---

10        </script>

---

This script manipulates the colour of an object in the scene (the one with id set as coloursphere) based on the position of the finger on the touchpad.

- Having established the interface with the hardware, the next step is to construct a more attractive scene.

Textures can be added to objects by uploading them to glitch, and accessing them via a URL. Select the assets region on the glitch project and upload the textures that you would like to use. Clicking on any individual asset that you've uploaded reveals the URL that can be used to access it.

Individual assets can be added directly to scene elements but it is more efficient to keep these under a separate assets section of the scene. This allows the assets to be named, and referred to by these names. This simplifies changes if any of these assets need to be replaced in the future. Assets that are reused also then need to be loaded only once, rather than each time they are required.

The assets section consists of:

---

```
1      <a-assets>
2          
3      </a-assets>
```

---

An asset (sky box) that uses the texture then invoke it as:

---

```
1      <a-sky src="#backgroundImage" radius="10"></a-
sky>
```

---

- Geometry can also be imported in the form of 3D models. As with the textures, upload the files in an appropriate format (such as OBJ) to the assets region on the glitch project.

The assets section contains references to the .obj and .mtl files:

```

1      <a-asset-item id="croc-obj" src="https://cdn
     .glitch.com/b4b3f58a-8112-4ded-a9be-6
     f0155a6ced1%2Fmodel.obj?1554006241707"
     ></a-asset-item>
2      <a-asset-item id="croc-mtl" src="https://cdn
     .glitch.com/b4b3f58a-8112-4ded-a9be-6
     f0155a6ced1%2Fmaterials.mtl
     ?1554006238081"></a-asset-item>
```

---

These objects can then be instantiated as required in the scene:

```

1      <a-obj-model src="#croc-obj" mtl="#croc-mtl"
     position="-2 0.5 -3"></a-obj-model>
2      <a-obj-model src="#croc-obj" mtl="#croc-mtl"
     position="3 0.5 -3"></a-obj-model>
```

---

6. Object movement is achieved through animation properties. So, for example, objects can be made to rain down using:

```

1      <script>
2          setInterval (function () {
3              var scene = document.querySelector ("a-scene
                 ");
4              var croc = document.createElement ("a-obj-
                 model");
5              var x = 10 * (Math.random () - 0.5);
6              var z = 10 * (Math.random () - 0.5);
7              croc.setAttribute ("src", "#croc-obj");
8              croc.setAttribute ("mtl", "#croc-mtl");
9              croc.setAttribute ("position", x + " 3.5 " +
                 z);
10             croc.setAttribute ("animation", "property:
                 position; to: " + x + " 0 " + z + "; dur:
                 : 5000; easing: linear; loop: false;");
11             // croc.setAttribute ("animation__2", "
                 property: rotation; to: 0 360 0; loop:
                 true; dur: 1000; easing: linear;");
12             croc.addEventListener ("animationcomplete",
                 function () { croc.parentNode.
                 removeChild (croc); });
13
14             scene.appendChild (croc);
15
16             }, 2000);
17         </script>
```

---

In this case, the animation property is added to a procedurally generation node. This uses the javascript setInterval function to repeat a process at regular intervals consisting of creating a new object, setting animations to both spin and lower the object, and defines an event listener that uses the end of animation (object reaching the ground) to signal a function that removes the object.

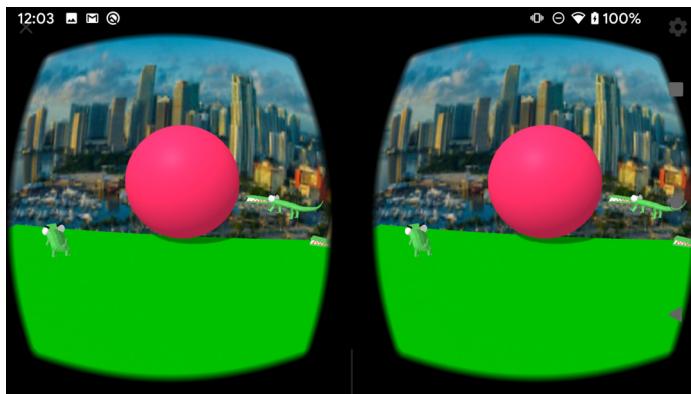


Figure 16.6.2: A virtual reality scene delivered through a mobile web browser.

## 16.7 Presenting augmented reality experiences using web technologies

### Component 16.7.1: WebAR.

**Web AR**

This example demonstrates a typical augmented reality application with 3D model overlaid on an image of a visible marker. The AR.js library is used (<https://github.com/AR-js-org/AR.js>) as this provides integration with previous material covering WebVR and A-Frame.

1. Start by creating a new project on [glitch.com](https://glitch.com). Create or modify the index.html file to contain a basic WebVR/A-Frame based virtual world.

```

1  <!DOCTYPE html>
2  <html lang="en">
3      <head>
4          <title>WebAR Application</title>
5          <script src="https://aframe.io/releases/1.2.0/
aframe.min.js"></script>
```

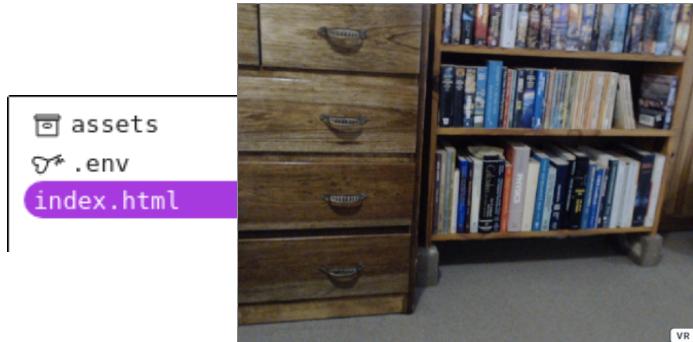


Figure 16.7.1:  
Marker based  
augmented reality  
in a web browser.

```

6   </head>
7   <body>
8     <a-scene>
9       <a-plane position="0 0 0" rotation="-90 0 0"
           width="10" height="10" color="#00AA00"
           shadow></a-plane>
10      <a-box position="0 1 0" rotation="0 0 0" color
            ="#999999" shadow></a-box>
11      <a-camera position="0 1 3" rotation="0 0 0"><
            a-camera>
12    </a-scene>
13  </body>
14 </html>
```

---

2. Access to the AR is achieved through an annotation to the scene tag. The ar.js script must also be imported into the head of the document.

```

1  <script src="https://raw.githubusercontent.com/AR-js-org/AR.js
   /master/aframe/build/aframe-ar.js"></script>
2  ...
3  <a-scene embedded arjs>
```

---

The view from the physical camera should now be visible in the background to the scene. Setting the camera position to the origin ensures that AR content is correctly placed.

3. The next step is to add a marker image. It may help at this stage to remove the plane and the box from the scene.

The markers supported by the AR Toolkit are rather limited in form. Barcode style markers can be used.

Image markers can be used but should be distinctive. In particular they need a solid border to be identified.

Visit the site: <https://jeromeetienne.github.io/AR.js/three.js/examples/marker-training/examples/generator.html> to generate a pattern marker. Upload your chosen image, and download the resulting marker, in the form of a .patt file. Barcode style markers can also be created using the generator located at: <http://augmented.com/app/marker/marker.php>.

This marker can be uploaded to the glitch.com project under the assets section. Click on the uploaded object to retrieve the URL to reference that asset.

The tag to identify the marker is then of the form:

```

1 <a-marker type="pattern" url="https://cdn.glitch.com/
  /...">
2   // Any content to be displayed when marker is
      recognised.
3 </a-marker>
```

---

4. It is possible to place boring content on the markers. However custom 3D models can also be included using the appropriate A-Frame facilities. Upload your 3D object in .obj format to the assets area in glitch.com. Click on the uploaded object and copy the address to where it is hosted.

The tag to include the model can then be nested under the marker tag.

```

1 <a-obj-model src="https://cdn.glitch.com/...
  rotation="-90 0 0" color="#999999"></a-obj-model>
```

---

5. The complete file to create a marker based augmented reality system using one of the WebAR libraries is provided in Algorithm 16.8.

Code Listing 16.8: index. WebAR allows AR applications to be generated using a small number of tags.

```

1 <!DOCTYPE html>
2 <html lang="en">
```

```
3   <head>
4     <meta
5       name="viewport"
6       content="width=device-width, user-scalable=no,
7       ↳   minimum-scale=1.0, maximum-scale=1.0"
8     />
9     <script src="https://aframe.io/releases/1.2.0/aframe.m
10    ↳ in.js"></script>
11     <script src="https://raw.githubusercontent.com/AR-js-org/AR.js/m
12    ↳ aster/aframe/build/aframe-ar.js"></script>
13   </head>
14
15   <body style="margin : 0px; overflow: hidden;">
16
17     <a-scene embedded arjs>
18       <a-marker
19         type="pattern"
20         url="https://cdn.glitch.com/3fb0eec5-5fbf-4b6a-b91
21         ↳ 0-bed35b5d2b9f%2Fpattern-m4.patt?v=16152615587"
22         ↳ 38"
23       >
24         <a-obj-model
25           src="https://cdn.glitch.com/3fb0eec5-5fbf-4b6a-b
26           ↳ 910-bed35b5d2b9f%2Fmonkey.obj?v=161516612152"
27           ↳ 9"
28             rotation="-90 0 0"
      color="#999999"
      ></a-obj-model>
    </a-marker>
    <a-entity camera></a-entity>
  </a-scene>
</body>
</html>
```

Figure 16.7.2: 3D object overlaid on a marker image.



# 17

## *Location (Setting) Components*

### Contents

---

17.1	Tracking your location globally using GPS . . . . .	226
17.2	Accessing and presenting your setting using map tile services . . . . .	230
17.3	Reconstructing your setting as a terrain mesh . . . . .	237
17.4	Mapping your physical location as a virtual point cloud . . . . .	246
17.5	Capturing your environment as a 3D mesh with the Kinect . . . . .	261
17.6	Capturing and presenting scenes using 360° cameras . . . . .	271
17.7	Capturing cylindrical panoramas of scenes using Cardboard Camera and a mobile phone . . . . .	278
17.8	Converting complex virtual scenes to stereo panoramic images using Unity	279
17.9	Converting complex virtual scenes to stereo panoramic images using Blender	282
17.10	Transitioning between realities using portals . . . . .	287
17.11	Aligning overlapping realities . . .	297

---

## 17.1 Tracking your location globally using GPS

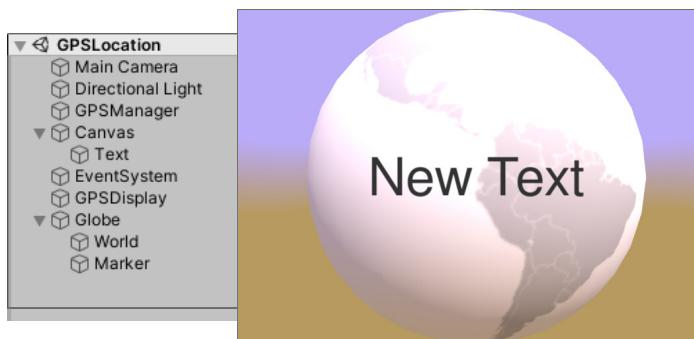
### Component 17.1.1: GPS based location.

**GPS Location**

*Motivation:* This component provides a way to track your position on a global scale using a global positioning system (GPS). While this does provide opportunities to track the participant and allow them to interact with the experience (the subject of component in Chapter 19) this component has been included in the Location (Setting) category in that it is primarily used to understand where you are in a global setting.

This example demonstrates the process to retrieve GPS position information. This typically only runs on a mobile device equipped with a GPS receiver, and where this receiver is turned on.

Figure 17.1.1: Using GPS to establish location.



1. Start with a new project. Configure to build on a mobile device.

Create a new empty and name this GPS manager. This will provide access to location information, via a GPSTracking component that we create. Create a new C# script named GPSTracking to provide the functionality. The full code for this script is provided in Algorithm 17.1.

Code Listing 17.1: GPSTracking. GPS tracking functionality is encapsulated into this component, which provides an interface to retrieve the latest position value.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class GPSTracking : MonoBehaviour {
6
7      /*
8       * Retrieve the location from the location service,
9       * typically using GPS. Returns true
10      * if the operation succeeded, or false if location is
11      * not available at the current
12      * time.
13      */
14
15      public bool retrieveLocation (out float latitude, out
16      float longitude, out float altitude)
17      {
18          latitude = 0.0f;
19          longitude = 0.0f;
20          altitude = 0.0f;
21
22          if (!Input.location.isEnabledByUser)
23          {
24              Debug.Log ("Location service needs to be enabled");
25              return false;
26          }
27          if (Input.location.status !=
28              LocationServiceStatus.Running)
29          {
30              Debug.Log ("Starting location service");
31              if (Input.location.status ==
32                  LocationServiceStatus.Stopped)
33              {
34                  Input.location.Start ();
35              }
36              return false;
37          }
38          else
39          {
40              // Valid data is available.
41              latitude = Input.location.lastData.latitude;
42              longitude = Input.location.lastData.longitude;
43              altitude = Input.location.lastData.altitude;
44              return true;
45          }
46      }
47 }
```

2. Initially the location can be displayed as text on the screen of the device. Add a new UI/Text element.  
Create another empty object and attach a script to

query the location service, and update the display. The code for this is shown in Algorithm 17.2. The script requires that the GPS Manager object be provided as the source of the location service, and the text element defines where the output is written.

Code Listing 17.2: `GPSSDisplay`. The GPS location service can be validated by querying the location component and writing the output to a text display.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class GPSSDisplay : MonoBehaviour {
7
8      public GPSTracking locationService;
9
10     public Text displayText;
11
12     void Update () {
13         float latitude;
14         float longitude;
15         float altitude;
16         if (locationService.retrieveLocation (out
17             latitude, out longitude, out altitude))
18         {
19             displayText.text = "Lat: " + latitude + "\n" +
20                             "Long: " + longitude + "\n"
21                             +
22                             "Alt: " + altitude;
23         }
24         else
25         {
26             displayText.text = "No location";
27         }
28     }
29 }
```

3. As a further refinement, we can display the coordinates on a sphere containing a map of the world. Create a new sphere object in the scene. Add a material containing a texture of the map of the world. Note that, unless the sphere is perfectly unwrapped to match the projection used in the map on the texture, GPS positions will not line up exactly with regions shown on the sphere. An equirectangular projection is most likely to match

typical sphere texture mappings. The prime meridian (0 longitude) may not be on the left hand side of the image. The texture offset can be tweaked in the Offset field of the material setting.

Create a small marker object that can be placed on the sphere to indicate current position.

Add a script to both rotate the sphere (spin it on its axis), and to update the position of the marker. The script is shown in Algorithm 17.3. Remember to assign the properties required: the GPS manager for the location service, and the marker object.

Code Listing 17.3: ShowPosition. GPS coordinates are in a spherical coordinate system that can be mapped to points on the surface of a sphere.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ShowPosition : MonoBehaviour {
6
7      public GPSTracking locationService;
8
9      public GameObject marker;
10
11     public float globeRadius = 10.0f;
12
13     public float rotationSpeed = 30.0f;
14
15     void Update () {
16         // Rotate the globe.
17         transform.rotation *= Quaternion.AngleAxis
18             (rotationSpeed * Time.deltaTime, Vector3.up);
19
20         // Plot position.
21         float latitude;
22         float longitude;
23         float altitude;
24         if (locationService.retrieveLocation (out
25             latitude, out longitude, out altitude))
26         {
27             Vector3 position = globeRadius *
new Vector3 (Mathf.Cos (latitude *
Mathf.Deg2Rad) * Mathf.Cos (longitude * Mathf.Deg2Rad),
            Mathf.Sin (latitude *
Mathf.Deg2Rad),

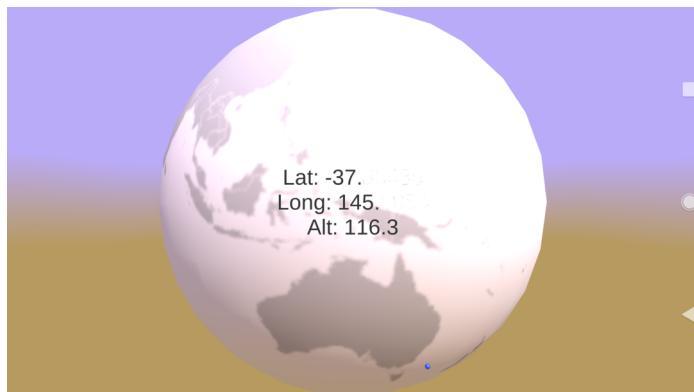
```

```

28           Mathf.Cos (latitude *
29             ↵ Mathf.Deg2Rad) * Mathf.Sin (longitude *
30             ↵ Mathf.Deg2Rad));
31             marker.transform.localPosition = position;
32         }
}

```

Figure 17.1.2:  
Establishing setting  
using GPS location.



## 17.2 Accessing and presenting your setting using map tile services

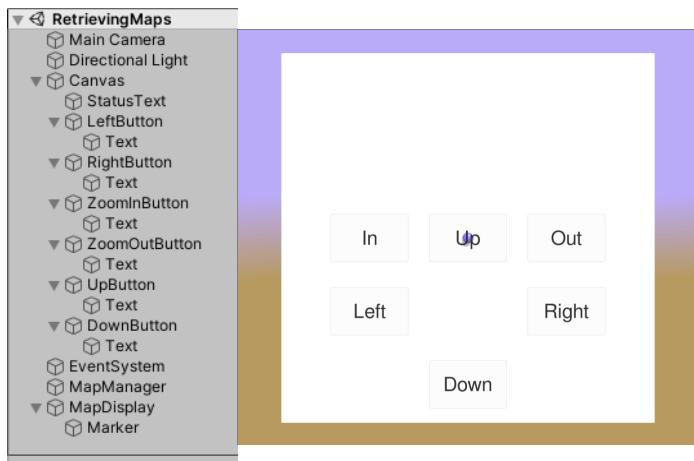
### Component 17.2.1: Retrieving maps.

Retrieving  
Maps

*This example focuses on retrieving map tiles from an online service. Location services are not used but since all access uses longitude and latitude it should be a simple matter to integrate such facilities as illustrated in other examples.*

1. Start with a new project. Set this up to run on a mobile device. However since the services being used are generic, this example works just as well on a desktop system.
2. Add some controls. We need four buttons for moving horizontally and vertically over the map area. Two buttons will be used to control the zoom level. A text

Figure 17.2.1:  
Retrieving setting  
information from  
an online service.



element provides a way of tracking the current coordinates. The UI/Button and UI/Text elements are fine for these purposes.

3. Create an empty object called MapManager. The first step is to add a script (C# script called MoveWithButtons) which receives events from each of the buttons, and adjusts the coordinates appropriately. The relevant portion at this stage is the function onButtonEvent, which each button connects to. Connect the OnClick property of each button to the corresponding function on the MapManager's MoveWithButtons component.

```

1  public void onButtonEvent (float dx, float dy, int
   dz)
2  {
3      zoom += dz;
4      // Calculate step so that it takes a few button
   presses
5      // to move across a tile at that level.
6      float step = 0.3f * 1.0f / Mathf.Pow (2.0f, zoom
   );
7      longitude += 360.0f * dx * step;
8      latitude += 180.0f * dy * step;
9
10     updateMapView ();
11 }
12 public void leftButton () { onButtonEvent (-1.0f,
   0.0f, 0); }
13 public void rightButton () { onButtonEvent (1.0f,
   0.0f, 0); }
```

```

14     public void upButton () { onButtonEvent (0.0f, 1.0
15         f, 0); }
16     public void downButton () { onButtonEvent (0.0f,
17         -1.0f, 0); }
18     public void inButton () { onButtonEvent (0.0f, 0.0
19         f, 1); }
20     public void outButton () { onButtonEvent (0.0f,
21         0.0f, -1); }

```

---

4. The next step is to flesh out the updateMapView function to download a map tile and display it on the screen. Add a plane to the scene. Create a new material called MapMaterial and add it to the plane. The plane displays nicely if the rotation is set to 90, 180, 0.

There are several steps in retrieving a map tile. The current coordinates (latitude and longitude) need to be converted into tile coordinates. The tile must be retrieved and copied to the texture of the material on the plane. If a marker needs to be shown on the tile, then the coordinates of the edges of the tile need to be converted back to latitude and longitude.

Converting between geographical and tile coordinates uses some well established functions.<sup>1</sup> Retrieving the tile texture can be done via a web request.

<sup>1</sup> [https://wiki.openstreetmap.org/wiki/Slippy\\_map\\_tilenames](https://wiki.openstreetmap.org/wiki/Slippy_map_tilenames)

```

1  private void updateTexture (int x, int y, int z)
2  {
3      string url = "https://a.tile.openstreetmap.org/" + z
4          + "/" + x + "/" + y + ".png";
5      // Similar process with another map service.
6      // Avoid use without considering terms of service.
7      // string url = "https://mt.google.com/vt/lyrs=m&x="
8          + x + "&y=" + y + "&z=" + z;
9      Debug.Log ("Retrieving: " + url);
10     WebRequest www = WebRequest.Create (url);
11     ((HttpWebRequest) www).UserAgent = "RetrievingMaps";
12
13     var response = www.GetResponse ();
14
15     Texture2D tex = new Texture2D (2, 2);
16     // Retrieve a large number of bytes - should be more
17         than in a tile texture.
18     ImageConversion.LoadImage (tex, new BinaryReader (
19         response.GetResponseStream ()).ReadBytes
20         (1000000));
21     mapMaterial.mainTexture = tex;

```

---

 17 }

Some care does have to be taken to ensure the function works on recent versions of Android. Specifically these only allow https connections (no more http) and additional settings (see full code) are required to check the certificates.

The setting for Internet access under Player Settings/Other Settings should be set to required.

5. To show a marker, create a small object to represent a point on the map. The position of this marker can be interpolated from the coordinates of the corners of the current tile.

```

1  getGeoCoordinates (x, y, zoom, out cornerLongA, out
                     cornerLatA);
2  getGeoCoordinates (x + 1, y + 1, zoom, out
                     cornerLongB, out cornerLatB);
3  // interpolate current coordinates relative to the
     coordinates of the
4  // tile corners. Assumes the plane coordinates run
     from (-5,-5) to (5,5).
5  float r = 10.0f * ((-(longitude - cornerLongA) / (
                     cornerLongB - cornerLongA)) + 5.0f;
6  float d = 10.0f * ((-(latitude - cornerLatA) / (
                     cornerLatB - cornerLatA)) + 5.0f;
7  marker.transform.position = mapPlane.transform.
     position - mapPlane.transform.forward * d +
     mapPlane.transform.right * r;

```

---

6. The full script used is provided in Algorithm 17.4. This could be made more efficient by caching tiles that are downloaded and reusing these, rather than re-downloading them even when the same tile is required.

Code Listing 17.4: MoveWithButtons. Map tile retrieval converts latitude and longitude to tile coordinates before sending a web request to download the corresponding texture.

1	<b>using</b> System.Collections;
2	<b>using</b> System.Collections.Generic;
3	<b>using</b> UnityEngine;
4	<b>using</b> UnityEngine.UI;
5	<b>using</b> UnityEngine.Networking;

```
6   using System.Net.Security;
7   using System.Security.Cryptography.X509Certificates;
8   using System.Net;
9   using System.IO;
10
11  public class MoveWithButtons : MonoBehaviour {
12
13      public Text statusText;
14
15      public Material mapMaterial;
16
17      public GameObject marker;
18      public GameObject mapPlane;
19
20      private float longitude = 0.0f;
21      private float latitude = 0.0f;
22      private int zoom = 0;
23
24      private static bool TrustCertificate (object sender,
25      X509Certificate x509Certificate, X509Chain x509Chain,
26      SslPolicyErrors sslPolicyErrors)
27      {
28          // All Certificates are accepted. Not good
29          // practice, but outside scope of this
30          // example.
31          return true ;
32      }
33
34      void Start ()
35      {
36          ServicePointManager.ServerCertificateValidationCallback =
37          k =
38          TrustCertificate;
39          updateMapView ();
40      }
41
42      private void getTileCoordinates (float longitude, float
43      latitude, int zoom, out int x, out int y)
44      {
45          x = (int) (Mathf.Floor ((longitude + 180.0f) / 360.0f
46          * Mathf.Pow (2.0f, zoom)));
47          y = (int) (Mathf.Floor ((1.0f - Mathf.Log (Mathf.Tan
48          (latitude * Mathf.PI / 180.0f) + 1.0f / Mathf.Cos
49          (latitude * Mathf.PI / 180.0f)) / Mathf.PI) / 2.0f *
50          Mathf.Pow (2.0f, zoom)));
51      }
52
53      private void getGeoCoordinates (int x, int y, int zoom,
54      out float longitude, out float latitude)
55      {
56          float n = Mathf.PI - 2.0f * Mathf.PI * y / Mathf.Pow
57          (2.0f, zoom);
```

```

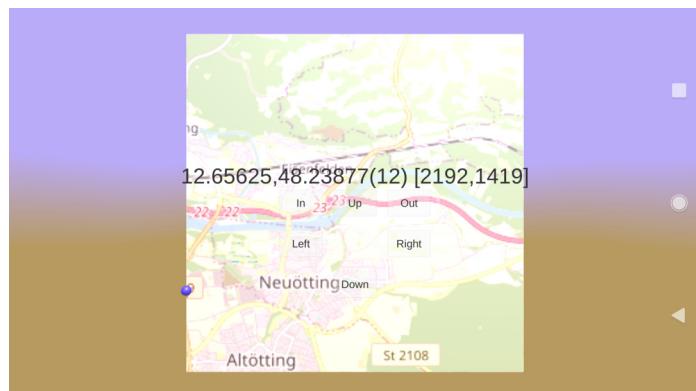
48     longitude = x / Mathf.Pow (2.0f, zoom) * 360.0f -
49     ↵ 180.0f;
50     latitude = 180.0f / Mathf.PI * Mathf.Atan (0.5f *
51     ↵ (Mathf.Exp (n) - Mathf.Exp (-n)));
52 }
53
54 private void updateTexture (int x, int y, int z)
55 {
56     string url = "https://a.tile.openstreetmap.org/" + z +
57     ↵ "/" + x + "/" + y + ".png";
58     // Similar process with another map service.
59     // Avoid use without considering terms of service.
60     // string url = "https://mt.google.com/vt/lyrs=m&x=" +
61     ↵ x + "&y=" + y + "&z=" + z;
62     Debug.Log ("Retrieving: " + url);
63     WebRequest www = WebRequest.Create (url);
64     ((HttpWebRequest) www).UserAgent = "RetrievingMaps";
65
66     var response = www.GetResponse ();
67
68     Texture2D tex = new Texture2D (2, 2);
69     // Retrieve a large number of bytes - should be more
70     ↵ than in a tile texture.
71     ImageConversion.LoadImage (tex, new BinaryReader
72     ↵ (response.GetResponseStream ()).ReadBytes (1000000));
73     mapMaterial.mainTexture = tex;
74 }
75
76
77 private void updateMapView ()
78 {
79     int x;
80     int y;
81     getTileCoordinates (longitude, latitude, zoom, out x,
82     ↵ out y);
83     updateTexture (x, y, zoom);
84
85     // Place a marker at the current position on the tile.
86     float cornerLatA;
87     float cornerLongA;
88     float cornerLatB;
89     float cornerLongB;
90     getGeoCoordinates (x, y, zoom, out cornerLongA, out
91     ↵ cornerLatA);
92     getGeoCoordinates (x + 1, y + 1, zoom, out
93     ↵ cornerLongB, out cornerLatB);
94     // interpolate current coordinates relative to the
95     ↵ coordinates of the
96     // tile corners. Assumes the plane coordinates run
97     ↵ from (-5,-5) to (5,5).
98     float r = 10.0f * ((-(longitude - cornerLongA) /
99     ↵ (cornerLongB - cornerLongA))) + 5.0f;
100    float d = 10.0f * ((-(latitude - cornerLatA) /
101     ↵ (cornerLatB - cornerLatA))) + 5.0f;

```

```

88     marker.transform.position =
89     ↳ mapPlane.transform.position -
90     ↳ mapPlane.transform.forward * d +
91     ↳ mapPlane.transform.right * r;
92
93     statusText.text = longitude + "," + latitude + "(" +
94     ↳ zoom + ")" + "[" + x + "," + y + "]";
95 }
96
97 public void onButtonEvent (float dx, float dy, int dz)
98 {
99     zoom += dz;
100    // Calculate step so that it takes a few button
101    // presses
102    // to move across a tile at that level.
103    float step = 0.3f * 1.0f / Mathf.Pow (2.0f, zoom);
104    longitude += 360.0f * dx * step;
105    latitude += 180.0f * dy * step;
106
107    updateMapView ();
108 }
109
110 public void leftButton () { onButtonEvent (-1.0f, 0.0f,
111     ↳ 0); }
112 public void rightButton () { onButtonEvent (1.0f, 0.0f,
113     ↳ 0); }
114 public void upButton () { onButtonEvent (0.0f, 1.0f, 0);
115     ↳ }
116 public void downButton () { onButtonEvent (0.0f, -1.0f,
117     ↳ 0); }
118 public void inButton () { onButtonEvent (0.0f, 0.0f, 1);
119     ↳ }
120 public void outButton () { onButtonEvent (0.0f, 0.0f,
121     ↳ -1); }
122 }
```

Figure 17.2.2: A representation of setting for location based augmented reality.



### 17.3 Reconstructing your setting as a terrain mesh

#### Component 17.3.1: Reconstructing Terrain Maps

**Terrain Maps**

Tiles can contain values other than colour layers. The following steps adapt the previous example to download elevation data which represents the height of the terrain at each point.

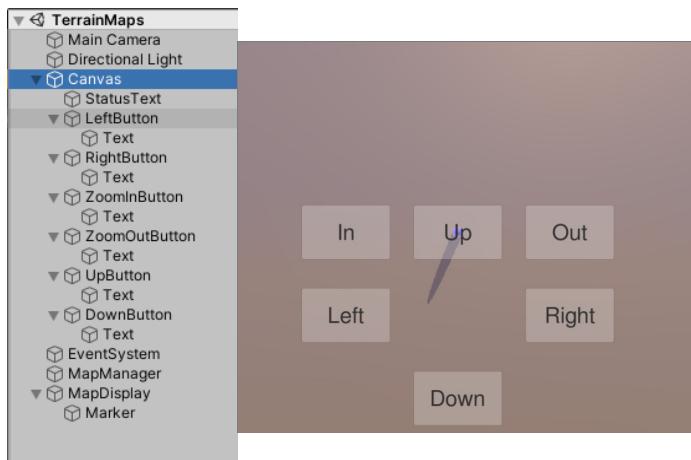


Figure 17.3.1:  
Reconstructing  
the terrain data.

- As before, a tile server provides the tiles. The process for retrieving each tile is identical, except that a different server needs to be used. The mapzen tiles server (<https://www.mapzen.com/blog/terrain-tile-service/>) is one convenient source of elevation data. The request code appears as follows, and replaces that used previously:

```

1 // Elevation tiles, see: https://www.mapzen.com/blog
   /terrain-tile-service/
2 string url = "https://s3.amazonaws.com/elevation-
   tiles-prod/terrarium/" + z + "/" + x + "/" + y +
   ".png";
3 Debug.Log ("Retrieving: " + url);
4 WebRequest www = WebRequest.Create (url);
5 ((HttpWebRequest) www).UserAgent = "TerrainMaps";
6 ...

```

The other difference is that the downloaded texture is not applied to the material (although you can do that

if you want to, it just won't look very recognizable). Instead the red, green and blue channels of the texture contain an encoded version of the altitude of each point.

The adapted updateTexture function thus ends with:

```

1 ...
2 ImageConversion.LoadImage (tex, new BinaryReader (
   response.GetResponseStream ()).ReadBytes
   (1000000));
3 updateMesh (tex, 128, 128, 0.1f);

```

---

The updateMesh function, described in the next step, converts the texture into a geometric mesh.

2. The process of converting altitude texture to geometry is provided in full in Algorithm 17.5.

Code Listing 17.5: MoveWithButtons. Additional functions required to decode the altitude values and produce a mesh.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using UnityEngine.Networking;
6  using System.Net.Security;
7  using System.Security.Cryptography.X509Certificates;
8  using System.Net;
9  using System.IO;
10
11 public class MoveWithButtons : MonoBehaviour
12 {
13
14     public Text statusText;
15
16     public MeshFilter mapObject;
17     public Material mapMaterial;
18
19     public GameObject marker;
20     public GameObject mapPlane;
21
22     private float longitude = 0.0f;
23     private float latitude = 0.0f;
24     private int zoom = 0;
25
26     private static bool TrustCertificate(object sender,
27     X509Certificate x509Certificate, X509Chain x509Chain,
28     SslPolicyErrors sslPolicyErrors)
29     {

```

```
28     // All Certificates are accepted. Not good
29     // practice, but outside scope of this
30     // example.
31     return true;
32 }
33
34 void Start()
35 {
36     ServicePointManager.ServerCertificateValidationCallback =
37     lback =
38     TrustCertificate;
39     updateMapView();
40 }
41
42 private void getTileCoordinates(float longitude, float
43 latitude, int zoom, out int x, out int y)
44 {
45     x = (int)(Mathf.Floor((longitude + 180.0f) /
46 360.0f * Mathf.Pow(2.0f, zoom)));
47     y = (int)(Mathf.Floor((1.0f -
48 Mathf.Log(Mathf.Tan(latitude * Mathf.PI / 180.0f) +
49 1.0f / Mathf.Cos(latitude * Mathf.PI / 180.0f)) /
50 Mathf.PI) / 2.0f * Mathf.Pow(2.0f, zoom)));
51 }
52
53 private void getGeoCoordinates(int x, int y, int zoom,
54 out float longitude, out float latitude)
55 {
56     float n = Mathf.PI - 2.0f * Mathf.PI * y /
57 Mathf.Pow(2.0f, zoom);
58
59     longitude = x / Mathf.Pow(2.0f, zoom) * 360.0f -
60 180.0f;
61     latitude = 180.0f / Mathf.PI * Mathf.Atan(0.5f *
62 (Mathf.Exp(n) - Mathf.Exp(-n)));
63 }
64
65 private float[] convertElevationTexture(Texture2D tex,
66 out float minheight, out float maxheight)
67 {
68     float[] htexdata = new float[tex.width *
69 tex.height];
70     bool setlimits = false;
71     minheight = 0.0f;
72     maxheight = 0.0f;
73
74     for (int y = 0; y < tex.height; y++)
75     {
76         for (int x = 0; x < tex.width; x++)
77         {
78             Color a = tex.GetPixel(x, y);
79             float height = ((a.r * 255.0f * 256.0f) +
80 (a.g * 255.0f) + (a.b * 255.0f / 256.0f)) - 32768.0f;
```

```

67             htexdata[y * tex.width + x] = height;
68
69             if ((!setlimits) || (height < minheight))
70             {
71                 minheight = height;
72             }
73             if ((!setlimits) || (height > maxheight))
74             {
75                 maxheight = height;
76             }
77             setlimits = true;
78         }
79     }
80
81     return htexdata;
82 }
83
84     private void updateMesh(Texture2D tex, int mwidth, int
85     ↪ mheight, float heightrange)
86     {
87         float minheight;
88         float maxheight;
89         float[] heighttex = convertElevationTexture(tex,
90         ↪ out minheight, out maxheight);
91
92         Vector3[] vertices = new Vector3[(mwidth + 1) *
93         ↪ (mheight + 1)];
94         Vector2[] uvs = new Vector2[(mwidth + 1) *
95         ↪ (mheight + 1)];
96         int[] triangles = new int[6 * mwidth * mheight];
97
98         int triangleIndex = 0;
99         for (int y = 0; y < mheight + 1; y++)
100         {
101             for (int x = 0; x < mwidth + 1; x++)
102             {
103                 float xc = (float)x / mwidth;
104                 float zc = (float)y / mheight;
105                 float yc = 0.0f;
106                 //yc = heighttex[(int)(zc * (tex.height -
107                 ↪ 1)) * tex.width + (int)(xc * (tex.width - 1))];
108                 //if (yc < 0.0f) yc = 0.0f;
109                 //yc = heightrange * (yc - minheight) /
110                 ↪ (maxheight - minheight);
111                 vertices[y * (mwidth + 1) + x] = new
112                 Vector3(xc - 0.5f, yc, zc - 0.5f);
113                 uvs[y * (mwidth + 1) + x] = new
114                 Vector2(xc, zc);
115
116                 // Skip the last row/col
117                 if ((x != mwidth) && (y != mheight))
118                 {
119                     int topLeft = y * (mwidth + 1) + x;
120                     int topRight = topLeft + 1;

```

```

112             int bottomLeft = topLeft + mwidth + 1;
113             int bottomRight = bottomLeft + 1;
114
115             triangles[triangleIndex++] = topRight;
116             triangles[triangleIndex++] = topLeft;
117             triangles[triangleIndex++] =
118                 bottomLeft;
119             triangles[triangleIndex++] = topRight;
120             triangles[triangleIndex++] =
121                 bottomLeft;
122             triangles[triangleIndex++] =
123                 bottomRight;
124         }
125     }
126
127     Mesh m = new Mesh();
128     m.vertices = vertices;
129     m.uv = uvs;
130     m.triangles = triangles;
131     m.RecalculateNormals();
132     mapObject.mesh = m;
133 }
134
135 private void updateTexture(int x, int y, int z)
136 {
137     // Elevation tiles, see:
138     // https://www.mapzen.com/blog/terrain-tile-service/
139     string url = "https://s3.amazonaws.com/elevation-t_
140     iles-prod/terrarium/" + z + "/" + x + "/" + y +
141     ".png";
142     Debug.Log("Retrieving: " + url);
143     WebRequest www = WebRequest.Create(url);
144     ((HttpWebRequest)www).UserAgent = "TerrainMaps";
145
146     var response = www.GetResponse();
147
148     Texture2D tex = new Texture2D(2, 2);
149     // Retrieve a large number of bytes - should be
150     // more than in a tile texture.
151     ImageConversion.LoadImage(tex, new BinaryReader(re_
152     sponse.GetResponseStream()).ReadBytes(1000000));
153     updateMesh(tex, 128, 128, 0.1f);
154     mapMaterial.mainTexture = tex;
155 }
```

**private void updateMapView()**

```

156 {
157     int x;
158     int y;
159     getTileCoordinates(longitude, latitude, zoom, out
160     x, out y);
161     updateTexture(x, y, zoom);
```

```
156
157         // Place a marker at the current position on the
158         // tile.
159         float cornerLatA;
160         float cornerLongA;
161         float cornerLatB;
162         float cornerLongB;
163         getGeoCoordinates(x, y, zoom, out cornerLongA, out
164         cornerLatA);
165         getGeoCoordinates(x + 1, y + 1, zoom, out
166         cornerLongB, out cornerLatB);
167         // interpolate current coordinates relative to the
168         // coordinates of the
169         // tile corners. Assumes the plane coordinates run
170         // from (-5,-5) to (5,5).
171         float r = 10.0f * (((longitude - cornerLongA) /
172         (cornerLongB - cornerLongA))) + 5.0f;
173         float d = 10.0f * (((latitude - cornerLatA) /
174         (cornerLatB - cornerLatA))) + 5.0f;
175         marker.transform.position =
176         mapPlane.transform.position -
177         mapPlane.transform.forward * d +
178         mapPlane.transform.right * r;
179
180         statusText.text = longitude + "," + latitude + "("
181         + zoom + ")" + "[" + x + "," + y + "]";
182     }
183
184     public void onButtonEvent(float dx, float dy, int dz)
185     {
186         zoom += dz;
187         // Calculate step so that it takes a few button
188         // presses
189         // to move across a tile at that level.
190         float step = 1.0f * 1.0f / Mathf.Pow(2.0f, zoom);
191         longitude += 360.0f * dx * step;
192         latitude += 180.0f * dy * step;
193
194         updateMapView();
195     }
196     public void leftButton() { onButtonEvent(-1.0f, 0.0f,
197         0); }
198     public void rightButton() { onButtonEvent(1.0f, 0.0f,
199         0); }
200     public void upButton() { onButtonEvent(0.0f, 1.0f, 0);
201     }
202     public void downButton() { onButtonEvent(0.0f, -1.0f,
203         0); }
204     public void inButton() { onButtonEvent(0.0f, 0.0f, 1);
205     }
206     public void outButton() { onButtonEvent(0.0f, 0.0f,
207         -1); }
```

191

}

In summary, however, there are two key steps:

- (a) The altitude values need to be decoded. The `convertElevationTexture` function maps the array of colour values received from the server to an equivalent array of altitude values represented as floating point values. The formula for converting to a value in meters is:  $red * 256 + green + blue / 256$ . This needs to be scaled by a factor of 255 since the Unity software represents colour using the range  $[0, 1]$  rather than the range of  $[0, 255]$  used by most byte formatted textures. An offset of -32768 is applied since elevation values on Earth include negative numbers for areas that are underwater.

Minimum and maximum height values for the region represented by the texture are also recorded. This allows the resulting mesh to be scaled so that the peaks and valleys are all visible within the camera frustum.

- (b) The height values need to be turned into a geometric mesh. The steps for this go as follows:

- i. Create a grid of vertices. The size of this is independent of the number of texels in the texture since the Unity software has a limit on the number of vertices that can exist in a single mesh (and the default tile size is on the limit of this). The x and z coordinates of the vertex are determined by the position of the vertex in the grid.
- ii. Connect neighbouring vertices together to form triangles, with each grid square produced from two triangles. Triangles are defined by using the index of each corner vertex (the index in the list of vertices) to describe the triangle.
- iii. Set the y coordinate of each vertex (the 'height') using the value retrieved from the corresponding

position in the altitude texture, after it has been decoded.

- iv. The uv values specify how any texture is mapped to vertices on the grid. This comes in useful if you assign a map texture to the mesh.

The mesh then replaces the mesh in one of the geometric objects in the scene. In this case, the MapDisplay object is a suitable candidate.

Replace the mapMaterial field with a mapObject, since we're going to be updating the mesh rather than a material.

---

```
1 public MeshFilter mapObject;
```

---

Drag the MapDisplay object into the Map Object property of the MapManager. You may also need to resize the MapDisplay object to fit comfortably onto the screen once the map geometry is produced.

The continental structures can be quite hard to identify in the raw altitude data, particularly when most of the planet is quite low compared to the extreme points. Some strategies to make regions easier to identify include:

- Turn down the lighting, so that glare does not obscure subtle surface details.
- Remove the area that is underwater (if you don't admire the amazing detail there). Include the following line before the height value is normalized.

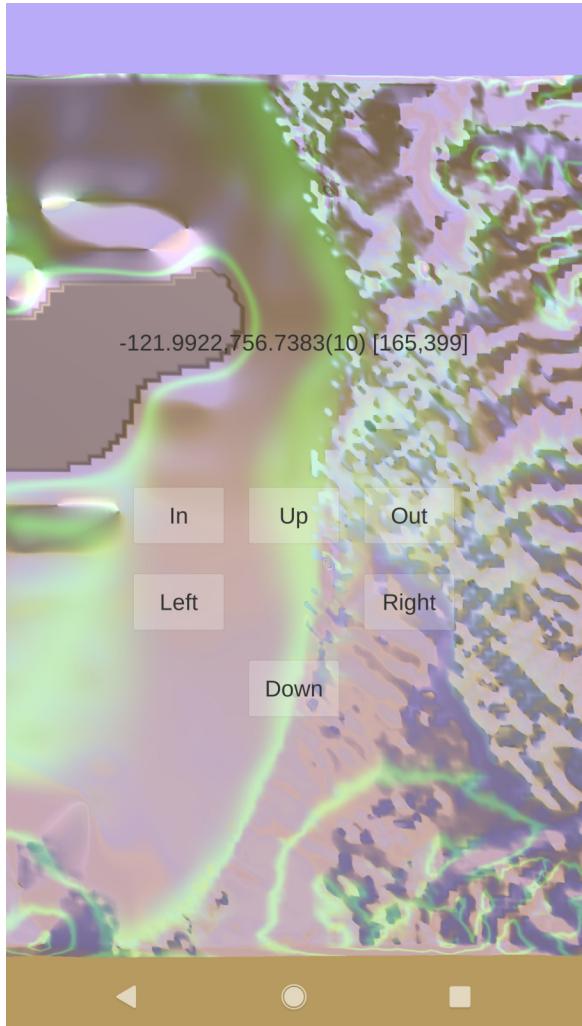
---

```
1 if (yc < 0.0f) yc = 0.0f;
```

---

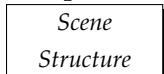
- Follow along using the original application to download the texture for the same region. Matching the two images when they show the same coordinates can help find your bearings.

Figure 17.3.2: Mesh containing the 3D representation of the terrain at any point on the surface of the planet.



## 17.4 Mapping your physical location as a virtual point cloud

### Component 17.4.1: Scene tracking with AR Core.



*Note:* This component is in the process of being replaced with another version, under the name Room Scanner. This version addresses some of the flaws in this application, particularly those related to managing the complexity of the scenes resulting from the scanning process. This component is still a useful example to demonstrate some of the aspects of capturing a setting.

The ability of a device to sense feature points can be employed to map out a representation of the environment. Given the position of a feature point relative to the camera (components 19.1.1 and 19.2.1) and then knowing the position of the camera relative to the world (component 18.2.1) means that each feature point can be located in the world coordinate system.

Collections of isolated feature points are known as point clouds and can be visualized in exactly that format; as a nebulous collection of pixel sized points. Sufficiently dense point clouds provide sufficient structure to present a reasonably cohesive and contiguous view of the surrounding space. This can be improved by splatting each point to fill in the space between it and its neighbours. Surface meshes can also be fitted to point clouds to estimate the boundaries that they represent.

Another approach is to treat each point as a sample of the volume that it occupies; a volume element (or voxel). These might be approximated by individual cubic objects, sized to fit into the space between neighbouring voxels.

Neighbourhood structure is something that must be reconstructed from the raw position information of the individual points. Naively, finding a neighbouring point would involve searching through all the other points to

find the ones within a threshold distance of our target point. This step would get very time consuming, particularly as our structure expanded to include millions or billions of points.

The solution is to organize each point as it is added in a way that reflects the spatial relationships between points. An initial point might suggest that the entire room consists of a single enormous voxel. The second point indicates that the space must be shared by two neighbour voxels. The initial volume would be divided into two smaller volumes in such a way that the two points each exist in their own volume of space.

The OctTree structure used in this example assumes that all volumes are perfect cubes. When a voxel is subdivided, it is split into eight smaller voxels (some unoccupied), and this process is repeated for the occupied voxels until each point controls a single (sub)voxel. In practice there may be some limits placed on this process, such as minimum voxel sizes. Multiple feature points may end up being averaged together into a single voxel to account for inaccuracies in the sensing process.

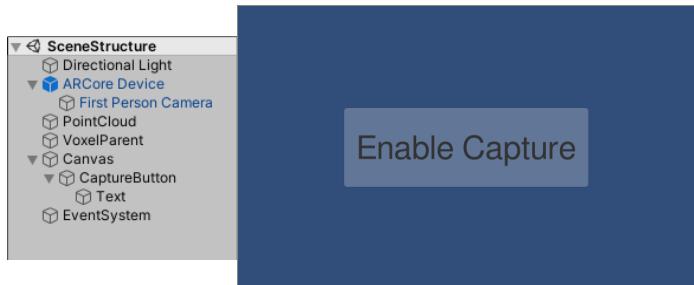
Feature points are one form of spatial information and indicate the presence of a surface point. The position of the camera is another piece of information that confirms the absence of a surface at the point. Thus the mapping process can confirm both occupied and unoccupied regions. The occupied areas have additional properties; the colour of the feature point can be recorded to help add richness to the resulting voxel visualization.

This process applies equally well to other sources of feature points. Feature points sourced only from camera images are dependent on sufficient visual variety in the scene to provide a dense map. Other forms of depth camera may add their own features by illuminating particular points with a laser, or using reflection of light or sound to provide an estimate of distance to the point.

This example demonstrates the construction of an OctTree structure from a set of feature points. Colour

of individual volume elements is set from the colour of the corresponding point on the camera image. The voxels are visualized as cubes, and could potentially be used to provide an occlusion solution for an augmented reality application.

Figure 17.4.1:  
Feature tracking to capture the setting.



1. Start with a new project. This needs to be set up to use AR Core. Set up the usual properties to deploy to a mobile platform. Once the platform is set, ensure that the ARCore Supported option is set under the XR Settings of the Player Settings. Download the AR Core package for the Unity software and add it to the project.
2. Remove the main camera, and replace it with an ARCore Device prefab. The default configuration can be used but it may be helpful to switch the camera focus mode to Auto.
3. Create a new empty object called PointCloud and include it in the scene. Create a new C# script called PointClouder and add it to this object. Prepare the following elements which are needed by this component:
  - (a) A cube derived prefab to represent each voxel.  
This can be created using the standard cube object provided by the Unity software.
  - (b) An object to be the parent for all the voxels that are instantiated. These instances may need to be cleaned up, and this is easier to do if they are located under a common parent dedicated to the purpose. A new empty called VoxelParent will serve this purpose.

- (c) The object in the scene representing the camera.

This is the first person camera object which is the child of the AR Core Device. The position and orientation of this object are required to map the position of a point in the world back to a position in the camera image so that the colour of that point in the image can be matched to the voxel representing the point.

- (d) The background renderer component from the AR camera. If this is provided, then it is possible to disable the display of the camera image. This helps view the point cloud directly on its own. This is selected if the first person camera object is dragged and dropped onto this field.
- (e) The AR Core Session object allows properties of the camera, such as the resolution, to be set. This is a component that is part of the ARCore Device, and can be provided as the AR Session Manager property.

A button on the user interface is used to toggle between capturing points in the scene, and displaying the resulting voxel structure. Create this button, and connect its OnClick function to the toggleCapture method of the PointClouder component. The text element of this button is provided to the PointClouder component, so that the label on the button can be updated as it is toggled.

The core process behind this component is provided in the Update method. When AR Core provides a new set of the feature points seen in the current frame, these points are appended to the Oct Tree. A colour value is associated with each point by taking the image from the camera, mapping the position of the point to a pixel in that image, and retrieving the colour from that pixel.

The camera mode used by AR Core captures images in the YUV colour space. Colours are still described with three coordinates but the Y coordinate coordinates

represents brightness while U and V represent ratios of red to green, and blue to green respectively. Often the colour channels (U and V) are provided at a lower resolution to the luminance channel (Y) as the eye has a relatively low sensitivity to resolution of colour changes.

The code for the PointClouder component is provided in Algorithm 17.6.

Code Listing 17.6: PointClouder. Point clouds are presented by capturing the position of feature points, associating a colour value with them, and adding them to a spatial data structure.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using GoogleARCore;
6  using System;
7  using System.Runtime.InteropServices;

8
9  public class PointClouder : MonoBehaviour {
10
11    [Tooltip ("An output string for debugging.")]
12    public TextMesh logger;
13
14    [Tooltip ("Shape to place around each target point.")]
15    public GameObject nodeShape;
16
17    [Tooltip ("An object to collect all voxel game objects
↳ under.")]
18    public GameObject voxelParent;
19
20    [Tooltip ("The label on the capture button - changes to
↳ enable/disable.")]
21    public Text captureMessage;
22
23    [Tooltip ("The camera, source of colours for the marker
↳ points.")]
24    public Camera arCamera;
25
26    [Tooltip ("Switch off view of the background to see the
↳ octtree more easily.")]
27    public ARCoreBackgroundRenderer background;
28
29    [Tooltip ("Session manager allows choice of camera
↳ configurations.")]
30    public ARCoreSession ARSessionManager;
31
32    // A tree for progressively mapping out the scene.

```

```
33
34     private OctTree tree;
35
36     // Only add points when true.
37     private bool addVoxels = false;
38
39     void Start () {
40         tree = new OctTree (nodeShape);
41
42         ARSessionManager.RegisterChooseCameraConfigurationCall
43         ↪ back
44         ↪ (chooseCameraConfiguration);
45         ARSessionManager.enabled = true;
46     }
47
48     private int chooseCameraConfiguration(List<CameraConfig>
49         ↪ supportedConfigurations)
50     {
51         // Pick the lowest grade camera.
52         return 0;
53     }
54
55     // Used by external controls to switch the mode of this
56     ↪ component
57     // between capturing and adding to the octtree, or to
58     ↪ just
59     // displaying the octtree as it stands.
60     public void toggleCapture ()
61     {
62         addVoxels = !addVoxels;
63         if (background != null)
64         {
65             background.enabled = addVoxels;
66         }
67         if (addVoxels)
68         {
69             captureMessage.text = "Disable capture";
70         }
71         else
72         {
73             captureMessage.text = "Enable capture";
74         }
75     }
76
77     // Derived from:
78     ↪ https://github.com/DavidSM64/N64-YUV2RGB-Library
79     private Color YUV2RGB (byte Y, byte U, byte V)
80     {
81         return new Color (Mathf.Clamp ((1.164f * (Y - 16) +
82             1.596f * (V - 128)) / 255.0f, 0.0f, 1.0f),
83                         Mathf.Clamp ((1.164f * (Y - 16) -
84                 0.813f * (V - 128) - 0.391f * (U - 128)) / 255.0f,
85                 0.0f, 1.0f));
86     }
```

```

76             Mathf.Clamp ((1.164f * (Y - 16) +
77     ↳ 2.018f * (U - 128)) / 255.0f, 0.0f, 1.0f));
78 }
79
80 // Find the colour at the given coordinates. The camera
81 // requires some care. It uses the YUV colour space so
82 // colour values
83 // need to be converted from this space to RGB for
84 // display. The Y plane
85 // uses twice the resolution of the other two, so
86 // calculating offsets to
87 // particular pixel resolutions needs to take this into
88 // account.
89 private Color getColourAt (CameraImageBytes cim, int cx,
90     int cy, out bool foundColour)
91 {
92     Color colour = new Color (0, 0, 0);
93     foundColour = false;
94
95     if ((cx >= 0) && (cy >= 0) && (cx < cim.Width) && (cy
96     < cim.Height))
97     {
98         byte [] buf = new byte [1];
99         byte y;
100        byte u;
101        byte v;
102
103        Marshal.Copy (new IntPtr (cim.Y.ToInt64 ()) + cy *
104            cim.YRowStride + cx * 1), buf, 0, 1);
105        y = buf[0];
106        // Watch out - the UV plane is a quarter of the
107        // resolution.
108        Marshal.Copy (new IntPtr (cim.U.ToInt64 ()) + (cy /
109            2) * cim.UVRowStride + (cx / 2) * cim.UVPixelStride),
110            buf, 0, 1);
111        u = buf[0];
112        Marshal.Copy (new IntPtr (cim.V.ToInt64 ()) + (cy /
113            2) * cim.UVRowStride + (cx / 2) * cim.UVPixelStride),
114            buf, 0, 1);
115        v = buf[0];
116        colour = YUV2RGB (y, u, v);
117        foundColour = true;
118    }
119
120    return colour;
121 }
122
123 void Update () {
124     if (Frame.PointCloud.IsUpdatedThisFrame)
125     {
126         if (logger != null)
127         {
128             logger.text = "Have Points";
129         }
130     }
131 }
```

```

115     }
116     if (addVoxels)
117     {
118         // Take a snapshot of the camera image at this
119         // time.
120         CameraImageBytes cim =
121         Frame.CameraImage.AcquireCameraImageBytes();
122
123         // Copy the point cloud points for mesh vertices.
124         for (int i = 0; i < Frame.PointCloud.PointCount;
125             i++)
126         {
127             Color colour = new Color (0, 0, 0);
128             bool foundColour = false;
129
130             // Get a point.
131             PointCloudPoint p =
132             Frame.PointCloud.GetPointAsStruct (i);
133
134             // Work out where in the camera image this point
135             // would be.
136             Vector3 cameraCoordinates =
137             arCamera.WorldToViewportPoint (p);
138
139             // Get the colour from the image, corresponding
140             // to this point.
141             if (cim.IsAvailable)
142             {
143                 var uvQuad = Frame.CameraImage.DisplayUvCoords;
144                 // FIXME : take frame display into account -
145                 // see uvQuad. This should involve just
146                 // interpolation, as done in the background
147                 // shader.
148                 int cx = (int) (cameraCoordinates.x *
149                 cim.Width);
150                 int cy = (int) ((1.0f - cameraCoordinates.y) *
151                 cim.Height);
152                 colour = getColourAt (cim, cx, cy, out
153                 foundColour);
154             }
155
156             // Add the point to the Oct Tree.
157             if (foundColour)
158             {
159                 tree.addPoint (p, colour);
160             }
161         }
162
163         cim.Release ();
164     }
165     // Update the scene description of the Oct Tree.
166     tree.renderOctTree (voxelParent);
167 }
```

```

156     else
157     {
158         if (logger != null)
159         {
160             logger.text = "No Points";
161         }
162     }
163 }
164 }
```

4. The PointClouder component makes use of another class that implements the Oct Tree. This class is shown in Algorithm 17.7. This particular implementation allows the tree to grow, to expand to cover any volume that is defined. When new elements are added to the tree they are initially added at the root level. If they do not fit (the tree does not currently encompass the new point) then larger root nodes are added above it. Once the tree is large enough to fit the point then the element is bubbled downwards until it reaches a leaf node.

Code Listing 17.7: OctTree. The Oct Tree class maintains a spatial data structure representing the relationship between points in the cloud.

```

1  using UnityEngine;
2  using System.Collections.Generic;
3  using GoogleARCore;
4
5  class OctTree
6  {
7      // The threshold level of leaf division. Ensure that the
8      // tree
9      // depth does not grow completely out of control.
10     protected float OctTreeLeafSize = 0.01f;
11
12     // The maximum number of elements stored per
13     // Oct Tree node.
14     protected int maxPointPerVoxel = 5;
15
16     // The object used to represent a single voxel.
17     protected GameObject nodeShape;
18
19     // The data stored in a node in an Oct Tree.
20     protected class OctTreeElement
21     {
22         // The coordinates of the feature point.
23         public PointCloudPoint point;
24         // The colour of the feature point.
25     }
```

```

24     public Color colour;
25     // A handy reference to the Oct Tree node
26     // containing this element.
27     public OctTreeNode containedIn;
28
29     public OctTreeElement (PointCloudPoint p, Color c)
30     {
31         point = p;
32         colour = c;
33         containedIn = null;
34     }
35 }
36
37 // An Oct Tree consists of a link to root node.
38 // Each node contains up to 8 links to child nodes
39 // representing subsets of the space belonging
40 // to the parent node.
41 protected class OctTreeNode
42 {
43     // The coordinates of the space represented by
44     // the node.
45     public Vector3 minCorner;
46     public Vector3 maxCorner;
47
48     // A reference to this node's parent node.
49     public OctTreeNode parent;
50     // References to the 8 child nodes.
51     public OctTreeNode [] children = new OctTreeNode [8];
52
53     // The data stored in this node.
54     public List <OctTreeElement> containedEntities;
55
56     // number of non-null children.
57     public int leafcount;
58
59     // The shape of this node.
60     public GameObject voxel;
61
62     // Convert a set of left/right, above/below,
63     ← front/back flags
64     // into the number of the child element (from 0 to 7).
65     public static int encode (bool x, bool y, bool z)
66     {
67         int r = 0;
68         if (x) { r += 4; }
69         if (y) { r += 2; }
70         if (z) { r += 1; }
71         return r;
72     }
73
74     public OctTreeNode(Vector3 minc, Vector3 maxc,
75     ← OctTreeNode parnt) {
76         minCorner = minc;

```

```

75         maxCorner = maxc;
76         parent = parnt;
77
78         for (int i = 0; i < 8; i++)
79         {
80             children[i] = null;
81         }
82
83         leafcount = 0;
84         containedEntities = new List <OctTreeElement> ();
85         voxel = null;
86     }
87 }
88
89 // The Oct Tree - reference to the root node.
90 protected OctTreeNode octtreeroot;
91
92 public OctTree (GameObject n)
93 {
94     nodeShape = n;
95     octtreeroot = null;
96 }
97
98 // Add a new feature point to the Oct Tree.
99 public void addPoint (PointCloudPoint point, Color c)
100 {
101     placeNodeInTree (new OctTreeElement (point, c));
102 }
103
104 // Take an element out of the Oct Tree.
105 private void removeFromContainment (OctTreeElement o)
106 {
107     // remove o from current containment.
108     if (o.containedIn != null)
109     {
110         if (!o.containedIn.containedEntities.Remove (o))
111         {
112             Debug.Log ("Oct Tree Element not found in
113             container");
114         }
115     }
116
117     // Move an element to a destination Oct Tree node,
118     // taking care of housekeeping
119     private void changeContainment (OctTreeElement o,
120     OctTreeNode dest)
121     {
122         removeFromContainment (o);
123
124         // add to dest.
125         o.containedIn = dest;

```

```

126     if (dest != null)
127     {
128         while (o.containedIn.containedEntities.Count >
129             maxPointPerVoxel)
130         {
131             o.containedIn.containedEntities.RemoveAt (0);
132         }
133
134         // update list
135         o.containedIn.containedEntities.Add (o);
136     }
137
138     // Place an element in the tree. This tree has the
139     // ability to grow
140     // to encompass all points added, even if it needs to
141     // add new root
142     // nodes.
143     protected void placeNodeInTree (OctTreeElement o)
144     {
145         if (o.containedIn == null)
146         {
147             // not in the tree. Place it in the root, then
148             // shuffle.
149             if (octtreeroot == null)
150             {
151                 float size = OctTreeLeafSize;
152                 // no tree yet either.
153                 octtreeroot = new OctTreeNode (new Vector3
154                     (o.point.Position.x - size / 2.0f, o.point.Position.y
155                     - size / 2.0f, o.point.Position.z - size / 2.0f), new
156                     Vector3 (o.point.Position.x + size / 2.0f,
157                     o.point.Position.y + size / 2.0f, o.point.Position.z +
158                     size / 2.0f), null);
159             }
160
161             changeContainment (o, octtreeroot);
162         }
163
164         // Check if node fits.
165         // move upwards if does not fit.
166         while ((o.point.Position.x <=
167             o.containedIn.minCorner.x) ||
168                 (o.point.Position.y <= o.containedIn.minCorner.y) ||
169                 (o.point.Position.z <= o.containedIn.minCorner.z) ||
170                 (o.point.Position.x > o.containedIn.maxCorner.x) ||
171                 (o.point.Position.y > o.containedIn.maxCorner.y) ||
172                 (o.point.Position.z > o.containedIn.maxCorner.z))
173         {
174             // does not fit - move up.
175             if (o.containedIn.parent == null)
176             {
177                 // need to grow the tree upwards.

```

```

169      // make sure we expand in a direction that works.
170      bool crossx = o.point.Position.x <=
171      ↵ o.containedIn.minCorner.x;
172      bool crossy = o.point.Position.y <=
173      ↵ o.containedIn.minCorner.y;
174      bool crossz = o.point.Position.z <=
175      ↵ o.containedIn.minCorner.z;

176      float size = (o.containedIn.maxCorner.x -
177      ↵ o.containedIn.minCorner.x);
178      Vector3 pmin = new Vector3 (crossx ?
179      ↵ o.containedIn.minCorner.x - size :
180      ↵ o.containedIn.minCorner.x,
181      ↵ crossy ? o.containedIn.minCorner.y - size :
182      ↵ o.containedIn.minCorner.y,
183      ↵ crossz ? o.containedIn.minCorner.z - size :
184      ↵ o.containedIn.minCorner.z);
185      Vector3 pmax = new Vector3 (crossx ?
186      ↵ o.containedIn.maxCorner.x :
187      ↵ o.containedIn.maxCorner.x + size,
188      ↵ crossy ? o.containedIn.maxCorner.y :
189      ↵ o.containedIn.maxCorner.y + size,
190      ↵ crossz ? o.containedIn.maxCorner.z :
191      ↵ o.containedIn.maxCorner.z + size);
192      OctTreeNode newoct = new OctTreeNode (pmin, pmax,
193      ↵ null);
194      newoct.children[OctTreeNode.encode (crossx,
195      ↵ crossy, crossz)] = octtreeroot;
196      newoct.leafcount++;
197      octtreeroot.parent = newoct;
198      octtreeroot = newoct;
199      }

200      changeContainment (o, o.containedIn.parent);

201      // prune empty leaves.
202      for (int i = 0; i < 8; i++)
203      {
204          if (o.containedIn.children[i] != null &&
205          ↵ o.containedIn.children[i].leafcount == 0 &&
206          ↵ o.containedIn.children[i].containedEntities.Count == 0)
207          {
208              o.containedIn.children[i] = null;
209              o.containedIn.leafcount--;
210          }
211      }

212      // now move downwards if possible.
213      while (o.containedIn.maxCorner.x -
214      ↵ o.containedIn.minCorner.x > OctTreeLeafSize)
215      {

```

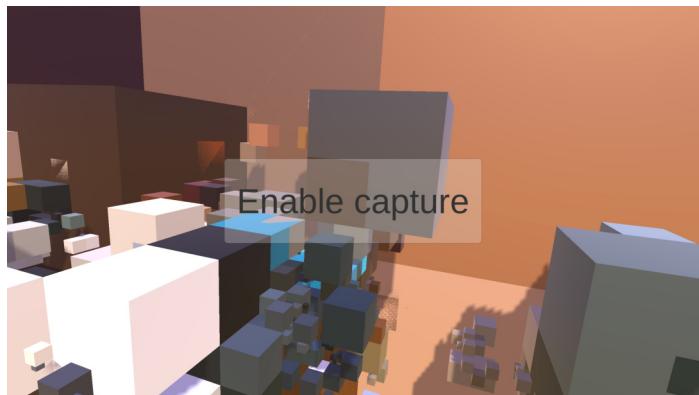
```

206     if ((o.containedIn.leafcount == 0) &&
207     (o.containedIn.containedEntities.Count <= 1))
208         break;
209
210     Vector3 mid = 0.5f * (o.containedIn.maxCorner +
211     o.containedIn.minCorner);
212
213     bool crossx = o.point.Position.x <= mid.x;
214     bool crossy = o.point.Position.y <= mid.y;
215     bool crossz = o.point.Position.z <= mid.z;
216     int cindex = OctTreeNode.encode (crossx, crossy,
217     crossz);
218     Vector3 cmin = new Vector3 (crossx ?
219     o.containedIn.minCorner.x : mid.x,
220                           crossy ?
221     o.containedIn.minCorner.y : mid.y,
222                           crossz ?
223     o.containedIn.minCorner.z : mid.z);
224     Vector3 cmax = new Vector3 (crossx ? mid.x :
225     o.containedIn.maxCorner.x,
226                           crossy ? mid.y :
227     o.containedIn.maxCorner.y,
228                           crossz ? mid.z :
229     o.containedIn.maxCorner.z);
230     if (o.containedIn.children[cindex] == null)
231     {
232         o.containedIn.children[cindex] = new OctTreeNode
233         (cmin, cmax, o.containedIn);
234         o.containedIn.leafcount++;
235     }
236     changeContainment (o,
237     o.containedIn.children[cindex]);
238 }
239
240 // Shrink tree from top if possible
241 while (octtreeroot != null && octtreeroot.leafcount <=
242 1 && octtreeroot.containedEntities.Count == 0)
243 {
244     for (int i = 0; i < 8; i++)
245     {
246         // no entities at this level, and at most 1 child
247         // occupied.
248         if (octtreeroot.children[i] != null)
249         {
250             octtreeroot = octtreeroot.children[i];
251             break;
252         }
253     }
254 }
255
256 // Update the scene representation of the Oct Tree.
257 public void renderOctTree (GameObject parent)

```

```
246 {
247     // Start at the root, and recurse.
248     renderOctTreeNode (octtreeroot, parent);
249 }
250
251 // Ensure that a node in the tree has a representation
252 // in the scene. Only leaf nodes are shown.
253 protected void renderOctTreeNode(OctTreeNode root,
254     GameObject parent)
255 {
256     if (root != null)
257     {
258         if (root.leafcount == 0)
259         {
260             if (root.voxel == null)
261             {
262                 root.voxel = UnityEngine.Object.Instantiate
263                     (nodeShape);
264
265                 Color c = new Color (0, 0, 0);
266                 foreach (OctTreeElement e in
267                     root.containedEntities)
268                 {
269                     c += e.colour;
270                 }
271                 c = (1.0f / root.containedEntities.Count) * c;
272                 root.voxel.transform.position =
273                     (root.minCorner + root.maxCorner);
274                 root.voxel.transform.localScale =
275                     root.maxCorner -
276                     root.minCorner;
277                 root.voxel.transform.SetParent (parent.transform);
278                 root.voxel.GetComponent <MeshRenderer>
279                     ().material.color = c;
280             }
281             else
282             {
283                 if (root.voxel != null)
284                 {
285                     GameObject.Destroy (root.voxel);
286                     root.voxel = null;
287                 }
288                 for (int i = 0; i < 8; i++)
289                 {
290                     renderOctTreeNode (root.children[i], parent);
291                 }
292             }
293         }
294     }
295 }
```

Figure 17.4.2:  
Tracked feature  
points used to  
represent objects in  
the location.



## 17.5 Capturing your environment as a 3D mesh with the Kinect

### Component 17.5.1: Scene capture with Kinect. Kinect Scan

*Depth cameras provide the ability to scan the physical space and capture a representation of the layout of surfaces so that virtual and physical content can mingle realistically. The facilities in the Kinect depth camera capture two relevant pieces of information:*

1. *A point cloud consists of all the surface points mapped, providing the three coordinates with the position of each point. In the case of a depth camera these can often be arranged in a grid, providing some insight into the relationship between neighbouring points (i.e. how they connect to form a surface).*
2. *An image from the colour camera. This provides either a colour value that can be associated with every point in the cloud, or a texture image that can be mapped to a surface reconstructed from the point cloud.*

*This example constructs a 3D mesh representing the view of the scene from the point of view of the depth camera. It is based on the DepthSourceView component provided by the Kinect for Unity software but includes the ability to save snapshots representing the geometry and texture so that they can be imported*

*into external modelling packages. This potentially allows further manipulation of the individual meshes such as aligning and welding them into a coherent view of the entire scene.*

Figure 17.5.1:  
Scanning with the  
Kinect.



1. Start with a new project. As with other examples (component 19.5.1) using the Kinect, the Kinect for Windows 2.0 SDK needs to be installed. This is available from: <http://www.microsoft.com/en-us/download/details.aspx?id=44561>. Also download the Unity software package for the Kinect from: <http://go.microsoft.com/fwlink/?LinkID=513177> and unpack this archive in a convenient location.

Import the Kinect.2.0.1410.19000.unitypackage (Assets/Import package menu option) from the unpacked archive. This adds two folders to the project assets: Plugins and Standard Assets.

Manually copy two files from the KinectView/Scripts folder in the unpacked archive into the Assets area of the project. The files required are ColorSourceManager.cs and DepthSourceManager.cs. These provide access to the colour texture image and depth point cloud respectively.

2. This component visualizes the information from these

two managers in the scene. When a particular key is pressed then the mesh and its material is saved to a file.

The scene requires an object to manage this process, and also to display the information from the depth camera. Create a new 3D such as a capsule or cube for this purpose. Note that this must be an object that has some geometry since the MeshFilter and MeshRenderer components are required to represent the mesh created from the Kinect data, and to display it.

Create a new C# script called ObjectScan and add the code shown in Algorithm 17.8..

Code Listing 17.8: ObjectScan. Data captured from the depth camera is converted into a textured mesh, and then exported to an external file when required.

```
1  using UnityEngine;
2  using System.Collections;
3  using Windows.Kinect;
4  using System.IO;
5  using UnityEditor;
6
7  public class ObjectScan : MonoBehaviour
8  {
9      private KinectSensor sensor;
10     private CoordinateMapper mapper;
11
12     private Mesh mesh;
13     private Vector3[] vertices;
14     private Vector2[] uv;
15     private int[] triangles;
16
17     // Size of Unity mesh is limited, so down sample to
18     // fit this limit.
19     private const int _DownsampleSize = 2;
20     private const double _DepthScale = 0.12f;
21
22     public ColorSourceManager colorManager;
23     public DepthSourceManager depthManager;
24
25     private float timeDelay = -1.0f;
26     private bool timerStarted = false;
27
28     public float timeBeforeCapture = 0.1f;
29
30     void Start()
31     {
32         sensor = KinectSensor.GetDefault();
33         if (sensor != null)
```

```

33     {
34         mapper = sensor.CoordinateMapper;
35         var frameDesc =
36             ↪ sensor.DepthFrameSource.FrameDescription;
37
38         CreateMesh(frameDesc.Width / _DownsampleSize,
39             ↪ frameDesc.Height / _DownsampleSize);
40
41         if (sensor.IsOpen)
42         {
43             sensor.Open();
44         }
45     }
46
47     void CreateMesh(int width, int height)
48     {
49         mesh = new Mesh();
50         GetComponent<MeshFilter>().mesh = mesh;
51
52         vertices = new Vector3[width * height];
53         uv = new Vector2[width * height];
54         triangles = new int[6 * ((width - 1) * (height -
55             ↪ 1))];
56
57         int triangleIndex = 0;
58         for (int y = 0; y < height; y++)
59         {
60             for (int x = 0; x < width; x++)
61             {
62                 int index = (y * width) + x;
63
64                 vertices[index] = new Vector3(x, -y, 0);
65                 uv[index] = new Vector2((float)x /
66                     ↪ (float)width), ((float)y / (float)height));
67
68                 // Skip the last row/col
69                 if (x != (width - 1) && y != (height - 1))
70                 {
71                     int topLeft = index;
72                     int topRight = topLeft + 1;
73                     int bottomLeft = topLeft + width;
74                     int bottomRight = bottomLeft + 1;
75
76                     triangles[triangleIndex++] = topLeft;
77                     triangles[triangleIndex++] = topRight;
78                     triangles[triangleIndex++] =
79                     ↪ bottomLeft;
80                     triangles[triangleIndex++] =
81                     ↪ bottomLeft;
82                     triangles[triangleIndex++] = topRight;
83                     triangles[triangleIndex++] =
84                     ↪ bottomRight;

```

```

79             }
80         }
81     }
82
83     mesh.vertices = vertices;
84     mesh.uv = uv;
85     mesh.triangles = triangles;
86     mesh.RecalculateNormals();
87 }
88
89 void Update()
90 {
91     if (sensor == null)
92     {
93         return;
94     }
95     game0bject.GetComponent<MeshRenderer>
96     → ().material.mainTexture =
97     → colorManager.GetColorTexture();
98     RefreshData(depthManager.GetData(),
99     → colorManager.ColorWidth, colorManager.ColorHeight);
100
101     if (Input.GetAxis("Fire1") > 0.0f)
102     {
103         timeDelay = timeBeforeCapture;
104         timerStarted = true;
105     }
106     if (timeDelay > 0.0f)
107     {
108         timeDelay -= Time.deltaTime;
109     }
110
111     if (timerStarted && (timeDelay < 0.0f))
112     {
113         timerStarted = false;
114
115         // For continuous capture. Recommend
116         → timeBeforeCapture of at least 1 s.
117         //timeDelay = timeBeforeCapture;
118         //timerStarted = true;
119
120         int count = 0;
121         string fn = "";
122         do
123         {
124             fn = "scan" + count.ToString("D4");
125             count++;
126         }
127         while (File.Exists(fn + ".obj"));
128         Debug.Log("Exporting: " + fn);
129         exportObj(fn);
130         EditorApplication.Beep();
131     }
132 }
```

```

128     }
129
130     private void RefreshData(ushort[] depthData, int
131     colorWidth, int colorHeight)
132     {
133         var frameDesc =
134             sensor.DepthFrameSource.FrameDescription;
135
136         ColorSpacePoint[] colorSpace = new
137             ColorSpacePoint[depthData.Length];
138         mapper.MapDepthFrameToColorSpace(depthData,
139             colorSpace);
140
141         for (int y = 0; y < frameDesc.Height -
142             (_DownsampleSize - 1); y += _DownsampleSize)
143         {
144             for (int x = 0; x < frameDesc.Width -
145                 (_DownsampleSize - 1); x += _DownsampleSize)
146             {
147                 int indexX = x / _DownsampleSize;
148                 int indexY = y / _DownsampleSize;
149                 int width = frameDesc.Width /
150                     _DownsampleSize;
151                 int smallIndex = indexY * width + indexX;
152
153                 double avg = GetAvg(depthData, x, y,
154                     frameDesc.Width, frameDesc.Height);
155
156                 avg = avg * _DepthScale;
157
158                 vertices[smallIndex].z = (float)avg;
159
160                 // Update UV mapping with CDRP
161                 var colorSpacePoint = colorSpace[(y *
162                     frameDesc.Width) + x];
163                 uv[smallIndex] = new
164                     Vector2(colorSpacePoint.X / colorWidth,
165                         colorSpacePoint.Y / colorHeight);
166             }
167         }
168
169         mesh.vertices = vertices;
170         mesh.uv = uv;
171         mesh.triangles = triangles;
172         mesh.RecalculateNormals();
173     }
174
175     private double GetAvg(ushort[] depthData, int x, int
176     y, int width, int height)
177     {
178         double sum = 0.0;
179         int count = 0;
180         for (int y1 = y; y1 < y + _DownsampleSize; y1++)

```

```

169         {
170             for (int x1 = x; x1 < x + _DownsampleSize;
171                 ++x1)
172             {
173                 if ((x < width) && (y < height))
174                 {
175                     int fullIndex = (y1 * width) + x1;
176
177                     if (depthData[fullIndex] == 0)
178                     {
179                         sum += 4500;
180                     }
181                     else
182                     {
183                         sum += depthData[fullIndex];
184                     }
185                     count++;
186                 }
187             }
188
189             return sum / count;
190         }
191
192         void exportObj(string filenameBase)
193     {
194         Debug.Log("Saving scan");
195
196         // Write obj file
197         using (System.IO.StreamWriter file = new
198             System.IO.StreamWriter(filenameBase + ".obj"))
199         {
200             file.WriteLine("mtllib " + filenameBase +
201             ".mtl");
202             file.WriteLine("o mesh");
203             foreach (Vector3 v in vertices)
204             {
205                 file.WriteLine("v " + v.x + " " + v.y +
206                 " " + v.z);
207             }
208             foreach (Vector2 v in uv)
209             {
210                 file.WriteLine("vt " + v.x + " " + v.y);
211             }
212             foreach (Vector3 v in mesh.normals)
213             {
214                 file.WriteLine("vn " + v.x + " " + v.y +
215                 " " + v.z);
216             }
217             file.WriteLine("usemtl meshMaterial");
218             file.WriteLine("s off");
219             for (int i = 0; i < triangles.Length; i += 3)
220             {

```

```

217             file.WriteLine("f " + (1 + triangles[i +
218     ↵  0]) + "/" + (1 + triangles[i + 0]) + "/" + (1 +
219     ↵  triangles[i + 0]) + " " +
220     ↵  (1 + triangles[i +
221     ↵  1]) + "/" + (1 + triangles[i + 1]) + "/" + (1 +
222     ↵  triangles[i + 1]) + " " +
223     ↵  (1 + triangles[i +
224     ↵  2]) + "/" + (1 + triangles[i + 2]) + "/" + (1 +
225     ↵  triangles[i + 2]));
226     }
227
228     // Write mtl file
229     using (System.IO.StreamWriter file = new
230     ↵  System.IO.StreamWriter(filenameBase + ".mtl"))
231     {
232         file.WriteLine("newmtl meshMaterial");
233         file.WriteLine("o mesh");
234         file.WriteLine("Ns 96.078431");
235         file.WriteLine("Ka 1.000000 1.000000
236         ↵  1.000000");
237         file.WriteLine("Kd 0.640000 0.640000
238         ↵  0.640000");
239         file.WriteLine("Ks 0.500000 0.500000
240         ↵  0.500000");
241         file.WriteLine("Ke 0.000000 0.000000
242         ↵  0.000000");
243         file.WriteLine("Ni 1.000000");
244         file.WriteLine("d 1.000000");
245         file.WriteLine("illum 2");
246         file.WriteLine("map_Kd " + filenameBase +
247         ↵  ".jpg");
248     }
249     File.WriteAllBytes (filenameBase + ".jpg",
250     ↵  ImageConversion.EncodeToJPG
251     ↵  (colorManager.GetColorTexture()));
252 }
}

```

3. Attach the ObjectScan component to the object in the scene. Also attach the ColourSourceManager and DepthSourceManager to the same object. Provide these as the properties of the ObjectScan component (for example, by dragging and dropping the object onto each of the fields in turn).

The application can be tested with a Kinect depth camera attached. The view of the mesh may vary depending on the layout of your room, and may be easiest to

monitor in the Scene tab of the Unity software editor. Press the Fire1 control (usually control key or mouse button) to save the current scene. The resulting obj, mtl and jpg files can be imported into 3D modelling packages that support this format.

4. The process of generating the mesh involves several stages. Firstly the point cloud is represented as a set of vertices.

---

```
1  vertices = new Vector3[width * height];
```

---

Since the point cloud is provided as a grid of points, the number of points is determined by the dimensions of this grid. However the Unity software has a limitation on the number of vertices allowed in a single mesh which means that detailed point clouds may have to be reduced in resolution by skipping over neighbouring points. This downsampling process typically aggregates these neighbouring points to create a single point representing the neighbourhood.

5. Colour is associated with each point through a texture mapping process. Each vertex corresponds to the colour at a particular coordinate in the colour camera image. These coordinates ( $u, v$ ) are associated with each vertex, and so a list with the same number of entries is required.

---

```
1  uv = new Vector2[width * height];
```

---

6. Finally the point cloud can be converted into a boundary representation such as a polygon mesh. In the grid layout 4 neighbouring points can be connected into a quadrilateral by forming a polygon from the 4 vertices. In practice we tend to use triangles, and so 2 triangles serve the same purpose. The triangles fit in the gaps between vertices, so the grid of triangles is smaller by 1 on both the horizontal and vertical dimensions.

---

```
1  triangles = new int[6 * ((width - 1) * (height - 1))
];
```

---

The values used to represent triangles are integers; the position in the list of vertices that actually contains the position of each of the 3 vertices of the triangle.

7. Having set up and initialized values for the triangle structure these values can remain unchanged throughout the remainder of the process. The only change is to the properties of each vertex as the structure of the scene changes. This update follows the process of:

```

1   for each vertex in vertices
2       get the depth values for points in the
            neighbourhood contributing to this vertex
3       aggregate the position of the points to get the
            current position of the vertex
4       update the uv coordinates of the vertex to
            correspond to the position in the colour
            camera image

```

---

Depth values captured by the depth camera often have a different scale to the horizontal and vertical coordinates. A scaling correction factor is provided that can be calibrated to match dimensions measured in physical reality.

8. Saving the mesh requires writing this information to a file. The specific formatting is determined by the standards for the 3D object file format used. The obj file format requires (apart from some header information):
  - (a) The name of the file containing the materials (including texture) for the object.
  - (b) A list of vertices. This consists of the prefix “v” followed by the x, y and z coordinates, with one entry per line for each vertex.
  - (c) A list of the texture coordinates. This consists of the prefix “vt” followed by the u and v coordinates.
  - (d) A list of the vertex normals. This consists of the prefix “vn” followed by the x, y and z coordinates of the vertex normal vector. These are derived by the Unity software from the vertex and triangle information and so do not need to be explicitly managed by this component.

- (e) The list of the triangles. This consists of the prefix “f” (for face) followed by the list of vertices involved in the triangle. Each vertex is defined as the index (offset) in the lists of vertices, texture coordinates and normals respectively, separated by the slash (/) character.

The material file allows customization of surface colours and properties. However this example fixes those values to suitable defaults and determines surface colour purely from the texture captured by the colour camera. This is converted to jpg format and written to a file using the handy functions provided by the Unity software.

---

```
1 File.WriteAllBytes (filenameBase + ".jpg",
    ImageConversion.EncodeToJPG (colorManager.
        GetColorTexture ()));
```

---

9. This example is intended as a gateway to further opportunities related to environment scanning. These include manipulation of point cloud data. Several modelling packages provide facilities to align and weld meshes taken from different angles. The current triangulation process is also imperfect and is not able to distinguish the separate objects in the scene.

There is also an opportunity to attach a tracking device of some kind to the Kinect so that each snapshot can be tagged with the position and orientation from which it is acquired. This might allow some automatic stitching of meshes, but also allows opportunities to capture information lost due to occlusion.

## 17.6 Capturing and presenting scenes using 360° cameras

**Component 17.6.1: Scene capture with 360° cameras.**

360 Capture
-------------

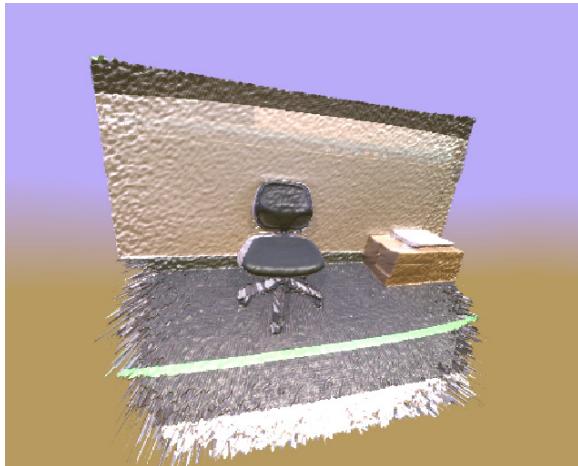


Figure 17.5.2: The Kinect is used as a depth camera to scan part of a room.

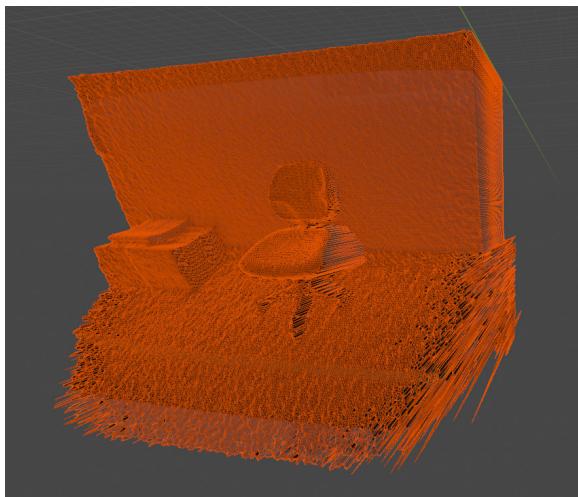


Figure 17.5.3: The scanned mesh can be exported to a file and imported into other packages.

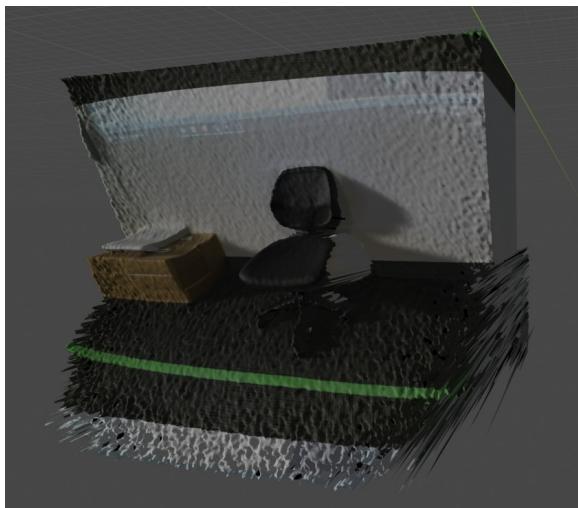


Figure 17.5.4: The physical scene is then recreated in a virtual environment.

*Modern 360 degree cameras simplify capture of the physical setting. Most have one or more fisheye-style lenses that allow them to capture panoramic images. These are usually done from a single viewpoint and so the images are best used as a distant background attached to a sky box or sky sphere. The lack of depth information associated with stereo images leaves the images looking flat.*

*Stereo 360 degree cameras are becoming more common and it is anticipated that more content in this format (effectively pairs of panoramic images) will be available. Display of these images uses the principles covered in component 16.4.1.*

*Capturing a 360 degree image is usually as simple as pressing a button on the camera. Challenges relate to the composition of the image. The photographer is visible in the image unless they hide behind objects, or trigger the camera remotely (often through a mobile app). Images are normally exported using an equirectangular mapping between the rectangular image and the surrounding spherical region that is captured. This does devote a significant portion of the image (top and bottom) to the region corresponding to the vertical extremes of the scene being captured. However many mapping processes onto spherical geometry still introduce significant distortion in these areas. The camera itself, or the stand holding the camera, is often visible at*

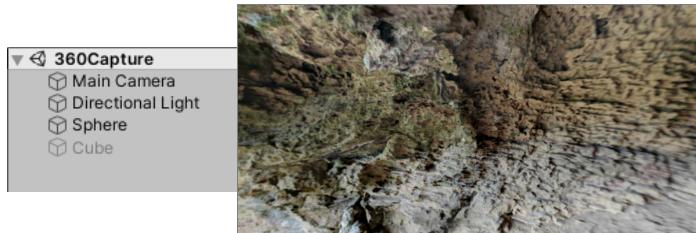
*the base of the image.*

*The image effectively freezes the structure of the scene and so there are limited opportunities to reposition elements at a later point. The viewpoint should be chosen to match the expected viewer position. Particularly the height of the camera should correspond with head height, unless special effects are intended. Movement of the participant is discouraged as the scene content remains unchanged (unless a new image is available for every position that the participant moves to).*

*Images and video can be captured and are handled similarly (apart from the time dimension).*

*This example assumes that a 360 degree image has been captured and demonstrates the process of displaying such an image.*

Figure 17.6.1: Scene represented using a 360° image.



1. Start with a new project. Set this up for building on a mobile device. There are several options for display, all of which rely on using the device's gyroscope to determine the direction of gaze (see component 18.3.1). The screen of the device can act as a window into an overlaid virtual world containing the captured image. Alternatively the device could be placed in a VR headset such as the Daydream to allow complete immersion in the virtual representation of the physical scene that was captured. Depth perception is possible in the latter case but will reduce to a view of the image on the surface of a sphere where monoscopic images are used.
2. The default sphere in the Unity software provides an adequate surface to display 360 degree images. Add a new sphere to the scene. Expand the size of the sphere

so that it comfortably encloses the viewer (a scale of about 10 units should be sufficient).

Also ensure that the sphere and the camera are in the same position, so that the camera is effectively at the center point of the sphere. Note that when this happens the sphere is usually invisible as the surface shader is set to render the outside of the sphere only.

3. Create a new material, called PhotoMaterial, and apply this to the sphere. A custom shader needs to be applied to this material, in the first instance to ensure that the inside of the sphere is visible.

Create a new Unlit Shader, named PhotoShader. Set this to be the shader of the PhotoMaterial. Add the line below to the SubShader (just below the line containing “LOD 100”) to disable backface culling and make both surfaces of the sphere visible.

---

1 Cull Off

---

Add a 360 degree image to the project, and set this as the texture for the PhotoMaterial.

Build, deploy and run the project. This provides the core of any 360 degree photo or video viewer.

4. Some of the artefacts in the display of the image result from the low polygon representation of the sphere. Texture unwrapping becomes quite awkward when dealing with the distortion around the top and bottom of the sphere. One solution would require that a higher resolution sphere object is used that reduces the size of the polygons affected by the singular points at the poles. An alternative strategy is to perform texture unwrapping dynamically in the shader itself.

In a spherical object each pixel in each polygon lies (roughly) on the surface of a sphere. The position of the pixel on the sphere can be matched to the colour in the physical scene by transforming the pixel position to a

point on the 360 degree image using the equirectangular mapping.

$$\begin{aligned} u &= \text{longitude} \\ &= \tan^{-1}\left(\frac{z}{x}\right)() \\ v &= \text{latitude} \\ &= \tan^{-1}\left(\frac{y}{\sqrt{x^2+z^2}}\right) \end{aligned}$$

This assumes that the coordinates  $(x, y, z)$  of pixels on the surface of the sphere use the convention that the y-axis is vertical. Mapping is achieved on a per-pixel basis by embedding the calculation into the pixel/fragment shader, as shown in Algorithm 17.9.

Code Listing 17.9: PhotoShader. Equirectangular image mapping is achieved by performing the texture mapping process in the fragment shader.

```

1 Shader "Unlit/PhotoShader"
2 {
3     Properties
4     {
5         _MainTex("Texture", 2D) = "white" {}
6     }
7     SubShader
8     {
9         Tags { "RenderType" = "Opaque" }
10        LOD 100
11        Cull Off
12
13        Pass
14        {
15            CGPROGRAM
16            #pragma vertex vert
17            #pragma fragment frag
18            // make fog work
19            #pragma multi_compile_fog
20
21            #include "UnityCG.cginc"
22
23            struct appdata
24            {
25                float4 vertex : POSITION;
26                float2 uv : TEXCOORD0;
27            };
28
29            struct v2f
30            {
31                float4 objvertex : TEXCOORD1;

```

```

32     float2 uv : TEXCOORD0;
33     UNITY_FOG_COORDS(1)
34     float4 vertex : SV_POSITION;
35 };
36
37     sampler2D _MainTex;
38     float4 _MainTex_ST;
39
40     v2f vert(appdata v)
41 {
42     v2f o;
43     o.objvertex = v.vertex;
44     o.vertex = UnityObjectToClipPos(v.vertex);
45     o.uv = TRANSFORM_TEX(v.uv, _MainTex);
46     UNITY_TRANSFER_FOG(o,o.vertex);
47     return o;
48 }
49
50     fixed4 frag(v2f i) : SV_Target
51 {
52     fixed2 uv;
53     float xz = sqrt(i.objvertex.x * i.objvertex.x +
54     i.objvertex.z * i.objvertex.z);
55     float latitude = atan2(i.objvertex.y, xz);
56     float longitude = atan2(i.objvertex.z,
57     i.objvertex.x);
58     uv.y = 0.5 + latitude / 3.14159;
59     uv.x = longitude / (2 * 3.14159);
60     fixed4 col = tex2D(_MainTex, uv);
61     // apply fog
62     UNITY_APPLY_FOG(i.fogCoord, col);
63     return col;
64 }
65 ENDCG
66 }
```

5. Sometimes seams appear in the sphere, particularly where the texture wraps back from right to left. These can be attributed to a side effect of the mip-mapping process used to filter the textures to fit into the visible portion of the screen. Filtering behaves differently at the borders of the textures. A quick solution is to disable the mipmap property for each texture involved. Since the textures are shown at a constant distance from the camera this should not have a significant effect on visual quality. The only consideration is

to make sure the image resolution used for the texture is similar to the pixel resolution that it is shown at so that sampling (aliasing) artefacts can be avoided.

Figure 17.6.2: Viewing a 360° scene.



## 17.7 *Capturing cylindrical panoramas of scenes using Cardboard Camera and a mobile phone*

**Component 17.7.1: Scene capture with Cardboard camera.** *Cylindrical stereo panoramas can be captured directly with an Android device. This provides opportunities for scene capture even in the absence of a dedicated 360° camera. There are some tradeoffs involved.*

*Pros include:*

- Capture stereo imagery.
- Requires only a single camera in a generic device.

However there are some limitations:

- Capture is not instantaneous and requires multiple images collected over a period of time. Moving elements in the scene produce undesirable effects.
- The capture is a cylindrical panorama and only captures a relative narrow horizontal band of the scene.

Scene capture uses the Cardboard Camera application. The capture process uses omni-direction stereo<sup>2</sup> which captures the images for the left and right eyes by finding regions on the cylindrical scan where the tangent to the cylinder lines up with the view direction through a column of pixels appropriate to each of the eyes.

1. Download and install the Cardboard Camera application from the Android Play store.
2. Photographs are taken by moving the camera in a horizontal circle. Opportunities for stereo viewing are enhanced by ensuring that the radius of the circle is wide enough to capture images that represent viewpoints from both eyes. Holding the camera at arm's length while rotating allows this.
3. The resulting photographs can be viewed directly on the device using a Cardboard or Daydream headset, together with the viewer provided by the application. Alternatively they can be downloaded and mapped to a cylinder in a Unity software project (see component 17.6.1 for some suggestions on shader structures).
4. The images downloaded from the device appear to be a single jpeg image. However both the left and right views are encoded into the file. The left image can be extracted relatively directly with most image manipulation tools. The right image is embedded in the meta-data and needs additional software to extract.<sup>3</sup>

Once the two images are available then they can be viewed as per the process in component 16.4.1.

## 17.8 Converting complex virtual scenes to stereo panoramic images using Unity

### Component 17.8.1: Augmented scenes with Unity.

<i>Unity</i>
<i>Capture</i>

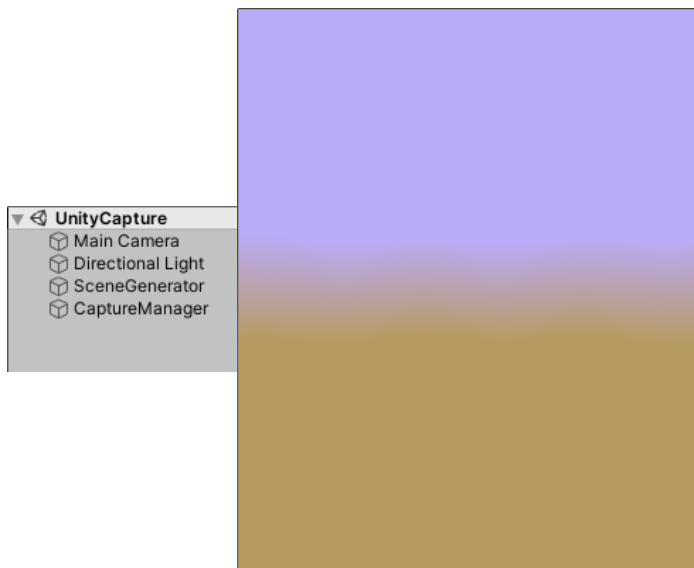
<sup>2</sup> <https://developers.google.com/vr/jump/rendering-ods-content.pdf>

<sup>3</sup> A python script for retrieving the additional content is described at this address: <http://vectorcult.com/2015/12/extracting-the-audio-stereo-pair-from-cardboard-camera-3d-panoramic-images/>. For small numbers of images, the web service at <https://cctoolkit.vectorcult.com/> may be more convenient. In case this service is discontinued, the source code is at: <https://bitbucket.org/pansapiens/cardboardcam>.

*Recent versions of the Unity software have introduced the ability to render the scene to a stereo panorama. These are produced as textures for a cube (cube map) but can readily be converted to an equirectangular project for compatibility to previous examples.*

*This functionality is useful for a number of reasons. Some VR application stores require snapshots of the application running in this form, as a way of showing off the application to potential customers. It can also be used in applications to replace complex static background scenery with a single sphere with stereo textures. Provided there is no interaction with the baked in background objects (which could produce occlusion issues), the visual appearance of the application is not affected yet the performance could be significantly improved.*

Figure 17.8.1: Scene capture within Unity can be added into any existing application.



1. Start with a new project. This example is intended to run on the desktop version of the application but could be modified to run on a mobile device by refining the activation process, and ensuring that files are written to writable directories.

Populate the scene. Normally this process is added to an existing application.

2. Create a new empty object in the scene, and attached the C# script component called CreateStereo360 to it. The code for this script is provided in Algorithm 17.10.

Code Listing 17.10: CreateStereo360. Stereo panoramas captured directly from Unity software applications.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Rendering;
5
6  public class CreateStereo360 : MonoBehaviour {
7
8      public Camera cam;
9      public RenderTexture cubemapLeft;
10     public RenderTexture cubemapRight;
11     public RenderTexture equirect;
12     public bool renderStereo = true;
13
14     void saveStereo360Image (string filename)
15     {
16
17         cam.RenderToCubemap(cubemapLeft, 63,
18             Camera.MonoOrStereoscopicEye.Left);
19         cam.RenderToCubemap(cubemapRight, 63,
20             Camera.MonoOrStereoscopicEye.Right);
21
22         if (equirect == null)
23             return;
24
25         if (renderStereo)
26         {
27             cubemapLeft.ConvertToEquirect(equirect,
28                 Camera.MonoOrStereoscopicEye.Left);
29             cubemapRight.ConvertToEquirect(equirect,
30                 Camera.MonoOrStereoscopicEye.Right);
31         }
32         else
33         {
34             cubemapLeft.ConvertToEquirect(equirect,
35                 Camera.MonoOrStereoscopicEye.Mono);
36         }
37
38         Texture2D image = new Texture2D (equirect.width,
39             equirect.height, TextureFormat.RGB24, false);
40         image.ReadPixels (new Rect(0, 0, equirect.width,
41             equirect.height), 0, 0);
42         byte[] bytes;
43         bytes = image.EncodeToPNG ();
44
45         System.IO.File.WriteAllBytes (filename, bytes);

```

```

39
40
41     void Update () {
42
43         if (Input.GetKey (KeyCode.F2))
44         {
45             saveStereo360Image ("render.png");
46         }
47     }
48 }
```

Provide the scene camera as the Cam property for this component. Also create three new RenderTexture assets to hold the intermediate images produced during capture. Provide these to the Cubemap Left, Cubemap Right and Equirect properties. Ensure that the render textures used for the left and right cubemaps have their dimension property set as Cube. The render texture used for the equirectangular texture should remain as a 2D image.

The dimensions of the render textures are also quite small by default. Dimensions of  $4096 \times 4096$  may be appropriate.

3. When the application is running, pressing the function key F2 causes it to render the left and right views to the two cubemap textures. These are then converted into equirectangular projections and stacked into a single image which is written to the file render.png.

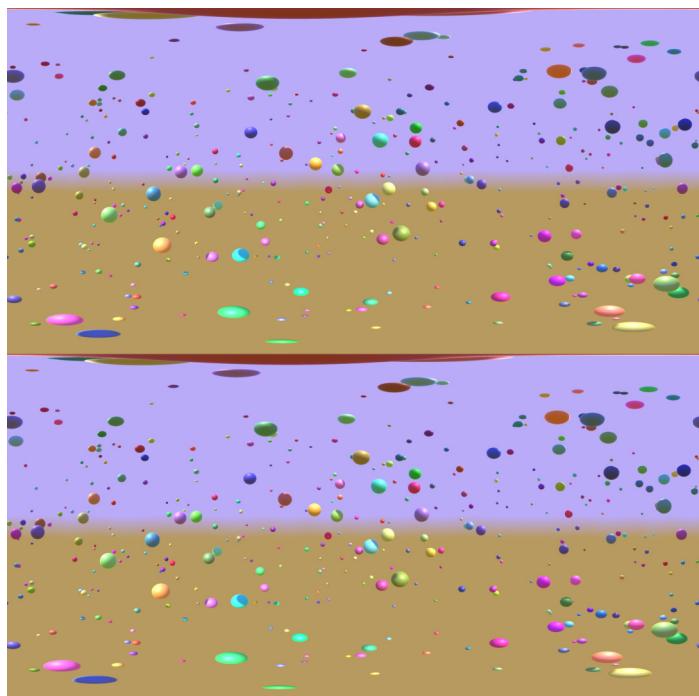
The renderer image can then be used in a dedicated stereo panorama viewer (component 16.4.1), or applied to a sky-sphere in the same application saving the need to including static background objects in the scene.

## 17.9 Converting complex virtual scenes to stereo panoramic images using Blender

### Component 17.9.1: Augmented scenes with Blender.

<i>Blender</i>
<i>Render</i>

Figure 17.8.2:  
Stereoscopic  
360°image cap-  
tured within a  
Unity scene.



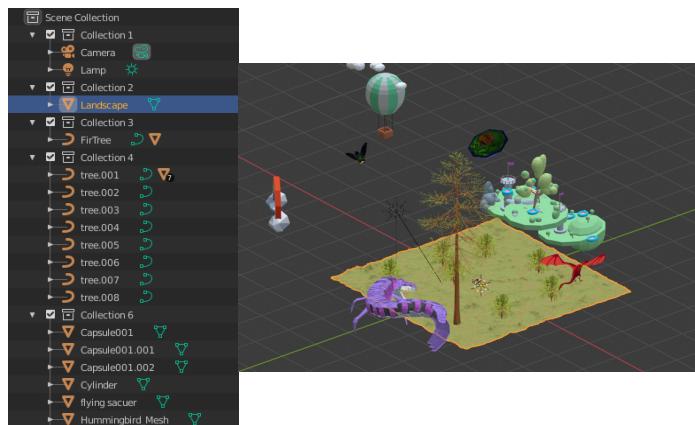
*Static 3D content for virtual and augmented reality scenes can also be generated offline using a 3D modelling package, and included as background layers in visual content presented through stereoscopic display mechanisms. Similarly to stereo 360° images captured from physical settings, such content can have effectively unlimited levels of detail since the performance overheads associated with rendering the image are all paid during the preparation of the images. Overhead during the actual running of the application is constant regardless of the detail included.*

*Modelling packages offer opportunities to completely control the content of the scene including objects that may not be possible to include in a physical scene because they are expensive, fictional or abstract. The price associated with modelling detailed scenes is the amount of effort required to produce the models in the first place.*

*This example demonstrates the process used to create a stereo pair of equirectangular renderings using Blender as the mod-*

elling package.

Figure 17.9.1: Scene structure for an environment in Blender.



1. Start Blender and create a new project. Position the camera at the origin, and raise it to head height. The orientation of the camera is not crucial since the rendering process does capture every direction. However the equirectangular images do distribute detail differently at the top and bottom, so it is suggested that the camera be horizontal particularly if likely viewing actions involve turning side to side. The coordinate settings in Blender often use the z-axis as the vertical axis, so position of 0, 0, 1.5 and rotation of 90, 0, 0 is a handy starting point.
2. The scene does need some content to be able to demonstrate the experience. A ground plane is fun to include since this does allow the rendered image to contain content that is both distant and right under the participant. Featureless content should be avoided since this tends to lack the cues required for the brain to extract depth from the left and right images.

Blender does contain some tools to generate landscapes. These may need to be enabled in the version you are using (File/User Preferences/Add Ons), but are accessible under the Add/Mesh/Landscape option when present. There are a number of preset options

that provide stimulating terrain. Suggested settings are: Operator Presets: use the Large Terrain. Set Mesh Size of 50 (both X and Y), Subdivisions to 1024 (both X and Y) and Depth to 12. Since the resulting object is large it is worth moving it to a layer of its own and only activating it when it is time to render.

3. Trees can be created with the tree generator (Add/Curve/Sapling Tree Gen) which may also need to be enabled.
4. Given the size of the scene it is also worth changing the light source into a Sun, and positioning this to illuminate the whole scene.

The panoramic rendering options require use of the Cycles rendering engine so switch to this from the Blender renderer. The control is usually centered at the top of the Blender window.

5. Materials should be added to the scene elements to brighten it up, but also to help provide some distinctive feature points.
6. The settings for stereoscopic panoramic rendering are contained in various areas:
  - (a) Under the Render Layers options:
    - i. Enable the checkbox next to Views.
    - ii. Ensure that Stereo 3D is selected, and that both the left and right views are enabled.
  - (b) Select the camera. Under the camera data settings:
    - i. Under the lens settings, select Panoramic. Set the type to Equirectangular.
    - ii. Under the stereoscopy settings, select the Parallel option (to keep both left and right camera facing forward).
    - iii. Enable the Spherical Stereo checkbox.
  - (c) Under the Render settings:

- i. Under Output, set the output file format to PNG and RGBA.
- ii. Under Dimensions, set the resolution to 512 by 256 at 100% (increase this when producing high resolution final output).
- iii. Also under Dimensions, set the start and end frames to 0.

Render the images. These can be tested in the stereo viewer application described in component 16.4.1.

7. While the content rendered is static and baked into the image there are still ways to achieve some variety and motion through the use of layers. Objects at different depths can be rendered into separate images and composited in the viewer.

To demonstrate this we turn off the background sky layer in our rendering and leave this transparent. We then render sky detail into a separate image which is placed on another shell around the viewer.

To achieve a transparent background in blender, ensure that RGBA is selected under the Output options of the Render settings, and that the Transparent checkbox is selected under the Film options also under Render settings.

Rerender the scene with a transparent background, and save the images.

8. Now add background to the scene in a separate layer. These objects need to be placed beyond the existing scene elements. Disable the transparent background setting. Enable only the layers containing camera, lights and background objects. Rerender the scene to another set of image files.

Extend the stereo image viewer (component 16.4.1) by adding an extra sphere at a slightly greater distance. Create a new material for this. Apply the images from the first render (with the transparent background) to

the material of the inner sphere, and the images from the second render (with background objects to the second sphere). The images with transparency need to have the “Alpha Is Transparency” option selected.

The StereoShader needs some modifications to support transparency. Adapt the lines configuring the SubShader as follows:

```

1   ...
2   SubShader
3   {
4       Tags { "Queue"="Transparent" "RenderType"=
5           "Transparent" }
6       LOD 100
7       Cull Off
8       ZWrite Off
9       Blend SrcAlpha OneMinusSrcAlpha
9   ...

```

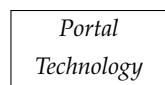
---

The material for the inner sphere also needs its render queue setting changed to 4000, so that it renders over the outer sphere pixels.

For a bit of extra fun, set the outer sphere to rotate slowly.

### 17.10 *Transitioning between realities using portals*

#### Component 17.10.1: Portal technology.



*Motivation:* A portal provides an opportunity to connect or overlay locations. For example: the area used for the experience (typically a room or office) is merged with the region accessed through the portal regardless of where or when this is.

One potential use of augmented content overlaid onto a camera image is to create a portal. This is a window to a virtual world visible in the overlay in the physical. Unlike most 3D content, the portal represents a transition to another pocket universe occupying the same space

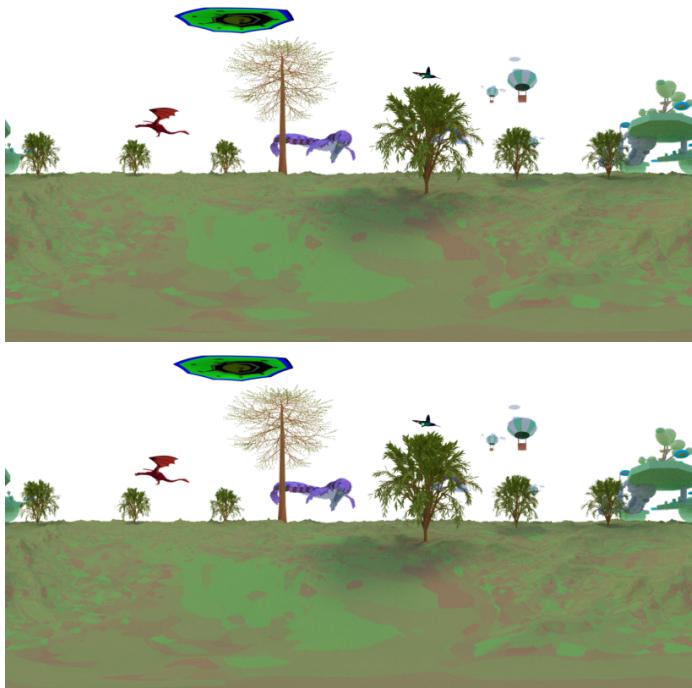


Figure 17.9.2:  
Stereoscopic  
panoramas ren-  
dered in Blender.

as the physical world. Participants can nominate which space they want to occupy by stepping through the portal. The act of stepping through represents the interaction mechanic of the portal.

There are several strategies for creating portals and this example presents just one approach. We assume the following rules for portals:

1. If you're in the physical world, then the display order is: portal and virtual content seen through the portal, and then camera image showing physical world.
2. If you're in the virtual world, then the display order is: portal and physical content (camera image) visible through the portal, and then the virtual world.
3. You can only change worlds by stepping through the portal (colliding with the door way).

The following process outlines how these steps are achie-

ved. This example does not reference any augmented reality toolkits, so would need to be integrated with existing approaches for displaying camera content if used with these.

It is also worth noting that although the two universes involved are named physical and virtual, there are no particular restrictions that enforce that breakdown. Both are treated equivalently, allowing combinations of multiple universes (physical, virtual or other) through many portals.

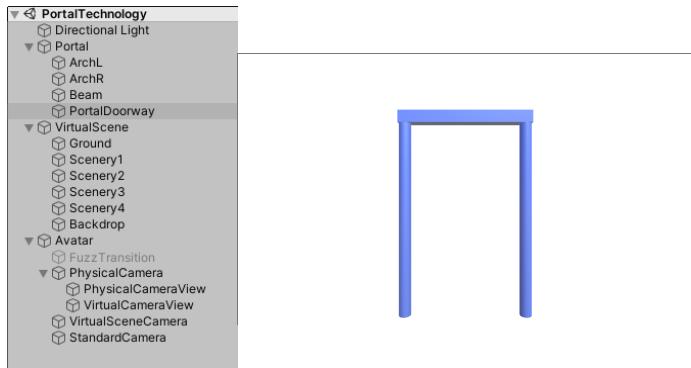


Figure 17.10.1:  
A portal is often presented as a doorway.

1. Start with a blank project. There are three sets of content that is rendered:
  - (a) The view from the physical camera is rendered onto a plane placed just in front of the scene camera.
  - (b) The view of the virtual world is rendered onto a plane placed just in front of the scene camera.
  - (c) The actual portal structure itself is then the only geometry that is directly rendered by the scene camera.
2. Create an empty object to represent the avatar, and move the main camera to be a child of this. Place the avatar at the origin, and the camera at eye level relative to this. A steering script on the avatar provides the facility required to test the portal.
3. Create a new empty object to hold the planes that are placed just in front of the camera. Attach a script to it

using Algorithm 17.11 that will lock this object just in front of the scene camera and force it to fill the field of view. Create two plane objects as children (one labelled PhysicalCameraView and the other VirtualCamer-aView) and set their rotation to: 90, 180, 0 and scale to 0.1, 0.1, 0.1.

Code Listing 17.11: ShowPhysicalCamera. Lock the object just in front of the scene camera. A secondary task is to start the physical camera view streaming to a material asset provided.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  // Show the view from from an image/texture on a plane
7  // just in front of the camera. Recommend adding this
8  // script to an empty
9  // containing any planes in this category.
10 // Set the plane transform to: rot: 90, 180, 0 and scale
11 // to 0.1, 0.1, 0.1
12 // to achieve a unit size plane facing the z axis. The
13 // transform for
14 // the empty is set by this script.
15 public class ShowPhysicalCamera : MonoBehaviour {
16
17     public Material camTexMaterial;
18     public Camera camera;
19
20     private WebCamTexture webcamTexture;
21
22     void Start () {
23         webcamTexture = new WebCamTexture ();
24
25         camTexMaterial.mainTexture = webcamTexture;
26         webcamTexture.Play ();
27     }
28
29     void Update ()
30     {
31         // With thanks: https://answers.unity.com/questions/314049/how-to-make-a-plane-fill-the-field-of-view.html
32         float pos = (camera.nearClipPlane + 0.01f);
33
34         transform.position = camera.transform.position +
35         camera.transform.forward * pos;
36
37         float h = Mathf.Tan (camera.fieldOfView *
38         Mathf.Deg2Rad * 0.5f) * pos * 2.0f;
39
40
41 }
```

```

35     transform.localScale = new Vector3(h * camera.aspect,
36     ↵ h, 1.0f);
37 }
```

Create a material for the physical camera image texture and provide it as a parameter to this script. This material needs to be applied to the plane corresponding to the view from the physical camera. When run, the physical camera image should be visible on the PhysicalCameraView plane (might need to hide the other plane to see it).

4. Create the objects to populate the virtual scene. These should all ideally be placed under a single empty object to make the next step easier.
5. Different content needs to be rendered by different cameras. Under the Inspector, and Layer, create new layers for Portal, PhysicalScene and VirtualScene. Set the scene objects to all belong to the Virtual Scene layer (starting at the parent is easier, and then ensure all children inherit this change). Set the physical and virtual camera view planes to render only to the Physical Scene layer.

Create an extra camera under the avatar, and set its culling mask to render only the VirtualScene layer (specifically exclude the portal layer and the physical scene layer). Add a Render Texture to the project and set this camera to render to this render texture. Add this render texture to a new material VirtualSceneMaterial, and assign this material to the VirtualCameraView plane. When the application runs, the virtual scene should be visible on the surface of this plane.

The main camera should be instructed not to render the VirtualScene layer.

6. Create portal structure. Use some geometry for the doorway supports, but also add a cube occupying the interior of the doorway - the boundary between realities.

This cube gets a material based on a custom shader that both renders it invisible but also marks the region of the doorway in the stencil buffer. Derive this shader from the standard surface shader. The relevant lines that need to be changed are shown in Algorithm 17.12. Ensure all parts of the portal are in the Portal layer.

Code Listing 17.12: `TransparentStencil`. The shader functions that prevent the doorway being visible, but also ensuring it writes to the stencil buffer.

```

1 Shader "Custom/TransparentStencil" {
2     Properties {
3         _Stencil_Level ("Stencil clip code", int) = 1
4         _Color ("Color", Color) = (1,1,1,1)
5         _MainTex ("Albedo (RGB)", 2D) = "white" {}
6         _Glossiness ("Smoothness", Range(0,1)) = 0.5
7         _Metallic ("Metallic", Range(0,1)) = 0.0
8     }
9     SubShader {
10         Tags { "RenderType"="Opaque" }
11         LOD 200
12         ColorMask 0
13         ZWrite Off
14         Stencil
15         {
16             Ref [_Stencil_Level]
17             Comp Always
18             Pass Replace
19         }
20
21         CGPROGRAM
22         // Physically based Standard lighting model, and
23         // enable shadows on all light types
24         #pragma surface surf Standard fullforwardshadows
25
26         // Use shader model 3.0 target, to get nicer looking
27         // lighting
28         #pragma target 3.0
29
30         sampler2D _MainTex;
31
32         struct Input {
33             float2 uv_MainTex;
34         };
35
36         half _Glossiness;
37         half _Metallic;
38         fixed4 _Color;
39 }
```

```

38     // Add instancing support for this shader. You need to
→      check 'Enable Instancing' on materials that use the
→      shader.
→      // See
→      https://docs.unity3d.com/Manual/GPUInstancing.html for
→      more information about instancing.
→      // #pragma instancing_options assumeuniformscaling
UNITY_INSTANCING_BUFFER_START(Props)
→      // put more per-instance properties here
UNITY_INSTANCING_BUFFER_END(Props)

44
45 void surf (Input IN, inout SurfaceOutputStandard o) {
46     // Albedo comes from a texture tinted by color
47     fixed4 c = tex2D (_MainTex, IN.uv_MainTex) * _Color;
48     o.Albedo = c.rgb;
49     // Metallic and smoothness come from slider variables
50     o.Metallic = _Metallic;
51     o.Smoothness = _Glossiness;
52     o.Alpha = c.a;
53 }
54 ENDCG
55 }
56 FallBack "Diffuse"
57 }
```

- The two planes need to be selectively rendered based on whether they are part of the current world, or just visible through the doorway of the portal. Since the stencil buffer now contains 0 outside the doorway, and 1 inside, we use this to filter which part of the two planes are visible.

Set the materials for both planes to use the ClipSurface shader shown in Algorithm 17.13.

Code Listing 17.13: ClipSurface. This shader writes only pixels to regions where the stencil buffer contains values matching the provided stencil level parameter. In addition it avoids writing to the depth buffer to avoid obscuring elements of the portal geometry.

```

1 Shader "Custom/ClipSurface" {
2     Properties {
3         _Stencil_Level ("Stencil clip code", int) = 1
4         _Color ("Color", Color) = (1,1,1,1)
5         _MainTex ("Albedo (RGB)", 2D) = "white" {}
6         _Glossiness ("Smoothness", Range(0,1)) = 0.5
7         _Metallic ("Metallic", Range(0,1)) = 0.0
8     }
9     SubShader {
10         Tags { "RenderType"="Opaque" }
```

```

11    LOD 200
12    Stencil {
13        Ref [_Stencil_Level]
14        Comp Equal
15    }
16    ZWrite Off // Avoid setting depth so we can draw
17    ↳ portal borders over this.
18
19    CGPROGRAM
20        // Physically based Standard lighting model, and
21        ↳ enable shadows on all light types
22        #pragma surface surf Standard fullforwardshadows
23
24        // Use shader model 3.0 target, to get nicer looking
25        ↳ lighting
26        #pragma target 3.0
27
28        sampler2D _MainTex;
29
30        struct Input {
31            float2 uv_MainTex;
32        };
33
34        half _Glossiness;
35        half _Metallic;
36        fixed4 _Color;
37
38        // Add instancing support for this shader. You need to
39        ↳ check 'Enable Instancing' on materials that use the
40        ↳ shader.
41        // See
42        ↳ https://docs.unity3d.com/Manual/GPUInstancing.html for
43        ↳ more information about instancing.
44        // #pragma instancing_options assumeuniformscaling
45        UNITY_INSTANCING_BUFFER_START(Props)
46            // put more per-instance properties here
47        UNITY_INSTANCING_BUFFER_END(Props)
48
49        void surf (Input IN, inout SurfaceOutputStandard o) {
50            // Albedo comes from a texture tinted by color
51            fixed4 c = tex2D (_MainTex, IN.uv_MainTex) * _Color;
52            o.Albedo = c.rgb;
53            // Metallic and smoothness come from slider variables
54            o.Metallic = _Metallic;
55            o.Smoothness = _Glossiness;
56            o.Alpha = c.a;
57        }
58        ENDCG
59    }
60    FallBack "Diffuse"
61

```

Set the render queue levels to 1000 for portal doorway material, and to 1500 for camera and virtual render materials so they render after the portal material but before the bounds of the portal (which have the default setting of 2000).

8. Add the script shown in Algorithm 17.14 to the avatar to change universe order when colliding with the portal. The avatar needs a rigidbody (no gravity) and a box collider (with is trigger set to true) in order for collisions to work. The box collider needs to be small (size of about 0.02) and positioned about the height of the camera. It is recommended that the near clipping of the camera be shortened (to about 0.03) to reduce issues during portal transition. There will be some issues as the portal doorway is clipped by the near clipping plane, destroying the stencil region, just before the transition is triggered. This could be carefully managed by positioning colliders, clipping planes and door dimensions, but a simpler approach is to have a visible transition. A smoke-like particle system can hide the details.

Code Listing 17.14: `FlipOrder`. The change of universe affects which of the physical or virtual world images are presented inside and outside of the portal doorway. Collision with the doorway triggers the change, and potentially activates some visual effects to conceal the transition.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class FlipOrder : MonoBehaviour {
6
7      public Material phyCamMaterial;
8      public Material virCamMaterial;
9      public GameObject transition;
10     public GameObject portal;
11
12     public bool inPhysical = true;
13
14     // Use this for initialization
15     void Start () {
16
17 }
```

```
18
19 // Update is called once per frame
20 void Update () {
21     if (inPhysical)
22     {
23         phyCamMaterial.SetInt ("_Stencil_Level", 1);
24         virCamMaterial.SetInt ("_Stencil_Level", 0);
25     }
26     else
27     {
28         phyCamMaterial.SetInt ("_Stencil_Level", 0);
29         virCamMaterial.SetInt ("_Stencil_Level", 1);
30     }
31 }
32
33
34 public void OnTriggerEnter(Collider other)
35 {
36     if (other.gameObject == portal)
37     {
38         transition.SetActive (true);
39         Debug.Log ("Changing universe");
40         inPhysical = !inPhysical;
41     }
42 }
43
44 public void OnTriggerExit (Collider other)
45 {
46     transition.SetActive (false);
47 }
48
49 }
```

Figure 17.10.2:  
Portals provide  
transitions between  
physical and virtual  
settings.



Now you're thinking with portals.

## 17.11 Aligning overlapping realities

### Component 17.11.1: Aligning worlds

 WorldAlignment

Registering the physical world with the virtual world, or even aligning multiple virtual worlds captured by different devices requires deliberate effort. For example, an application that carefully places virtual furniture to match up with the physical setting displayed using a passthrough camera may work until the application is restarted, and the virtual coordinate system aligns with a new origin. The change in the origin may result from pressing a re-center button on the device, or just loss of settings between sessions. Alternatively, you may have multiple devices each with their own coordinate system, such as a multi-participant session in the same physical environment, or combining readings from several tracking systems (e.g. a kinect for external pose tracking while using a head mounted display for displaying the experience).

One way to partially address the problem is to start or re-center each device in the same starting pose. While this requires little effort from a development perspective, it adds additional challenge to testing the experience and overhead when deploying it. This example creates components to allow each individual device to reset its alignment to conform to a global coordinate system, at a time defined by the user. This does require deliberate effort on the part of the software developer as the facilities to support this are at the root of the scene hierarchy description. However, once the pattern is established it can become a standard facility offered across multiple applications.

Key concepts required are:

**Local coordinate system:** The coordinate system used by the device. This is determined by where the device starts or is calibrated (re-centered). We expect this to change several times during the experience, but offer the facility to re-align this to the common coordinate system shared amongst all participating devices each time the local coordinate system is reset.

*Global coordinate system:* This is the coordinate system in which all shared assets in the scene are defined. Once defined it never changes. Rather, we find a way of relating each device's coordinates to this common global coordinate system. Scene descriptions can even be saved using these coordinates, and be assured that they can be restored.

*Local to global transformation:* The relationship between the coordinate systems is defined by a transformation. This represents all the translation, scaling and rotation required to take a coordinate in the local system and produce the corresponding coordinate referenced relative to the origin and axes of the global coordinate system. This transformation will change each time the local coordinate system is reset. This transformation can be deduced by manually aligning a representation of an object in local coordinates, with the representation of the same object in global coordinates.

*Global to local transformation:* This is the inverse of the local to global transformation, used to take the coordinates of objects defined in the global system, and produce their coordinates in the local coordinate system being used on a individual device.

# 18

## *Objects for Interaction Components*

### Contents

---

18.1	■ Object control during testing and debugging . . . . .	299
18.2	■ Tracking the user and objects to allow interaction . . . . .	304
18.3	■ Sensing the orientation of a device . . . . .	308
18.4	■ Synthetic buttons and dials for use in an augmented reality user interface . . . . .	313
18.5	■ Identifying and recognising physical objects . . . . .	319
18.6	■ Capturing the structure of objects in the scene using structured light . . . . .	336
18.7	■ Aligning multiple point clouds to reconstruct the appearance of objects . . . . .	354

---

18.1 ■ *Object control during testing and debugging*

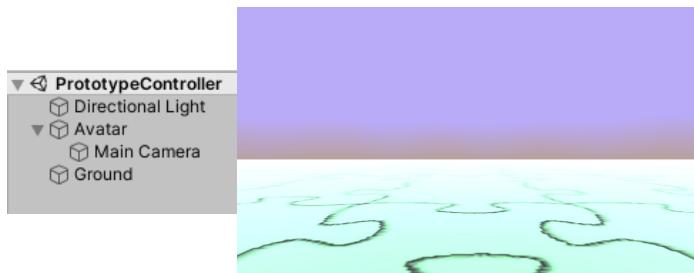
Component 18.1.1: Prototyping location tracking component.

Prototype  
Controller

The need for testing with augmented reality applications is complicated by the nature of the applications that require that the user move around and explore the physical setting. This can be time consuming, but also makes

it harder to replicate any issues that occur since the paths taken may not be exactly the same each time. Device emulators often provide synthetic sensor data from which a location tracking component can derive its position. This prototyping component just controls the pose directly allowing either control from mouse and keyboard, or by simulating a known trajectory.

Figure 18.1.1: A synthetic controller to support development processes.



The component is directly attached to the user's avatar object. Without input it is passive and so can be used concurrently with any other location tracking component. When activated it directly controls the object's pose and may then clash with any other components attempting to do the same.

Controls are derived from the Unity software input system. The existing horizontal and vertical axes are used for changing position, with a third (Depth) axis needing to be added to the input manager to achieve movement in all three dimensions. The offsets in the mouse coordinates are used to modify the orientation of the user.

A key aspect of location tracking to support prototyping involves being able to have the user follow a set and repeatable trajectory. A path following algorithm follows the circumference of a regular  $n$ -sided polygon. For small values of  $n$ , the edge segments are long straight lines with distinct abrupt turns at each vertex. For large values of  $n$  the path approximates a circle providing smooth and continuous motion.

The script for this component is available as Algorithms 18.1.

Code Listing 18.1: `PrototypeLocationTracking`. The prototyping loca-

tion component allows both direct manipulation of location as well as a path following process for hands-off testing of movement.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  // Provide a simulated location tracking facility for use in
6  // test
7  // applications.
8  public class PrototypeLocationTracking : MonoBehaviour {
9
10     public float moveSpeed = 10.0f;
11     public float turnSpeed = 100.0f;
12     public float pathSpeed = 1.0f;
13
14     [Tooltip ("If checked, step size depends on rate at
15     which time passes, otherwise step size is constant and
16     repeatable per call to getLocation")]
17     public bool realTime = true;
18     public bool manualMove = true;
19     public bool manualRotate = false;
20     public bool autoFollowPath = false;
21
22     public float pathRadius = 10.0f;
23     public int pathSides = 4;
24
25     private Vector3 lastPosition;
26     private Vector3 velocity;
27
28     void Start () {
29
30     }
31
32     private void processPositionControls ()
33     {
34         // Handle change in position using input axes.
35         float h = 0.0f;
36         float v = 0.0f;
37         float d = 0.0f;
38         try
39         {
40             h = Input.GetAxis ("Horizontal");
41             v = Input.GetAxis ("Vertical");
42             d = Input.GetAxis ("Depth");
43         }
44         catch (UnityException)
45         {
46             Debug.Log ("Unable to read from one of input
47             axes: Horizontal, Vertical, Depth");
48         }
49
50         float step = moveSpeed;
```

```
47         if (realTime)
48     {
49         step *= Time.deltaTime;
50     }
51     transform.position += step * (h * transform.right +
52     ← d * transform.up + v * transform.forward);
53 }
54
55     private void processOrientationControls ()
56     {
57         float mx = 0.0f;
58         float my = 0.0f;
59         float fire = 0.0f;
60         try
61     {
62         mx = Input.GetAxis ("Mouse X");
63         my = Input.GetAxis ("Mouse Y");
64         fire = Input.GetAxis ("Fire1");
65     }
66     catch (UnityException)
67     {
68         Debug.Log ("Unable to read from one of input
69     ← axes: Mouse X, Mouse Y, Fire1");
70     }
71
72         float step = turnSpeed;
73         if (realTime)
74     {
75         step *= Time.deltaTime;
76     }
77         if (fire > 0.0f)
78     {
79             transform.rotation *= Quaternion.AngleAxis (step
80     ← * my, Vector3.right) * Quaternion.AngleAxis (step * mx,
81     ← Vector3.up);
82     }
83
84         private void processControls ()
85     {
86             if (manualMove)
87     {
88                 processPositionControls ();
89             }
90             if (manualRotate)
91     {
92                 processOrientationControls ();
93             }
94         }
95
96         // Wander in a circular (polygonal with n sides)
97         ← horizontal path
98         private void followPath ()
```

```

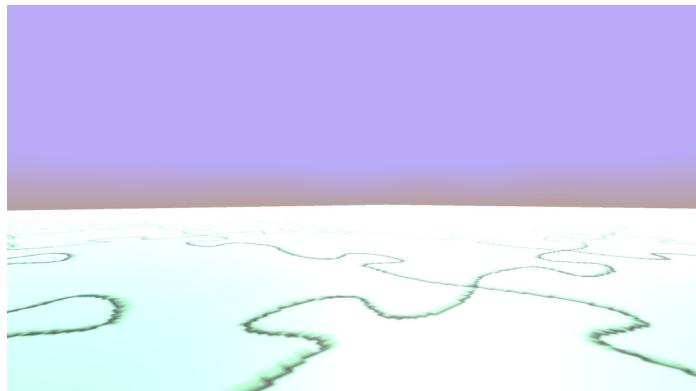
95
96     {
97         // Work around the angular position, based on
98         // angular speed * time.
99         float angle = pathSpeed * Time.time / pathRadius;
100
101        // Quantize the angle according to the number of
102        // sides available.
103        float quantAngle = (2.0f * Mathf.PI / pathSides) *
104        Mathf.Floor (angle * pathSides / (2.0f * Mathf.PI));
105
106        // Calculate the length of an edge of the inscribed
107        // polygon.
108        float edgeLength = 2.0f * pathRadius * Mathf.Sin
109        (Mathf.PI / pathSides);
110
111        // Work out how far the edge, assuming constant
112        // speed.
113        float distanceAlongEdge = edgeLength * (angle -
114        quantAngle) / (2.0f * Mathf.PI / pathSides);
115
116        // Find the position of the initial point on the
117        // edge (start vertex)
118        float x = pathRadius * Mathf.Sin (quantAngle);
119        float z = pathRadius * Mathf.Cos (quantAngle);
120        // Find direction of edge, based on the circle
121        // tangent at the midpoint.
122        float dx = pathRadius * Mathf.Sin (quantAngle +
123        Mathf.PI / pathSides);
124        float dz = pathRadius * Mathf.Cos (quantAngle +
125        Mathf.PI / pathSides);
126
127        // Set position and direction.
128        Vector3 forward = Vector3.Normalize (new Vector3
129        (dz, 0, -dx));
130        transform.position = new Vector3 (x, 0, z) +
131        distanceAlongEdge * forward;
132        transform.up = Vector3.up;
133        transform.forward = forward;
134    }
135
136    void Update () {
137        lastPosition = transform.position;
138
139        processControls ();
140
141        if (autoFollowPath)
142        {
143            followPath ();
144        }
145
146        velocity = 1.0f / Time.deltaTime *
147        (transform.position - lastPosition);
148    }

```

```

134
135     public Transform getLocation ()
136     {
137         return transform;
138     }
139
140     public Vector3 getVelocity ()
141     {
142         return velocity;
143     }
144 }
```

Figure 18.1.2: The prototype controller provides navigation facilities during development.



<sup>1</sup> Location usually required 6 numbers to define it relative to a particular coordinate system. Position requires 3 values, such as x, y, and z in a Cartesian coordinate system or , and in a spherical polar coordinate system. Orientation also requires 3 values, such as the angles relative to the x-, y- and z- axes in one of the Euler representations. The 4 values used in a quaternion includes 1 constraint on axis length when used to represent rotations, so also only has 3 degrees of freedom.

## 18.2 Tracking the user and objects to allow interaction

### Component 18.2.1: Location tracking component.

*Location*

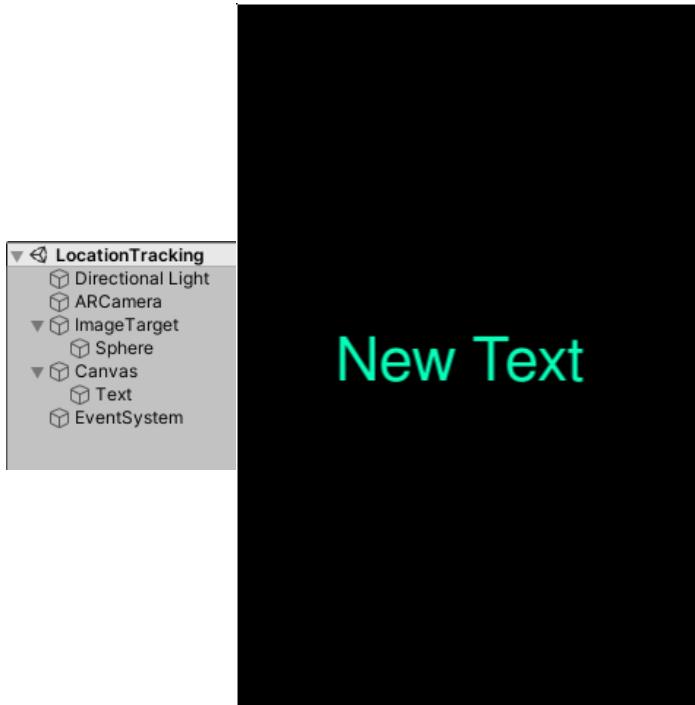
*Tracking*

*Previous examples have emphasized the value of knowing the location<sup>1</sup> of tracked markers in an augmented reality setting. More significantly, each of these examples are also effectively tracking the position of the participant (at least, that of their camera) relative to the markers during the process.*

This example deposits an object at regular intervals at the position of the camera device.

1. Create a new project, and set it up to use Android and Vuforia. In particular, enable Vuforia Augmented

Figure 18.2.1:  
Visual tracking  
of markers with  
feature points can  
be achieved using  
Vuforia.



Reality under the XR settings. It may be necessary to disable the Android TV compatibility setting (under Other Settings).

In newer versions of the Unity software, you will need to add in the Vuforia Engine AR package, using the Package Manager under the Windows menu option. If Vuforia becomes integrated with the XR systems, then you may also need to enable the XR Plug-in Management system under Project Settings.

2. Remove the Main Camera and replace it with an AR Camera (Create/Vuforia/AR Camera under the Hierarchy tab).
3. Retrieve a license key and database of target images from the Vuforia web site. Add these to the appropriate fields in the Vuforia configuration property of the AR Camera.
4. Add at least one image target to the scene. You may

also want to add a small 3D object as a child to this marker as a check see when the marker has been recognized. Ensure that the database and image target properties of the marker have been set appropriately.

5. Create a UI/Text element and add it to the scene. This is used to show the position of the camera, to confirm that the location tracking is working as expected. Increasing the font size makes it more readable, but may require that either the width and height of the text area be increased, or that horizontal and vertical overflow be enabled. The text may disappear completely otherwise.

Create a `DisplayLocation` C# script which can be attached to the AR Camera. This feeds the position and rotation fields to the text element. Use the code provided in Algorithm 18.2. Set the tracked element field to be the AR Camera, and the text output field to be the UI Text element.

Code Listing 18.2: `DisplayLocation`. A location monitoring script which continuously feeds the position and rotation of the defined object to the given text element.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class DisplayLocation : MonoBehaviour {
7
8      [Tooltip ("The element whose position and rotation are
9      ↵ reported.")]
10     public GameObject trackedElement;
11
12     [Tooltip ("The text element that will be updated.")]
13     public Text textOutput;
14
15     void Update () {
16         textOutput.text =
17             trackedElement.transform.position + "\n" +
18             trackedElement.transform.rotation;
19     }
20 }
```

6. Testing the application at this stage shows that the camera remains fixed at the origin, even when an im-

age target is identified. The following changes ensure that a world coordinate system is created and maintained:

- (a) Modify the AR Camera property: World Center Mode, to use the First Target identified as the origin for the world. This assume one of the markers (the first seen) is fixed relative to the world, and instead tracks the camera position.
  - (b) The Vuforia configuration has a Device Tracker section that can enable positional tracking, using a SLAM process together with visual inertial tracking to estimate camera motion. Switch this on so that tracking is maintained even when image targets leave the camera field of view.
7. To demonstrate how the camera device is now useful as a tracked element in the scene, we attach a breadcrumb script to it to mark the trail it follows.<sup>2</sup> This script creates a new virtual object every few seconds at the location of the camera device so that the trajectory can be shown explicitly. This script is provided in Algorithm 18.3. The script will need to be provided with a prefab which will be instantiated at each location when a virtual crumb is dropped.

Code Listing 18.3: Breadcrumb. The “breadcrumb” script which leaves markers at the position of the tracked object at regular intervals.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Breadcrumb : MonoBehaviour {
6
7      [Tooltip ("Time in seconds between drops")]
8      public float dropInterval = 0.1f;
9
10     [Tooltip ("Template for object to be dropped")]
11     public GameObject dropPrefab;
12
13     [Tooltip ("The object being tracked which provides
14     location for crumbs")]
15     public GameObject trackedObject;

```

<sup>2</sup> See the story of Hansel and Gretel. <http://www.gutenberg.org/ebooks/20748>

```

16 // amount of time since last drop.
17     private float timeInterval = 0.0f;
18
19     void Update () {
20         timeInterval += Time.deltaTime;
21         if (timeInterval > dropInterval)
22         {
23             timeInterval = 0.0f;
24             Instantiate (dropPrefab,
25             ↵ trackedObject.transform.position,
26             ↵ trackedObject.transform.rotation);
27         }
28     }
29 }
```

Test out the application. In particular explore the limitations of the tracking provided. Look out for cases where inertial trackers introduce errors and the coordinate frame starts to drift, and where lack of feature points reduce the effectiveness of the visual support.

Figure 18.2.2: An object appears on a marker. The marker can be printed onto paper, or displayed on the screen of a device.



### 18.3 Sensing the orientation of a device

#### Component 18.3.1: Gyroscope based orientation control.

Orientation Control
------------------------

*Motivation:* Sensor information from a device can be used both to manage behaviour of objects, and to support interaction and feedback with the participant (Chapter 19). Many existing toolkits include this component

directly when managing the participant's view through the camera and so this component is presented in this section for cases when technology enhanced objects can also make use of a similar information stream from a sensor.

Camera orientation control is provided automatically by many of the augmented reality toolkits.

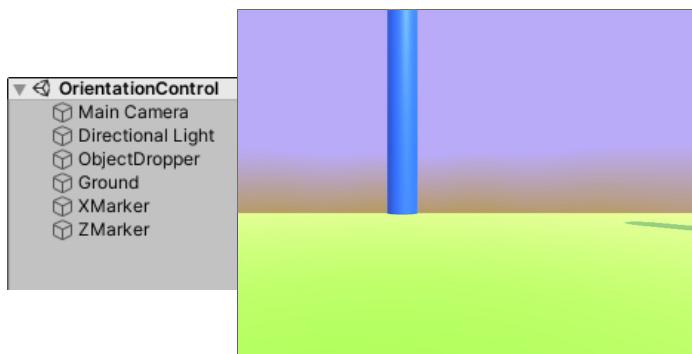


Figure 18.3.1:  
Sensor information  
used to drive  
virtual camera  
orientation.

1. Start with a new project. Set this up to run on a mobile device.
2. This example requires a virtual scene. Start with a ground plane, and place a few other elements to provide variety and, more importantly, visual reference points. The main camera needs to be located at about head height in the center of the scene since we are only controlling camera orientation and the position is fixed. A script to populate the scene with a fixed number of objects dropped randomly into it is provided in Algorithm 18.4 for your amusement.

Code Listing 18.4: `ObjectDropper`. Objects with random positions and colours are dropped into the scene starting from the specified starting point. The prefab used should have a rigidbody component with gravity enabled.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ObjectDropper : MonoBehaviour

```

```

6   {
7     public GameObject objectTemplate;
8
9     public Vector3 startPoint;
10
11    public int numberofObjects = 30;
12
13    public float initialSpeed = 1.0f;
14
15    public float timeInterval = 0.3f;
16
17    private float currentTime = 0.0f;
18
19    // Update is called once per frame
20    void Update()
21    {
22      currentTime += Time.deltaTime;
23
24      if ((currentTime > timeInterval) &&
25        (numberofObjects > 0))
26      {
27        currentTime = 0.0f;
28        numberofObjects--;
29        GameObject g = Instantiate (objectTemplate);
30        g.transform.position = startPoint;
31        g.GetComponent <Rigidbody> ().velocity =
32        Random.onUnitSphere * initialSpeed;
33        g.GetComponent <MeshRenderer>
34        ().material.color = Random.ColorHSV (0, 1, 0.5f, 1,
35        0.5f, 1);
36      }
37    }
38  }

```

3. Create two distinctive objects, in the direction of the positive x and z-axes respectively. This provides an opportunity to determine if these remain consistently aligned with reference points in the physical world.
4. Create a C# script named GyroTrack and add it to the camera. This is used to set the orientation of the camera to match that derived from the gyroscope. The script for this is relatively straightforward as shown in Algorithm 18.5. The only refinement required is to rotate the coordinate system returned by the gyroscope so that it is aligned with that used by the Unity software.

Code Listing 18.5: GyroTrack. Device orientation is provided by readings from the gyroscope.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class GyroTrack : MonoBehaviour
6  {
7      private Gyroscope gyro;
8      private bool gyroSupported;
9
10     // Start is called before the first frame update
11     void Start()
12     {
13         // ensure this code is only used if there is
14         // actually
15         // a gyroscope.
16         gyroSupported = SystemInfo.supportsGyroscope;
17         if (gyroSupported)
18         {
19             // Get hold of the system gyroscope.
20             gyro = Input.gyro;
21             // Switch it on.
22             gyro.enabled = true;
23         }
24     }
25
26     // Update is called once per frame
27     void Update()
28     {
29         if (gyroSupported)
30         {
31             // Map the gyro rotation into the same
32             // coordinate system as the unity camera.
33             transform.rotation = Quaternion.Euler (90, 0,
34             90) * gyro.attitude * Quaternion.Euler (180, 180, 0);
35         }
36     }
37 }

```

Experiment with this application and determine how reliable the gyroscope reading is. In particular look out for:

- Drift, where the virtual scene does not return to the same alignment with the physical world after (vigorous) movement.
- Relative versus absolute orientation where the position of the reference points differ once the application or device is restarted.

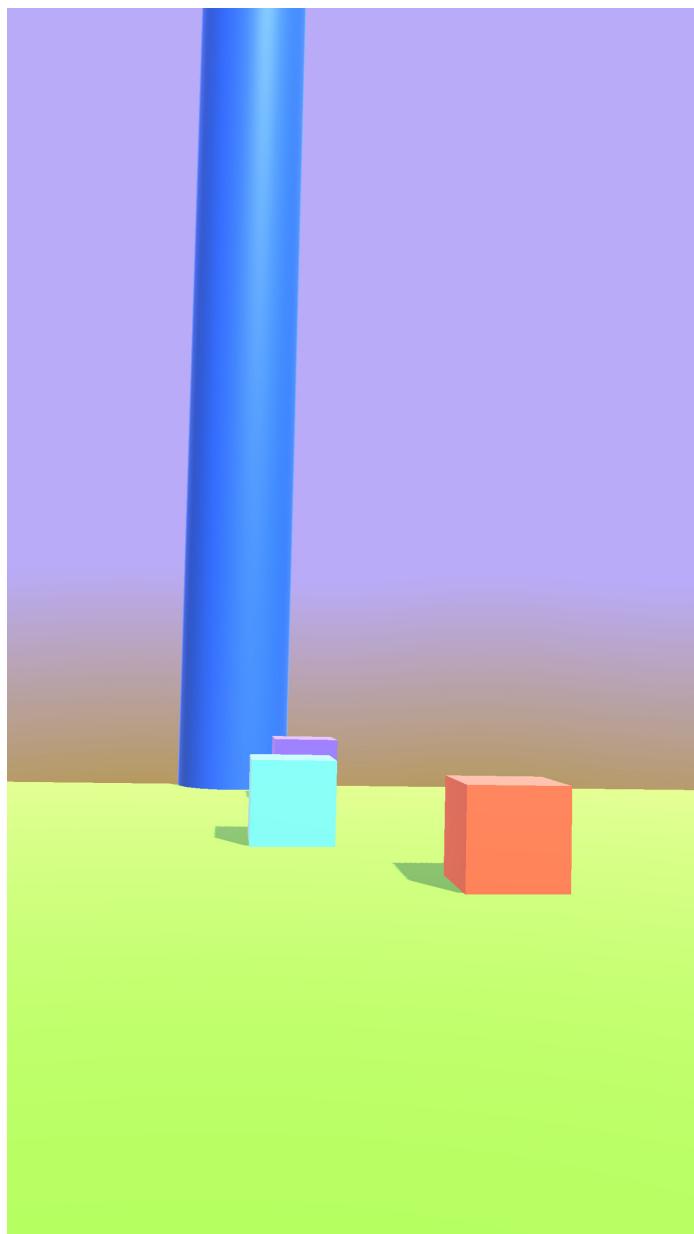


Figure 18.3.2: Scene orientation driven by the device's gyroscope.

**18.4  Synthetic buttons and dials for use in an augmented reality user interface**

**Component 18.4.1: Buttons and dials.**

*Buttons And  
Dials*

*Motivation:* This example creates interactable objects. In particular physical elements are given a concurrent virtual existence and are able to respond to physical interaction while triggering virtual actions.

The goal of this example is to demonstrate some ways in which interaction with virtual content can occur. This is can be done in several ways.

Physical interaction with the markers can be used to simulate button presses, for example. This mechanism is supported under Vuforia by detecting when a particular feature rich portion of the image is obscured. Further details are available at this link: <https://library.vuforia.com/content/vuforia-library/en/articles/Solution/How-To-Implement-Virtual-Buttons.html>. Another way that physical interaction with the markers can be used is by shifting or rotating the markers to mimic the behaviour of sliders or dials. This example supports the rotation of a marker to control the turning of a dial. However tracking of moving markers is currently not well supported with the toolkit used and so this motion can be rather slow and jerky.

Virtual interaction is another option. The device containing the camera can also be viewed as a controller by attaching a virtual pointer to it. This pointer exists in the same virtual space as the virtual content associated with any markers and so can interact directly with these markers through the use of collision detection in the virtual world. This can be used to push buttons, or rotate dials.

This example uses buttons and dials to control the colour of a virtual lamp light.

1. Set up the project to use AR Core. Switch the build

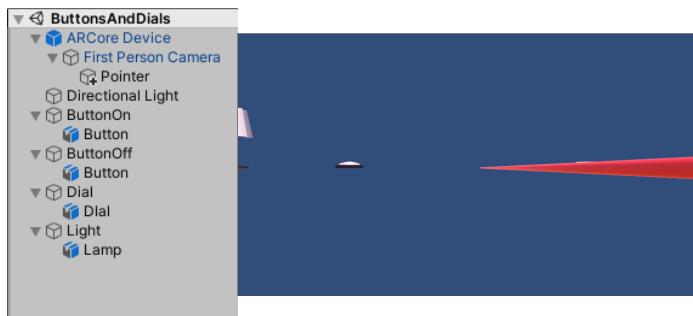


Figure 18.4.1:  
Physically interaction with virtual controls.

settings to use Android, and ensure that ARCore Supported is enabled under the XR Settings. Download the ARCore SDK for Unity and add it to the project.

2. Add a selection of marker images to the project. At least 4 are required for this example. Select the images, and under the Assets area, create a new target database (Create/GoogleARCore/AugmentedImageDatabase). Create a session configuration (Create/-GoogleARCore/SessionConfig) and add the database to the Augmented Image Database field. It may also help to set the camera focus mode to Auto.
3. Add an ARCore Device prefab to the scene and ensure that its Session Config field contains the newly created session configuration.
4. Having an 3D object associated with each target image is a useful facility to have. We achieve this by creating a new empty object for each of the elements used in this example. These include an on button, an off button, a dial, and a lamp. Attach the 3D object for each of these objects as a child object of the empty object. Then create a new C# script named ShowObjectOnMarker, and include the code provided in Algorithm 18.6. This scripts checks to see if the image target named is tracked, and if so, ensure a 3D object is made visible and positioned on the marker.

Code Listing 18.6: ShowObjectOnMarker. ARCore utility script to show an object at the position of a tracked marker.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Events;
5  using GoogleARCore;
6
7  [System.Serializable]
8  public class MarkerTrackEvent : UnityEvent <Vector3,
9    Quaternion> {}
10
11 public class ShowObjectOnMarker : MonoBehaviour
12 {
13     [Tooltip ("The name of the marker in the database")]
14     public string trackableName;
15     [Tooltip ("The object that is shown when the marker is
16     found")]
17     public GameObject asset;
18     [Tooltip ("The functions that are called while the
19     marker is tracked")]
20     public MarkerTrackEvent updatedHandler;
21
22     private Anchor anchor;
23
24     void Start ()
25     {
26         asset.SetActive (false);
27     }
28
29     void Update()
30     {
31         List<AugmentedImage> arImages = new List
32         <AugmentedImage> ();
33         Session.GetTrackables <AugmentedImage> (arImages,
34         TrackableQueryFilter.Updated);
35
36         foreach (AugmentedImage image in arImages)
37         {
38             if ((image.Name == trackableName) &&
39                 (image.TrackingState == TrackingState.Tracking ))
40             {
41                 if (anchor == null)
42                 {
43                     anchor =
44                     image.CreateAnchor(image.CenterPose);
45                 }
46                 asset.SetActive (true);
47                 updatedHandler.Invoke
48                 (anchor.transform.position, anchor.transform.rotation);
49                 break;
50             }
51         }
52     }
53 }
```

```

45
46     public void defaultTrackHandler (Vector3 p, Quaternion
47     ←   r)
48     {
49         asset.transform.position = p;
50         asset.transform.rotation = r;
51     }
}

```

Make sure the fields on the Show Object On Marker are filled in for each empty object. The name of the trackable has to correspond to the name used in the image database. The Asset that is shown when the marker is tracked must be supplied with the child object of the empty. The Updated Handler function should be linked (press the + symbol to create a new handler) to the defaultTrackHandler on the ShowObjectOnMarker component of the corresponding empty object.

5. The controls manipulate the colour of the lamp. Attach the script provided in Algorithm 18.7 to the lamp, and provide it with the material that controls the lamp colour. This script provides a number of public functions that can be called by scripts attached to the buttons and dial.

Code Listing 18.7: LightControls. Lamp colour is manipulated by providing a number of public functions that can respond to click events, or to more complex dial operations that provide colour directly.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class LightControls : MonoBehaviour
6  {
7      public Material lampMaterial;
8
9      public void setColour (Color col)
10     {
11         lampMaterial.color = col;
12     }
13
14     public void lightOff ()
15     {
16         setColour (new Color (0, 0, 0));
}

```

```

17     }
18
19     public void lightOn ()
20     {
21         setColour (new Color (1, 1, 0.2f));
22     }
23 }
```

6. Create a pointer attached to the camera. Start with a cylinder object that is attached as a child of the First Person Camera (below the ARCore Device) and scaled to the dimensions of a pointer (0.01, 1, 0.01). Rotate it by 90 degrees about the x-axis to ensure that it is facing forwards. It may also help to offset it by 0.1 in the x direction so that it is visible (you are not staring directly down the inside of the tube). Add a rigidbody component to this pointer so that it will cause collision events. Make sure gravity is switched off, and Is Kinematic is switched on.
7. Add a component to each button object to respond to collision events, as shown in Algorithm 18.8. For each button create a new Collision Handler call by clicking on the + symbol to make a new entry. Then drag the lamp object onto the object field, and select the lightOn (and lightOff) functions from the LightControls script to be called when the collision event is triggered. Make sure each button has a collider such as a Box Collider with the Is Trigger set to true so that it generates the OnTriggerEnter event.

Code Listing 18.8: CollisionCallback. This function calls a user defined handler whenever a collision event is triggered.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Events;
5
6  public class CollisionCallback : MonoBehaviour
7  {
8      public UnityEvent collisionHandler;
9
10     private void OnTriggerEnter(Collider other)
11     {
```

```

12         Debug.Log("Button active");
13         collisionHandler.Invoke ();
14     }
15 }
```

8. The dial is controlled in a similar fashion. When the collision occurs, the script attached to the dial works out the closest intersection point and rotates the dial to face that direction. At the same time, the update function monitors the orientation of the dial and uses the value to determine a hue which it sends to the lamp. This means that the dial works both when controlled via collision with the pointer, but also through the rotation of the physical marker.

The script for the dial is provided in Algorithm 18.9. This needs to be attached to the dial object, which also needs a collider which generates trigger events on collision. The dial control should be connected to the lamp's setColour function via the Collision Handler property. The Updated Handler on the empty object that is the parent for the dial needs to be updated (replace the default handler) with the dialTrackHandler on the DialControl component of the dial object. This integrates the orientation of the marker, with any angle resulting from collisions.

Code Listing 18.9: `DialControl`. The dial sets its orientation to place the marked point as close as possible to the point of intersection with a colliding object. The orientation of the dial then drives the colour change operation it performs.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Events;
5
6  [System.Serializable]
7  public class ColourChangeEvent : UnityEvent <Color> {}
8
9  public class DialControl : MonoBehaviour
10 {
11     public ColourChangeEvent collisionHandler;
12
13     private float angle = 10.0f;
```

```

14     private Vector3 direction;
15     private Quaternion baseRotation;
16
17     public void dialTrackHandler (Vector3 p, Quaternion r)
18     {
19         baseRotation = r;
20         transform.position = p;
21         transform.rotation = baseRotation;
22         transform.rotation = Quaternion.AngleAxis (angle,
23             transform.up) * transform.rotation;
24
25         ShowUpdate ();
26     }
27
28     private float getRotationAngle (Vector3 direction)
29     {
30         return Mathf.Atan2 (Vector3.Dot (direction,
31             baseRotation * Vector3.right), Vector3.Dot (direction,
32             baseRotation * Vector3.forward)) * Mathf.Rad2Deg;
33     }
34
35     private void OnTriggerStay(Collider other)
36     {
37         direction = other.ClosestPoint (transform.position)
38         - transform.position;
39         // Force rotation about the y (up) axis.
40         angle = getRotationAngle (direction);
41         dialTrackHandler (transform.position, baseRotation);
42     }
43
44     public void ShowUpdate ()
45     {
46         // assumes up is y axis.
47         float hue = Mathf.Atan2 (transform.forward.z,
48             transform.forward.x) / (Mathf.PI * 2.0f);
49         Color c = Color.HSVToRGB (hue, 1, 1);
50         collisionHandler.Invoke (c);
51
52     }
53 }
```

## 18.5 Identifying and recognising physical objects

### Component 18.5.1: Visual recognition.

<i>Visual Recognition</i>
-------------------------------

This example serves two purposes. It demonstrates the

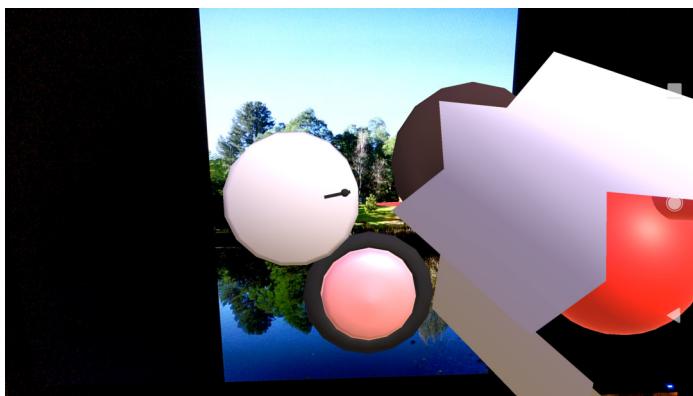
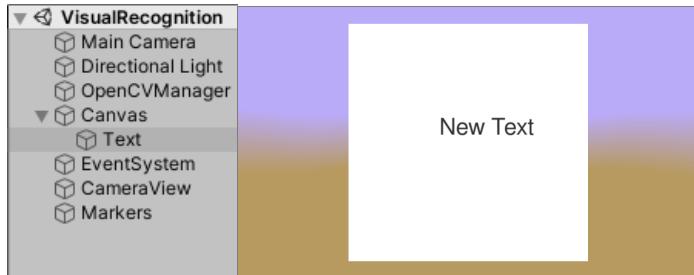


Figure 18.4.2:  
Buttons and dials.

ability to identify and locate physical world objects in the camera's view. Secondly it also employs an external software libraries and integrates this with the development environment provided by the Unity software.

Note: The instructions provided are sufficient to build basic OpenCV native applications, but may give errors with some of the neural network functions when compiled using Visual Studio. The recommended approach is to compile the native plugins under Linux using the instructions provided below for this purpose.

Figure 18.5.1: Using the camera to recognise objects in the physical setting.



1. Start with a new project. Set this up to export to a mobile device.

The example captures an image from the physical camera on the device, passes this off to a recognition process implemented in an external library, and retrieves the results. These results are then presented as a visual overlay on the original image.

The first steps of this example set up the scene in the Unity software.

Create a plane object to display the image from the physical camera. A rotation of 90, 180, 0 aligns it perpendicularly to the scene camera and a distance of 10 units from the scene camera ensures that it fills most of the view frustum. Leave the scale settings on the plane unchanged as later components rely on this to position annotations relative to the image. Create a new material and apply it to the plane. Also create a script (similar to that used in component 16.1.1, see Algorithm 18.10) to feed the physical camera feed to a texture, and to attach that texture to the material applied to the plane. It is possible to test this aspect on a desktop platform with a web camera attached.

Code Listing 18.10: PlayCamera. Images from the physical camera are applied to the material provided as the CamTexMaterial property.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PlayCamera : MonoBehaviour {
6
7      public Material camTexMaterial;
8
9      private WebCamTexture webcamTexture;
10
11     // Update is called once per frame
12     void Update () {
13         if (webcamTexture == null)
14         {
15             webcamTexture = new WebCamTexture ();
16             camTexMaterial.mainTexture = webcamTexture;
17         }
18         if (!webcamTexture.isPlaying)
19         {
20             webcamTexture.Play ();
21         }
22     }
23 }
```

A UI Text element can also be created. This is used mainly for debugging purposes to monitor the status of information being provided by the external library.

2. Regions of interest identified by the recognition process are annotated visually above the image on the plane.

A prefab to represent such an annotation can be constructed from a standard cube, and a 3D text object (make this a child of the cube) which is placed slightly in front of the cube. A transparent material should be created and applied to the cube so that the image content responsible for the annotation is still visible through the marker.

Markers are placed under their own parent object, so create a new empty object named Markers for this purpose.

3. The recognition process is handled on a new empty object named OpenCVManager. This interfaces with the external library: OpenCV (computer vision) and uses facilities from this package to perform the recognition of objects.

Create a new C# script called AccessOpenCV. The contents of this script are provided in Algorithm 18.11, but the process is broken down into smaller sections in the following steps.

Code Listing 18.11: AccessOpenCV. Visual recognition of objects is performed by calling an external library.

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Runtime.InteropServices;
4  using UnityEngine.UI;
5  using System.IO;
6  using System;
7
8  public class AccessOpenCV : MonoBehaviour
9  {
10    public Material cameraMaterial;
11
12    public GameObject markerTemplate;
13    public GameObject markerParent;
14
15    private bool modelReady = false;
16
17    private float delayTime = 0.0f;
18
19    public Text text;
```

```

20
21     private string [] CLASSES = { "background", "aeroplane",
22     ↪ "bicycle", "bird", "boat", "bottle", "bus", "car",
23     ↪ "cat", "chair", "cow", "diningtable", "dog", "horse",
24     ↪ "motorbike", "person", "pottedplant", "sheep", "sofa",
25     ↪ "train", "tvmonitor" };

26
27     [DllImport("VisualRecognition")]
28     private static extern void prepareModel (string dirname);
29
30     [DllImport("VisualRecognition")]
31     private static extern int doRecognise (byte []
32     ↪ imageData, int width, int height);
33
34     [DllImport("VisualRecognition")]
35     private static extern void retrieveMatch (int i, ref int
36     ↪ category, ref float confidence, ref float sx, ref
37     ↪ float sy, ref float ex, ref float ey);
38
39     void Start ()
40     {
41         StartCoroutine (prepareModel ());
42     }
43
44     IEnumerator prepareModel ()
45     {
46         yield return StartCoroutine(extractFile ("",
47         ↪ "MobileNetSSD_deploy.caffemodel"));
48         yield return StartCoroutine(extractFile ("",
49         ↪ "MobileNetSSD_deploy.prototxt"));

50         prepareModel (Application.persistentDataPath);

51         modelReady = true;
52     }
53
54     private void clearVisuals ()
55     {
56         foreach (Transform child in markerParent.transform)
57         {
58             GameObject.Destroy(child.gameObject);
59         }
60     }
61
62     private void addVisual (string name, float confidence,
63     ↪ float sx, float sy, float ex, float ey)
64     {
65         GameObject g = GameObject.Instantiate (markerTemplate);
66         g.transform.position = new Vector3 (5.0f * (sx + ex) -
67         ↪ 5.0f, -5.0f * (sy + ey) + 5.0f, 0);
68         g.transform.localScale = new Vector3 (10.0f *
69         ↪ Mathf.Abs (sx - ex), 10.0f * Mathf.Abs (sx - ex), 1);

```

```
61      g.GetComponentInChildren <TextMesh> ().text = name +
62      " \n" + confidence;
63      g.transform.SetParent(markerParent.transform, false);
64    }
65
66    void Update ()
67    {
68      delayTime += Time.deltaTime;
69
70      if (modelReady && (delayTime > 2.0f))
71      {
72        clearVisuals ();
73        delayTime = 0.0f;
74
75        Texture2D image = new Texture2D
76        (cameraMaterial.mainTexture.width,
77         cameraMaterial.mainTexture.height,
78         TextureFormat.RGBA32, false);
79        RenderTexture renderTexture = new
80        RenderTexture(cameraMaterial.mainTexture.width,
81        cameraMaterial.mainTexture.height, 32);
82        Graphics.Blit(cameraMaterial.mainTexture,
83        renderTexture);
84        RenderTexture.active = renderTexture;
85        image.ReadPixels (new Rect(0, 0,
86        renderTexture.width, renderTexture.height), 0, 0);
87        image.Apply();
88
89        int numMatch = doRecognise (image.GetRawTextureData
90        (), image.width, image.height);
91
92        text.text = "Matches: " + numMatch;
93
94        for (int i = 0; i < numMatch; i++)
95        {
96          int category = -1;
97          float confidence = 0.0f;
98          float sx = 0, sy = 0, ex = 0, ey = 0;
99          retrieveMatch (i, ref category, ref confidence,
100          ref sx, ref sy, ref ex, ref ey);
101          if (confidence > 0.2f)
102          {
103            Debug.Log ("Match: " + category + " " +
104            confidence + " " + sx + " " + sy + " " + ex + " " +
105            ey);
106            addVisual (CLASSES[category], confidence, sx,
107            sy, ex, ey);
108          }
109        }
110      }
111    }
```

```

101 // Copy file from the android package to a
102 // readable/writeable region of the host file system.
103 IEnumerator extractFile (string assetPath, string
104 assetFile)
105 {
106     // Source is the streaming assets path.
107     string sourcePath = Application.streamingAssetsPath +
108     "/" + assetPath + assetFile;
109     if ((sourcePath.Length > 0) && (sourcePath[0] == '/'))
110     {
111         sourcePath = "file://" + sourcePath;
112     }
113     string destinationPath =
114     Application.persistentDataPath + "/" + assetFile;
115
116     // Files must be handled via a WWW to extract.
117     WWW w = new WWW (sourcePath);
118     yield return w;
119     try
120     {
121         File.WriteAllBytes (destinationPath, w.bytes);
122     }
123     catch (Exception e)
124     {
125         Debug.Log ("Issue writing " + destinationPath + " "
126         + e);
127     }
128     Debug.Log (sourcePath + " -> " + destinationPath + " "
129     + w.bytes.Length);
130 }
131 }
```

4. The AccessOpenCV component requires a number of properties:
  - (a) The material that contains the webCamTexture and that is also applied to the preview plane.
  - (b) The prefab used to show annotations, and the empty object that serves as parent object to the active annotations.
  - (c) The UI.Text element for any debugging information.
5. Access to functions in an external library needs to be handled carefully. This is particularly necessary when using multiple languages. The code in the Unity software project uses C#, but that in the external library

uses C and C++. Some care has to be taken when passing parameters between a language like C# which manages memory for the programmer and hides the existence of pointers, and languages such as C/C++ where the opposite occurs.

A limited set of functions is exposed by the external library, and must be mapped into equivalent functions in C#. The three functions used in this example are all accessed from a (library) file called libVisualRecognition.so. As show in the code below, the library is referenced by leaving out the initial “lib” and trailing “.so”. The functions are marked as being provided externally.

```

1  [DllImport("VisualRecognition")]
2  private static extern void prepareModel (string
   dirname);
3
4  [DllImport("VisualRecognition")]
5  private static extern int doRecognise (byte []
   imageData, int width, int height);
6
7  [DllImport("VisualRecognition")]
8  private static extern void retrieveMatch (int i,
   ref int category, ref float confidence, ref
   float sx, ref float sy, ref float ex, ref
   float ey);

```

---

6. The visual recognition process employs a number of operations that are common to other uses of the OpenCV library. Some preparation has to be done, involving reading in a large data set into memory. The current image is passed to OpenCV and the image processing operation is performed. Finally the results are read back to the Unity software side of the application and used to support the augmented reality experience.

The data used by the visual recognition process is a previously trained model that is able to recognise a particular set of objects. The model is available at <https://github.com/chuanqi305/MobileNet-SSD>. The two files required are the MobileNetSSD\_deploy.caffemodel (on the deploy link in the README.md file) and the MobileNetSSD\_deploy.prototxt (in the voc

folder, make sure you download the raw file).

The data files used are stored in the StreamingAssets folder of the Unity software project. These files are accessible in a mobile application but with some limitations. They have been packaged into the application bundle and so are not modifiable, nor are they easily accessed through standard file operations.

The first step is thus to unpack these files into a readable and writable area of the file system on the mobile device. This makes it easier for generic code, such as that used to perform the image processing, to access the files. The function extractFile (Algorithm 18.11) uses a Unity software WWW object to read from the package format and to write to the Unity software directory defined by Application.persistentDataPath which is an available scratch directory.

Once the two large data files have been copied the first of the external functions (prepareModel) is invoked, and is provided with the directory where the files can be found.

---

```

1     yield return StartCoroutine(extractFile ("",
2         MobileNetSSD_deploy.caffemodel"));
3     yield return StartCoroutine(extractFile ("",
4         MobileNetSSD_deploy.prototxt"));
      prepareModel (Application.persistentDataPath);

```

---

7. The image processing operation can reuse the data files that have been extracted, and so the Update function only needs to call doRecognise with the physical camera image, and display the results.

---

```

1 int numMatch = doRecognise (image.GetRawTextureData
 (), image.width, image.height);

```

---

The camera image needs to be copied into a block of memory that OpenCV can access. This does require a rather involved process of taking the image as a Texture, converting it into a RenderTexture and then finally

into a Texture2D. The Texture2D class has the GetRawTextureData function which returns the byte array representing the image in memory. The concept of a pointer to this array is shared among all the languages involved and so is suitable for transferring the image to the external library.

8. Each region recognized is returned as a set of values:
  - (a) A number identifying the object identified. This is turned into a human readable string by looking it up in the CLASSES array.
  - (b) A number representing the confidence associated with the recognition, from 0 to 1.
  - (c) The 2D box representing the region containing the recognized object, represented using opposite corners:  $(s_x, s_y)$  and  $(e_x, e_y)$ .

---

```
1 retrieveMatch (i, ref category, ref confidence, ref
    sx, ref sy, ref ex, ref ey);
```

---

The box values are used to position and scale a marker prefab to fit into the appropriate region of the image.

9. The three functions accessed by the Unity software are written in C/C++ and in turn make calls to specific functions in the OpenCV library. Plentiful examples exist of how to perform various image processing operations, and this one is based on object detection processes such as those described in <https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/> and <http://www.ebenezertechs.com/mobilenet-ssd-using-opencv-3-4-1-deep-learning-module-python/>.

The full code for these functions is provided in Algorithm 18.12 but relevant sections of each are discussed below.

Code Listing 18.12: VisualRecognition. The native code that uses the OpenCV library to perform objection detection and recognition.

```

1 #include "VisualRecognition.h"
2 #include <opencv2/opencv.hpp>
3 #ifdef ANDROID
4 #include <android/log.h>
5 #endif
6
7 extern "C"
8 {
9     // The recognition model, created during prepareModel.
10    cv::dnn::Net net;
11
12    // The details of detected matches from the last round
13    // of doRecognise.
14    cv::Mat detections;
15
16    void prepareModel (char * dirname)
17    {
18        const char * protoFile =
19            "MobileNetSSD_deploy.prototxt.txt";
20        const char * modelFile =
21            "MobileNetSSD_deploy.caffemodel";
22        #ifdef ANDROID
23            __android_log_print (ANDROID_LOG_ERROR, "Unity",
24            "Unity Opening file: %s %s\n", protoFile, dirname);
25        #endif
26        net = cv::dnn::readNetFromCaffe (std::string (dirname)
27            + std::string ("/") + std::string (protoFile),
28            std::string (dirname) + std::string ("/") +
29            std::string (modelFile));
30
31        std::vector<cv::String> n = net.getLayerNames ();
32        for (cv::String s : n)
33        {
34            #ifdef ANDROID
35                __android_log_print (ANDROID_LOG_ERROR, "Unity",
36                "Layer %s\n", s.c_str ());
37            #else
38                printf ("Layer %s\n", s.c_str ());
39            #endif
40        }
41    }
42
43    int doRecognise (char * imageData, int width, int height)
44    {
45        std::string CLASSES [] = {"background", "aeroplane",
46            "bicycle", "bird", "boat",
47            "bottle", "bus", "car", "cat", "chair", "cow",
48            "diningtable",
49            "dog", "horse", "motorbike", "person",
50            "pottedplant", "sheep",
51            "sofa", "train", "tvmonitor" };
52    }

```

```

43         #ifdef ANDROID
44             __android_log_print (ANDROID_LOG_ERROR, "Unity",
45             "Image data %p %d %d\n", imageData, width, height);
46         #endif
47         //     const char * imageFile = "example_01.jpg";
48
49         //     cv::Mat image = cv::imread(std::string
50         // (dirname) + std::string ("") + std::string
51         // (imageFile));
52         //     cv::Mat image = cv::imread(imageFile);
53         cv::Mat rawimage = cv::Mat (height, width, CV_8UC4,
54         imageData);
55         cv::Mat image;
56         cv::cvtColor (rawimage, image, cv::COLOR_RGBA2BGR,
57         3);
58         cv::flip (image, image, 0);
59         int h = image.rows;
60         int w = image.cols;
61         //     imwrite ("/storage/emulated/0/Android/data/vi
62         sualrecognition.olde.grim/files/test2.jpg",
63         image);
64         #ifdef ANDROID
65             __android_log_print (ANDROID_LOG_ERROR, "Unity",
66             "Image %d %d\n", h, w);
67         #else
68             printf ("Image %d %d\n", h, w);
69         #endif
70         cv::Mat resized;
71         cv::resize (image, resized, cv::Size(300, 300));
72         #ifdef ANDROID
73             __android_log_print (ANDROID_LOG_ERROR, "Unity",
74             "Image2 %d %d\n", image.rows, image.cols);
75         #else
76             printf ("Image2 %d %d\n", image.rows, image.cols);
77         #endif
78         cv::Mat blob = cv::dnn::blobFromImage (resized,
79         0.007843, cv::Size(300, 300), cv::Scalar(127.5));
80         #ifdef ANDROID
81             __android_log_print (ANDROID_LOG_ERROR, "Unity",
82             "Blob %d %d %d\n", blob.size ().height, blob.size
83             ().width, blob.total ());
84         #else
85             printf ("Blob %d %d %d\n", blob.size ().height,
86             blob.size ().width, blob.total ());
87         #endif
88
89         net.setInput(blob);
90         detections = net.forward();
91
92         return detections.size[2];
93     }

```

```

82     void retrieveMatch (int i, int & category, float &
83         confidence, float & sx, float & sy, float & ex, float
84         & ey)
85     {
86         int pos [4] = { 0, 0, 0, 0 };
87         pos[2] = i;
88
89         pos[3] = 1; category = detections.at <float> (pos);
90         pos[3] = 2; confidence = detections.at <float> (pos);
91         pos[3] = 3; sx = detections.at <float> (pos);
92         pos[3] = 4; sy = detections.at <float> (pos);
93         pos[3] = 5; ex = detections.at <float> (pos);
94         pos[3] = 6; ey = detections.at <float> (pos);
95     }
96 }
```

10. The interface with the Unity software requires that the functions exposed use internal naming formats consistent with the C language. This is achieved by wrapping them in:

---

```

1  extern "C"
2  {
3  }
```

---

11. The prepareModel function creates the OpenCV data structure that contains the data for identifying objects.

---

```

1  void prepareModel (char * dirname)
2  {
3      const char * protoFile = "MobileNetSSD_deploy.
4          prototxt.txt";
5      const char * modelFile = "MobileNetSSD_deploy.
6          caffemodel";
7      #ifdef ANDROID
8          __android_log_print (ANDROID_LOG_ERROR, "Unity",
9              "Unity Opening file: %s %s\n", protoFile,
10             dirname);
11      #endif
12      net = cv::dnn::readNetFromCaffe (std::string (
13          dirname) + std::string ("/") + std::string (protoFile),
14          std::string (dirname) + std::string ("/") + std::string (modelFile));
15  }
```

---

This function needs to be provided with the path to the folder containing the data files. As mentioned previously, these are not easily available in the packages on

mobile devices and so have previously been extracted to a directory where standard file operations apply.

The OpenCV function `readNetFromCaffe` is invoked to parse the files.

Debugging is a significant part of developing applications that use external native library. The `__android_log_print` function provides one way to communicate information from the native library. This is written to the android log on the device, and can be accessed through tools that show this log file (typically logcat applications, or programs such as adb on the host computer).

12. The recognition process is relatively simple thanks to the facilities provided by OpenCV. Most of the code provided supports checking of progress which is quite important when porting applications to new platforms.

The essential elements are outlined below.

```

1  int doRecognise (char * imageData, int width, int
                  height)
2  {
3      cv::Mat rawimage = cv::Mat (height, width,
                               CV_8UC4, imageData);
4      cv::Mat image;
5      cv::cvtColor (rawimage, image, cv::
                     COLOR_RGBA2BGR, 3);
6      cv::flip (image, image, 0);
7      int h = image.rows;
8      int w = image.cols;
9      cv::Mat resized;
10     cv::resize (image, resized, cv::Size(300, 300)
                  );
11     cv::Mat blob = cv::dnn::blobFromImage (resized,
12                                             0.007843, cv::Size(300, 300), cv::Scalar
13                                             (127.5));
14     net.setInput(blob);
15     detections = net.forward();
16 }
```

---

The image data provided by the Unity software needs some transformations before recognition can take place. It needs to be converted from a 4 channel RGBA format into the 3 channel BGR format expected by OpenCV. It also comes in upside down and needs to be flipped

vertically. It is then resized before the neural network is applied to it.

13. The process for retrieving the individual data values is provided for completeness.

```

1 void retrieveMatch (int i, int & category, float &
    confidence, float & sx, float & sy, float & ex,
    float & ey)
2 {
3     int pos [4] = { 0, 0, 0, 0 };
4     pos[2] = i;
5
6     pos[3] = 1; category = detections.at <float> (
7         pos);
8     pos[3] = 2; confidence = detections.at <float> (
9         pos);
10    pos[3] = 3; sx = detections.at <float> (pos);
11    pos[3] = 4; sy = detections.at <float> (pos);
12    pos[3] = 5; ex = detections.at <float> (pos);
13    pos[3] = 6; ey = detections.at <float> (pos);
14 }
```

---

Relevant points include the difference between the format for a reference parameter (ref in C#, & in C++).

The list of matches is returned as a 4D matrix with each matrix element being a list of 7 values. This requires some effort in addressing the required values.

14. The details provided to this point outline how image processing is performed (easily, thanks to the OpenCV library, and trained neural network), and how access to the native library is achieved. The final step is to compile the native library and link it in with the Unity software project. The goal is to produce the libVisu-alRecognition.so and place it in the Plugins/Android directory in the project assets.
15. Download the pre-compiled version of the OpenCV (computer vision) library for android from <https://opencv.org/releases.html>. Make folders: Standard Assets/OpenCV in the directory below the Assets folder and copy the contents into this. This part of the project does not need to be accessible to Unity.

Create folders in the assets area called: Plugins / Android. The OpenCV library is compiled together with any additional native code into a shared library for an ARM based android device and placed here. Copy the libopencv\_java4.so file into this folder as well (from the sdk/native/libs/armeabi-v7a directory).

16. The various C/C++ files can be compiled in various ways. The most straightforward is to use a Linux platform. An instruction such as shown below can then build the required library file:

```
1 armv7a-linux-androideabi24-clang -isystem ~android-
  sdks/ndk-bundle/sysroot/usr/include/arm-linux-
  androideabi -shared -DANDROID -I~android-sdks/
  ndk-bundle/toolchains/llvm/prebuilt/linux-x86_64
  /sysroot/usr/include/c++/v1/ -L../../Standard
  Assets/OpenCV-android-sdk/sdk/native/libs/
  armeabi-v7a/ -fPIC -Wl,-soname,
  libVisualRecognition.so -o Android/
  libVisualRecognition.so VisualRecognition.cpp -I
  . -I../../Standard Assets/OpenCV-android-sdk/sdk/
  /native/jni/include -Wl,--whole-archive -
  libopencv_java4 -Wl,--no-whole-archive
```

---

This does assume that the Android SDK and NDK are installed in appropriate directories.

It is theoretically possible to produce this file on other platforms. Those that support Visual Studio can try out the remainder of the instructions.

17. Start Visual Studio and create a new project in the Standard Assets folder called VisualRecognition. The template should be chosen from the Visual C++/Cross Platform/Android/Dynamic Shared library option. Add in the VisualRecognition C++ files as shown previously.

Under the Project/Properties menu, switch to “All Configurations” under the Configuration box, and select ARM as the Platform.

Under the VC++ directories, add to the Include Directories: \$(ProjectDir)..\..\..\OpenCV\include\

(check this does find the OpenCV folder created previously). Add to the Additional Library Directories under Linker/General/: \$(ProjectDir)\..\..\..\OpenCV\libs\armeabi-v7a\. Set the Output Directory to write directly to the Plugins/Android folder by using: \$(SolutionDir)..\..\Assets\Plugins\Android\. Under Linker/Input/Additional Dependencies, add opencv\_java4 (make sure this matches the version of OpenCV you are using).

Depending on the version of OpenCV you may also need to set the C++/Language/C++ Language Standard to support C++ 11. Change the Solution Platforms setting at the top of the Visual Studio window to ARM as well.

Also in project properties, to add further OpenCV libraries:

- (a) C++/Language/Enable Run-Time Information: Yes (-frtti)
- (b) C++/Code Generation/Enable C++ Exceptions: Yes (-fexceptions)

Use Build/BuildSolution to confirm that the project builds and creates a libVisualRecognition.so in the Plugins/Android folder in the Unity project.

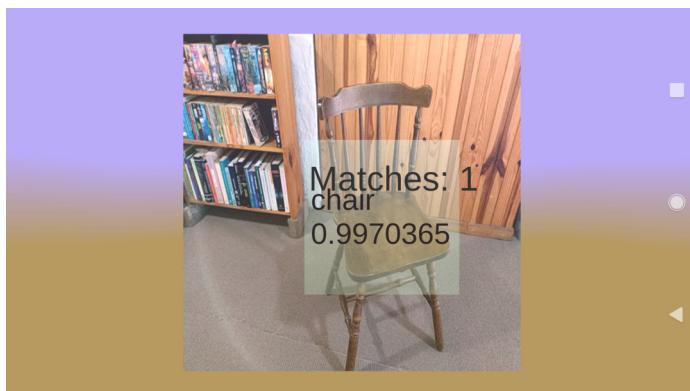


Figure 18.5.2:  
Tracking objects in  
the location.

## 18.6 *Capturing the structure of objects in the scene using structured light*

### Component 18.6.1: Using phase shift structured light to

#### capture objects and scenes

Structured  
Light

*A structured light scanner usually requires a projector to display patterns that are then projected onto the surface being scanned, and a separate camera to capture images of the patterned surface. In an effort to make this component more accessible this example uses virtual projectors and cameras within a simulated Unity environment. There is sufficient functionality in this component that would allow it to be applied to the physical world but that step is left as an exercise to the reader. A benefit of the virtual setting is that some of the subtleties of the process can be explored without the variations in image quality, lighting and noise effects that would be present in the physical setting. There are sufficient challenges in phase unwrapping that are already adequate visible in the virtual environment.*

1. Start with a new Unity project. The first thing to do is to set up a scene to scan. You can experiment later with the scene complexity but for a starting point a few primitive objects are sufficient. Include a floor plane, and a back wall plane to ensure that all visible points are covered by a surface that can reflect light (that will be provided by the projector). Add a few spheres and cubes to provide contiguous surfaces in different areas of the scene. Finally, make sure the camera can see all of these details.
2. By default, there will be a light already in the scene. This will become the projector in the scene. You may want to rename it to 'ProjectedLightSource' to help remember this. Change the type of light to be 'Spot' (rather than directional). This ensures that the light has a clear direction and cone of projection. Pose the light so that the spot illuminates the visible portion of

the scene. You may need to expand the spot angle to ensure adequate coverage.

Projecting a pattern from the light source can be done using the ‘Cookie’ parameter. However, we want to project a very specific pattern and one that is more convenient to generate procedurally. Procedural generation will also allow us to easily vary properties of the projected pattern and investigate the effect that these properties have.

The effect of lighting is actually achieved when calculating illumination on the surfaces in the scene. The pattern is thus created in a surface shader, which is associated with a material applied to each object in the scene.

Create a shader called StructuredLightShader using the code provided in Algorithm 18.13.

Code Listing 18.13: StructuredLightShader. The shader computes the surface illumination as if illuminated by a structure light pattern being projected by the light source.

```

1 // Generate a periodic pattern projected from the light
2   → position.
3 Shader "Custom/StructuredLightShader"
4 {
5   Properties
6   {
7     _Color ("Color", Color) = (1,1,1,1)
8     _MainTex ("Albedo (RGB)", 2D) = "white" {}
9     _Glossiness ("Smoothness", Range(0,1)) = 0.5
10    _Metallic ("Metallic", Range(0,1)) = 0.0
11    _Frequency ("Frequency", float) = 1.0
12    _Phase ("Phase", float) = 3.14159
13 }
14 SubShader
15 {
16   Tags { "RenderType"="Opaque" }
17   LOD 200
18
19   CGPROGRAM
20     // Physically based Standard lighting model, and
21   → enable shadows on all light types
22     //##pragma surface surf Standard fullforwardshadows
23     #pragma surface surf SimpleSpecular vertex:vert
24
25     // Use shader model 3.0 target, to get nicer
26   → looking lighting

```

```

24         #pragma target 3.0
25
26         sampler2D _MainTex;
27
28         struct Input
29     {
30             float2 uv_MainTex;
31             float3 PatternPos;
32         };
33
34         struct SurfaceOutputCustom {
35             fixed3 Albedo;
36             fixed3 Normal;
37             fixed3 Emission;
38             half Metallic;
39             half Smoothness;
40             half Specular;
41             fixed Alpha;
42             float3 worldPos;
43         };
44
45             half _Glossiness;
46             half _Metallic;
47             fixed4 _Color;
48             float _Frequency;
49             float _Phase;
50
51         void vert (inout appdata_full v, out Input o) {
52             UNITY_INITIALIZE_OUTPUT(Input,o);
53             o.PatternPos = mul (unity_ObjectToWorld,
54             v.vertex);
55             }
56
57             half4 LightingSimpleSpecular (SurfaceOutputCustom
58             s, half3 lightDir, half3 viewDir, half atten) {
59                 half3 h = normalize (lightDir + viewDir);
60
61                 half diff = max (0, dot (s.Normal, lightDir));
62
63                 float nh = max (0, dot (s.Normal, h));
64                 float spec = pow (nh, 48.0);
65
66                 half4 c;
67
68                 #if defined (SPOT)
69                     float4 lightCoord = normalize
70                     (mul(unity_WorldToLight, float4 (s.worldPos, 1)));
71 //                     float4 lightCoord = mul(unity_WorldToLight,
72 //                     float4 (s.worldPos, 1));
73 //                     float u = lightCoord.x / lightCoord.z; //
74 //                     avoid projection from spotlight's angle using .w
75 //                     float u = lightCoord.x; // avoid projection
76 //                     from spotlight's angle using .w

```

```

71         atten = atten * 0.5 * (1.0 + sin (_Frequency *
72             u + _Phase));
73
74         // See if we can get a zero phase line.
75         if (abs (u) < 0.001)
76         {
77             atten = 0;
78         }
79         else
80         {
81             atten = 1;
82         }
83     #endif
84
85         c.rgb = (s.Albedo * _LightColor0.rgb * diff +
86             _LightColor0.rgb * spec) * atten;
87         c.a = s.Alpha;
88         return c;
89     }
90
91     // Add instancing support for this shader. You
92     // need to check 'Enable Instancing' on materials that
93     // use the shader.
94     // See
95     // https://docs.unity3d.com/Manual/GPUInstancing.html for
96     // more information about instancing.
97     // #pragma instancing_options assumeuniformscaling
98     UNITY_INSTANCING_BUFFER_START(Props)
99     // put more per-instance properties here
100    UNITY_INSTANCING_BUFFER_END(Props)
101
102    void surf (Input IN, inout SurfaceOutputCustom o)
103    {
104        // Albedo comes from a texture tinted by color
105        fixed4 c = tex2D (_MainTex, IN.uv_MainTex) *
106            _Color;
107        o.Albedo = c.rgb;
108        // Metallic and smoothness come from slider
109        // variables
110        o.Metallic = _Metallic;
111        o.Smoothness = _Glossiness;
112        o.Alpha = c.a;
113        o.worldPos = IN.PatternPos;
114    }
115    ENDCG
116 }
117 FallBack "Diffuse"
118 }
```

Create a new material named StructuredLightMaterial and assign this shader to that material. Apply this material to all the objects in the scene that were created

in the previous steps.

In the Inspector window for the material, the shader will present a few parameters that define the structured light pattern. The key value to set is the frequency of the periodic pattern that is projected. Ideally set this to a value that provides 5-10 cycles over the visible portion of the scene. Too low a frequency will limit the precision with which depth can be measured, while too high a frequency is more likely to confuse phase boundaries with scene boundaries when phase unwrapping.

You can experiment with the phase value. This represents the offset applied to where the cyclic pattern starts. This does not need to be set, as it will be directly controlled in software.

3. The scanning process will be managed by an additional component. Create a new C# script named PhaseUnwrap and add the code provided in Algorithm 18.14.

Code Listing 18.14: PhaseUnwrap. The scanning process is managed by this component. Three different snapshots of the scene are rendered using different phase offsets when the space bar is pressed. The left-alt key then triggers the phase unwrapping process which attempts to reconstruct a mesh from the pixel + phase coordinates.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class PhaseUnwrap : MonoBehaviour
6  {
7      public Material structuredLightMaterial;
8
9      public RenderTexture [] phaseImages;
10
11     ThreePhase t;
12     public MeshFilter scanObject;
13
14     // TODO: force main camera to have same aspect as
15     // rendertextures.
16
17     private IEnumerator captureAndUnwrap ()
18     {

```

```

18     // Render the camera view, updating the phase of
→   the shader each time.
19     for (int i = 0; i < phaseImages.Length; i++)
20     {
21       Camera.main.targetTexture = phaseImages[i];
22       structuredLightMaterial.SetFloat ("_Phase", i
→ * Mathf.PI * 2.0f / phaseImages.Length);
23       yield return null; // complete render cycle.
24     }
25   Camera.main.targetTexture = null;
26   structuredLightMaterial.SetFloat ("_Phase", 0.0f);
27
28   t = new ThreePhase ();
29   t.unwrapPhase (phaseImages[0], phaseImages[1],
→ phaseImages[2]);
30 }
31
32 // To register camera view and 3D environment, define
→ a number of
33 // points on surfaces in the scene, and provide an
→ interface to
34 // click on each and record their positions.
35 Vector3 [] regLoc =
36 {
37   new Vector3 (-0.4f, 0.4f, -0.5f),
38   new Vector3 (0.4f, 0.4f, -0.5f),
39   new Vector3 (-1.0f, -1.0f, 1.0f),
40   new Vector3 (-0.5f, 2.0f, 1.0f),
41   new Vector3 (1.9f, 2.5f, 1.0f),
42   //      new Vector3 (1.0f, 1.0f, -1.5f),
43   new Vector3 (-0.7f, -1.0f, -1.5f),
44   new Vector3 (0.6f, -1.0f, 0.5f),
45   new Vector3 (0.0f, -1.0f, 0.0f),
46 };
47
48 void Update()
49 {
50   // Capture images under various phases of
→ lighting, and unwrap.
51   if (Input.GetKeyDown (KeyCode.Space))
52   {
53     StartCoroutine (captureAndUnwrap ());
54   }
55
56   // Calibrate phase against 3D scene, and generate
→ mesh.
57   if (Input.GetKeyDown (KeyCode.LeftAlt))
58   {
59     Vector2 [] mapLoc = new Vector2
→ [regLoc.Length];
60
61     for (int i = 0; i < regLoc.Length; i++)
62     {

```

```

63             Debug.Log (" " +
64     ↵     Camera.main.WorldToScreenPoint (regLoc[i]));
65             Vector3 p = Camera.main.WorldToScreenPoint
66             (regLoc[i]);
67             mapLoc[i] = new Vector2 (p.x /
68             Screen.width, p.y / Screen.height);
69         }
70     }
71 }
```

Create a new empty object in the scene (named StructuredLight) and add the PhaseUnwrap script to this. The required parameters include the StructuredLight-Material since this script manipulates the phase offset property and captures the rendered scene for three different offsets.

The rendered scenes need to be stored in RenderTextures. Create three new RenderTexture objects. Each RenderTexture has a Dimension of 2D. The quality of the scanning process also depends on the Resolution of these RenderTextures. A resolution of  $512 \times 512$  is a reasonable starting point.

In the Phase Images property of the PhaseUnwrap component of the StructuredLight object, add additional elements and assign the created RenderTextures to each one of these elements.

Finally, the Scan Object property is used to identify one of the scene objects whose mesh will be replaced with the scanned output. For convenience, pick one of the existing objects in the scene and assign it to the Scan Object property. It does need to be an object that has a mesh filter component.

4. The phase unwrapping process makes use of a class that implements the process described in section 8.3.2 on page 118. The code for this class is found in Algorithm 18.15 on the next page.

Code Listing 18.15: ThreePhase. Phase unwrapping and mesh reconstruction facilities are provided in this class. The Math.Net library is a requirement.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using System.IO;
4  using UnityEngine;
5  using MathNet.Numerics.LinearAlgebra.Single;
6
7  // Developed thanks to the following resources:
8  // ThreePhase-2-source.zip at https://code.google.com/archive/p/structured-light/downloads, referenced
9  // from:
10 // https://www.instructables.com/Structured-Light-3D-Scanning/. Tracked to: Kyle McDonald,
11 // http://kylemcdonald.net/
12 // Fast phase unwrapping:
13 // https://github.com/mfkasim91/unwrap_phase/
14 public class ThreePhase
15 {
16     // Step (how many vertices to skip over) when
17     // generating mesh.
18     public int renderDetail = 2;
19
20     // Find the 4x4 homogenous transformation matrix that
21     // transforms map into reg (phase space to
22     // 3D points). Then apply this to create a mesh.
23     public void findMatrix (Vector3 [] reg, Vector2 []
24     map, MeshFilter scanObject)
25     {
26         // A transforms from p to q. It must transform
27         // points in phase space, to 3D coordinates.
28         // p = { map[i].x, map[i].y, v, 1 }
29         // q = { xx[i].x, xx[i].y, xx[i].z }
30         Matrix M = new DenseMatrix (reg.Length * 3, 16);
31         for (int i = 0; i < reg.Length; i++)
32         {
33             float v = phase[(int) (map[i].y *
34             phaseHeight), (int) (map[i].x * phaseWidth)];
35             M[i * 3 + 0, 0] = -map[i].x;
36             M[i * 3 + 0, 1] = -map[i].y;
37             M[i * 3 + 0, 2] = -v;
38             M[i * 3 + 0, 3] = -1.0f;
39             M[i * 3 + 0, 12] = reg[i].x * map[i].x;
40             M[i * 3 + 0, 13] = reg[i].x * map[i].y;
41             M[i * 3 + 0, 14] = reg[i].x * v;
42             M[i * 3 + 0, 15] = reg[i].x;
43
44             M[i * 3 + 1, 4] = -map[i].x;
45             M[i * 3 + 1, 5] = -map[i].y;
46             M[i * 3 + 1, 6] = -v;
47             M[i * 3 + 1, 7] = -1.0f;

```

```

40         M[i * 3 + 1, 12] = reg[i].y * map[i].x;
41         M[i * 3 + 1, 13] = reg[i].y * map[i].y;
42         M[i * 3 + 1, 14] = reg[i].y * v;
43         M[i * 3 + 1, 15] = reg[i].y;
44
45         M[i * 3 + 2, 8] = -map[i].x;
46         M[i * 3 + 2, 9] = -map[i].y;
47         M[i * 3 + 2, 10] = -v;
48         M[i * 3 + 2, 11] = -1.0f;
49         M[i * 3 + 2, 12] = reg[i].z * map[i].x;
50         M[i * 3 + 2, 13] = reg[i].z * map[i].y;
51         M[i * 3 + 2, 14] = reg[i].z * v;
52         M[i * 3 + 2, 15] = reg[i].z;
53
54     }
55
56     MathNet.Numerics.LinearAlgebra.Factorization.Svd<float> svdResult = M.Svd
57     ();
58     Matrix Vt = (Matrix) svdResult.VT;
59
60     // Last row of Vt has the desired solution.
61     Vector pp = (Vector) Vt.Row (15);
62     Matrix A = (Matrix) new DenseMatrix (4, 4, new
63     float []
64     {
65         pp[0], pp[1], pp[2], pp[3],
66         pp[4], pp[5], pp[6], pp[7],
67         pp[8], pp[9], pp[10], pp[11],
68         pp[12], pp[13], pp[14], pp[15]
69     }).Transpose ();
70
71     makeMesh (renderDetail, scanObject, A);
72 }
73
74 // Internal phase arrays.
75 private int phaseWidth;
76 private int phaseHeight;
77 private float[,] phase;
78 private float[,] unwrappedphase;
79
80 // Colour from original images, to be used for texture.
81 private Color[,] colours;
82
83 // Convert the 3 images provided into an unwrapped
84 // phase image.
85 public void unwrapPhase (RenderTexture p1,
86 RenderTexture p2, RenderTexture p3)
87 {
88     Texture2D phase1Image, phase2Image, phase3Image;
89
90     phase1Image = accessTexture (p1);
91     phase2Image = accessTexture (p2);
92     phase3Image = accessTexture (p3);
93 }
```

```

88         phaseWidth = phase1Image.width;
89         phaseHeight = phase1Image.height;
90         phase = new float[phaseHeight, phaseWidth];
91         unwrappedphase = new float[phaseHeight,
92             phaseWidth];
93             colours = new Color[phaseHeight, phaseWidth];
94
95         initializePhase (phase1Image, phase2Image,
96             phase3Image);
97         exportPhase (phase, "Phase");
98         unwrap_phase ();
99         exportPhase (unwrappedphase, "UnwrappedPhase");
100        phase = unwrappedphase;
101    }
102
103    // A single texture, to which the colour data is
104    // extracted.
105    private Texture2D meshTex;
106
107    private void makeMesh (int step, MeshFilter
108        scanObject, Matrix A)
109    {
110        if (meshTex == null)
111        {
112            meshTex = new Texture2D(2, 2);
113        }
114
115        Vector3[] vertices = new Vector3[(phaseWidth /
116            step) * (phaseHeight / step)];
117        Vector2[] uvs = new Vector2[(phaseWidth / step) *
118            (phaseHeight / step)];
119        int[] triangles = new int[6 * ((phaseWidth / step)
120            - 1) * ((phaseHeight / step) - 1)];
121
122        meshTex.Resize (phaseWidth / step, phaseHeight /
123            step);
124
125        int triangleIndex = 0;
126        for (int y = 0; y < phaseHeight; y += step)
127        {
128            for (int x = 0; x < phaseWidth; x += step)
129            {
130                float xc = (float)x / phaseWidth;
131                float zc = (float)y / phaseHeight;
132                float yc = 0.0f;
133
134                float v = phase[y, x];
135                Vector inv = new DenseVector (new float []
136                    { xc, zc, v, 1 });
137                Vector o = (Vector) A.Multiply (inv);
138
139                meshTex.SetPixel (x / step, y / step,
140                    colours[y, x]);

```

```

131             xc = (o[0] / o[3]) + 0.5f;
132             yc = (o[1] / o[3]);
133             zc = (o[2] / o[3]) + 0.5f;
134
135             vertices[(y / step) * (phaseWidth / step)
136             + (x / step)] = new Vector3(xc - 0.5f, yc, zc - 0.5f);
137             uvs[(y / step) * (phaseWidth / step) + (x
138             / step)] = new Vector2( (float)x / phaseWidth,
139             (float)y / phaseHeight);
140
141             // Skip the last row/col
142             if ((x < phaseWidth - step) && (y <
143             phaseHeight - step))
144             {
145                 int topLeft = (y / step) * (phaseWidth
146                 / step) + (x / step);
147                 int topRight = topLeft + 1;
148                 int bottomLeft = topLeft + (phaseWidth
149                 / step);
150                 int bottomRight = bottomLeft + 1;
151
152                 triangles[triangleIndex++] = topRight;
153                 triangles[triangleIndex++] = topLeft;
154                 triangles[triangleIndex++] =
155                 bottomLeft;
156
157                 triangles[triangleIndex++] = topRight;
158                 triangles[triangleIndex++] =
159                 bottomLeft;
160                 triangles[triangleIndex++] =
161                 bottomRight;
162             }
163         }
164     }
165
166     meshTex.Apply ();
167
168     Mesh m = new Mesh();
169     m.indexFormat =
170     UnityEngine.Rendering.IndexFormat.UInt32;
171     m.vertices = vertices;
172     m.uv = uvs;
173     m.triangles = triangles;
174     m.RecalculateNormals();
175     scanObject.mesh = m;
176
177     scanObject.gameObject.GetComponent <MeshRenderer>
178     ().material.mainTexture = meshTex;
179 }
180
181
182 // Convert a render texture, to a texture with easier
183 // pixel access.
184 Texture2D accessTexture (RenderTexture originalTexture)

```

```

172     {
173         Texture2D tex = new
174         Texture2D(originalTexture.width,
175         originalTexture.height);
176         RenderTexture.active = originalTexture;
177         tex.ReadPixels (new Rect (0, 0,
178         originalTexture.width, originalTexture.height), 0, 0);
179         tex.Apply();
180         return tex;
181     }
182
183     void initializePhase (Texture2D phase1Image, Texture2D
184     phase2Image, Texture2D phase3Image)
185     {
186         float sqrt3 = Mathf.Sqrt (3.0f);
187         for (int y = 0; y < phaseHeight; y++) {
188             for (int x = 0; x < phaseWidth; x++) {
189
190                 Color color1 = phase1Image.GetPixel (x, y);
191                 Color color2 = phase2Image.GetPixel (x, y);
192                 Color color3 = phase3Image.GetPixel (x, y);
193
194                 float phase1 = averageBrightness(color1);
195                 float phase2 = averageBrightness(color2);
196                 float phase3 = averageBrightness(color3);
197
198                 // Fundamental expression of phase shift -
199                 // recover phase from image intensities.
200                 phase[y, x] = Mathf.Atan2(sqrt3 * (phase1
201                 - phase3), 2 * phase2 - phase1 - phase3); // + Mathf.PI;
202
203                 colours[y, x] =
204                 blendColorLightest(blendColorLightest(color1, color2),
205                 color3);
206             }
207         }
208     }
209
210     // A support function, to show the phase maps.
211     void exportPhase (float [,] phase, string fn)
212     {
213         Texture2D tex = new Texture2D(phaseWidth,
214         phaseHeight);
215         for (int y = 0; y < phaseHeight; y++) {
216             for (int x = 0; x < phaseWidth; x++) {
217                 float p = (phase[y, x] / (2.0f *
218                 Mathf.PI)) + 0.5f;
219                 tex.SetPixel (x, y, new Color (p, p, p));
220             }
221         }
222         tex.Apply();
223         byte[] bytes = tex.EncodeToPNG();
224         var dirPath = Application.dataPath;

```

```
215     File.WriteAllBytes(dirPath + "/" + fn + ".png",  
216     bytes);  
217 }  
218  
219 // https://github.com/processing/processing/blob/a6e0e  
220 227a948e7e2dc042c04504d6f5b8cf0c1a6/core/src/processing/  
221 g/core/PImage.java  
222 Color blendColorLightest (Color dst, Color src)  
223 {  
224     float a = src.a;  
225  
226     float s_a = a;  
227     float d_a = 1.0f - s_a;  
228  
229     return new Color (dst.r * d_a + Mathf.Max (src.r,  
230 dst.r) * s_a,  
231                         dst.g * d_a + Mathf.Max (src.g,  
232 dst.g) * s_a,  
233                         dst.b * d_a + Mathf.Max (src.b,  
234 dst.b) * s_a,  
235                         Mathf.Min(dst.a + a, 1.0f));  
236 }  
237  
238 float averageBrightness(Color c) {  
239     return (c.r + c.g + c.b) / (3.0f);  
240 }  
241  
242 float gamma (float x)  
243 {  
244     return Mathf.Sign(x) * (Mathf.Abs(x) % Mathf.PI);  
245 }  
246  
247 float[,] get_reliability (float[,] img)  
248 {  
249     float[,] rel = new float [phaseHeight,  
250 phaseWidth];  
251  
252     for(int y = 0; y < phaseHeight; y++) {  
253         for(int x = 0; x < phaseWidth; x++) {  
254             float img_im1_jm1 = ((x > 0) && (y > 0)) ?  
255 img[y - 1, x - 1] : 0.0f;  
256             float img_i_jm1 = (y > 0) ? img[y - 1,  
257 x] : 0.0f;  
258             float img_ip1_jm1 = ((x < phaseWidth - 1)  
259 && (y > 0)) ? img[y - 1, x + 1] : 0.0f;  
260             float img_im1_j = (x > 0) ? img[y, x -  
261 1] : 0.0f;  
262             float img_i_j = img[y, x];  
263             float img_ip1_j = (x < phaseWidth - 1)  
264 ? img[y, x + 1] : 0.0f;  
265             float img_im1_jp1 = ((x > 0) && (y <  
266 phaseHeight - 1)) ? img[y + 1, x - 1] : 0.0f;
```

```

254         float img_i_jp1 = (y < phaseHeight - 1)
255         ? img[y + 1, x] : 0.0f;
256         float img_ip1_jp1 = ((x < phaseWidth - 1))
257         && (y < phaseHeight - 1)) ? img[y + 1, x + 1] : 0.0f;
258         float H = gamma(img_im1_j - img_i_j) -
259         gamma(img_i_j - img_ip1_j);
260         float V = gamma(img_i_jm1 - img_i_j) -
261         gamma(img_i_j - img_ip1_j);
262         float D1 = gamma(img_im1_jm1 - img_i_j) -
263         gamma(img_i_j - img_ip1_jm1);
264         float D2 = gamma(img_im1_jp1 - img_i_j) -
265         gamma(img_i_j - img_ip1_jm1);
266
267         float D = Mathf.Sqrt (H * H + V * V + D1 *
268         D1 + D2 * D2);
269
270         if (D != 0.0f)
271         {
272             rel[y, x] = 1.0f / D;
273         }
274         else
275         {
276             rel[y, x] = 0.0f;
277         }
278     }
279     return rel;
280 }
281
282 void unwrap_phase ()
283 {
284     int Nx = phaseWidth;
285     int Ny = phaseHeight;
286
287     float[,] reliability = get_reliability(phase);
288
289     float[,] h_edges = new float [phaseHeight,
290     phaseWidth];
291     float[,] v_edges = new float [phaseHeight,
292     phaseWidth];
293     for(int y = 0; y < phaseHeight; y++) {
294         for(int x = 0; x < phaseWidth; x++) {
295             unwrappedphase[y, x] = phase[y, x];
296
297             h_edges[y, x] = (x < phaseWidth - 1) ?
298                 reliability[y, x] + reliability[y, x + 1] : float.NaN;
299                 v_edges[y, x] = (y < phaseHeight - 1) ?
300                 reliability[y, x] + reliability[y + 1, x] : float.NaN;
301             }
302         }
303
304     List <(float, int)> edges = new List <(float,
305     int)> ();

```

```

295     for(int x = 0; x < phaseWidth; x++) {
296         for(int y = 0; y < phaseHeight; y++) {
297             edges.Add ((h_edges[y, x], edges.Count));
298         }
299     }
300     for(int x = 0; x < phaseWidth; x++) {
301         for(int y = 0; y < phaseHeight; y++) {
302             edges.Add ((v_edges[y, x], edges.Count));
303         }
304     }
305
306     int edge_bound_idx = Ny * Nx;
307     edges.Sort ((a, b) => b.Item1.CompareTo (a.Item1));
308
309     int [] idxs1 = new int [edges.Count];
310     int [] idxs2 = new int [edges.Count];
311     for (int i = 0; i < edges.Count; i++)
312     {
313         idxs1[i] = edges[i].Item2 % edge_bound_idx;
314         idxs2[i] = idxs1[i] + 1 + (edges[i].Item2 <
315         edge_bound_idx ? Ny - 1 : 0);
316     }
317
318     int [] group = new int [Ny * Nx];
319     bool [] is_grouped = new bool [Ny * Nx];
320     List <int> [] group_members = new List <int> [Ny *
321     Nx];
322     int [] num_members_group = new int [Ny * Nx];
323     for (int i = 0; i < Ny * Nx; i++)
324     {
325         group[i] = i;
326         is_grouped[i] = false;
327         group_members[i] = new List <int> ();
328         group_members[i].Add (i);
329         num_members_group[i] = 1;
330     }
331
332     for (int i = 0; i < edges.Count; i++)
333     {
334         int idx1 = idxs1[i];
335         int idx2 = idxs2[i];
336
337         if ((idx1 != -1) && (idx2 != -1) && (idx2 <
338         edge_bound_idx) && (!float.IsNaN (edges[idx1].Item1))
339         && (!float.IsNaN (edges[idx2].Item1)))
340         {
341             // Debug.Log (idx1 + " " +
342             idx2);
343             if (!(group[idx1] == group[idx2]))
344             {
345                 bool all_grouped = false;
346                 if (is_grouped[idx1])
347                 {

```

```

343             if (!is_grouped[idx2])
344             {
345                 int idxt = idx1;
346                 idx1 = idx2;
347                 idx2 = idxt;
348             }
349             else if
350             (
351                 num_members_group[group[idx1]] >
352                 num_members_group[group[idx2]])
353             {
354                 int idxt = idx1;
355                 idx1 = idx2;
356                 idx2 = idxt;
357                 all_grouped = true;
358             }
359             else
360             {
361                 all_grouped = true;
362             }
363         }
364
365         // At this point, either all grouped,
366         // or idx1 is the not grouped.
367         float dval =
368         Mathf.Floor((unwrappedphase[idx2 % phaseHeight, idx2 /
369         phaseHeight] - unwrappedphase[idx1 % phaseHeight, idx1
370         / phaseHeight] + Mathf.PI) / (2.0f * Mathf.PI)) *
371         (2.0f * Mathf.PI);
372
373         int g1 = group[idx1];
374         int g2 = group[idx2];
375         List<int> pix_idxs;
376         if (all_grouped)
377         {
378             pix_idxs = group_members[g1];
379         }
380         else
381         {
382             pix_idxs = new List<int> ();
383             pix_idxs.Add (idx1);
384         }
385
386         if (dval != 0.0f)
387         {
388             for (int j = 0; j <
389             pix_idxs.Count; j++)
390             {
391                 unwrappedphase[pix_idxs[j] %
392                 phaseHeight, pix_idxs[j] / phaseHeight] += dval;
393             }
394         }
395
396         group_members[g2].AddRange (pix_idxs);

```

```

387             for (int j = 0; j < pix_idxs.Count;
388                 j++)
389             {
390                 group[pix_idxs[j]] = g2;
391             }
392             num_members_group[g2] +=
393                 pix_idxs.Count;
394             is_grouped[idx1] = true;
395             is_grouped[idx2] = true;
396         }
397     }
398     for(int y = 0; y < phaseHeight; y++) {
399         for(int x = 0; x < phaseWidth; x++) {
400             // Arbitrary scaling, to make the exported
401             // version visible - remap to range 0 to 1 just
402             // for the test data used.
403             unwrappedphase[y, x] = 0.2f *
404             unwrappedphase[y, x] + 2.0f;
405         }
406     }
407 }
```

This class requires the Math.Net numerics package which can be downloaded from: <https://github.com/mathnet/mathnet-numerics>. Download the zip file from this site, and unzip it into the Unity project. When last used, there were several components that report errors when used within Unity. Fortunately, we don't require any of the affected elements and so the simplest solution is to delete the problematic elements. Delete all of the following folders within the Math.Net package whose names start with:

- Numerics.Tests
- TestData
- Benchmark
- Data.Matlab
- Data.Tests
- Data.Text
- Providers.MKL

- Providers.OpenBLAS
- Providers.CUDA
- NativeProviders

There should be no errors reported in the remaining functions.

5. At this point, you should be able to test the projection and scene capture. Run the application within the Unity Editor, and press the space bar. The camera view may be interrupted briefly as the projection is changed, and the scene is rendered to the render textures. Stop the application, and examine the individual render textures in the Inspector pane. You should see views of the scene, each with the projected pattern offset from the pattern shown in the other render textures.
6. The next step involves unwrapping the phase and attempting to recover the geometry of the scene. The mapping between the original 3D geometry, and the pixel + phase coordinates requires some calibrated points. These are kept within the PhaseUnwrap code, in the variable regLoc. Replace the values listed with a set of coordinates representing points that are both: on the surface of objects in the scene, and are visible within the camera view. Ideally these should be well distributed and not all in the same plane.

Run the application again, and press the space bar, followed by the left Alt key. The application writes two image files to the root of the Assets folder. These represent the wrapped phase (should have several cycles of light to dark) and the unwrapped phase (which should just be a continuous grey variation across the scene). If this is not clear, there is a colour scaling value at the end of the `unwrap_phase` function in `ThreePhase.cs` that might need to be tweaked for your particular scene layout.

You are likely to see some issues in the unwrapped phase, particularly in areas shadowed from the light

source which do not received any phase information from the projector.

While the application is running, the reconstructed mesh should be visible in the scene view. If you hide the other objects in the scene you will be able to see this more clearly. If the resulting mesh looks like it wandered into a small nuclear explosion then you may need to either check your calibration points or adjust (probably lower) the frequency of the projected pattern in the material's parameters. The latter case is more likely if the unwrapped phase image still retains any abrupt changes in brightness remaining from the wrapped phase.

7. Experiment with this process. While this component is a useful facility, it is still working with ambiguous data. Research into phase unwrapping algorithms is ongoing and a range of different strategies have been developed to get better results in different contexts.

If you have access to a projector, you can try capturing the images in the physical world. This will introduce further challenges with managing physical lighting, capturing it accurately, and measuring the calibration points.

Some of the distortions produced in the reconstructed image could also result from assumptions or errors in the way the projected lighting is simulated. A more accurate implementation of projection, for example, might be able to straighten the curvature in the reconstructed planes.

### 18.7 Aligning multiple point clouds to reconstruct the appearance of objects

#### Component 18.7.1: Assembling objects from partial

scans

Point Cloud  
Registration

This component could equally well be considered in terms of the location dimension. It is intended to be employed when transferring physical objects or settings into the virtual world. Assuming you have some form of scanner, such as a depth camera that can provide a representation of a limited region of space, usually as a cloud of points on the surface, and potentially with further detail (such as mesh structure, or colour or texture information). A single scan does not show the entire object. At the very least, occluded surfaces such as the back of the object will not be captured in the scan.

The solution is to take many scans from different viewpoints and stitch these together. A challenge with this process is aligning the various scans. While it can be painstakingly done in a 3D modelling tool, a preferred option would be to automate the process. This would allow the enhanced reality experience to immediately (or at least, after a short time) complete the scan and be able to use the object as part of the experience.

This component demonstrates one process that registers two point clouds by identifying and aligning their overlapping regions. The example uses purely synthetic data, generating the two point cloud examples by sampling from an object that you provide it. One of the point clouds is (randomly) transformed and the test of the registration process is to see if it can reverse this transformation.

Synthetic data is used because the source of the point clouds can come from a range of sources, including several other components (e.g.  Component 17.5.1 on page 261). These components can be integrated with this example to produce a multi-scan object reconstruction solution.

1. Start with a new Unity project. The project needs an object to ‘scan’, to simulate the process of scanning a physical object and producing a point cloud. Find a suitable object to add to your scene. Ideally this object would have a reasonable number of vertices. Too few

vertices reduces the opportunity to sample a subset and meaningfully align the sampled point clouds. The vertices should also be reasonably evenly spaced. This is more to help you visually infer the shape of the object when checking the outcome, rather than a particular requirement of the process. An object with about 100,000 vertices is a good start.

2. The FastGlobalRegistration class is required to do the heavy lifting. The code in Algorithm 18.16 is based on the software provided at <https://github.com/isl-org/FastGlobalRegistration>. The principles behind this process are described in section 8.4.

Code Listing 18.16: *FastGlobalRegistration*. The Fast Global Registration class includes functions to compute feature values, identify corresponding points and calculate the transformation that best aligns them.

```

1 // -----
2 // -          Fast Global Registration
3 // -----
4 // The MIT License (MIT)
5 //
6 // Copyright (c) Intel Corporation 2016
7 // Qianyi Zhou <Qianyi.Zhou@gmail.com>
8 // Jaesik Park <syncle@gmail.com>
9 // Vladlen Koltun <vkoltun@gmail.com>
10 //
11 // Permission is hereby granted, free of charge, to any
12 // person obtaining a copy
13 // of this software and associated documentation files
14 // (the "Software"), to deal
15 // in the Software without restriction, including without
16 // limitation the rights
17 // to use, copy, modify, merge, publish, distribute,
18 // sublicense, and/or sell
19 // copies of the Software, and to permit persons to whom
20 // the Software is
// furnished to do so, subject to the following conditions:
// 
// The above copyright notice and this permission notice
// shall be included in
// all copies or substantial portions of the Software.
// 
```

```

21 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF
22 // ANY KIND, EXPRESS OR
23 // IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
24 // MERCHANTABILITY,
25 // FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
26 // IN NO EVENT SHALL THE
27 // AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
28 // DAMAGES OR OTHER
29 // LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
30 // OTHERWISE, ARISING
31 // FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE
32 // USE OR OTHER DEALINGS
33 // IN THE SOFTWARE.
34 // -----
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53

```

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Text;
using MathNet.Numerics.LinearAlgebra.Single;
using MathNet.Numerics.Statistics;
using UnityEngine;
using KdTree;
using KdTree.Math;

class FastGlobalRegistration
{
    class Points : List <Vector> {} // of 3 element
    // vectors.
    class Feature : List <Vector> {} // of X element
    // vectors.
    class Correspondences : List <(int, int)> {}

    public const double DIV_FACTOR =
        1.4; // Division factor used for
        // graduated non-convexity
    public const bool USE_ABSOLUTE_SCALE =
        false; // Measure distance in
        // absolute scale (1) or in scale relative to the
        // diameter of the model (0)
    public const double MAX_CORR_DIST = 0.25; // Maximum correspondence distance (also see comment
        // of USE_ABSOLUTE_SCALE)
    public const int ITERATION_NUMBER =
        64; // Maximum number of iteration
    public const float TUPLE_SCALE = 0.95f; // Similarity measure used for tuples of feature
        // points. (tau)
    public const int TUPLE_MAX_CNT = 1000; // Maximum tuple numbers.
}

```

```

54     // containers
55     private List <Points> pointcloud_;
56     private List <Feature> features_;
57     private Matrix4x4 TransOutput_;
58     private List <(int, int)> corres_;
59
60     // for normalization
61     private Points Means;
62     private float GlobalScale = 1.0f;
63     private float StartScale = 1.0f;
64
65     private double div_factor_;
66     private bool use_absolute_scale_;
67     private double max_corr_dist_;
68     private int iteration_number_;
69     private float tuple_scale_;
70     private int tuple_max_cnt_;
71     private float searchRadius; // r
72     private int maxNeighbours; // k
73
74     public FastGlobalRegistration ()
75     {
76         div_factor_ = DIV_FACTOR;
77         use_absolute_scale_ = USE_ABSOLUTE_SCALE;
78         max_corr_dist_ = MAX_CORR_DIST;
79         iteration_number_ = ITERATION_NUMBER;
80         tuple_scale_ = TUPLE_SCALE;
81         tuple_max_cnt_ = TUPLE_MAX_CNT;
82
83         searchRadius = 0.85f;
84         maxNeighbours = 80;
85
86         pointcloud_ = new List <Points> ();
87         features_ = new List <Feature> ();
88
89         Means = new Points ();
90     }
91
92     // Adapted to deal with normals that might face
93     // inwards or outwards.
94     // Try both directions, and select largest value (as
95     // presumably better to discriminate).
96     private void computeAngles (Vector ps, Vector ns,
97     → Vector pt, Vector nt, out float alpha, out float
98     → phi, out float theta)
99     {
100         Vector3 psv = new Vector3 (ps[0], ps[1], ps[2]);
101         Vector3 ptv = new Vector3 (pt[0], pt[1], pt[2]);
102         Vector3 ntv = new Vector3 (nt[0], nt[1],
103         → nt[2]).normalized;
104         Vector3 ntvalt = -ntv;
105         Vector3 u = new Vector3 (ns[0], ns[1],
106         → ns[2]).normalized;

```

```

101     Vector3 ualt = -u;
102     Vector3 v = Vector3.Cross (u, (ptv -
103         ↳ psv).normalized);
104     Vector3 valt = Vector3.Cross (-u, (ptv -
105         ↳ psv).normalized);
106     Vector3 w = Vector3.Cross (u, v);
107     Vector3 walt = Vector3.Cross (-u, v);
108     float d = (ptv - psv).magnitude;
109     alpha = Mathf.Max (Vector3.Dot (v, ntv),
110         ↳ Vector3.Dot (v, ntvalt), Vector3.Dot (valt,
111         ↳ ntv), Vector3.Dot (valt, ntvalt));
112     phi = Mathf.Max (Vector3.Dot (u, (ptv - psv) / d),
113         ↳ Vector3.Dot (ualt, (ptv - psv) / d));
114     //           Debug.Log ("D " + d + " " + u + " "
115         ↳ + new Vector3 (ns[0], ns[1], ns[2]));
116     float thetad = Mathf.Max (Vector3.Dot (w, ntv),
117         ↳ Vector3.Dot (w, ntvalt), Vector3.Dot (walt,
118         ↳ ntv), Vector3.Dot (walt, ntvalt));
119     float thetan = Mathf.Max (Vector3.Dot (u, ntv),
120         ↳ Vector3.Dot (u, ntvalt), Vector3.Dot (ualt,
121         ↳ ntv), Vector3.Dot (ualt, ntvalt));
122     theta = Mathf.Atan2 (thetad, thetan);
123 }
124
125 private Points findNormals (Points pts, List
126     ↳ <List<int>> neighbourCorres, List <List<float>>
127     ↳ neighbourDis)
128 {
129
130     Points normals = new Points ();
131
132     for (int i = 0; i < pts.Count; i++)
133     {
134         List <Vector> p = new List <Vector> ();
135         for (int j = 0; j < neighbourCorres[i].Count;
136             ↳ j++)
137         {
138             p.Add (pts[neighbourCorres[i][j]]);
139         }
140
141         if (p.Count == 0)
142         {
143             // no neighbours. Fudge normal.
144             normals.Add (pts[i]);
145         }
146         else
147         {
148             // Fit:  $(p_i - x).n = 0$ 
149             // with  $p_i$  is point on plane. Use  $pts[i]$ .
150             ↳  $x$  is current point,  $p[j]$ 
151             Matrix M = new DenseMatrix (p.Count, 3);
152             for (int j = 0; j < p.Count; j++)
153             {

```

```

140             M[j, 0] = pts[i][0] - p[j][0];
141             M[j, 1] = pts[i][1] - p[j][1];
142             M[j, 2] = pts[i][2] - p[j][2];
143         }
144
145         MathNet.Numerics.LinearAlgebra.Factorizati_
146             ↪ on.Svd<float> svdResult = M.Svd
147             ↪ ();
148         Matrix Vt = (Matrix) svdResult.VT;
149
150         // Last row of Vt has the desired solution.
151         Vector pp = (Vector) Vt.Row (Vt.RowCount -
152             ↪ 1);
153
154         normals.Add (pp);
155     }
156
157     private Feature calculateSPFH (Points pts, Points
158             ↪ normals, List <List<int>> neighbourCorres, List
159             ↪ <List<float>> neighbourDis)
160     {
161         Feature spfh = new Feature ();
162         float alpha, phi, theta;
163         for (int i = 0; i < pts.Count; i++)
164         {
165             List <double> alphas = new List <double> ();
166             List <double> phis = new List <double> ();
167             List <double> thetas = new List <double> ();
168             for (int j = 0; j < neighbourCorres[i].Count;
169                 ↪ j++)
170             {
171                 computeAngles (pts[i], normals[i],
172                     ↪ pts[neighbourCorres[i][j]],
173                     ↪ normals[neighbourCorres[i][j]], out
174                     ↪ alpha, out phi, out theta);
175                 alphas.Add (alpha);
176                 phis.Add (phi);
177                 thetas.Add (theta);
178             }
179             Vector feat_v = new DenseVector (33);
180             if (alphas.Count > 0)
181             {
182                 Histogram alphah = new Histogram (alphas,
183                     ↪ 11);
184                 Histogram phih = new Histogram (phis, 11);
185                 Histogram thetah = new Histogram (thetas,
186                     ↪ 11);
187
188                 for (int j = 0; j < 11; j++)
189                 {

```

```

182         feat_v[j * 3 + 0] = ((float)
183             ↵ alphah[j].Count) /
184             ↵ alphah.BucketCount;
185         feat_v[j * 3 + 1] = ((float)
186             ↵ phih[j].Count) / phih.BucketCount;
187         feat_v[j * 3 + 2] = ((float)
188             ↵ thetah[j].Count) /
189             ↵ thetah.BucketCount;
190     }
191 }
192 else
193 {
194     for (int j = 0; j < 33; j++)
195     {
196         feat_v[j] = 0.0f;
197     }
198 }
199
200 // Fast point feature histograms:
201 // https://pcl.readthedocs.io/projects/tutorials/en/l
202 // atest/fpfh_estimation.html
203 private Feature calculateFPFH (Points pts, Points
204     ↵ normals, Feature spfh, List <List<int>>
205     ↵ neighbourCorres, List <List<float>> neighbourDis)
206 {
207     Feature fpfh = new Feature ();
208     for (int i = 0; i < pts.Count; i++)
209     {
210         Vector feat_v = spfh[i];
211
212         int k = neighbourCorres[i].Count;
213
214         for (int j = 0; j < neighbourCorres[i].Count;
215             ↵ j++)
216         {
217             feat_v = (Vector) (feat_v +
218                 ↵ spfh[neighbourCorres[i][j]] / (k *
219                 ↵ neighbourDis[i][j]));
220         }
221
222         fpfh.Add (feat_v);
223     }
224
225     return fpfh;
226 }
227
228 public void AddFeature(List <Vector3> pointCloud)
229 {

```

```

223 Points pts = new Points ();
224
225     for (int v = 0; v < pointCloud.Count; v++) {
226         Vector pts_v = new DenseVector (new float [] {
227             ← pointCloud[v].x, pointCloud[v].y,
228             ← pointCloud[v].z });
229         pts.Add(pts_v);
230     }
231
232     // Find the neighbours for each point, limited by
233     // → radius and maximum number.
234     // This information is used in several steps, so
235     // → get this once and reuse.
236     KdTree<float, int> tree = new KdTree<float, int>
237     ← (3, new FloatMath());
238     BuildKDTree(pts, tree);
239     List <List<int>> neighbourCorres = new List
240     ← <List<int>> ();
241     List <List<float>> neighbourDis = new List
242     ← <List<float>> ();
243
244     // Find neighbours.
245     for (int i = 0; i < pts.Count; i++)
246     {
247         List<int> corres = new List<int> ();
248         List<float> dis = new List<float> ();
249
250         SearchKDTree(tree, pts[i], corres, dis,
251             ← maxNeighbours);
252
253         List<int> validCorres = new List<int> ();
254         List<float> validDis = new List<float> ();
255         List <Vector> p = new List <Vector> ();
256         for (int j = 0; j < corres.Count; j++)
257         {
258             if ((corres[j] != i) && (dis[j] <
259                 ← searchRadius) && (dis[j] > 0.0f))
260             {
261                 validCorres.Add (corres[j]);
262                 validDis.Add (dis[j]);
263             }
264         }
265         neighbourCorres.Add (validCorres);
266         neighbourDis.Add (validDis);
267     }
268
269     Points normals = findNormals (pts,
270         ← neighbourCorres, neighbourDis);
271     Feature spfh = calculateSPFH (pts, normals,
272         ← neighbourCorres, neighbourDis);
273     Feature fpfh = calculatePPFH (pts, normals, spfh,
274         ← neighbourCorres, neighbourDis);

```

```

264     int ndim = fpfh[0].Count;
265     Debug.LogFormat("{0} points with {1} feature
266     → dimensions.\n", pointCloud.Count, ndim);
267
268     pointcloud_.Add(pts);
269     features_.Add(fpfh);
270 }
271
272 private float sqrnorm (Vector a)
273 {
274     return (float) (a.Norm (2.0) * a.Norm (2.0));
275 }
276
277 private float norm (Vector a)
278 {
279     return (float) (a.Norm (2.0));
280 }
281
282 private void BuildKDTree (List <Vector> data,
283     → KdTree<float, int> tree)
284 {
285     int rows, dim;
286     rows = (int)data.Count;
287     dim = (int)data[0].Count;
288     for (int i = 0; i < rows; i++)
289     {
290         float [] datarow = new float [dim];
291         for (int j = 0; j < dim; j++)
292         {
293             datarow[j] = data[i][j];
294         }
295         tree.Add (datarow, i);
296     }
297 }
298
299 private void SearchKDTree (KdTree<float, int> tree,
300     → Vector input,
301                 List <int> indices,
302                 List <float> dists, int nn)
303 {
304     int rows_t = 1;
305     int dim = input.Count;
306
307     float [] datarow = new float [dim];
308     for (int j = 0; j < dim; j++)
309     {
310         datarow[j] = input[j];
311     }
312     Vector datarowv = new DenseVector (datarow);
313
314     KdTreeNode<float, int>[] result =
315     → tree.GetNearestNeighbours(datarow, nn);
316     indices.Clear ();
317 }
```

```

313     dists.Clear ();
314     for (int a = 0; a < nn; a++)
315     {
316         indices.Add (result[a].Value);
317         dists.Add (sqrnorm ((Vector) (new DenseVector
318             → (result[a].Point) - datarowv)));
319     }
320 }
321
322 private List <T> buildList<T> (int size, T
323 → initialValue)
324 {
325     List <T> l = new List <T> ();
326     for (int i = 0; i < size; i++)
327     {
328         l.Add (initialValue);
329     }
330     return l;
331 }
332
333 public void AdvancedMatching()
334 {
335     int fi = 0;
336     int fj = 1;
337
338     Debug.LogFormat("Advanced matching : [{0} ,
339 → {1}]\n", fi, fj);
340     bool swapped = false;
341
342     if (pointcloud_[fj].Count > pointcloud_[fi].Count)
343     {
344         int temp = fi;
345         fi = fj;
346         fj = temp;
347         swapped = true;
348     }
349
350     int nPtI = pointcloud_[fi].Count;
351     int nPtJ = pointcloud_[fj].Count;
352
353     KdTree<float, int> feature_tree_i = new
354     → KdTree<float, int> (features_[fi][0].Count,
355     → new FloatMath());
356     BuildKDTree(features_[fi], feature_tree_i);
357
358     KdTree<float, int> feature_tree_j = new
359     → KdTree<float, int> (features_[fj][0].Count,
360     → new FloatMath());
361     BuildKDTree(features_[fj], feature_tree_j);
362
363     bool crosscheck = true;
364     bool tuple = true;
365 }
```



```

406     /// input : corres_ij, corres_ji
407     /// output : corres
408     /////////////////////////////////
409     if (crosscheck)
410     {
411         Debug.LogFormat("\t[cross check] ");
412
413         // build data structure for cross check
414         corres.Clear();
415         corres_cross.Clear();
416         List<List<int>> Mi = new List<List<int>>
417             ↳ (nPti);
418         List<List<int>> Mj= new List<List<int>>
419             ↳ (nPtj);
420
421         for (int i = 0; i < nPti; ++i)
422         {
423             Mi.Add (new List <int> ());
424
425             for (int j = 0; j < nPtj; ++j)
426             {
427                 Mj.Add (new List <int> ());
428
429             int ci, cj;
430             for (int i = 0; i < ncorres_ij; ++i)
431             {
432                 ci = corres_ij[i].Item1;
433                 cj = corres_ij[i].Item2;
434                 Mi[ci].Add(cj);
435             }
436             for (int j = 0; j < ncorres_ji; ++j)
437             {
438                 ci = corres_ji[j].Item1;
439                 cj = corres_ji[j].Item2;
440                 Mj[cj].Add(ci);
441             }
442
443             // cross check
444             for (int i = 0; i < nPti; ++i)
445             {
446                 for (int ii = 0; ii < Mi[i].Count; ++ii)
447                 {
448                     int j = Mi[i][ii];
449                     for (int jj = 0; jj < Mj[j].Count;
450                         ↳ ++jj)
451                     {
452                         if (Mj[j][jj] == i)
453                         {
454                             corres.Add((i, j));
455                             corres_cross.Add((i, j));
456                         }
457                     }
458                 }
459             }
460         }
461     }

```

```

456         }
457     }
458     Debug.LogFormat("Number of points that remain
459     → after cross-check: {0}\n",
460     → (int)corres.Count);
461
462     ///////////////////////////////
463     /// TUPLE CONSTRAINT
464     /// input : corres
465     /// output : corres
466     ///////////////////////////////
467     if (tuple)
468     {
469         UnityEngine.Random.InitState ((int)
470         → System.DateTime.Now.Ticks);
471
472         Debug.LogFormat("\t[tuple constraint] ");
473         int rand0, rand1, rand2;
474         int idi0, idil, idi2;
475         int idj0, idj1, idj2;
476         float scale = tuple_scale_;
477         int ncorr = corres.Count;
478         int number_of_trial = ncorr * 100;
479         List<int, int> corres_tuple = new List<(int,
480         → int)> ();
481
482         int cnt = 0;
483         int i;
484         for (i = 0; i < number_of_trial; i++)
485         {
486             rand0 = UnityEngine.Random.Range (0,
487             → ncorr);
488             rand1 = UnityEngine.Random.Range (0,
489             → ncorr);
490             rand2 = UnityEngine.Random.Range (0,
491             → ncorr);
492
493             idi0 = corres[rand0].Item1;
494             idj0 = corres[rand0].Item2;
495             idi1 = corres[rand1].Item1;
496             idj1 = corres[rand1].Item2;
497             idi2 = corres[rand2].Item1;
498             idj2 = corres[rand2].Item2;
499
500             // collect 3 points from i-th fragment
501             Vector pti0 = pointcloud_[fi][idi0];
502             Vector pti1 = pointcloud_[fi][idil];
503             Vector pti2 = pointcloud_[fi][idi2];
504
505             float li0 = norm ((Vector) (pti0 - pti1));
506             float li1 = norm ((Vector) (pti1 - pti2));
507             float li2 = norm ((Vector) (pti2 - pti0));

```

```

502
503         // collect 3 points from j-th fragment
504         Vector ptj0 = pointcloud_[fj][idj0];
505         Vector ptj1 = pointcloud_[fj][idj1];
506         Vector ptj2 = pointcloud_[fj][idj2];
507
508         float lj0 = norm ((Vector) (ptj0 - ptj1));
509         float lj1 = norm ((Vector) (ptj1 - ptj2));
510         float lj2 = norm ((Vector) (ptj2 - ptj0));
511
512         if ((li0 * scale < lj0) && (lj0 < li0 /
513             → scale) &&
514             (li1 * scale < lj1) && (lj1 < li1 /
515                 → scale) &&
516                 (li2 * scale < lj2) && (lj2 < li2 /
517                     → scale))
518             {
519                 corres_tuple.Add((idi0, idj0));
520                 corres_tuple.Add((idi1, idj1));
521                 corres_tuple.Add((idi2, idj2));
522                 cnt++;
523             }
524
525         if (cnt >= tuple_max_cnt_)
526             break;
527     }
528
529     Debug.LogFormat("{0} tuples ({1} trial, {2}
530     → actual).\n", cnt, number_of_trial, i);
531     corres.Clear();
532
533     for (i = 0; i < corres_tuple.Count; ++i)
534         corres.Add((corres_tuple[i].Item1,
535             → corres_tuple[i].Item2));
536     }
537
538     if (swapped)
539     {
540         List<int, int> temp = new List<(int, int)>
541             ();
542         for (int i = 0; i < corres.Count; i++)
543             temp.Add((corres[i].Item2,
544                 → corres[i].Item1));
545         corres.Clear();
546         corres = temp;
547     }
548
549     Debug.LogFormat("\t[final] matches {0}.\n",
550         (int)corres.Count);
551     corres_ = corres;
552 }
553
554 // Normalize scale of points.

```

```

547 // X' = (X - |mu|)/scale
548 public void NormalizePoints()
549 {
550     int num = 2;
551     float scale = 0;
552
553     Means.Clear();
554
555     for (int i = 0; i < num; ++i)
556     {
557         float max_scale = 0;
558
559         // compute mean
560         Vector mean = new DenseVector (3);
561         mean.Clear();
562
563         int npti = pointcloud_[i].Count;
564         for (int ii = 0; ii < npti; ++ii)
565         {
566             Vector p = (Vector) new DenseVector (new
567                 float [] {pointcloud_[i][ii][0],
568                         pointcloud_[i][ii][1],
569                         pointcloud_[i][ii][2]});
570             mean = (Vector) (mean + p);
571         }
572         mean = (Vector) (mean / npti);
573         Means.Add(mean);
574
575         Debug.LogFormat ("normalize points ::\n"
576             → mean[0] = [{1} {2} {3}]\n", i, mean[0],
577             → mean[1], mean[2]);
578
579         for (int ii = 0; ii < npti; ++ii)
580         {
581             pointcloud_[i][ii][0] -= mean[0];
582             pointcloud_[i][ii][1] -= mean[1];
583             pointcloud_[i][ii][2] -= mean[2];
584         }
585
586         // compute scale
587         for (int ii = 0; ii < npti; ++ii)
588         {
589             Vector p = (Vector) new DenseVector (new
590                 float [] {pointcloud_[i][ii][0],
591                         pointcloud_[i][ii][1],
592                         pointcloud_[i][ii][2]});
593             float temp = norm (p); // because we
594             → extract mean in the previous stage.
595             if (temp > max_scale)
596                 max_scale = temp;
597         }
598
599         if (max_scale > scale)

```

```

591             scale = max_scale;
592         }
593
594         //// mean of the scale variation
595         if (use_absolute_scale_) {
596             GlobalScale = 1.0f;
597             StartScale = scale;
598         } else {
599             GlobalScale = scale; // second choice: we keep
600             → the maximum scale.
601             StartScale = 1.0f;
602         }
603         Debug.LogFormat("normalize points :: global scale
604             → : {0}\n", GlobalScale);
605
606         for (int i = 0; i < num; ++i)
607         {
608             int npti = pointcloud_[i].Count;
609             for (int ii = 0; ii < npti; ++ii)
610             {
611                 pointcloud_[i][ii][0] /= GlobalScale;
612                 pointcloud_[i][ii][1] /= GlobalScale;
613                 pointcloud_[i][ii][2] /= GlobalScale;
614             }
615         }
616
617         public double OptimizePairwise(bool decrease_mu_)
618     {
619         Debug.LogFormat("Pairwise rigid pose
620             → optimization\n");
621
622         double par;
623         int numIter = iteration_number_;
624         TransOutput_ = Matrix4x4.identity;
625
626         par = StartScale; // mu
627
628         int i = 0;
629         int j = 1;
630
631         // make another copy of pointcloud_[j].
632         Points pcj_copy = new Points ();
633         int npcj = pointcloud_[j].Count;
634         for (int cnt = 0; cnt < npcj; cnt++)
635             pcj_copy.Add (pointcloud_[j][cnt]);
636
637         if (corres_.Count < 10)
638             return -1;
639
640         List<double> s = buildList (corres_.Count, 1.0);
641
642         Matrix4x4 trans = Matrix4x4.identity;

```

```

641
642     for (int itr = 0; itr < numIter; itr++) {
643
644         // graduated non-convexity.
645         if (decrease_mu_)
646         {
647             if (itr % 4 == 0 && par > max_corr_dist_) {
648                 par /= div_factor_;
649             }
650         }
651
652         const int nvariable = 6;           // 3 for
653             → rotation and 3 for translation
654         Matrix JTJ = (Matrix) Matrix.Build.Dense
655             → (nvariable, nvariable, 0.0f);
656         Matrix JTr = (Matrix) Matrix.Build.Dense
657             → (nvariable, 1, 0.0f);
658         Matrix J;
659
660         double r;
661         double r2 = 0.0;
662
663         for (int c = 0; c < corres_.Count; c++) {
664             int ii = corres_[c].Item1;
665             int jj = corres_[c].Item2;
666             Vector p, q;
667             p = pointcloud_[i][ii];
668             q = pcj_copy[jj];
669             Vector rpq = (Vector) (p - q);
670
671             int c2 = c;
672
673             float temp = (float) (par /
674             → (rpq.DotProduct(rpq) + par));
675             s[c2] = temp * temp;
676
677             J = (Matrix) Matrix.Build.Dense
678             → (nvariable, 1, 0.0f);
679             J[1, 0] = -q[2];
680             J[2, 0] = q[1];
681             J[3, 0] = -1;
682             r = rpq[0];
683             JTJ = (Matrix) (JTJ + J * J.Transpose() *
684             → (float) s[c2]);
685             JTr = (Matrix) (JTr + J * (float) (r *
686             → s[c2]));
687             r2 += r * r * s[c2];
688
689             J = (Matrix) Matrix.Build.Dense
690             → (nvariable, 1, 0.0f);
691             J[2, 0] = -q[0];
692             J[0, 0] = q[2];
693             J[4, 0] = -1;

```

```

686             r = rrpq[1];
687             JTJ = (Matrix) (JTJ + J * J.Transpose() *
688             ↪ (float) s[c2]);
689             JTr = (Matrix) (JTr + J * (float) (r *
690             ↪ s[c2]));
691             r2 += r * r * s[c2];
692
693             J = (Matrix) Matrix.Build.Dense
694             ↪ (nvariable, 1, 0.0f);
695             J[0, 0] = -q[1];
696             J[1, 0] = q[0];
697             J[5, 0] = -1;
698             r = rrpq[2];
699             JTJ = (Matrix) (JTJ + J * J.Transpose() *
700             ↪ (float) s[c2]);
701             JTr = (Matrix) (JTr + J * (float) (r *
702             ↪ s[c2]));
703             r2 += r * r * s[c2];
704
705             r2 += (par * (1.0 - Math.Sqrt(s[c2])) *
706             ↪ (1.0 - Math.Sqrt(s[c2])));
707         }
708
709         Matrix result;
710         result = (Matrix) (-JTJ.Solve(JTr)); ////
711         ↪ Removed LLT
712
713         Matrix4x4 aff_mat = new Matrix4x4 ();
714         aff_mat.SetTRS (new Vector3 (result[3, 0],
715             ↪ result[4, 0], result[5, 0]),
716             ↪ Quaternion.AngleAxis (result[2, 0] *
717             ↪ Mathf.Rad2Deg, Vector3.forward) *
718             ↪ Quaternion.AngleAxis (result[1, 0] *
719             ↪ Mathf.Rad2Deg, Vector3.up) *
720             ↪ Quaternion.AngleAxis (result[0, 0] *
721             ↪ Mathf.Rad2Deg, Vector3.right),
722             ↪ Vector3.one);
723
724         Matrix4x4 delta = aff_mat;
725
726         trans = delta * trans;
727         TransformPoints(pcj_copy, delta);
728
729     }
730
731     TransOutput_ = trans * TransOutput_;
732     return par;
733 }
734
735 private void TransformPoints(Points points, Matrix4x4
736     ↪ Trans)
737 {
738     int npc = (int)points.Count;
739     Vector3 temp;

```

```

723     for (int cnt = 0; cnt < npc; cnt++) {
724         temp = Trans.MultiplyPoint (new Vector3
725             → (points[cnt][0], points[cnt][1],
726                 → points[cnt][2]));
727         points[cnt] = new DenseVector (new float [] {
728             → temp.x, temp.y, temp.z });
729     }
730 }
731
732 public List <Vector3> GetTransformedPoints (List
733     → <Vector3> src)
734 {
735     Points pcj_copy = new Points ();
736     int npcj = src.Count;
737     for (int cnt = 0; cnt < npcj; cnt++) {
738         pcj_copy.Add (new DenseVector (new float [] {
739             → src[cnt].x, src[cnt].y, src[cnt].z }));
740         TransformPoints(pcj_copy, GetOutputTrans ());
741
742         List <Vector3> v = new List <Vector3> ();
743         for (int cnt = 0; cnt < npcj; cnt++)
744         {
745             v.Add (new Vector3 (pcj_copy[cnt][0],
746                 → pcj_copy[cnt][1], pcj_copy[cnt][2]));
747         }
748         return v;
749     }
750
751 public Matrix4x4 GetOutputTrans()
752 {
753     Quaternion R = TransOutput_.rotation;
754     Vector3 t = new Vector3 (TransOutput_[0, 3],
755         → TransOutput_[1, 3], TransOutput_[2, 3]);
756     Matrix4x4 transtemp = TransOutput_;
757     Vector3 tt = -(R* new Vector3 (Means[1][0],
758         → Means[1][1], Means[1][2])) + t*GlobalScale +
759         → new Vector3 (Means[0][0], Means[0][1],
760             → Means[0][2]);
761     transtemp[0, 3] = tt.x;
762     transtemp[1, 3] = tt.y;
763     transtemp[2, 3] = tt.z;
764
765     return transtemp;
766 }
767 }
```

3. To demonstrate the use of the registration process, we need to be able to extract point clouds from the base object that we created in the previous steps. Add a new object to the scene - anything with 3D geometry will do, such as a cube or sphere. Switch to the scene view

and click on the Gizmos button to enable drawing of gizmos. You might want to switch the rendering type from shaded to wireframe in order to better see the details. Now add the ShowVertices component from Algorithm 18.17 to this object. The vertices in the geometry mesh should now be highlighted. ShowVertices uses the gizmo rendering facility to draw spheres at the positions occupied by the mesh vertices. Normally these would be invisible if the mesh has no faces, making it impossible to see a point cloud. Convert this object to a template named PointCloudTemplate by dragging it to the Assets area.

Code Listing 18.17: ShowVertices. The highlighting of vertices is just available in the Scene view.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ShowVertices : MonoBehaviour
6  {
7      void OnDrawGizmosSelected()
8      {
9          Mesh m = GetComponent <MeshFilter> ().mesh;
10         Vector3 [] v = m.vertices;
11         int vlen = v.Length;
12
13         for (int i = 0; i < vlen; i++)
14         {
15             Gizmos.color = Color.yellow;
16             Gizmos.DrawSphere (transform.TransformPoint
17             (v[i]), 0.01f);
18         }
19     }
}

```

4. The next component extracts two point clouds from the reference object, randomly transforms one, and then tries to realign them. Add this to a new empty object in the scene. You will need to provide the base object from the first step as the Point Cloud Source, and the PointCloudTemplate as the Cloud Template.

Code Listing 18.18: TestMeshAlign. Point cloud alignment is demonstrated by creating two clouds from the same source, randomly changing the pose of one, and then trying to align it with the other.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class TestMeshAlign : MonoBehaviour
6  {
7      public MeshFilter pointCloudSource;
8
9      public GameObject cloudTemplate;
10
11     public int pointCloudSize = 100;
12
13     private List <Vector3> extractRandomCloud (int n, bool
14     scramble)
15     {
16         List <Vector3> p = new List <Vector3> ();
17
18         Mesh m = pointCloudSource.GetComponent
19         <MeshFilter> ().mesh;
20         Vector3 [] v = m.vertices;
21         int vlen = v.Length;
22
23         Matrix4x4 randTrans = Matrix4x4.identity;
24         if (scramble)
25         {
26             Vector3 t = Random.insideUnitSphere;
27             float angle = Random.Range (0.0f, 360.0f);
28             Vector3 direction = Random.insideUnitSphere;
29             randTrans.SetTRS (t, Quaternion.AngleAxis
30             (angle, direction), Vector3.one);
31             Debug.Log ("Transformation: " + t.ToString
32             ("F4") + " " + angle.ToString ("F4") + " - "
33             + direction.ToString ("F4") + "\n\n" +
34             randTrans.ToString ("F4"));
35         }
36
37         for (int i = 0; i < n; i++)
38         {
39             p.Add (randTrans.MultiplyPoint (v[Random.Range
40             (0, vlen)]));
41         }
42
43         return p;
44     }
45
46     private void displayCloud (List <Vector3> p)
47     {
48         Vector3 [] vertices = p.ToArray ();
49
50         GameObject g = Instantiate (cloudTemplate);
51         Mesh m = g.GetComponent <MeshFilter> ().mesh;
```

```

45         m.indexFormat =
46             UnityEngine.Rendering.IndexFormat.UInt32;
47         m.vertices = vertices;
48         m.triangles = new int [0];
49         m.RecalculateNormals();
50     }
51
52     void Start()
53     {
54         List <Vector3> p1 = extractRandomCloud
55             (pointCloudSize, false);
56         List <Vector3> p2 = extractRandomCloud
57             (pointCloudSize, true);
58
59         displayCloud (p1);
60         displayCloud (p2);
61
62         FastGlobalRegistration fgr = new
63             FastGlobalRegistration ();
64         fgr.AddFeature (p1);
65         fgr.AddFeature (p2);
66         fgr.NormalizePoints();
67         fgr.AdvancedMatching();
68         fgr.OptimizePairwise(true);
69     }
}

```

Depending on the parameters used this can take some time (several minutes for complex objects) so you will need to wait until the play button is highlighted before you can view the result. To see the result, again switch to the Scene view. Three clones of the PointCloudTemplate are created. The first point cloud is sampled from the base object but not transformed. The second point cloud is sampled from the base object and randomly transformed. The third point cloud is the second point cloud, realigned to the first point cloud (and hence the base object). As you switch between these and see the vertices, ideally first and third clouds should be aligned with the base object even though the set of vertices used may be different.

5. Do experiment with the various parameters. The number of points sampled to create the point clouds is a

good place to start. This should give a feel for how densely the points need to be spread over the surface to find enough distinguishing features to achieve alignment. Other parameters to change (these are actually inside the code for the `FastGlobalRegistration` class) include the neighbourhood radius, and number of neighbours to target. This will affect the way in which the feature descriptor represents the feature. If the detail is too sparse then many spurious correspondences will be created. Too much detail may risk being too specific, and it will cause difficulty in identify similar (but not exactly the same) regions.



# 19

## *Interaction and Feedback Components*

### Contents

---

19.1	Marker tracking with Vuforia . . . . .	379
19.2	Marker tracking with AR Core . . . . .	383
19.3	A handheld controller that can be dynamically adapted to the needs of the experience . . . . .	387
19.4	Interaction using gesture recognition with the Leap Motion . . . . .	400
19.5	Interaction using gesture recognition with the Kinect . . . . .	412
19.6	Managing interaction using body tracking on a general purpose device . . . . .	416
19.7	Tracking facial expression and head pose . . . . .	438
19.8	Supporting interaction using speech recognition . . . . .	458

---

### 19.1 Marker tracking with Vuforia

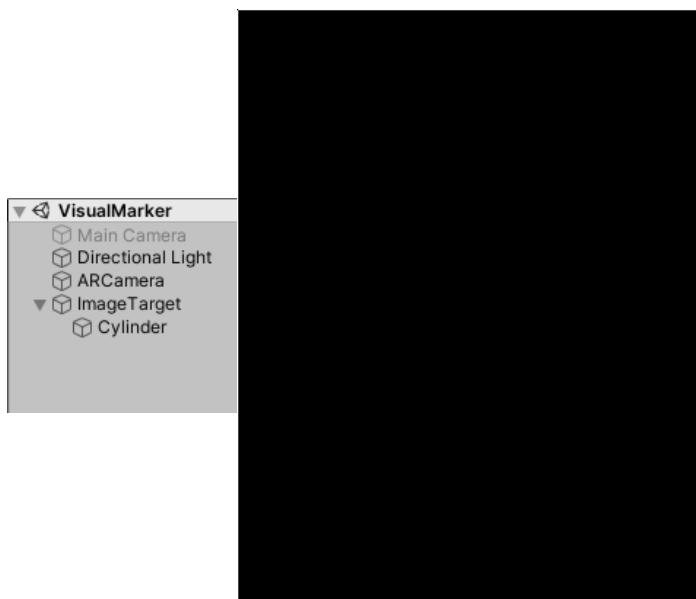
Component 19.1.1: Marker tracking with Vuforia.

Visual

Marker

*A number of suppliers offer components that add visual marker tracking facilities. Vuforia is one such organization and their software is currently included as part of Unity software. This example demonstrates the development of a standardized augmented reality component as abused by countless applications already. A 3D model is associated with a particular marker, and appears whenever the marker is recognized. The only aspect of tracking demonstrated is the placement of the marker relative to the camera; no consideration is given to other elements in the world.*

Figure 19.1.1:  
Tracking markers  
with Vuforia.



1. Start a new unity project. Start the package manager (under the Window menu option) and ensure that the 'Vuforia Engine AR' package is installed.

In older versions of the Unity software you would enable 'Vuforia Augmented Reality Supported' under player settings, and then XR Settings. Consider any license agreement that is presented. In newer versions of the Unity software, this is achieved by installing the Vuforia Engine AR package using the Package Manager option on the Windows menu.

2. Having enabled Vuforia, there are now extra options available under the Create button in the Hierarchy tab. Create a new ARCamera, and remove the default Main Camera. The AR Camera component displays the view from an attached physical camera.
3. The next step is to configure various target images for Vuforia to track. This involves a diversion via the Vuforia web site to create the required resources in the appropriate format.

The ARCamera has a Vuforia Behaviour component. Part of this is a button to access to the Vuforia configuration settings. Open the Vuforia configuration.

Marker recognition requires a database of target markers. Select Add Database, which then opens a browser to the Vuforia site.

Create an account if you do not already have one. First time use also requires a license key associated with each application that you develop. Use the option provided to get a development key, which just requires you provide a name for the application you're working on. Add your developer key to the Unity project, under the appropriate section of the Vuforia configuration.

On the same web site, switch to the target manager tab. Individual target images are grouped into databases. First add a database for use on your device, and then add a sequence of targets (images) to that database. These images should be things that you also have in the physical world. Images that you can print out will work, as will photographs of feature rich regions of the room.

Once an image is uploaded it receives a rating indicating the richness of the feature points that are available. Clicking on the image in the database allows the option of inspecting the feature points found for the image.

Once all the images have been added to the database, download it as a unity package file. Clicking on the

downloaded file should open a prompt to import it into your project. The databases that have been imported are listed above the Add Database button in the Vuforia configuration.

4. Now add an image target to the scene (under Create/Vuforia/Image). Place this at the origin, and configure the image target behaviour property to select both your database, and one of the target images from that database. When that image is recognized, then the image target will activate any content that it is a parent of.

Create a geometric object and make it a child of the image target. Run your application and confirm that the object is displayed attached to the image of the target object.

5. You should also be able deploy the application to a mobile device. Change the build settings as appropriate. Note that you may have reenable use of Vuforia in the XR Settings, and disable building for Android TV (an option in the Other Settings under the Player Settings).
6. Some further options to explore include:

- (a) Have multiple targets being tracked at the same time. This introduces possibilities associated with having objects on one target interact with objects on another. Hint: have a look at the Vuforia configuration options and see what can be quickly achieved.
- (b) Having more than one of a given target image being tracked at the same time does not seem to be directly supported by Vuforia. However a workaround involves adding the same image multiple times to a database. A new image target needs to be added to the scene for each duplicate database entry.
- (c) Download the Vuforia object scanner application. Tracking of physical 3D objects requires that a set of

photographs from different angles be collected. Scan an object that you have available and add it to your application as an object target.

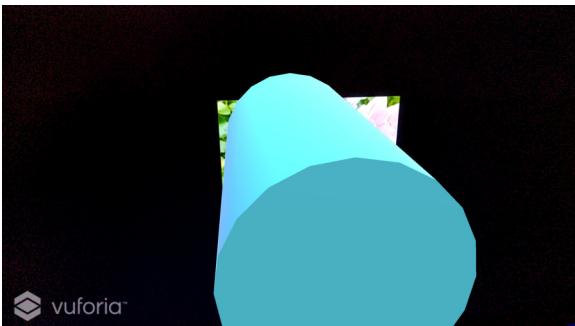


Figure 19.1.2:  
Virtual object  
registered relative  
to the marker's  
position.

## 19.2 Marker tracking with AR Core

### Component 19.2.1: Marker tracking with AR Core.

Marker
Tracking

*AR Core is another library for supporting augmented reality applications. This has a greater dependence on Android and will need an Android phone connected during development for access to the camera on the phone.*

*This example also demonstrates basic visual marker tracking with AR Core. This library has a number of other relevant features which are further explored in later examples.*

1. Start a new unity project. Configure it as an Android project under the Build settings (File/Build Settings), and set up the usual parameters associated with an Android project. Under player settings, and then XR Settings, enable ARCore Supported.
2. Both the Unity software and ARCore under go rapid changes, making it tricky for each to remain compatible with updates to the other package. Some further changes are required. These specific instructions related



Figure 19.2.1:  
Marker tracking  
with AR Core.

to Unity software version 2019.4.3f1 and ARCore SDK for Unity version 1.18.0.

- (a) ARCore makes use of a Unity component named TrackedPoseDriver which is not longer included with the Unity software. This is available in the XR Legacy Input Helpers package which needs to be installed using the package manager (under Window/Package Manager).
- (b) A number of settings are required in the project settings, under player settings:
  - i. As before, make sure the ARCore Supported is enabled under the XR Settings. The project does need to be set to build to Android to access this.
  - ii. Under Other Settings, remove Vulkan as a Graphics API, leaving only OpenGL ES3.
  - iii. Disable Multithreaded Rendering.
  - iv. Set the minimum API level to 29.
- (c) Once you've downloaded and imported the ARCore SDK (next step) you are likely to get errors from the CloudAnchors example. This is not an issue with cloud anchors, but rather with the Unity networking functionality which has been deprecated. It is possible to installed the Multiplayer HLAPI package to provide these dependencies, or

alternatively the entire GoogleARCore/Examples/-CloudAnchors folder can just be deleted if you're not planning to refer to this example.

- (d) Once you've downloaded and imported the ARCore SDK (next step), it is worth deploying the HelloAR example to verify that everything is working as expected before trying to create your own scene. This might avoid pointless debugging resulting from compatibility issues.
  - (e) The permission dialog used by ARCore applications may not pop up automatically. If the application is failing on startup, then it is worth going into the phone's settings, finding the application and manually enabling permissions, particularly for the camera. Also double check that the settings changed above have retained their values after importing the ARCore SDK.
3. Download the ARCore SDK for Unity from: <https://github.com/google-ar/arcore-unity-sdk/releases> (the relevant unitypackage may be at the bottom of the page). Import this package into the project.
  4. Add an ARCore Device prefab to the scene from the Prefabs folder supplied with the ARCore SDK. The default Main Camera object can be disabled. This provides the connection to the physical camera and displays the camera image on the screen of the Android device.
  5. In the Assets folder in Unity, create:
    - (a) An ARCore Session Config (under Create/GoogleARCore/Session Config). You might want to set the camera focus mode to auto which helps adapt the camera focus for close up views. Assign the object to the Session Config property of the ARCore Device object.
    - (b) Copy your target image files to a folder in your project. Select them all, and then create an image

database (Create/GoogleARCore/AugmentedImageDatabase). You can optionally set a width for each image in the database which helps determine the depth of the marker quickly without requiring camera movement to estimate the size.

- (c) Add the database created to the appropriate field of the ARCore Session Config that you created previously.
6. Attaching objects to tracked images requires a scripted element. Create an object that will be shown on any tracked element, and set its size to be about the same size as a marker object. Create an empty object called ARShowObject, create a new C# script called ShowObjectOnTrackable and include the code provided in Algorithm 19.1. Drag the object into the displayObject property of this script. The script demonstrates how to retrieve the position and rotation of each currently tracked object.

Code Listing 19.1: ShowObjectOnTrackable. A component that finds all currently trackable markers in the scene and attaches an object to one of them.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using GoogleARCore;
5
6  public class ShowObjectOnTrackable : MonoBehaviour {
7
8      public GameObject displayObject;
9
10     private List<AugmentedImage> trackedImages = new
11         List<AugmentedImage>();
12
13     // Update is called once per frame
14     void Update () {
15         displayObject.SetActive (false);
16
17         Session.GetTrackables<AugmentedImage> (trackedImages,
18             TrackableQueryFilter.All);
19         foreach (var image in trackedImages)
20         {
21             if (image.TrackingState == TrackingState.Tracking)
22             {

```

```

21     displayObject.transform.position =
22     image.CenterPose.position;
23     displayObject.transform.rotation =
24     image.CenterPose.rotation;
25     displayObject.SetActive (true);
26   }
27 }
```

The script does behave strangely once more than one marker is tracked, as the object ends up on the last tracked object on the list. This is relevant since AR Core does keep track of markers even when they are not visible in the camera view. An immediate extension to this project would be to place an object on each marker that is being tracked.

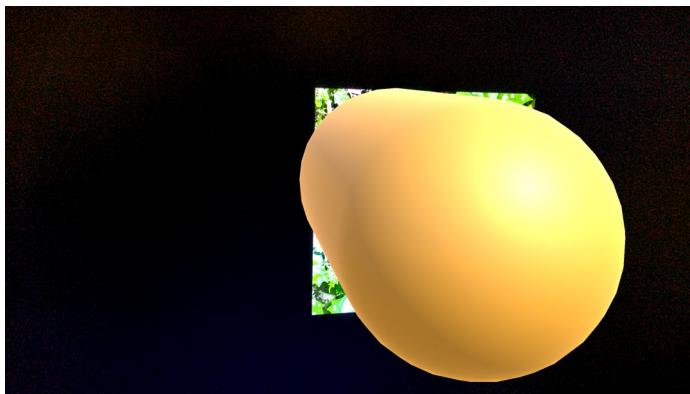


Figure 19.2.2:  
Adding objects to  
markers.

### 19.3 A handheld controller that can be dynamically adapted to the needs of the experience

#### Component 19.3.1: Flexible controller.

Mobile devices have been largely used as a tool for perceiving and experiencing virtual content. Some of the examples have also shown ways in which participants can interact with content at the same time. This example focuses on re-purposing a

Flexible  
Controller

*mobile device primarily as a control element. Smart phones have most of the hardware used in many controllers as is evidenced by the existence of simulators for some devices that run as smart phone apps.*

*This distinction also highlights another assumption about mobile devices; that only one per user is allowed. This example uses a controller derived from a mobile device in conjunction with other equipment. Such a setup may have one, fixed or mobile, device used to perceive the augmented experience, with one or more additional devices used to provide controllers. There are contexts where other permutations may be relevant. Sensing and tracking movements in multiple limbs may be required Jablonsky et al. [2017], or different types of controller may be required to effectively interact using multiple tools.*

*The characteristics of a smart phone based controller that this example demonstrates are:*

Figure 19.3.1: The flexible controller displays a customized set of controls.

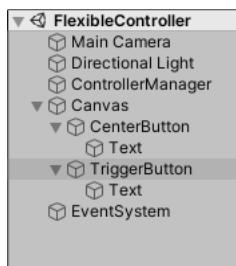
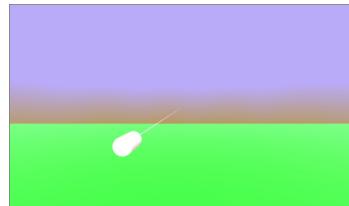
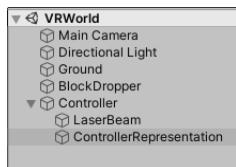


Figure 19.3.2:  
Applications  
employ the flexible  
controller in ways  
suited to their  
purpose.



1. As illustrated with previous examples (component 18.3.1) the gyroscope in the device allows it to be used as a

3DOF (three degrees of freedom) controller where the orientation of the device can be provided. This is sufficient to create a 3D mouse, or directional pointer. Positional tracking examples suggest that this could be extended to a 6DOF device using inside-out tracking from the device's camera (component 18.2.1).

2. Flexible and dynamic allocations of buttons. Virtual buttons and controls are provided via the touch screen of the device. These can be customized according to the application being used, or even the current context.
3. Wireless communication to other physical components in the system. The control information may be relevant to multiple (i.e. more than one) other nodes in the system (i.e. the controller may not be associated with a particular participant, but rather with an aspect of the experience by making smart objects).
4. Use of the screen and other output modalities to present details relevant to the affordances of the controller. The images shown can be used to present feedback to the participants or to customize the appearance of the controller.

This example is built as a single project but consists of two components that are represented with separate scenes. One scene presents a simple virtual world and allows testing of the controller. The other scene is the controller itself, and will need to be deployed onto a mobile platform.

1. Start with a new project. Create two scenes, one labelled VRWorld, and the other named FlexibleController. Set the build settings to support mobile platforms. During development the VRWorld scene runs on the desktop development platform, while the FlexibleController runs on the mobile device. In the build settings, ensure that both scenes are added to the build but that only the FlexibleController is enabled.

2. The VRWorld scene can be populated with some basic content, as described for component 18.3.1. Create an additional object to represent the controller and place this somewhere visible in the camera view (at about where a hand based controller would be relative to the camera).

Create a new C# component called ControllerInformationSink and add this to the controller object. This component will pick up information from the controllers and use this to manage interaction with this virtual world.

3. The FlexibleController scene requires an empty object to serve as the controller manager with a C# script called ControllerInformationSource attached. Also create two UI Buttons to serve as a minimum set of basic controls for a generic controller. One is used to calibrate the rest position of the controller and can be labelled “Center”. The other is the trigger input and labelled “Trigger”.

The click event for these buttons is likely to be too fast. A missed packet during communication may cause the event to be lost. Instead of using the click event on the buttons, add additional “Event Trigger” components in the Inspector panel. Create entries for Pointer Down and Pointer Up and connect these to the appropriate entries on the controller manager once the ControllerInformationSource component is complete.

The script for the ControllerInformationSource is provided in Algorithm 19.2.

Code Listing 19.2: ControllerInformationSource. The basic controller connects to two user defined buttons, and to the device’s gyroscope and transmits this information to other components in the application.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
```

```

6  public class ControllerInformationSource : MonoBehaviour {
7
8      private Gyroscope gyro;
9      private bool gyroSupported;
10
11     private DatagramCommunication dc;
12     private Quaternion gyroAttitude;
13     private bool centerPressed;
14     private bool triggerPressed;
15
16     void Start()
17     {
18         dc = new DatagramCommunication ();
19
20         gyroSupported = SystemInfo.supportsGyroscope;
21         if (gyroSupported)
22         {
23             gyro = Input.gyro;
24             gyro.enabled = true;
25         }
26     }
27
28     public void centerClicked () { centerPressed = true; }
29     public void triggerClicked () { triggerPressed = true; }
30     ←
31     public void centerRelease () { centerPressed = false; }
32     ←
33     public void triggerRelease () { triggerPressed = false; }
34     ←
35     void Update () {
36         if (gyroSupported)
37         {
38             gyroAttitude = gyro.attitude;
39         }
40         dc.sendControllerDetails (new ControllerDetails
41             ←
42             (gyroAttitude, centerPressed, triggerPressed));
43     }
44 }
```

4. Back in the VRWorld scene, the ControllerInformation-Sink receives this data from the controller and takes appropriate actions. The rotation of the controller object is controlled using the data from the remote device's gyroscope.

The center button is used to set the neutral position of the controller. The variable centeredAttitude represents the inverse of the orientation of the controller when the center button is pressed. Multiplying the orientation

by this factor cancels out this portion of the pose of the controller.

The trigger is a more flexible control that we can adapt to the context. In this case we'll use it to activate a laser beam, and inflate objects that it is aimed at until they pop. Create a long narrow cylinder object to represent the laser beam and make it a child of the controller. Make sure the collider on the beam is removed. The script ensures this beam is only visible when the trigger is pressed. Make sure the beam object is passed as a property to the script component.

The beam only applies to objects that have the tag "Inflatable". Add this tag (in the Inspector panel) and set it on every object that the beam should affect. Some sound effects can also be provided. If required, add some AudioSource components to the controller object and supply them with appropriate audio files. Make sure to switch off the "Play On Awake" option.

The script for the ControllerInformationSink is given in Algorithm 19.3.

Code Listing 19.3: ControllerInformationSink. Information from the remote controller is used for interaction in the virtual world. Some aspects are common to most scenarios, such as updating orientation and supporting re-centering. Other functionality is also broadly useful, such as using a pointer mechanic to select objects.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ControllerInformationSink : MonoBehaviour {
6
7      [Tooltip ("A cylindrical beam object attached to the
8      ↵ controller.")]
9      public GameObject laserBeam;
10     [Tooltip ("An adjustment factor for how fast objects
11     ↵ blow up.")]
12     public float inflationRate = 1.0f;
13     [Tooltip ("A sound effect played during inflation.")]
14     public AudioSource hiss;
15     [Tooltip ("A sound effect played when the object is
16     ↵ destroyed.")]
17     public AudioSource pop;

```

```

16 // Link to the network functions.
17 private DatagramCommunication dc;
18 // The controller orientation (inverse) for the centered
19 // pose.
20 private Quaternion centeredAttitude;
21
22 void Start () {
23     dc = new DatagramCommunication ();
24     centeredAttitude = Quaternion.identity;
25 }
26
27 void Update () {
28     ControllerDetails cd = dc.receiveControllerDetails ();
29     // decay hiss so it stops if no button is pressed.
30     hiss.volume *= 0.9f;
31     if ((cd != null))
32     {
33         // calculate the rotation to cancel out the current
34         // pose.
35         if (cd.center)
36         {
37             centeredAttitude = Quaternion.Inverse
38             (Quaternion.Euler (90, 0, 90) * new Quaternion
39             (cd.gyrox, cd.gyroy, cd.gyroz, cd.gyrow) *
40             Quaternion.Euler (180, 180, 0));
41         }
42         // make the laser beam active if the trigger is
43         // pressed.
44         laserBeam.SetActive (cd.trigger);
45         if (cd.trigger)
46         {
47             // Raycast, inflate, explode.
48             RaycastHit hit;
49             if ((Physics.Raycast(transform.position,
50                 transform.forward, out hit, Mathf.Infinity)) &&
51                 (hit.collider.gameObject.tag == "Inflatable"))
52             {
53                 // Inflate the object by manipulating scale.
54                 hit.collider.gameObject.transform.localScale *=
55                 1.0f + (inflationRate * Time.deltaTime);
56                 // Play the inflation sound.
57                 if (hiss != null) { hiss.volume = 1.0f; if
58                     (!hiss.isPlaying) hiss.Play (); }
59                     // Pop the object if it gets too big.
60                     if (hit.collider.gameObject.transform.localScale]
61                     .magnitude >
62                     3)
63                     {
64                         Destroy (hit.collider.gameObject);
65                         if (pop != null) pop.Play ();
66                     }
67                 }
68             }
69         }
70     }

```

```

58      // Match the pose of the virtual controller to that
59      ← of the remote controller device.
60      ← transform.rotation = Quaternion.Euler (90, 0, 90) *
61      ← new Quaternion (cd.gyrox, cd.gyroy, cd.gyroz,
62      ← cd.gyrow) * Quaternion.Euler (180, 180, 0) *
63      ← centeredAttitude;
64
65    }
66  }
67
68 }
```

5. The communication between controller and application makes use of similar networking techniques to those used in component 21.5.1. Significant differences however lie in the use of a datagram based protocol for sending the controller updates. These offer direct communication opportunities that are open to packet loss, but otherwise have very little delay associated with any communication overheads.

Another key difference is the use of a broadcast mechanism to communicate the messages. This means that all devices on the local network will receive these messages (often including the sender). Some network administrators may object, but this does save on having to define destination addresses and allows new controllers to be added with minimal overhead. Broadcast messages are usually constrained to a local sub-network. It is expected that all devices making up the components of an application using these controllers are on the same sub-network.

The unit of communication is the ControllerDetails object which contains the fields that are being used by this particular version of the flexible controller component. Fields include:

- An identifier for the controller, in case you don't want any controller in the region to work in your application.
- The orientation of the controller.
- Some boolean variables holding the state of software defined buttons on the controller.

Serialization functions have been built directly into this class, as shown in Algorithm 19.4.

Code Listing 19.4: ControllerDetails. The controller state is represented by this class. The fields listed can be modified to support other variations on this controller pattern.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System;
5  using System.Runtime.Serialization.Formatters.Binary;
6  using System.IO;
7
8  [Serializable]
9  public class ControllerDetails {
10
11    public string id;
12    public float gyrox;
13    public float gyroy;
14    public float gyroz;
15    public float gyrow;
16    public bool center;
17    public bool trigger;
18
19    public ControllerDetails (Quaternion ori, bool centre,
20    ←   bool trig)
21    {
22      id = getID ();
23      gyrox = ori.x;
24      gyroy = ori.y;
25      gyroz = ori.z;
26      gyrow = ori.w;
27      center = centre;
28      trigger = trig;
29    }
30
31    public byte [] serialize ()
32    {
33      BinaryFormatter bf = new BinaryFormatter ();
34      MemoryStream ms = new MemoryStream ();
35      bf.Serialize (ms, this);
36      return ms.ToArray ();
37    }
38
39    public static ControllerDetails deserialize (byte [] b)
40    {
41      BinaryFormatter formatter = new BinaryFormatter();
42      ControllerDetails cd = (ControllerDetails)
43      ← formatter.Deserialize (new MemoryStream (b));
44      return cd;
45    }
46 }
```

```
45 public static string getID ()  
46 {  
47     return SystemInfo.deviceUniqueIdentifier;  
48 }  
49 }
```

6. The connection-less networking approach used here simplifies significantly the complexity of the networking functions. These can be reduced to just a send and receive operation. The receive operation is slightly more complex because of the need to avoid blocking the entire application while waiting for messages. However this approach does avoid the need for additional threads of execution.

The class that handles the networking is provided in Algorithm 19.5.

Code Listing 19.5: `DatagramCommunication`. Networking using datagram broadcast reduces overhead of tracking connections or managing addresses. Different types of controller can be distinguished by using a unique port number for each application.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System.Net.Sockets;
5  using System.Net;
6  using System;
7
8  public class DatagramCommunication {
9
10    private int port = 8081;
11
12    private UdpClient udpClient;
13
14    private ControllerDetails lastController;
15    private bool controlValid = false;
16
17    public DatagramCommunication ()
18    {
19        udpClient = new UdpClient ();
20        udpClient.Client.Bind (new IPEndPoint (IPAddress.Any,
21        port));
22
23        udpClient.BeginReceive (new AsyncCallback
24        (udpReceive), null);
25    }
26}
```

```

25  // Retrieve the most recently received controller
26  ↵ details.
27  public ControllerDetails receiveControllerDetails ()
28  {
29      if (controlValid)
30      {
31          controlValid = false;
32          return lastController;
33      }
34      return null;
35  }
36
37  // Wait for a new message to arrive, and set that as the
38  ↵ last message received.
39  private void udpReceive (IAsyncResult res)
40  {
41      IPEndPoint from = new IPEndPoint(0, 0);
42      byte [] buffer = udpClient.Receive(ref from);
43
44      ControllerDetails details =
45      ControllerDetails.deserialize (buffer);
46      lastController = details;
47      controlValid = true;
48
49      udpClient.BeginReceive (new AsyncCallback
50      ↵ (udpReceive), null);
51  }
52
53  // Broadcast the controller parameters.
54  public void sendControllerDetails (ControllerDetails
55  ↵ details)
56  {
57      byte [] data = details.serialize ();
58      udpClient.Send(data, data.Length, "255.255.255.255",
59      ↵ port);
60
61  }
62
63 }
```

7. An extension to consider is the use of multi-casting rather than broadcasting. This potentially extends the reach of the packets, and is slightly less aggressive in sharing information with disinterested parties.
8. What to do if this doesn't work. Network configurations and restrictions are likely to differ from network to network. Here are some suggestions if this doesn't work as described.
  - (a) The devices may require permission to access net-

work functionality. Check that suitable permission has been granted, for example under the permissions associated with the particular application under a settings menu. The player settings for the Unity software also have options that mark applications as requiring network access (Project Settings/Player/Internet Access). Permission levels are quite fluid in recent versions on Android at the time of writing and may continue to be a moving target.

- (b) The network may not like to share broadcast packets. The broadcast address used is 255.255.255.255 as set in Algorithm 19.5. You can try either the local network broadcast address (if you know how to find this), or just directly replace the address with the IP address of the destination device (the device running the VRWorld scene).
- (c) Network traffic is quite hard to debug, since you're never sure whether a message was stopped from leaving the source, was intercepted somewhere on the network or was filtered by the destination. Adding debugging information to the two scenes to report the status of messages (what is sent, what has arrived) can help narrow this down. If one of the machines is capable of running a network packet sniffer, such as wireshark (<https://www.wireshark.org/>) then this can also help identify where to investigate. Filtering on the port number (8081 is the value used in the sample code) can help identify the relevant messages amongst the flood of packets likely to be on the network.
- (d) If all else fails, review the log files on the device in case an error message is reported.

Figure 19.3.3: The interface presented on the screen of the controller device.

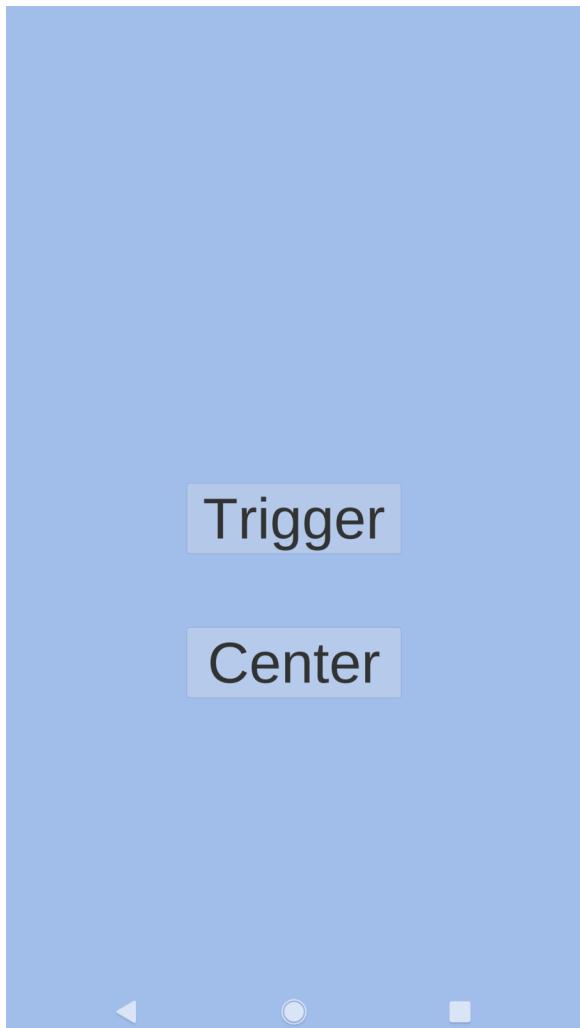


Figure 19.3.4: An application making use of a flexible controller.



## 19.4 Interaction using gesture recognition with the Leap Motion

### Component 19.4.1: Gesture recognition with Leap Motion.

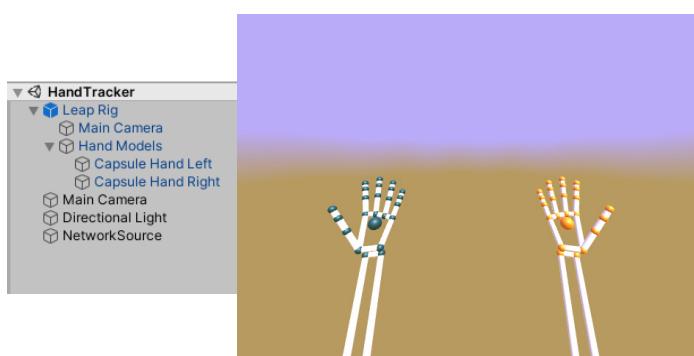
**Hand Gesture**

*This example demonstrates the process of performing gesture recognition to manage interaction in a virtual environment.*

*In this case gesture recognition is specific to hand movements and gestures and so this example makes use of the Leap Motion Controller (<https://www.ultraleap.com/>). This device does much of the work involved in tracking the hands and extracting the skeletal structure. The associated software then provides tools for reasoning about hand and finger pose and recognizing particular configurations.*

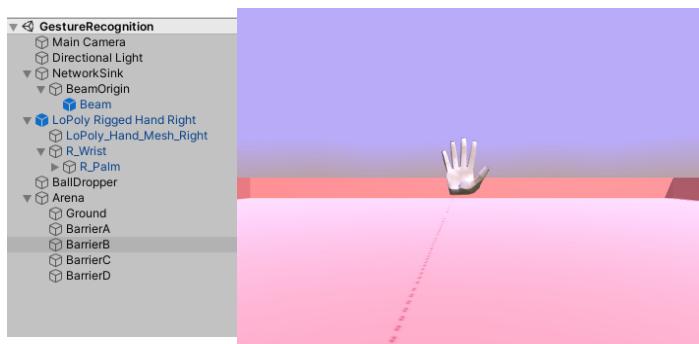
*The Leap Motion Controller can currently be employed in two modes: placed on a desktop with the hands moving above it, or attached to the front of a head mounted display so that it can track the hand movements of the person wearing the headset. It does require a wired connection to the host machine running the application. This example extends this functionality to support mobile headsets by adding a network transport layer allowing the controller to be used with mobile AR and VR devices. The controller can then either be wired to a desktop platform if the participants remain stationary, or attached to a compatible mobile and wearable device.*

Figure 19.4.1: The desktop scene, connected to the Leap Motion controller.



1. Start with a new project. Change the build settings so that it deploys on a mobile device. Download the

Figure 19.4.2: The mobile scene, receiving hand pose updates from the desktop.



Leap Motion packages from <https://developer.leapmotion.com/get-started>. In particular you need the software for the Leap Motion Controller for your particular operating system, including the drivers and control panel if these are not already installed. You also need the package for the Unity software (<https://developer.leapmotion.com/unity>) which should be downloaded and imported into the project. At time of revision, the file downloaded consists of three packages. All three can be imported into the project.

2. This example uses two scenes, one to interact with the leap motion device and track hands while the other receives hand data over a network link and performs gesture recognition. Create two scenes, with one named HandTracker and the other GestureRecognition. Set up the build settings to only deploy GestureRecognition to the mobile device.
3. In the scene HandTracker add a LeapRig prefab from the Prefabs folder in the Leap Motion Core assets. The Leap XR Service Provider component on the camera that is part of this prefab provides parameters that can be set if the leap motion controller is mounted on the desktop. Other mounting options, such as on a head mounted display, may be appropriate depending on how you intend to use the device.

The LeapRig already has some hand models included.

You can experiment with adding or replacing these with other left and right hand models from the hand models available in the Leap Motion assets. Suggested models are the: LoPoly Rigged Hand (Left and Right). The physical hand models often don't have visual representations and should be avoided.

Open up the Model Pool property on the Hand Model Manager and set the pool size to 1. Give element 0 an appropriate group name ("Hands") and fill in the model properties with the hand models just added to the scene. Set the Is Enabled property to allow this group to be used.

Run the application and ensure that the leap motion controller tracks hand movement and that this is displayed in the scene. By default the camera is quite far away from the action and can be moved closer, say to 0, 0, -1.

4. Since the leap motion controller is not supported by many (any?) mobile devices at the moment we can use network message to send state to mobile devices that require hand based input. The controller can then be attached to a fixed desktop location, or to a compatible portable device carried by the participant.

The communication process builds on the mechanism created for the flexible controller (component 19.3.1). The state of a hand is shared with the application running on the mobile device. A Leap Motion hand is represented in the Leap Motion software by the HandModelBase class which wraps the detailed hand state contained in the Hand class. Since this class is already designed for serialization it is a simple matter to produce the network message that communicates hand state, shown in Algorithm 19.6. As in component 19.3.1 an identifier is included to track the device supplying the data. An immediate extension would also be to include information to identify a particular hand since the Leap Motion Controller can track multiple hands

simultaneously. This example currently only allows supports a single hand.

Code Listing 19.6: HandDetails. Communication of hand state is simplified by the existence of a serialization hand object from the Leap Motion SDK containing all the required information.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System;
5  using System.Runtime.Serialization.Formatters.Binary;
6  using System.IO;
7  using Leap.Unity;
8  using Leap;

9
10 [Serializable]
11 public class HandDetails {
12
13     public string id;
14     public Hand hand;
15
16     public HandDetails (HandModelBase handBase)
17     {
18         id = getID ();
19         hand = handBase.GetLeapHand ();
20     }
21
22     public byte [] serialize ()
23     {
24         BinaryFormatter bf = new BinaryFormatter ();
25         MemoryStream ms = new MemoryStream ();
26         bf.Serialize (ms, this);
27         return ms.ToArray ();
28     }
29
30     public static HandDetails deserialize (byte [] b)
31     {
32         BinaryFormatter formatter = new BinaryFormatter();
33         HandDetails cd = (HandDetails) formatter.Deserialize
34             (new MemoryStream (b));
35         return cd;
36     }
37
38     public static string getID ()
39     {
40         return SystemInfo.deviceUniqueIdentifier;
41     }
}
```

5. The network communication process also uses a similar strategy to that employed in component 19.3.1. As

in that case, packet based communication using UDP datagrams is used to avoid network communication overheads. A difference is the addressing mechanism used. In the previous example both devices are likely to be connected to the same subnetwork and able to communicate using broadcast traffic. In this scenario the controller may be attached to a desktop device connected to a wired network, and the mobile platform connected to a wireless network. There is a good chance that routers connecting these networks will filter broadcast traffic.

Options used instead include multicast traffic, where each communicating device subscribes to a multicast address. Many well behaved network routers then ensure that the packets relevant to those addresses are transmitted between the subnetworks involved. However in the worst case scenario, an option is included for the source device to specify the address of an individual destination and the packets will be transmitted directly to that endpoint. This does remove the fun opportunity to have multiple mobile devices all controlled by the same physical hand but is likely to work in most network configurations.

The network communication functions used to achieve this functionality are presented in Algorithm 19.7.

Code Listing 19.7: `DatagramCommunication`. Multicast and unicast datagram communication are supported by this version.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System.Net.Sockets;
5  using System.Net;
6  using System;
7
8  public class DatagramCommunication
9  {
10
11      private int port = 8082;
12
13      private UdpClient udpClient;
14 }
```

```
15  private HandDetails lastHand;
16  private bool controlValid = false;
17
18  private IPEndPoint remoteep;
19
20  public DatagramCommunication()
21  {
22      udpClient = new UdpClient();
23
24      udpClient.Client.Bind(new
25          IPEndPoint(IPAddress.Any, port));
26      IPAddress multicastaddress =
27          IPAddress.Parse("239.0.0.222");
28      udpClient.JoinMulticastGroup(multicastaddress);
29      remoteep = new IPEndPoint(multicastaddress, port);
30
31      udpClient.BeginReceive(new
32          AsyncCallback(udpReceive), null);
33  }
34
35  // Retrieve the most recently received hand details.
36  public HandDetails receiveHandDetails()
37  {
38      if (controlValid)
39      {
40          controlValid = false;
41          return lastHand;
42      }
43      return null;
44  }
45
46  // Wait for a new message to arrive, and set that as
47  // the last message received.
48  private void udpReceive(IAsyncResult res)
49  {
50      IPEndPoint from = new IPEndPoint(0, 0);
51      byte[] buffer = udpClient.Receive(ref from);
52
53      HandDetails details =
54      HandDetails.deserialize(buffer);
55      lastHand = details;
56      controlValid = true;
57
58      udpClient.BeginReceive(new
59          AsyncCallback(udpReceive), null);
56
57  public void sendHandDetails(string hostname,
58  HandDetails details)
59  {
60      if (hostname != null)
61      {
```

```

60           remoteep = new
61             IPEndPoint(IPAddress.Parse(hostname), port);
62           }
63         byte[] data = details.serialize();
64         udpClient.Send(data, data.Length, remoteep);
65       }
66     }
67   }

```

6. This example selects one of the hand models and transmits its state to the mobile device. Create a new empty object named NetworkSource. Attach a new C# script named HandInformationSource, whose code is provided in Algorithm 19.8, to this object. Provide one of the hand models from the LeapHandController as the Hand property for this script. Depending on your network configuration you may also need to find the IP address of your mobile device (usually under the device's setting menu) and provide that as the hostname used to address the packets.

Code Listing 19.8: HandInformationSource. Transmission of hand state is straightforward since this structure is provided directly from the model managed by the Leap Motion software components.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using Leap.Unity;
6
7  public class HandInformationSource : MonoBehaviour {
8
9    [Tooltip ("The hand that is transmitted over the
10   → network")]
11   public HandModelBase hand;
12
13   [Tooltip ("The device that receives the hand updates.
14   → Leave blank for multicast.")]
15   public string hostname;
16
17   private DatagramCommunication dc;
18
19   void Start()
20   {
21     dc = new DatagramCommunication ();

```

```

21    }
22
23    void Update () {
24        dc.sendHandDetails (hostname, new HandDetails (hand));
25    }
26}

```

7. Switch to the GestureRecognition scene. The hand object is reconstructed in the scene which represents the virtual environment in which gesture recognition takes place. Ideally we want to recreate a HandModelBase object from the state received over the network as this can then be provided to the gesture recognition components provided by the Leap Motion software. The newly added network layer then effectively becomes invisible and the Leap Motion software continues to behave as if it were running with a directly connected device.

The application uses the same concept as covered in component 19.3.1. A collection of objects are dropped into the scene and individual objects can be selected and inflated. The difference in this case is that the selection process is achieved by pointing with the index finger and the inflation process is triggered with a gesture (pointing, with index finger extended and other fingers clenched).

Set up the scene with a ground plane, a prefab representing the objects being dropped and an object dropper script (see component 19.3.1). Remember to add a rigidbody with gravity to the dropped object prefab, and to create and set an Inflatable tag on that prefab.

8. Create a new empty object and name this NetworkSink. Attach a C# script named HandInformationSink to this object. This script uses the code in Algorithm 19.9 to receive incoming hand state, to copy that state to a hand model, and then manage the raycasting and inflation operation if the boolean variable beamActive is set.

Code Listing 19.9: HandInformationSink. The hand model is recreated for hand state information received from network packets. The

setLaserActive function allows the selection and inflation processes to be controlled from an external gesture recognition process.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using Leap.Unity;
6  using Leap;
7
8  public class HandInformationSink : MonoBehaviour
9  {
10     [Tooltip ("The hand model to work with.")]
11     public HandModelBase hand;
12     [Tooltip("A cylindrical beam object attached to the
13     controller.")]
14     public GameObject laserBeam;
15     [Tooltip("An adjustment factor for how fast objects
16     blow up.")]
17     public float inflationRate = 1.0f;
18     [Tooltip("A sound effect played during inflation.")]
19     public AudioSource hiss;
20     [Tooltip("A sound effect played when the object is
21     destroyed.")]
22     public AudioSource pop;
23
24     // The laser is switched on when the correct gesture
25     // is performed.
26     public bool beamActive;
27
28     // Link to the network functions.
29     private DatagramCommunication dc;
30
31     public void setLaserActive (bool value)
32     {
33         beamActive = value;
34     }
35
36     void Start()
37     {
38         dc = new DatagramCommunication();
39         hand.BeginHand();
40     }
41
42     void Update()
43     {
44         // decay hiss so it stops if no button is pressed.
45         if (hiss != null)
46         {
47             hiss.volume *= 0.9f;
48         }
49
50         HandDetails cd = dc.receiveHandDetails();
```

```

47     if (cd != null)
48     {
49         hand.SetLeapHand(cd.hand);
50
51         hand.UpdateHand();
52
53         laserBeam.SetActive(beamActive);
54         if (beamActive)
55         {
56             Finger f = hand.GetLeapHand().Fingers[(int]
57             ↵ )Finger.FingerType.TYPE_INDEX]; // index
58             ↵ finger.
59                 Bone b = f.Bone(Bone.BoneType.TYPE_DISTAL);
60                 Vector3 bCenter = new Vector3(b.Center.x,
61                 ↵ b.Center.y, b.Center.z);
62                 Vector3 bDirection = new
63                 ↵ Vector3(b.Direction.x, b.Direction.y, b.Direction.z);
64                 laserBeam.transform.position = bCenter;
65                 laserBeam.transform.forward = bDirection;
66
67                 // Raycast, inflate, explode.
68                 RaycastHit hit;
69                 if ((Physics.Raycast(bCenter, bDirection,
70                 ↵ out hit, Mathf.Infinity)) &&
71                 ↵ (hit.collider.gameObject.tag ==
72                 "Inflatable"))
73                 {
74                     // Inflate the object by manipulating
75                     ↵ scale.
76
77                     hit.collider.gameObject.transform.localScale *= 1.0f +
78                     (inflationRate * Time.deltaTime);
79                     // Play the inflation sound.
80                     if (hiss != null) { hiss.volume =
81                     ↵ 1.0f; if (!hiss.isPlaying) hiss.Play(); }
82                     // Pop the object if it gets too big.
83                     if (hit.collider.gameObject.transform.]
84                     ↵ localScale.magnitude >
85                     ↵ 1.5)
86                     {
87                         Destroy(hit.collider.gameObject);
88                         if (pop != null) pop.Play();
89                     }
90                 }
91             }
92         }
93     }

```

9. The network sink needs a number of properties. One is the hand object, whose state is provided by the net-

work updates. Add in a hand prefab using one of those provided by the Leap Motion software package. Remove the Hand Enable/Disable component to ensure that it is visible, even when no hand state data is available. Drag the hand model to the hand property of the network sink.

Create a laser beam object using a long thin cylinder aligned with the z-axis and whose origin is at one end (using an empty object as the parent helps with this). Remove the collider on the laser beam.

Sound effects can also be included by adding Audio Source components.

10. Gesture recognition can be achieved by responding to particular hand orientations or finger configurations.

The Leap Motion software provides a number of components that can be used in different contexts. In this case we recognize a pointing gesture as this is consistent with the goal of selecting objects, and because the gesture can also be used to trigger an action.

From the Scripts/DetectionUtilities in the Leap Motion assets, add an Extended Finger Detector to the Network Sink object. Set it up as a point gesture (all fingers not extended except the index finger which is extended). Create connections from the On Activate and On Deactivate events to the setActive function of the HandInformationSink component. Tick the checkbox on the On Activate option, to send the true value when this event is passed, and leave the checkbox empty on the On Deactivate.

11. Build and deploy the gesture recognition scene to the mobile device. Run the hand tracker scene on the device attached to the leap motion controller. These should not both be on the same machine as the network communication currently expects to send and receive on the same port (this can be changed if required). Also make sure that the hand tracker's network source has the address of the device with the

network sink, unless multicast communication works on your network.

The hand model in the gesture recognition scene should mirror that of the hand model tracked in the hand tracker scene. Make the point gesture should make the laser beam visible and allow selected objects to be inflated until they pop.

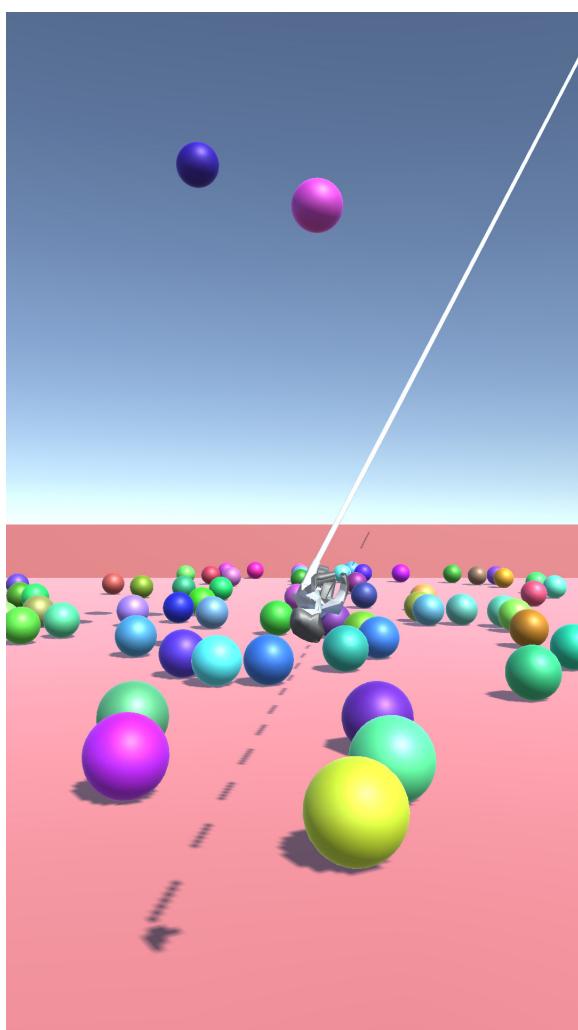
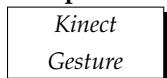


Figure 19.4.3:  
Hand tracking and  
gesture recognition.

## 19.5 Interaction using gesture recognition with the Kinect

### Component 19.5.1: Gesture recognition with the Kinect.



*The Kinect motion controller is a device built around a depth camera; a sensor able to capture images where both colour and depth are associated with each pixel.*

*Originally developed as a gaming peripheral it was discontinued in this form around 2017 although there are suggestions that it may be restored for different purposes in the future. Software updates have stopped and so this example is unlikely to be directly relevant in its current form in the future.*

Figure 19.5.1: Pose capture and gesture recognition with the Kinect.



1. This example requires the Kinect for Windows 2.0 SDK. This is available from: <http://www.microsoft.com/en-us/download/details.aspx?id=44561>. This software needs to be installed on each machine used. The installation includes Kinect Studio, which provides a way to both verify that the Kinect camera is working, and to see some of the data streams that the device provides.
2. Start a new project. Download the Unity software package for the Kinect from: [Downloadtheunityplugin: http://go.microsoft.com/fwlink/?LinkID=513177](http://go.microsoft.com/fwlink/?LinkID=513177). Unpack this archive in a convenient location.

Import the Kinect.2.0.1410.19000.unitypackage (Assets/Import package menu option) from the unpacked archive. This should add two folders to the project assets: Plugins and Standard Assets.

Also manually copy two files from the KinectView/Scripts folder in the unpacked archive into the Assets area of the project. The files required are BodySourceManager.cs and BodySourceView.cs. These might be upgraded when the Unity software imports them.

3. Create a new empty object in the scene and name it SkeletonManager. Attach the two script components: BodySourceManager and BodySourceView. The SkeletonManager object needs to be provided as the Body Source Manager property to the Body Source View component.

Running the application at this stage should show a representation of the skeletons tracked by the Kinect device.

4. Some gesture recognition functionality is already present in the skeleton tracking and reconstruction. The state of the two hands is reported directly, and can either be open, closed, or performing the lasso gesture (thumb and two smallest fingers closed, the other two fingers extended).

Create a simple object, such as a cube, that can be used to test hand gestures. Create a C# script called HandGesture and apply it to the object. The script provided in Algorithm 19.10 retrieves the various skeletons (bodies) from the body manager. If the left hand is performing the lasso gesture then the colour of the object is set to green. Remember to provide this component with a BodySourceManager by dragging the SkeletonManager object onto the Body Manager property.

Code Listing 19.10: HandGestures. Hand gestures are directly accessible as part of the body state returned by the Kinect software.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Windows.Kinect;
5
6  public class HandGestures : MonoBehaviour {
7
8      public BodySourceManager bodyManager;
9
10     void Update () {
11         GetComponent <MeshRenderer>
12         → (.material.color = new Color (0, 0, 1);
13         if (bodyManager == null)
14         {
15             return;
16         }
17
18         Body[] data = bodyManager.GetData ();
19         if (data == null)
20         {
21             return;
22         }
23
24         foreach (Body body in data)
25         {
26             if (body == null)
27             {
28                 continue;
29             }
30
31             if (body.HandLeftState ==
32             → Windows.Kinect.HandState.Lasso) {
33                 GetComponent
34                 → <MeshRenderer> (.material.color = new Color (0, 1, 0));
35             }
36         }
37     }
38 }
```

- More complex gesture recognition is achieved by using program logic to test for particular configurations. For example, a hand up gesture can be recognized by checking if the left hand is above the left shoulder (greater y coordinate, assuming y is the vertical axis), and that the right hand is below the right shoulder. A similar condition is used to check if the right hand is raised.

A gesture recognizer that tests for either hand raised

is based on this principle. Again, construct another object, such as a sphere and create a new RaiseHand C# script that can be applied to it. The script, shown in Algorithm 19.11, implements these tests. This script also requires access to the BodySourceManager available on the SkeletonManager object.

**Code Listing 19.11:** RaiseHand. Identifying a raised hand involves comparing the relative position of particular joints.

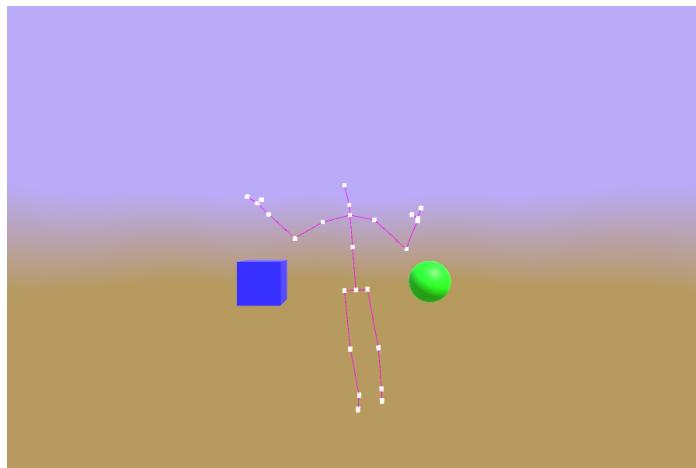
```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Windows.Kinect;
5
6  public class RaiseHand : MonoBehaviour {
7
8      public BodySourceManager bodyManager;
9
10     void Update () {
11         GetComponent <MeshRenderer>
12         .material.color = new Color (0, 0, 1);
13         if (bodyManager == null)
14         {
15             return;
16         }
17
18         Body[] data = bodyManager.GetData ();
19         if (data == null)
20         {
21             return;
22         }
23
24         foreach (Body body in data)
25         {
26             if (body == null)
27             {
28                 continue;
29             }
30
31             Windows.Kinect.Joint LHandJoint =
32             body.Joints [JointType.HandLeft];
33             Windows.Kinect.Joint RHandJoint =
34             body.Joints [JointType.HandRight];
35             Windows.Kinect.Joint
36             LShoulderJoint = body.Joints [JointType.ShoulderLeft];
37             Windows.Kinect.Joint
38             RShoulderJoint = body.Joints [JointType.ShoulderRight];
39             if (((LHandJoint.Position.Y >
40             LShoulderJoint.Position.Y) &&
```

```

35             (RHandJoint.Position.Y <
36             RShoulderJoint.Position.Y)) ||
37             ((LHandJoint.Position.Y <
38             LShoulderJoint.Position.Y) &&
39             (RHandJoint.Position.Y >
40             RShoulderJoint.Position.Y)))
41         {
42             GetComponent
43             <MeshRenderer> ().material.color = new Color (0, 1, 0);
44         }
45     }
46 }

```

Figure 19.5.2:  
Pose information  
used for gesture  
recognition.



## 19.6 Managing interaction using body tracking on a general purpose device

### Component 19.6.1: Body Tracking.

**Body  
Tracking**

The demise of the Kinect as a general purpose consumer product opens up opportunities for other forms of skeletal tracking that can be achieved with widely accessible devices. The depth camera functionality is an advantage when tracking motion but is not completely essential to achieving useful outcomes. This example demonstrates the use of the PoseNet model from the tensorflow package which achieves credible results even

*when running on a generic mobile device.*

*The approach described in this example involves compiling the tensorflow package as a native library on Android devices, so that the facilities can be accessed through the Unity software. There are several other developments in supporting tensorflow packages on the Unity software at the moment which are still in development or experimental at this stage. These may become usable in future but at time of writing the direct use of tensorflow as described here had the minimal set of dependencies.*

*The process of building the native library requires a Linux platform. This could potentially be achieved by a similar cross-compilation process in other environments. The resulting binary file is specific to Android though, and can be used for projects being developed across other platforms.*

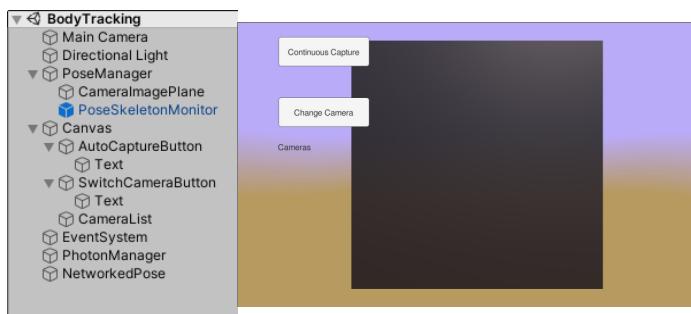


Figure 19.6.1: Body tracking using a standard camera.

1. Start with a new project. Set this up as an Android project, by making the changes to the settings as described in component 16.1.1. An extra change is to enable unsafe code, which is a checkbox under the Other Settings as part of the Player Settings. This simplifies some of the access to memory resources that can be shared between the native C++ layer and the Unity C# code.
2. We then need to download and build the tensorflow library. This involves retrieving the Tensorflow Lite repository from within the main tensorflow git repository at <https://github.com/tensorflow/tensorflow.git/trunk/tensorflow>.

This step assumes that you have access to a copy of the project files provided with this book. A shell script is provided to do this, and can be invoked from the command line as:

```
1 sh ./buildTFLite.sh
```

---

The script assumes that the Android NDK (Native Development Kit) is installed at an appropriate location. Edit the script and modify the value assigned to the variable ANDROID\_NDK to match the location on your system.

This does modify the downloaded source code to remove some unnecessary dependencies, before building both the android and linux native versions of the Tensorflow Lite library. These are then copied to folders within the Assets/Plugins folder in the Unity software project. If you do not have access to a system that can compile these from source then you can use pre-compiled copies of these and still follow the remaining steps of this project.

If you do modify this library, please note that the Unity software needs to be restarted in order for any changes to take effect.

3. The next step is to invoke the tensorflow library from within the Unity project. The overall steps of using the PoseNet model (and potentially any other model) are:
  - (a) When the project starts, load the model from the file that it is shipped in. In this case, the model is available as posenet\_mobilenet\_v1\_100\_257x257\_-multi\_kpt\_stripped.tflite which can be retrieved from: [https://www.tensorflow.org/lite/models/pose\\_estimation/overview](https://www.tensorflow.org/lite/models/pose_estimation/overview). However better versions of this model may become available in future.
  - (b) Once a new camera image becomes available it needs to be provided as input to the tensorflow library and the model invoked. Typically models

expect the input data to be formatted in the same way as the images upon which they were trained. In this case the size of the image needs to be standardized (to  $257 \times 257$ ) and the colour intensity values normalized relative to an average value.

The output from the model is also not likely to be in a directly consumable format. In this case, the 17 possible joints that are tracked are reported with respect to their being within each cell of a  $9 \times 9$  grid superimposed on the image. A second output array then gives a finer position estimate relative to the position of the most likely cell.

It is convenient to pre-process this output, so that it can be converted to an array describing the x, y and confidence values for each of the 17 joints.

Ensuring this process works on a mobile device requires a few extra steps.

- (a) Directly sharing the camera texture as the source of the image may not work on all mobile devices. An alternative process in this case is to manually copy the camera image into a separate texture image, and then to pass this texture image to the tensorflow library.
- (b) The actual PoseNet model file is embedded in the package file installed on the device. This hinders access to this file using the conventional file access calls used to read the model. The solution in this case is a second step which extracts the PoseNet model file from the Android package and copies it to a region of the file system on the device. It can then be treated as a normal file at this point.

The software to achieve all the steps described exists at both a native code level, and as a C# script. The work is divided between these two stages in a way that minimizes the amount of information that needs to be passed across the C#/C++ boundary.

The C++ script shown in Algorithm 19.12 has already been compiled during the previous step when the native library was built. This exposes only two functions to the C# layer: the initPose function which loads the model, and the computePoseData which receives an image, and returns a list of joint positions.

Code Listing 19.12: poseinterface. Native code library to invoke the PoseNet model on an image.

```

1  #include <stdio.h>
2
3  #include <tensorflow/lite/model.h>
4  #include <tensorflow/lite/core/api/error_reporter.h>
5  #include <tensorflow/lite/kernels/register.h>
6
7  #ifdef ANDROID
8  #include <GLES3/gl31.h>
9  #else
10 #include <GL/gl.h>
11 #endif
12
13 extern "C"
14 {
15     void initPose (char * modelfile);
16     int computePose (int texture, int w, int h, float *
17     ↳ results);
18     int computePoseData (char * imageData, int w, int h,
19     ↳ float * results);
20 }
21
22 // For debugging, read the input image from a ppm file.
23 // Generic function
24 // from external sources.
25 unsigned char * readPPMfile (const char* filename, int
26     ↳ *wp, int *hp)
27 {
28     FILE* input;
29     int w, h, max;
30     int i, j, k;
31     char rgb [3];
32     unsigned char* pixels;
33     char buffer[200];
34
35     if ((input = fopen (filename, "r")) == NULL)
36     {
37         fprintf (stderr, "Cannot open file %s \n", filename);
38         exit (1);
39     }
40
41     /* read a line of input */

```

```

38     fgets (buffer, 200, input);
39     if (strncmp (buffer, "P6", 2) != 0)
40     {
41         fprintf (stderr, "%s is not a binary PPM file \n",
42             filename);
43         exit (1);
44     }
45
46     /* get second line, ignoring comments */
47     do
48     {
49         fgets (buffer, 200, input);
50     }
51     while (strncmp (buffer, "#", 1) == 0);
52
53     if (sscanf (buffer, "%d %d", &w, &h) != 2)
54     {
55         fprintf (stderr, "can't read sizes! \n");
56         exit (1);
57     }
58
59     /* third line, ignoring comments */
60     do
61     {
62         fgets (buffer, 200, input);
63     }
64     while (strncmp (buffer, "#", 1) == 0);
65
66     if (sscanf (buffer, "%d", &max) != 1)
67     {
68         fprintf (stderr, "what about max size? \n");
69         exit (1);
70     }
71
72     pixels = (unsigned char*) malloc (w * h * 3 *
73         sizeof(unsigned char));
74     for (i = 0; i < h; i++)
75     {
76         for (j = 0; j < w; j++)
77         {
78             fread (rgb, sizeof(char), 3, input);
79             for (k = 0; k < 3; k++)
80             {
81                 *(pixels + (i) * w * 3 + j * 3 + k) = (unsigned
82                     char) rgb[k];
83             }
84         }
85     }
86
87     *wp = w;
88     *hp = h;
89     return pixels;
90 }
```

```

88
89 // Write the image file, useful for showing output.
90 // Generic function
91 // from external sources.
92 void writePPM (unsigned char * data, int w, int h)
93 {
94     int i, j;
95     FILE *fp = fopen ("posedetection.ppm", "wb");
96     (void) fprintf (fp, "P6\n%d %d\n255\n", w, h);
97     for (j = 0; j < h; ++j)
98     {
99         for (i = 0; i < w; ++i)
100        {
101            static unsigned char color[3];
102            color[0] = data[3 * ((j * w) + i) + 0];
103            color[1] = data[3 * ((j * w) + i) + 1];
104            color[2] = data[3 * ((j * w) + i) + 2];
105            (void) fwrite(color, 1, 3, fp);
106        }
107        (void) fclose(fp);
108    }

109
110 // Make a mark on the image data. Useful for plotting
111 // output.
112 void plot (unsigned char * data, int w, int h, int x, int
113 // y)
114 {
115     int size = 5;
116     for (int i = -size; i < size; i++)
117     {
118         for (int j = -size; j < size; j++)
119         {
120             if ((x + i >= 0) && (x + i < w) && (y + j >= 0) &&
121             (y + j < h))
122             {
123                 data[3 * (((y + j) * w) + (x + i)) + 0] = 255;
124                 data[3 * (((y + j) * w) + (x + i)) + 1] = 0;
125                 data[3 * (((y + j) * w) + (x + i)) + 2] = 0;
126             }
127         }
128     }
129 }
130
131 std::unique_ptr<tflite::FlatBufferModel> model;
132 tflite::ops::builtin::BuiltinOpResolver resolver;
133 std::unique_ptr<tflite::Interpreter> interpreter;
134
135 // Load the reusable elements of the process, such as
136 // model.
137 void initPose (char * modelfile)
138 {

```

```

135     model = tflite::FlatBufferModel::BuildFromFile
136     ↪ (modelfile);
137     tflite::InterpreterBuilder(*model,
138     ↪ resolver)(&interpreter);
139
140 #ifdef TESTPOSEINTERFACE
141     printf ("Inputs %d Outputs %d\n", interpreter->inputs
142     ↪ ().size (), interpreter->outputs ().size ());
143 #endif
144
145     interpreter->AllocateTensors();
146 }
147
148 // Analyse the image provided and return the pose data.
149 void callPose (unsigned char * data, int width, int
150   ↪ height, float * results)
151 {
152     // Find the dimensions of the input image required by
153     // this model.
154     TfLiteIntArray* dims =
155     ↪ interpreter->tensor(interpreter->inputs()[0])->dims;
156     int wanted_height = dims->data[1];
157     int wanted_width = dims->data[2];
158     int wanted_channels = dims->data[3];
159
160 #ifdef TESTPOSEINTERFACE
161     printf ("Wanted %d: %d %d %d\n", dims->size,
162     ↪ dims->data[0],wanted_height, wanted_width,
163     ↪ wanted_channels);
164 #endif
165
166     // Find the dimensions of the output heatmap as well.
167     TfLiteIntArray* output_dims =
168     ↪ interpreter->tensor(interpreter->outputs()[0])->dims;
169
170 #ifdef TESTPOSEINTERFACE
171     printf ("Output dims %d - %d %d %d\n",
172     ↪ output_dims->size, output_dims->data[0],
173     ↪ output_dims->data[1], output_dims->data[2],
174     ↪ output_dims->data[3]);
175 #endif
176
177     int outheight = output_dims->data[1];
178     int outwidth = output_dims->data[2];
179     int numKeypoints = output_dims->data[3];
180
181
182     // Copy the image into the input buffer, rescaling at
183     // the same time.
184     float * inputBuffer =
185     ↪ interpreter->typed_input_tensor<float>();
186     float shrinkx = width / ((float) wanted_width);
187     float shirky = height / ((float) wanted_height);
188
189     float mean = 128.0;
190     float std = 128.0;
191     unsigned char pixelValue;

```

```

174     for (int row = 0; row < wanted_height; row++)
175     {
176         for (int col = 0; col < wanted_width; col++)
177         {
178             pixelValue = data[3 * (((int) ((wanted_height - 1 -
179             row) * shinky)) * width + ((int) (col * shrinkx))) +
180             0];
181             inputBuffer[3 * (row * wanted_width + col) + 0] =
182             (((float) (pixelValue - mean)) / std);
183             pixelValue = data[3 * (((int) ((wanted_height - 1 -
184             row) * shinky)) * width + ((int) (col * shrinkx))) +
185             1];
186             inputBuffer[3 * (row * wanted_width + col) + 1] =
187             (((float) (pixelValue - mean)) / std);
188             pixelValue = data[3 * (((int) ((wanted_height - 1 -
189             row) * shinky)) * width + ((int) (col * shrinkx))) +
190             2];
191             inputBuffer[3 * (row * wanted_width + col) + 2] =
192             (((float) (pixelValue - mean)) / std);
193         }
194     }
195
196
197     // Do the recognition.
198     interpreter->Invoke();
199
200
201     // Extract meaning from the output. Combine the most
202     // reliable heatmap entry with the offset to determine
203     // exact position of each point.
204     float * heatmaps =
205     interpreter->typed_output_tensor<float>(0);
206     float * offsets =
207     interpreter->typed_output_tensor<float>(1);
208
209     TfLiteTensor * heatmaps_tensor = interpreter->tensor (0);
210
211     for (int keypoint = 0; keypoint < numKeypoints;
212         keypoint++)
213     {
214         float maxVal = heatmaps[numKeypoints * (outwidth * (0)
215             + 0) + keypoint];
216         int maxRow = 0;
217         int maxCol = 0;
218         for (int row = 0; row < outheight; row++)
219         {
220             for (int col = 0; col < outwidth; col++)
221             {
222                 float h = heatmaps[numKeypoints * (outwidth *
223                     (row) + col) + keypoint];
224                 if (h > maxVal)
225                 {
226                     maxVal = h;
227                     maxRow = row;
228                     maxCol = col;

```

```

212         }
213     }
214 }
215
216
217     float positionX = (((float) maxCol) / (outwidth-1)) *
218     ↪ wanted_width + offsets[2 * numKeypoints * (outwidth *
219     ↪ (maxRow) + maxCol) + keypoint + numKeypoints]) *
220     ↪ shrinkx;
221     float positionY = (wanted_height - (((((float) maxRow)
222     ↪ / (outheight-1)) * wanted_height + offsets[2 *
223     ↪ numKeypoints * (outwidth * (maxRow) + maxCol) +
224     ↪ keypoint))) * shrinky;
225 #ifdef TESTPOSEINTERFACE
226     printf ("Out %d %d %d %f -- %f %f\n", keypoint,
227     ↪ maxCol, maxRow, maxVal, positionX, positionY);
228     plot (data, width, height, positionX, positionY);
229 #endif
230
231     results[keypoint * 3 + 0] = positionX / width;
232     results[keypoint * 3 + 1] = positionY / height;
233     results[keypoint * 3 + 2] = maxVal;
234 }
235
236 #ifdef TESTPOSEINTERFACE
237     writePPM (data, width, height);
238 #endif
239 }
240
241 // Interface between Unity and native code.
242 int computePose (int texture, int w, int h, float *
243     ↪ results)
244 {
245     glEnable (GL_TEXTURE_2D);
246     glBindTexture (GL_TEXTURE_2D, texture);
247     unsigned char * dd = new unsigned char [w * h * 3];
248 #ifdef ANDROID
249     // Thanks: https://stackoverflow.com/questions/53993820/
250     ↪ opengl-es-2-0-android-c-glgetteximage-alternative
251     GLuint fbo;
252     glGenFramebuffers (1, &fbo);
253     glBindFramebuffer (GL_FRAMEBUFFER, fbo);
254     glFramebufferTexture2D (GL_FRAMEBUFFER,
255     ↪ GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, texture, 0);
256
257     glReadPixels(0, 0, w, h, GL_RGB, GL_UNSIGNED_BYTE, dd);
258
259     glBindFramebuffer(GL_FRAMEBUFFER, 0);
260     glDeleteFramebuffers(1, &fbo);
261
262 #else
263     glGetTexImage (GL_TEXTURE_2D, 0, GL_RGB,
264     ↪ GL_UNSIGNED_BYTE, dd);

```

```

254     #endif
255
256     glBindTexture (GL_TEXTURE_2D, 0);
257     glDisable (GL_TEXTURE_2D);
258
259     callPose (dd, w, h, results);
260     delete [] dd;
261     int r = 99;
262     r = glGetError ();
263     return r;
264 }
265
266 // Variation on compute pose due to issues with android
267 // accessing camera data.
267 int computePoseData (char * imageData, int w, int h, float
268 // * results)
269 {
270     callPose ((unsigned char *) imageData, w, h, results);
271     int r = 222;
272     return r;
273 }
274
275 #ifdef TESTPOSEINTERFACE
276 int main (int argc, char * argv [])
277 {
278     int width;
279     int height;
280     unsigned char * data = readPPMfile ("pose.ppm", &width,
281     // &height);
282     printf ("Read %d %d\n", width, height);
283     float results [17 * 3];
284     initPose ("Assets/StreamingAssets/posenet_mobilenet_v1_1_
285     // 00_257x257_multi_kpt_stripped.tflite");
286     callPose (data, width, height, results);
287     for (int i = 0; i < 17; i++)
288     {
289         printf ("Result %d: (%f, %f)\n", i, results[i * 2 +
290         0], results[i * 2 + 1]);
291     }
292 }
293
294 #endif

```

The C# script shown in Algorithm 19.13 takes care of the Unity related functions of unpacking the PoseNet model file, activating and retrieving images from the physical camera, and invoking the pose detection process whenever the mouse is clicked. This also includes a call to a separate PoseVisualizer component to display the skeleton (described in a later step).

Code Listing 19.13: FetchPose. Convert camera image into joint positions.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Unity.Collections;
5  using Unity.Collections.LowLevel.Unsafe;
6  using System;
7  using System.Runtime.InteropServices;
8  using System.Threading;
9  using System.IO;
10 using UnityEngine.UI;
11
12 // Retrieve pose information from the native
13 // ↪ posenet/tensorflow lite facilities.
14 // Also includes some visual elements, to show camera feed
15 // ↪ and monitor retrieved
16 // values.
17 public class FetchPose : MonoBehaviour
18 {
19     [DllImport ("poseinterface")]
20     unsafe private static extern void initPose (string
21         ↪ modelfile);
22     [DllImport ("poseinterface")]
23     unsafe private static extern int computePose (IntPtr
24         ↪ texture, int w, int h, float * results);
25     [DllImport ("poseinterface")]
26     unsafe private static extern int computePoseData (byte
27         ↪ [] imageData, int w, int h, float * results);
28
29     public Material camTexMaterial;
30
31     private WebCamTexture webcamTexture;
32
33     private int numPointsInPose = 17;
34
35     public PoseSkeleton poseVisualizer;
36     public NetworkedPose poseTransmitter;
37
38     private bool dataReady;
39
40     private int currentCamera = 0;
41     public Text outputText;
42
43     private bool overrideCapture = false;
44
45     public void toggleCapture ()
46     {
47         overrideCapture = !overrideCapture;
48     }
49
50     private void showCameras ()
```

```

46
47     {
48         outputText.text = "";
49         foreach (WebCamDevice d in WebCamTexture.devices)
50         {
51             outputText.text += d.name + (d.name ==
52             → webcamTexture.deviceName ? "*" : "") + "\n";
53         }
54     }
55
56     public void nextCamera ()
57     {
58         currentCamera = (currentCamera + 1) %
59         → WebCamTexture.devices.Length;
60         // Change camera only works if the camera is stopped.
61         webcamTexture.Stop ();
62         webcamTexture.deviceName =
63         → WebCamTexture.devices[currentCamera].name;
64         webcamTexture.Play ();
65         showCameras ();
66     }
67
68     // Start is called before the first frame update
69     void Start()
70     {
71         webcamTexture = new WebCamTexture ();
72         showCameras ();
73
74         camTexMaterial.mainTexture = webcamTexture;
75         webcamTexture.Play ();
76
77         dataReady = false;
78         StartCoroutine (prepareModel ());
79     }
80
81     IEnumerator prepareModel ()
82     {
83         string modelfile = "posenet_mobilenet_v1_100_257x257_m_"
84         → ulti_kpt_stripped.tflite";
85         yield return StartCoroutine(extractFile ("",
86         → modelfile));
87
88         initPose (Application.persistentDataPath + "/" +
89         → modelfile);
90
91         dataReady = true;
92     }
93
94     // Retrieve the pose from the native library. This
95     → returns
96     // as an array of floats, containing x, y, and confidence
97     // values for each point.
98     unsafe float [] retrievePose ()
99

```

```

92     {
93         NativeArray <float> pose = new NativeArray <float>
94             (numPointsInPose * 3, Allocator.Temp);
95         int result = computePose
96             (webcamTexture.GetNativeTexturePtr (),
97             webcamTexture.width, webcamTexture.height, (float *)
98             NativeArrayUnsafeUtility.GetUnsafePtr (pose));
99         Debug.Log ("Got result " + result + " " + pose[0] +
100            " " + pose[1] + " " + pose[2]);
101        return pose.ToArray ();
102    }
103
104
105    // Version that retrieves the pose that passes image data
106    // directly to the native library, rather than relying on
107    // accessing the texture directly.
108    unsafe float [] retrievePoseData ()
109    {
110        Texture2D image = new Texture2D
111            (camTexMaterial.mainTexture.width,
112             camTexMaterial.mainTexture.height,
113             TextureFormat.RGB24, false);
114        RenderTexture renderTexture = new
115            RenderTexture(camTexMaterial.mainTexture.width,
116            camTexMaterial.mainTexture.height, 24);
117        Graphics.Blit(camTexMaterial.mainTexture,
118            renderTexture);
119        RenderTexture.active = renderTexture;
120        image.ReadPixels (new Rect(0, 0, renderTexture.width,
121            renderTexture.height), 0, 0);
122        image.Apply();
123
124        NativeArray <float> pose = new NativeArray <float>
125            (numPointsInPose * 3, Allocator.Temp);
126        int result = computePoseData (image.GetRawTextureData
127            (), image.width, image.height, (float *)
128            NativeArrayUnsafeUtility.GetUnsafePtr (pose));
129        //    Debug.Log ("Got result " + result + " " + pose[0] +
130            " " + pose[1] + " " + pose[2]);
131
132        Destroy (renderTexture);
133        Destroy (image);
134        return pose.ToArray ();
135    }
136
137    // Update is called once per frame
138    void Update()
139    {
140        if (dataReady && (overrideCapture || (Input.GetAxis
141            ("Fire1") > 0)))
142        {
143            float startTime = Time.realtimeSinceStartup;
144            float [] pose = retrievePose ();
145            float [] pose = retrievePoseData ();
146        }
147    }

```

```

128         float endTime = Time.realtimeSinceStartup;
129         Debug.Log ("Pose tracked in " + (endTime -
130             startTime).ToString ("F6") + " seconds");
131
132         poseVisualizer.updatePose (pose);
133         poseTransmitter.updatePose (pose);
134     }
135
136     // Copy files into an area where they are accessible.
137     // This is particularly
138     // relevant to packages created for mobile platforms.
139     IEnumerator extractFile (string assetPath, string
140         assetFile)
141     {
142         // Source is the streaming assets path.
143         string sourcePath = Application.streamingAssetsPath +
144             "/" + assetPath + assetFile;
145         if ((sourcePath.Length > 0) && (sourcePath[0] == '/'))
146         {
147             sourcePath = "file://" + sourcePath;
148         }
149         string destinationPath =
150             Application.persistentDataPath + "/" + assetFile;
151
152         // Files must be handled via a WWW to extract.
153         WWW w = new WWW (sourcePath);
154         yield return w;
155         try
156         {
157             File.WriteAllBytes (destinationPath, w.bytes);
158         }
159         catch (Exception e)
160         {
161             Debug.Log ("Issue writing " + destinationPath + " "
162                 + e);
163         }
164         Debug.Log (sourcePath + " -> " + destinationPath + " "
165                 + w.bytes.Length);
166     }
167 }
```

4. Create an empty object in the scene called PoseManager.

Attach a plane to this as a child object, leaving position, orientation and scale at their default settings. Then rotate the PoseManager by 90° about the x axis, and 180° about the y axis so that it faces the camera and it upright when the camera texture is added.

Create a new material called CameraMaterial, and set this as the material for the plane.

Add the FetchPose script (Algorithm 19.13) to the PoseManager. This needs the CameraMaterial provided as a parameter, so that it can display the camera image on this. For this step, comment out the lines in the FetchPose script that refer to the PoseSkeleton as this will be added in a later step.

Running the application at this point should the feed from the camera. Clicking the mouse may produce some console messages indicating that a human skeleton has been spotted. This is likely to work better if the camera can see a human.

5. The visual representation of the skeleton can be separated from the process of tracking it. Create another empty object as the child of the PoseManager, and call this PoseSkeletonManager. Add the script from Algorithm 19.14 as a component of this object. This component does need some prefabs. A standard cylinder can be used for a bone. A small object, such as a scaled capsule or sphere can be used mark the joint positions.

Code Listing 19.14: PoseSkeleton. Visual representation of a pose.

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  public class PoseSkeleton : MonoBehaviour
{
7      enum BodyPart : int
8      {
9          Nose = 0,
10         Left_eye = 1,
11         Right_eye = 2,
12         Left_ear = 3,
13         Right_ear = 4,
14         Left_shoulder = 5,
15         Right_shoulder = 6,
16         Left_elbow = 7,
17         Right_elbow = 8,
18         Left_wrist = 9,
19         Right_wrist = 10,
```

```
21         Left_hip = 11,
22         Right_hip = 12,
23         Left_knee = 13,
24         Right_knee = 14,
25         Left_ankle = 15,
26         Right_ankle = 16
27     };
28
29     [Tooltip ("A marker prefab to represent pose points in
29      ← the skeleton")]
30     public GameObject pointMarkerTemplate;
31
32     [Tooltip ("A prefab to represent a bone. Size and
32      ← shape matching default cylinder")]
33     public GameObject boneTemplate;
34
35     private GameObject [] markers;
36
37     private int numPointsInPose;
38
39     class SkeletonBone
40     {
41         public GameObject boneObject;
42         public BodyPart from;
43         public BodyPart to;
44
45         public SkeletonBone (BodyPart f, BodyPart t)
46         {
47             boneObject = null;
48             from = f;
49             to = t;
50         }
51     };
52
53     private List <SkeletonBone> bones;
54
55     public PoseSkeleton ()
56     {
57         numPointsInPose = Enum.GetNames (typeof
57          ← (BodyPart)).Length;
58         markers = new GameObject [numPointsInPose];
59
60         bones = new List <SkeletonBone> ();
61
62         // head
63         bones.Add (new SkeletonBone (BodyPart.Left_eye,
63          ← BodyPart.Right_eye));
64         bones.Add (new SkeletonBone (BodyPart.Left_eye,
64          ← BodyPart.Left_shoulder));
65         bones.Add (new SkeletonBone (BodyPart.Right_eye,
65          ← BodyPart.Right_shoulder));
66         bones.Add (new SkeletonBone (BodyPart.Left_eye,
66          ← BodyPart.Left_ear));
```

```

67     bones.Add (new SkeletonBone (BodyPart.Right_eye,
68     BodyPart.Right_ear));
69     bones.Add (new SkeletonBone (BodyPart.Left_eye,
70     BodyPart.Nose));
71     bones.Add (new SkeletonBone (BodyPart.Right_eye,
72     BodyPart.Nose));

73         // body
74     bones.Add (new SkeletonBone (BodyPart.Left_hip,
75     BodyPart.Left_shoulder));
76     bones.Add (new SkeletonBone (BodyPart.Right_hip,
77     BodyPart.Right_shoulder));
78     bones.Add (new SkeletonBone (BodyPart.Left_hip,
79     BodyPart.Right_hip));
80     bones.Add (new SkeletonBone (BodyPart.Left_shoulder,
81     BodyPart.Right_shoulder));

82         // arms
83     bones.Add (new SkeletonBone (BodyPart.Left_shoulder,
84     BodyPart.Left_elbow));
85     bones.Add (new SkeletonBone (BodyPart.Left_elbow,
86     BodyPart.Left_wrist));
87     bones.Add (new SkeletonBone
88     (BodyPart.Right_shoulder, BodyPart.Right_elbow));
89     bones.Add (new SkeletonBone (BodyPart.Right_elbow,
90     BodyPart.Right_wrist));

91         // legs
92     bones.Add (new SkeletonBone (BodyPart.Left_hip,
93     BodyPart.Left_knee));
94     bones.Add (new SkeletonBone (BodyPart.Left_knee,
95     BodyPart.Left_ankle));
96     bones.Add (new SkeletonBone (BodyPart.Right_hip,
97     BodyPart.Right_knee));
98     bones.Add (new SkeletonBone (BodyPart.Right_knee,
99     BodyPart.Right_ankle));
100 }

101 private Vector3 poseToVector (float [] rawPoseData,
102 int i)
103 {
104     return new Vector3 (- (10.0f * rawPoseData[i * 3 +
105     0] - 5.0f), 0.0f, - (10.0f * rawPoseData[i * 3 + 1] -
106     5.0f));
107 }

108 private void drawPosePoints (float [] rawPoseData)
109 {
110     for (int i = 0; i < numPointsInPose; i++)
111     {
112         if (markers[i] == null)
113         {
114             markers[i] = Instantiate (pointMarkerTemplate);

```

```

102         markers[i].transform.SetParent (this.transform,
103         false);
104     }
105
106     markers[i].transform.localPosition = poseToVector
107     (rawPoseData, i);
108     if (rawPoseData[i * 3 + 2] < 0.0f)
109     {
110         markers[i].SetActive (false);
111     }
112     else
113     {
114         markers[i].SetActive (true);
115     }
116 }
117
118 private void drawSkeleton (float [] rawPoseData)
119 {
120     foreach (SkeletonBone b in bones)
121     {
122         Vector3 from = poseToVector (rawPoseData, (int)
123         b.from);
124         Vector3 to = poseToVector (rawPoseData, (int)
125         b.to);
126         float len = 0.5f * (to - from).magnitude;
127
128         if (len > 0.01f)
129         {
130             if (b.boneObject == null)
131             {
132                 b.boneObject = Instantiate (boneTemplate);
133                 b.boneObject.transform.SetParent
134                 (this.transform, false);
135             }
136
137             b.boneObject.transform.up =
138             transform.worldToLocalMatrix.MultiplyVector (to -
139             from);
140             b.boneObject.transform.localPosition = (from +
141             to) / 2.0f;
142             b.boneObject.transform.localScale = new Vector3
143             (0.3f, 1.0f * len, 0.3f);
144
145             if ((rawPoseData[(int) b.from * 3 + 2] < 0.0f)
146             || (rawPoseData[(int) b.to * 3 + 2] < 0.0f))
147             {
148                 b.boneObject.SetActive (false);
149             }
150             else
151             {
152                 b.boneObject.SetActive (true);
153             }
154 }
```

```

145     }
146   }
147 }
148
149 public void updatePose (float [] rawPoseData)
150 {
151   drawPosePoints (rawPoseData);
152   drawSkeleton (rawPoseData);
153 }
154 }
```

Uncomment the lines in the FetchPose script that refer to the PoseSkeleton. Provide the PoseSkeletonManager as extra required parameter for the PoseManager. Running the application should show a visual representation of the skeleton superimposed on the camera image.

6. This application can now be deployed on a mobile device. Some extra settings can be required:
  - Under Player Settings/Other Settings, set the Target Architecture to ARM64. The build script doesn't produce a 32 bit version of the tensorflow library although this could be attempted if actually required.
  - Under Player Settings/Other Settings, set the Scripting Backend to IL2CPP to better handle the native code.
  - Make sure the PoseNet model file is in the StreamingAssets folder of the project. This is a special folder which is packaged with the application when it is transferred to the mobile device.

Remember to tap on the screen whenever you want to invoke the pose recognition.

7. The lack of a depth camera means that the resulting skeletons are two dimensional. This can be sufficient for some forms of gesture recognition in this form. If we do want to have a Kinect equivalent though, it would be helpful to perform the pose detection on remote devices that have a full view of the participant.

Several devices could then potentially provide more detailed three dimensional information about the pose.

As with component 21.1.1, import the PUN 2 Free package from the Unity store and provide an AppID created by registering a new application on the dashboard on the Photon web site. Create a PhotonManager object in the scene containing the script component outlined in Algorithm 19.15, to join a defined room.

Code Listing 19.15: PhotonManager. Minimal room joining steps.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using Photon.Pun;
6  using Photon.Realtime;
7
8  public class PhotonManager : MonoBehaviourPunCallbacks
9  {
10    void Start()
11    {
12      PhotonNetwork.ConnectUsingSettings();
13    }
14
15    public override void OnConnectedToMaster()
16    {
17      RoomOptions roomopt = new RoomOptions ();
18      PhotonNetwork.JoinOrCreateRoom ("ApplicationRoom",
19      roomopt, new TypedLobby ("ApplicationLobby",
20      LobbyType.Default));
21    }
22
23    public override void OnJoinedRoom()
24    {
25      Debug.Log ("Joined room with " +
26      PhotonNetwork.CurrentRoom.PlayerCount + "
27      participants.");
28    }
29 }
```

8. Transmission of the pose captured from the tracking devices to the devices used by the participant's makes use of the remote procedure call (RPC) functionality provided by Photon. This is equivalent to calling a function, but just that the function called is on another machine. The version in this example shares the

pose with all other devices. The example also creates a separate scene element and component for this functionality, to clearly isolate the networking portion of the process. The functionality could equally well be included as part of the Fetch Pose script.

Create a new empty object in the scene called NetworkedPose, and a C# script of the same name. Add a PhotonView component to this, which will take care of the actual network communication. The code for the NetworkedPose script is provided in Algorithm 19.16.

Code Listing 19.16: NetworkedPose. Transferring a pose from one device to all others.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Photon.Pun;
5
6  public class NetworkedPose : MonoBehaviourPun
7  {
8      [Tooltip ("The skeleton visualizer prefab")]
9      public GameObject skeletonVisualizerTemplate;
10
11     [Tooltip ("The pose manager object to hold each
12     ↳ skeleton")]
13     public GameObject poseManager;
14
15     private Dictionary <Photon.Realtime.Player, GameObject>
16     ↳ skeletons;
17
18     void Start ()
19     {
20         skeletons = new Dictionary <Photon.Realtime.Player,
21         ↳ GameObject> ();
22     }
23
24     [PunRPC]
25     void NetworkPose (float [] pose, PhotonMessageInfo info)
26     {
27         if (!skeletons.ContainsKey (info.Sender))
28         {
29             GameObject skel = Instantiate
30             (skeletonVisualizerTemplate);
31             if (poseManager != null)
32             {
33                 skel.transform.SetParent (poseManager.transform,
34                 ↳ false);
35             }
36             skeletons[info.Sender] = skel;
37     }
38 }
```

```

32     }
33     skeletons[info.Sender].GetComponent <PoseSkeleton>
34     ↵ ().updatePose (pose);
35   }
36 
37   public void updatePose (float [] rawPoseData)
38   {
39     PhotonView photonView = PhotonView.Get (this);
40     photonView.RPC ("NetworkPose", RpcTarget.All,
41     ↵ rawPoseData);
42   }
43 }
```

This script emulates the updatePose method provided by the PoseSkeleton. The difference is that this method invokes an RPC call to the function NetworkPose which runs on every machine (including the one sending the message). This function in turn identifies the source of the message, creates a new PoseSkeleton if none exists for that source, and update the pose of that skeleton. Thus a copy of that pose is displayed on all participating devices.

The NetworkPose component does need a skeleton template to instantiate. This can be created from the PoseSkeletonManager created previously. The original PoseSkeletonManager can now actually be removed, and the references in the Fetch Pose script can be replaced with calls to the NetworkedPose.

---

```

1  //public PoseSkeleton poseVisualizer;
2  public NetworkedPose poseTransmitter;
3  ...
4  //poseVisualizer.updatePose (pose);
5  poseTransmitter.updatePose (pose);
```

---

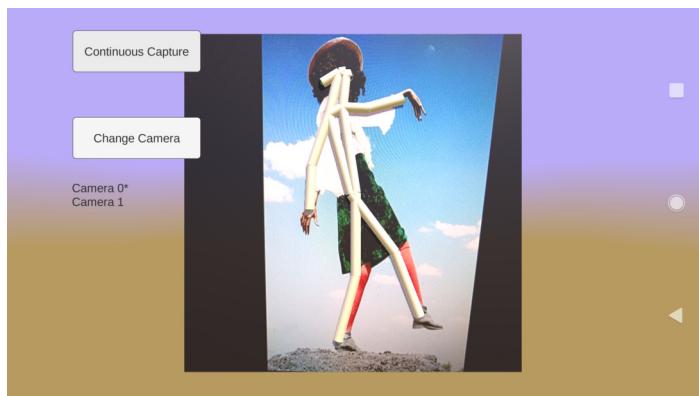
## 19.7 Tracking facial expression and head pose

### Component 19.7.1: Face Tracking.

Anamorphic

Face tracking is a convenient way of capturing facial state which can then be used to animate avatars, performed expression recognition, or support presence in

Figure 19.6.2:  
Representation of  
the body pose.



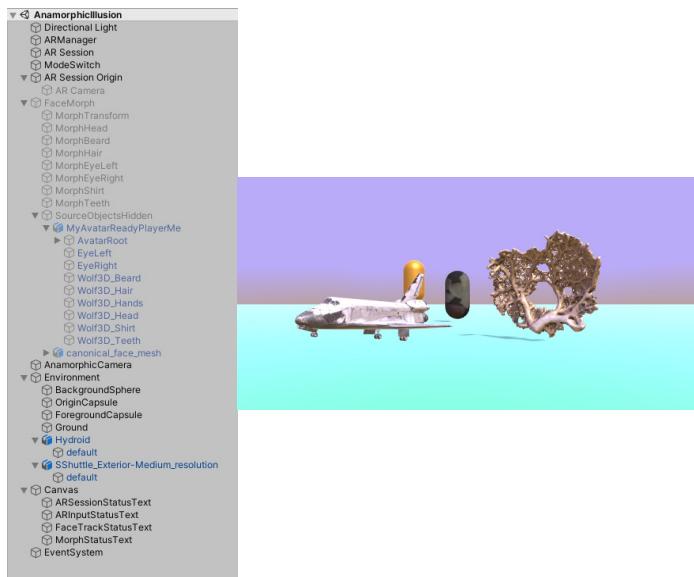
a multi-participant virtual environment. One relatively underutilized aspect of face tracking is to use this facility as a form of gaze tracking. Marker tracking functions have already had a dual role: both to track the position of a marker relative to the camera, but also to track the camera and its associated device relative to fixed reference points in the environment. Face tracking thus also offers the ability to track the position of the eye (the viewpoint in the physical environment) which provides insights into the relative positions of the eye and the device that the participant is looking at.

Gaze tracking becomes relevant when considering large scale projected augmented realities. The large scale immersive environments that use images projected onto several walls of a room (CAVEs) use head tracking to provide the correct perspective view of the 3D environment being represented. The importance of being able to correctly represent the view from a given point is effectively demonstrated in the creation of anamorphic illusions.

This component demonstrates two opportunities with face tracking. The first is the animation of an avatar. The second recreates a small scale CAVE, with a single screen that just happens to be the size of the screen of your mobile device. The device is placed at a fixed point in the room, and the participant allowed to move around in front of it. The component demonstrates how the 3D

scene can be presented just through the motion parallax without the need for any stereoscopic display hardware. This example could be extended to larger fixed screens by separating the head tracking and display elements, for example by using networking functionality.

Figure 19.7.1: Face tracking can be used to present 3D content through movement rather than stereopsis.



1. Start a new Unity project. Set this up for deployment to an Android platform.
2. Open the Package Manager (under the Window Menu option), and make sure that the ARFoundation package is added to the project. Install the ARCore XR Plugin to allow AR Foundation to use the facilities provided by AR Core. Note that in some versions of Unity this might cause incompatibilities with the gradle build system. Selecting a slightly older version (such as 4.0.8 for both ARFoundation and the ARCore XR Plugin) is only way of addressing this without having to reconfigure your software.
3. Under the project settings, and XR Plug-in Management, make sure that AR Core is enabled under the android settings.

4. Add an AR Session object to the scene (in the Hierarchy window, under the XR category of objects). This acts as a manager of the augmented reality session for the application. Only one of these objects may be present in any application.

This object provides ways of monitoring the status of the AR experience. To illustrate the information available, create a few text objects (using Text Mesh Pro) in the scene, and an empty object named ARManager and connect them up to the scripts provided in Algorithm 19.17. You may wish to disable this component at a later point as it is not required for any of the later functionality. However it is useful in validating the state of the ARFoundation functionality on your device while testing.

Code Listing 19.17: FoundationStatus. Reviewing the level of support on your device for AR Foundation.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using UnityEngine.XR.ARFoundation;
6  using TMPro;
7
8  public class FoundationStatus : MonoBehaviour
9  {
10     [Tooltip ("Provide the AR Session object for this
11     field")]
12     public ARInputManager arInputManager;
13
14     public TextMeshProUGUI sessionStatusText;
15     public TextMeshProUGUI inputStatusText;
16
17     void Update()
18     {
19         // Update information from the ARSession object.
20         sessionStatusText.text = "State: " + ARSession.state +
21         " - " + ARSession.notTrackingReason;
22
23         // Update information from the ARInputManager object.
24         inputStatusText.text = "Input: ";
25         if (arInputManager.subsystem != null)
26         {
27             inputStatusText.text +=
28             arInputManager.subsystem.running + " - ";
```

```

26         List<UnityEngine.XR.InputDevice> devices = new List<
27             UnityEngine.XR.InputDevice> ();
28         arInputManager.subsystem.TryGetInputDevices (devices);
29         if (devices.Count > 0)
30         {
31             Vector3 position;
32             Quaternion rotation;
33             devices[0].TryGetFeatureValue
34             (UnityEngine.XR.CommonUsages.colorCameraPosition, out
35             position);
36             devices[0].TryGetFeatureValue
37             (UnityEngine.XR.CommonUsages.colorCameraRotation, out
38             rotation);
39             inputStatusText.text += position.ToString ("F4") +
40             ", " + rotation.ToString ("F4") + " ";
41         }
42
43         // This code is not strictly necessary. However it
44         // is handy when encountering
45         // a new device, to see what input is available.
46         var inputDevices = new
47         List<UnityEngine.XR.InputDevice>();
48         UnityEngine.XR.InputDevices.GetDevices(inputDevices);
49
50         foreach (var device in inputDevices)
51         {
52             inputStatusText.text += string.Format ("\nDevice
53             name '{0}' characteristics '{1}': ", device.name,
54             device.characteristics);
55
56             var inputFeatures = new
57             List<UnityEngine.XR.InputFeatureUsage>();
58             if (device.TryGetFeatureUsages(inputFeatures))
59             {
60                 foreach (var feature in inputFeatures)
61                 {
62                     inputStatusText.text += string.Format ("Feature
63                     {0} ", feature.name);
64                 }
65             }
66         }
67     }
68 }

```

5. Add an AR Session Origin to the scene (also in the Hierarchy window, under the XR category). This provides the coordinate system for the objects tracked in the physical space, and matches them to the coordinate system used in virtual space by the objects in the Unity

scene.

The AR Session Origin has its own camera as a child object. At this point, disable any other cameras in the scene.

When deployed to the android phone, this provides another camera whose position is based on the physical position of the phone so that the physical camera on the phone and the virtual camera in the application undergo the same motion. This allows virtual content to be registered to a physical position. The AR Camera Background component of the AR Camera child of the AR Session Origin replicates the view from the physical camera as a background image on the screen. This can be disabled if only the virtual content should be visible.

6. Plane tracking can be achieved by adding a AR Plane Manager component to the AR Session Origin object (using the Inspector window). This component requires a prefab parameter. The AR Default Plane is an appropriate value for this parameter. Add an AR Default Plane to the scene (from the XR category in the Hierarchy view), and then drag this into the Prefabs folder. Use this prefab as the parameter for the AR Plane Manager.

Deploy this version to the phone, and confirm that planes are detected on the floor and walls.

7. The same process can be repeated for Point Clouds and Face Tracking respectively. Note that face tracking currently only works with the front facing camera. This can be changed using the Facing Direction parameter of the AR Camera Manager component attached to the camera which is the child of the AR Session Origin. Add a BasicFace as the face prefab field of the AR Face Manager in order to see the face tracking in action. The following steps require the AR Face Manager to be enabled, but will replace the default face. The face prefab field will need to be set to None, to prevent two overlapping faces from being displayed.

8. Add a FaceProperties component to the ARManager object, using the code provided in Algorithm 19.18. This captures face update events from the face tracking system, and can share them with other components.

Code Listing 19.18: FaceProperties. Updates on the face tracking are received by this class, and can be relayed to other components that use the information.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using UnityEngine.XR.ARFoundation;
6  using UnityEngine.XR.ARSubsystems;
7  using TMPro;
8
9  public class FaceProperties : MonoBehaviour
10 {
11     public TextMeshProUGUI faceStatusText;
12
13     public ARFaceManager faceManager;
14
15     [Tooltip ("Provide the AR Session object for this
16     ↵ field")]
17     public ARInputManager arInputManager;
18
19     public GameObject sess0origin;
20
21     public delegate void FaceEventHandler (ARFace face);
22     public event FaceEventHandler faceHandler;
23
24     void OnEnable()
25     {
26         faceManager.facesChanged += faceChanged;
27     }
28
29     void OnDisable()
30     {
31         faceManager.facesChanged -= faceChanged;
32     }
33
34     private void faceChanged (ARFacesChangedEventArgs ev)
35     {
36 //      faceStatusText.text = "change - " + ev.added.Count
37 //      ↵ + ev.updated.Count + ev.removed.Count;
38
39     if (ev.updated.Count > 0)
40     {
41         ARFace face = ev.updated[0];
42         faceHandler.Invoke (face);
43     }
44 }
```

42  
43

}

9. The face tracking provided is rather limited. Rather than provide specific facial features which could be used to directly animate a model, the AR Foundation facial tracking has elected to provide a facial mesh that can be rendered directly. This does make it harder to extract key facial landmarks, but can be employed as the basis for deforming existing objects with the expressions that have been captured.

The next steps describe a component that deforms or warps an existing avatar object to correspond with the movements of the animated face mesh provided by AR Foundation. The way it works is to compute the difference between the current animated face, and the face in its rest position. These changes are then added to the avatar object so that it deforms in the same way. Ideally the avatar object should have a human face to it, although it is possible to adapt this process to embed facial animation into any mesh.

10. To create an avatar:

- (a) Create an avatar at <https://readyplayer.me/>. You can style this directly, or if you create a new avatar it will allow you the opportunity to take a photograph as input. Once satisfied, download the resulting file. This will be a .glb file.
- (b) Unity prefers a .fbx file. There are several ways to convert the file. One approach is to:
  - i. Start blender. Delete everything in the scene.
  - ii. Import the .glb file.
  - iii. Export the .fbx file. Copy the .fbx file to your Assets folder of your unity project. You may need to take extra steps to access the textures and materials. While exporting in blender, make sure in the fbx export settings (on the file save window) have

the Path Mode set to Copy, and the icon just next to that set (to Embed Textures). In Unity, click on the fbx file in the Assets folder, and use the ‘Extract Textures’ and ‘Extract Materials’ tools in the Materials tab of the Import Settings to ensure materials and textures are imported, and mapped to their corresponding objects. All of these import settings are available in the inspector panel when the fbx file is selected.

- iv. Some of the textures might still not make it into Unity (such as the eye). Select the material in blender, find the node providing the texture and save it to a file. That file can then be added to the Unity project, and set as a texture on the corresponding material.
  - v. Set Read/Write enabled on the Model properties of the import settings. This will allow access to the vertex information from these models.
11. Extract the canonical face mesh from the AR Core SDK (<https://developers.google.com/ar/develop/downloads>). The only file required is canonical\_face\_mesh.fbx, which can also be added to the Unity project.

When importing this, we need the number of vertices to match the number provided by the face tracking system (currently 468). Setting Normals to None in the Model pane of the Import Settings prevents Unity from trying to set these, and splitting the mesh. Since we need only geometry information from this mesh, this will not affect the visual appearance.

We also really need the canonical face mesh to be an exact duplicate of the mesh that will be returned by the face tracking system. This means that Unity is not allowed to reorganize or optimize the mesh structure in any way. Make sure Optimize Mesh is set to ‘Nothing’.

12. Add a new empty object to the scene, and name it FaceMorph. This will be the parent of all of the facial

animation objects created in the subsequent steps. Ensure that the transform component is set to the default value (at the origin). This process involves some careful alignment of objects so care does need to be taken to avoid introducing stray offsets.

The first step in being able to apply the offsets from one object to another is to ensure both are aligned. Create a child empty object for FaceMorph and name it SourceObjects. Make sure the transform component is at defaults, and add the avatar and canonical face objects as children. Manipulate the two objects to make sure the face portions overlap as accurately as possible. While it doesn't really matter where the two objects end up (as long as they are aligned) it is worth positioning them in front of the camera so that they can be monitored for alignment while completing the other stages. These need to remain in the scene, but need not be visible. Once the application is working, the visibility of the SourceObjects object can be toggled to off.

13. Add the FaceMorph script in Algorithms 19.19 to the FaceMorph object. This needs a number of parameters to be supplied.
  - (a) Firstly, create an empty object named MorphTransform as a child of FaceMorph. This needs to have its transform component set to the values required to move the center of the avatar's head to the origin, and rotate and scale it to match the canonical face. This is used to position the avatar head at the position of the participant's head as tracked by the AR Foundation face tracker. Any alignment issues can be addressed by tweaking the values here.
  - (b) The Canonical Object field needs to be assigned the facemesh object that is found in the canonical face mesh. Use the object that is part of the SourceObjects child object of the FaceMorph object.

- (c) The Face Source field needs to be given the AR Manager object, which has the FaceProperties component.
- (d) The Morph Status Text field can be assigned a text field on the canvas in case any debugging is required.
- (e) The Morphs list needs to have its size set to the number of individual objects making up the avatar. Each of those objects in turn requires:
  - i. A source mesh. This is the object belonging to the avatar, which is found under the SourceObjects object within the FaceMorph object.
  - ii. A target object. Create a new child object of FaceMorph for each of these. This child object needs to have a MeshFilter and MeshRenderer component. Make sure that the mesh field of the MeshFilter has the same mesh as used by the source mesh (in theory this should not be necessary but the texture coordinates don't seem to copy correctly if this is not the case. Bug fixes welcome). Assign the material used on the source mesh to this target object as well.
  - iii. The transform object to origin. Set this to the MorphTransform object that was created above.

Deploy this to the mobile device. If everything works as intended, you should be able to animate your avatar with your own facial expressions, as shown in Figure 19.7.2.

Code Listing 19.19: FaceMorph. Avatar deformation is achieved by transferring the changes to the canonical face mesh to the avatar mesh provided.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Unity.Collections;
5
6  using UnityEngine.XR.ARFoundation;
7  using UnityEngine.XR.ARSubsystems;
```

```
8  using TMPro;
9
10 public class FaceMorph : MonoBehaviour
11 {
12     [Tooltip ("Text element for status and debugging
13      ↳ information")]
14     public TextMeshProUGUI morphStatusText;
15
16     // Handle multiple objects linked to the face morph.
17     ↳ Useful
18     // for avatar objects that have several pieces.
19     [System.Serializable]
20     public class FaceMorphElement
21     {
22
23         public GameObject morphSourceMesh;
24         public GameObject morphTargetObject;
25         public Transform morphTransformToObject;
26
27         // Internal element, used to map vertices on
28         // the avatar to vertices on the canonical face.
29         [System.NonSerialized]
30         public int [ ] [ ] morphCanonicalToFace;
31     }
32
33     [SerializeField]
34     [Tooltip ("Provide a list of objects that contain:
35      ↳ original mesh, the destination mesh, and any offsets
36      ↳ required")]
37     public FaceMorphElement [ ] morphs;
38
39     [Tooltip ("This is the object with the FaceProperties
40      ↳ component, that connects to the face tracker from AR
41      ↳ Foundation")]
42     public FaceProperties faceSource;
43
44     [Tooltip ("This is the reference face, positioned and
45      ↳ aligned to the avatar object")]
46     public GameObject canonicalObject;
47
48     private Mesh canonicalMesh;
49
50     void Start()
51     {
52         canonicalMesh = canonicalObject.GetComponent<SkinnedMeshRenderer>().sharedMesh;
53
54         createMapping ();
55
56         faceSource.faceHandler += drawFace;
57     }
58
59     // Presently only using the first neighbour found, but
60     ↳ with the
```

```

52    // intention that smoother animation could be achieved
53    // by using
54    // several close neighbours from the control mesh to
55    // drive the
56    // deformation.
57    private int [] findWeightsAndBasis (Vector3 p, Vector3
58    [] source)
59    {
60
61        int numNeighbours = 4; // How many points to use as
62        // the basis.
63
64        List <(int, float)> best = new List <(int, float)> ();
65
66        for (int i = 0; i < source.Length; i++)
67        {
68            float distance = Vector3.Distance (p, source[i]);
69
70            best.Add ((i, distance));
71        }
72
73        best.Sort ((x, y) => x.Item2.CompareTo (y.Item2));
74
75        int [] basisIndices = new int [numNeighbours];
76        float [] basisWeights = new float [numNeighbours];
77        for (int i = 0; i < numNeighbours; i++)
78        {
79            basisIndices[i] = best[i].Item1;
80        }
81
82        return basisIndices;
83    }
84
85    private void createMapping ()
86    {
87        Vector3 [] refVerts = new Vector3
88        [canonicalMesh.vertices.Length];
89        for (int i = 0; i < canonicalMesh.vertices.Length; i++)
90        {
91            refVerts[i] =
92            canonicalObject.transform.TransformPoint
93            (canonicalMesh.vertices[i]);
94        }
95
96        foreach (FaceMorphElement fme in morphs)
97        {
98            Mesh sourceMesh = fme.morphSourceMesh.GetComponent
99            <SkinnedMeshRenderer> ().sharedMesh;
100           // Note that retrieving vertices is a time consuming
101           // operation. Do this
102           // once, outside the loop.
103           Vector3 [] morphSourceVertices = sourceMesh.vertices;
104           fme.morphCanonicalToFace = new int
105           [morphSourceVertices.Length][];

```

```

95     for (int i = 0; i < morphSourceVertices.Length; i++)
96     {
97         // Get the position of the vertex in world
98         // coordinates,
99         // so we can match the two aligned objects.
100        Vector3 fw = 
101        fme.morphSourceMesh.transform.TransformPoint
102        (morphSourceVertices[i]);
103        fme.morphCanonicalToFace[i] = findWeightsAndBasis
104        (fw, refverts);
105        // Debug.Log ("fv " + fw.ToString ("F4"))
106        + " " + canonicalToFace[i][0]);
107    }
108}
109
110public void drawFace (ARFace face)
111{
112    // Step 1: Just use the mesh provided by the ARFace
113    // directly.
114    // target.mesh.SetVertices (face.vertices);
115    // target.mesh.SetIndices (face.indices,
116    // MeshTopology.Triangles, 0, false);
117    // target.mesh.RecalculateBounds ();
118    // target.mesh.SetNormals (face.normals);
119    // target.mesh.RecalculateNormals ();
120
121    // Step 2: Use the face Mesh from the avatar.
122    // FIXME: tweak transform.
123    // target.mesh.SetVertices (faceMesh.vertices);
124    // target.mesh.SetIndices (faceMesh.GetIndices
125    // (0, false), MeshTopology.Triangles, 0, false);
126    // target.mesh.RecalculateBounds ();
127    // target.mesh.SetNormals (faceMesh.normals);
128    // target.mesh.RecalculateNormals ();
129
130    // Step 3: Deform avatar to match deformation on
131    // ARFace mesh.
132    Vector3 [] canVertices = canonicalMesh.vertices;
133    NativeArray<Vector3> faceVertices = face.vertices;
134    foreach (FaceMorphElement fme in morphs)
135    {
136        Mesh sourceMesh = fme.morphSourceMesh.GetComponent
137        <SkinnedMeshRenderer> ().sharedMesh;
138        Vector3 [] morphSourceVertices = sourceMesh.vertices;
139        Vector3 [] morphTargetVertices = new Vector3
140        [morphSourceVertices.Length];
141        for (int i = 0; i < morphSourceVertices.Length; i++)
142        {
143            int cIndex = fme.morphCanonicalToFace[i][0];
144            Vector3 disp = faceVertices[cIndex] -
145            canVertices[cIndex];

```

```

135           disp =
136             ↵ fme.morphSourceMesh.transform.InverseTransformVector
137             ↵ (canonicalObject.transform.TransformVector (disp));
138             ↵ morphTargetVertices[i] =
139               ↵ fme.morphTransformObjectToOrigin.TransformPoint
140               ↵ (morphSourceVertices[i] + disp);
141             ↵ }
142
143
144           // FIXME: This doesn't seem to work right unless the
145           // meshfilter is assigned a
146           // copy of the correct mesh. Colours or uvs are not
147           // being copied correctly.
148           Mesh mesh = fme.morphTargetObject.GetComponent
149             <MeshFilter> () .mesh;
150             ↵ //           mesh.Clear ();
151             ↵ mesh.triangles = null;
152             ↵ mesh.vertices = morphTargetVertices;
153             ↵ mesh.triangles = sourceMesh.triangles;
154             ↵ mesh.RecalculateBounds ();
155             ↵ mesh.RecalculateNormals ();
156             ↵ mesh.uv = sourceMesh.uv;
157
158           fme.morphTargetObject.transform.position =
159             ↵ face.transform.position;
160             ↵ fme.morphTargetObject.transform.rotation =
161             ↵ face.transform.rotation;
162             ↵ fme.morphTargetObject.transform.localScale = new
163             ↵ Vector3 (1, 1, 1);
164             ↵ }
165           }
166         }
167       }
168     }
169   }
170 }
```

Figure 19.7.2:  
Avatar animation using facial expressions.



14. The second modality for this component involves using the face tracking facility to provide a view direction

dependent display of the 3D structure of a scene, producing dynamic anamorphic illusions that can recreate the original scene through a structure from motion approach as the head moves.

At this point, disable the FaceMorph and AR Camera objects in the scene by setting them to be inactive (using the checkbox at the top of the inspector panel, for each of them). A new camera object should be added to the scene and named AnamorphicCamera. Add a new C# script based component to this named AnamorphicProjection by using the code provided in Algorithms 19.20.

Code Listing 19.20: AnamorphicProjection. The anamorphic camera projects the scene onto a region offset to match the position of the tracked viewer.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using UnityEngine.XR.ARFoundation;
6  using UnityEngine.XR.ARSubsystems;
7  using TMPro;
8
9  public class AnamorphicProjection : MonoBehaviour {
10
11     [Tooltip ("This is the object with the FaceProperties
12     component, that connects to the face tracker from AR
13     Foundation")]
14     public FaceProperties faceSource;
15
16     [Tooltip ("Text element for status and debugging
17     information")]
18     public TextMeshProUGUI anamorphStatusText;
19
20     void Start()
21     {
22         faceSource.faceHandler += drawFace;
23     }
24
25     // See the unity example on the camera's perspective
26     // matrix.
27     static Matrix4x4 PerspectiveOffCenter(float left, float
28     right, float bottom, float top, float near, float far)
29     {
30         float x = 2.0F * near / (right - left);
31         float y = 2.0F * near / (top - bottom);
32         float a = (right + left) / (right - left);
33
34         Matrix4x4 matrix = new Matrix4x4();
35         matrix[0][0] = x;
36         matrix[1][1] = y;
37         matrix[2][2] = -far / (near - far);
38         matrix[3][2] = -1.0F;
39         matrix[0][3] = -left * x / (right - left);
40         matrix[1][3] = -bottom * y / (top - bottom);
41         matrix[2][3] = -far * (near + far) / (near - far);
42
43         return matrix;
44     }
45 }
```

```

28     float b = (top + bottom) / (top - bottom);
29     float c = -(far + near) / (far - near);
30     float d = -(2.0F * far * near) / (far - near);
31     float e = -1.0F;
32     Matrix4x4 m = new Matrix4x4();
33     m[0, 0] = x; m[0, 1] = 0; m[0, 2] = a; m[0, 3] = 0;
34     m[1, 0] = 0; m[1, 1] = y; m[1, 2] = b; m[1, 3] = 0;
35     m[2, 0] = 0; m[2, 1] = 0; m[2, 2] = c; m[2, 3] = d;
36     m[3, 0] = 0; m[3, 1] = 0; m[3, 2] = e; m[3, 3] = 0;
37     return m;
38 }
39
40     public void drawFace (ARFace face)
41 {
42 //     anamorphStatusText.text =
43 //         transform.position.ToString ("F4") + " " +
44 //         transform.forward.ToString ("F4");
45
46     float px = -face.transform.position.x;
47     float py = face.transform.position.y;
48     float pz = -face.transform.position.z;
49     transform.position = new Vector3 (px, py, pz);
50     // It doesn't matter which way the viewer is looking.
51     // The
52     // display will still show what is visible in the
53     // direction
54     // of the display.
55     transform.forward = Vector3.forward;
56     transform.up = Vector3.up;
57
58     // Size of the display device.
59     float sizex = 0.1f;
60     float sizey = 0.05f;
61
62     float near = 0.05f;
63     float far = 10.0f;
64
65     // calculate position of display screen projected onto
66     // the near clipping plane.
67     float left = ((-px - sizex) * (near / -pz));
68     float right = ((-px + sizex) * (near / -pz));
69     float bottom = ((-py - sizey) * (near / -pz));
70     float top = ((-py + sizey) * (near / -pz));
71
72     Matrix4x4 anamorphicProjection = new Matrix4x4 ();
73     anamorphicProjection = PerspectiveOffCenter (left,
74         right,
75                                         bottom,
76         top,
77                                         near, far);
78
79     GL.invertCulling = false;

```

```
74     GetComponent <Camera> () .nearClipPlane = near;
75     GetComponent <Camera> () .farClipPlane = far;
76     GetComponent <Camera> () .projectionMatrix =
77     ↵ anamorphicProjection;
78 }
```

This component requires the FaceManager object to be provided as the Face Source parameter. This allows the component to receive updates from the face tracking, from which it extracts the position of the eye (in this case, it is approximated using the position of the head). The anamorphic projection component changes the projection operation within the camera component to ensure that the display region corresponds to the position of the screen of the mobile device which varies as the head moves relative to the device. Depending on the size of the screen on your mobile device, you may want to adjust the sizex and sizey values used to match the size of the screen in meters.

15. We need to include some objects in a sample 3D scene to be presented using the dynamic anamorphic illusion. The scene set up assumes the device (and its screen) are position at the origin, with the screen pointing down the negative z-axis (i.e. objects with a negative z coordinate should appear to be between viewer and the screen, while objects with a positive z coordinate should appear to be behind the screen). The screen is likely to be quite small, so objects may need to be scaled to similar proportions so that they are visible on the screen.

Create a new empty and name this Environment. Add a number of child objects to represent the contents of the virtual environment that will be shown. A reference object at the origin is useful, since this will be the center of the display device and should remain fixed even with the face position changes. Objects in front of, and behind, the display are also useful when testing the effect.

16. Deploy the application to the mobile device. The expected configuration for this component is to have the device fixed to a point in the physical world, such as resting on a table. It should be placed to match the proportions provided to the sizex and sizey values in the AnamorphicProjection script. In particular, if the horizontal dimension corresponds to the longer side of the device then it should be placed to show in the screen in a landscape view. The front camera should be used for face tracking, so the person being tracked can also see the screen at the same time.

The display should respond as you move your head around so make sure your face stays within the view frustum of the camera. You should be able to get an impression of the 3D structure of the scene based just on your head movements. As this example does not need stereopsis, you can try closing one eye to avoid perceiving the flat nature of the screen.

Figure 19.7.3: One frame of a dynamic anamorphic illusion.



17. Create a new empty object called ModeSwitch and attach the ToggleActive script from Algorithm 19.21 to it. This component allows a tap on the screen to cycle between different sets of enabled objects. Create two sets. The first set is the functionality for the facial animation. Include the FaceMorph object and the AR Camera in this list. The second set provides the anamorphic dis-

play driven by the face tracking. Add the Environment and AnamorphicCamera objects to this.

You should now be able to switch between facial animation of your avatar, and gaze tracked viewing of the environment you created.

Code Listing 19.21: ToggleActive. Face tracking can switch between two modes based on a mouse click or screen touch.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5
6  public class ToggleActive : MonoBehaviour
7  {
8      [System.Serializable]
9      public class GameObjectList
10     {
11         public List <GameObject> objects;
12     }
13
14     public List <GameObjectList> selections;
15
16     private int currentSelection = 0;
17
18     private void switchSelection ()
19     {
20         if (selections.Count > 0)
21         {
22             currentSelection = (currentSelection + 1) %
23             selections.Count;
24             for (int i = 0; i < selections.Count; i++)
25             {
26                 foreach (GameObject g in selections[i].objects)
27                 {
28                     g.SetActive (currentSelection == i);
29                 }
30             }
31         }
32
33         void Update()
34         {
35             if (Input.GetMouseButtonUp (0))
36             {
37                 switchSelection ();
38             }
39         }
40     }
```

## 19.8 Supporting interaction using speech recognition

### Component 19.8.1: Speech recognition.

Speech  
Recognition

Individual platforms often have their own speech services, for tasks such as synthesizing speech from text, converting spoken words into a written form (speech to text), or translating from one language to another. This example specifically focuses on a platform independent solution using an online service to reduce as far as possible the need to recreate system specific components on each targeted platform.

The goal is to demonstrate the use of application programming interfaces (APIs) provided by online service providers. These interfaces specify the form of data that can be exchanged with the service and define the operations that can be performed on this data. The category of API that is used is a representational state transfer (REST) interface. A defining property of the REST interfaces is that they are used for creating web services and hence available to any device that can access the web without the need for any additional network configuration. Systems that use REST services typically make a request to a server operated by the third party service provider. These requests may provide some information (such as the HTTP POST request) to the server, or may request information (such as the HTTP GET request) which is provided in the response returned from the server. The relationship with the server lasts only for the duration of a single transaction (memoryless or stateless) and so any information required for subsequent transactions must be included in the next request.

An access key is a common requirement for using such services. Keys are available by signing up to the service, and either subscribing to particular services or creating projects for which specific keys are required. The

key is used to manage the number of transactions that an individual application may conduct with the service. Higher levels of service are usually associated with more expensive keys.

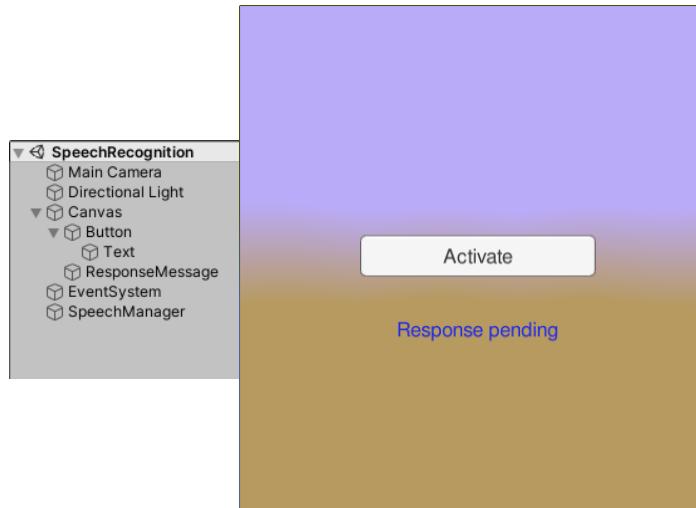
The key itself is usually a string of characters. Anyone with access to the key can make use of the service and so it is recommended that these are not shared publicly, or added to the version controlled copies of source code. Keys can be replaced if they do leak but this generally means that any application using the old key will immediately stop working until it is updated.

The individual operations provided by the particular REST API are specified in the documentation provided. Since most of these use the HTTP protocol used for communication on the web, each specification states:

- The type of operation required. Usually a GET if most of the data transferred is in the response from the server, or a POST if data is being transferred *to* the server.
- Any parameters or state information that must be transferred to the server. Information in an HTTP message can be included in several forms:
  - The query string. These are values added to the URL address of the service, usually separated from the address of the server and service by a '?' symbol.
  - Request headers. Extra short amounts of data are included in the header of the request sent to the server. These are normally of the form name:value, where the names are defined in the API documentation, and the values are data corresponding to the named variables.
  - Body. Large blocks of data can be appended after the header in the body of the HTTP message. This is most often used for transferring large media files.
- The form of the response. Even when no data is being transferred, there will be a status code returned to

confirm the success, or explain the reason behind any error that occurs.

Figure 19.8.1:  
Speech recognition  
using a web service.



1. Start with a new project. Set up the project to deploy to the mobile device. Under the player settings, ensure the Internet Access is set to Require, to ensure that the application has permissions to access remote sites.

This example makes use of the speech-to-text facility provided by the Speech Service REST APIs provided by Microsoft Azure (<https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/rest-apis>). Use of these services requires an API key. Short term trial keys are available at <https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/get-started>.

Note that Microsoft have updated their process to remove convenient access to short term trial keys. Accounts can still be created but this process is no longer recommended for prototyping purposes. An alternative process using a comparable API provided by AssemblyAI is described in the last step of these instructions. This uses the same scene structure but replaces the

component accessing the Microsoft API with an equivalent accessing AssemblyAI.

2. This example uses a button to record audio for a particular period of time, typically 5 seconds. Create a new UI/Button which is used to start the recording.
3. The process of converting speech to text through the web service is handled by the SpeechToText function in Algorithms 19.22. This uses a POST operation employing all three forms of parameter passing. Parameters such as the language spoken and the format of the response are provided as part of the query string. The key to access the service, and the definition of the type of audio data being transferred are added to the request header. The actual sound file itself is included in the body of the request.

Code Listing 19.22: SpeechRecognitionREST. A speech to text facility is one of the facilities that can be accessed as an online service.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using System.Net.Security;
6  using System.Security.Cryptography.X509Certificates;
7  using System.Net;
8  using System.IO;
9  using System;
10
11 public class SpeechRecognitionREST : MonoBehaviour {
12
13     public Text outputText;
14
15     // This will need to be replaced with your own key.
16     private string subscriptionKey =
17         "dbeee4817ada647d7acf6f1b781920107";
18     private string token;
19
20     // length of any recording sent. 10 s is the current
21     // limit.
22     public int recordDuration = 5;
23
24     private static bool TrustCertificate(object sender,
25         X509Certificate x509Certificate, X509Chain x509Chain,
26         SslPolicyErrors sslPolicyErrors)
27     {

```

```
24         // all Certificates are accepted
25         return true;
26     }
27
28     void Start ()
29     {
30         ServicePointManager.ServerCertificateValidationCallback =
31             delegate (object sender, X509Certificate certificate,
32                     X509Chain chain, SslPolicyErrors sslPolicyErrors)
33         {
34             // Not really needed, but useful to test access to
35             // service.
36             public void Authentication()
37             {
38                 // Unity webforms do not handle the certificates
39                 // required for https servers.
40
41                 HttpWebRequest request =
42                     (HttpWebRequest)WebRequest.Create("https://westus.api.]
43                     cognitive.microsoft.com/sts/v1.0/issueToken");
44                 request.ContentType =
45                     "application/x-www-form-urlencoded";
46                 request.Method = "POST";
47                 request.Headers["Ocp-Apim-Subscription-Key"] =
48                     subscriptionKey;
49
50                 var response = (HttpWebResponse)request.GetResponse();
51
52                 var responseString = new
53                     StreamReader(response.GetResponseStream()).ReadToEnd();
54                 Debug.Log ("Token received: " + responseString);
55
56                 token = responseString;
57             }
58
59             public void SpeechToText (byte [] wavData)
60             {
61                 // Send the request to the service.
62                 string fetchUri =
63                     "https://westus.stt.speech.microsoft.com/speech/recognition/conversation/cognitiveservices/v1";
64                 HttpWebRequest request =
65                     (HttpWebRequest)WebRequest.Create(fetchUri +
66                     "?language=en-US&format=detailed");
67                 request.ContentType =
68                     "application/x-www-form-urlencoded";
69                 request.Method = "POST";
70                 request.ContentType = "audio/wav; codecs=audio/pcm;
71                     samplerate=16000";
72                 request.Headers["Ocp-Apim-Subscription-Key"] =
73                     subscriptionKey;
74             }
75         }
76     }
77 }
```

```

61     // Read a wav file off the filesystem and send that to
62     // the service. Note that the paths
63     // used may not work when packaged for a mobile
64     // platform. This is useful for testing
65     // under controlled and repeatable conditions (same
66     // input file each time).
67     //
68     // Stream rs = request.GetRequestStream();
69     // FileStream fileStream = new
70     // FileStream(Application.dataPath +
71     // "/SpeechRecognition/Sound/untitled.wav",
72     // FileMode.Open, FileAccess.Read);
73     // byte[] buffer = new byte[4096];
74     // int bytesRead = 0;
75     // while ((bytesRead = fileStream.Read(buffer, 0,
76     // buffer.Length)) != 0) {
77     //     rs.Write(buffer, 0, bytesRead);
78     // }
79     // fileStream.Close();
80     // rs.Close();
81     Stream rs = request.GetRequestStream();
82     rs.Write(wavData, 0, wavData.Length);
83     rs.Close();

84
85
86
87
88
89     var response = (HttpWebResponse)request.GetResponse();
90
91
92
93
94
95
96
97
98
99
100
101
102

```

`var responseString = new  
 StreamReader(response.GetResponseStream()).ReadToEnd();  
 Debug.Log ("Response from service: " + responseString);  
 if (outputText != null)  
 {  
 outputText.text = responseString;  
 }  
}  
  
private IEnumerator recordAudio ()  
{  
 // Set the microphone recording. Service requires 16  
 // kHz sampling.  
 AudioClip audio = Microphone.Start (null, false,  
 recordDuration, 16000);  
yield return new WaitForSeconds (recordDuration);  
 Microphone.End (null);

 // Play the recording back, to validate it was  
 // recorded correctly.  
 AudioSource audioSource = GetComponent<AudioSource>();  
 audioSource.clip = audio;  
 audioSource.Play();

 // Convert it to a wav file, and upload to the service.  
byte [] wavData = ConvertToWav (audio);
}`

```
103     SpeechToText (wavData);
104 }
105
106 public void Trigger ()
107 {
108     //Authentication ();
109
110     StartCoroutine (recordAudio ());
111 }
112
113 // Remaining functions adapted from:
114 // https://gist.github.com/darktable/2317063
115
116 const int HEADER_SIZE = 44;
117
118 static byte [] ConvertToWav (AudioClip clip) {
119
120     var samples = new float[clip.samples];
121
122     clip.GetData(samples, 0);
123
124     Int16[] intData = new Int16[samples.Length];
125     //converting in 2 float[] steps to Int16[], //then
126     // Int16[] to Byte[]
127
128     Byte[] bytesData = new Byte[HEADER_SIZE +
129     // samples.Length * 2];
130     //bytesData array is twice the size of
131     //dataSource array because a float converted in Int16
132     // is 2 bytes.
133
134     int rescaleFactor = 32767; //to convert float to Int16
135
136     WriteHeader (bytesData, clip);
137
138     for (int i = 0; i<samples.Length; i++) {
139         intData[i] = (short) (samples[i] * rescaleFactor);
140         Byte[] byteArr = new Byte[2];
141         byteArr = BitConverter.GetBytes(intData[i]);
142         byteArr.CopyTo(bytesData, HEADER_SIZE + i * 2);
143     }
144
145     return bytesData;
146 }
147
148 static void WriteHeader(byte [] bytesData, AudioClip
149     clip) {
150
151     var hz = clip.frequency;
152     var channels = clip.channels;
153     var samples = clip.samples;
```

```
150     Byte[] riff =
151     ↳ System.Text.Encoding.UTF8.GetBytes("RIFF");
152     riff.CopyTo(bytesData, 0);
153
153     Byte[] chunkSize = BitConverter.GetBytes(HEADER_SIZE +
154     ↳ clip.samples * 2 - 8);
155     chunkSize.CopyTo(bytesData, 4);
156
156     Byte[] wave =
157     ↳ System.Text.Encoding.UTF8.GetBytes("WAVE");
158     wave.CopyTo(bytesData, 8);
159
159     Byte[] fmt = System.Text.Encoding.UTF8.GetBytes("fmt
160     ↳ ");
161     fmt.CopyTo(bytesData, 12);
162
162     Byte[] subChunk1 = BitConverter.GetBytes(16);
163     subChunk1.CopyTo(bytesData, 16);
164
164     UInt16 one = 1;
165
167     Byte[] audioFormat = BitConverter.GetBytes(one);
168     audioFormat.CopyTo(bytesData, 20);
169
170     Byte[] numChannels = BitConverter.GetBytes(channels);
171     numChannels.CopyTo(bytesData, 22);
172
173     Byte[] sampleRate = BitConverter.GetBytes(hz);
174     sampleRate.CopyTo(bytesData, 24);
175
176     Byte[] byteRate = BitConverter.GetBytes(hz * channels
177     ↳ * 2); // sampleRate * bytesPerSample*number of
178     ↳ channels, here 44100*2*2
179     byteRate.CopyTo(bytesData, 28);
180
181     UInt16 blockAlign = (ushort) (channels * 2);
182     BitConverter.GetBytes(blockAlign).CopyTo(bytesData,
183     ↳ 32);
184
185     UInt16 bps = 16;
186     Byte[] bitsPerSample = BitConverter.GetBytes(bps);
187     bitsPerSample.CopyTo(bytesData, 34);
188
188     Byte[] datastring =
189     ↳ System.Text.Encoding.UTF8.GetBytes("data");
190     datastring.CopyTo(bytesData, 36);
191
191     Byte[] subChunk2 = BitConverter.GetBytes(samples *
192     ↳ channels * 2);
193     subChunk2.CopyTo(bytesData, 40);
194 }
```

193

}

Create a new empty object in the scene named SpeechManager. Attach the speech to text component to this empty object. Add an event handler to the button and set this to invoke the Trigger function on the SpeechManager when pressed.

The SpeechManager should also have an AudioSource component included which plays back the recording to verify what was heard. Ensure that this component is also added to the speech manager.

4. The other aspect of this example is the preparation of the media that are used. In this case, a short sound sample is recorded from the default microphone and transferred to the speech to text service. The service then returns a message, containing the text of any words identified, along with additional parameters defining other insights into the sound such as the confidence level of the analysis.

The sound recorded is in the form of a set of amplitude (loudness) levels sampled many times per second. The sampling rate in this case is 16 kHz, representing 16000 samples per second. This is adequate for speech and is specified in the API documentation. Higher rates are typical where better audio fidelity is required. The values recorded by the Unity software from the microphone need to be modified into a slightly different format to be compatible with the WAV format expected by the REST interface. The OGG format can also be used but requires a more complex transformation that compresses the data to use less memory and to take less time to transmit. The WAV format conversion requires just converting the amplitude levels into integers in a defined range, and adding a header containing information about the length and format of the data.

5. The returned value from the speech recognition service is logged to the console. For the benefit of mobile de-

vices, create a UI/Text object as well where this output can be shown on the screen. Connect this text field to the OutputText property of the SpeechManager. Since the output can be quite long, make sure the horizontal and vertical overflow properties of this text object are set to Wrap and Overflow respectively.

6. To use the AssemblyAI API rather than the Microsoft API, complete the steps above and then make these changes:

- (a) Create an SpeechRecognitionAssembly C# script with the code provided in Algorithms 19.23.

Code Listing 19.23: SpeechRecognitionAssembly. The equivalent speech to text facility using AssemblyAI.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using System.Net.Security;
6  using System.Security.Cryptography.X509Certificates;
7  using System.Net;
8  using System.IO;
9  using System;
10
11 public class SpeechRecognitionAssembly : MonoBehaviour {
12
13     public Text outputText;
14
15     // This will need to be replaced with your own key.
16     private string subscriptionKey =
17         "eeaba993aad6483ea613b81be172d079";
18     private string token;
19
20     // length of any recording sent. 10 s is the current
21     // limit.
22     public int recordDuration = 5;
23
24     private static bool TrustCertificate(object sender,
25         X509Certificate x509Certificate, X509Chain
26         x509Chain, SslPolicyErrors sslPolicyErrors)
27     {
28         // all Certificates are accepted
29         return true;
30     }
31
32     void Start ()
33     {

```

```
30     ServicePointManager.ServerCertificateValidationCall];
31     ↵ back =
32     ↵ TrustCertificate;
33   }
34
35   // These structures contain the relevant fields from
36   // the response strings/JSON that is returned by the
37   ↵ service.
38   [System.Serializable]
39   class AssemblyResponse
40   {
41     public string id;
42     public string status;
43     public string text;
44   };
45
46   [System.Serializable]
47   class AssemblyUploadResponse
48   {
49     public string upload_url;
50   };
51
52   // Upload the wav data to the service, and return the
53   ↵ link to this file.
54   private string uploadData (byte [] wavData)
55   {
56     string fetchUri =
57     "https://api.assemblyai.com/v2/upload";
58     HttpWebRequest request =
59     (HttpWebRequest)WebRequest.Create(fetchUri + "");
60     request.Headers["authorization"] = subscriptionKey;
61
62     request.Method = "POST";
63     Stream rs = request.GetRequestStream();
64     rs.Write(wavData, 0, wavData.Length);
65     rs.Close();
66
67     var response =
68     (HttpWebResponse)request.GetResponse();
69
70     var responseString = new StreamReader(response.GetResponse());
71     responseString.ReadToEnd();
72     Debug.Log ("Response from service: " +
73     responseString);
74     if (outputText != null)
75     {
76       outputText.text = responseString;
77     }
78     AssemblyUploadResponse r = JsonUtility.FromJson<AssemblyUploadResponse> (responseString);
79     Debug.Log ("Got id " + r.upload_url);
80     return r.upload_url;
81 }
```

```

72
73     public void SpeechToText (byte [] wavData)
74     {
75         // convert wav to a file on the server.
76         string wavURL = uploadData (wavData);
77
78         // Send the request to the service.
79         string fetchUri =
80             "https://api.assemblyai.com/v2/transcript";
81         HttpWebRequest request =
82             (HttpWebRequest)WebRequest.Create(fetchUri + "");
83         request.ContentType =
84             "application/application/json";
85         request.Headers["authorization"] = subscriptionKey;
86
87         request.Method = "POST";
88         byte [] wData = System.Text.Encoding.ASCII.GetBytes
89             ("{\"audio_url\": \"" + wavURL + "\"}");
90         Stream rs = request.GetRequestStream();
91         rs.Write(wData, 0, wData.Length);
92         rs.Close();
93
94         var response =
95             (HttpWebResponse)request.GetResponse();
96
97         var responseString = new StreamReader(response.GetResponse()
98             .GetResponseStream()).ReadToEnd();
99         Debug.Log ("Response from service: " +
100             responseString);
101        if (outputText != null)
102        {
103            outputText.text = responseString;
104        }
105
106        AssemblyResponse r = JsonUtility.FromJson
107            <AssemblyResponse> (responseString);
108        Debug.Log ("Got id " + r.id);
109
110        // Wait for the speech to text to complete.
111        StartCoroutine (waitForTranscript (r.id));
112    }
113
114    private IEnumerator waitForTranscript (string id)
115    {
116        AssemblyResponse r = null;
117        // We only poll 10 times, at 5 second intervals.
118        // Longer portions of
119        // speech would require changing this.
120        for (int i = 0; i < 10; i++)
121        {
122            string fetchUri =
123                "https://api.assemblyai.com/v2/transcript";

```

```
113     HttpWebRequest request =
114     (HttpWebRequest)WebRequest.Create(fetchUri + "/" +
115     id);
116     request.ContentType =
117     "application/application/json";
118     request.Headers["authorization"] =
119     subscriptionKey;
120     request.Method = "GET";
121     var response =
122     (HttpWebResponse)request.GetResponse();
123
124     var responseString = new StreamReader(response.GetResponseStream()).ReadToEnd();
125     Debug.Log ("Response from service: " +
126     responseString);
127     if (outputText != null)
128     {
129         outputText.text = responseString;
130     }
131     r = JsonUtility.FromJson <AssemblyResponse>
132     (responseString);
133     if (r.status == "completed")
134     {
135         break;
136     }
137     yield return new WaitForSeconds (5.0f);
138
139     }
140
141     private IEnumerator recordAudio ()
142     {
143         // Set the microphone recording. Service requires
144         // 16 kHz sampling.
145         AudioClip audio = Microphone.Start (null, false,
146         recordDuration, 16000);
147         yield return new WaitForSeconds (recordDuration);
148         Microphone.End (null);
149
150         // Play the recording back, to validate it was
151         // recorded correctly.
152         AudioSource audioSource =
153         GetComponent<AudioSource>();
154         audioSource.clip = audio;
155         audioSource.Play();
156
157         // Convert it to a wav file, and upload to the
158         // service.
159         byte [] wavData = ConvertToWav (audio);
160         SpeechToText (wavData);
161     }
```

```

153
154     public void Trigger ()
155     {
156         StartCoroutine (recordAudio ());
157     }
158
159     // Remaining functions adapted from:
160     // https://gist.github.com/darktable/2317063
161
162     const int HEADER_SIZE = 44;
163
164     static byte [] ConvertToWav (AudioClip clip) {
165
166         var samples = new float[clip.samples];
167
168         clip.GetData(samples, 0);
169
170         Int16[] intData = new Int16[samples.Length];
171         //converting in 2 float[] steps to Int16[], //then
172         //Int16[] to Byte[]
173
174         Byte[] bytesData = new Byte[HEADER_SIZE +
175             samples.Length * 2];
176         //bytesData array is twice the size of
177         //dataSource array because a float converted in
178         //Int16 is 2 bytes.
179
180         int rescaleFactor = 32767; //to convert float to
181         //Int16
182
183         WriteHeader (bytesData, clip);
184
185         for (int i = 0; i<samples.Length; i++) {
186             intData[i] = (short) (samples[i] * rescaleFactor);
187             Byte[] byteArr = new Byte[2];
188             byteArr = BitConverter.GetBytes(intData[i]);
189             byteArr.CopyTo(bytesData, HEADER_SIZE + i * 2);
190         }
191
192         return bytesData;
193     }
194
195     static void WriteHeader(byte [] bytesData, AudioClip
196     // clip) {
197
198         var hz = clip.frequency;
199         var channels = clip.channels;
200         var samples = clip.samples;
201
202         Byte[] riff =
203         System.Text.Encoding.UTF8.GetBytes("RIFF");
204         riff.CopyTo(bytesData, 0);

```

```

199     Byte[] chunkSize =
200     ↳ BitConverter.GetBytes(HDR_SIZE + clip.samples *
201     ↳ 2 - 8);
202     ↳ chunkSize.CopyTo(bytesData, 4);

203     Byte[] wave =
204     ↳ System.Text.Encoding.UTF8.GetBytes("WAVE");
205     ↳ wave.CopyTo(bytesData, 8);

206     Byte[] fmt =
207     ↳ System.Text.Encoding.UTF8.GetBytes("fmt ");
208     ↳ fmt.CopyTo(bytesData, 12);

209     Byte[] subChunk1 = BitConverter.GetBytes(16);
210     ↳ subChunk1.CopyTo(bytesData, 16);

211     UInt16 one = 1;

212     Byte[] audioFormat = BitConverter.GetBytes(one);
213     ↳ audioFormat.CopyTo(bytesData, 20);

214     Byte[] numChannels =
215     ↳ BitConverter.GetBytes(channels);
216     ↳ numChannels.CopyTo(bytesData, 22);

217     Byte[] sampleRate = BitConverter.GetBytes(hz);
218     ↳ sampleRate.CopyTo(bytesData, 24);

219     Byte[] byteRate = BitConverter.GetBytes(hz *
220     ↳ channels * 2); // sampleRate *
221     ↳ bytesPerSample*number of channels, here 44100*2*2
222     ↳ byteRate.CopyTo(bytesData, 28);

223     UInt16 blockAlign = (ushort) (channels * 2);
224     ↳ BitConverter.GetBytes(blockAlign).CopyTo(bytesData,
225     ↳ 32);

226     UInt16 bps = 16;
227     Byte[] bitsPerSample = BitConverter.GetBytes(bps);
228     ↳ bitsPerSample.CopyTo(bytesData, 34);

229     Byte[] datastring =
230     ↳ System.Text.Encoding.UTF8.GetBytes("data");
231     ↳ datastring.CopyTo(bytesData, 36);

232     Byte[] subChunk2 = BitConverter.GetBytes(samples *
233     ↳ channels * 2);
234     ↳ subChunk2.CopyTo(bytesData, 40);
235   }

236 }

237 }

238 }

239 }

```

- (b) Add this component to the speech manager. Dis-

- able any other speech recognition components.
- (c) Provide the text field to use to show the resulting text version of the speech as the parameter to this component.
  - (d) Connect the button to the trigger function of this component.
  - (e) Sign up for the free plan at assembly.ai: <https://app.assemblyai.com/login/>. Get the API token and enter that as the subscription key in the SpeechRecognitionAssembly script.

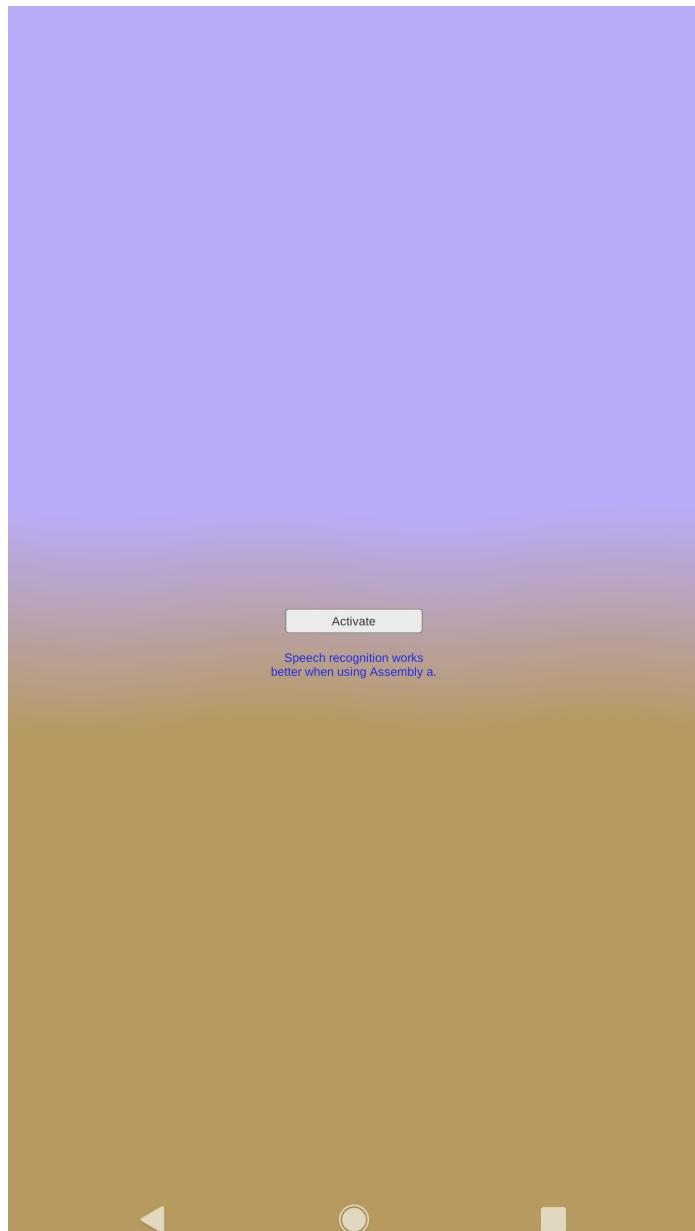


Figure 19.8.2:  
Speech recognition  
achieved using  
the AssemblyAI  
service.

## 20

# *Concepts Explored Components*

## Contents

---

20.1	 Frequency domain signal processing applied to create a heart rate monitor .	475
20.2	 Sensory synesthesia and haptic rendering applied to create a vibration display . . . . .	482

---

20.1  Frequency domain signal processing applied to create a heart rate monitor

### Component 20.1.1: Heart rate monitor.

Heart Rate

Several categories of sensor have been repurposed to extract biomarkers. In particular there are examples of using accelerometers and cameras on existing smartphones to measure heart rate and breathing. This example demonstrates an approach to repurpose the accelerometer in a smartphone as a heart rate monitor.

The principle is straightforward and can be readily adapted to other tasks. A series of samples from the sensor is taken at regular time intervals. This signal contains a combination of all the individual signals that are producing sensor readings. In this case acceleration of the phone results from any movements of the surface that the phone is resting on. If that surface is the chest of a patient then these movements will be the combined

*effects of chest movement from breathing, smaller and more frequent pulses resulting from heart beat, effects of any movement of the limbs of the patient and also any movement of the furniture the patient is sitting on. Separating these signals is difficult when considering the time series but potentially simpler when considering the signal as a combination of signals of different frequencies. Heart beats which occur at about 1-1.5 beats per second are separate from slower breathing motion (less than 1/3 of a breath per second) and also distinct from faster movements of the body.*

*Converting a signal into a frequency based representation is achieved through a process called a Fourier transform. This produces a series of values, representing the amount of each different frequency present in the signal. For computational processes we work with the Fast Fourier Transform function which produces discrete values corresponding to a series of bins representing frequency ranges present in the signal. The bin with the largest value represent the dominant frequency present in the signal.*

*This example extracts the peak in the heart rate range of frequencies from accelerometer data. The same process can be applied to any other sensor that might have a particular signal embedded in it. The heart rate is quite a small signal, so accuracy may be limited if other signals also exist in the targeted range.*

1. Start with a new project. Set up for mobile deployment.

Create an empty object named HeartRate, and attach a new C# script called HeartRate to this. The code for this script is provided in Algorithm 20.1.

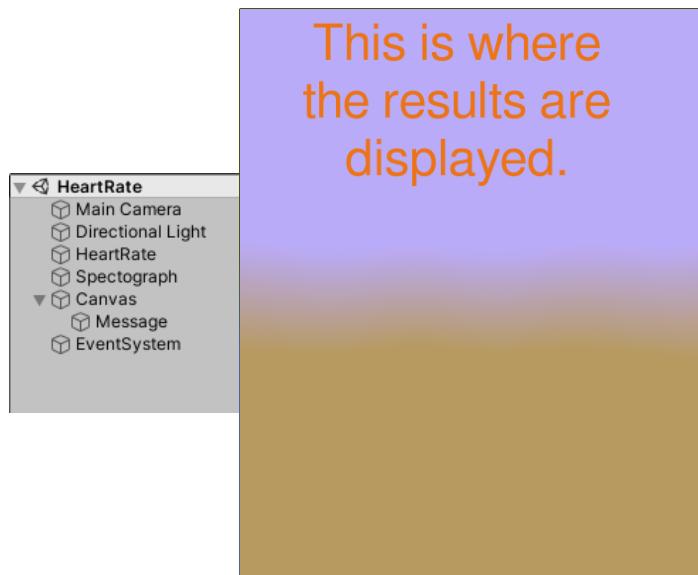
Code Listing 20.1: HeartRate. Heart rate sensing from accelerometer values is just one of the applications of a component that extracts peak frequencies within a defined range.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5

```

Figure 20.1.1: Measuring heart rate with an accelerometer.



```

6   public class HeartRate : MonoBehaviour {
7
8     [Tooltip ("An object to act as parent for the spectrum
9     → chart")]
10    public GameObject spectograph;
11    [Tooltip ("A template object for a bar in the spectrum
12    → chart")]
13    public GameObject barTemplate;
14
15    [Tooltip ("A text object where data is output")]
16    public Text message;
17
18    // The number of sensor readings to use for analysis.
19    // Larger
20    // numbers allow smaller frequency intervals but slow
21    // down
22    // processing and response times.
23    private int maxReadings = 256;
24
25    // A list of the last maxReadings sensor values.
26    private List <double> accelerometerReadings = new List
27    → <double> ();
28
29    // The width of the spectrum chart.
30    private float spectroScale = 4.0f;
31    // An offset, to center, the spectrum chart.
32    private float spectroOffset = -2.0f;
33    // Number of adjacent bins in the spectrum that are
34    → collapsed in
35    // a single bar. Speeds up display of the chart.
36    private int spectroStep = 4;

```

```

31
32 // An smoothed value for the peak measured at each frame.
33 private float smoothedPeak = 0.0f;
34 // The proportion of the smoothed value retained on each
35 ← frame.
36
37 private float smoothingFactor = 0.98f;
38
39
40 // When searching for peak amplitudes, restrict to
41 // frequencies in this range. These ones are tuned
42 // to likely heart rate values (in beats per second).
43 private float minFreq = 0.50f;
44 private float maxFreq = 3.20f;
45
46 // Use a fixed update to get a regular sample of the
47 ← sensor.
48 void FixedUpdate () {
49
50     // Sample the sensor.
51     Vector3 acc = Input.acceleration;
52     accelerometerReadings.Add (acc.magnitude);
53
54     // Add new readings to the list, and discard excess at
55 ← the start.
56     while (accelerometerReadings.Count > maxReadings)
57     {
58         accelerometerReadings.RemoveAt (0);
59     }
60
61     if (accelerometerReadings.Count == maxReadings)
62     {
63         // Prepare for the fast fourier transform. Convert
64 ← all
65         // real sensor readings to complex numbers.
66         double [] readings = accelerometerReadings.ToArray
67         ();
68         double [] readingsComplex = new double [maxReadings];
69         for (int i = 0; i < maxReadings; i++)
70         {
71             readingsComplex[i] = 0.0f;
72         }
73         int m = (int) (Mathf.Log (maxReadings) / Mathf.Log
74 ← (2.0f));
75         // Transform sensor data into frequency values.
76         FFTLibrary.Complex.FFT (1, m, readings,
77         readingsComplex);
78
79         // The sensor sample rate.
80         float baseFreq = 1.0f / Time.deltaTime;
81         // The frequency range occupied by each bin.
82         float binFreq = baseFreq / maxReadings;
83
84         // Clear the spectrum chart.
85         if (spectograph != null)
86     }

```

```

77     {
78         foreach (Transform child in spectograph.transform)
79         {
80             GameObject.Destroy(child.gameObject);
81         }
82     }
83
84     // Find the peak amplitude.
85     float maxLevel = 0.0f;
86     bool maxFound = false;
87     float peakFreq = 0.0f;
88
89     // Element 0: DC (constant) level.
90     for (int i = 1; i < maxReadings / 2; i +=
91         spectroStep)
92     {
93         float total = 0.0f;
94         for (int j = 0; j < spectroStep; j++)
95         {
96             int index = i + j;
97             // Get magnitude of complex number (squared).
98             float v = 1000.0f * (float) (readings[index] *
99             readings[index] + readingsComplex[index] *
100            readingsComplex[index]);
101            float freq = index * binFreq;
102            if ((freq >= minFreq) && (freq <= maxFreq))
103            {
104                if ((!maxFound) || (maxLevel < v))
105                {
106                    maxLevel = v;
107                    maxFound = true;
108                    peakFreq = freq;
109                }
110            }
111
112            total += v;
113        }
114
115        // Create one bar every spectroStep bins.
116        GameObject g = GameObject.Instantiate(barTemplate);
117        g.transform.position = new Vector3(spectroScale *
118        (float) i / (float) (maxReadings / 2) + spectroOffset,
119        0, 0);
120        g.transform.localScale = new Vector3(spectroScale
121        / (float) (maxReadings / (2 * spectroStep)), 0.1f +
122        total, 1.0f / (float) (maxReadings / (2 *
123        spectroStep)));
124        g.transform.SetParent(spectograph.transform);
125    }
126
127    // Smooth peak frequency so not too jumpy.
128    smoothedPeak = smoothingFactor * smoothedPeak +
129    (1.0f - smoothingFactor) * peakFreq;

```

```

121
122     if (message != null)
123     {
124         message.text = "Base freq: " + baseFreq + "Hz" +
125             "\n" +
126             "Bin freq: " + binFreq + "Hz" +
127             "\n" +
128             "Peak freq: " + smoothedPeak + "Hz" +
129             " = " + (smoothedPeak * 60) + "bpm";
130     }
131     else
132     {
133         if (message != null)
134         {
135             message.text = "Starting: " + (100.0f *
136             accelerometerReadings.Count / maxReadings) + "%";
137         }
     }
}

```

This script provides a visual representation of the frequency spectrum by placing a number of bar objects together, and modifying their heights to match that of the frequency bins. This functionality needs a new empty object to act as the parent for the bar objects, and a prefab to represent the bar. Any cube or capsule like object is adequate for the shape of the bar.

The script also displays the frequency of the bin with the highest value. This requires a UI/Text object to be added to the scene, and provided as a property of this component.

2. The Fast Fourier Transform operation is provided by an external library. There are many versions of this operation available, but the one used is sourced unchanged from: <https://code.msdn.microsoft.com/FFTLibrary-e1942867>.
3. The processing of sensing starts with collecting values from the sensor. In this case the vector of accelerations from the accelerometer is converted into a single scalar value by taking the magnitude of the vector. This makes the process tolerant of placing the phone on the

body in any orientation, but does lose some potential filtering that might have resulted from placing the device so that the heart beat was aligned with one of the accelerometer axes.

The Fast Fourier Transform requires a series of samples from the sensor to determine the frequencies present. The length of this sequence determines how finely the individual bins can resolve frequency values. In this case, taking samples 50 times per second, and using a sequence length of 256 samples provides sufficient accuracy for our purposes. Many Fast Fourier Transform functions prefer to have their input in lengths that are a power of 2.

---

```

1 Vector3 acc = Input.acceleration;
2 accelerometerReadings.Add (acc.magnitude);
3 while (accelerometerReadings.Count > maxReadings)
4 {
5     accelerometerReadings.RemoveAt (0);
6 }
```

---

4. The Fast Fourier Transform works with complex numbers (values with both a real and imaginary component) which can also be interpreted as an amplitude and phase value when considering frequency representations. The samples taken are all real (so have no imaginary component), but the frequency values after transformation have non-zero real and imaginary elements.

We need only the magnitude (amplitude) value, so square each component and add these together. Strictly speaking we should then take the square root, but since we are only looking for the biggest value we can skip this step.

5. The lowest frequency element represents the constant (zero frequency) component and is both large but also not interesting. We generally ignore this in most cases.

A search over the set of amplitude values allows us to identify the peak amplitude. The bin in which this

occurs is our frequency of interest. We can further restrict this by only considering bins that fall within the accepted limits for heart rates.

```
1 float freq = index * binFreq;
2 if ((freq >= minFreq) && (freq <= maxFreq))
3 ...
```

---

6. The resulting peak frequency can still be quite erratic. We can smooth this with a moving average. The reported peak frequency becomes a combination of the previous peak frequency and the latest reading. Adjusting the weights with which old and new values are combined allows choice between an erratic but responsive value, or a stable but slow to adapt value.

```
1 smoothedPeak = smoothingFactor * smoothedPeak + (1.0
f - smoothingFactor) * peakFreq;
```

---

7. Experiment with the application and the values used in it. Things to try:
  - (a) Use a different sensor as the source of the values.
  - (b) Try to detect other signals, such as breathing.
  - (c) Verify the accuracy of the reading by shaking the device at a known rate (such as every time the second hand moves on a clock).

## 20.2 Sensory synesthesia and haptic rendering applied to create a vibration display

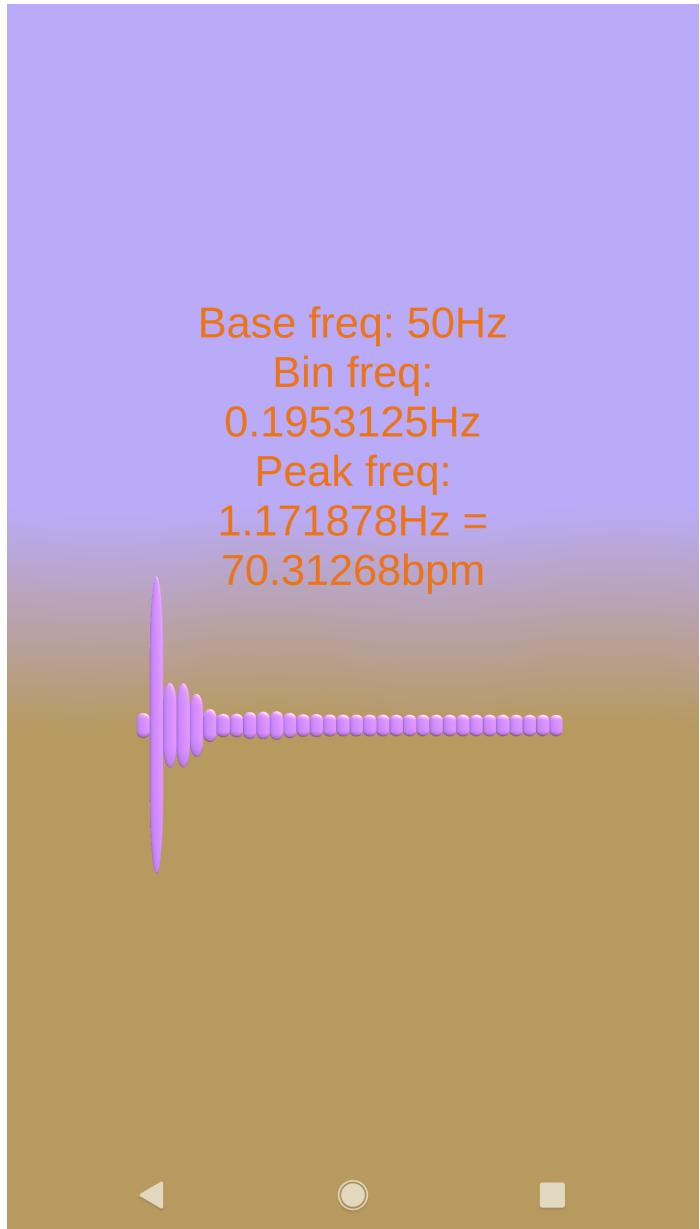
### Component 20.2.1: Vibration display.

Vibration
Display

*The haptic response available on most commodity mobile devices is fairly limited, and often restricted to just a vibration device. Older devices will generally just have a vibration motor which spins an unbalanced mass to create the vibration effect. Control is usually limited to the length (duration) of the vibration with little to no way to influence the strength of*

Figure 20.1.2: Heart rate measurements.

This component uses exaggerated movements to show the frequency domain representation.



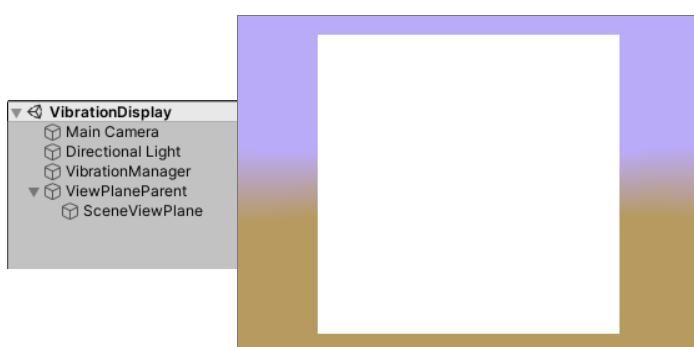
*the vibration or its frequency. Newer devices may use linear actuators which just move the mass back and forth in a single direction. These may offer more control of the nature of the vibration to better emulate the tactile sensations to match the touch experience associated with physical interactions.*

*This example demonstrates use of the vibration facility on mobile devices. It does so in the context of a modality transformation; the opportunity to sense one modality and convert it to another. This concept is useful for granting super-powers by enhancing the human senses. The device can sense modalities that a human cannot (such as magnetic fields, or network signal strengths) and convert them to sensations that humans can perceive (such as visual or tactile). In this case we translate a visual modality to a tactile one which can both augment existing senses but also offer support where an existing sense may be impaired.*

*This example supplements the human visual sense by capturing an image with the camera, processing it to identify significant features and allows the person holding the mobile device to feel the sensations directly through the phone. Moving the phone allows the surrounding area to be perceived through the haptic response.*

*Access to the vibration device using Unity software is currently limited. This example also demonstrates how some of the native Android functionality can be invoked to provide greater control.*

Figure 20.2.1:  
Visual to haptic  
translation.



1. Start with a new project. Set this up to export to the mobile device.

Create a plane to display the image from the physical camera. It is also worth creating a material for this plane since the texture of the material is manipulated by the scripts used, to show the camera image. This display is not required for the application to run, but does help show off the image processing stages that are used. Rotation settings of 90, 180, 0 make sure the plane is facing the camera and oriented appropriately.

We need to ensure that the image remains correctly oriented on the screen, even when the phone is rotated. In those cases, the physical camera is rotated but the display produced by the virtual scene camera is corrected. Add an empty parent object to the plane, and attach the script shown in Algorithm 20.2 to this empty object to ensure that the physical camera image aligns with the virtual camera. This script requires access to the image from the physical camera object which is provided by a component created in a later step (the CameraView component starting in Algorithm 20.4). Once this component is created it needs to be provided as a property to this camera orientation script.

Code Listing 20.2: `PhysicalCameraOrientation`. Orientation of an object facing the scene camera is controlled by this script so that the texture provided by a physical camera is oriented correctly.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;

4
5  public class PhysicalCameraOrientation : MonoBehaviour {
6      public CameraView camView;
7      void Update () {
8          transform.rotation = Quaternion.AngleAxis
9              (-camView.getWebcamTex ().videoRotationAngle,
10              Vector3.forward);
11      }
12 }
```

2. Vibration access under the Unity software is limited to a single vibration of fixed duration and amplitude. This can be demonstrated when the application starts by

attaching the script shown below to one of the objects in the scene.

```

1  public class VibrationControl : MonoBehaviour
2  {
3      void Start()
4      {
5          Handheld.Vibrate(); // Force vibration
6              permission.
7      }

```

---

This has the side-effect of also ensuring that the Unity software includes a request for permission to use the vibration motor in the application that is built. Without this requests to actuate the vibrator are silently ignored.

3. Access to the vibration effects requires that native Android libraries be invoked. Unlike the native code calls in component 18.5.1 the vibration facilities are already part of the existing java libraries available to mobile applications and so this access does not require special external libraries to be built and added. Instead some very Android device specific functions can be directly invoked from the C# scripts.

The scripts used are based on code snippets provided by Arturo Volpe and Andy Bacon. The vibrator service is accessed through `AndroidJavaObject` variables initialized as follows:

```

1  public static AndroidJavaClass unityPlayer = new
   AndroidJavaClass("com.unity3d.player.UnityPlayer
   ");
2  public static AndroidJavaObject currentActivity =
   unityPlayer.GetStatic<AndroidJavaObject>(""
   currentActivity");
3  public static AndroidJavaObject vibrator =
   currentActivity.Call<AndroidJavaObject>(""
   getSystemService", "vibrator");

```

---

Vibrations can then be triggered using:

```

1  vibrator.Call("vibrate", milliseconds);

```

---

Newer versions of Android allow both duration and amplitude of vibrations to be specified.

The full C# script to access the vibration motor is provided in Algorithm 20.3. This script does not need to be associated with any object in the scene as it only consists of functions that are invoked by other scripts.

**Code Listing 20.3: VibrationControl.** Vibration motor access is achieved through directly accessing the Java classes in Android that provide this service. Two version of the Vibration class are provided which support older and newer versions of Android respectively.

```

38     if (isAndroid())
39         vibrator.Call("vibrate", milliseconds);
40     else
41         Handheld.Vibrate();
42     }
43
44     public static void Vibrate(long[] pattern, int repeat)
45     {
46         if (isAndroid())
47             vibrator.Call("vibrate", pattern, repeat);
48         else
49             Handheld.Vibrate();
50     }
51
52     public static bool HasVibrator()
53     {
54         return isAndroid();
55     }
56
57     public static void Cancel()
58     {
59         if (isAndroid())
60             vibrator.Call("cancel");
61     }
62
63     private static bool isAndroid()
64     {
65         #if UNITY_ANDROID && !UNITY_EDITOR
66         return true;
67         #else
68         return false;
69         #endif
70     }
71
72 // Based on: https://gist.github.com/playfulbacon/4ff08fcdf7ab0c023118874f5339bf7a
73
74     public static class Vibration2
75     {
76         #if UNITY_ANDROID && !UNITY_EDITOR
77         public static AndroidJavaClass unityPlayer = new
78             AndroidJavaClass("com.unity3d.player.UnityPlayer");
79         public static AndroidJavaObject currentActivity = unityP
80             layer.GetStatic<AndroidJavaObject>("currentActivity");
81         public static AndroidJavaObject vibrator = currentActivi
82             ty.Call<AndroidJavaObject>("getSystemService",
83             "vibrator");
84         public static AndroidJavaClass vibrationEffectClass =
85             new AndroidJavaClass("android.os.VibrationEffect");
86         public static int defaultAmplitude = vibrationEffectClas
87             s.GetStatic<int>("DEFAULT_AMPLITUDE");
88         public static AndroidJavaClass androidVersion = new
89             AndroidJavaClass("android.os.Build$VERSION");

```

```
83     public static int apiLevel =
84         androidVersion.GetStatic<int>("SDK_INT");
85     #else
86     public static AndroidJavaClass unityPlayer;
87     public static AndroidJavaObject vibrator;
88     public static AndroidJavaObject currentActivity;
89     public static AndroidJavaClass vibrationEffectClass;
90     public static int defaultAmplitude;
91     #endiff
92
92     public static void Vibrate(long milliseconds, int
93         → amplitude = -1)
94     {
95         if (amplitude < 0)
96         {
97             amplitude = defaultAmplitude;
98         }
99         CreateOneShot(milliseconds, amplitude);
100    }
101
101    public static void CreateOneShot(long milliseconds, int
102        → amplitude)
103    {
104        CreateVibrationEffect("createOneShot", new object[] {
105            → milliseconds, amplitude });
106    }
107
106    public static void CreateWaveform(long[] timings, int
107        → repeat)
108    {
109        CreateVibrationEffect("createWaveform", new object[] {
110            → timings, repeat });
111    }
111
111    public static void CreateWaveform(long[] timings, int[]
112        → amplitudes, int repeat)
113    {
114        CreateVibrationEffect("createWaveform", new object[] {
115            → timings, amplitudes, repeat });
116    }
116
116    public static void CreateVibrationEffect(string
117        → function, params object[] args)
118    {
119        if (isAndroid() && HasAmplitudeControl())
120        {
121            AndroidJavaObject vibrationEffect = vibrationEffectC
122            lass.CallStatic<AndroidJavaObject>(function,
123            args);
124            vibrator.Call("vibrate", vibrationEffect);
125        }
126        else
127            Handheld.Vibrate();
```

```

125     }
126
127     public static bool HasVibrator()
128     {
129         return vibrator.Call<bool>("hasVibrator");
130     }
131
132     public static bool HasAmplitudeControl()
133     {
134         #if UNITY_ANDROID && !UNITY_EDITOR
135         if (apiLevel >= 26)
136             return vibrator.Call<bool>("hasAmplitudeControl");
137         // API 26+ specific
138         else
139             return false; // no amplitude control below API
140         level 26
141         #else
142             return false;
143         #endif
144     }
145
146     public static void Cancel()
147     {
148         if (isAndroid())
149             vibrator.Call("cancel");
150     }
151
152     private static bool isAndroid()
153     {
154         #if UNITY_ANDROID && !UNITY_EDITOR
155         return true;
156         #else
157         return false;
158         #endif
159     }

```

4. The remaining step is to process the camera image into a form where the vibration output provides an adequate representation of the details in the image. The colour or brightness levels in the image could be used but these often still require significant mental processing to make sense of the scene. Given that the vibration represents only a single degree of freedom (maybe two if both duration and amplitude can be controlled) a level of feature extraction is called for.
- Edges in the image are features that would translate

into a haptic response, since we are used to feeling the physical response to touching a physical boundary. Edges can be extracted from the image by applying a particular filter to the image. In this case we use an approximated Laplacian filter applied to each image.

The filter looks like:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

The center value (8) is multiplied by the brightness of the current pixel, and the remaining (-1) values are multiplied by the brightness of the neighbouring pixels in the corresponding positions. When the image is uniform (all pixels have the same brightness) then the filter returns a value of 0. When the brightness of neighbouring pixels differ from that of the center pixel, then the filter returns a value that is different from 0, and the magnitude of this value indicates the sharpness of the edge in the image.

If we apply this filter to every pixel in the image then we can identify where edges are. Setting the level of the vibration to the strength of the edge provides a natural mapping between the visual stimulus and the corresponding haptic response.

5. In practice we make some concessions to performance. Filtering high resolution images is a time consuming task. This could be done with a shader using the graphics processing unit, but these are not present on all mobile devices. We don't actually need to perform edge detection on the full resolution image since this involves more detail than the vibration motor is likely to be able to represent.

The image needs to be reduced in resolution before edge detection. Again Unity software lacks much support for this. Conveniently though the texture representation includes mip-maps (lower resolution versions of

the texture that are used when the texture only covers a few pixels). Each mip-map level is half the size of the image above it. The top most level is the full size image. These can be accessed to immediately retrieve low resolution versions of the image.

Thus the steps in the edge detection process are:

- (a) Reduce resolution of the image by retrieving a low resolution mip-map image.

---

```

1  reduceTex.SetPixels(sourceCopyTex.GetPixels (
      mipLevel));
2  reduceTex.Apply();

```

---

- (b) Apply the edge detection filter, using the convolution operation which applies it to every pixel in the image.

---

```

1  TextureFilter.Convolution(reduceTex, edgeTex,
      TextureFilter.EDGEDETECT_KERNEL_3, 1);

```

---

- (c) Convert the edges in each of the red, green and blue brightness channels to a single intensity value for each pixel.

---

```

1  for (int k = 0; k < cols.Length; k++)
2      float v = (cols[k].r + cols[k].g + cols[k].b) /
            3;

```

---

6. Having established a level of vibration with each of the pixels we can now focus on determining the mode of interaction that best suits the application.

One approach might be to hold the device in a fixed position, and run a finger over the screen to feel the image. This could be useful as a form of picture or text reader, or for reading printed but not embossed Braille. The finger touch position is translated into a screen position, connected to a position on the plane displaying the camera image by firing a ray from the scene camera viewpoint through the screen until it strikes the plane, and reading the edge intensity value of the texture on the plane.

---

```

1 Vector3 pos = Input.mousePosition;
2 Ray ray = Camera.main.ScreenPointToRay(pos);
3 RaycastHit hit;
4 if (Physics.Raycast (ray, out hit)) {
5     float edgeStrength = edgeTex.
6         GetPixelBilinear (hit.textureCoord.x, hit.
7             textureCoord.y).r;
8     Vibration.Vibrate ((int) (edgeStrength * 10.0f));
9 }
```

---

Alternatively we can use physical movement of the device to scan out the scene (remembering that the physical camera is attached to the device and thus follows its movements automatically - no gyroscope required). The center point in the image then determines the level of vibration. In this case the device acts as a virtual cane allowing the scene to be scanned for obstacles.

---

```

1 float edgeStrength = edgeTex.GetPixelBilinear (0.5f,
2                                         0.5f).r;
3 Vibration2.Vibrate ((int) (Time.deltaTime * 1000), (
4                                         int) (edgeStrength * 100.0f));
```

---

The full script for performing edge detection is provided in Algorithm 20.4.. The filter code is based on material provided at: <http://wiki.unity3d.com/index.php/TextureFilter>.

This script should be attached to the plane object created in a previous step. It ensures that the video feed from the physical camera is also visualized by setting this as the texture for the plane.

Code Listing 20.4: CameraView. Edge detection and haptic response translate from a visual modality to one that can be appreciated from the tactile sensations.

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class CameraView : MonoBehaviour
6 {
7     // Copies of the textures.
8     private WebCamTexture webcamTex = null;
9     private Texture2D edgeTex;
10    private Texture2D sourceCopyTex;
11    private Texture2D reduceTex;
```

```
12
13     private int mipLevel = 5;
14
15     // Accessor methods.
16     public Texture getCameraImage()
17     {
18         return webcamTex;
19     }
20
21     public Texture2D getEdgeImage()
22     {
23         return edgeTex;
24     }
25
26     public WebCamTexture getWebcamTex()
27     {
28         if (webcamTex == null)
29         {
30             webcamTex = new WebCamTexture();
31             webcamTex.Play();
32         }
33
34         return webcamTex;
35     }
36
37     // Initialization of the textures.
38     void Start()
39     {
40         // Start live camera feed.
41         webcamTex = getWebcamTex();
42
43         // Set the sizes of the other textures based on the
44         // camera reference.
45         sourceCopyTex = new Texture2D(webcamTex.width,
46         ←   webcamTex.height);
47         reduceTex = new Texture2D(webcamTex.width >> mipLevel,
48         ←   webcamTex.height >> mipLevel);
49         edgeTex = new Texture2D (reduceTex.width,
50         ←   reduceTex.height);
51
52         GetComponent<MeshRenderer>().sharedMaterial.mainTexture =
53         ←   e =
54         ←   edgeTex;
55
56         // Update buffers asynchronously and partially to
57         // minimize run time overhead.
58         StartCoroutine(updateEdgeImage());
59     }
60
61     void Update ()
62     {
63         // Length of pulse replaces lack of amplitude control.
64     }
65
66 }
```

```

58     //// Mode where running a finger over the screen
59     → provides some feel for the
60     //// edges in the image.
61     //Vector3 pos = Input.mousePosition;
62
63     //Ray ray = Camera.main.ScreenPointToRay(pos);
64
65     //RaycastHit hit;
66     //float edgeStrength;
67     //if (Physics.Raycast (ray, out hit))
68     //{
69         // edgeStrength = edgeTex.GetPixelBilinear
70         → (hit.textureCoord.x, hit.textureCoord.y).r;
71         // Vibration.Vibrate ((int) (edgeStrength * 10.0f));
72         //}
73
74     // Mode where direction camera is aimed controls edge
75     → related vibration.
76     float edgeStrength;
77     edgeStrength = edgeTex.GetPixelBilinear (0.5f, 0.5f).r;
78         // Vibration.Vibrate ((int) (edgeStrength *
79         → 10.0f));
80     Vibration2.Vibrate ((int) (Time.deltaTime * 1000),
81         → (int) (edgeStrength * 100.0f));
82   }
83
84   // Extract the edge image. Do it block by block with
85   → yield after
86   // each block to avoid significant performance impacts.
87   private IEnumerator updateEdgeImage()
88   {
89     while (true)
90     {
91       sourceCopyTex.SetPixels(webcamTex.GetPixels ());
92       sourceCopyTex.Apply ();
93       reduceTex.SetPixels(sourceCopyTex.GetPixels
94       (mipLevel));
95       reduceTex.Apply ();
96
97       // edge detect.
98       TextureFilter.Convolution(reduceTex, edgeTex,
99           → TextureFilter.EDGEDETECT_KERNEL_3, 1);
100      Color[] cols = edgeTex.GetPixels();
101      // convert to grayscale.
102      for (int k = 0; k < cols.Length; k++)
103      {
104        float v = (cols[k].r + cols[k].g + cols[k].b) / 3;
105        cols[k].r = v;
106        cols[k].g = v;
107        cols[k].b = v;
108      }
109
110      edgeTex.SetPixels (cols);

```

```
103     edgeTex.Apply ();
104
105     GetComponent<MeshRenderer>().sharedMaterial.mainText =
106     ↪ ure =
107     ↪ edgeTex;
108     ↪     yield return new WaitForSeconds(0.01f);
109 }
```

7. Further enhancements could allow the phone to be held in front of the user, and a vertical slice of the image scanned out to the vibration actuator. This would save having to constantly move the device to check for moving obstacles. Alternatively employ some of the feature point tracking facilities explored in earlier examples to map out the distance to obstacles. This could then be provided using another modality: sound, in the form of a sonar like ping response.

Figure 20.2.2:  
Visual edges can be  
felt by translating  
them into vibration.



## **21**

# *Participant Engagement Components*

## **Contents**

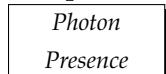
---

<b>21.1</b>	 Achieving mutual presence for multi-participant experiences by using Photon	498
<b>21.2</b>	 Coordinating multiple worlds and groups of participants using lobbies and rooms	507
<b>21.3</b>	 Allowing participants to talk to one another using voice communication . . .	526
<b>21.4</b>	 Creating a multi-participant experience with video communication using web technologies . . . . .	532
<b>21.5</b>	 Customizing the data shared in a multi-participant experience . . . . .	549
<b>21.6</b>	 Enabling persistent augmented reality experiences and shared location recognition using cloud anchors . . . . .	570
<b>21.7</b>	 Legacy multi-participant collaboration using the now obsolete UNet . . .	583

---

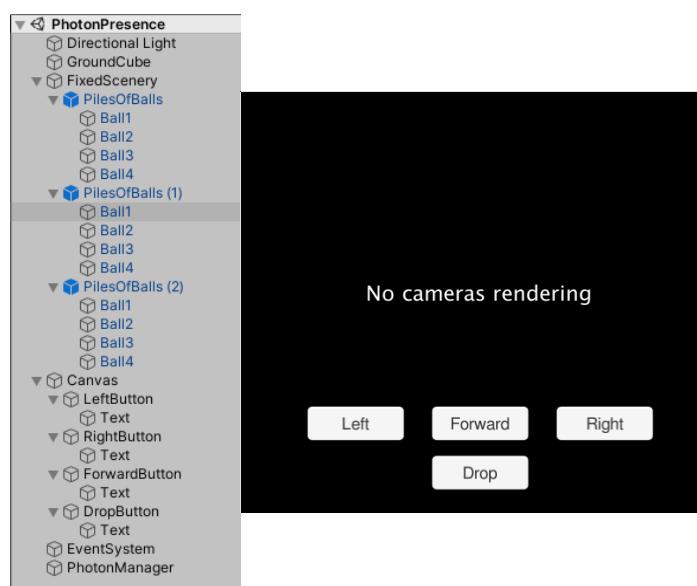
## 21.1 Achieving mutual presence for multi-participant experiences by using Photon

### Component 21.1.1: Mutual presence with Photon.



This example focuses on achieving mutual presence with minimal effort using the facilities provided by Photon. A more robust application would take more care with the processes involved in getting each participant to connect to the shared experience. These matchmaking steps are separated out and described in section 3.4.

Figure 21.1.1: A shared environment for mutual presence.



1. Start with a new project. Set up a minimal scene consisting of at least a ground plane, and any fixed scenery.  
Set the project up to deploy to a mobile device.
2. The next step involves creating an avatar to represent each participant in the space. Create a new empty object in the scene to represent this avatar. For convenience this avatar parent object is placed at ground

level. Add an appropriate mesh, or series of primitive objects, to represent the body of the avatar. For a first person perspective, move the camera to become a child of the avatar's head.

Since we want this example template to be as general as possible, the controls to manipulate the avatar are based on screen based controls. These should support desktop, touch screen and non-VR mode applications in virtual reality headsets which should be sufficient to test the networking functionality prior to building more appropriate interaction tools.

Create a series of buttons, using the UI/Button option. These can be labelled according to their planned function: (turn) left, (move) forward, (turn) right and drop (block). Once these are laid out as desired, set the UI Scale Mode in the Canvas Scaler component of the Canvas object to 'Scale with Screen Size' so that this interface layout retains its size, shape and position regardless of the device it is deployed to.

The script to manipulate the avatar is provided in Algorithm 21.1. Attach this to the avatar (the empty object that is the root of the avatar object's hierarchy). While we could use the OnClick event of each button to drive the movement, this requires a complete click action for every step taken. To get continuous movement while the button is pressed, add an EventTrigger component to each of the buttons. Add a PointerDown event that connects to each of the movement functions provided, and a PointerUp event that connects to the Stop function. The drop function also requires a prefab for the object it creates and drops. A standard cube, converted into a prefab, can be used for this purpose. Including a Rigidbody component on the cube allows it to be managed by the physics simulation in the application.

Code Listing 21.1: AvatarActions. Button based controls for manipulating the avatar.

```
1  using System.Collections;
```

```
2   using System.Collections.Generic;
3   using UnityEngine;
4   using UnityEngine.Events;
5   using Photon.Pun;
6
7   public class AvatarActions : MonoBehaviourPun
8   {
9     [Tooltip ("The prefab for objects created when the drop
10    ↵ event is triggered.")]
11    public GameObject blockTemplate;
12
13    [Tooltip ("Turn speed in degrees per second")]
14    public float turnSpeed = 100.0f;
15
16    [Tooltip ("Movement speed in meters per second (assumes
17    ↵ 1 unit = 1 meter)")]
18    public float moveSpeed = 10.0f;
19
20    private float turn = 0.0f;
21
22    private float move = 0.0f;
23
24    private void setButtonCallbacks ()
25    {
26      // Assumes each button has already been set with two
27      ↵ triggers; a pointer down followed by a pointer up.
28      GameObject.Find ("Canvas/LeftButton").GetComponent
29      <EventTrigger> ().triggers[0].callback.AddListener
30      ((data) => { Left (); });
31      GameObject.Find ("Canvas/LeftButton").GetComponent
32      <EventTrigger> ().triggers[1].callback.AddListener
33      ((data) => { Stop (); });
34      GameObject.Find ("Canvas/RightButton").GetComponent
35      <EventTrigger> ().triggers[0].callback.AddListener
36      ((data) => { Right (); });
37      GameObject.Find ("Canvas/RightButton").GetComponent
38      <EventTrigger> ().triggers[1].callback.AddListener
39      ((data) => { Stop (); });
40      GameObject.Find ("Canvas/ForwardButton").GetComponent
41      <EventTrigger> ().triggers[0].callback.AddListener
42      ((data) => { Forward (); });
43      GameObject.Find ("Canvas/ForwardButton").GetComponent
44      <EventTrigger> ().triggers[1].callback.AddListener
45      ((data) => { Stop (); });
46      GameObject.Find ("Canvas/DropButton").GetComponent
47      <EventTrigger> ().triggers[0].callback.AddListener
48      ((data) => { Drop (); });
49      GameObject.Find ("Canvas/DropButton").GetComponent
50      <EventTrigger> ().triggers[1].callback.AddListener
51      ((data) => { Stop (); });
52    }
53
54    public void Start ()
```

```
36
37     {
38         if (photonView.IsMine == true ||
39             PhotonNetwork.IsConnected == false)
40         {
41             setButtonCallbacks ();
42         }
43         else
44         {
45             transform.Find ("Head/Camera").gameObject.SetActive
46             (false);
47         }
48     }
49
50     public void Update ()
51     {
52         if (photonView.IsMine == true ||
53             PhotonNetwork.IsConnected == false)
54         {
55             transform.rotation *= Quaternion.AngleAxis (turn *
56             turnSpeed * Time.deltaTime, Vector3.up);
57             transform.position += move * moveSpeed *
58             transform.forward;
59         }
60     }
61
62     private void dropObject ()
63     {
64         // Create the new object up and to the front, so it is
65         // away from the avatar.
66         GameObject o = PhotonNetwork.Instantiate
67         (blockTemplate.name, transform.position +
68         transform.forward + transform.up, Quaternion.identity);
69     }
70
71     public void Left ()
72     {
73         turn = -1.0f;
74     }
75     public void Right ()
76     {
77         turn = 1.0f;
78     }
79     public void Forward ()
80     {
81         move = 1.0f;
82     }
83     public void Drop ()
84     {
85         dropObject ();
86     }
87     public void Stop ()
88     {
89         turn = 0.0f;
90     }
```

```
81     move = 0.0f;  
82 }  
83 }
```

3. Once the single participant process is working, it is time to include the networking functionality. This involves two steps; adding the functionality to the project, and setting up an account to make use of the cloud based server that relays messages.

Create an account to access the Photon services at <https://dashboard.photonengine.com/en-US/account/SignUp>. Once the account is created, use the web site to create a new app, corresponding to this project. The project type is 'Photon PUN' and the name can correspond to the name you provided for the project. At time of writing, free projects have a limit of 20 concurrent users connected to the server at any one time. Each project has an AppID, typically a sequence of 32 hexadecimal digits.

Download and import the 'PUN 2 Free' package from the asset store. During the final stages of the import process, the setup will request the AppID generated on the Photon web site. Enter this into the field provided. This value is written into the PhotonServerSettings asset in the Photon/PhotonUnityNetworking/Resources folder in case it ever needs to be changed in the future.

The PhotonServerSettings also include a 'PUN Logging' field. Setting this to log all messages can be helpful when monitoring the progress and status of connections.

4. The process of adding new participants is rudimentary in this example. A single shared multi-participant environment is represented as a room in Photon terminology. Normally identifying an appropriate room, finding your friends and joining the room would be a more extensive process, as outlined in section 3.4. In this case, we use a single room with a name known to

all participants, and all hosted on the same server in the same region.

Create a new empty object called PhotonManager in the scene and add a new C# script, also called PhotonManager. The full program code for this is available in Algorithm 21.2. The sequence of operations is as follows:

Code Listing 21.2: PhotonManager. Minimal participant connection process for a Photon application.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  using Photon.Pun;
6  using Photon.Realtime;
7
8  public class PhotonManager : MonoBehaviourPunCallbacks
9 {
10
11     public GameObject avatarPrefab;
12
13     void Start()
14     {
15         Debug.Log ("Connected: " + PhotonNetwork.IsConnected);
16         PhotonNetwork.ConnectUsingSettings();
17     }
18
19     public override void OnConnectedToMaster()
20     {
21         Debug.Log ("Connected to master.");
22         RoomOptions roomopt = new RoomOptions ();
23         PhotonNetwork.JoinOrCreateRoom ("ApplicationRoom",
24             roomopt, new TypedLobby ("ApplicationLobby",
25             LobbyType.Default));
26     }
27
28     public override void OnJoinedRoom()
29     {
30         Debug.Log ("Joined room with " +
31             PhotonNetwork.CurrentRoom.PlayerCount + "
32             participants.");
33
34         PhotonNetwork.Instantiate (avatarPrefab.name, new
35             Vector3 (), Quaternion.identity, 0);
36     }
37 }
```

To ensure that all participants connect to the same

server, set the Fixed Region field in the PhotonServerSettings to a single region.

The ConnectUsingSettings method then uses the values in the PhotonServerSettings to establish a connection to the server in that region. If it is successful then it triggers a callback to the OnConnectedToMaster event handler. Note that the class must derive from MonoBehaviourPunCallbacks in order to be able to override these functions.

Once the connection has been established to the server, then the JoinOrCreateRoom method registers the application as being associated with a specific logical room on that server. A unique identifier for the user is used to allow different connections to that room to be distinguished. The event handler OnJoinedRoom is called once entry into the room is achieved. In the interests of brevity the failure case which calls OnJoinFailed is ignored in this example. The Photon logging facility can be used to detect error cases during development.

Once the room has been joined it is time to spawn an instance of the participant in the room. This is done using the Instantiate function provided by the Photon network. The avatar prefab created previously must be provided as a parameter to the PhotonManager script. This avatar needs to have a PhotonView component added to it, to ensure that it shares its state across the network. Make sure any existing avatar prefabs are removed from the scene. The avatar prefab also needs to be in a Resources folder. It is possible to create such a folder inside your existing Prefabs folder, and move the avatar into that.

The PhotonView ensures that an object has an identifier that allows it to be tracked across the network. The original object and each of its copies on other devices can be matched to one another. Note that this doesn't mean that state is shared between them. This is the

responsibility of other components introduced in later steps.

5. Test out the application and verify that it connects to the room and spawns the avatar. The spawned avatar has lost the connections to the buttons used to control the avatar's actions. These links need to be recreated once the avatar is instantiated. Provided the buttons are appropriately named, the function below can be called from the avatar's Start method to reconnect the links.

```

1  private void setButtonCallbacks ()
2  {
3      // Assumes each button has already been set with
        // two triggers; a pointer down followed by a
        // pointer up.
4      GameObject.Find ("Canvas/LeftButton").
        GetComponent <EventTrigger> ().triggers[0].
        callback.AddListener ((data) => { Left ();
        });
5      GameObject.Find ("Canvas/LeftButton").
        GetComponent <EventTrigger> ().triggers[1].
        callback.AddListener ((data) => { Stop ();
        });
6      GameObject.Find ("Canvas/RightButton").
        GetComponent <EventTrigger> ().triggers[0].
        callback.AddListener ((data) => { Right ();
        });
7      GameObject.Find ("Canvas/RightButton").
        GetComponent <EventTrigger> ().triggers[1].
        callback.AddListener ((data) => { Stop ();
        });
8      GameObject.Find ("Canvas/ForwardButton").
        GetComponent <EventTrigger> ().triggers[0].
        callback.AddListener ((data) => { Forward ();
        });
9      GameObject.Find ("Canvas/ForwardButton").
        GetComponent <EventTrigger> ().triggers[1].
        callback.AddListener ((data) => { Stop ();
        });
10     GameObject.Find ("Canvas/DropButton").
        GetComponent <EventTrigger> ().triggers[0].
        callback.AddListener ((data) => { Drop ();
        });
11     GameObject.Find ("Canvas/DropButton").
        GetComponent <EventTrigger> ().triggers[1].
        callback.AddListener ((data) => { Stop ();
        });
12 }
```

---

6. Deploying and testing the application on separate, network connected devices reveals several issues. These are explicitly noted here since these symptoms may occur in other aspects of applications that you build.

The avatars of other players may be either static or invisible, depending on whether you moved before they joined the session. This is because the input from the controls is being provided to all instances of the avatars on each device. Hence all avatars may end up mirroring the actions from a single device. The solution to this is to check if the avatar is a representation of the local participant by protecting it with the program structure below.

---

```

1  if (photonView.IsMine == true || PhotonNetwork.
   IsConnected == false)
2  {
3 }
```

---

Input on the local device is only acted upon if the avatar is the one associated with the participant holding the device, or if the application is running without a network connection. The component needs to derive from a parent class such as MonoBehaviourPun in order to provide these values.

In this case, there are multiple cameras created in the scene since each avatar contains its own camera. The following code fragment disables the camera component in the non-local avatars.

---

```

1  if (photonView.IsMine == true || PhotonNetwork.
   IsConnected == false)
2  ...
3 else
4 {
5     transform.Find ("Head/Camera").gameObject.
      SetActive (false);
6 }
```

---

7. Even when the controls have been restricted to only one avatar per device, there are still issues with some avatars not seeming to appear. This is because the state

(such as position and orientation) of the avatars are not being updated across the network. The PhotonTransformView component is required for this. Add this component to the avatar. Add the PhotonTransformView component to the list of observed components in the PhotonView component. This then ensures that the state from the transform is updated on other devices.

8. Repeat the same process with the blocks that the participants create. Specifically: move it into the Resources folder, add a PhotonView and PhotonTransformView, set the transform view as an observed component of the PhotonView, and change the Instantiate call to use the PhotonNetwork.Instantiate.

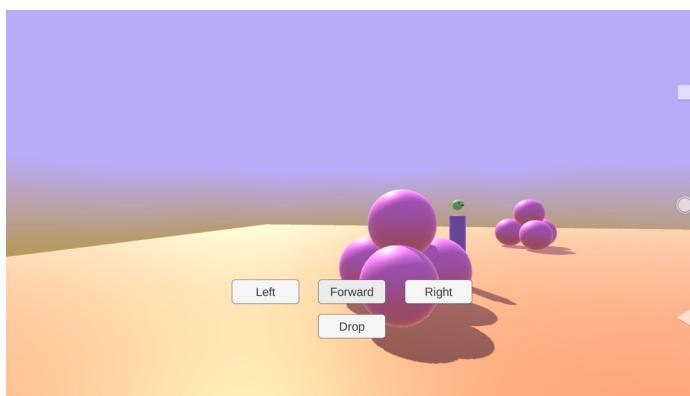


Figure 21.1.2: A second participant sharing the virtual environment.

## 21.2 Coordinating multiple worlds and groups of participants using lobbies and rooms

### Component 21.2.1: A Sense of Distributed Presence

**MeetingRooms**

1. Start with a new project. Configure this for deployment on a mobile device. As described in component 21.1.1, add the Photon Unity Networking (PUN 2) package



Figure 21.2.1: A foyer for selecting rooms.

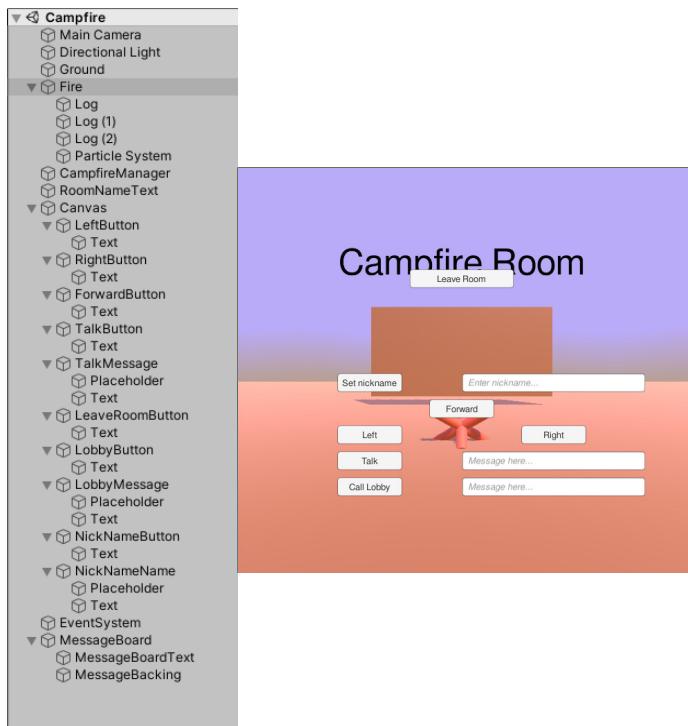


Figure 21.2.2:  
Participants can then join the selected room.

from the Unity store and set that up with an application key for a Photon PUN application retrieved from the Photon dashboard.

2. Create a new empty object and name it RoomManager. Create a new C# script, also named RoomManager, and add it to the RoomManager object in the scene.

The next steps explain the steps required to build the RoomManager script. These will be presented in an order that gradually builds up the required functionality. However, for reference, the final script is provided in Algorithm 21.3.

Code Listing 21.3: RoomManager. The lobby manages the list of rooms, with opportunities to create new rooms and enter existing ones.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using Photon.Pun;
6  using Photon.Realtime;
7  using UnityEngine.SceneManagement;
8
9  public class RoomManager : MonoBehaviourPunCallbacks
10 {
11     public GameObject roomPrefab;
12     public Canvas roomCanvas;
13
14     private bool allowingJoining = false;
15
16     List <GameObject> displayRooms = new List <GameObject>
17     ();
18
19     void Start()
20     {
21         PhotonNetwork.ConnectUsingSettings();
22     }
23
24     public static string getName (GameObject o)
25     {
26         if (o.GetComponent <PhotonView> () != null)
27         {
28             if ((o.GetComponent <PhotonView> () .Owner.NickName
29             != null) && !(o.GetComponent <PhotonView>
30             () .Owner.NickName.Equals ("")))
31             {
32                 return o.GetComponent <PhotonView>
33             () .Owner.NickName;
34         }
35     }

```

```

30     }
31     else
32     {
33         return o.GetComponent <PhotonView> ().Owner.UserId;
34     }
35 }
36 else
37 {
38     // Not a networked object. Just return the current
39     player's id
40     if ((PhotonNetwork.NickName != null) &&
41     !(PhotonNetwork.NickName.Equals ("")))
42     {
43         return "X" + PhotonNetwork.NickName;
44     }
45     else
46     {
47         return "X" + PhotonNetwork.AuthValues.UserId;
48     }
49 }
50
51 // Find the display version of the room, creating one
52 // if none exists.
53 GameObject getRoomObject (string name)
{
54     foreach (GameObject g in displayRooms)
55     {
56         DisplayRoom dr = g.GetComponent <DisplayRoom> ();
57         if (dr.getName () .Equals (name))
58         {
59             return g;
60         }
61     }
62     GameObject room = Instantiate (roomPrefab);
63     room.transform.SetParent (roomCanvas.transform);
64     room.GetComponent<DisplayRoom>().setName(name);
65     room.GetComponent<LocalRoomBehaviour>().setManager(this);
66     displayRooms.Add (room);
67     return room;
68 }
69
70 void removeRoomObject (GameObject room)
{
71     displayRooms.Remove (room);
72     Destroy (room);
73 }
74
75
76 // Lay the room controls out in a grid.
77 void updateRooms ()
{
78     int row = 0;

```

```
80     int col = 0;
81     int columnLimit = 2;
82     foreach (GameObject room in displayRooms)
83     {
84         room.transform.localPosition = new Vector3 (col *
85             300 - 100, row * 100, 0);
86
87         col += 1;
88         if (col >= columnLimit)
89         {
90             col = 0;
91             row -= 1;
92         }
93     }
94
95     public void JoinRoom (string roomName)
96     {
97         allowingJoining = true;
98         PhotonNetwork.JoinRoom(roomName);
99         PhotonNetwork.LoadLevel ("Campfire");
100    }
101
102    public override void OnConnectedToMaster()
103    {
104        Debug.Log ("Connected to master.");
105        PhotonNetwork.JoinLobby ();
106    }
107
108    public override void OnRoomListUpdate
109        (List<RoomInfo> roomList)
110    {
111        foreach (RoomInfo ri in roomList)
112        {
113            GameObject room = getRoomObject (ri.Name);
114
115            if (ri.RemovedFromList)
116            {
117                removeRoomObject (room);
118            }
119            else
120            {
121                room.GetComponent <DisplayRoom> ().display
122                    (ri.Name + "\nwith " + ri.PlayerCount + " players\n"
123                     + ri.CustomProperties["notices"]);
124            }
125            updateRooms ();
126        }
127
128        public override void OnJoinedLobby ()
129        {
130            Debug.Log ("Joined lobby");
```

```

129     }
130
131     public override void OnJoinedRoom ()
132     {
133         Debug.Log ("Room joined");
134
135         // Leave a lobby message with details of who joined.
136         Room r = PhotonNetwork.CurrentRoom;
137         ExitGames.Client.Photon.Hashtable p =
138             r.CustomProperties;
139         p["notices"] = RoomManager.getName (this.gameObject) +
140             ":" + Time.time + ":joined\n";
141         r.SetCustomProperties (p);
142         // Since this is part of the create room process,
143         // leave if just creating.
144         if (!allowingJoining)
145         {
146             PhotonNetwork.LeaveRoom();
147         }
148     }
149
150
151     public override void OnCreatedRoom ()
152     {
153         Debug.Log ("Room created");
154     }
155
156     public override void OnCreateRoomFailed (short
157         resultCode, string message)
158     {
159         Debug.Log ("Failed to create room " + resultCode + " "
160             + message);
161     }
162
163     public void addRoom (Text name)
164     {
165         Debug.Log ("Adding new room: " + name.text);
166         RoomOptions ro = new RoomOptions ();
167         ro.EmptyRoomTtl = 100000; // 100 * 1000 ms
168
169         // Export the notices property to the lobby.
170         string [] roomPropsInLobby = { "notices" };
171         ro.CustomRoomPropertiesForLobby = roomPropsInLobby;
172         ExitGames.Client.Photon.Hashtable customRoomProperties
173             = new ExitGames.Client.Photon.Hashtable () { {
174                 "notices", "Room Start\n" } };
175         ro.CustomRoomProperties = customRoomProperties;
176         PhotonNetwork.JoinOrCreateRoom (name.text, ro, null);
177     }
178 }
```

3. The PhotonServerSettings under Photon/PhotonUnity-Networking/Resources will be used to manage the

network configuration. For the moment it can be convenient just to turn on all logging functionality so you can see the connection sequence in the Unity software console.

Another setting that will be useful in future is the region selection which defines the server that your application will connect to. Typically the Photon functions will try to locate the closest server to you.

To start things off, ensure that RoomManager inherits from MonoBehaviourPunCallbacks (rather than MonoBehaviour) and make a call to PhotonNetwork.ConnectUsingSettings() in the start method. You'll need to include the line: `using Photon.Pun;` at the top of the file.

If you run the application at this point, you should be able to see log messages (if you turned logging on) indicating that your application is connecting to a Photon server.

4. Next we will join the default lobby. This doesn't provide access to other people but will allow us to get a list of rooms available.

Override the `OnConnectedToMaster` event handler. This is called by Photon once the `ConnectUsingSettings` connection call has successfully completed. Make a call to `PhotonNetwork.JoinLobby()` inside this function so that we join the default lobby.

---

```
1  public override void OnConnectedToMaster()
2  {
3      PhotonNetwork.JoinLobby ();
4 }
```

---

5. Maintaining the list of rooms is achieved by subscribing to the `RoomListUpdate` events. These are sent whenever the status of the room is updated, so can be used to keep track of rooms being added and removed, or when properties such as number of participants changes.

Override the `OnRoomListUpdate` event handler which is called once we successfully join a lobby. The parameter provided to this function is a list of the rooms available. Printing this list out to the console provides a way of monitoring which rooms currently exist.

---

```

1  public override void OnRoomListUpdate (List<RoomInfo
                                         > roomList)
2  {
3      foreach (RoomInfo ri in roomList)
4      {
5          ...
6      }
7 }
```

---

The full script provided in Algorithm 21.3 creates a scene entry for each room consisting of details of the room, as well as a control for entering that room. A list of these entries is maintained and updated every time the `RoomListUpdate` event is received. Details of this process are described in a later step.

Running the application at this stage will not display any rooms, since none have been created yet. We'll take care of that in the next step.

6. We need some user interface elements to allow us to create new rooms. Create two new UI elements in the scene: a Button and an InputField. Place these in a convenient location and label them as `RoomAddButton` and `RoomNameInput` respectively.

For convenience when exporting to displays with different resolutions, set the UI Scale Mode of the Canvas Scaler component of the Canvas to 'Scale with Screen Size'. Likewise setting the Horizontal and Vertical Overflow properties of any text elements will ensure they don't get clipped if too much text is entered.

Add a public method to the `RoomManager` called `addRoom`, that takes a `Text` field as a parameter.

Select the `RoomAddButton` and add an entry to the `On Click ()` handlers. Provide the `RoomManager` as the object field, and the `RoomManager.addRoom` as the

function. The parameter of the function needs to be the Text object that is part of RoomNameInput.

When the program runs, clicking the RoomAddButton should make a call to addRoom, passing in the text from RoomNameInput.

The addRoom function makes a call to the JoinOrCreateRoom facility provided by Photon. This performs several steps: creating a new room if none exists (and providing an OnCreatedRoom event), automatically forcing the participant to join the room (there doesn't seem any way to avoid this currently) and producing an OnJoinedRoom event, and triggering one or more RoomListUpdate events.

```

1  public void addRoom (Text name)
2  {
3      ...
4      PhotonNetwork.JoinOrCreateRoom (name.text, ro,
5          null);
6  }
```

---

In this example we largely ignore the OnCreatedRoom event, and provide a minimal response to the OnJoinedRoom event by immediately leaving the room. This is so we can remain in the lobby to monitor the state of the available rooms including the one we've just created.

```

1  public override void OnJoinedRoom ()
2  {
3      ...
4      // Since this is part of the create room process,
5      // leave if just creating.
6      if (!allowingJoining)
7      {
8          PhotonNetwork.LeaveRoom();
9      }
```

---

7. Having returned to the lobby we expect to get the RoomListUpdate events, and it is helpful to show and update a list of available rooms. Our room manager will maintain this list and manipulate the scene to provide a visible representation of each room.

Create a new empty object on the Canvas in the scene and name it Room. As children of the Room object, create a text element (named RoomText) and a button named EnterRoomButton. For convenience I suggest placing the button above the text area. The button, as you can guess from the name, will be used to enter the specific room. The text area will be used to show the room status. In your own adaptation of this component you may want to replace these 2D widgets with a more visually accurate representation of the properties of each room.

Move the Room object to the assets area of the project to turn it into a prefab. You can then remove the original version from the scene. The RoomManager class can now be updated to:

- (a) Have a public roomPrefab variable that should be assigned the newly created Room prefab.
- (b) A private list variable that will keep track of each active room, made up of instances of the Room prefab.
- (c) Extensions to the OnRoomListUpdate event handler to create new instances if new rooms are identified, remove any inactive rooms and to update the text area with the latest properties of each room.

These updates, together with their supporting functions, are provided in detail in Algorithms 21.3.

8. The room prefab needs a few extra supporting elements. Create a C# script named DisplayRoom which contains the script provided in Algorithm 21.4. This mainly provides a function to set the text element associated with an instance of the room prefab. It does set the first text element found so do make sure the RoomText object is the first child of the Room prefab.

Code Listing 21.4: `DisplayRoom`. Class to map properties of a room to the visual display elements.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5
6  public class DisplayRoom : MonoBehaviour
7  {
8      private string roomName;
9
10     public string getName ()
11     {
12         return roomName;
13     }
14
15     public void setName (string name)
16     {
17         roomName = name;
18     }
19
20     public void display (string message)
21     {
22         GetComponentInChildren <Text> ().text = message;
23     }
24 }
```

Create another C# script named LocalRoomBehaviour and include the code given in Algorithm 21.5.

Code Listing 21.5: LocalRoomBehaviour. Mechanism used for entry into one of the rooms.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Photon.Pun;
5  using Photon.Realtime;
6
7  public class LocalRoomBehaviour : MonoBehaviour
8  {
9      private RoomManager roomManager;
10
11     public void setManager(RoomManager manager)
12     {
13         roomManager = manager;
14     }
15
16     public void enterRoom ()
17     {
18         string roomName =
19             GetComponent<DisplayRoom>().getName();
             Debug.Log("Entering room: " + roomName);
```

```
20         roomManager.JoinRoom ( roomName );  
21     }  
22 }
```

Both these script components need to be added to the Room prefab. Also connect the On Click event of the EnterRoomButton to the enterRoom function of the LocalRoomBehaviour component of the Room prefab.

9. Test the functionality in the current version. Once the application has started and connected to the cloud server, it should be possible to enter a new room name, press the create room button and have a new room entry appear in the scene. You may need to tweak some of the positioning values in the updateRooms function of the RoomManager component to match your preferred layout.

You can even try to join one of the rooms you create but are unlikely to see any effect yet as we have still to create a scene to represent the virtual world inside a room.

10. The focus of this example is on achieving communication between the lobby and each of the rooms. In Photon, each room has a property that allows for additional information to customize the room for purposes specific to the application context. We will use two extra properties: ‘notices’ for lobby-room communication and ‘talk’ for communication only within the room. By attaching these to the room, rather than any participant, the values provided are independent of any single client.

Some extra steps are included in the addRoom method of the RoomManager component to set some of these options. Specifically we set a lifetime for the room to be 100 s, so it will stay in existence for a reasonable amount of time even when it contains no participants. Secondly we explicitly add the custom ‘notices’ property and flag that as a property shared with the lobby.

This way it will be updated when RoomListUpdates are received.

```

1 RoomOptions ro = new RoomOptions ();
2 ro.EmptyRoomTtl = 100000; // 100 * 1000 ms
3
4 string [] roomPropsInLobby = { "notices" };
5 ro.CustomRoomPropertiesForLobby = roomPropsInLobby;
6 ExitGames.Client.Photon.Hashtable
    customRoomProperties = new ExitGames.Client.
    Photon.Hashtable () { { "notices", "Room Start\n"
      " } } ;
7 ro.CustomRoomProperties = customRoomProperties;

```

---

11. The next step is to create the scenes for the rooms themselves. This example assumes the first scene created is named MeetingRooms. Create a second scene named Campfire and furnish this with some relevant content. Mine contains a floor plane, a few cylinders representing logs and a particle system to provide a fire, to provide a theme of conversations around a camp fire.

Make sure both scenes are added to the ‘Scenes in Build’ region of the Build Settings (on the File menu). This ensures that we can switch from one scene to the other within the application.

Equip the room with two 3D text objects, one to show the name of the room, and the other to display text messages between the participants in the room.

Provide a button labelled ‘Leave Room’ so that we have a way back to the lobby.

Create a new empty object called CampfireManager and attach a C# script also named CampfireManager. The source for this component is provided in Algorithm 21.6.

Code Listing 21.6: CampfireManager. Room management for the camp fire scene.

```

1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using Photon.Pun;

```

```
5   using Photon.Realtime;
6   using System;
7   using System.Linq;
8
9   public class CampfireManager : MonoBehaviourPunCallbacks
10 {
11     public TextMesh roomLabel;
12     public GameObject avatarPrefab;
13     public TextMesh messageBoard;
14
15     public override void OnJoinedRoom ()
16     {
17       if (roomLabel != null)
18       {
19         if (PhotonNetwork.CurrentRoom != null)
20         {
21           roomLabel.text = "Room:\n" +
22             PhotonNetwork.CurrentRoom.Name;
23         }
24         else
25         {
26           roomLabel.text = "Room:\n" + "Not available";
27         }
28       }
29
30       PhotonNetwork.Instantiate (avatarPrefab.name, new
31         Vector3 (), Quaternion.identity, 0);
32       OnRoomPropertiesUpdate
33       (PhotonNetwork.CurrentRoom.CustomProperties);
34     }
35
36     public void LeaveRoom ()
37     {
38       PhotonNetwork.LeaveRoom();
39       PhotonNetwork.LoadLevel ("MeetingRooms");
40     }
41
42     public override void OnRoomPropertiesUpdate
43       (ExitGames.Client.Photon.Hashtable
44       propertiesThatChanged)
45     {
46       Debug.Log ("Room property update");
47       // Trim last n lines.
48       int n = 10;
49       messageBoard.text = string.Join (Environment.NewLine,
50         ((string) propertiesThatChanged["talk"]).Split
51         (Environment.NewLine.ToCharArray()).Reverse ().Take
52         (n).Reverse ().ToArray ());
53     }
54
55   }
```

The camp fire manager provides a number of support services for this scene.

It manages the OnJoinedRoom event by setting the text for the name of the room. It then also instantiates an avatar for the participant (described in the next step).

The ‘Leave Room’ button needs to have its On Click event handler connected to the LeaveRoom method of the CampfireManager. This disconnects from the room, and reloads the lobby scene.

The camp fire manager also catches the event associated with updates to the custom properties of the room. It retrieves (a portion of) the text messages exchanged by avatars in the room, and shows them on the text object created for this purpose.

Make sure both text properties for the CampfireManager are associated with the two text objects created at the start of this step.

12. The process for managing the avatar is very similar to that described in section 3.3.

Create a new empty object in the scene and name it Avatar. Provide some suitable geometric representation. This should include some marker for the forward direction placed in the direction of the positive z axis. A 3D text field attached to the avatar can be used to display the identity of that particular participant. This object also needs a PhotonView and PhotonTransformView component. Make sure the PhotonTransformView is added to the ObservedComponents property of the PhotonView.

Convert the Avatar object into a prefab (drag it into the assets area, and remove it from the scene). Make sure it is stored under a folder named Resources in the assets area.

Add this avatar prefab to the corresponding property field of the CampfireManager.

Random advice: apparently photon will give error messages of the form 'DefaultPool failed to load Avatar'. Make sure it's in a Resources folder'. Apparently this can be fixed by just recompiling. One way to achieve this is to make a cosmetic change to the file that is trying to instantiate the avatar. Other solutions suggested include renaming, or reimporting the avatar prefab.

13. Control of the avatar requires a number of buttons added to the user interface. You will need:
  - (a) Buttons named LeftButton, RightButton and ForwardButton, all with EventTrigger components added to them. Each event trigger component needs a PointerDown event handler in the first position and a PointerUp event handler in the second position.
  - (b) Buttons named TalkButton, LobbyButton and NickNameButton, each with an EventTrigger component. Each event trigger needs a PointerClick event handle in the first position. Each button also needs a corresponding Input Field named TalkMessage, LobbyMessage and NickNameName respectively.

Create a new C# script named AvatarActions. Include the script provided in Algorithm 21.7. Most of the functionality is as described for section 3.3. Functions specific to this example are:

- Lobby: which copies the contents of the LobbyMessage input field to the 'notices' custom property of the room so that the message is visible to anyone in the lobby.
- Talk: which appends the contents of the TalkMessage input field to the 'talk' custom property of the room. This change is picked up by the event handler in the camp fire manager and use to update the display in the room.

Code Listing 21.7: AvatarActions. Actions for an avatar within a room.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.EventSystems;
5  using UnityEngine.UI;
6  using Photon.Pun;
7  using Photon.Realtime;
8
9  public class AvatarActions : MonoBehaviourPun
10 {
11     [Tooltip ("Turn speed in degrees per second")]
12     public float turnSpeed = 100.0f;
13
14     [Tooltip ("Movement speed in meters per second
15     ↪ (assumes 1 unit = 1 meter)")]
16     public float moveSpeed = 10.0f;
17
18     private float turn = 0.0f;
19
20     private float move = 0.0f;
21
22     private void setButtonCallbacks ()
23     {
24         // Assumes each button has already been set with
25         // two triggers; a pointer down followed by a pointer
26         // up.
27         GameObject.Find ("Canvas/LeftButton").GetComponent
28             <EventTrigger> ().triggers[0].callback.AddListener
29             ((data) => { Left (); });
30         GameObject.Find ("Canvas/LeftButton").GetComponent
31             <EventTrigger> ().triggers[1].callback.AddListener
32             ((data) => { Stop (); });
33         GameObject.Find ("Canvas/RightButton").GetComponent
34             <EventTrigger> ().triggers[0].callback.AddListener
35             ((data) => { Right (); });
36         GameObject.Find ("Canvas/RightButton").GetComponent
37             <EventTrigger> ().triggers[1].callback.AddListener
38             ((data) => { Stop (); });
39         GameObject.Find
40             ("Canvas/ForwardButton").GetComponent
41             <EventTrigger> ().triggers[0].callback.AddListener
42             ((data) => { Forward (); });
43         GameObject.Find
44             ("Canvas/ForwardButton").GetComponent
45             <EventTrigger> ().triggers[1].callback.AddListener
46             ((data) => { Stop (); });
47         GameObject.Find ("Canvas/TalkButton").GetComponent
48             <EventTrigger> ().triggers[0].callback.AddListener
49             ((data) => { Talk (); });
50         GameObject.Find ("Canvas/LobbyButton").GetComponent
51             <EventTrigger> ().triggers[0].callback.AddListener
52             ((data) => { Lobby (); });
```

```
32     GameObject.Find
33     ("Canvas/NickNameButton").GetComponent
34     <EventTrigger> ().triggers[0].callback.AddListener
35     ((data) => { Nickname (); });
36     }
37
38     public void Start ()
39     {
40         if (photonView.IsMine == true ||
41             PhotonNetwork.isConnected == false)
42         {
43             setButtonCallbacks ();
44             transform.Find ("Camera").gameObject.SetActive
45             (true);
46         }
47         photonView.RPC ("showNickname", RpcTarget.All,
48             RoomManager.getName (this.gameObject));
49     }
50
51     public void Update ()
52     {
53         if (photonView.IsMine == true ||
54             PhotonNetwork.isConnected == false)
55         {
56             transform.rotation *= Quaternion.AngleAxis (turn
57             * turnSpeed * Time.deltaTime, Vector3.up);
58             transform.position += move * moveSpeed *
59             transform.forward;
60         }
61     }
62
63     public void Left ()
64     {
65         turn = -1.0f;
66     }
67     public void Right ()
68     {
69         turn = 1.0f;
70     }
71     public void Forward ()
72     {
73         move = 1.0f;
74     }
75
76     // Write a message in the "talk" custom property
77     // which is restricted just to this room.
78     public void Talk ()
79     {
80         GameObject t = GameObject.Find
81         ("Canvas/TalkMessage/Text");
82         if (t != null)
83         {
84             Room r = PhotonNetwork.CurrentRoom;
```

```
75     ExitGames.Client.Photon.Hashtable p =
76     r.CustomProperties;
77     p["talk"] += RoomManager.getName
78     (this.gameObject) + ":" + Time.time + ":" +
79     t.GetComponent <Text> ().text + "\n";
80     r.SetCustomProperties (p);
81   }
82 }
83
84 // Write a message into the "notices" custom property
85 // which is shared with the lobby.
86 public void Lobby ()
87 {
88   GameObject t = GameObject.Find
89   ("Canvas/LobbyMessage/Text");
90   if (t != null)
91   {
92     Room r = PhotonNetwork.CurrentRoom;
93     ExitGames.Client.Photon.Hashtable p =
94     r.CustomProperties;
95     p["notices"] = RoomManager.getName
96     (this.gameObject) + ":" + Time.time + ":" +
97     t.GetComponent <Text> ().text + "\n";
98     r.SetCustomProperties (p);
99   }
100 }
101
102 [PunRPC]
103 void showNickname (string name)
104 {
105   transform.Find ("NameText").gameObject.GetComponent
106   <TextMesh> ().text = name;
107 }
108
109 public void Nickname ()
110 {
111   GameObject t = GameObject.Find
112   ("Canvas/NickNameName/Text");
113   if (t != null)
114   {
115     GetComponent <PhotonView> ().Owner.NickName =
116     t.GetComponent <Text> ().text;
117     photonView.RPC ("showNickname", RpcTarget.All,
118     RoomManager.getName (this.gameObject));
119   }
120 }
121
122 public void Stop ()
123 {
124   turn = 0.0f;
125   move = 0.0f;
126 }
```

116 }

- Nickname: which retrieves the contents of the NickNameName input field. It then makes a remote procedure call (RPC) to send this name to every imposter for the avatar on every client in the room. The destination for the remote procedure call is the showNickname function which runs on every client to update the text field associated with the imposter of the avatar for the originating client, so that the name shown above that particular avatar is changed.
14. The example is finally finished. Try it out by deploying to several mobile devices, or to multiple instances running on one or more desktop platforms. Ensure that you can track numbers of participants in each room, send messages to one another within a room, and also make calls to anyone watching from the lobby.

Figure 21.2.3:  
Rooms can be  
created from the  
lobby.



### 21.3 *Allowing participants to talk to one another using voice communication*

#### **Component 21.3.1: Conversations with Photon Chat**

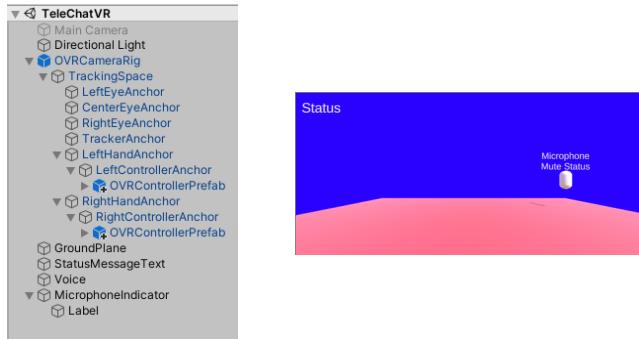
TeleChatVR
------------

1. Start a new project. This needs to be configured for deployment on a mobile device, and the Photon Voice

Figure 21.2.4:  
Participants interact  
within the room,  
and to the lobby.



Figure 21.3.1: Scene  
set up for voice  
communication.



and Oculus Integration packages should be added from the Unity store. In summary:

- Switch the platform to Android.
- Set the package name to a suitably unique identifier. Also ensure that the minimum API level is sufficiently high (a value such as 24 for Android Nougat should be sufficient).
- Download and install the PhotonVoice package from the Unity store.
- Create a new Photon Voice application using the dashboard on the Photon web site, and enter the resulting AppID when prompted. Check that the AppID has been entered under the Voice AppID as well, in the Photon / PhotoUnityNetworking / Resources / PhotonServerSettings asset.
- Download and install the Oculus Integration package from the Unity store.

- 
- 
- 
- 
- 
- 
- 
- (f) Under the XR Setting, enable 'Virtual Reality Supported' and ensure the Oculus SDK is added. Turn on V2 signing if using the Quest.
  - (g) Remove the Vulkan API from the list of graphics APIs, under Player/Other Settings.
2. To set the scene for the Quest, disable the main camera and replace it with the OVRCameraRig from the Oculus/VR/Prefabs folder in the Assets area. Add copies of the OVRControllerPrefab to the Left- and Right-ControllerAnchors in the OVRCameraRig hierarchy and set them to the appropriate handedness.  
A ground plane can also be added to the scene, just to provide a visual reference.
3. Add a new 3D text object to the scene. This will be used to display the status of the connection, and is particularly intended to support debugging during the development phase. A 3D text object ensures that the text is part of the scene and not an overlaid user interface layer which may not be visible on some head mounted displays. A text object using the TextMeshPro library would be appropriate for this. Make sure this comes from the '3D Object' category as there are also TextMeshPro Text objects available under the UI category which are not appropriate for use in this situation. Place this in a visible region of the scene and set the font size so that several lines of text will be visible.  
You may need to agree to prompts to include resources from the TextMeshPro into the project.
4. Create a new empty object in the scene and name it Voice. Also create a new C# script called VoiceManager and add it to the Voice object.  
Add a VoiceConnection component to the Voice object. Copy the AppID into App ID Voice parameter in the Settings portion of the VoiceConnection component.  
Add a Recorder component to the Voice object. Drag

this component into the Recorder property of the Voice-Connection component.

The script for VoiceManager is provided as Algorithm 21.8..

Code Listing 21.8: VoiceManager. Voice based communication requires first establishing a connection.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Photon.Voice.Unity;
5  using Photon.Realtime;
6  using Photon.Pun;
7
8  public class VoiceManager : MonoBehaviourPunCallbacks
9  {
10    [Tooltip ("TextMeshPro object for displaying call
11    ↵ status")]
12    public TMPPro.TextMeshPro status;
13
14    [Tooltip ("Maximum length of status message in
15    ↵ characters")]
16    public int statusMaxLength = 100;
17
18    [Tooltip ("Controller input used to toggle microphone")]
19    public OVRInput.RawButton button =
20    ↵ OVRInput.RawButton.LThumbstickUp;
21
22    public GameObject microphoneIndicator;
23    public Material microphoneOn;
24    public Material microphoneOff;
25
26    private string previousMessage = "";
27
28    private void setStatusText (string message)
29    {
30      if (!message.Equals (previousMessage))
31      {
32        Debug.Log (message);
33        status.text += "\n" + message;
34        if (status.text.Length > statusMaxLength)
35        {
36          status.text = status.text.Remove (0,
37          ↵ status.text.Length - statusMaxLength);
38        }
39        previousMessage = message;
40      }
41
42      void Start()
43      {
44
```

```
41     status.text = "";
42     setStatusText ("Application started");
43
44     PhotonNetwork.ConnectUsingSettings();
45
46     VoiceConnection vc = GetComponent <VoiceConnection> ();
47     vc.Client.AddCallbackTarget (this);
48     vc.ConnectUsingSettings();
49 }
50
51 public override void OnConnectedToMaster()
52 {
53     VoiceConnection vc = GetComponent <VoiceConnection> ();
54     RoomOptions roomopt = new RoomOptions ();
55     TypedLobby lobby = new TypedLobby ("ApplicationLobby",
56     ↳ LobbyType.Default);
57     vc.Client.OpJoinOrCreateRoom (new EnterRoomParams {
58     ↳ RoomName = "ApplicationRoom", RoomOptions = roomopt,
59     ↳ Lobby = lobby });
60 }
61
62 public override void OnJoinedRoom()
63 {
64     setStatusText ("Joined room with " +
65     ↳ PhotonNetwork.CurrentRoom.PlayerCount + "
66     ↳ participants.");
67 }
68
69 public void OnDisconnected(DisconnectCause cause)
70 {
71     setStatusText ("Disconnected " + cause);
72 }
73
74 void switchMicrophone ()
75 {
76     VoiceConnection vc = GetComponent <VoiceConnection> ();
77     if ((OVRInput.GetDown (button)) ||
78     ↳ (Input.GetButtonDown ("Fire1")))
79     {
80         vc.PrimaryRecorder.TransmitEnabled =
81         !vc.PrimaryRecorder.TransmitEnabled;
82     }
83     if (microphoneIndicator != null)
84     {
85         if (vc.PrimaryRecorder.TransmitEnabled)
86         {
87             microphoneIndicator.GetComponent <MeshRenderer>
88             ().material = microphoneOn;
89         }
90         else
91         {
92             microphoneIndicator.GetComponent <MeshRenderer>
93             ().material = microphoneOff;
```

```

85         }
86     }
87 }
88
89 void Update()
90 {
91     VoiceConnection vc = GetComponent <VoiceConnection> ();
92
93     string otherParticipants = "";
94     if (vc.Client.InRoom)
95     {
96         Dictionary <int, Player>.ValueCollection pts =
97         vc.Client.CurrentRoom.Players.Values;
98         foreach (Player p in pts)
99         {
100             otherParticipants += p.ToStringFull ();
101         }
102     string room = "not in room";
103     if (vc.Client.CurrentRoom != null)
104     {
105         room = vc.Client.CurrentRoom.Name;
106     }
107     setStatusText (vc.Client.State.ToString () + " server:
108     " + vc.Client.CloudRegion + ":" +
109     vc.Client.CurrentServerAddress +
110     " room: " + room + " participants: " +
111     otherParticipants);
112
113     switchMicrophone ();
114 }
115 }
```

The bulk of this focuses on providing status information about the connection. The important steps in the process follow the same procedure as for other Photon networking operations; specifically arranging to connect and join a room. The other supporting function likely to be reused in other applications is the method used to toggle access to the microphone.

5. A Speaker object needs to be created for each incoming connection. This is handled by the VoiceConnection component but we do need to provide a suitable template object. Create this by adding a new object to the scene, and giving this an AudioSource and a Speaker component. Convert this to a prefab, and provide this

prefab as the Speaker Prefab property of the VoiceConnection.

6. Add another object to serve as the indicator of whether the microphone is muted. Create two materials that can be used to represent the on and off states. Provide these as properties to the VoiceManager component. The button property can be set to the specific control on the controllers that will mute the microphone.
7. The application can be deployed to other devices. Once correctly connected, the status message should indicate that it has joined the room named 'ApplicationRoom'. The controller trigger, mouse click, or screen tap can be used to toggle the microphone.

When running make sure the application has permission to use the microphone on the device. If you miss this prompt when the application starts then you may need to exit the application and restart it.

It should run on a standard mobile phone but may require enabling the regular camera and disabling the OVRCameraRig for that to work.

## 21.4 *Creating a multi-participant experience with video communication using web technologies*

### **Component 21.4.1: Video conferencing.**

<i>Video Conferencing</i>
-------------------------------

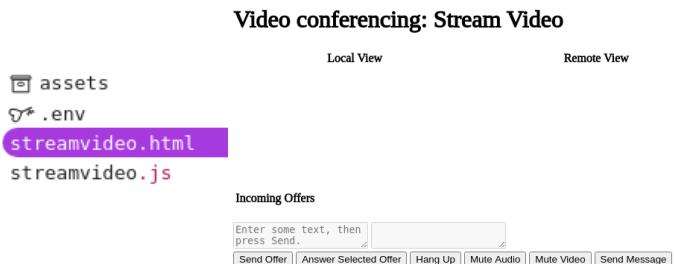
*This example works through the process of creating a video conferencing application that runs in a browser, either on a desktop or mobile device. It follows the process outlined in the introduction to WebRTC provided at: <https://codelabs.developers.google.com/codelabs/webrtc-web/>.*

1. One advantage of using web based technologies is that the software distribution infrastructure is built in. The software consisting of HTML and javascript files is

Figure 21.3.2: The voice communication facility included in a virtual environment. Connection information provided for diagnostic purposes.



Figure 21.4.1:  
Layout of the  
video conferencing  
application.



hosted on a web server, and downloaded as part of the web page when the application is accessed.

This example needs a web server to host the application. Each client (and two are the minimum to have a conversation) can then access the server and have immediate access to the most recent version that is also only stored in a single place (on the server). This avoids one of the complicating issues in developing networking software which is making sure all participants are using the same version of the application.

A convenient web hosting solution is available at: <https://glitch.com>. This allows short term hosting of content accessible via unique URL. Visit glitch using a web browser and create a new project. The simplest web page template is adequate for the purposes of this example.

2. The template usually provides some minimal html and javascript files for a basic web page. You can either modify these, or create new files. The Show Live button opens a new tab which displays the page as downloaded from the server. Use the URL (address) on this page in other browsers on other devices to test out the communication elements developed throughout this example.
3. The first step is to access the video content on the local device. Create a basic index.html page that accesses the javascript code in video.js.

The key element in the index.html page is a video tag. The javascript code attaches the video stream from the camera to this tag so that the video content is displayed on the web page.

```
1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Video conferencing: Play Video</title>
5  </head>
6
7  <body>
```

```

8   <h1>Video conferencing: Play Video</h1>
9   <video autoplay playsinline></video>
10  <video width="320" height="240" controls></video>
11  <script src="video.js"></script>
12 </body>
13 </html>

```

---

The javascript consists of the following steps:

- (a) `getUserMedia`: this requests permission to access particular media sources. In this case, access to the video stream from the camera.
- (b) `getLocalMediaStream`: this attaches the video stream from the camera to any video tags found.

```

1  'use strict';
2
3  const mediaStreamConstraints = {
4      video: true,
5  };
6
7  function gotLocalMediaStream(mediaStream) {
8      var videoTags = document.getElementsByTagName ("video");
9      console.log ("Got " + videoTags.length + " video
tags");
10     for (var i = 0; i < videoTags.length; i++)
11     {
12         videoTags[i].srcObject = mediaStream;
13     }
14 }
15
16 function handleLocalMediaStreamError(error) {
17     console.log('navigator.getUserMedia error: ',
error);
18 }
19
20 navigator.mediaDevices.getUserMedia (
    mediaStreamConstraints) .then(
        gotLocalMediaStream) .catch(
            handleLocalMediaStreamError);

```

---

Note that when testing this on a mobile device you should access the web site through the secure `https://` version of the link (camera access is blocked when using the default `http://`).

4. These steps demonstrate how local access to media on the device is achieved via the browser, and some

HTML and javascript. The next step is to build a full video conferencing system by using WebRTC to communicate these streams to another device. Most of the complexity of this process is involved in negotiating the connection. This is not part of the WebRTC services, and must be carried out separately. In this case we create a small networked server that receives messages from one party interested in communicating, and shares them with all the others who would like to participate.

For the moment we'll assume the existence of this signalling server (described in a later step) and concentrate on the steps involved in establishing the connection.

The initial stage is to lay out the web page with the required elements. Create a file called streamvideo.html on your glitch site and include the contents provided in Algorithm 21.9.

Code Listing 21.9: `streamvideo`. The page layout for the video conferencing application includes regions for local and remote video streams, exchange of test messages and the usual set of buttons to control calls and manage media.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <title>Video conferencing: Stream Video</title>
5  </head>
6
7  <body>
8      <h1>Video conferencing: Stream Video</h1>
9
10 <table>
11 <tr>
12     <th>Local View</th>
13     <th>Remote View</th>
14 </tr>
15 <tr>
16     <td><video id="localVideo" width="320" autoplay
17         &gt;playsinline</video></td>
18     <td><video id="remoteVideo" width="320" autoplay
19         &gt;playsinline</video></td>
20 </tr>
</table>
```

```

21   <table>
22     <tr>
23       <th>Incoming Offers</th>
24     </tr>
25     <tr>
26       <td>
27         <ol id="offerList"></ol>
28       </td>
29     </tr>
30   </table>
31
32
33   <textarea id="dataChannelSend" disabled
34     ↳ placeholder="Enter some text, then press
35     ↳ Send."></textarea>
36   <textarea id="dataChannelReceive" disabled></textarea>
37
38   <div>
39     <button id="sendOfferButton">Send Offer</button>
40     <button id="answerOfferButton">Answer Selected
41     ↳ Offer</button>
42     <button id="hangupButton">Hang Up</button>
43     <button id="muteAudioButton">Mute Audio</button>
44     <button id="muteVideoButton">Mute Video</button>
45     <button id="sendMessageButton">Send Message</button>
46   </div>
47
48   <script src="https://webrtc.github.io/adapter/adapter-l
49     ↳ test.js"></script>
50   <script src="streamvideo.js"></script>
51 </body>
52 </html>

```

Two video tags are provided for the display of the camera feeds from each side. The traditional buttons are provided to mute the audio and video feeds. Text areas are used to provide a minimal text messaging facility. Messages are sent when the Send Message button is pressed.

The communication protocol involves one side sending offers to other parties, offering to communicate. These are triggered by the Send Offer button. A list is provided on the web page to show the offers that have been received. One of the offers can be selected, and the Answered Selection Option button pressed to send an answer to that offer, and to establish communication.

5. The WebRTC connection process requires the transmission of offers which are completed by receiving an answer. Interleaved with this process is the exchange of ICE candidates. An ICE (Interactive Connectivity Establishment) is used to determine an address to connect to that will work with firewalls and network address translation (NAT) devices. This can involve negotiation with STUN (Session Traversal Utilities) servers which help figure out what your address looks like to the outside world, and TURN (Traversal Using Relays around NAT) which relay data for you when your connection is highly “secure”. This example doesn’t explicitly include this phase.

The outline of the connection process is:

- (a) Get the local video and audio stream.

```

1  navigator.mediaDevices.getUserMedia({
2    audio: true,
3    video: true
4  })
5  .then(gotStream)
6  .catch(function(e) {
7    alert('getUserMedia() error: ' + e.name);
8 });

```

---

- (b) When the stream is available, the function gotStream is called which connects the stream to the local video window, and creates a new RTCPeerConnection object.

The local video stream is attached to the peer connection at this stage.

```

1  pc.addStream(localStream);

```

---

- (c) When the user presses the Send Offer button, then an offer is generated and sent to the signalling server.

```

1  function sendOffer() {
2    pc.createOffer(setLocalAndSendMessage,
3      handleCreateOfferError);
3 }

```

---

This offer is copied to all other devices connected to this web site by the signalling server.

- (d) When the offer is received by another device, this device updates its list of displayed offers.

```

1  function receiveMessage (event) {
2    var message = JSON.parse (event.data);
3    // able to handle incoming messages.
4    if (message.type == "offer")
5    {
6      // display incoming offers to the user.
7      displayReceivedOffer (message);
8    }
9    ...

```

---

- (e) The user on another device then selects an offer from the displayed list and presses the Answered Selected Offer button. This sends an answer to the signalling server. The local device also updates the peer connection with the details of the offer selected. The signalling server relays the answer to all other connected devices.

- (f) When the source of the offer receives the answer, then the remote properties of the connection are defined and the connection is established.

```

1      else if (message.type === 'answer')
2      {
3        // if an offer is answered, then set up the
4        // connection.
5        pc.setRemoteDescription(new
          RTCSessionDescription(message));
6      }

```

---

- (g) This triggers an event that connects the remote stream of the peer connection to the remote video window.

```

1  function handleRemoteStreamAdded(event) {
2    remoteStream = event.stream;
3    remoteVideo.srcObject = remoteStream;
4  }

```

---

6. While this is happening, ICE candidate information is being shared.

7. In addition to the audio and video media connections, a data connection is also established to allow other forms of information to be communicated. At present only text messages are exchanged but this could be extended to support other information exchanges in a multi-participant enhanced reality experience.

- (a) A data channel is added to the peer connection when it is created.

---

```

1   dataChannel = pc.createDataChannel (""
2       MessageChannel");
3   dataChannel.onopen = onDataChannelStateChange;
4   dataChannel.onclose = onDataChannelStateChange;
5   pc.ondatachannel = receiveChannelCallback;
```

---

- (b) Once the connection is established, a callback can be added to handle incoming messages on the channel.

---

```

1   function receiveChannelCallback(event) {
2       event.channel.onmessage =
3           onReceiveMessageCallback;
4   }
```

---

- (c) When the Send Message button is pressed then the text from the text area is transmitted into the channel.

---

```

1   function sendMessageData() {
2       var data = dataChannelSend.value;
3       dataChannel.send(data);
4   }
```

---

- (d) Conversely, when the receive callback is invoked then the message received is written to the text area on the web page.

---

```

1   function onReceiveMessageCallback(event) {
2       dataChannelReceive.value = event.data;
3   }
```

---

8. The complete code for the streamvideo.js file is provided in Algorithm 21.10.

Code Listing 21.10: streamvideo. Audio, video and data streaming requires the connection establishment process outlined in this code.

```
1 'use strict';
2
3 // The peer connection.
4 var pc;
5 // The local media stream.
6 var localStream;
7 // The media stream from the remote device.
8 var remoteStream;
9
10 // Individual elements on the web page.
11 var localVideo = document.getElementById('localVideo');
12 var remoteVideo = document.getElementById('remoteVideo');
13 var sendOfferButton =
14     document.getElementById('sendOfferButton');
15 var offerList = document.getElementById('offerList');
16 var answerOfferButton =
17     document.getElementById('answerOfferButton');
18 var hangupButton = document.getElementById('hangupButton');
19 var muteAudioButton =
20     document.getElementById('muteAudioButton');
21 var muteVideoButton =
22     document.getElementById('muteVideoButton');
23 var activeOffer;
24 var dataChannelSend =
25     document.getElementById('dataChannelSend');
26 var dataChannelReceive =
27     document.getElementById('dataChannelReceive');
28 var sendMessageButton =
29     document.getElementById('sendMessageButton');
30 var dataChannel;
31
32 // Connect the call control buttons.
33 sendOfferButton.onclick = sendOffer;
34 answerOfferButton.onclick = sendAnswer;
35 hangupButton.onclick = hangup;
36 muteAudioButton.onclick = function () {
37     localStream.getAudioTracks()[0].enabled =
38         !localStream.getAudioTracks()[0].enabled };
39 muteVideoButton.onclick = function () {
40     localStream.getVideoTracks()[0].enabled =
41         !localStream.getVideoTracks()[0].enabled };
42 sendMessageButton.onclick = sendMessageData;
43
44 // The signalling socket, used for establishing
45 // connections.
46 var socket = new WebSocket ("wss://10.1.1.4:5000/");
47 socket.onmessage = receiveMessage;
48
49 // First establish access to the local camera/microphone
50 // when the page loads.
51 navigator.mediaDevices.getUserMedia({
52     audio: true,
```

```

40     video: true
41   })
42   .then(gotStream)
43   .catch(function(e) {
44     alert('getUserMedia() error: ' + e.name);
45   });
46
47   function initializePeerConnection ()
48   {
49     // 1. Create a new RTC Peer Connection
50     // argument used to specify STUN and TURN servers.
51     // additional effort is required to manage these.
52     pc = new RTCPeerConnection(null);
53     // 2. Set the ice candidate event handler.
54     pc.onicecandidate = handleIceCandidate;
55     pc.onaddstream = handleRemoteStreamAdded;
56     pc.onremovestream = handleRemoteStreamRemoved;
57
58     // For an additional data channel.
59     dataChannel = pc.createDataChannel ("MessageChannel");
60     dataChannel.onopen = onDataChannelStateChange;
61     dataChannel.onclose = onDataChannelStateChange;
62     pc.ondatachannel = receiveChannelCallback;
63   }
64
65   // Once access to the camera/microphone are achieved, then
66   // to connect
67   // to the signalling server and make an offer.
68   function gotStream(stream) {
69     console.log('Adding local stream.');
70
71     // Attach the media stream to an output on the local
72     // page.
73     localStream = stream;
74     localVideo.srcObject = stream;
75
76     // Now advertise existence to the signalling server.
77     initializePeerConnection ();
78
79     // 3. Add the local media stream.
80     pc.addStream(localStream);
81   }
82
83   // Send an offer to the signalling server, to be shared
84   // with other connected parties.
85   function sendOffer() {
86     pc.createOffer(setLocalAndSendMessage,
87       handleCreateOfferError);
88   }
89
90   function sendAnswer() {
91     // 4. Press the answer once an incoming offer has been
92     // received.

```

```

88     pc.setRemoteDescription (new
89         → RTCSessionDescription(activeOffer));
90
91     console.log('Sending answer to peer.');
92     pc.createAnswer().then(
93         setLocalAndSendMessage,
94         onCreateSessionDescriptionError
95     );
96 }
97
98 function hangup ()
99 {
100     dataChannel.close ();
101     pc.close ();
102     initializePeerConnection ();
103 }
104
105 // Show all incoming offers as a list, with option to
106 // check one.
107 function displayReceivedOffer (offer)
108 {
109     var x = document.createElement ("input");
110     x.onclick = function () { activeOffer = offer;
111         → console.log ("Setting acti ", activeOffer) };
112     x.setAttribute("type", "radio");
113     x.setAttribute("name", "offers");
114     x.setAttribute("checked", "checked");
115     activeOffer = offer;
116
117     var node = document.createElement ("li");
118     node.appendChild (x);
119     var textnode=document.createTextNode (offer.sdp.toString
120         → ().substring (0, 50));
121     node.appendChild(textnode);
122     offerList.appendChild(node);
123 }
124
125 // Receive a message from the signalling server.
126 function receiveMessage (event) {
127     var message = JSON.parse (event.data);
128     // able to handle incoming messages.
129     if (message.type == "offer")
130     {
131         // display incoming offers to the user.
132         displayReceivedOffer (message);
133     }
134     else if (message.type === 'answer')
135     {
136         // if an offer is answered, then set up the
137         → connection.
138         pc.setRemoteDescription(new
139             → RTCSessionDescription(message));
140     }

```

```

135      // any candidate address updates are registered with
136      // the peer connection.
137      else if (message.type === 'candidate')
138      {
139          var candidate = new RTCIceCandidate({
140              sdpMLineIndex: message.label,
141              candidate: message.candidate
142          });
143          pc.addIceCandidate(candidate);
144      }
145
146 // Send a message to the signalling server.
147 function sendMessage (message)
148 {
149     socket.send (JSON.stringify (message));
150 }
151
152 // When new addressing options are available, these are
153 // shared via the signalling server.
154 function handleIceCandidate(event) {
155     console.log('icecandidate event: ', event);
156     if (event.candidate) {
157         sendMessage({
158             type: 'candidate',
159             label: event.candidate.sdpMLineIndex,
160             id: event.candidate.sdpMid,
161             candidate: event.candidate.candidate
162         });
163     }
164 }
165
166 // 5. Called when remote stream is added, and this
167 // is connected to the remote media element on the web
168 // page.
169 function handleRemoteStreamAdded(event) {
170     console.log('Remote stream added.');
171     remoteStream = event.stream;
172     remoteVideo.srcObject = remoteStream;
173 }
174
175 // When an offer is created, store it and share with
176 // signalling server.
177 function setLocalAndSendMessage(sessionDescription) {
178     pc.setLocalDescription(sessionDescription);
179     sendMessage (sessionDescription);
180 }
181
182 // Called when the data channel becomes active.
183 function onDataChannelStateChange() {
184     if (dataChannel.readyState === 'open') {
185         console.log ("Data channel open");
186         dataChannelSend.disabled = false;

```

```

185     dataChannelSend.focus();
186     sendMessageButton.disabled = false;
187 } else {
188     dataChannelSend.disabled = true;
189     sendMessageButton.disabled = true;
190 }
191 }

// A new connection that has a data channel has been
// created. Set up a function to handle incoming messages.
193 function receiveChannelCallback(event) {
194     console.log('Receive Channel Callback');
195     event.channel.onmessage = onReceiveMessageCallback;
196 }
197

// Send a message on the data channel when the send
// button is pressed.
199 function sendMessageData() {
200     var data = dataChannelSend.value;
201     dataChannel.send(data);
202     console.log('Sent Data: ' + data);
203 }

// Any incoming data messages are placed in the
// received message box.
205 function onReceiveMessageCallback(event) {
206     console.log('Received Message');
207     dataChannelReceive.value = event.data;
208 }

209 function handleRemoteStreamRemoved(event) {
210     console.log('Remote stream removed. Event: ', event);
211 }

212 function handleCreateOfferError(event) {
213     console.log('createOffer() error: ', event);
214 }

215 function onCreateSessionDescriptionError(error) {
216     console.log('Failed to create session description: ' +
217         ← error.toString());
218 }
219
220
221
222
223
224
225

```

9. The signalling service can be implemented using any message exchange platform, including carrier pigeon. The approach used here employs web sockets as a convenient multi-platform communication facility.

A connection to a server is achieved using:

```

1  var socket = new WebSocket ("wss://myaddresshere
   :5000/");

```

---

```
2   socket.onmessage = receiveMessage;
```

---

Modern web browsers expect secure network communication and this encryption is achieved by using the wss:// (rather than ws://).

To reduce expense in this example we are using self-signed certificates on the server. This requires permission to be granted in each browser used. To set this permission, visit https://myaddresshere:5000/ (note https rather than wss) while the server is running. You should receive a security alert. Use the advanced option to accept the certificate and grant access to the site.

10. The signalling server is written in python since this provides the desired functionality with minimal code. Certificates need to be created. If you have the openssl utilities installed this can be done from a command prompt using the instruction:

---

```
1  openssl req -new -newkey rsa:4096 -nodes -x509 -
   keyout key.pem -out cert.pem -days 365
```

---

The server operation consists of either:

- (a) Receiving a new connection, in which case the connection is added to the list of active connections.
- (b) Receiving a message on one connection, in which case the message is sent to all other connections.

The complete signalling server is provided in Algorithm 21.11.

Code Listing 21.11: *connectionservice*. One potential signalling server for use in establishing WebRTC connections. This needs to be run using a python (version 3) interpreter.

```
1  # Simple connection service for webRTC
2  # Requires python 3.
3
4  # Import the support libraries required.
5  # Concurrency support, to avoid delays handling network
   ↳ messages.
```

```

6   import asyncio
7   # Encryption for network stream required by browsers.
8   import ssl
9   # A network library that interacts with similar libraries
10  ↪ supported in browsers.
11  import websockets
12
13  # Keep a list of the clients that have connected so far.
14  clients = []
15
16  # When a new connection comes in, add it to the list of
17  ↪ clients.
18  async def register(websocket):
19      if not websocket in clients:
20          clients.append (websocket)
21
22  # Handle communication with one client.
23  async def handler (websocket, path):
24      print ("Received client.")
25      await register(websocket)
26
27  while True:
28      # Wait for incoming message.
29      message = await websocket.recv()
30
31      print ("Received message", message[:20], "... from"
32          ↪ client", (1+clients.index (websocket)), "of", len
33          ↪ (clients))
34
35      # Copy that message to all other clients.
36      for client in clients:
37          if client != websocket:
38              try:
39                  await client.send (message)
40              except Exception as e:
41                  pass
42                  # Since we don't remove closed connections, some
43                  ↪ errors are expected.
44                  # print ("Problem sending", e)
45
46  # Start up the server.
47
48  # create certificate with: openssl req -new -newkey
49  ↪ rsa:4096 -nodes -x509 -keyout key.pem -out cert.pem
50  ↪ -days 365
51  #
52  # Browsers will have issues accessing the site because of
53  ↪ the self-signed certificate
54  # Visit the address: https://[fill in server address
55  ↪ here]:5000/
56  # It should complain about the certificate and allow it to
57  ↪ be accepted (under "Advanced").
58  ssl_context = ssl.SSLContext(ssl.PROTOCOL_TLS_SERVER)

```

```

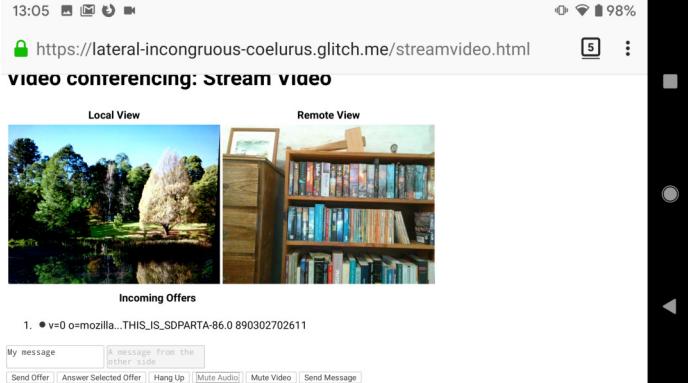
49     ssl_context.load_cert_chain(certfile = 'cert.pem',
50         keyfile = "key.pem")
51     server = websockets.server.serve(handler, '0.0.0.0',
52         5000, ssl=ssl_context)
53
54     asyncio.get_event_loop().run_until_complete(server)
55     asyncio.get_event_loop().run_forever()

```

11. To use the WebRTC video conferencing application, follow these steps:
  - (a) Start the signalling server.
  - (b) Make sure the address of the signalling server is entered into the web socket address used in streamvideo.js.
  - (c) Open the streamvideo.html page (via glitch) on each participating device.
  - (d) On one device, press the Send Offer button.
  - (e) Verify that an offer has been received on another device. Select the offer, and press the Answer Selected Offer button on that device.
  - (f) The connection should be established, sharing audio, video and text messages.

The process can be extended to handle more participants by increasing the number of peer connections used.

Figure 21.4.2:  
Video conferencing runs within a web browser, and requires the signalling server.



**21.5  Customizing the data shared in a multi-participant experience**

Shared Location
--------------------

**Component 21.5.1: Shared location.**

This example provides the typical functionality that might be expected from a multi-participant location based experience. Each participant operates as a client, being able to move their own representation around (usually through moving in the physical world and being tracked by GPS, but using buttons on the screen in this case). Other participants are also represented in the virtual view of the world, and their avatars are updated when their positions change. This example also allows each participant to drop a series of waypoint objects which are also visible to other participants in the experience.

Several options could be considered for building the networking infrastructure used by this example:

- The networking features included in the Unity software could be used. These features have been demonstrated previously Bangay [2019].
- Custom web servers and clients could be used as these can be quickly created using classes such as `HttpListener` and `HttpWebRequest` in C#, or `UnityWebRequest` in the Unity software. The trend towards using secure web communication on mobile devices does require that cryptographic certificates be managed and exchanged which does make this approach more challenging to present in a concise example.
- Networking protocols such as TCP and UDP are also well supported under C# and neatly packaged to remove most of the complexity of their use.

In this case TCP is used and provides a reliable communication protocol for sharing data. Performance overheads associated with providing reliability are not considered an

issue in this particular case as participants are not engaging physically with one another and so real-time interaction is not a primary consideration. Unreliable protocols such as UDP improve responsiveness but require that missing data be compensated for in the communication protocol.

The communication patterns used assume the existence of a server and multiple clients.

1. Clients send the following types of messages to the server:
  - (a) Register a new participant (*Add*). Information sent is: type of element = participant, location, identifier = -1.  
This message generates a response from the server (*IdSet*): type of element = participant, location, identifier = server assigned unique identifier.
  - (b) Update the position of a participant (*Update*): Information sent is: type of element = participant, location, identifier = server assigned unique identifier.  
An update usually trigger transmission of all information about all elements to every client. This is not particularly efficient and could be enhanced to:
    - Only send the updated information to all clients.
    - Send the updates at regular intervals after several changes rather than immediately after a single change.
  - (c) Create a marker (*Add*). Information sent is: type of element = marker, location, identifier = -1.
2. Servers send the following types of messages to clients:
  - (a) Updates on any element: Information sent is: type of element = participant or marker or defunct, location, identifier = unique identifier.  
This information is sufficient for the client to layout and present a representation of all elements.

Defunct elements are those created by a client that is no longer connected.

Since both the clients and server are relatively lightweight (don't have major resource requirements to run on mobile devices) each device can actually run several servers and clients at any one time. This is not particularly useful in this example as clients can only connect to one server at a time, and the visual representations provided by the client may clash when multiple clients are trying to update the view at any time. However this is intended to illustrate that networked experiences can be built around configurations of communicating smart phones without the need for additional hardware.

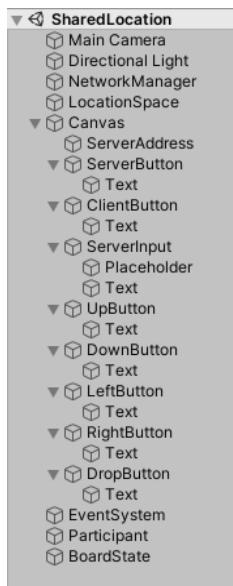


Figure 21.5.1:  
Sharing locations  
across a network  
link.

1. Start with a new project. Set the build settings as required for deployment to a mobile device.
2. The interface to the application consists of a depiction of the space in which the participants exist. This example uses a simple 2D space with coordinates ranging from -5 to 5 in either direction. Given the context of

this chapter, this could represent a map tile with position defined in terms of latitude and longitude.

Create a plane to represent the available space for participants. Add the following elements:

- (a) A button to start a server on this device. A text field next to the button to display the address of the device so that clients can connect.
  - (b) A button to start a client on the device. An input field next to the button so that the client can enter the address of the server that the client is connected to.
  - (c) Four buttons to move the participant's avatar around the location (left, down, right and up).
  - (d) A button to drop a marker at the current location.
3. Create an object to represent the local participant. Add a C# script to this called MoveParticipant, which takes in the button presses, and updates the position of this object. This script also makes a call to the active client to pass on relevant information that can be shared with other clients via the server. The contents of the script are provided in Algorithm 21.12.

Code Listing 21.12: MoveParticipant. Participant movement and actions are driven by events that make use of the public interface provided by this component. Participant state is then relayed to the network manager for communication to the server.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class MoveParticipant : MonoBehaviour {
6
7      private float latitude = 0.0f;
8      private float longitude = 0.0f;
9      private float stepSize = 0.1f;
10
11     public SharedLocation networkManager;
12
13     private void setPosition ()
14     {
15         transform.position = new Vector3 (longitude, latitude,
16             0.0f);
17     }
18 }
```

```

16   // update status via the local client.
17   networkManager.sendPosition (latitude, longitude);
18 }
19
20 private void setDropPosition ()
21 {
22   // inform client of dropped marker.
23   networkManager.sendMarker (latitude, longitude);
24 }
25
26 public void moveUp () { latitude += stepSize;
27   ← setPosition (); }
28 public void moveDown () { latitude -= stepSize;
29   ← setPosition (); }
30 public void moveLeft () { longitude -= stepSize;
31   ← setPosition (); }
32 public void moveRight () { longitude += stepSize;
33   ← setPosition (); }
34 public void dropMarker () { setDropPosition (); }
35 }
```

Connect each of the movement buttons to their corresponding function on the participant object and the MoveParticipant component.

4. Create a Network Manager empty and add a C# script named SharedLocation to this. Connect the server button to the startServer function on the network manager. Provide the text field used to hold the server address to the server address property on the Shared Location component so that it can fill in the address of the server and show this on the user interface.

Similarly, connect the client button to the startClient function, and provide the input field to the target server property.

The network manager script also requires some further resources to provide a representation of the data received from the server. A new empty object called BoardState is created to act as a parent object for the representation of the elements in this database. A prefab used to represent each element also needs to be provided to the Piece Template property.

5. The messages that are exchanged contain similar infor-

mation and so can be represented using a single class that contains fields for the nature of the communication, the type of element being communication, the identifier for that element, and the position (in the form of latitude and longitude values) being shared.

Several enumerated types are created to represent values such as the nature of the communication and type of element to ensure that all options are explicitly defined and to help make matching the code to the communication protocol explicit.

The class is also designated as Serializable since it will need to be able to be transformed from its usual object representation into a byte array for transmission over the network, and then back to object representation at the destination. The details of this message class are provided in Algorithm 21.13.

Code Listing 21.13: LocationData. The message class used to engage information in the multi-participant location based experience.

```

1  using UnityEngine;
2  using System;
3
4  /*
5   * Definitions of the messages that are communicated
6   * across the network.
7   */
8
9  /*
10  * The different types of message (control information).
11  * This includes:
12  * - Add: register a new object on the server.
13  * - IdSet: return confirmation of a new object, with
14  * its identifier.
15  * - Update: share the location of an existing object
16  */
17
18  /*
19  * The different types of object that may exist:
20  * - Participant: User operating one of the clients.
21  * - Waypoint: Marker dropped by one of the users.
22  * - Defunct: Object related to a client that is not
23  * longer connected.
24  */
25  public enum MessageStatus { Add, IdSet, Update };
26
27  /*
28  * The different types of object that may exist:
29  * - Participant: User operating one of the clients.
30  * - Waypoint: Marker dropped by one of the users.
31  * - Defunct: Object related to a client that is not
32  * longer connected.
33  */
34  public enum LocationType { Participant, Waypoint, Defunct
35  };

```

```

24
25  /*
26   * The message exchanged. For simplicity, only one type of
27   * message is communicated, with both control and data
28   * mixed
29   * into a single class. The communication protocol is
30   * simple
31   * enough that there is not too much wasted data.
32   */
33 [Serializable]
34 public class LocationData
35 {
36     // Control information.
37     public MessageStatus status;
38     // Type of object whose location is provided.
39     public LocationType locType;
40     // A unique identifier distinguishing this object from
41     // any other.
42     public int identifier;
43     // Position of the object.
44     public float latitude;
45     public float longitude;
46 }

```

- The SharedLocation script provides facilities that are shared by both client and server. Specifically it provides entry points to create new client and the server objects. It also provides access to the facilities provided by Unity. Write and read access to the server address fields is managed through this class, rather than complicating clients and servers with the need to integrate with the presentation aspects.

A number of utility functions are also provided. These are all static functions as they don't need an instance of the SharedLocation class to operate but are able to perform actions using only the state provided through their parameters. These include the functions to send and receive the LocationData messages, by serializing them to a byte array on transmission, and deserializing them back to objects when they are received.

Unique identifiers are also required to distinguish elements. This can be achieved using a counter which gets incremented each time a new identifier is requested. Identifiers are only requested on the server

which avoids having to coordinate these values across multiple clients. The use of a static function ensures that multiple sources of identifiers cannot exist on the server.

Networked applications on a single device identify themselves through the use of a port number. Typically port numbers in the range 1024-65000 are available for user space applications. Since the clients need to know the address of the server to connect to it, the port number part of the address is defined in this class and set for both clients and servers when the application is compiled. The hostname or IP number of the server is provided through the input field on the user interface before a client is started.

The elements in the shared reality are represented as a list of LocationData objects. The master copy exists on the server and is updated by messages from the various clients. The server then sends copies of this list to each client so that they can present these elements by adding visual elements at corresponding locations in the scene.

Both the client and server run multiple threads of execution; functions that are running concurrently (effectively at the same time). This prevents the application coming to a stop when one function needs to wait to send or receive information over the network. Bad things happen when two functions try to modify the same data at the same time. In particular, the list of LocationData objects representing the elements in the shared reality must be protected against simultaneous access. Mutex objects pause one thread if another thread already has access to the protected region. The protected region starts with a WaitOne operation and ends with a ReleaseMutex operation. All manipulation of shared variables must occur within the protected region. This region should be as small as possible to avoid slowing down the application, and also try to avoid invoking protected regions for other mutexes

since this can lead to deadlock where every threads waits for other threads to finish before they can get a turn in the protected area.

The contents of the SharedLocation script are provided in Algorithm 21.14.

Code Listing 21.14: SharedLocation. The core of the network manager functionality. These functions are used by both clients and servers to communicate, and to maintain the information represents the points of interest in the application.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using System.Net;
6  using System.Threading;
7  using System.IO;
8  using System.Net.Sockets;
9  using System;
10 using System.Runtime.Serialization.Formatters.Binary;

11
12 public class SharedLocation : MonoBehaviour
13 {
14     [Tooltip("A text field where the server can display
15     its own address.")]
16     public Text serverAddress;
17
18     [Tooltip("An input field to enter the address of a
19     server that the client connects to.")]
20     public InputField serverInput;
21
22     [Tooltip("A template for the object created to
23     represent a piece at a given location.")]
24     public GameObject pieceTemplate;
25
26     [Tooltip("A parent object for any items added by this
27     component. Children are deleted regularly so should be
28     an object created just for this purpose.")]
29     public GameObject boardState;
30
31     // The port number that the server uses.
32     static public Int32 port = 8080;
33
34     // Server name field written by any servers that are
35     // created.
36     static public string serverName = null;
37
38     // If this flag is set, then the scene is regenerated
39     // from the client info.
40     public static bool updateRequired = false;

```

```

34
35     // A counter used to assign identifiers.
36     private static int id = 99;
37
38     // Add one to the counter to get a new ID. Using a
39     // singleton
40     // pattern to ensure multiple counters don't get
41     // created.
42     private static int getUniqueID()
43     {
44         id += 1;
45         return id;
46     }
47
48     // Wait until a complete message has been received,
49     // extract
50     // the data, and return the message object.
51     static public LocationData receiveMessage(Stream
52     stream, Mutex receiveMutex)
53     {
54         BinaryFormatter formatter = new BinaryFormatter();
55         receiveMutex.WaitOne();
56         LocationData ld =
57         (LocationData)formatter.Deserialize(stream);
58         receiveMutex.ReleaseMutex();
59         return ld;
60     }
61
62     // Write the message to the given stream. This needs
63     // to be
64     // serialized, and the channel locked.
65     static public void sendMessage(LocationData message,
66     Stream stream, Mutex sendMutex)
67     {
68         MemoryStream ms = new MemoryStream();
69         new BinaryFormatter().Serialize(ms, message);
70         byte[] serializedData = ms.ToArray();
71
72         sendMutex.WaitOne();
73         stream.Write(serializedData, 0,
74         serializedData.Length);
75         sendMutex.ReleaseMutex();
76     }
77
78     // Find all objects with the given identifier in the
79     // database, and change
80     // the type to the given targetType.
81     static public void markObject (List<LocationData>
82     list, Mutex mutex, int identifier, LocationType
83     targetType)
84     {
85         // Ensure exclusive access to the list.
86         mutex.WaitOne();

```

```

76     foreach (LocationData ld in list)
77     {
78         if (ld.identifier == identifier)
79         {
80             ld.locType = targetType;
81         }
82     }
83     // Allow others to access the list.
84     mutex.ReleaseMutex();
85 }
86
87     // Add a location entry to one of the database. To be
88     // thread safe, we also require
89     // a mutex to lock the list before updating it.
90     static public void addLocation(List<LocationData>
91     list, Mutex mutex, LocationData ld)
92     {
93         // Ensure exclusive access to the list.
94         mutex.WaitOne();
95         // Assign identifiers if none are provided (value
96         // of -1)
97         if (ld.identifier < 0)
98         {
99             ld.identifier = getUniqueID();
100        }
101        // Update an existing entry if one is found.
102        bool found = false;
103        for (int i = 0; i < list.Count; i++)
104        {
105            if (list[i].identifier == ld.identifier)
106            {
107                list[i] = ld;
108                found = true;
109                break;
110            }
111        }
112        if (!found)
113        {
114            list.Add(ld);
115        }
116        // Allow others to access the list.
117        mutex.ReleaseMutex();
118    }
119
120    public void sendPosition(float lati, float longi)
121    {
122        activeClient.sendPosition (lati, longi);
123    }
124
125    public void sendMarker(float lati, float longi)
126    {
127        activeClient.sendMarker (lati, longi);
128    }

```

```

126
127 // The most recent client created.
128 private SharedLocationClient activeClient;
129
130
131 public void startClient()
132 {
133     activeClient = new SharedLocationClient
134     ↪ (serverInput.text);
135 }
136
137 public void startServer ()
138 {
139     SharedLocationServer s = new SharedLocationServer
140     ↪ ();
141 }
142
143 void Update()
144 {
145     // Any interaction with Unity has to be done
146     // in the main thread.
147
148     // Show server name in text box.
149     if (serverName != null)
150     {
151         serverAddress.text = serverName;
152     }
153
154     // Update the scene to reflect the
155     // elements provided from the server.
156     if (updateRequired)
157     {
158         activeClient.updateScene (boardState,
159         ↪ pieceTemplate);
160         updateRequired = false;
161     }
162 }
163
164 }
```

7. The client sends position updates for the participant associated with that client, drops markers at particular locations, and keeps a copy of the latest state of the shared virtual environment provided by the server. The client starts off by making a connection to the server before starting a separate thread of execution to deal with information provided by the server. This information comes in two forms, based on the communication protocol used:
  - (a) An IdSet message is a confirmation that the client's

position has been received and an identifier has been allocated.

- (b) Update messages contain the latest information about one of the elements in the scene.

Clients also send information when position changes are made or when markers are dropped. The communication channel is a shared resource and must be protected by mutexes when used. Since sending and receiving channels are separate these can be managed using two mutexes.

The client contains a utility function for updating the Unity scene based on the data it has available. This does break the separation between core client functionality and the presentation mechanism and could be removed to the SharedLocation class. However it does need protected access to the list of elements that the client maintains. The code for the client class is provided in Algorithm 21.15..

Code Listing 21.15: SharedLocationClient. The client side of the communication protocol is implemented through a thread to manage incoming messages, and functions to transmit updates and markers.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System.Net;
5  using System.Net.Sockets;
6  using System;
7  using System.Threading;

8
9  public class SharedLocationClient
10 {
11
12     // The connection to the server.
13     private TcpClient clientSocket = null;
14     // Mutex to ensure sending/receiving on the client
15     // socket are done one message at a time.
16     private Mutex clientSendMutex = new Mutex();
17     private Mutex clientReceiveMutex = new Mutex();

18     // An identifier, to be supplied by the server,
19     // identifying this client.
20     // -1 corresponds to no identifier yet assigned.
21     private int clientIdentifier = -1;

```

```
21      //  
22      private List<LocationData> clientDatabase = new  
23      private Mutex clientDatabaseMutex = new Mutex();  
24  
25      // The client actions which run in their own thread.  
26      This  
27      // handles the processing of any messages which are  
28      sent by  
29      // the server.  
30      public void handleClient(TcpClient clientSocket)  
31      {  
32          NetworkStream stream = clientSocket.GetStream();  
33  
34          while (true)  
35          {  
36              try  
37              {  
38                  LocationData ld =  
39                  SharedLocation.receiveMessage(stream,  
40                  clientReceiveMutex);  
41  
42                  switch (ld.status)  
43                  {  
44                      case MessageStatus.IdSet:  
45                          clientIdentifier = ld.identifier;  
46                          break;  
47                      case MessageStatus.Update:  
48                          SharedLocation.addLocation(clientDatabase,  
49                          clientDatabaseMutex, ld);  
50                          SharedLocation.updateRequired =  
51                          true;  
52                          break;  
53                      default:  
54                          Debug.Log("Unexpected message type  
55                          from server: " + ld.status);  
56                          break;  
57                  }  
58              }  
59              catch (Exception e)  
60              {  
61                  Debug.Log("Client Communication Failed : "  
62                  + e + " " + e.Message);  
63                  stream.Close();  
64  
65                  break;  
66              }  
67          }  
68      }  
69  
70      public SharedLocationClient(string host)
```

```
64      {
65          clientIdentifier = -1;
66          try
67          {
68              Debug.Log("Connecting to: " + host);
69              clientSocket = new TcpClient(host,
70      ↳ SharedLocation.port);
71          }
72          catch (Exception e)
73          {
74              Debug.Log("Client Exception: " + e.Message);
75          }
76          Thread t = new Thread(() =>
77      ↳ handleClient(clientSocket));
78          t.Start();
79      }
80
81      // Send the client's position to the server.
82      public void sendPosition(float lati, float longi)
83      {
84          if (clientSocket == null)
85              return;
86
87          LocationData ld = new LocationData();
88          ld.status = MessageStatus.Add;
89          ld.locType = LocationType.Participant;
90          ld.identifier = clientIdentifier;
91          ld.latitude = lati;
92          ld.longitude = longi;
93
94          SharedLocation.sendMessage (ld,
95      ↳ clientSocket.GetStream(), clientSendMutex);
96      }
97
98      // Add a new marker on the server at the given
99      // position.
100     public void sendMarker(float lati, float longi)
101     {
102         if (clientSocket == null)
103             return;
104
105         LocationData ld = new LocationData();
106         ld.status = MessageStatus.Add;
107         ld.locType = LocationType.Waypoint;
108         ld.identifier = -1;
109         ld.latitude = lati;
110         ld.longitude = longi;
111
112         SharedLocation.sendMessage (ld,
113     ↳ clientSocket.GetStream(), clientSendMutex);
114     }
```

```

112     // Convert the client database into a scene element,
113     // under the object boardState.
114     // Each element is created from the pieceTemplate, set
115     // to different colours according
116     // to its role.
117     // Needs to be run from the Unity main thread.
118     public void updateScene (GameObject boardState,
119     GameObject pieceTemplate)
120     {
121         foreach (Transform child in boardState.transform)
122         {
123             GameObject.Destroy(child.gameObject);
124         }
125
126         clientDatabaseMutex.WaitOne();
127
128         foreach (LocationData ld in clientDatabase)
129         {
130             GameObject g =
131             GameObject.Instantiate(pieceTemplate);
132             g.transform.position = new
133             Vector3(ld.longitude, ld.latitude, 0);
134             Color c = new Color();
135             switch (ld.locType)
136             {
137                 case LocationType.Participant: c = new
138                 Color(1, 0, 0); break;
139                 case LocationType.Waypoint: c = new
140                 Color(0, 1, 0); break;
141                 case LocationType.Defunct: c = new
142                 Color(0.4f, 0.4f, 0.4f); break;
143                 default: c = new Color(0, 0.5f, 0.5f);
144             }
145             g.GetComponent<MeshRenderer>().material.color
146             = c;
147             g.transform.SetParent(boardState.transform);
148         }
149         clientDatabaseMutex.ReleaseMutex();
150     }
151 }
```

8. The server maintains the list of elements in the scene, receiving updates and then transmitting the complete state to each client. It also keeps track of all connected clients. The sequence of steps in the server consist of:
  - (a) Starting a thread to listen on the port used by the application. New connections are picked up by this

thread.

- (b) Each new connection then has its own thread that receives messages with location information from that client, and any markers provided by that client. On receiving a message, the server thread for that connection will:
  - i. Update the entry in the list of elements (or create a new entry and assign an identifier if none exists).
  - ii. Reply with the identifier if the message was the updated location of a participant.
  - iii. Send the complete list of all elements to every client.

The number of threads involved and the amount of access to shared information (the list of elements in the scene, the list of clients connected) require mutexes to protect the regions in the code when this shared information is manipulated.

Exceptions may get through by the sending and receiving operations if one of the clients disconnect. When this happens the client is removed from the list of all clients, any objects created by that client are marked as defunct (and will appear in a different colour on the clients) and the thread for that client is ended.

The script for the server class is provided in Algorithm 21.16.

Code Listing 21.16: SharedLocationServer. The server class manages the location information for all elements created by the various clients, and also manages the list of which clients are active.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System.Threading;
5  using System.Net;
6  using System.Net.Sockets;
7  using System;
8
9  public class SharedLocationServer
10 {

```

```
11   // The master list of all objects maintained by the
12   // server.
13   private List<LocationData> serverDatabase = new
14   new List<LocationData>();
15   // Since the server runs on multiple threads, all access
16   // to the database must be locked by a mutex before use.
17   private Mutex serverDatabaseMutex = new Mutex();
18
19
20
21   // The list of all connections that are operational, so
22   // that properties of an individual connection can be
23   // identified.
24   private List<ConnectionInfo> connections = new
25   new List<ConnectionInfo>();
26   // The list of connections is shared between threads, so
27   // must be locked by a mutex before use.
28   private Mutex connectionsMutex = new Mutex();
29
30
31   // Details an individual connection from a client.
32   private class ConnectionInfo
33   {
34     // The connected socket to a client.
35     public TcpClient socket;
36     // A mutex to ensure only one thread is
37     // reading/writing a socket at any time.
38     // This ensures that messages from different processes
39     // don't get intermingled.
40     public Mutex sendMutex = new Mutex();
41     public Mutex receiveMutex = new Mutex();
42     // List of identifiers for objects associated with
43     // this connection.
44     public List<int> myObjects = new List<int>();
45   }
46
47
48   // https://stackoverflow.com/questions/6803073/get-local-
49   // -ip-address
50   public static string getLocalIPAddress()
51   {
52     var host = Dns.GetHostEntry(Dns.GetHostName());
53     foreach (var ip in host.AddressList)
54     {
55       if (ip.AddressFamily == AddressFamily.InterNetwork)
56       {
57         return ip.ToString();
58       }
59     }
60     throw new Exception("No network adapters with an IPv4
61     // address in the system!");
62   }
63
64
65   private void removeConnection(ConnectionInfo ci)
66   {
67     foreach (int id in ci.myObjects)
68     {
```

```

52     SharedLocation.markObject(serverDatabase,
53     serverDatabaseMutex, id, LocationType.Defunct);
54   }
55   connectionsMutex.WaitOne();
56   connections.Remove(ci);
57   connectionsMutex.ReleaseMutex();
58 }
59
60 // Send a complete copy of the server database to all of
61 // the clients.
62 // This is not particularly efficient and could be
63 // improved by sending
64 // updates only where they are required (such as when an
65 // object changes
66 // location).
67 private void sendUpdatesToAllClients()
68 {
69   // Make a copy of the connections list so we don't
70   // have to lock it for an extended period.
71   List<ConnectionInfo> connectionsCopy;
72
73   connectionsMutex.WaitOne();
74   connectionsCopy = new
75   List<ConnectionInfo>(connections);
76   connectionsMutex.ReleaseMutex();
77
78   foreach (ConnectionInfo ci in connectionsCopy)
79   {
80     // Send a copy of the database to this client.
81     serverDatabaseMutex.WaitOne();
82
83     try
84     {
85       foreach (LocationData ld in serverDatabase)
86       {
87         ld.status = MessageStatus.Update;
88
89         SharedLocation.sendMessage(ld,
90         ci.socket.GetStream(), ci.sendMutex);
91       }
92     }
93     catch (Exception e)
94     {
95       Debug.Log("Server update Failed : " + e + " " +
96       e.Message);
97     }
98
99     serverDatabaseMutex.ReleaseMutex();
100   }
101 }
```

```

97    // Handle all interactions with a single client. This
98    // runs on its own thread to avoid holding up the rest
99    // of the application.
100   // It responds to messages from the client, and sends
101   // updates.
102   private void handleServerMessages(ConnectionInfo ci)
103   {
104       NetworkStream stream = ci.socket.GetStream();
105       while (true)
106       {
107           try
108           {
109               LocationData ld =
110                   SharedLocation.receiveMessage(stream, ci.receiveMutex);
111                   // add to database.
112                   SharedLocation.addLocation(serverDatabase,
113                   serverDatabaseMutex, ld);
114                   if (!ci.myObjects.Contains(ld.identifier))
115                   {
116                       ci.myObjects.Add(ld.identifier);
117                   }
118
119                   if (ld.locType == LocationType.Participant)
120                   {
121                       // reply back to client.
122                       ld.status = MessageStatus.IdSet;
123                       SharedLocation.sendMessage(ld, stream,
124                       ci.sendMutex);
125                   }
126
127                   sendUpdatesToAllClients();
128
129               }
130
131               catch (Exception e)
132               {
133                   Debug.Log("Communication Failed : " + e + " " +
134                   e.Message);
135                   stream.Close();
136                   // remove connection.
137                   removeConnection(ci);
138
139                   break;
140               }
141
142   }
143
144   // The activity of the server, run in its own thread.
145   // Listens for connections, and then creates an
146   // additional thread to handle communications on each
147   // new connection.
148   private void server()
149   {

```

```
143 TcpListener server = null;
144
145 try
146 {
147     server = new TcpListener(IPAddress.Any,
148     SharedLocation.port);
149     // Report server in case anyone wants to see its
150     // address.
151     SharedLocation.serverName = getLocalIPAddress();
152
153     server.Start();
154
155     while (true)
156     {
157         // Wait for a new client. This will block until
158         // a client connects.
159         TcpClient client = server.AcceptTcpClient();
160
161         // Create a record of the new connection received.
162         ConnectionInfo ci = new ConnectionInfo();
163         ci.socket = client;
164
165         // Add it to a list of connections. This is also
166         // manipulated in other threads so must be locked
167         // before being changed.
168         connectionsMutex.WaitOne();
169         connections.Add(ci);
170         connectionsMutex.ReleaseMutex();
171
172         // Create a thread for this client to handle client
173         // specific communications.
174         Thread t = new Thread(() =>
175             handleServerMessages(ci));
176         t.Start();
177     }
178     catch (Exception e)
179     {
180         Debug.Log("Server Exception: " + e.Message);
181     }
182
183     server.Stop();
184 }
185
186 public SharedLocationServer()
187 {
188     Thread t = new Thread(server);
189     t.Start();
190 }
```

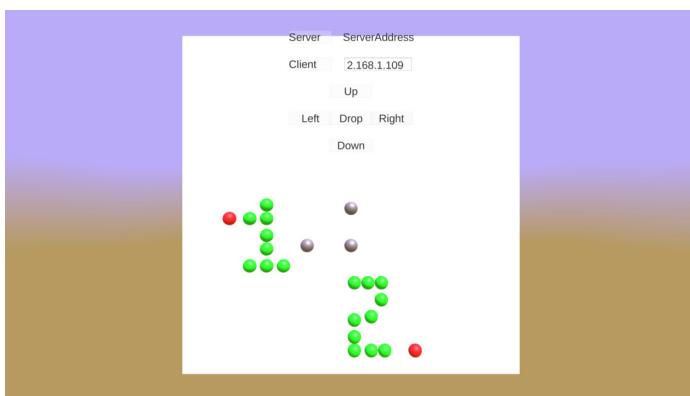


Figure 21.5.2: Two participating clients (and one that joined and left) leaving trails in a shared environment.

## 21.6 Enabling persistent augmented reality experiences and shared location recognition using cloud anchors

### Component 21.6.1: Cloud anchors.

Cloud  
Anchors

*Augmented reality applications capture large amounts of data in the process of tracking feature points and markers, and in performing localization and mapping of the environments in which activities take place. All this work is then lost when the session comes to an end. Better value can be achieved through persistence (ensuring that the data is available when the application resumes) and sharing (allowing other participants to benefit from your efforts).*

*Central storage (in the cloud) of this data would potentially achieve both of these goals. This example demonstrates a starting point for this process; the cloud anchor facility provided with AR Core. This still has many limitations that unnecessarily complicate the demonstration.*

*Cloud anchors identify particular trackable features. They allow these features to be shared among participants, currently for up to 1 day. This achieves some of the persistence and sharing opportunities that are desired. However sharing identifiers for the anchors themselves, and any application specific association, is still the problem of the application developer.*

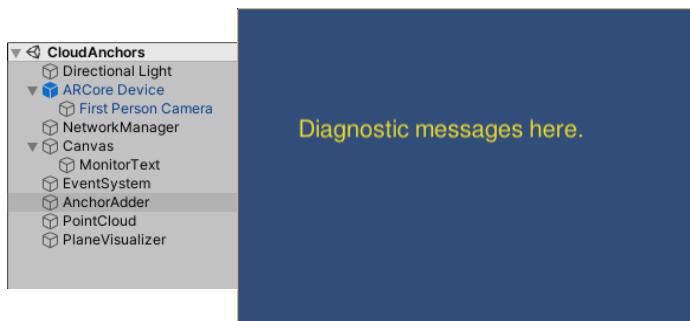


Figure 21.6.1:  
Persistent locations  
with cloud anchors.

1. Start with a new project. Set this up to build on the mobile device. Enable AR Core (under the Player Settings/XR Settings). Import the AR Core SDK for Unity package into the project.

Remove the Main Camera from the scene and replace it with an ARCoreDevice prefab from the imported assets.

The cloud anchor service requires an API key to access.

Create an API key by visiting the APIs & Services→Credentials panel in GCP Console <https://console.cloud.google.com/apis/credentials?refresh=1&pli=1> and signing in (create an account if required). Use the controls provided to create an API key. The ARCore Cloud Anchor API also needs to be enabled on your account by visiting <https://console.cloud.google.com/apis/library/arcorecloudanchor.googleapis.com?project=projectbuilder-204123&folder&organizationId>, and pressing the enable button (or search for the ARCore Cloud Anchor API in the Dashboard). This also provides a dashboard for monitoring content uploaded.

The API key is added to your project under the AR Core settings (usually Edit/Project Settings/AR Core). The ARCoreDevice also have a configuration property which needs to be opened. Enable cloud anchors in this configuration window.

2. The application needs to handle its own networking to ensure exchange of the anchor list. Create a new empty object in the scene called network manager.

Add the built-in components: NetworkManager and NetworkManagerHUD.

The networking system requires a player equivalent object which is created on the client, and replicated on the server and other clients. This example uses a analogous object that allows the participant to create anchors on trackable elements, register these with the cloud service, and then share the resulting cloud anchor identifier with the server and thus with the other clients.

Create a new cube object and name this AnchorManager. This needs a NetworkIdentity component added to it so that it can be managed by the network manager. Also create a new C# script called AnchorList and attach this component to the anchor manager. Drag the anchor manager into the Prefabs folder in the Assets area and remove it from the scene.

In the network manager object, set the Player Prefab (under Spawn Info) to be the AnchorManager prefab.

3. It is helpful in this case to have a text object visible on the screen to display progress update messages. Create a new UI/Text object for this purpose.
4. Visualization of the trackable points can be achieved by reusing components created for some of the example projects in the AR Core SDK. Add a sphere object to the scene and name it PointCloud. Add the PointCloudVisualizer script from the GoogleARCore/Examples/Common/Scripts to this object. Add the PointCloud material from GoogleARCore/Examples/Common/Materials to the object as well.

This ensures that target points are displayed on the mobile device, which also helps confirm that the tracking is working correctly.

Likewise, add a sphere object named PlaneVisualizer. Add the Detected Plane Generator component to this,

and give it a Detected Plane Visualizer prefab (from the AR Core examples) as its property.

5. Interaction with the trackable points and planes is managed by a new C# component called AnchorInteraction. Create a new empty object called AnchorAdder and add the AnchorInteraction component to this object. The script for the anchor interaction component is provided in Algorithm 21.17. This checks for touches on the screen of the mobile device, and tries to find a trackable object on the ray from eye point through the touch into the virtual scene. If a suitable trackable is found it adds a marker to the anchor associated with that marker, and calls a function on the AnchorList script to start the process of registering that anchor with the cloud service.

Code Listing 21.17: AnchorInteraction. Anchors are added by identifying suitable trackable objects from the scene reconstruction performed by AR Core.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using GoogleARCore;
6  using GoogleARCore.CrossPlatform;
7  using System.IO;
8
9  public class AnchorInteraction : MonoBehaviour {
10
11    public AnchorList anchorList;
12    public Camera firstPersonCamera;
13    public Text updateMessage;
14
15    void Update () {
16      Touch touch;
17      if (Input.touchCount < 1 || (touch =
18        Input.GetTouch(0)).phase != TouchPhase.Began)
19      {
20        return;
21      }
22
23      updateMessage.text += "Touch at: " + touch.position;
24
25      // Screen touched, find a trackable in the image under
26      // the touch point.
27      TrackableHit hit;
```

```

26     if (Frame.Raycast(touch.position.x, touch.position.y,
27         TrackableHitFlags.PlaneWithinPolygon
28         |TrackableHitFlags.FeaturePointWithSurfaceNormal, out
29         hit))
30     {
31         if (!(hit.Trackable is DetectedPlane) ||
32             Vector3.Dot(firstPersonCamera.transform.position -
33             hit.Pose.position, hit.Pose.rotation * Vector3.up) >=
34             0)
35         {
36             // Hit a point or front of a plane.
37             updateMessage.text += "Hit " + anchorList;
38
39             // Create a local anchor on that trackable.
40             Component anchor =
41                 hit.Trackable.CreateAnchor(hit.Pose);
42             if (anchorList != null)
43             {
44                 string label = anchorList.getLabel ();
45                 updateMessage.text += "Adding anchor " + label;
46                 // Attach object to the anchor.
47                 anchorList.addInstanceToAnchor (label,
48                     anchor.transform, hit.Pose.position,
49                     hit.Pose.rotation);
50
51                 // Copy the anchor to the cloud.
52                 anchorList.createCloudAnchor (label, (Anchor)
53                     anchor);
54             }
55         }
56     }
57 }
58 }
```

A visible marker is also instantiated each time a anchor is created. Create a new object to represent this marker (a small 0.1 scale capsule would work for this). Also add a TextMesh object as a child of the marker, to hold a label that identifies the anchor. Drag this marker into the Prefabs folder as well, and remove it from the scene.

Note that the Anchor List property of the Anchor Interaction component does not need to be filled in. The Anchor Manager object does this when the local player object is spawned on the client.

6. The remainder of this example describes how the AnchorList component functions. The complete code for the anchor list is provided in Algorithm 21.18.

Code Listing 21.18: AnchorList. Anchor information is maintained both in the cloud service (scene related information) and in the client-server system which shares anchor identifiers with other participants. Persistence is achieved by also writing anchor identifiers to file on the server.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using System.IO;
4  using UnityEngine;
5  using UnityEngine.Networking;
6  using UnityEngine.UI;
7  using GoogleARCore;
8  using GoogleARCore.CrossPlatform;

9
10 // Keep a list of cloud anchor IDs and have these shared
11 // between
12 // all clients.
13 public class AnchorList : NetworkBehaviour {
14
15     // The details of an anchor: the cloud ID, and the
16     // custom properties of the object at that ID.
17     public class AnchorDetails : MessageBase
18     {
19         public string anchor;
20         public string objectAtAnchor;
21     }
22
23     // Debugging/logging message.
24     private Text monitor;
25
26     [Tooltip ("The object created to mark each anchor.
27     Should have a text child object")]
28     public GameObject markerPrefab;
29
30     // Some autogenerated text to label each marker in
31     // sequence.
32     private string counterName;
33     private int counter = 0;
34
35     // The list of known anchors (both on server and each
36     // client).
37     private List<AnchorDetails> anchorDetails = new
38     List<AnchorDetails> ();
39
40     // A message type used to send updates to each client.
41     private short AnchorMessageType = MsgType.Highest + 1;
42
43     void Awake ()

```

```
39     {
40         // Create a new label for markers that is unique to
41         // this session.
42         string consonants = "bcdfghjklmnpqrstwz";
43         string vowels = "aeiouy";
44         counterName = "" + System.Char.ToUpper
45             (consonants[Random.Range (0, consonants.Length)]) +
46             vowels[Random.Range (0, vowels.Length)] +
47             consonants[Random.Range (0, consonants.Length)]);
48     }
49
50     // Get a new ID for each marker created.
51     public string getLabel ()
52     {
53         counter += 1;
54         return counterName + " " + counter;
55     }
56
57     // Copy an anchor into the cloud service. Call
58     // havePlaced when complete.
59     public void createCloudAnchor (string label, Anchor
60         anchor)
61     {
62         monitor.text += "Adding anchor: Step 1";
63         AsyncTask <CloudAnchorResult> done =
64             XPSession.CreateCloudAnchor ((Anchor) anchor);
65         monitor.text += "Adding anchor: Step 2";
66         done.ThenAction (x => havePlaced (label, x));
67         monitor.text += "Adding anchor: Step 3";
68     }
69
70     // Confirm that the anchor has been placed.
71     public void havePlaced (string label, CloudAnchorResult
72         result)
73     {
74         // Send the anchor details to the server.
75         monitor.text += "Adding anchor: Step 4";
76         Debug.Log ("Anchor result: " + result);
77         Debug.Log ("Anchor IDA: " + result.Anchor + " " +
78             result.Response);
79         Debug.Log ("Anchor ID: " + result.Anchor.CloudId);
80         CmdAddAnchor (result.Anchor.CloudId, label);
81         monitor.text = "Anchor added to cloud: " +
82             result.Response + "\n" + result.Anchor.CloudId;
83     }
84
85     // Make sure cloud anchors are persistent. On the
86     // server, read all the anchors from file.
87     private void retrieveAnchors ()
88     {
89         string anchorID = "";
90         string objectID = "";
```

```

81     try
82     {
83         StreamReader file = new StreamReader
84             (Application.persistentDataPath + "/" + "cid.txt");
85         while ((anchorID = file.ReadLine ()) != null)
86         {
87             objectID = file.ReadLine ();
88             updateList (anchorID, objectID);
89             monitor.text = "Retrieving anchor " + anchorID;
90         }
91         file.Close ();
92     }
93     catch (System.Exception)
94     {
95         // file does not exist yet.
96     }
97
98     // Make sure cloud anchors are persistent. On the
99     // server, write them all to file.
100    private void persistList ()
101    {
102        if (isServer)
103        {
104            StreamWriter file = new StreamWriter
105                (Application.persistentDataPath + "/" + "cid.txt",
106                 false);
107            if (monitor != null)
108            {
109                monitor.text = "";
110            }
111            for (int i = 0; i < anchorDetails.Count; i++)
112            {
113                file.WriteLine (anchorDetails[i].anchor);
114                file.WriteLine (anchorDetails[i].objectAtAnchor);
115                if (monitor != null)
116                {
117                    monitor.text += "Written: " +
118                     anchorDetails[i].anchor + "\n";
119                }
120            }
121            file.Close ();
122        }
123
124        // When the server starts, read previously known anchors
125        // from file.
126        public override void OnStartServer()
127        {
128            if (isServer)
129            {
130                retrieveAnchors ();
131            }
132        }

```

```
128     }
129
130     public override void OnStartLocalPlayer ()
131     {
132         // Find the debugging message.
133         monitor = GameObject.Find ("MonitorText").GetComponent
134         <Text> ();
135         Debug.Log ("Monitor: " + monitor + " " +
136         isLocalPlayer);
137
138         if (isLocalPlayer)
139         {
140             // Find the component that adds anchors, and inform
141             // them where to find this service.
142             AnchorInteraction anchorAdder = GameObject.Find
143             ("AnchorAdder").GetComponent <AnchorInteraction> ();
144             anchorAdder.anchorList = this;
145             monitor.text += "Added AnchorList";
146         }
147     }
148
149     // When the client starts, set up the event handler for
150     // new anchor messages.
151     public override void OnStartClient()
152     {
153         if (isClient)
154         {
155             NetworkManager.singleton.client.RegisterHandler
156             (AnchorMessageType, clientUpdate);
157         }
158     }
159
160     // When a new anchor ID is received, retrieve it from
161     // the cloud.
162     private void clientUpdate (NetworkMessage m)
163     {
164         AnchorDetails amt = m.ReadMessage <AnchorDetails> ();
165         if (findInList (amt.anchor) == null)
166         {
167             monitor.text = "Retrieving cloud anchor " +
168             amt.anchor;
169             AsyncTask <CloudAnchorResult> done =
170             XPSession.ResolveCloudAnchor (amt.anchor);
171             done.ThenAction (x => gotAnchor (amt.objectAtAnchor,
172             x));
173
174             updateList (amt.anchor, amt.objectAtAnchor);
175         }
176     }
177
178     // When a response from the cloud service is received,
179     // create the marker.
```

```

169  public void gotAnchor (string label, CloudAnchorResult
170    → result)
171  {
172    if (result.Response == CloudServiceResponse.Success)
173    {
174      monitor.text = "Got anchor " + result.Response +
175      "\n" + result.Anchor.CloudId;
176
177      GameObject obj = addInstanceToAnchor (label,
178      → result.Anchor.transform,
179      → result.Anchor.transform.position,
180      → result.Anchor.transform.rotation);
181      obj.GetComponent <MeshRenderer> ().material.color =
182      new Color (0,1,0);
183    }
184  }
185
186  // Create a new marker.
187  public GameObject addInstanceToAnchor (string label,
188    → Transform parent, Vector3 position, Quaternion
189    → orientation)
190  {
191    GameObject obj = Instantiate(markerPrefab, position,
192    → orientation);
193    obj.GetComponentInChildren <TextMesh> ().text = label;
194    obj.transform.parent = parent;
195    return obj;
196  }
197
198  // Send updates every few seconds.
199  private float timer = 0;
200
201  void Update () {
202    timer += Time.deltaTime;
203    if (timer > 3.0f)
204    {
205      if (isServer)
206      {
207        // Send anchors to all clients.
208        for (int i = 0; i < anchorDetails.Count; i++)
209        {
210          NetworkServer.SendToAll (AnchorMessageType,
211          anchorDetails[i]);
212        }
213      }
214    }
215  }
216
217  // Look for a given anchor ID in the list of anchors.
218  public AnchorDetails findInList (string details)
219  {
220    for (int i = 0; i < anchorDetails.Count; i++)
221    {

```

```

212         if (anchorDetails[i].anchor == details)
213     {
214         return anchorDetails[i];
215     }
216     return null;
217 }
218
219
220 // Update the anchor ID, create a new one if it does not
221 // exist.
221 public void updateList (string details, string
222 identifier)
223 {
224     AnchorDetails d = findInList (details);
225     if (d == null)
226     {
227         d = new AnchorDetails ();
228         d.anchor = details;
229         anchorDetails.Add (d);
230     }
231     d.objectAtAnchor = identifier;
232 }
233
234 [Command]
235 private void CmdAddAnchor (string anchor, string
236 identifier)
237 {
238     updateList (anchor, identifier);
239     persistList ();
240 }

```

The sequence of actions that occurs is as follows:

- An anchor is added to a trackable object in the Anchor Interaction script. The initial anchor that is created is a local anchor that is valid only for the particular device being used. The Anchor Interaction script gets a unique label from the Anchor List component and invokes a function on the Anchor List component to create a new cloud anchor. Cloud anchors are stored on the cloud service, currently for about a day, and contain the scene information that identifies a particular tracked position so that local anchors can be created on other devices. The details are all handled by the AR Core SDK. The software only needs to keep track of what anchors exist. Each

cloud anchor is identified by an identifier (a very long hexadecimal string).

---

```
1 anchorList.createCloudAnchor (label, (Anchor)
    anchor);
```

---

- (b) The anchor is then passed to the AR Core SDK to register the anchor with the cloud service. This may take some time as potentially some detailed information about the scene around the tracked object is uploaded. The function havePlaced is set to be called when this operation has completed.

---

```
1 AsyncTask <CloudAnchorResult> done = XPSession.
    CreateCloudAnchor ((Anchor) anchor);
2 done.ThenAction (x => havePlaced (label, x));
```

---

- (c) Once the anchor is uploaded to the cloud, confirmation is received in the form of an identifier for that cloud anchor. This is then sent to the local server which maintains the list of all cloud anchor identifiers.

---

```
1 CmdAddAnchor (result.Anchor.CloudId, label);
```

---

- (d) The server updates the list of cloud anchor identifiers to include the new identifier. It also writes the updated list out to a file, so that the anchor identifiers are still accessible even if the server needs to be restarted.

---

```
1 updateList (anchor, identifier);
2 persistList ();
```

---

- (e) At regular intervals the server sends a message to all clients (including the one which created the anchor in the first place) to let it know about all the cloud anchor identifiers. This allows clients that have just joined to catch up.

---

```
1 for (int i = 0; i < anchorDetails.Count; i++)
2 {
3     NetworkServer.SendToAll (AnchorMessageType,
        anchorDetails[i]);
4 }
```

---

- (f) When a client receives a message about a cloud anchor identifier, it checks to see if it already has a marker for that identifier. If not, it calls the AR Core SDK to retrieve the details of that cloud anchor. This can also take some time, so the handler `gotAnchor` is scheduled to be invoked when the details of the anchor have been received.

---

```

1  AsyncTask <CloudAnchorResult> done = XPSession.
   ResolveCloudAnchor (amt.anchor);
2  done.ThenAction (x => gotAnchor (amt.
   objectAtAnchor, x));

```

---

- (g) Once the anchor details have been received (by the AR Core SDK) the AR Core SDK provides enough information about the pose of the anchor for a new marker to be added to that location.

---

```

1  GameObject obj = addInstanceToAnchor (label,
   result.Anchor.transform, result.Anchor.
   transform.position, result.Anchor.transform.
   rotation);

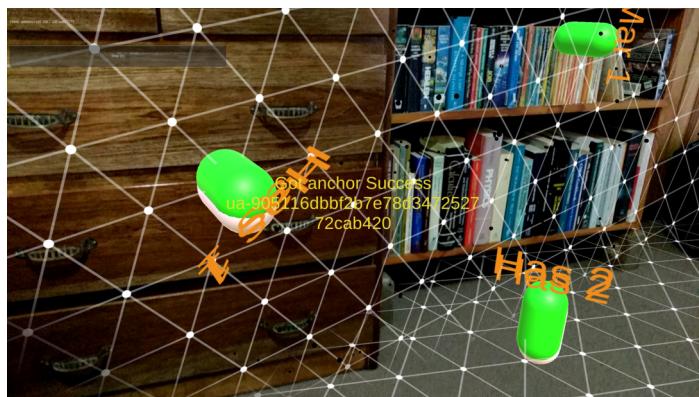
```

---

7. When testing this example run the server on one device (for example, the desktop platform) and start each device as a client. Touch any tracked surface to add a new anchor. Each client should be able to lay down their own trail of markers, automatically named with a unique code and sequence number. These markers may take some time to add to the cloud service (several seconds when last tested) but are set to be replaced with green markers once this has been completed. Other clients should be able to see these markers once they are added provided those clients are in the same location and can see the same trackable objects.

This does have potential as the basis for a treasure hunt (or paper chase) style experience. The application could also be extended to allow participants to add or remove markers in a common area, for example to play a game of virtual chess.

Figure 21.6.2: Local and cloud anchors.



## 21.7 Legacy multi-participant collaboration using the now obsolete UNet

### Component 21.7.1: Mutual presence.

Mutual  
Presence

This example develops an experience supporting synchronous mutual presence for both physically colocated and virtual participants around an anchored augmented overlay. This component is typical of many shared augmented reality experiences such as a multi-player game, or a distributed meeting. This experience assumes a fixed marker present in the physical environment to serve as the reference point to place the augmented content.

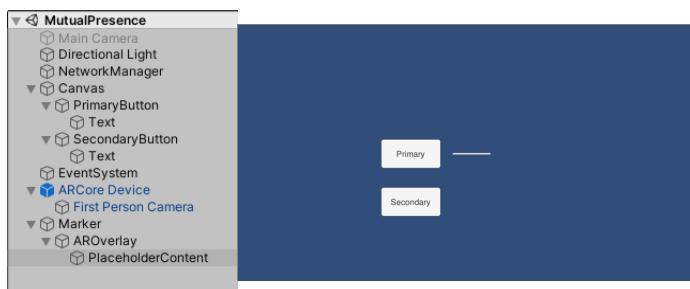


Figure 21.7.1:  
Providing a synchronous shared experience.

1. Start with a new project. Leave the build settings active for the desktop platform. This is used boost the number of devices (even this involves multiple copies of

the application running on the same machine) that can participate while testing during the development of the experience. The switch to a mobile platform and use of appropriate augmented reality SDKs occurs in a later step.

2. The bulk of the scene content is provided by the physical view from the device's camera. As such the only elements to be provided are the objects that make up the augmented reality overlay, and the representations of the various participants.

Start with the content for the overlay. Create a new empty object to act as the parent for all of this content, named AROverlay. This makes it easier to relocate all of this content to match the position of a marker during future steps. Place a cube as child of the AROverlay to act as a placeholder until this is developed in a later step.

Also create an empty object to act as the parent for the avatar (named Avatar). Given the limited controls that may be available for the participants in the experience we'll assume that all participants will have viewpoint tracking (required to view and manipulate augmented content) and that some may have a controller device. Create head and hand child objects for the avatar from externally sourced 3D models and name them Head and Hand respectively. Set the offset of the head to 0, 0, 0 and that of the hand to an appropriate position relative to the head.

Some manual control of the avatar is convenient during development, so add in the simulated location tracking component developed in component 18.1.1. Remember to add in the Depth axis to the Input Manager.

3. The networking infrastructure uses the facilities provided by the Unity software despite their deprecation. Since this uses the standard client-server structure it can be replaced by any other implementation of the

same pattern in future. One device runs the server which collects all updates from each clients, manages any centrally coordinated activities, and distributes changes to the clients. Clients handle interaction and presentation.

Create a new empty object named NetworkManager and add the built-in components for NetworkManager and NetworkManagerHUD.

Add a NetworkTransform and NetworkIdentity (set the LocalPlayerAuthority property) to the avatar object, and convert this into a prefab. Remove the avatar from the scene. Add the avatar prefab to the Player Prefab property (under Spawn Info) of the NetworkManager component of the Network Manager object.

The control script for the avatar needs to be updated to ensure that it doesn't control every avatar in the scene. Change the script to derive from NetworkBehaviour rather than MonoBehaviour (you may need to also import the UnityEngine.Networking library). Ensure that control information is only processed if the isLocalPlayer variable is set.

---

```

1  if (isLocalPlayer)
2  {
3      processControls ();
4 }
```

---

The basic networking functionality can be tested at this point by building the application (as a desktop version) and running several instances of it. Select one to act as the server (or host which includes server functionality) and the other clients to connect to the server (using the address of the device that the server is running on, or localhost to refer to the local machine if everything is running on the same device).

4. Interaction with virtual objects is achieved through pointer based controls. On devices that only have gaze directed interaction the pointer is controlled by gaze, with buttons to activate since it is assumed that the

viewing device is handheld. Devices with separate controllers can use the direction of the controller and controller buttons to achieve the same outcome.

The gaze controller works for both the desktop context and for the single mobile device scenario. Add a laser beam (cylinder) to the avatar. This beam is emitted from the center of the face. Also place two buttons (UI/Button) on the screen. These need to be named LaserBeam, PrimaryButton and SecondaryButton so that the gaze controller script can find them. Add a new C# script called GazeController to the avatar prefab and provide it with the code given in Algorithm 21.19.

Code Listing 21.19: GazeController. The gaze controller uses the direction of the viewing device, and two buttons placed on the screen, to provide a pointer based controller.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.UI;
5  using UnityEngine.Networking;
6  using UnityEngine.EventSystems;
7
8  public class GazeController : NetworkBehaviour {
9
10    // The laser beam object visible on the local player
11    private GameObject beam;
12    // The primary button for the local player.
13    private GameObject button1;
14    // The secondary button for the local player.
15    private GameObject button2;
16
17    // The state of the primary button.
18    private bool button1Pressed;
19    // The state of the secondary button.
20    private bool button2Pressed;
21
22    // The camera if run on a desktop.
23    private GameObject defaultCamera;
24    // The camera if run on an AR capable device.
25    private GameObject ARCamera;
26
27    void Start ()
28    {
29        defaultCamera = GameObject.Find ("Main Camera");
30        ARCamera = GameObject.Find ("First Person Camera");
```

```
31
32 // Find child beam.
33 beam = transform.Find ("Head/LaserBeam").gameObject;
34 beam.SetActive (false);
35
36 if (isLocalPlayer)
37 {
38     EventTrigger trigger;
39     EventTrigger.Entry pointerDown;
40     EventTrigger.Entry pointerUp;
41
42     // Find in scene.
43     button1 = GameObject.Find ("PrimaryButton");
44
45     trigger =
46     ↵ button1.gameObject.AddComponent<EventTrigger>();
47     pointerDown = new EventTrigger.Entry();
48     pointerDown.eventID = EventTriggerType.PointerDown;
49     pointerDown.callback.AddListener((e) =>
50     ↵ button1Pressed = true);
51     trigger.triggers.Add(pointerDown);
52     pointerUp = new EventTrigger.Entry();
53     pointerUp.eventID = EventTriggerType.PointerUp;
54     pointerUp.callback.AddListener((e) => button1Pressed
55     ↵ = false);
56     trigger.triggers.Add(pointerUp);
57
58     button2 = GameObject.Find ("SecondaryButton");
59
60     trigger =
61     ↵ button2.gameObject.AddComponent<EventTrigger>();
62     pointerDown = new EventTrigger.Entry();
63     pointerDown.eventID = EventTriggerType.PointerDown;
64     pointerDown.callback.AddListener((e) =>
65     ↵ button2Pressed = true);
66     trigger.triggers.Add(pointerDown);
67     pointerUp = new EventTrigger.Entry();
68     pointerUp.eventID = EventTriggerType.PointerUp;
69     pointerUp.callback.AddListener((e) => button2Pressed
70     ↵ = false);
71     trigger.triggers.Add(pointerUp);
72
73     setActive (true);
74
75 #if UNITY_ANDROID && !UNITY_EDITOR
76     defaultCamera.SetActive (false);
77     ARCamera.SetActive (true);
78 #else
79     defaultCamera.SetActive (true);
80     ARCamera.SetActive (false);
81     transform.position = new Vector3 (0, 1.5f, 0);
82     defaultCamera.transform.SetParent (transform.Find
83     ↵ ("Head"));
```

```

77         defaultCamera.transform.localPosition = new Vector3
    ↵     (0, 0.1f, 0);
78     #endif
79     }
80 }
81
82     void Update ()
83 {
84 #if UNITY_ANDROID && !UNITY_EDITOR
85     // Calculate position to set the avatar so that head
    ↵ is at camera position.
86     // Small offset so the beam is visible.
87     if (isLocalPlayer)
88     {
89         transform.position = ARCamera.transform.position +
    ↵ new Vector3 (0, 0.1f, -0.2f);
90         transform.rotation = ARCamera.transform.rotation;
91     }
92 #endif
93 }
94
95     public Ray getRay ()
96 {
97     return new Ray (beam.transform.position,
    ↵ beam.transform.forward);
98 }
99     public bool getPrimary ()
100 {
101     return button1Pressed;
102 }
103     public bool getSecondary ()
104 {
105     return button2Pressed;
106 }
107
108     public void setActive (bool active)
109 {
110     if (isLocalPlayer)
111     {
112         beam.SetActive (active);
113         button1.SetActive (active);
114         button2.SetActive (active);
115     }
116 }
117 }
```

The bulk of this script is involved in providing buttons that can remain pressed for an extended period of time. The default click based semantics triggers events after a paired press and release action. Extra event handlers can be added to the button to detect just the press

and just the release event. The script creates handlers for these two events to change the value of a boolean variable representing the state of the button.

Ironically later components try to restore the click functionality by debouncing the buttons. This involves waiting for the button to enter a release state before allowing a second press event.

The script also provides specific behaviours for when it is used on a desktop system or under the Unity software Editor. In those cases it disables the AR Core camera and replaces it with the standard scene camera. When used on a mobile device it copies the pose of the AR camera to that of the avatar (the object using the gaze controller script) so that the avatar follows the tracked device.

5. The actual interaction process intended for the application makes use of one of the controller devices provided. In this case it is the gaze controller but similar functionality could be achieved using a separate pointer based controller such as that developed in component 19.3.1. In this component, the controllers are used to build structures by placing blocks on one of the visible sides of an existing block in the scene. Aiming the controller at a block in the scene highlights both the block selected and outlines the position of a new block. The first button on the controller adds the new block, while the second removes the original block.

Ensure that the scene contains a starting block by modifying the existing cube object to become the base layer. Create a new tag called ActiveBlock which is applied to any object that can have blocks placed next to it.

Create a new C# script called BlockPlacer and add it to the avatar prefab. The code for this script is provided in Algorithm 21.20..

Code Listing 21.20: BlockPlacer. Block are placed adjacent to one of the faces of an existing block that is visible to the controller. Blocks are placed at discrete positions to ensure that they fit together neatly.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Networking;
5
6  public class BlockPlacer : NetworkBehaviour {
7
8      public GameObject blockTemplate;
9      public float blockSize = 0.1f;
10
11     private GameObject parentObject;
12
13     private GameObject blockHint;
14     public Material hintMaterial;
15     private GazeController controller;
16
17     private GameObject currentActive = null;
18
19     // Have to let the button go before it will reactivate.
20     private bool debouncePrimary = false;
21     private bool debounceSecondary = false;
22
23     void Start ()
24     {
25         // Indirect find, since AROverlay is not active at
26         // this stage and Find won't work directly.
27         parentObject = GameObject.Find
28             ("Marker").transform.Find ("AROverlay").gameObject;
29
30         controller = GetComponent <GazeController> ();
31
32         blockHint = Instantiate (blockTemplate);
33         blockHint.transform.localScale = new Vector3
34             (blockSize, blockSize, blockSize);
35         blockHint.GetComponent <MeshRenderer> ().material =
36             hintMaterial;
37         blockHint.SetActive (false);
38         blockHint.GetComponent <Collider> ().enabled = false;
39         blockHint.transform.SetParent (parentObject.transform);
40         blockHint.tag = "Untagged";
41     }
42
43     Vector3 quantize (Vector3 p)
44     {
45         return blockSize * new Vector3 (Mathf.Round (p.x /
46             blockSize), Mathf.Round (p.y / blockSize), Mathf.Round
47             (p.z / blockSize));
48     }
49
50     [Command]
51     void CmdCreateBlock (Vector3 position, float blockSize)
52     {
```

```

47     GameObject g = Instantiate (blockTemplate);
48     g.transform.SetParent (parentObject.transform);
49     g.transform.localScale = new Vector3 (blockSize,
50         blockSize, blockSize);
51     g.transform.localPosition = position;
52     g.transform.localRotation = Quaternion.identity;
53     NetworkServer.Spawn (g);
54     Debug.Log ("Server put at" + (100.0f * position) + " "
55     + (100.0f * g.transform.localPosition) + " " + (100.0f
56     * g.transform.position));
57 }
58
59 [Command]
60 void CmdRemoveBlock (GameObject g)
61 {
62     NetworkServer.Destroy (g);
63 }
64
65 void Update () {
66     if (isLocalPlayer)
67     {
68         // Only reset the debounce controls when the buttons
69     are not being pressed.
70     if (!controller.getPrimary ())
71     {
72         debouncePrimary = false;
73     }
74     if (!controller.getSecondary ())
75     {
76         debounceSecondary = false;
77     }
78
79     RaycastHit hit;
80     if (Physics.Raycast (controller.getRay (), out hit))
81     {
82         if (hit.collider.tag == "ActiveBlock")
83         {
84             if (currentActive != null)
85             {
86                 currentActive.GetComponent <MeshRenderer>
87                 ().material.color = new Color (1,1,1);
88             }
89             currentActive = hit.collider.gameObject;
90             currentActive.GetComponent <MeshRenderer>
91             ().material.color = new Color (1,0,0);
92             Vector3 hintPosition = quantize
93             (parentObject.transform.InverseTransformPoint
94             (hit.point + hit.normal * 0.5f * blockSize));
95             blockHint.transform.localPosition = hintPosition;
96             blockHint.transform.localRotation =
97             Quaternion.identity;
98             blockHint.SetActive (true);
99         }

```

```

91     }
92     else
93     {
94         blockHint.SetActive (false);
95     }
96
97     // Add a new block.
98     if (blockHint.activeSelf && controller.getPrimary ()
99     && !debouncePrimary)
100    {
101        Debug.Log ("Place at " + (100.0f *
102        blockHint.transform.localPosition) + " " +
103        blockHint.transform.position));
104        CmdCreateBlock (blockHint.transform.localPosition,
105        blockSize);
106        debouncePrimary = true;
107    }
108
109    // Delete a block.
110    if (controller.getSecondary () && !debounceSecondary)
111    {
112        CmdRemoveBlock (currentActive);
113        currentActive = null;
114        blockHint.SetActive (false);
115        debounceSecondary = true;
116    }
117}
118
119}
120
121}
122
123}
124
125}

```

This code is designed to support its later role both as part of a networked multi-participant system and for use in an augmented reality context. Block creation and removal is performed through the remote procedure call mechanism (the Command attributes) which ensures that this takes place on the server. Positions are all calculated locally relative to a parent object which is the tracked marker position in an augmented reality setting.

The boolean debounce variables are used to ensure that only one block is created (or destroyed) per click of the button. When the button is pressed and an action is taken the debounce variable is set to true. This is only cleared when it is detected that the button is released. Actions are not permitted while the debounce is true which avoids a constant stream of objects being created

(or destroyed) if the user does not release the button fast enough.

6. The next step ensures that the blocks are replicated on the server, and on the other clients. Enhance the block prefab by adding NetworkIdentity and NetworkTransform components. The block prefab also needs to be added to the NetworkManager object, to its lists of Spawnable Prefabs.

Block creation involves a call to the server:

---

```
1 CmdCreateBlock (blockHint.transform.position,
    blockSize);
```

---

This function runs on the server and instantiates a block. The block is then spawned onto all of the other clients.

---

```
1 [Command]
2 void CmdCreateBlock (Vector3 position, float
    blockSize)
3 {
4     GameObject g = Instantiate (blockTemplate);
5     g.transform.localScale = new Vector3 (blockSize,
        blockSize, blockSize);
6     g.transform.position = position;
7     NetworkServer.Spawn (g);
8 }
```

---

There are unfortunately a few limitations with the current networking functionality in the Unity software. Firstly the size of the blocks is not transferred when the blocks are created on the other clients. Also the blocks need to have the AROverlay as a parent, and this is not achieved.

To address this, remove the NetworkTransform component from the block and replace it with a new C# script called BlockTransform. The code for this is provided in Algorithm 21.21.

Code Listing 21.21: `BlockTransform`. A simplified network transform component that sends position and scale updates at regular intervals. This also ensures that the object is positioned relative to a particular parent.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Networking;
5
6  public class BlockTransform : NetworkBehaviour {
7
8      [SyncVar (hook = "updatePosition")]
9      Vector3 localPosition;
10
11     [SyncVar (hook = "updateScale")]
12     Vector3 localScale;
13
14     void updatePosition (Vector3 p)
15     {
16         transform.localPosition = p;
17         transform.localRotation = Quaternion.identity;
18     }
19     void updateScale (Vector3 s)
20     {
21         transform.localScale = s;
22     }
23
24     void Start ()
25     {
26         GameObject parentObject = GameObject.Find
27             ("AROverlay");
28         transform.SetParent (parentObject.transform);
29         if (localScale.x > 0)
30         {
31             updatePosition (localPosition);
32             updateScale (localScale);
33         }
34
35     private float timer = 0.0f;
36     void Update ()
37     {
38         if (isServer)
39         {
40             timer += Time.deltaTime;
41             if (timer > 0.1f)
42             {
43                 timer = 0.0f;
44                 localPosition = transform.localPosition;
45                 localScale = transform.localScale;
46             }
47         }
48     }
49 }
```

7. There is now sufficient support for mutual presence in this application. The next steps integrate the application with the augmented reality overlay that it is designed for.

Set the project to build on a mobile device. We also need to choose one of the augmented reality toolkits to support tracking and interaction.

This example uses AR Core. In the player settings, under the XR Settings, tick the checkbox for AR Core Supported. Import the AR Core SDK for Unity into the project. Add the ARCore Device from the prefabs folder of the SDK to the scene.

Download version 1.18 or lower to avoid having to upgrade the gradle installation. In the event of an error about missing Tracked Pose Driver, this can be supplied by adding the XR Legacy Input Helpers package using the package manager (under the Windows menu).

8. The next step is to anchor the overlay content to a particular trackable element in the physical scene. Add a marker image to the project. Select this image, and create a new image database (Create/GoogleARCore/AugmentedImageDatabase). Set this database as the Augmented Image Database property of the Session Config configuration of the ARCore Device.

Create a new empty object and add the C# script ShowObjectOnMarker provided in Algorithm 21.22. Move the AROverlay object to be a child of this (for convenience) and provide the marker name and AROverlay object as the properties. Create a new Update Handler and set this to the defaultTrackHandler of the ShowObjectOnMarker component of the empty object.

Code Listing 21.22: ShowObjectOnMarker. An AR Core component to show an object when a marker is tracked.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.Events;
```

```
5     using GoogleARCore;
6
7     [System.Serializable]
8     public class MarkerTrackEvent : UnityEvent <Vector3,
9         Quaternion> {}
10
11    public class ShowObjectOnMarker : MonoBehaviour
12    {
13        [Tooltip ("The name of the marker in the database")]
14        public string trackableName;
15        [Tooltip ("The object that is shown when the marker is
16        found")]
17        public GameObject asset;
18        [Tooltip ("The functions that are called while the
19        marker is tracked")]
20        public MarkerTrackEvent updateHandler;
21
22        private Anchor anchor;
23
24        void Start ()
25        {
26 #if UNITY_ANDROID && !UNITY_EDITOR
27            asset.SetActive (false);
28 #endif
29        }
30
31        void Update()
32        {
33            List<AugmentedImage> arImages = new List
34            <AugmentedImage> ();
35            Session.GetTrackables <AugmentedImage> (arImages,
36            TrackableQueryFilter.Updated);
37
38            foreach (AugmentedImage image in arImages)
39            {
40                if ((image.Name == trackableName) &&
41                    (image.TrackingState == TrackingState.Tracking ))
42                {
43                    if (anchor == null)
44                    {
45                        anchor =
46                            image.CreateAnchor(image.CenterPose);
47                    }
48                    asset.SetActive (true);
49                    updateHandler.Invoke
50                        (anchor.transform.position, anchor.transform.rotation);
51                    break;
52                }
53            }
54        }
55
56        public void defaultTrackHandler (Vector3 p, Quaternion
57            r)
```

```

49     {
50         asset.transform.position = p;
51         asset.transform.rotation = r;
52     }
53 }
```

Reshape the content in the AROverlay so that it represents a desktop sized object that appears flat on top of the marker image.

9. The next step is to get the gaze control working with the ARCore Device. This is achieved by ensuring that the avatar head movements match those of the physical camera. The code already included in the gaze controller transfer the physical camera coordinates to the avatar.

---

```

1 transform.position = ARCamera.transform.position +
2     new Vector3 (0, 0.1f, -0.2f);
2 transform.rotation = ARCamera.transform.rotation;
```

---

The position is offset slightly relative to the center of the head to ensure that the laser beam is visible, and that the front of the head does not block the view. It may help to extend the near clipping plane on the First Person Camera on the ARCore Device, if the inside of the head object is still visible.

10. This example has several options for extensions. A tracked controller can be integrated to allow some tracking of both head and hands. The elements of the activity can be refined to be either a game based on the block manipulation mechanic, or the process can be extended to suit the needs of a particular application context where the block manipulation can be replaced with actions more appropriate to the needs of a collaborative meeting.

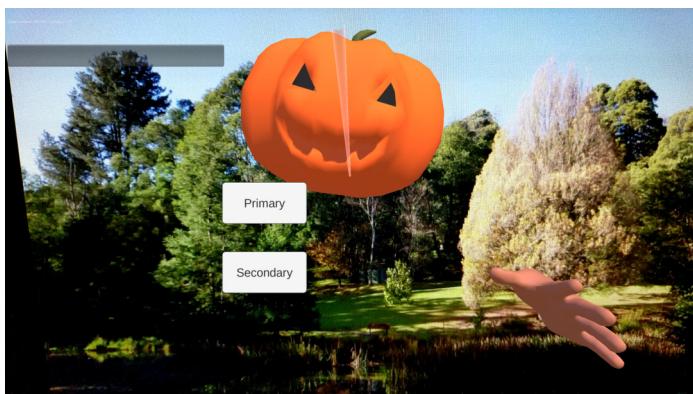


Figure 21.7.2:  
Interacting with  
another avatar.

**22**

## *Design of Experience Components*



## 23

# *Production Components*

## Contents

---

23.1	 Coordinating the efforts of a team of developers using version control . . . . .	601
23.2	 Detecting issues early by writing unit tests . . . . .	605
23.3	 Publishing enhanced reality experiences for the Daydream . . . . .	612

---

23.1  *Coordinating the efforts of a team of developers using version control*

**Component 23.1.1: Version control for enhanced reality projects.**

Management of a project through a version control system provides a number of advantages. The most significant is that the version control system acts as a way of achieving automatic backups, provided that version control runs through an external server. Even when that is not the case, the copies that exist on machines of collaborators or on other systems that you're working on, represent at least partial backups.

An associated benefit is the ability to manage copies on multiple machines. This is helpful when operating on multiple development machines, or developing on one

system and deploying on another. Testing networked applications across multiple machines also benefits from not having to make multiple copies that need to be synchronized.

Version control itself is an important benefit. If properly managed then you always have a runnable copy of your application even while in the process of trying to add further features. This record of previous versions allows you to roll back to a working version if something goes horribly wrong, or to compare files where some small change has made a crucial difference and you can't remember what it was.

This example doesn't go into detail into the mechanics of any single version control system. Most systems, such as git, svn or cvs, offer similar facilities at a small group level. Supporting bigger teams with more complex projects may require using some of the more specialized features. The focus in this example is instead on the processes and procedures that should be used when managing enhanced reality projects and the associated files. The points described are recommended practices, rather than a sequence of instructions.

1. Most version control systems, regardless of their original architecture, are hosted on a central server. Many of these are provided by hosting services that offer free hosting for projects involving small teams and where the content is publicly available. Administration of the version control system can be done through a web front end provided, through custom applications developed to support the hosting service, or through generic tools provided as part of the version control system. Many of these work at a command line level and are useful when debugging issues with a repository.
2. New repositories are registered through the version control system and then populated by committing a new project. The sequence of steps is typically:
  - (a) Create a new (blank) repository through the version

control system.

- (b) Check it out into a particular directory on your computer.
  - (c) Create a new empty project in that directory.
  - (d) Commit your changes back to the repository.
  - (e) Share the repository details with your colleagues so they can use it as well.
3. Projects, such as those created using the Unity software, consist of many files all under a common directory. Many of these files are automatically generated to improve the efficiency of the application and do not need to be retained as part of the project (i.e. they can be easily recreated as required when needed). This leads to the principle:

*Keep only those files in the repository that are needed.*

In particular, do not keep any files that are generated automatically. This includes intermediate files produced during compilation. Any file added to the repository is tracked through all its changes which wastes the space required to store all the versions, and slows access to the repository as all this extra information is transferred across the network.

Most repositories have a way of indicating which files are not part of the project. This avoids having them presented to be committed. For example, the git software keeps details of these in a `.gitignore` file (note the leading `.`). Examples of such files for Unity projects can be found online (such as <https://github.com/github/gitignore>).

4. The version control system takes care of cases when a file is changed by two different people at the same time - most of the time. In other cases it will flag the issue and require a human to make a decision. When the file is a text file (such as program code) the changes are highlighted and the human manually reconciles the code alternatives and marks the issue as resolved.

When the file is a binary file, such as an art asset in an enhanced reality project, then the change may need to be made one after the other (i.e. one person will have to reapply their fix on top of the file changed by the other person). This is a bit more of an inconvenience and is best resolved by good teamwork behaviour.

The suggested working arrangements following this recipe:

- (a) Start a new working session by identifying a task on the task tracking system and moving/mark it as allocated to yourself. This should discourage others from working on the same issue and ideally avoid them tampering with files you have changed.
- (b) Check out the repository. This ensures you have a copy of the most recent version of the project including any changes that any other team member has made up to this point.
- (c) Undertake the task. Work on your local copy of the repository. Make changes, and test the outcomes. If anything goes badly wrong, you can delete all the local files and retrieve a fresh copy from the repository.
- (d) Once the task is complete and the changes all work then commit the changes back to the repository.  
*Only ever commit working files to the repository.* Breaking the version in the repository affects everyone working on it and is considered undesirable. However the retribution of your peers discourages this happening twice.  
When you commit changes back to the repository you have the option of including a comment with the update. Make use of this to leave an informative message about what has changed.  
Have you considered including some unit tests with any new components that you have developed?
- (e) Mark the task complete in the task tracking system.

## 23.2 Detecting issues early by writing unit tests

### Component 23.2.1: Writing unit tests.

**Unit Tests**

This example makes use of the NUnit testing framework <http://nunit.org/> as supported by Unity software.

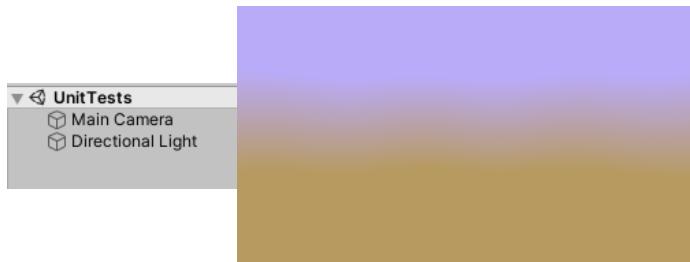


Figure 23.2.1: The unit test structure is not reflected in the active scene.

1. In a new (or in future, an existing) project, open up the Test Runner (under Window/General/Test Runner). If this opens as a separate window, the tab can be docked with other Unity software panels to conserve screen real-estate if desired.

The testing infrastructure is not intended to be part of the final deployed application but is only used during the development phase of the project. In the context of the Unity software it must be placed in a different assembly to that of your production code. This will require first grouping your production code into its own named assembly as well.

- (a) Create a Scripts folder for your production code.

In the Assets area, use Create/Assembly Definition to create an assembly for all the production code. Giving this a meaningful name (such as “ProductionCode” in this instance) helps to identify it when we use it later. Each module you develop can be part of its own assembly, which also helps speed up the project build process by limiting changes to only those assemblies that have been updated.

- i. In the Scripts folder, also create a new C# script named SceneChanger. This will initially contain

the skeleton of the function we plan to build. Initially just give it the minimum that will be needed for the tests to compile, such as the function definition and a default return value, as shown:

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class SceneChanger : MonoBehaviour {
6
7      // Switch to the scene named. Return true if
8      // the operation succeeds.
9      public static bool changeScene (string
10         destSceneName)
11     {
12         return true;
13     }
14 }
```

---

- (b) Once again in the Assets folder, create a new folder to hold the test functions. Call this SceneChangeTests, although all tests for a given project can be all part of a common Tests folder. The test runner has options to create these folder, but it can also be achieved through two steps:
- i. Create the folder (under Create/Testing).
  - ii. Create an assembly definition in the folder.

The assembly definition should have the test assemblies checkbox ticked so that it doesn't end up in the production build. Add a reference (first property in the Inspector pane, once an assembly is selected) that links to the ProductionCode assembly as well so that the tests can access the SceneChanger module.

2. A skeleton for some unit test functions can now also be created in the SceneChangeTests folder (Create/Test-ing/C# Test Script). This contains two automatically generated functions to illustrate how to write tests. The two examples show how to write a function that takes an input and provides an immediate response, as well as a test function that needs to monitor progress over multiple frames of the enhanced reality application.

Two aspects of the proposed scene change function are tested, to illustrate each of these cases.

3. The scene change should fail if the destination scene does not exist. The test function to achieve this consists of:

```

1 [Test]
2 public void SceneChangeChecksDestination() {
3     bool result = SceneChanger.changeScene (""
4         ThisSceneShouldNotExist");
4     Assert.AreEqual (result, false, "Changing to a non
5         existent scene should fail - instead
5         succeeded");
5 }
```

---

The scene change function is called with the name of a (hopefully!) non-existent scene. The Assert suite of functions provides ways of testing various forms of result. In this case, the test must have the result returned by the boolean value: false. A further message is provided to help our future selves understand what the test is looking for and what might have gone wrong.

4. A second property of the scene change function is that it actually work under some conditions. If we are not currently in a particular scene (need to check that), and we do change scenes to a valid target then we expect the scene change operation to succeed (check that also) and then be in that scene a few frames later (also check this).

The test code to do this is:

```

1 [UnityTest]
2 public IEnumerator SceneChangeResultsInNewScene() {
3     Assert.AreNotEqual (SceneManager.GetActiveScene
4         ().name, "TestDestScene", "Test is starting
4         in the wrong scene.");
4     yield return null;
5
6     bool result = SceneChanger.changeScene (""
7         TestDestScene");
7     Assert.AreEqual (result, true, "Attempt to
7         change scene failed.");
8
9     // Give a few frames for new scene to load.
```

```
10     yield return null;
11     yield return null;
12     yield return null;
13
14     Assert.AreEqual (SceneManager.GetActiveScene () .
15         name, "TestDestScene", "Destination scene
not reached after scene change");
16 }
```

---

If this test fails, it might be because the scene TestDestScene has not been created. This would help demonstrate that the unit test is working correctly. To solve this problem, save a scene of that name in the Scene-ChangeTests folder.

If the test continues to fail then read further.

5. Use the test runner to run all the tests. This may take a second or two because the tests are intended for play mode which provides the more authentic environment of an actual running application.
6. The remainder of this example describes the implementation of the changeScene function and reflects on the issues encountered.
  - (a) Invoking SceneManager.LoadScene should provide the required operation. Adding this in still results in both tests failing. This is great, the test framework is immediately paying back the time invested.
  - (b) The first test requires that the changeScene function return an indication of success. First check to be made is that the destination scene is valid. The GetSceneByName operation checks the available scenes and returns a scene marked as invalid if the required one is not found. Adding this in, and the first test passes. Immediate gratification.
  - (c) The second test now fails for a different reason. Initially it was because the destination scene could not be loaded, now it is because the changeScene function returns false to indicate that attempted to load the destination scene would fail. A scene called

“TestDestScene” is added to the folder with the tests.  
The second test still fails.

- (d) The TestDestScene is added to the list of scenes in the project, under the build settings (File/Build Settings). The second test still fails. A bit of investigation reveals that the GetSceneByName operation only works for scenes that are already loaded. An alternative approach is required. Again testing has paid off.

A suggested strategy is to enumerate the scenes in the build settings, and see if the required name can be found.

- (e) Hooray, all tests pass as shown in Figure 23.2.2. The scene change function is a lot more robust than it would have been without these tests. There are still a number of aspects that remain to be validated before this function is used as a fundamental module across different projects. An asynchronous version would be particularly useful, with the ability to initiate the scene change, and then to provide notification when ready to make the swap. Other useful features include the ability to transport some objects to the new scene, or to clean up any old scenes left behind. The unit tests already created help ensure that aspects of the existing functionality are not lost should these enhancements be developed.
7. For reference, the final code for the scene change operation is provided in Algorithm 23.1, and the complete set of test functions is given in Algorithm 23.2.

Code Listing 23.1: SceneChanger. The working changeScene function, complete with additional function to ensure expected behaviour.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5  using System.IO;
6
7  public class SceneChanger : MonoBehaviour {
```

```

8      // Check if the scene name is in the build settings.
9      ↳ Return
10     // the index if so, otherwise return -1.
11     private static int findSceneNameInBuild (string name)
12     {
13         for (int i = 0; i <
14             SceneManager.sceneCountInBuildSettings; i++)
15         {
16             if (Path.GetFileNameWithoutExtension
17                 (SceneUtility.GetScenePathByBuildIndex (i)).Equals
18                 (name))
19             {
20                 return i;
21             }
22         }
23         return -1;
24     }
25
26     // Switch to the scene named. Return true if the
27     ↳ operation
28     // succeeds.
29     public static bool changeScene (string destSceneName)
30     {
31         // Checks to see if the destination scene can be
32         ↳ invoked.
33         if (findSceneNameInBuild (destSceneName) < 0)
34         {
35             return false;
36         }
37         SceneManager.LoadScene (destSceneName);
38         return true;
39     }
40 }
```

Code Listing 23.2: SceneTest. The pair of test functions implemented to validate the scene change operation.

```

1  using UnityEngine;
2  using UnityEngine.TestTools;
3  using NUnit.Framework;
4  using System.Collections;
5  using UnityEngine.SceneManagement;
6
7  public class SceneTest {
8
9      [Test]
10     // Make sure the scene change returns an
11     // error if the name of the destination scene
12     // is invalid.
13     public void SceneChangeChecksDestination() {
```

```

14     bool result = SceneChanger.changeScene
15     ("ThisSceneShouldNotExist");
16     Assert.AreEqual (result, false, "Changing to a non
17     existent scene should fail - instead succeeded");
18 }
19
20 [UnityTest]
21 // Make sure that the scene change can actually
22 // change scenes, and end up in a different scene.
23 public IEnumerator SceneChangeResultsInNewScene() {
24     Debug.Log (SceneManager.GetActiveScene ().name + " "
25     + SceneManager.sceneCount + " " +
26     SceneManager.sceneCountInBuildSettings);
27     string targetScene = "TestDestScene";
28     //SceneManager.LoadScene (targetScene);
29     Assert.AreNotEqual (SceneManager.GetActiveScene
30     ().name, targetScene, "Test is starting in the wrong
31     scene.");
32     yield return null;
33
34     bool result = SceneChanger.changeScene
35     (targetScene);
36     Assert.AreEqual (result, true, "Attempt to change
37     scene to " + targetScene + " failed.");
38
39     // Give a few frames for new scene to load.
40     yield return null;
41     yield return null;
42     yield return null;
43
44     Assert.AreEqual (SceneManager.GetActiveScene
45     ().name, targetScene, "Destination scene not reached
46     after scene change");
47 }
48 }
```

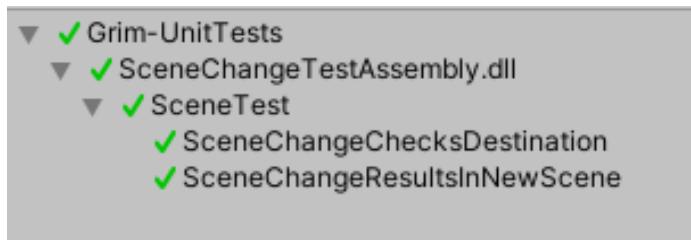


Figure 23.2.2: Unit test report in the test runner.

### 23.3 Publishing enhanced reality experiences for the Daydream

**Component 23.3.1: Google Daydream publication.** This example outlines the publication process for one particular store, while stressing aspects that are common to many. Daydream applications are published through the same Google Play store that houses traditional Android applications. However, as is common with several other stores offering virtual reality applications, there is an additional layer of quality control involved in receiving the Daydream compatibility tag.

The steps below outline some of the key stages of preparing the application for publishing, and suggest strategies that can be used to meet requirements.

1. Ensure that you have the desired package name set for your project (under Player Settings in the Unity software). This is the name that identifies your application in the store, and is used to ensure that updates are shared as part of the same application. It is hard (impossible?) to change this once the store entry has been created so it is important to get set it correctly before starting the publishing process.

The package name needs to be globally unique, so normally consists of an application name unique in your organization, followed by the unique name of your organization.

Version numbers and codes are also often visible to users of the application so it is worth setting these to appropriate and meaningful values (such as 1.0 for the first release).

2. Packages prepared for publishing are normally compiled using release settings, rather than the debug options that are commonly utilized during the development phase. This strips out extra information that might support debugging producing a more compact package that might also perform slightly better.

These packages often need to be signed in some way to ensure that fake versions of the package cannot be substituted at later stages of the distribution process.

The Unity software includes a set of Publishing Settings (under the player settings menu) which provide the opportunity to create a key store. This contains the encryption keys used to sign the package and is normally controlled by a password that needs to be entered whenever a new version of the package is built. A new key store needs to be created the first time a release version of the application is built. The file containing the key store must be kept safe since it will need to be reused for all subsequent updates to the package. Ideally it should not be kept in a public version control repository unless you're really confident about the strength of your password.

3. The store listing typically requires that the application receive a rating. This indicates the age group that the application is suitable for. Standards for rating differ across individual countries. Most stores provide a way of achieving a rating by completing a questionnaire about elements included in the application that may be controlled in some areas.

It should be noted as well that individual stores also have restrictions on the type of content that is allowed to be present in applications. Material that is clearly illegal in most countries will disqualify the application from the store. Internet censorship laws are also being propagated in many locations which affects the way in which content generated by users is managed. The latter typically requires some way of reporting objectionable content and mechanisms for removing this.

Daydream applications may not be used by children under the age of 13 due to concerns about health and safety for virtual reality use by that age group. Hence ratings that exclude this group do not significantly affect the potential user base for the application.

4. The store listing contains all the information about the application that is presented in the various listings.

This includes the listing on the Play store web site but also variations that might be present in the virtual reality version of the store for Daydream applications, as well as the launchers on the device. Descriptive summaries of the application are required in several forms; short summaries to introduce the application and longer descriptions for potential users looking for more information.

Visual representations of the application are required as well. These range from icons to represent the application in launch screens but also screenshots, logos and graphics, and videos to demonstrate and promote the application. The store listing defines the number and properties of each of these assets.

The Daydream listing requires a 3D panoramic image representing the application. These can be created using 3D modelling packages, such as in component 17.9.1 or taken directly from the running application using the process described in component 17.8.1. Take care when capturing from a live version of the application that the controller is disabled (invisible) so that it does not appear in the image. The person viewing the 3D panorama will have their own controller visible and don't need to see the original version baked into the scene.

Daydream applications also need a VR icon. This is just a two-layered image that can be rendered with one layer in front of the other. The icon itself is uploaded as the two layers represented by two separate images. Templates are available for preparing and aligning the contents of the two layers to ensure compliance with layout requirements.

5. Access to facilities (such as particular sensors) and information stored on the device is a common requirement for enhanced reality applications. Applications

on mobile devices have to apply for permission from the operating system (and ultimately the user) on the device. At one stage this was all done at the time the application was installed but recent trends involve the application applying for permission when the resource is about to be used. This provides an opportunity to explain the need for that permission to the user. If the permission is refused then the application can continue to operate as best possible in the absence of that facility.

Permission requests on Android devices require that the request be managed by the operating system itself. This ensures that applications cannot fake the request or impersonate the user's response. Access to this via a Unity software application can be achieved via a direct call to the underlying java library, or through new scripting functions being added to recent versions of the Unity software. These functions allow the application to check whether a particular permission has been previously granted, and to request the permission if it has not.

Daydream applications complicate the process further since, as of time of writing, these have no way of asking for permissions within the VR environment. Permission requests need to instruct the user to remove the headset, extract the phone, accept the permission and then return to the VR experience. The Google VR API provides functions to support this. For example, requesting read access to external storage would use the following steps:

```
1 // The particular permission required.  
2 string permission = "android.permission.  
    READ_EXTERNAL_STORAGE";  
3 GvrPermissionsRequester permissionRequester =  
    GvrPermissionsRequester.Instance;  
4 if (permissionRequester != null) {  
5     // Only request if permission not already granted.  
6     if (!permissionRequester.IsPermissionGranted (  
            permission)) {  
7         // Request new permission, including callback  
            // function to handle result.  
8         permissionRequester.RequestPermissions (new
```

```

9     string [] { permission },
10    (GvrPermissionsRequester.PermissionStatus []
11     permissionResults) => { });
12    // Check if the permission was actually granted.
13    bool granted = permissionRequester.
14      IsPermissionGranted (permission);
15  }
16 }
```

---

6. Daydream applications must satisfy a range of usability requirements associated with sound virtual reality interface design principles<sup>1</sup>. Fortunately many of these are achieved easily using the default options provided in the Unity software or through the Google VR API. There are some requirements that the developer still needs to pay attention to.
7. The Daydream controller can be used in either hand. Since there is not position tracking on the controller, the choice of which hand is represented in the application is controlled by a system level setting. The application needs to monitor the handedness setting currently in use, and change controllers appropriately. The left and right controller variations typically use different parameters for the arm models so that they appear to move as if grasped in the appropriate hand. Change of handedness can occur at any point during the application.

Code such as that shown below can be run during each frame to update controller settings. This code assumes that both left and right controller objects exist and track the controller, but that only one is visible at any time.

```

1  if (GvrSettings.Handedness == GvrSettings.
2   UserPrefsHandedness.Left)
3  {
4    controllerObject = leftControllerObject;
5    leftControllerObject.SetActive (true);
6    rightControllerObject.SetActive (false);
7  }
8  else
9  {
10   controllerObject = rightControllerObject;
11   leftControllerObject.SetActive (false);
12   rightControllerObject.SetActive (true);
13 }
```

---

<sup>1</sup> <https://developers.google.com/vr/distribute/daydream/app-quality>

This functionality can be tested by pressing the system button on the controller, changing the handedness setting, and resuming the application by pressing the system button a second time.

8. The application must shutdown when the user requires it. The application can support various quit menus inside the virtual reality experience but must also respond to the controls on the device itself, specifically the close and back button on the system interface.

The function to exit from the application back to the Daydream Home experience is:

```
1 GvrDaydreamApi.LaunchVrHomeAsync (null);
```

---

Responding to the close button is achieved using this code fragment:

```
1 if (Input.GetKeyDown(KeyCode.Escape))  
2 {  
3     Application.Quit();  
4 }
```

---

9. Application performance is a common requirement for virtual reality applications across a range of stores. Typically such stores expect 60 to 90 frames per second sustained over a period of time on a typical device. This does require some care in the application design and implementation given that the reference platforms specified are often fairly modest in their processing and rendering abilities.

There are some strategies that can be used to manage perform in applications developed using Unity software. Setting the property: Application.targetFrameRate can sometimes improve performance if the application can benefit from not being throttled to the default value (30 or 60 frames per second on some platforms).

There are also a range of graphics settings that can be manipulated under the Project Settings menu. Particular quality levels can be defined for applications

running on the Android platform. These properties can be tuned for a trade-off between visual quality of the particular application, and corresponding level of performance. Likely settings to try include those relating to the filtering of textures (trades-off between visibility/readability of fine detail against the time taken to render the texture) and the values associated with the lighting and shadowing effects in the scene.

10. Daydream applications are created in the same way as other applications in the Play store listing. To ensure that the application is distinguished as a Daydream application it must have certain optional properties defined in the store listing. These include the VR icon, and 3D panorama mentioned in previous steps. The motion intensity setting ranges from no motion (using only head tracking) to moderate or intense motion where translation of the viewpoint occur either user controlled or in response to events in the application. Once these are provided the Daydream checkbox can be enabled under the Pricing and Distribution section. Once this checkbox is selected your application undergoes additional review to make sure that it satisfies all of the quality requirements associated with a Daydream app. Review feedback is provided within a few days if there are any issues identified. Since each update to the app is reviewed it is worth submitting updates to a non-live track (such as one of the alpha or beta release tracks) until that version has passed review. It can then be upgraded to the live store version without the danger of new feature failing a compliance check.  
The review needs to check all aspects of the application so test accounts may need to be supplied for applications that may use of external services.

**24**

## *Resources*

### *Software*

A repository with the project files for all components is available at: <https://github.com/incshaun/OldeGrimoire>.



25

## Bibliography

- A. Aluri. Mobile augmented reality (mar) game as a travel guide: insights from pokemon go. *Journal of Hospitality and Tourism Technology*, 8(1):55–72, 2017. ISSN 1757-9899. DOI: [10.1108/Jhtt-12-2016-0087](https://doi.org/10.1108/Jhtt-12-2016-0087).
- N. Avouris and N. Yiannoutsou. A review of mobile location-based games for learning across physical and virtual spaces. *Journal of Universal Computer Science*, 18(15):2120–2142, 2012.
- R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *IEEE Computer Graphics and Applications*, 21(6):34–47, 2001. ISSN 0272-1716. DOI: [10.1109/38.963459](https://doi.org/10.1109/38.963459).
- Shaun Bangay. *The Monster Fun Book of Patterns for Developing Virtual Reality Applications: With examples using Unity software*. Kindle Direct Publishing, Geelong, Australia, 2019. ISBN 978-1729142400.
- Shaun Bangay, James Gain, Greg Watkins, and Kevan Watkins. Rhover: Building the second generation of parallel/distributed virtual reality systems. *Parallel Computing*, 23(7):991–100, July 1997.
- W. Barfield, C. Rosenberg, and W. A. Lotens. *Virtual Environments and Advanced Interface Design*, chapter Augmented-reality Displays, pages 542–575. Oxford

University Press, Inc., New York, NY, USA, 1995. ISBN 0-19-507555-2.

Steve Benford, Andy Crabtree, Stuart Reeves, Jennifer Sheridan, Alan Dix, Martin Flintham, and Adam Drozd. The frame of the game: Blurring the boundary between fiction and reality in mobile experiences. *ACM Conference on Human Factors in Computing Systems (CHI)*, pages 427–436, 2006. DOI: [10.1145/1124772.1124836](https://doi.org/10.1145/1124772.1124836).

M. Bonfert, I. Lehne, R. Morawe, M. Cahnbley, G. Zachmann, and J. Schoning. Augmented invaders : A mixed reality multiplayer outdoor game. In *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology – VRST ’17*, VRST ’17, pages 7–8, New York, USA, 2017. Acm. ISBN 9781450355483. DOI: [10.1145/3139131.3141208](https://doi.org/10.1145/3139131.3141208).

B. Brederode, P. Markopoulos, M. Gielen, A. Vermeeren, and H. de Ridder. powerball: the design of a novel mixed-reality game for children with mixed abilities. *Interaction Design and Children*, pages 32–39, 2005. DOI: [10.1145/1109540.1109545](https://doi.org/10.1145/1109540.1109545).

E. Brynjolfsson, Y. J. Hu, and M. S. Rahman. Competing in the age of omnichannel retailing. *MITSloan: Management Review Summer 2013*, 54(4), 2013.

R. Caillois. *The Game Design Reader: A Rules of Play Anthology*, chapter The Definition of Play, The Classification of Games, pages 122–155. MIT Press, Cambridge, Massachusetts,, 2006.

A. Cameron. Dissimulations: illusions of interactivity. *Millennium Film Journal*, 28:33–47, 1995.

K. Carlson, P. Sun, S. Cuykendall, and M. Lantin. Beyond the here and now: Exploring threaded presence in mediated, improvised performance. *Presence: Teleoperators and Virtual Environments*, 26(2):97–110, 2018. DOI: [10.1162/PRES\\_a\\_00288](https://doi.org/10.1162/PRES_a_00288).

- L. M. Castaneda, S. W. Bindman, A. Cechony, and M. Sidhu. The disconnect between real and virtually real worlds: The challenges of using vr with adolescents. *Presence: Teleoperators and Virtual Environments*, 26: 453, 2018. DOI: 10.1162/PRES\_a\_00310.
- T. Chatzidimitris, D. Gavalas, and D. Michael. Soundpac-man: Audio augmented reality in location-based games. In *18th Mediterranean Electrotechnical Conference: Intelligent and Efficient Technologies and Services for the Citizen, MELECON 2016*, Cyprus, 2016. ISBN 9781509000579. DOI: 10.1109/Melcon.2016.7495414.
- L. Chen, G. Chen, and S. Benford. Your way your missions: A location-aware pervasive game exploiting the routes of players. *International Journal of Human-Computer Interaction*, 29(2):110–128, 2013. ISSN 1044-7318. DOI: 10.1080/10447318.2012.694790.
- A. D. Cheok, X. Yang, Z. Z. Ying, M. Billinghurst, and H. Kato. Touch-space: Mixed reality game space based on ubiquitous, tangible, and social computing. *Personal and Ubiquitous Computing*, 6:430–442, 2002. ISSN 1617-4909. DOI: 10.1007/s007790200047.
- E. Ch'ng, D. Harrison, and S. Moore. Shift-life interactive art: Mixed-reality artificial ecosystem simulation. *Presence: Teleoperators and Virtual Environments*, 26(2): 157–181, 2017. DOI: 10.1162/PRES\_a\_00291.
- A. M. Clark and M. T. G. Clark. Pokemon go and research: Qualitative, mixed methods research, and the supercomplexity of interventions. *International Journal of Qualitative Methods*, 15(1), 2016. DOI: 10.1177/1609406916667765.
- S. W. Cole, D. J. Yoo, and B. Knutson. Interactivity and reward-related neural activation during a serious videogame. *Plos One*, 7(3):e33909, 2012. DOI: 10.1371/journal.pone.0033909.

- J. Collins, H. Regenbrecht, and T. Langlotz. Visual coherence in mixed reality: A systematic enquiry. *Presence: Teleoperators and Virtual Environments*, 26(1):16–41, 2017. DOI: [10.1162/PRES\\_a\\_00284](https://doi.org/10.1162/PRES_a_00284).
- C. J. Cutter, R. S. Schottenfeld, B. A. Moore, S. A. Ball, M. Beitel, J. D. Savant, M. A. Stults-Kolehmainen, C. Doucette, and D. T. Barry. A pilot trial of a videogame-based exercise program for methadone maintained patients. *Journal of Substance Abuse Treatment*, 47(4):299–305, 2014. ISSN 0740-5472. DOI: [10.1016/j.jsat.2014.05.007](https://doi.org/10.1016/j.jsat.2014.05.007).
- S. de Ribaupierre, B. Kapralos, F. Haji, E. Stroulia, A. Dubrowski, and R. Eagleson. *Virtual, Augmented Reality and Serious Games for Healthcare 1*, chapter Healthcare Training Enhancement Through Virtual Reality and Serious Games, pages 9–27. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-642-54816-1. DOI: [10.1007/978-3-642-54816-1\\_2](https://doi.org/10.1007/978-3-642-54816-1_2).
- S. Deterding, D. Dixon, R. Khaled, and L. Nacke. From game design elements to gamefulness: defining “gamiification”. In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, pages 9–15, Tampere, Finland, 2011. Acm. DOI: [10.1145/2181037.2181040](https://doi.org/10.1145/2181037.2181040).
- A. Dey, M. Billinghurst, R. W. Lindeman, and J. E. Swan. A systematic review of 10 years of augmented reality usability studies: 2005 to 2014. *Frontiers in Robotics and AI*, 5:37, 2018. ISSN 2296-9144. DOI: [10.3389/frobt.2018.00037](https://doi.org/10.3389/frobt.2018.00037).
- R. Diaconu, J. Deng, J. Bacon, and J. Singh. Comflux: External composition and adaptation of pervasive applications. In *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 427–429, ArXiv, 2018. Cornell University.
- M. Dunleavy, C. Dede, and R. Mitchell. Affordances and limitations of immersive participatory augmented

- reality simulations for teaching and learning. *Journal of Science Education and Technology*, 18:7–22, 2009. ISSN 1059-0145. DOI: [10.1007/s10956-008-9119-1](https://doi.org/10.1007/s10956-008-9119-1).
- T. J. Ellmers, W. R. Young, and I. T. Paraskevopoulos. Integrating fall-risk assessments within a simple balance exergame. In *2017 9th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games)*, pages 245–248, 2017. DOI: [10.1109/Vs-Games.2017.8056608](https://doi.org/10.1109/Vs-Games.2017.8056608).
- T. C. Endsley, K. A. Sprehn, R. M. Brill, K. J. Ryan, E. C. Vincent, and J. M. Martin. Augmented reality design heuristics: Designing for dynamic interactions. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 61, pages 2100–2104, 2017. DOI: [10.1177/1541931213602007](https://doi.org/10.1177/1541931213602007).
- R. Fedorov, D. Frajberg, and P. Fraternali. A framework for outdoor mobile augmented reality and its application to mountain peak detection. In Lucio Tommaso De Paolis and Mongelli, editors, *Augmented Reality, Virtual Reality, and Computer Graphics*, pages 281–301. Springer International Publishing, 2016. ISBN 978-3-319-40621-3. DOI: [10.1007/978-3-319-40621-3](https://doi.org/10.1007/978-3-319-40621-3).
- C. Fenu and F. Pittarello. Svevo tour: The design and the experimentation of an augmented reality application for engaging visitors of a literary museum. *International Journal of Human Computer Studies*, 2018. DOI: [10.1016/j.ijhcs.2018.01.009](https://doi.org/10.1016/j.ijhcs.2018.01.009).
- V. Fernandez-Cervantes, E. Stroulia, and B. Hunter. A grammar-based framework for rehabilitation exergames. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9926 Lncs:38–50, 2016. DOI: [10.1007/978-3-319-46100-7\\_4](https://doi.org/10.1007/978-3-319-46100-7_4).
- C. Fernandez-Vara. Play's the thing: A framework to study videogames as performance. In *DiGRA '09 –*

*Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory.* Brunel University, 2009.

- C. Freschi, S. Parrini, N. Dinelli, M. Ferrari, and V. Ferrari. Hybrid simulation using mixed reality for interventional ultrasound imaging training. *International Journal of Computer Assisted Radiology and Surgery*, 10(7):1109–1115, 2015. ISSN 1861-6429. DOI: [10.1007/s11548-014-1113-x](https://doi.org/10.1007/s11548-014-1113-x).
- E. Fujinawa, S. Yoshida, Y. Koyama, T. Narumi, T. Tanikawa, and M. Hirose. Computational design of hand-held vr controllers using haptic shape illusion. *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology – VRST ’17*, pages 1–10, 2017. DOI: [10.1145/3139131.3139160](https://doi.org/10.1145/3139131.3139160).
- S. Ganapathy. *Human Factors in Augmented Reality Environments*, chapter Design Guidelines for Mobile Augmented Reality: User Experience, pages 165–180. Springer New York, New York, NY, 2013. ISBN 978-1-4614-4205-9. DOI: [10.1007/978-1-4614-4205-9\\_7](https://doi.org/10.1007/978-1-4614-4205-9_7).
- Jason Geng. Structured-light 3d surface imaging: a tutorial. *Advances in Optics and Photonics*, 3(2):128, mar 2011. DOI: [10.1364/aop.3.000128](https://doi.org/10.1364/aop.3.000128).
- S. Guven. *Authoring and Presenting Situated Media in Augmented and Virtual Reality*. PhD thesis, New York, NY, USA, 2006.
- F. A. Hansen. Ubiquitous annotation systems: Technologies and challenges. In *Proceedings of the Seventeenth Conference on Hypertext and Hypermedia*, HYPERTEXT ’06, pages 121–132, New York, NY, USA, 2006. Acm. ISBN 1-59593-417-0. DOI: [10.1145/1149941.1149967](https://doi.org/10.1145/1149941.1149967).
- M. B. N. Hansen. *Bodies in Code: Interfaces with Digital Media*. Routledge, New York, 2012.

J. M. Harley, E. G. Poitras, A. Jarrell, M. C. Duffy, and S. P. Lajoie. Comparing virtual and location-based augmented reality mobile learning: emotions and learning outcomes. *Educational Technology Research and Development*, 64:359–388, 2016. DOI: [10.1007/s11423-015-9420-7](https://doi.org/10.1007/s11423-015-9420-7).

M. A. Harris. Beat the street: A pilot evaluation of a community-wide gamification-based physical activity intervention. *Games for Health*, 7(3):208–212, 2018. DOI: [10.1089/g4h.2017.0179](https://doi.org/10.1089/g4h.2017.0179).

Miguel Arevallilo Herráez, David R. Burton, Michael J. Lalor, and Munther A. Gdeisat. Fast two-dimensional phase-unwrapping algorithm based on sorting by reliability following a noncontinuous path. *Applied Optics*, 41(35):7437, dec 2002. DOI: [10.1364/ao.41.007437](https://doi.org/10.1364/ao.41.007437).

T. N. Hoang and T. N. Cox. Alternating reality: An interweaving narrative of physical and virtual cultural exhibitions. *Presence: Teleoperators and Virtual Environments*, 26(4):402–419, 2018. DOI: [10.1162/PRES\\_a\\_00307](https://doi.org/10.1162/PRES_a_00307).

T. Hollerer, S. Feiner, and J. Pavlik. Situated documentaries: embedding multimedia presentations in the real world. In *Digest of Papers. Third International Symposium on Wearable Computers*, pages 79–86, 1999a. DOI: [10.1109/Iswc.1999.806664](https://doi.org/10.1109/Iswc.1999.806664).

T. Hollerer, S. Feiner, T. Terauchi, G. Rashid, and D. Hallaway. Exploring mars: developing indoor and outdoor user interfaces to a mobile augmented reality system. *Computers & Graphics*, 23(6):779–785, 1999b. ISSN 0097-8493. DOI: [10.1016/S0097-8493\(99\)00103-X](https://doi.org/10.1016/S0097-8493(99)00103-X).

Keijo Inkilä. Homogeneous least squares problem. *Photogrammetric Journal of Finland*, 19(2), 2005.

N. Jablonsky, S. McKenzie, S. Bangay, and T. Wilkin. Evaluating sensor placement and modality for activity recognition in active games. In *Proceedings of the Australasian Computer Science Week Multiconference, ACSW*

- 2017, pages 61–1, New York, NY, USA, 2017. Acm. ISBN 978-1-4503-4768-6. DOI: [10.1145/3014812.3014875](https://doi.org/10.1145/3014812.3014875).
- J. Joo-Nagata, F. M. Abad, J. G.-B. Giner, and F. J. Garcia-Pesalvo. Augmented reality and pedestrian navigation through its implementation in m-learning and e-learning: Evaluation of an educational program in chile. *Computers & Education*, 111:1–17, 2017. ISSN 0360-1315. DOI: [10.1016/j.compedu.2017.04.003](https://doi.org/10.1016/j.compedu.2017.04.003).
- A. M. Kamarainen, S. Metcalf, T. Grotzer, A. Browne, D. Mazzuca, M. S. Tutwiler, and C. Dede. Ecomobile: Integrating augmented reality and probeware with environmental education field trips. *Computers and Education*, 68, 2013. ISSN 0360-1315. DOI: [10.1016/j.compedu.2013.02.018](https://doi.org/10.1016/j.compedu.2013.02.018).
- J. Kysela and P. Storkova. Using augmented reality as a medium for teaching history and tourism. *Procedia – Social and Behavioral Sciences*, 174:926–931, 2015. ISSN 1877-0428. DOI: [10.1016/j.sbspro.2015.01.713](https://doi.org/10.1016/j.sbspro.2015.01.713).
- B. Laurel. *Computers as Theatre*. Addison-Wesley Publishing Company, Reading, MA, 2013.
- J. J. Lee, P. Ceyhan, W. Jordan-Cooley, and W. Sung. Greenify: A real-world action game for climate change education. *Simulation and Gaming*, 44:349–365, 2013. DOI: [10.1177/1046878112470539](https://doi.org/10.1177/1046878112470539).
- Kangdon Lee. Augmented reality in education and training. *TechTrends*, 56(2):13–21, feb 2012. ISSN 8756-3894. DOI: [10.1007/s11528-012-0559-3](https://doi.org/10.1007/s11528-012-0559-3).
- H-F Lin and C-H Chen. Design and application of augmented reality query-answering system in mobile phone information navigation. *Expert Systems with Applications*, 42(2):810–820, 2015. ISSN 0957-4174. DOI: [10.1016/j.eswa.2014.07.050](https://doi.org/10.1016/j.eswa.2014.07.050).
- R. Lindgren and J. M. Moshell. Supporting children’s learning with body-based metaphors in a mixed re-

- ality environment. In *Proceedings of the 10th International Conference on Interaction Design and Children*, pages 177–180, Ann Arbor, Michigan, 2011. Acm. doi: 10.1145/1999030.1999055.
- R. Lukyanenko. Information quality research challenge: Information quality in the age of ubiquitous digital intermediation. *J. Data and Information Quality*, 7(1-2): 1–3, 2016. ISSN 1936-1955. doi: 10.1145/2856038.
- S. Lundgren and S. Bjork. Game mechanics: Describing computer-augmented games in terms of interaction. In *Proceedings of Technologies for Interactive Digital Storytelling and Entertainment (TIDSE 2003)*, pages 45–56, 2003. doi: 10.1057/jors.1984.31.
- A. P. Macvean. Task-involved versus ego-involved: Motivating children to exercise in a pervasive exergame. In *2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*, pages 405–406, 2011. doi: 10.1109/Percomw.2011.5766922.
- Sophie McKenzie, Shaun Bangay, Lisa M. Barnett, Nicola D. Ridgers, and Jo Salmon. Design elements and feasibility of an organized multiplayer mobile active videogame for primary school-aged children. *Games for Health Journal*, 3(6):379–387, December 2014. doi: 10.1089/g4h.2013.0097.
- Sophie McKenzie, Shaun Bangay, and Andrew Ferguson. Design principles for developing place attachment with location-based augmented reality interactive experiences. White Paper 117-01, GIVE Group, School of Information Technology, Deakin University, Australia, November 2018.
- P. Milgram, H. Takemura, A. Utsumi, and F. Kishino. Augmented reality: a class of displays on the reality-virtuality continuum. In *Proc. SPIE 2351, Telemanipulator*

- and Telepresence Technologies*, volume 2351, pages 282–292, 1995. ISBN 0277-786x. DOI: [10.1117/12.197321](https://doi.org/10.1117/12.197321).
- M. Nakevska, A. van der Sanden, M. Funk, J. Hu, and M. Rauterberg. Interactive storytelling in a mixed reality environment: The effects of interactivity on user experiences. In Yusuf Pisan and Sgouros, editors, *Entertainment Computing – ICEC 2014*, volume 8770, pages 52–59. Springer Berlin Heidelberg, 2014. ISBN 978-3-662-45212-7.
- T. Nilsen, S. Linton, and J. Looser. Motivations for augmented reality gaming. In *Fuse 04, New Zealand Game Developers Conference*, pages 86–93, 2004.
- T. Oleksy and A. Wnuk. Catch them all and increase your place attachment! the role of location-based augmented reality games in changing people – place relations. *Computers in Human Behavior*, 76:3–8, 2017. ISSN 0747-5632 (Issn). DOI: [10.1016/j.chb.2017.06.008](https://doi.org/10.1016/j.chb.2017.06.008).
- W-C Pang, C-Y Wong, and G. Seet. Exploring the use of robots for museum settings and for learning heritage languages and cultures at the chinese heritage centre. *Presence: Teleoperators and Virtual Environments*, 26(4):420–435, 2018. DOI: [10.1162/PRES\\_a\\_00306](https://doi.org/10.1162/PRES_a_00306).
- D. Papathanasiou-Zuhrt, D. F. Weiss-Ibanez, and A. Di Russo. The gamification of heritage in the unesco enlisted medieval town of rhodes. *CEUR Workshop Proceedings*, 1857:60–70, 2017.
- J. V. Pavlik and F. Bridges. The emergence of augmented reality (ar) as a storytelling medium in journalism. *Journalism & Communication Monographs*, 15(1):4–59, 2013. DOI: [10.1177/1522637912470819](https://doi.org/10.1177/1522637912470819).
- R. Planinc, I. Nake, and M. Kampel. Exergame design guidelines for enhancing elderly's physical and social activities. In *AMBIENT 2013 : The Third International Conference on Ambient Computing, Applications, Services and Technologies*, pages 58–64, Portugal, 2013. Iaria.

- R. Pryss, P. Geiger, M. Schickler, J. Schobel, and M. Reichenert. Advanced algorithms for location-based smart mobile augmented reality applications. *Procedia Computer Science*, 94:97–104, 2016. DOI: [10.1016/j.procs.2016.08.017](https://doi.org/10.1016/j.procs.2016.08.017).
- I. Radu. Why should my students use ar? a comparative review of the educational impacts of augmented-reality. In *2012 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 313–314, 2012. DOI: [10.1109/ismar.2012.6402590](https://doi.org/10.1109/ismar.2012.6402590).
- D. Richardson. Exploring the potential of a location based augmented reality game for language learning. *International Journal of Game-Based Learning*, 6:34–49, 2016. DOI: [10.4018/Ijgbl.2016070103](https://doi.org/10.4018/Ijgbl.2016070103).
- J. S. Roo and M. Hachet. One reality: Augmenting how the physical world is experienced by combining multiple mixed reality modalities. *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology – UIST ’17*, pages 787–795, 2017. DOI: [10.1145/3126594.3126638](https://doi.org/10.1145/3126594.3126638).
- R. Rouse, M. Engberg, N. JafariNaimi, and J. D. Bolter. Mrx: an interdisciplinary framework for mixed reality experience design and criticism. *Digital Creativity*, 26:175–181, 2015. ISSN 1462-6268. DOI: [10.1080/14626268.2015.1100123](https://doi.org/10.1080/14626268.2015.1100123).
- Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation*. IEEE, may 2009. DOI: [10.1109/robot.2009.5152473](https://doi.org/10.1109/robot.2009.5152473).
- K. Salen and E. Zimmerman. *Rules of Play: Game Design Fundamentals*. 2003. ISBN 9780262240451.
- J. Schneider, S. Schaal, and C. Schlieder. Geogames in education for sustainable development: Transferring

- a simulation game in outdoor settings. In *2017 9th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games)*, 2017.
- E. A. Schoneveld, M. Malmberg, A. Lichtwarck-Aschoff, G. P. Verheijen, R. C. M. E. Engels, and I. Granic. A neurofeedback video game (mindlight) to prevent anxiety in children: A randomized controlled trial. *Computers in Human Behavior*, 63:321–333, 2016. ISSN 0747-5632. DOI: [10.1016/j.chb.2016.05.005](https://doi.org/10.1016/j.chb.2016.05.005).
- A. R. Silva, E. Clua, L. Valente, and B. Feijo. First steps towards live-action virtual reality games. *SBC Journal on Interactive Systems*, 7(1):3–16, 2016.
- M. Slater and S. Wilbur. A framework for immersive virtual environments five: Speculations on the role of presence in virtual environments. *Presence: Teleoper. Virtual Environ.*, 6(6):603–616, 1997. ISSN 1054-7460. DOI: [10.1162/pres.1997.6.6.603](https://doi.org/10.1162/pres.1997.6.6.603).
- Olga Sorkine-Hornung and Michael Rabinovich. Least-squares rigid motion using svd. Technical report, Department of Computer Science, ETH Zurich, 2017.
- J. C. Spohrer. Information in places. *IBM Systems Journal*, 38(4):602–628, 1999. ISSN 0018-8670. DOI: [10.1147/sj.384.0602](https://doi.org/10.1147/sj.384.0602).
- C. Stapleton, C. Hughes, and J. M. Moshell. Mixed reality and the interactive imagination: Adding the art to the science and technology of mixed reality for training, education and entertainment introduction: Putting the stimulation in simulation. *First Swedish-American Workshop on Modeling and Simulation (SAWMAS)*, pages 30–31, 2002.
- J. Steuer. Defining virtual reality: Dimensions determining telepresence. *Journal of Communication*, 42(4):73–93, 1992. DOI: [10.1111/j.1460-2466.1992.tb00812.x](https://doi.org/10.1111/j.1460-2466.1992.tb00812.x).

- T.-H. Tsai, C.-Y. Shen, Z.-S. Lin, H.-R. Liu, and W.-K. Chiou. Exploring location-based augmented reality experience in museums. In Margherita Antonia and Stephanidis, editors, *Universal Access in Human–Computer Interaction. Designing Novel Interactions*, pages 199–209. Springer International Publishing, 2017. ISBN 978-3-319-58703-5. DOI: [10.1007/978-3-319-58703-5\\_15](https://doi.org/10.1007/978-3-319-58703-5_15).
- W. Walk, D. Gorlich, and M. Barrett. *Game Dynamics: Best Practices in Procedural and Dynamic Game Content Generation*, chapter Design, Dynamics, Experience (DDE): An Advancement of the MDA Framework for Game Design, pages 27–45. Springer International Publishing, 2017. ISBN 978-3-319-53088-8.
- J. Weber. *Designing Engaging Experiences With Location-Based Augmented Reality Games for Urban Tourism Environments*. PhD thesis, 2016.
- F. Xu, F. Tian, D. Buhalis, J. Weber, and H. Zhang. Tourists as mobile gamers: Gamification for tourism marketing. *Journal of Travel and Tourism Marketing*, 33:1124–1142, 2016. DOI: [10.1080/10548408.2015.1093999](https://doi.org/10.1080/10548408.2015.1093999).
- F. Xu, D. Buhalis, and J. Weber. Serious games and the gamification of tourism. *Tourism Management*, 60:244–256, 2017. ISSN 0261-5177. DOI: [10.1016/j.tourman.2016.11.020](https://doi.org/10.1016/j.tourman.2016.11.020).
- S. Yan, G. Ding, Z. Guan, N. Sun, H. Li, and L. Zhang. Outsideme: Augmenting dancer’s external self-image by using a mixed reality system. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, pages 965–970, Seoul, Republic of Korea, 2015. Acm. DOI: [10.1145/2702613.2732759](https://doi.org/10.1145/2702613.2732759).
- Ronald Zamora-Musa, Jeimy Vélez, and Heyder Paez-Logreira. Evaluating learnability in a 3d heritage tour.

*Presence: Teleoperators and Virtual Environments*, 26(4):  
366–377, August 2018. doi: 10.1162/pres\_a\_00305.

Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Fast  
global registration. In *Computer Vision – ECCV 2016*,  
pages 766–782. Springer International Publishing, 2016.  
doi: 10.1007/978-3-319-46475-6\_47.

