

Beginning Windows Mixed Reality Programming

For HoloLens and Mixed Reality
Headsets

Blending 3D visualizations with
your physical environment

Sean Ong

Apress®

www.allitebooks.com

Beginning Windows Mixed Reality Programming

For HoloLens and Mixed
Reality Headsets



Sean Ong

Apress®

Beginning Windows Mixed Reality Programming: For HoloLens and Mixed Reality Headsets

Sean Ong

Tukwila, Washington, USA

ISBN-13 (pbk): 978-1-4842-2768-8

ISBN-13 (electronic): 978-1-4842-2769-5

DOI 10.1007/978-1-4842-2769-5

Library of Congress Control Number: 2017949686

Copyright © 2017 by Sean Ong

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Cover image by NASA

Managing Director: Welmoed Spahr

Editorial Director: Todd Green

Acquisitions Editor: Jonathan Gennick

Development Editor: Laura Berendson

Technical Reviewer: Bart Trzynadlowski

Coordinating Editor: Jill Balzano

Copy Editor: Corbin Collins

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit <http://www.apress.com/rights-permissions>.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484227688. For more detailed information, please visit <http://www.apress.com/source-code>.

Printed on acid-free paper

This book is dedicated to my mother, Connie. She's perfect. Maybe a little too perfect. If I didn't know any better, I'd say she's a hologram.

Contents at a Glance

About the Author	xv
About the Technical Reviewer	xvii
Acknowledgments	xix
Introduction: The Holographic Future.....	xxi
■ Part I: Getting Started.....	1
■ Chapter 1: Gear Up: The Necessary Hardware and Software Tools	3
■ Chapter 2: Unity Crash Course.....	29
■ Part II: Building Holographic Experiences	53
■ Chapter 3: Creating Your First Hologram.....	55
■ Chapter 4: Introduction to the HoloToolkit.....	81
■ Chapter 5: Interacting with Holograms	95
■ Chapter 6: Using Spatial Mapping	115
■ Chapter 7: Spatial Sound.....	141
■ Part III: Growing as a Holographic Developer.....	153
■ Chapter 8: Awe-Inspiring Experiences	155
■ Chapter 9: Turning Holograms into Money	195
■ Chapter 10: Community Resources	201
Index.....	217

Contents

About the Author	xv
About the Technical Reviewer	xvii
Acknowledgments	xix
Introduction: The Holographic Future.....	xxi
■ Part I: Getting Started.....	1
■ Chapter 1: Gear Up: The Necessary Hardware and Software Tools	3
Making Sure Your PC Is Ready	3
Using a HoloLens, Emulator, or Other Mixed Reality Hardware	5
Testing with the HoloLens	6
Testing with Emulation	7
Understanding the HoloLens and Other Windows Mixed Reality Hardware.....	9
Inside-Out Tracking and Spatial Mapping.....	9
Spatial Sound	11
Transparent vs. Immersive Headsets.....	11
Downloading and Installing the Required and Optional Software Tools	12
Installing Visual Studio	13
Installing Unity	18
Downloading the HoloToolkit	25
Summary	27

■ Chapter 2: Unity Crash Course.....	29
What Is Unity?	29
Free vs. Paid Tiers of Unity	29
Your First Unity App	30
Step 1: Create a New Unity Project.....	30
Step 2: Save Your Scene	33
Step 3: Create a Ground Plane.....	34
Step 4: Rename Your Plane.....	36
Step 5: Reset Ground Plane Position	36
Step 6: Zoom to Your Ground Plane	37
Step 7: Scale Your Ground Plane	37
Step 8: Create the Ball	38
Step 9: Rename Your Ball	39
Step 10: Reset the Ball's Position	39
Step 11: Zoom to Your Ball.....	39
Step 12: Raise the Ball's Position	40
Step 13: Color the Ground Blue.....	40
Step 14: Add Physics to the Ball	44
Step 15: Enable Keyboard Control	45
Step 16: Testing Your App	50
Summary	51
■ Part II: Building Holographic Experiences	53
■ Chapter 3: Creating Your First Hologram.....	55
Getting Unity Ready for Mixed Reality Development	55
Step 1: Import HoloToolkit to a New Unity Project.....	56
Step 2: Use HoloToolkit to Prepare Your Scene for Mixed Reality Development.....	57

Your First Hologram.....	59
Step 1: Create a Cube	60
Step 2: Zoom to Your Cube.....	60
Step 3: Move the Cube Away from the Camera	61
Step 4: Resize the Cube.....	62
Step 5: Test Your App	63
Step 6: Install Your App on the HoloLens	64
Test Your App Using Holographic Remoting.....	72
Step 1: Install and Run the Holographic Remoting Player to Your HoloLens.....	73
Step 2: Connect to Your HoloLens with Unity's Holographic Remoting.....	74
Step 3: Test Your App Using Holographic Remoting	76
Test Your App Using Holographic Simulation.....	76
Step 1: Enable Holographic Simulation	77
Step 2: Connect Your Controller	77
Step 3: Test Your App Using Holographic Simulation	78
Summary	78
■ Chapter 4: Introduction to the HoloToolkit.....	81
What Is the HoloToolkit?	81
HoloToolkit Setup	81
HoloToolkit Components.....	84
HoloToolkit: Input	85
HoloToolkit: Sharing.....	88
HoloToolkit: Spatial Mapping	89
HoloToolkit: Spatial Understanding	89
HoloToolkit: Spatial Sound	90
HoloToolkit: Utilities	90
HoloToolkit: Build	91

HoloToolkit Online.....	92
The Two HoloToolkit Repositories	92
What Is GitHub?	92
HoloToolkit Help and Documentation.....	92
Summary	93
■ Chapter 5: Interacting with Holograms	95
Input Methods	95
Gaze Tutorial.....	96
Step 1: Set Up the Unity Scene.....	96
Step 2: Try the Scene.....	97
Step 3: Understand the Scene	98
Step 4: Use Gaze in Your Project.....	100
Gestures Tutorial	101
Step 1: Load the Test Scene	101
Step 2: Try It Out	101
Step 3: Use Air-Tap or Select Gesture.....	102
Step 4: Enter and Exit Focus.....	104
Step 5: Move Objects.....	105
Step 6: Implementing Gestures in Your Application	107
Voice Command Tutorial	107
Step 1: Load the Test Scene	108
Step 2: Try It Out	108
Step 3: Understand the Scene	109
Step 4: Add Your Own Voice Command.....	111
Step 5: Use Voice Commands in Your Own Project.....	112
Best Practices for Voice Commands.....	113
Other Hardware Input.....	113
Summary	114

■ Chapter 6: Using Spatial Mapping	115
What Is Spatial Mapping?.....	115
Spatial Mapping Tutorial.....	116
Step 1: Set Up Unity Scene	116
Step 2: Try It Out	117
Step 3: Understand the Scene	118
Step 4: Use Spatial Mapping in Your Application	120
Spatial Plane Finding Tutorial	121
Step 1: Set Up the Unity Scene.....	121
Step 2: Try It Out	122
Step 3: Load the Spatial Processing Scene	123
Step 5: Try Out the SpatialProcessing Scene.....	124
Step 6: Understand the SpatialProcessing Scene	125
Step 7: Use Spatial Processing in Your Application	126
Occlusion Tutorial	127
Step 1: Load the TapToPlace Scene	127
Step 2: Apply Occlusion	127
Step 3: Try It Out	128
Step 4: Use Occlusion in Your Application	130
Spatial Understanding Tutorial	131
Step 1: Set Up the Unity Scene.....	131
Step 2: Try It Out	132
Step 3: Use Spatial Understanding in Your Application.....	134
Spatial Anchors and Persistence.....	136
How to Use Spatial Anchors.....	136
Hologram Persistence	137
A Note on Sharing Anchors.....	138
Summary	139

■ Chapter 7: Spatial Sound	141
Spatial Sound Tutorial	142
Step 1: Set Up the Unity Scene	142
Step 2: Try It Out	142
Step 3: Understand the Scene	144
Step 4: Enable Spatial Sound in Your Application	148
Spatial Sound Design Considerations	150
When to Use Spatial Sound	151
What to Avoid When Using Spatial Sound	151
Summary	152
■ Part III: Growing as a Holographic Developer	153
■ Chapter 8: Awe-Inspiring Experiences	155
What Makes an App Awe-Inspiring?	155
Optimization and Performance	156
How to Monitor for Performance	157
Best Practices for Performance	159
Simplygon	165
Holographic Remoting	166
Stabilization Plane	166
Design and Magic	169
Best Practices for Design	169
Adding Magic: Vuforia	174
Capstone Project	176
Step 1: Import HoloToolkit to a New Unity Project	176
Step 2: Apply HoloLens Settings	177
Step 4: Insert and Configure InputManager	179
Step 5: Add a Cursor	179
Step 6: Create Responsive Ball	180

Step 7: Download Assets	184
Step 8: Create Your Lava Scene	186
Step 9: Add the Ability to Move Lava Scene.....	189
Step 10: Add and Configure Spatial Mapping	190
Step 11: Add Spatial Sound Effects	191
Step 12: Next Steps and Beyond	193
Summary	194
■ Chapter 9: Turning Holograms into Money	195
Publishing Your App to the Windows Store.....	195
Freelancing.....	197
Finding Mixed Reality Freelance Opportunities	197
Increasing Your Chances of Winning a Contract	198
Future Opportunities Today	199
Summary	200
■ Chapter 10: Community Resources	201
Microsoft's Official Mixed Reality Forum.....	201
HoloDevelopers Slack Team	203
What Is Slack?	203
What Is the HoloDevelopers Slack Team?.....	204
How to Join the HoloDevelopers Slack Team	205
Participating in the HoloDevelopers Slack Team	205
Other Online Communities and Resources.....	206
HoloLens Developers Facebook Group	206
Unity and Unity HoloLens Forum	208
HoloLens Subreddit	208
Next Reality News	209
YouTube	210

Local Events and Meetups	211
Europe Meetups.....	212
North America Meetups.....	213
Asia Pacific Meetups	214
Hackathons	214
Notable Industry Events	215
Summary	216
Index.....	217

About the Author



Sean Ong is an author, engineer, entrepreneur, and tech influencer who has written on topics ranging from renewable energy to augmented reality. Sean's love for virtual and augmented reality began at the age of five when he first tried the View-Master toy. Later, at the age of 14, he coded his first virtual home tour. Sean was among the first people to own a HoloLens and has developed numerous experiences for the new platform since its release. He is president of the Virtual Reality and Augmented Reality Association's Seattle chapter and is well known in tech circles for his informative tutorials and articles that have helped over seven million people. He resides in Seattle, Washington, with his wife and three kids and enjoys pushing the boundaries of technology.

About the Technical Reviewer



Bart Trzynadlowski has been programming since the fifth grade. After taking a detour deep into the world of semiconductor device physics and earning his PhD in electrical engineering, he presently develops low-latency software for a leading algorithmic options trading desk. Excited by the potential of Mixed Reality, Bart has been developing HoloLens apps in his spare time since June 2016.

Acknowledgments

I'd like to thank Bart Trzynadlowski for his thorough technical review of this book. I also thank Dwayne Lamb and Jesse McCulloch for establishing amazing HoloLens communities from which I attribute most of the HoloLens knowledge I have gained. Finally, I want to thank Jonathan Gennick and Jill Balzano for their friendship, persistence, and editorial support.

Introduction: The Holographic Future

Congratulations! If you're reading this, it means that you will probably be among the very first people responsible for building the mixed reality and holographic future that will dominate the next era of computing. For decades, science fiction has promised us a future filled with holograms and virtual experiences. We are finally on the verge of a technological revolution where our digital world intertwines with physical reality. This is known as *mixed reality*.

Imagine a future scenario where no screens exist. Instead, when you sit down to watch TV, a holographic screen appears on your wall. Because the screen is virtual, you can resize it to be as big as you like. You could also move the screen to any other room, or have it follow you around the house. You sit down at an empty desk, and several holographic computer monitors appear, along with virtual photos, a calendar, and a notepad. You're now ready to check your e-mail, work on a spreadsheet, and get started on a good day's work. You no longer need to carry around a physical smartphone. Instead, a holographic screen appears in your palm when needed. Holographic computing has the potential to replace every screen, and there's no reason to believe that it won't.

■ **Note** The *holograms* referred to in this book are *digital* holograms and do not operate on the same optical principles of traditional holography.

Does this sound like sci-fi technology that's still several years away? You may be surprised to know that everything I just mentioned in this "future" scenario is completely possible (and available) today with the Microsoft HoloLens. Figure I-1 illustrates how I use the HoloLens as a virtual desk. If you had asked me about living in a holographic world a few years ago, I would have predicted that we'd see capable devices within 15 to 20 years, and that would have been optimistic. But that all changed January 2015, when Microsoft announced the HoloLens and the Windows Mixed Reality platform. It caught the tech world off guard and inspired people to think about what a true holographic future would look like.



Figure I-1. The HoloLens enables an empty desk (top image) to be filled with holographic computer monitors and desk decorations (bottom image)

In my previous “future scenario” example, you’ll notice that I only give examples of holographic 2D screens. To some, my examples may have sounded amazing or revolutionary. They are, in fact, dull examples that don’t adequately capture what the HoloLens and other holographic headsets can achieve. The challenge and opportunity of building experiences for these headsets is unlike anything that the technology industry has faced to date. Until now, the vast majority of software experiences have been designed for flat, two-dimensional screens. Think of televisions, smartphones, tablets, laptops, or even the flat page or screen on which you are reading this book. Video games, 3D movies, and other so-called “3D” advances over the past few years are nothing more than a glorified 2D experience we view on our flat, rectangular screens.

The Windows Mixed Reality platform breaks this status quo by allowing us to develop true three-dimensional applications in our real world. Early applications that have emerged for the HoloLens suffer from developers “thinking inside the box” by creating 2D experiences such as floating holographic screens or 2D menus and buttons for navigation. Many in the industry believe that a functional and intuitive 3D user experience has yet to be discovered and developed. As we go through the tutorials and example projects in this book, we will pay particular attention to 3D design elements while discussing ways to think *outside* the box and move beyond the 2D status quo.

It is a very exciting time to be a holographic developer. The devices are capable, the computing paradigm is new, and ideas for good applications seem to be endless. We holographic developers are the engineers, architects, and builders that will create the forthcoming holographic world.

The holographic future is inevitable. As with all high-tech gadgets, devices like the HoloLens will only become smaller and more powerful over time. I anticipate that it won't be more than a few years until we see holographic glasses that are as thin and light as the Google Glass device that was announced in 2012.¹ It's not hard to image a future where many (if not most) people will wear a pair of these holographic glasses, whether or not they require prescription eyewear. Being equipped with these headsets will enable us to augment physical reality with relevant information, have more immersive digital experiences, and free us from the unnecessary screens that fill up our desks, walls, pockets, and purses.

How important will holographic devices be in daily life? One could speculate that most people in the near future might not be able to participate fully in society without a pair of holographic glasses. At first, this might sound like a dystopian prediction of our future. But think about how we use computers and smartphones today. It's very difficult to participate fully in today's modern society if you don't own or know how to use a computer. A vast majority of jobs in the United States require the use of a computer. We use e-mail and online messages as primary forms of communication. Surely, if you told someone 30–40 years ago that they would not be able to fully participate in a future society without owning or knowing how to use a computer, they would have been hesitant about such a future. Yet many of us today probably can't imagine daily life without our trusty PC or smartphone. Likewise, in 20 years I think we'll look back and wonder how we ever lived without our trusty holographic glasses.

Perhaps I've given you a glimpse of the future. More importantly, I hope to have inspired you to start thinking about the holographic apps and experiences that will fill the world around us. All of us are relying on people like you to build our holographic future. My motivation for writing this book is to get as many people started on holographic development as possible. It's written to be easily accessible, whether you're an experienced software developer or new to the world of programming. This book is intended to get you started with everything you need to begin developing amazing holographic experiences on the HoloLens and other Mixed Reality headsets.

This book is organized into ten chapters spread across three parts. In Part I, which contains Chapters 1 and 2, you will be guided through the installation and explanation of all the necessary software and tools for developing Windows Mixed Reality applications.

Everything you need to get started is contained in Chapter 1. You can begin developing Mixed Reality apps with or without a HoloLens. Things I cover in Chapter 1 include the following:

- Making sure your PC is ready for Mixed Reality development
- Using a HoloLens, HoloLens Emulator, or other Mixed Reality hardware

¹Google Glass is a small device that projected notifications, images, and other information to a small glass display near the user's right eye. Unlike the HoloLens, it did not place 3D holographic objects in the user's world.

- Downloading and installing the required and optional software tools
- Understanding the HoloLens and other Windows Mixed Reality hardware

Chapter 2 dives into the basics of Unity. Unity is the preferred software platform for developing Windows Mixed Reality experiences. Things we'll cover in Chapter 2 include the following:

- Understanding Unity
- Creating your first application in Unity
- Unity and Windows Mixed Reality

In Part II, comprised of Chapters 3–7, we'll start building holographic experiences. This is where you're guided through the fundamentals of creating a full-featured Mixed Reality application.

You'll learn how to make digital holograms in Chapter 3. You'll be guided through the creation of a basic holograms that can be viewed in the HoloLens. Here's what Chapter 3 covers:

- Preparing Unity for Windows Mixed Reality development
- Creating a cube in Unity
- Building and deploying the Unity application to the HoloLens

I discuss the HoloToolkit in Chapter 4. Manually preparing Unity for HoloLens development can be cumbersome and prone to error. This chapter introduces the HoloToolkit and how you can leverage this community resource:

- Understanding the HoloToolkit
- Downloading and using the HoloToolkit

In Chapter 5, we start interacting with holograms. I discuss the use of gestures, voice commands, and other ways of interacting with Holographic content. Here's what I cover in Chapter 5:

- Voice commands
- Gestures
- Gaze
- Clickers and other accessories

Things start getting interesting in Chapter 6, where you begin to leverage the power of the HoloLens by learning about using spatial mapping. I walk you through the technology, concept, and utilization of spatial mapping in the context of holographic applications. Chapter 6 covers the following topics:

- What is spatial mapping?
- How to use spatial mapping in projects

- Spatial understanding
- Anchors and persistence

Chapter 7 talks about spatial sound. You'll learn about the importance of spatial sound and how to utilize it in your projects. We'll discuss the following:

- What is spatial sound and how is it different than "regular" sound?
- How to use spatial sound in projects
- Best practices for spatial sound
- Additional sound resources

Part III (Chapters 8–10) is about growing as a Mixed Reality developer. At this point in the book, you'll be familiar with the basics of creating a Mixed Reality application. The three chapters introduce ways for you to optimize and enhance your experiences, publish and monetize your apps, and join the broader holographic community for support and visibility.

In Chapter 8, I discuss tips and tricks for awe-inspiring experiences. This chapter provides you with a primer on elements that give holographic experiences additional flair and magic, such as color choice, ambient elements, music, size, and more. Finally, we'll complete a capstone project together using key skills gained in this book. Here's what's covered in Chapter 8:

- Optimization and performance
- Design
- Magic
- Capstone project

Let's make some money! In Chapter 9, I discuss the details of publishing and monetizing your applications. You're presented with strategies for monetization, from publishing your app in the Windows Store to freelancing as an independent Mixed Reality developer. Here's what Chapter 9 covers:

- Monetization with the Windows Store
- Freelancing
- Thinking big: revolutionary opportunities

In Chapter 10, I suggest community resources and additional information for holographic developers. This chapter introduces resources that are available to you, including relevant community forums and online groups, notable events, and other information that will help during the development process:

- Why are community resources important?
- The official Windows Mixed Reality forums
- The HoloDevelopers Slack channel

- Events and local groups
- More information

As you embark on your journey to becoming one of the first Mixed Reality developers, I encourage you to keep two things in mind. First, always think outside the box or outside the “2D rectangle” that has dominated computing up until this point in time. Second, understand that you are responsible for building a new industry and the holographic world of tomorrow. You are a technological pioneer. Understanding this will inspire you to reach new heights and explore new ways of creating amazing experiences.

PART I



Getting Started

CHAPTER 1



Gear Up: The Necessary Hardware and Software Tools

In this chapter, you'll learn everything you need to be equipped for Mixed Reality development. We'll make sure your PC is ready for development and walk through some recommended computer specifications. I provide a brief discussion on how your computer hardware impacts Mixed Reality development and performance. We'll also go over your hardware and emulator options for testing your app during development. I provide an overview of the HoloLens and some key features that you'll want to be familiar with before you start developing apps. Finally, you'll be guided through installing all the necessary software tools needed to dive into the world of making Mixed Reality experiences.

■ **Tip** You don't need a HoloLens or Mixed Reality device to get started with development. You can still test your apps through holographic emulation (discussed shortly).

Making Sure Your PC Is Ready

Before getting started, you need to make sure you have a computer capable of handling Mixed Reality development. This section outlines the recommended system requirements and provides some additional context around these requirements. Fortunately, you don't need a high-end PC setup to make HoloLens apps. Unsurprisingly, Microsoft recommends Windows 10 as the operating system of choice. Other operating systems also work, including Windows 8.1, Windows 8, Windows 7, and more. Several HoloLens developers also have reported success developing on their Mac devices when running Windows virtually.

Here are my recommended system specs for Mixed Reality development:

- 64-bit Windows 10
- 6 GB RAM
- 30 GB available hard drive space

That's intended to be a fairly small list to illustrate how you don't need much to get started. You may technically get away with even lower system specs, but then you'll have a very painful and slow experience and I would recommend against it.

If getting a capable device is of particular interest to you, I've included a deeper discussion here of the various spec categories and what it means for your development experience.

- *RAM (Random Access Memory)* is your computer's way of storing memory that's quickly accessible. If you have a lot of open windows, websites, and applications, then you'll want to have more RAM to speed up multitasking on your computer. I recommend a minimum RAM size of 6 GB. For an optimal experience, you should aim for 12–16 GB of RAM. This allows you to have multiple (20+) browser tabs open, multiple windows open, Unity, Visual Studio, your music application, and background PC tasks running without slowing down your system.
- *The processor or CPU* is responsible for doing all the computational work. When the processor starts working hard, you'll immediately notice the difference between a slow processor and a fast one. These crucial times include *compiling* your app (the computer's way of converting the code you've written into something your HoloLens can install and understand), loading and working with complex 3D objects in Unity, and any other processing work that involves many objects in your Unity scene. I recommend the Intel Core i5 or Core i7 processors (or another processor with similar speed).
- *The OS (operating system)* of your computer is typically Windows, Mac OS, or Linux. As mentioned earlier, you can develop under a range of Windows versions (and even Windows on a Mac), but the recommended OS for Mixed Reality development is Windows 10. Microsoft is heavily promoting the use of 3D in Windows via the Windows 10 Creators Update made available in 2017. Tethered Windows Mixed Reality headsets will also rely on Windows 10 to work. Based on this, I consider it worthwhile to get a Windows 10 computer for Mixed Reality development.

- *The GPU (Graphics Processing Unit)* or graphics card is often poorly understood and gets less attention than the other items listed so far. Many laptops and low- to mid-end PCs don't include a dedicated graphics card but rather rely on graphics capabilities that are built in to the processor (also known as *integrated graphics*). When developing Mixed Reality applications, having a GPU is not required but can boost editing performance, especially when using Unity's holographic emulation features, especially when dealing with complex scenes and textures. That said, any Unity application struggling to run on an integrated desktop GPU is probably going to fare worse on the HoloLens itself, due to the limited graphics capabilities of the headset.

If you intend to work with the HoloLens emulator (not to be confused with holographic emulation within Unity—more on that in the next section), then you'll have higher system requirements. Because Unity has a simple and efficient holographic emulation feature, you will likely not need to use the HoloLens emulator. Some reasons for using the emulator may include testing HoloLens applications built outside of Unity (for example, C++ or Direct3D) or for debugging with Visual Studio (which may save a few minutes compared to deploying to the HoloLens device). If you need to use the emulator, these are the minimum system requirements:

- 64-bit Windows 10 Pro/Enterprise/Education Edition (*not* Home Edition)
- CPU with a minimum of 4 cores
- 8 GB RAM
- BIOS must support and enable Hardware-assisted virtualization, Second Level Address Translation (SLAT), and hardware-based Data Execution Prevention (DEP)
- GPU with DirectX 11.0 or later, WDDM 1.2 driver or later

Using a HoloLens, Emulator, or Other Mixed Reality Hardware

In this section I discuss the hardware and software options for testing your Mixed Reality applications. As mentioned, you don't need an actual HoloLens or Mixed Reality headset to get started with development. Even though I have a HoloLens device, I still regularly use the Unity emulator and a gamepad to quickly test my applications as I work on them. Of course, deploying your app to a HoloLens device is the ultimate way to verify that your app behaves as expected and allows you to do some real-world testing. In this section, I provide a general discussion of the advantages and disadvantages of the various hardware and software options.

Testing with the HoloLens

Testing your applications with the HoloLens is the best way to see and experience how your Mixed Reality app performs in the real world. Until recently, users who wanted to test their applications with the HoloLens needed to go through a long deployment procedure that took roughly 20 minutes, depending on how large the app was. Now you can stream directly from Unity to the HoloLens. This cuts down testing to seconds, versus the former 20+ minutes.

This streaming capability, called *holographic remoting* in Unity, is a special feature that allows you to stream the app you are building to your HoloLens via WiFi. The HoloLens scans your area, detects your voice and hand gestures, and sends all inputs back to Unity (again, via WiFi) so that testing via streaming feels like you have the app installed on your HoloLens.

Figure 1-1 illustrates what a typical holographic remoting setup looks like. The primary benefit of streaming is the time savings. Typically, when developing applications, you will make hundreds if not thousands of small changes to your code as your perfect app. It's often necessary to do a quick test after each little change to make sure that you are getting desired results. Being able to stream to the HoloLens instantly from Unity (as opposed to installing the test app to the HoloLens, which can take about 20 minutes) is a great way to save time. I walk through how to set up holographic streaming in Chapter 3.



Figure 1-1. Illustration of holographic remoting. The PC streams the app to the HoloLens for viewing, without the need to install the test app on the HoloLens.

Of course, testing your full app by installing it onto your HoloLens is recommended to accurately gauge the app's performance with your device's hardware. This is especially true if you will be publishing your app to the *Windows Store*, which is Microsoft's online shop for apps and media.

Testing with Emulation

If you don't have a Mixed Reality headset such as the HoloLens, you can still test (and even publish) your application. There are published HoloLens apps (for example, a 3D chess game) that were developed without the use of a HoloLens. This is possible with *holographic emulation*, which is the ability to simulate how the app would behave if your computer were a Mixed Reality headset. Currently, there are two primary ways to emulate:

- *Holographic simulation in Unity* is an extremely fast way to test your applications. It's just as fast as holographic streaming and allows you to use an Xbox controller or gamepad to "walk around" and uses buttons as taps and gestures. Figure 1-2 shows what holographic simulation typically looks like. I often use holographic simulation if I don't want to bother with wearing my HoloLens, or if I need to test a quick action that doesn't require the use of the HoloLens. Holographic simulation includes several 3D models of rooms and areas, so you can still test your app's ability to recognize and interact with walls, floors, and other surfaces. Both holographic simulation and holographic remoting are found under the Holographic Emulation menu in Unity. Tutorials on using Unity's holographic emulation features are provided in Chapter 3.



Figure 1-2. Illustration of testing applications using holographic simulation and a gamepad

- *The HoloLens emulator* is a more powerful and full-featured emulator than the one provided in Unity. The HoloLens emulator requires strong system specifications (see previous section for specifics) and allows you to test the performance of your application. It includes all the UI elements of the HoloLens, including access to the Windows Store, Settings menu, Device Portal, and more. Unlike holographic simulation in Unity, testing in the emulator takes roughly the same amount of time as testing on the actual HoloLens because you are still required to build your app and deploy/install it to the emulator. Using the HoloLens emulator is a good idea if you don't have a HoloLens and want to put your app through some final, robust testing before publishing your app to the Windows Store. Figure 1-3 shows what the HoloLens emulator looks like.

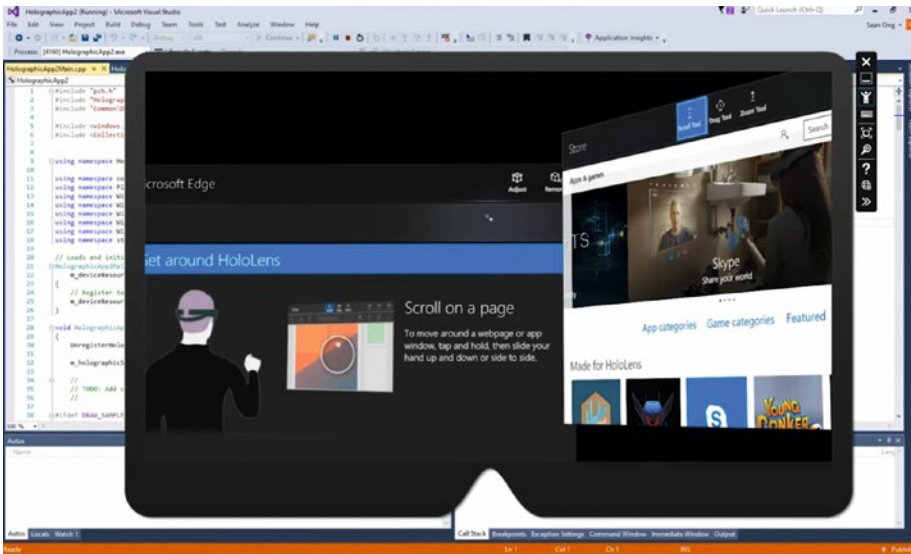


Figure 1-3. HoloLens emulator running on a PC

I don't include a detailed step-by-step emulator installation tutorial in this book, but I do have a step-by-step installation video, should you decide to install the emulator. You can find it at www.youtube.com/watch?v=0ImaZ_Aqe3I.

In summary, I recommend regularly testing your application using Unity's holographic simulation for when the use of the HoloLens is not necessary (for example, testing an animation or testing that something is moving as intended) and using Unity's holographic remoting when the use of the HoloLens is required (for example, testing gesture accuracy or testing interaction with your room). I recommend deploying to your HoloLens or device only occasionally to make sure there are no surprises, and also near the end of your project to make sure things are performing well when installed on the actual device.

Understanding the HoloLens and Other Windows Mixed Reality Hardware

This section covers some basics of the way the HoloLens works. An entire book could be written on the technological miracle of the HoloLens and the science behind it, but I cover just enough so that we can design the best app experiences for this and other Mixed Reality headsets.

Inside-Out Tracking and Spatial Mapping

What sets the HoloLens and other Windows Mixed Reality headsets apart from other popular headsets (as of this writing) is the ability to perform *inside-out tracking*, which is the ability for the headset to track its environment without the need for external sensors. *Outside-in tracking* headsets require the user to set up a few sensors around a room or area, which allows the headset to know where it is as the user moves around. Inside-out tracking avoids the cumbersome need to set up external sensors and can work in nearly all environments. Some basic virtual reality headsets you may have heard about (Google Cardboard, Samsung Gear VR) have no positional tracking, with only the ability to “look around.” Figure 1-4 shows a diagram of the HoloLens cameras, several of which are used for inside-out spatial tracking.

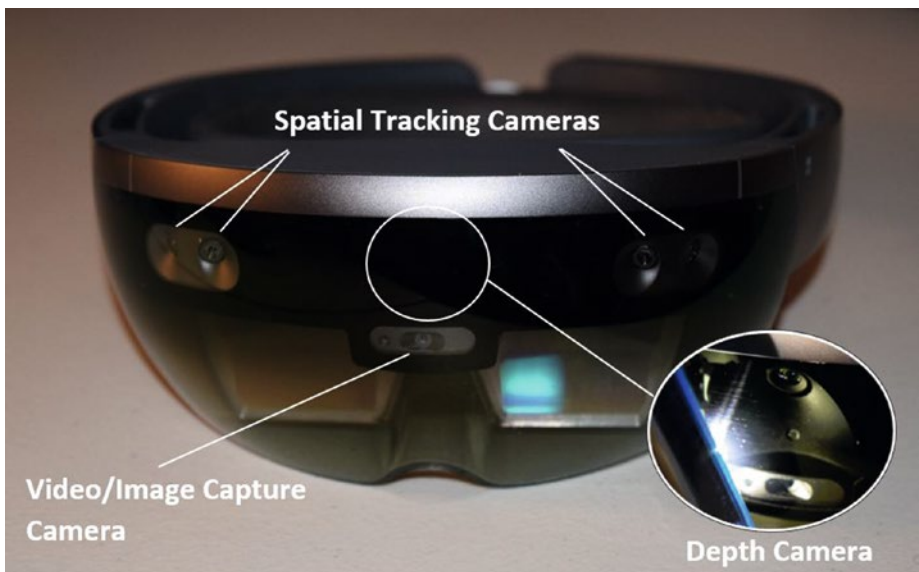


Figure 1-4. HoloLens cameras and their functions

The HoloLens has six cameras on the headset, five of which are used to track its environment (four environment tracking cameras and one depth camera, plus one regular camera for recording video or taking pictures). The HoloLens is constantly tracking its environment and building a 3D model of the area it's in. This is called *spatial mapping*. Spatial mapping is important for several reasons:

- It tells the HoloLens which holograms to hide from view. For example, if you place a hologram in your hallway and then walk into another room, the spatial map of that room's walls will prevent you from seeing the hologram in your hallway. If there were no spatial map, you would see the hologram as if it were visible through your walls, causing an unrealistic experience.
- It allows users to interact with the spatial map—for example, pin items to your walls, allow characters to sit on your sofa (as seen in Microsoft's "Fragments" app), or automatically decorate your surroundings.
- It allows for *hologram persistence*, which is the ability for holograms to stay where the user left them—even after turning off your device. Your HoloLens will (remarkably) be able to remember your space and restore any holograms that you had placed in that space.

Figure 1-5 illustrates how the HoloLens uses a mesh made of polygons to recreate a digital version of the user's surroundings.

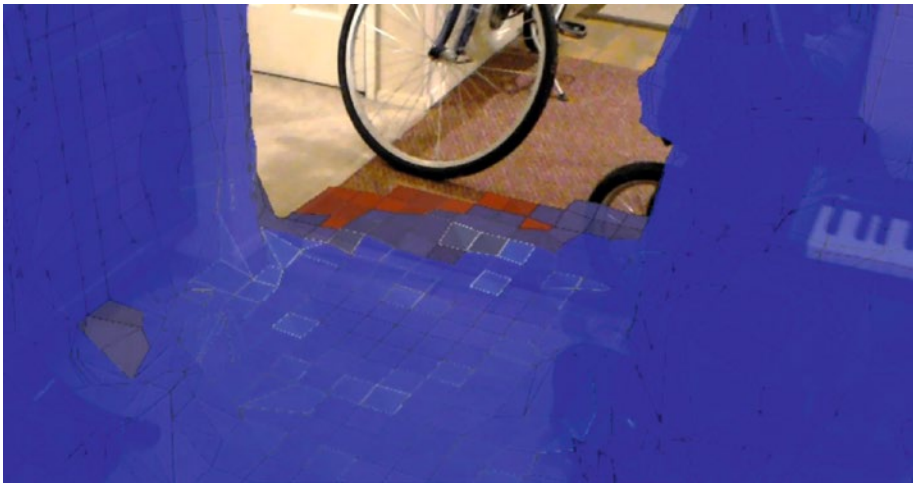


Figure 1-5. HoloLens using spatial mapping to track its environment

At the time of this writing, the spatial data available on Windows Mixed Reality does not include raw access to the depth camera's frame buffer but rather only a mesh generated from that data. This means that it would be difficult, if not impossible, to intercept raw depth data for use in your application—for example, for programming your own gestures or hand recognition.

Spatial Sound

We rely heavily on our ears to precisely locate objects around us. In the context of Mixed Reality, this is called *spatial sound*. The HoloLens has the ability to send spatial sound to the user's ears using a sound wave phase-shift approach. This increases the feeling of immersion. Users can still hear objects (if the objects are intended to make noise) around them, even if they can't see those objects. This increases the user's perception of these objects and makes the holograms feel like they are actually in the user's area.

Transparent vs. Immersive Headsets

Some Windows Mixed Reality devices like the HoloLens use transparent displays that allow you to see holograms placed in your real world. Other devices, such as Acer's Windows Mixed Reality headset, are *occluded*, meaning that you can't see your surroundings. Instead, these headsets immerse you in a virtual world. Immersive headsets still boast inside-out tracking, which is helpful for making the user aware of walls or other obstructions while wearing the headset. Some immersive headsets may also utilize spatial mapping, allowing the user to see a virtual representation of their surroundings, although such headsets have not been announced at the time of this writing. Figure 1-6 shows several examples of Windows Mixed Reality headsets.



Figure 1-6. Examples of Windows Mixed Reality devices include both transparent headsets such as the HoloLens and immersive headsets (source: Microsoft)

All existing devices as of this writing have a *field-of-view* (FOV) limitation, which means that you can't see holograms in all of your peripheral vision. As shown in Figure 1-7, the HoloLens FOV limitation causes holograms to be limited to a small “window” through which they can be viewed. As headset display technology improves, we can expect resolution to increase and FOV limitations to reduce over time. The FOV limitation in today's devices means that you'll need to be creative when designing your app to be as immersive as possible. Using spatial sound is one key element to increasing the user's perception of holograms. Other visual cues (such as arrows pointing to holograms outside of the FOV) are also strategies for helping the user. I discuss these in more detail in Part III.



Figure 1-7. HoloLens field-of-view limitation, photographed from behind the device

Downloading and Installing the Required and Optional Software Tools

This section guides through installing all the necessary software and tools for developing Windows Mixed Reality applications. You only need two applications to get started with Mixed Reality development: Unity and Visual Studio. In this book, I define and frame everything in the context of Mixed Reality development. However, both these applications are widely used in the software and gaming industries and have been around for many years. Here's a brief description of each:

- *Unity* is the preferred software platform for developing Windows Mixed Reality experiences. All your app development happens within Unity. It's where you will program holograms to do things. Outside the Mixed Reality world, Unity is widely used for game development. This is excellent news because it means that there are years of tutorials, resources, and forum discussions to help answer almost any question you may have as you are developing your Mixed Reality applications.

- *Visual Studio* is primarily responsible for editing the code of your app and is also used to *deploy* your app to your Mixed Reality headset for testing and debugging. Deploying simply means installing the app to your headset. When your app is complete, you may also use Visual Studio to deploy your app to the Windows Store.

■ **Note** The tools shown in this book are updated regularly, so the screenshots may not look exactly like the most current versions of these applications. Keep that in mind as you follow the instructions.

In addition to installing Unity and Visual Studio, you will also need to download the HoloToolkit. The HoloToolkit is not an application but rather a collection of useful Mixed Reality scripts and features to import into Unity. Rest assured, I have an entire chapter dedicated to the HoloToolkit and all it has to offer.

Installing Visual Studio

This section walks through how to download and set up Visual Studio for Mixed Reality development. As of this writing, the version required for Mixed Reality development is Visual Studio 2017 or Visual Studio 2015 Update 3. Versions are updated regularly, but the installation process will be very similar between versions.

To check the latest version of Visual Studio for Mixed Reality development, use Microsoft's Installation Checklist, located at https://developer.microsoft.com/en-us/windows/Mixed-Reality/install_the_tools. You will see a table, similar to the one shown in Figure 1-8.

Installation checklist	
Download and Install	Notes
Visual Studio 2015 Update 3	If you choose a custom install, ensure that Tools (1.4) and Windows 10 SDK (10.0.10586) is enabled under Universal Windows App Development Tools node. All editions of Visual Studio 2015 Update 3 are supported, including Community.
HoloLens Emulator (build 10.0.14393.0)	The emulator allows you to run apps on Windows Holographic in a virtual machine without a HoloLens. Build 10.0.14393.0 includes the latest updates to the HoloLens OS. If you have already installed a previous build of the emulator, this build will install side-by-side. This package also includes holographic DirectX project templates for Visual Studio. Note: Your system must support Hyper-V for the Emulator installation to succeed. Please reference the System Requirements section below for the details.
Unity 5.5	Last known release: 5.5.0f3 on November 30th, 2016 The Unity engine is an easy way to get started building a holographic app .
Vuforia	Last known release: 6.1 issued November 16th, 2016 Vuforia enables you to create holographic apps that can recognize specific things in the environment and attach experiences to them. Review the getting started guide to learn how easy it is to extend the capabilities of your holographic apps with the Vuforia Engine. You can get a free development license at developer.vuforia.com .

Figure 1-8. Be sure to check Microsoft’s installation checklist for the most recent versions of tools to install. To download Visual Studio, click the corresponding title (circled).

You can also download Visual Studio from <https://developer.microsoft.com/en-us/windows/downloads>.

If you don’t already have a Visual Studio subscription, you can download the free version of Visual Studio, also known as Visual Studio Community. Figure 1-9 shows what the download button may look like.

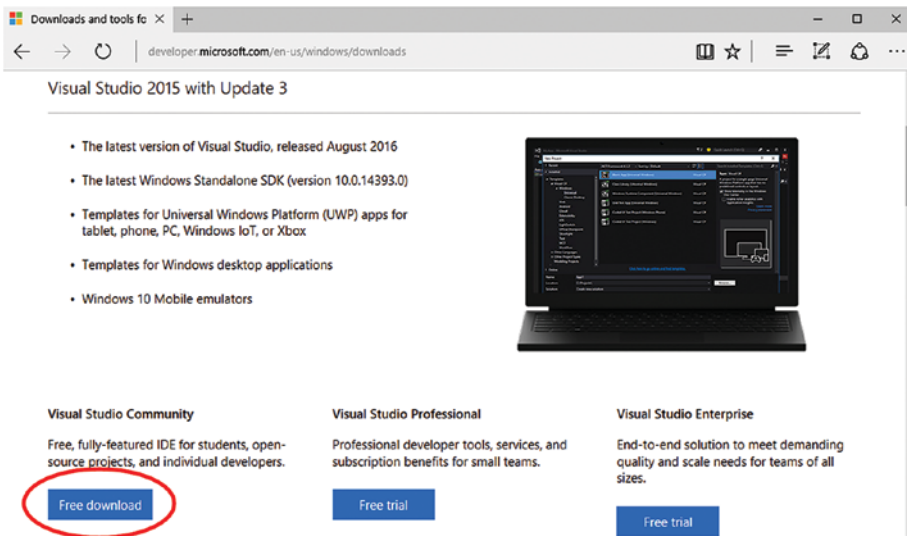


Figure 1-9. Click the “Free download” button for Visual Studio Community to begin your download

Save the Visual Studio installer to a location of your choice. After the download is complete, you may run the Visual Studio installer to begin the installation process. Figures 1-10 and 1-11 show how saving and running the installer appears in the Microsoft Edge browser.

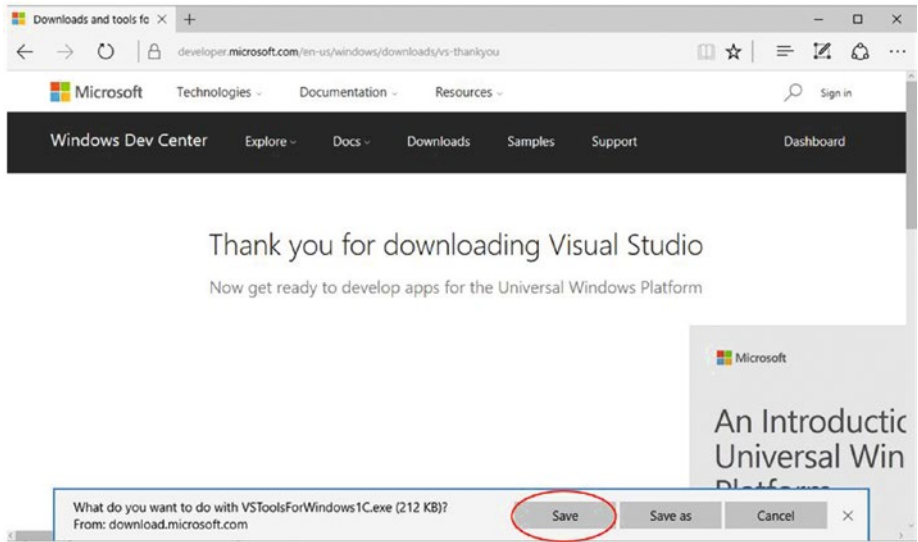


Figure 1-10. Save the Visual Studio installer to your PC

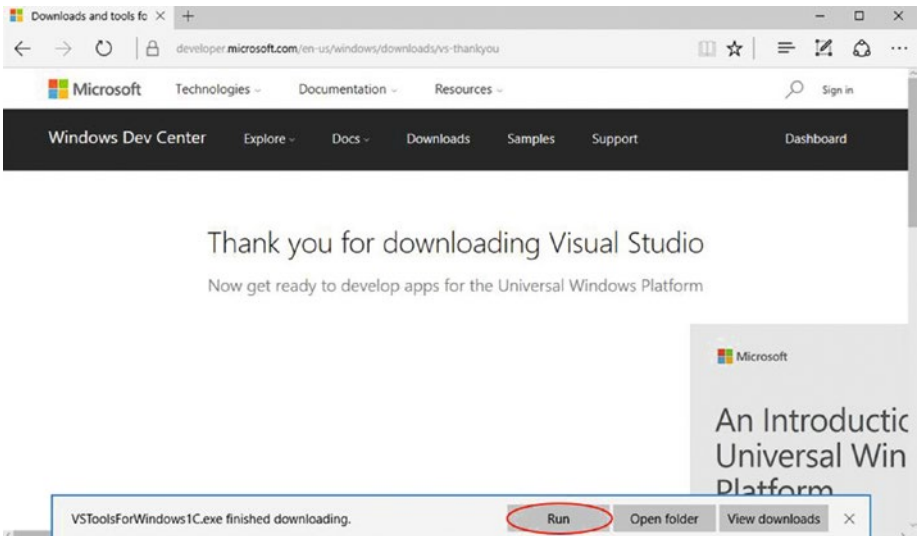


Figure 1-11. After the installer has saved to your PC, run the installer

When you launch the Visual Studio installer, you will encounter a few seconds of initialization followed by the option to select features, as shown in Figure 1-12a (for Visual Studio 2017) and Figure 1-12b (for Visual Studio 2015). Although Visual Studio 2017 is the latest edition as of this writing, I show both editions of Visual Studio here because both are currently supported for Windows Mixed Reality development, and there have been some reports of compatibility and stability issues with Visual Studio 2017, causing some developers to remain on the 2015 edition.

In the features list of Visual Studio 2017, be sure to select the “Universal Windows Platform development” checkbox. Also select the “Game development with Unity” checkbox. You may deselect the Unity Editor checkbox in the right panel because you will be installing the most current version of Unity later in this Chapter. Click Next or Install after making the appropriate selections.

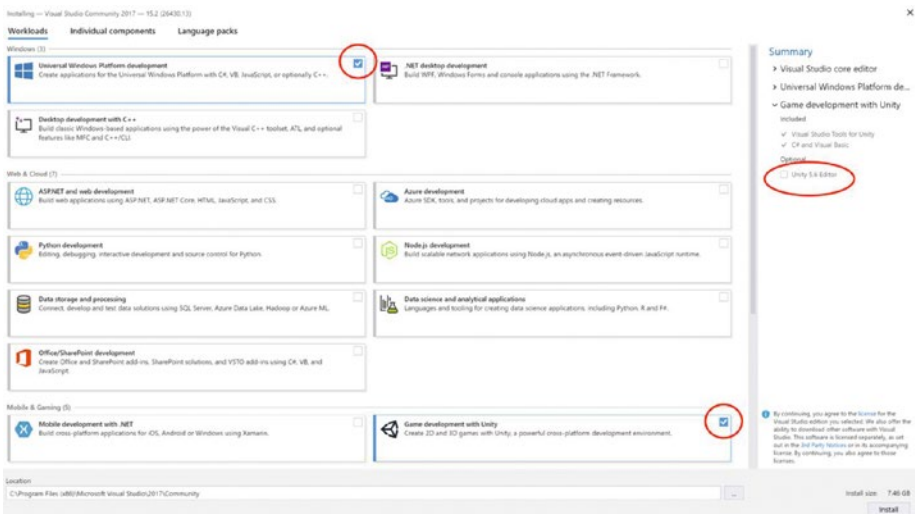


Figure 1-12a. For Visual Studio 2017, be sure you select “Universal Windows Platform development” and “Game development with Unity”

In the features list of Visual Studio 2015, be sure to select the Tools checkbox under the Universal Windows App Development Tools category, as shown in Figure 1-12b. The latest tools version as of this writing is 1.4.1. You may notice a higher version available. Be sure to refer to Microsoft’s installation checklist (mentioned earlier) for the appropriate tools version to select. Also select the most recent version of the Windows 10 SDK or the recommended version from Microsoft’s installation checklist. Click Next after making the appropriate selections. If you accidentally omitted an important selection, you can always re-launch the Visual Studio installer and install the missing features at a later time.

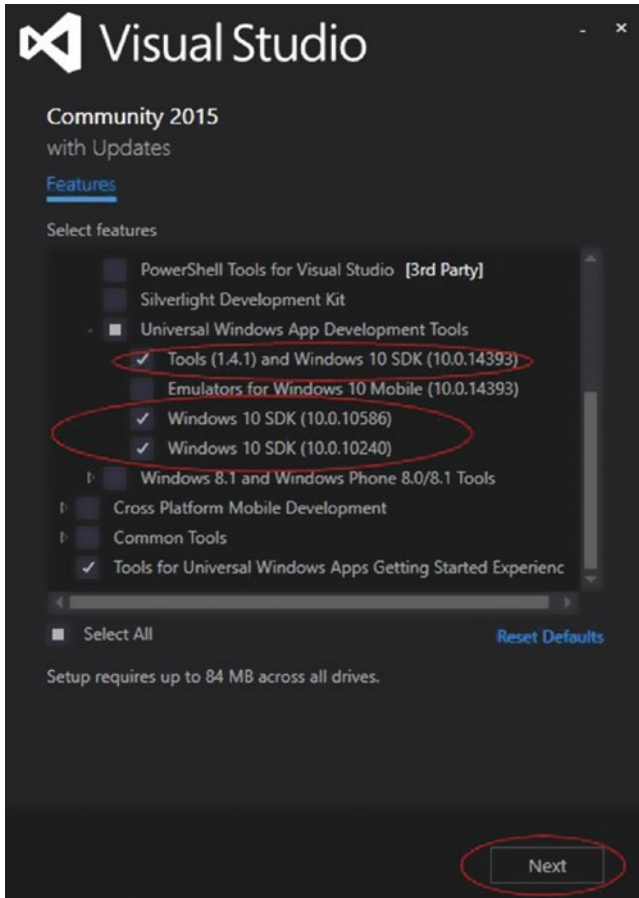


Figure 1-12b. For Visual Studio 2015 Update 3, be sure to enable Tools (version with the highest number) and all Windows 10 SDK options

After clicking Next once more to confirm your selections, Visual Studio will begin downloading and installing your selected features. Visual Studio is a very large application and could take several hours to download and install, depending on your Internet connection, so prepare for the installation process to take a while. After the installation has completed, you may be prompted to restart your PC. After restarting, you may verify that the installation completed successfully by opening Visual Studio. When you first open Visual Studio, it should look similar to the welcome page shown in Figure 1-13, depending on your edition of Visual Studio.

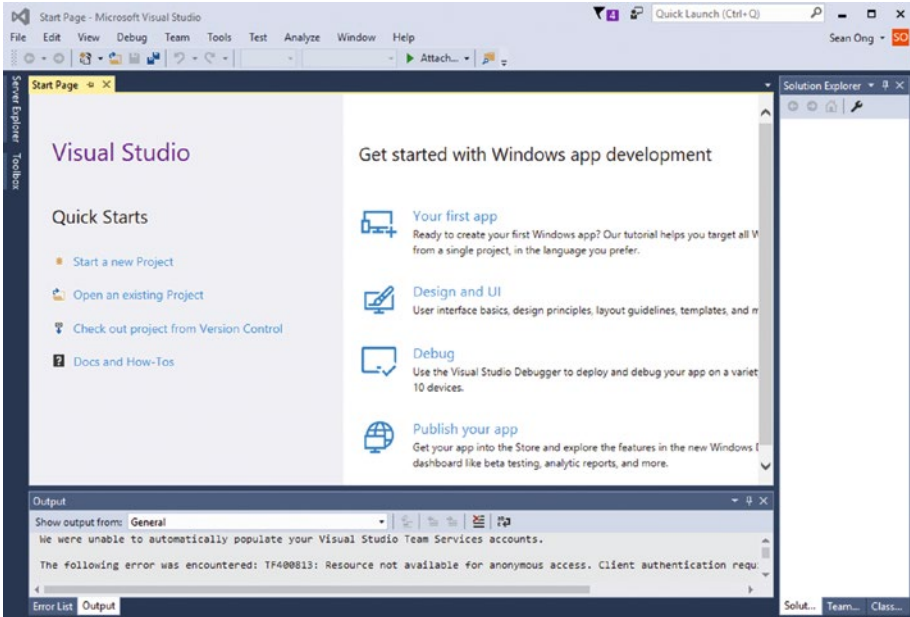


Figure 1-13. You may optionally verify that Visual Studio installed correctly by launching it after setup is complete

Congratulations! You have successfully installed Visual Studio. You don't need to do anything with Visual Studio for now. Later, when we deploy apps to our HoloLens via Visual Studio, we will go through an initial pairing process. If you haven't already, you may need to log in to Visual Studio with your Microsoft account. Visual Studio may also apply a license to your account to use Visual Studio. Next, we will install Unity.

Installing Unity

This section walks through how to download and set up Unity for Mixed Reality development. As of this writing, the version required for Mixed Reality development is Unity 5.5. Versions are updated regularly, but the installation process will be very similar between versions.

To check the latest version of Unity for Mixed Reality development, use Microsoft's installation checklist, located at https://developer.microsoft.com/en-us/windows/Mixed-Reality/install_the_tools. You will see a table, similar to the one shown in Figure 1-14.

Installation checklist

Download and Install	Notes
Visual Studio 2015 Update 3	If you choose a custom install, ensure that Tools (1.4) and Windows 10 SDK (10.0.10586) is enabled under Universal Windows App Development Tools node. All editions of Visual Studio 2015 Update 3 are supported, including Community.
HoloLens Emulator (build 10.0.14399.0)	The emulator allows you to run apps on Windows Holographic in a virtual machine without a HoloLens. Build 10.0.14399.0 includes the latest updates to the HoloLens OS. If you have already installed a previous build of the emulator, this build will install side-by-side. This package also includes holographic DirectX project templates for Visual Studio. Note: Your system must support Hyper-V for the Emulator installation to succeed. Please reference the System Requirements section below for the details.
Unity 5.5	Last known release: 5.5.0f3 on November 30th, 2016 The Unity engine is an easy way to get started building a holographic app .
Vuforia	Last known release: 6.1 issued November 16th, 2016 Vuforia enables you to create holographic apps that can recognize specific things in the environment and attach experiences to them. Review the getting started guide to learn how easy it is to extend the capabilities of your holographic apps with the Vuforia Engine. You can get a free development license at developer.vuforia.com .

Figure 1-14. Be sure to check Microsoft's installation checklist for the most recent versions of tools to install. To download Unity, click the corresponding title (circled).

You can also download Unity from <https://store.unity.com/download>.

Once you arrive at Unity's download page, you will see a download button similar to the one shown in Figure 1-15. Save the installer to your PC (Figure 1-16) and run the installer once the download has completed (Figure 1-17).



Figure 1-15. Download the latest version of Unity from the Unity website

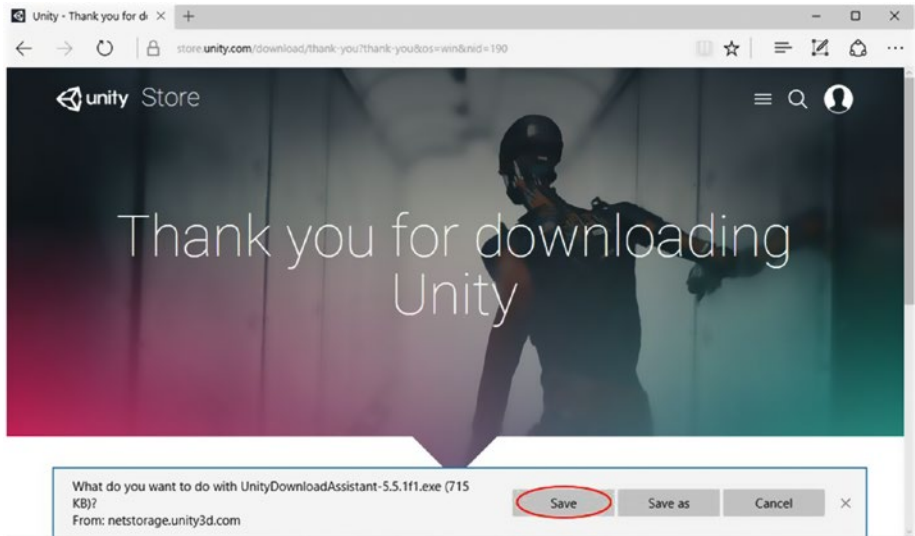


Figure 1-16. Save the installer to your PC

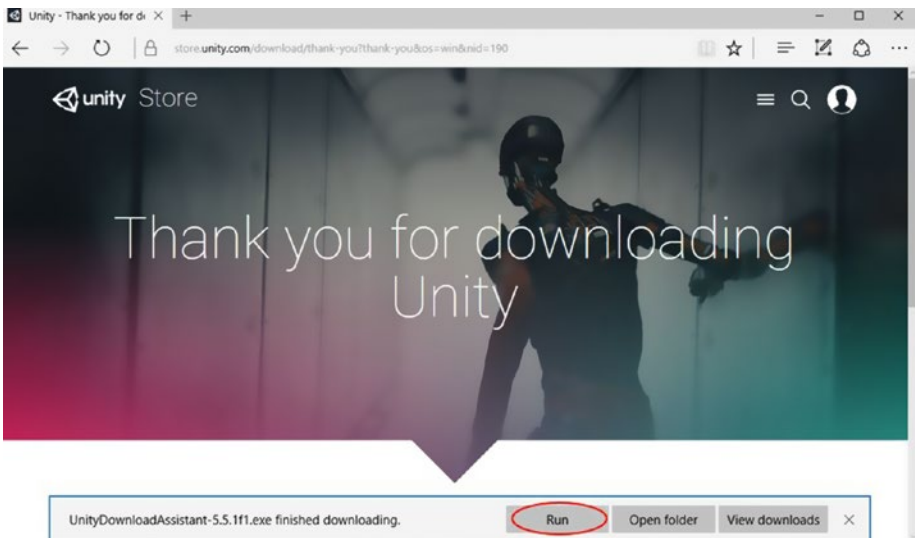


Figure 1-17. After the download is complete, run the Unity installer

After launching the Unity installer, you will be greeted with the Download Assistant, as shown in Figure 1-18. Click Next to begin the installation process.



Figure 1-18. Launch the Unity installer and click Next

On the next page, the Unity installer will show you the License Agreement. If you accept the terms of the agreement, check the checkbox and click Next, as shown in Figure 1-19.

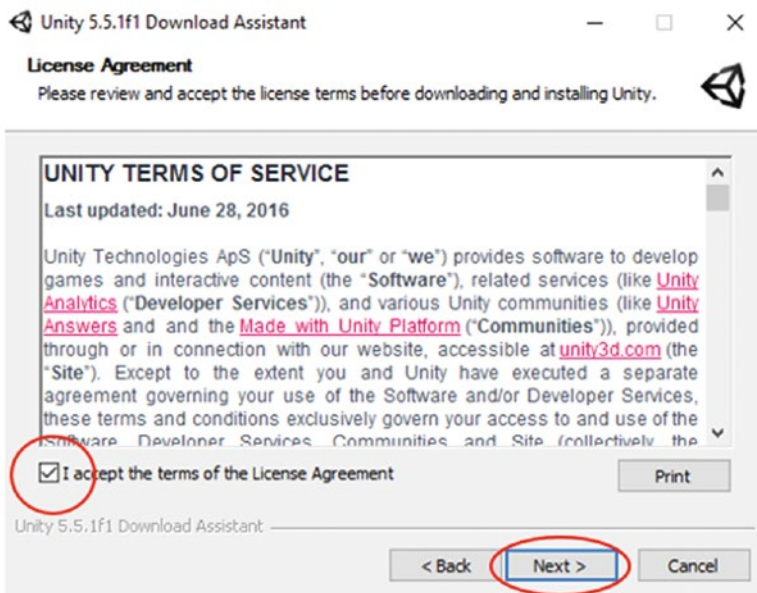


Figure 1-19. Read and accept the Unity License Agreement and click Next to continue

On the next screen, choose either the 64-bit edition or the 32-bit edition and click Next to continue, as shown in Figure 1-20. In general, you can select either 32-bit or 64-bit without any noticeable difference. If you have a 64-bit operating system, running the 64-bit edition of Unity will allow you greater access to RAM, and therefore enable better performance when editing extremely large scenes. For the HoloLens, though, it's unlikely that you will be developing extremely large scenes. The 32-bit edition may have better compatibility with some third-party plugins, should you ever decide to install plugins.

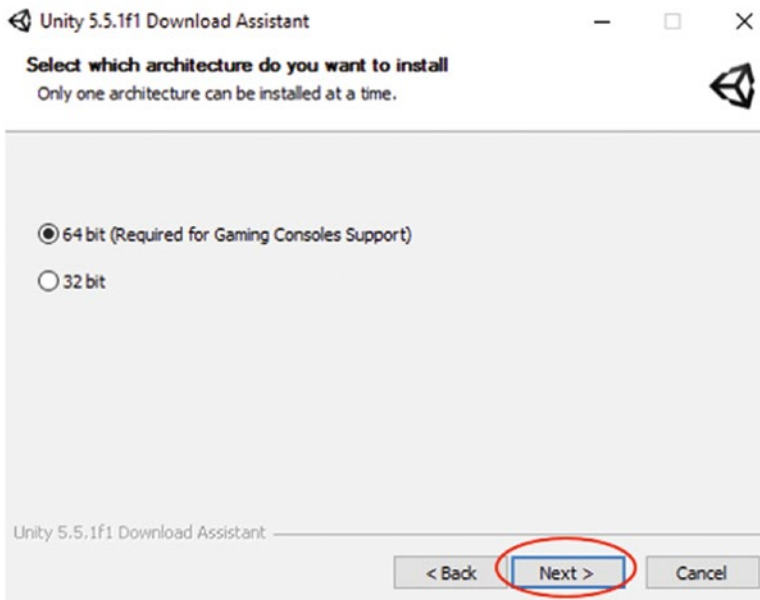


Figure 1-20. Choose 32-bit or 64-bit and click Next to continue

On the next screen, you'll be presented with several options to select. You should only have to select two additional options beyond the default selections:

- Windows Store .NET Scripting Backend
- Windows Store IL2CPP Scripting Backend

Make sure that Microsoft Visual Studio Tools for Unity is also checked (it should already be checked by default). See Figure 1-21 for all items that need to be checked before proceeding. Click Next to proceed.

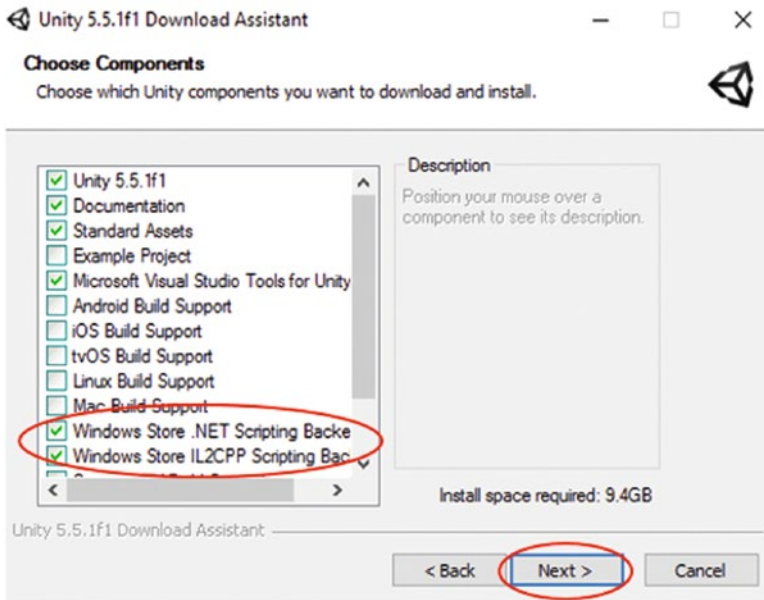


Figure 1-21. Be sure to select the required components shown in this figure. Click Next to continue.

As shown in Figure 1-22, Unity will begin to download and install the selected components. This may take several hours, depending on the speed of your Internet connection. Be prepared for a long wait. Once the installation is complete, click the Finish button.

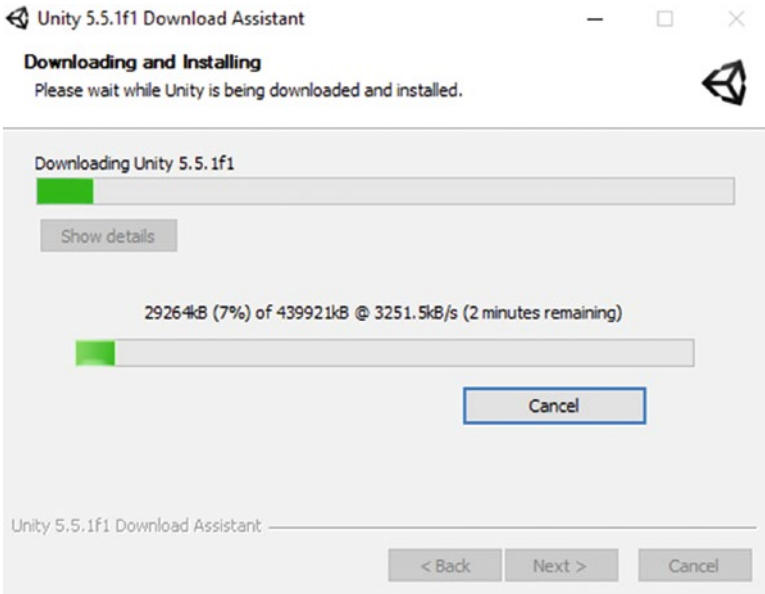


Figure 1-22. The downloading and installation of Unity components may take a long time

Congratulations! You now have Unity installed on your PC. Before using Unity, you will be required to create a Unity account (if you don't already have one). If you need to create a new Unity account, click the blue “create one” link as shown in Figure 1-23. Your web browser will open a page where you can sign up for a new Unity account, as shown in Figure 1-24.

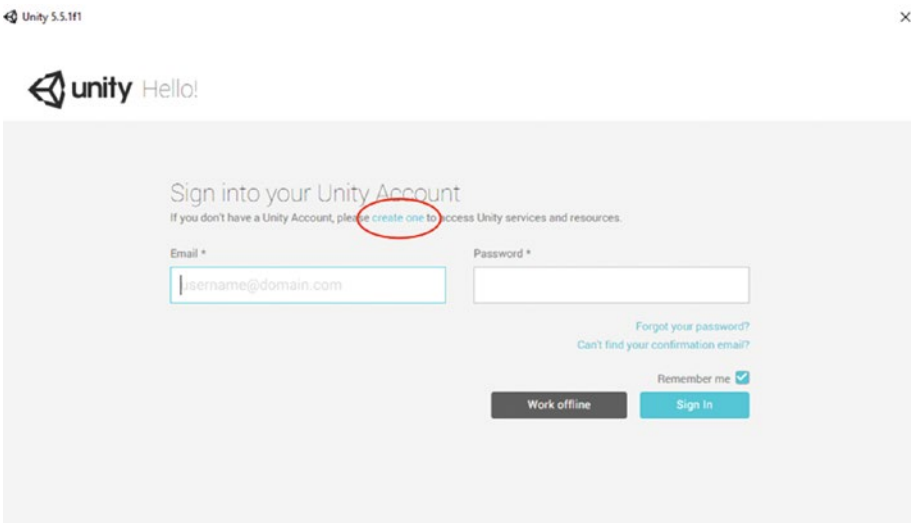
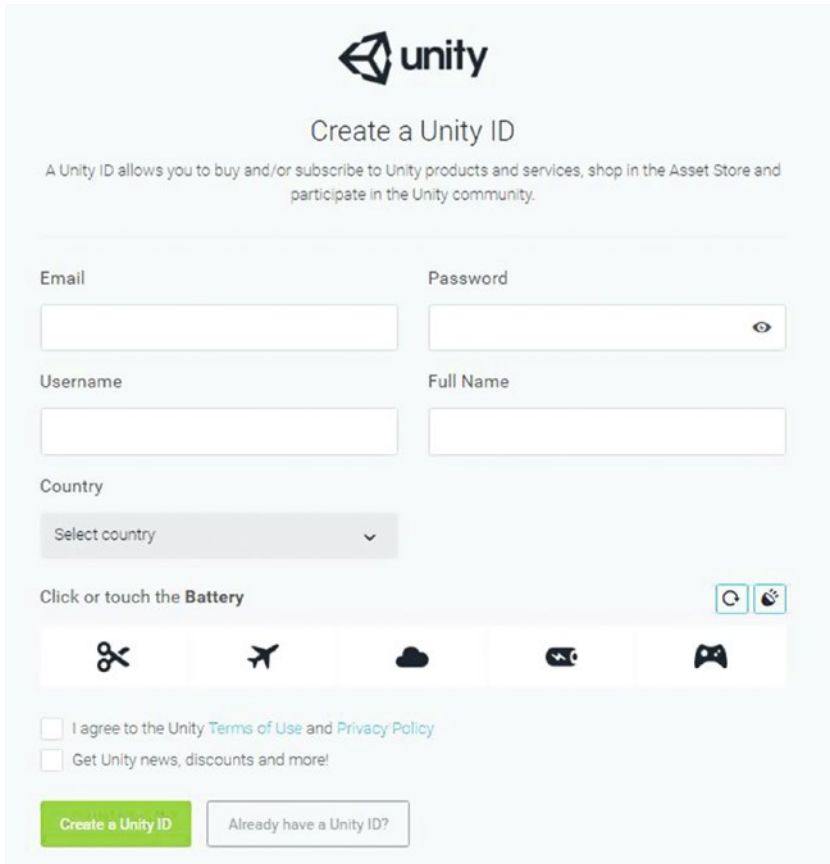


Figure 1-23. Log in to your existing Unity account or create a new one if you don't have an account



The image shows the Unity website's 'Create a Unity ID' page. At the top is the Unity logo. Below it, the heading 'Create a Unity ID' is followed by a description: 'A Unity ID allows you to buy and/or subscribe to Unity products and services, shop in the Asset Store and participate in the Unity community.' The form contains several input fields: 'Email' and 'Password' (with an eye icon for toggling visibility), 'Username' and 'Full Name', and a 'Country' dropdown menu labeled 'Select country'. Below these is a section titled 'Click or touch the Battery' with five icons: a pair of scissors, an airplane, a cloud, a battery, and a game controller. At the bottom, there are two checkboxes: 'I agree to the Unity Terms of Use and Privacy Policy' and 'Get Unity news, discounts and more!'. Two buttons are at the very bottom: a green 'Create a Unity ID' button and a grey 'Already have a Unity ID?' button.

Figure 1-24. Unity will open a web page where you can sign up for a new Unity account

Downloading the HoloToolkit

This section walks you through downloading the HoloToolkit, which isn't a program but rather a collection of useful scripts and features to import into Unity. To download the HoloToolkit Unity package, go to <https://github.com/Microsoft/HoloToolkit-Unity/releases>.

Note There are actually two HoloToolkit repositories online. The first is called HoloToolkit, and the second is called HoloToolkit-Unity. You can get them at the following links:

HoloToolkit: <https://github.com/Microsoft/HoloToolkit>

HoloToolkit-Unity: <https://github.com/Microsoft/HoloToolkit-Unity>

The HoloToolkit-Unity repository contains Unity-specific components and will be what we focus on throughout this book. The “regular” HoloToolkit is a generalized version of the HoloToolkit for developers that use other platforms for development. It contains the core C++ code base that many of the HoloToolkit-Unity features are built upon or are merely wrappers around. Throughout this book, I refer to the HoloToolkit-Unity as just the HoloToolkit.

Make sure you download the latest release of the HoloToolkit, typically located near the top of the page. Be sure that the HoloToolkit version you download is compatible with the version of Unity you downloaded. Typically, compatibility is announced in the title of the HoloToolkit version, as shown in Figure 1-25. To download the HoloToolkit Unity package, click the download link that has the extension .unitypackage. For example, in Figure 1-25, the appropriate download link (circled) is named HoloToolkit-Unity-v1.5.5.0.unitypackage.

Latest release

v1.5.5.0
0d477ff

HoloToolkit-Unity Compatible with Unity Engine 5.5.0f3+

HodgsonSDAS released this on Jan 10 · 172 commits to master since this release

Release Notes:

1. Includes new Extensible Input System.
2. Master Moved to Unity Engine 5.5.0f3.
3. Updated HoloLens Project Configuration Wizard.
4. Spatial Mapping Namespace updated and assets organized.
5. Shaders now support stereo instancing and near clip fading.
6. Will also provide a release version Unity package in the External\Unitypackages folder.

Breaking Changes:

1. No Longer compatible with Unity technical preview builds.
 - o Please use latest engine release from Unity.
2. All Input Systems updated to new Extensible Input System.

Downloads

HoloToolkit-Unity-v1.5.5.0.unitypackage	13.3 MB
Source code (zip)	
Source code (tar.gz)	

Figure 1-25. Browse to the HoloToolkit download page and download the HoloToolkit Unity Package, circled here

Save the HoloToolkit to your PC. We will import this package into Unity later.

Summary

You now have everything you need to get started with Mixed Reality development! In the upcoming chapters, we'll learn some basics of Unity, make our very first Mixed Reality application, and then start diving into the details of making amazing Mixed Reality experiences. Let's recap what you've learned in this chapter:

- I discussed the recommended PC hardware specifications for developing Mixed Reality experiences.
- We talked about various ways to test your Mixed Reality apps, including using headsets and emulation.
- We took a brief hardware tour of the HoloLens and other Windows Mixed Reality headsets, and what makes them unique.
- Finally, we went through step-by-step installation instructions for Visual Studio, Unity, and the HoloToolkit.

CHAPTER 2



Unity Crash Course

In this chapter, we'll dive into the world of Unity, the preferred software platform for developing Mixed Reality applications. If you want to master Mixed Reality development, you first need to master Unity. We'll take a tour of Unity and understand what Unity is used for. I'll also walk you through building your very first Unity app.

What Is Unity?

Before we begin, you might be wondering what Unity is and how people use it. *Unity is a powerful program for building both 2D and 3D games and apps.* It is very popular among game developers, especially mobile game developers. As of Q3 2016, more than 5 billion Unity games have been downloaded. Unity also supports a wide range of platforms. A few examples include iOS, Android, PlayStation, Nintendo, Xbox One, Windows, Mac, HoloLens, Oculus, and many more.

A very basic Mixed Reality workflow in Unity look like this:

- Import your 3D objects and other items (called *assets*) into Unity.
- Program how you will interact with the objects and how the objects will interact with you, other objects, and the world.
- Test your app.
- Export your app so you can install it on your device.

Unity is a very large, nuanced platform. In this book, we'll only cover the essential parts of Unity needed for Mixed Reality development. As you continue to grow as a Windows Mixed Reality developer, you'll invest much of your time into mastering Unity and programming for Unity.

Free vs. Paid Tiers of Unity

Several pricing tiers of Unity are available for developers. Most individuals looking to learn Unity can get started with a personal (free) account. However, there are restrictions and benefits to be aware of for the various pricing levels. Table 2-1 explains these tiers.

Table 2-1. *Unity Pricing Tiers*

Pricing Tier	Restrictions and Benefits
Unity Personal (Free)	You can only use this tier if your company makes less than \$100K per year (including investor funding). Your app is forced to have Unity's logo when launched (splash screen).
Unity Plus (\$35/person monthly)	You can only use this tier if your company makes less than \$200K per year (including investor funding). You can use a custom (or no) splash screen. Additional Plus services
Unity Pro (\$125/person monthly)	No revenue/finding cap. You can use a custom (or no) splash screen. Additional Pro services.

Your First Unity App

In this section, we'll build our very first Unity application—a game to control a rolling ball with your keyboard. As we walk through creating your first Unity app, You will also become familiar with some basic Unity components and learn about the Unity interface. Because this tutorial is intended to familiarize you with Unity, we won't be making a Mixed Reality application yet (we'll do that in the next chapter).

Before we begin, be sure that you have already installed Unity and set up your unity account per the instructions in Chapter 1.

■ **Tip** This basic Unity tutorial is known as the Roll-A-Ball tutorial. It's recommended for all Unity beginners. In addition to following along in this book, you can also access a video of this tutorial at the following link. For users new to Unity and 3D application development, I recommend completing the full tutorial: <https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial>.

Step 1: Create a New Unity Project

Every Unity (and Mixed Reality) project begins with creating a new Unity Project. To start a new project, launch Unity and select the NEW icon as shown in Figure 2-1. If you haven't already, you may need to enter in your account information (created in Chapter 1) prior to seeing the screen shown in Figure 2-1.

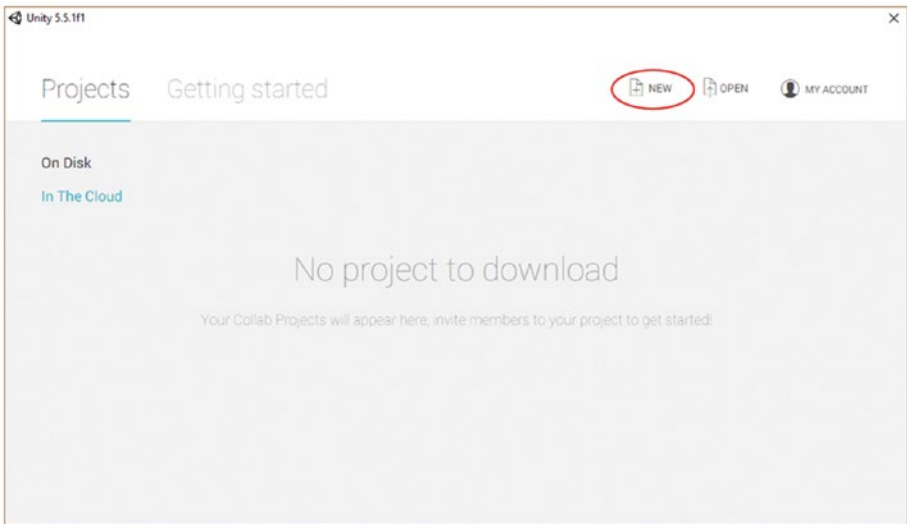


Figure 2-1. Click the NEW icon to start a new Unity project

Alternatively, if you're already within a Unity project, you can create a new project by going to File ► New Project, as shown in Figure 2-2.

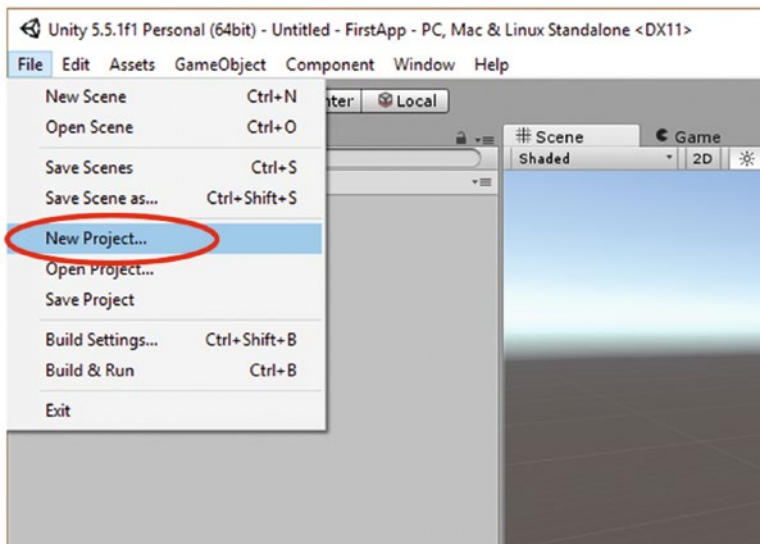


Figure 2-2. You can also start a new Unity project by going to File ► New Project

Unity will then open a pop-up window where you can set up your new project. As shown in Figure 2-3, give your new project a name. I chose FirstApp. If you want, you can also select a different location for your project to be stored. The project should already be a 3D project, but if not, be sure to select the 3D option button. When done, click the Complete Project button.

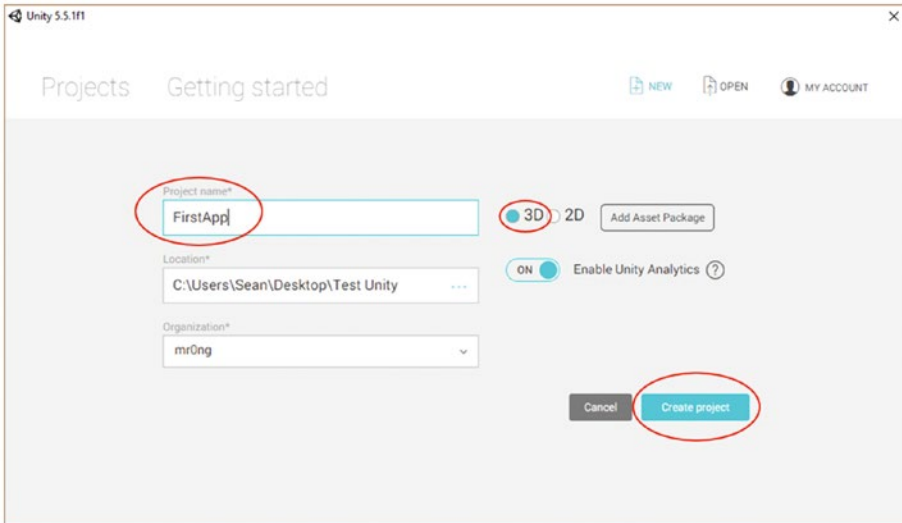


Figure 2-3. Tell Unity about your new project. Be sure to select 3D and give your new project a name.

■ **Warning** Avoid storing your Unity project on an SD card or MicroSD card. Visual Studio won't be able to properly build your project if it's on an SD card, due to an unknown bug. You may store your Unity project on an external hard drive.

You will now see a new empty project—something similar to Figure 2-4. This is called the *Unity Editor* and it's where most of your project editing will take place. Figure 2-4 also provides brief descriptions for important panels contained within the editor.

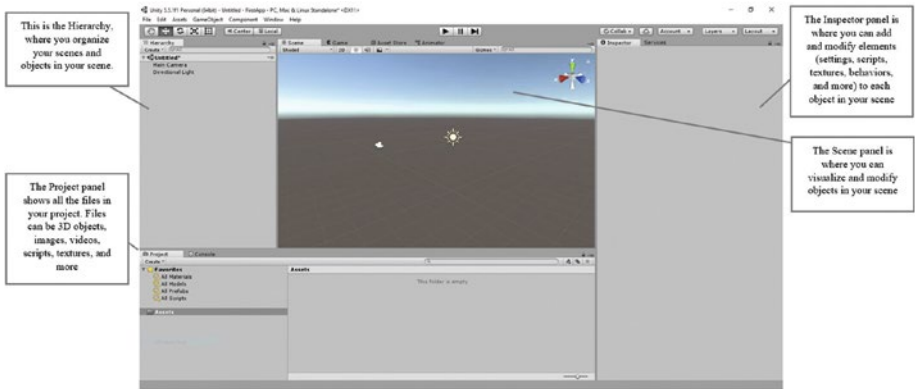


Figure 2-4. The Unity Editor window contains several important panels. This is where most Unity project editing occurs.

In your Scene panel, you will see something that resembles a blue sky and a brown or grey ground. This is your empty *scene*. Right now, it only contains some light (the sun icon) to illuminate the scene and the camera through which you see the world when operating the game (the camera icon).

Step 2: Save Your Scene

Before we begin editing this empty scene, let's save the scene. Go to File ► Save Scenes. Figure 2-4 shows where to find the Save Scenes option. A dialog box will pop up, allowing you to name your scene and choose a location to save it. Name your scene *MiniGame* and feel free to use the default location. You can name your scene with any name, but I recommend using the same name I use so that it will be easier for you to follow along as we go through this tutorial.

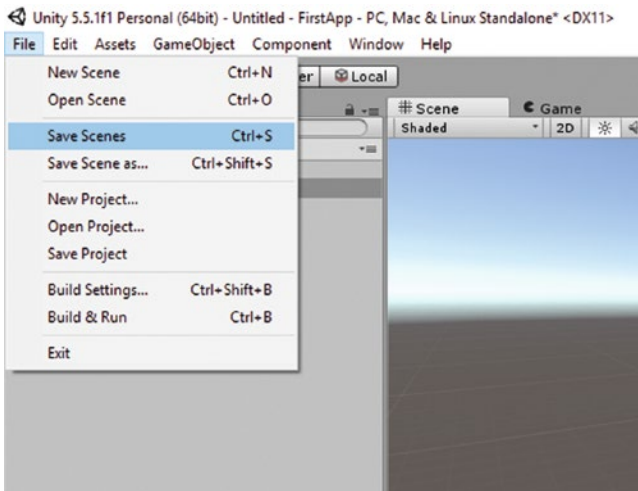


Figure 2-5. Save your scene before editing it

The scene is now saved as `MiniGame.Unity` in your assets folder. In your Hierarchy panel, you should now see your scene named `MiniGame` with two objects under it (Main Camera and Directional Light).

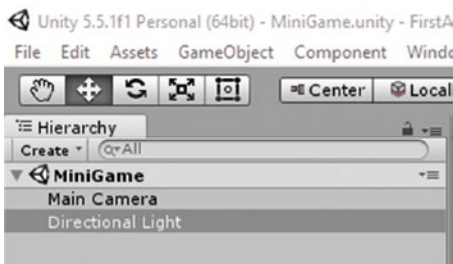


Figure 2-6. Your new scene should have two default objects under it in the Hierarchy panel

Step 3: Create a Ground Plane

We can now start building out our game scene. First, we'll need to create the *ground plane* for the ball to roll on. Click the `MiniGame` scene in the Hierarchy so that it's highlighted (to ensure that you're not highlighting the Main Camera or Directional Light). Go to `GameObject` ► `3D Object` ► `Plane`. See Figure 2-7 for where to access the Plane object.

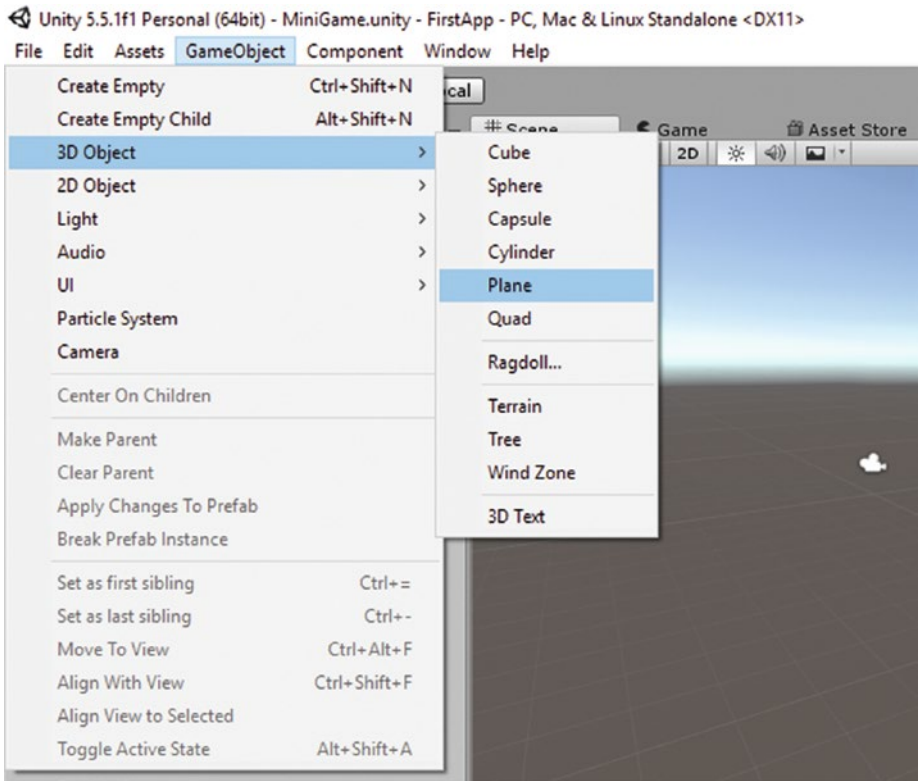


Figure 2-7. Create a new plane for the ground

The Plane object you just created will appear in your scene panel as a white plane on the ground. It will also appear in your Hierarchy panel, as shown in Figure 2-8. Objects in the Hierarchy are referred to as *game objects*.

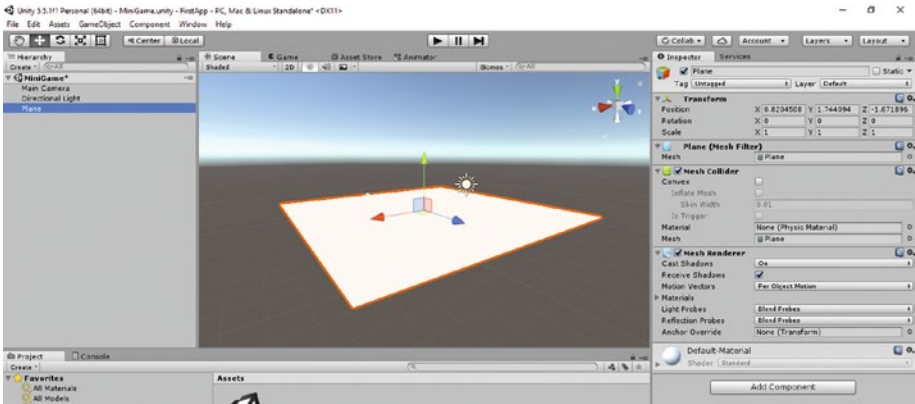


Figure 2-8. Illustration of what the Plane object looks like after it is created

Step 4: Rename Your Plane

Let's rename our Plane game object to Ground. To do this, highlight (click) the Plane game object in the Hierarchy once. After waiting one second, click the Plane game object again. You will now be able to edit the name of the game object. You may also rename the object by right-clicking and selecting Rename. Rename it to Ground, as shown in Figure 2-9.

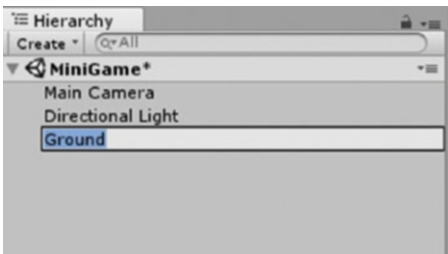


Figure 2-9. Rename the Plane game object to Ground

Step 5: Reset Ground Plane Position

Next, let's reset the position of our Ground game object to 0,0,0. In the Inspector panel, click the gear icon for the Transform element. After the context menu opens, select Reset, as shown in Figure 2-10.

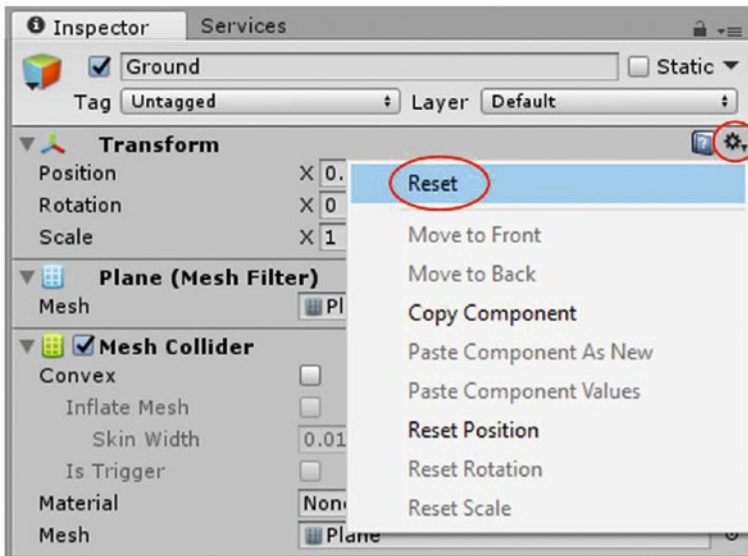


Figure 2-10. Reset the Ground game object's position

Step 6: Zoom to Your Ground Plane

In the Hierarchy, select the Ground game object and press *F* on your keyboard. Doing this activates the Frame Selected command, which causes the scene window to zoom in or zoom out so that the game object you selected fills your scene window.

■ **Tip** If you can't find your object in the scene, just select the game object's name in the Hierarchy and Type *F* to automatically zoom to your object. This command is also helpful for quickly zooming to very large or very small objects in your scene.

Step 7: Scale Your Ground Plane

Next, we'll learn how to scale game objects by scaling the Ground plane. To *scale* an object means to resize it. There are several different ways to scale game objects. I've listed some common methods below, and Figure 2-11 illustrates where to find these scaling commands:

- Press the scale icon and then click and drag one of the colored axes on the game object. Dragging the red axis scales in the X-direction, green for the Y-directions, and blue for the Z-direction.
- Press *R* on your keyboard as a shortcut to pressing the scale icon.

- Click and drag the X, Y, or Z titles of the Scale fields.
- Directly type in the scale of your choice in the Scale fields.

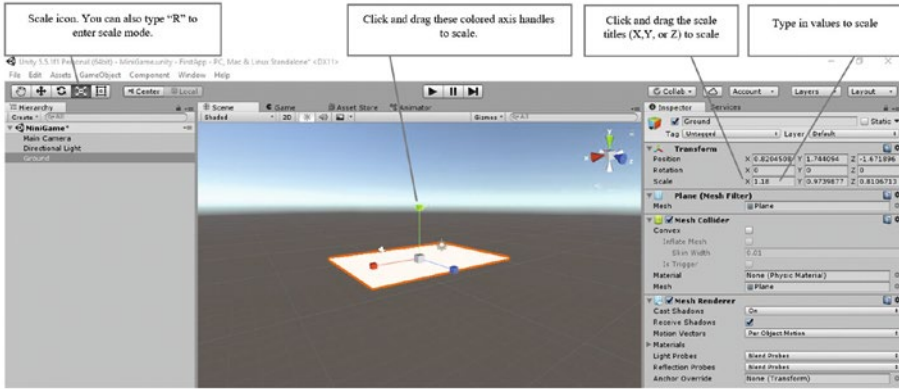


Figure 2-11. There are many ways to scale your game object

Scale your object so that the X, Y, and Z axes are all set to a scale of 1.

Step 8: Create the Ball

Remember, we're creating a game called Roll-A-Ball. You will be controlling a rolling ball on a flat surface. To create a ball, we need to add a sphere to our scene. Adding a sphere is similar to adding a plane. Click our scene, MiniGame (to make sure we've not selected another game object), and then go to **GameObject** ► **3D Object** ► **Sphere**. Figure 2-12 illustrates where to find this command.

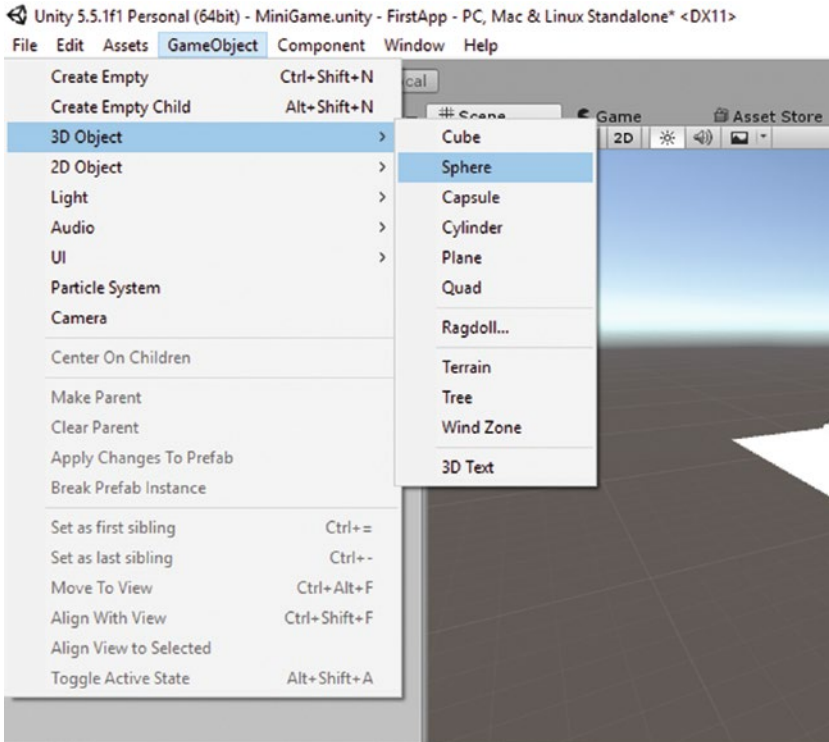


Figure 2-12. Create a Sphere game object for your ball

Step 9: Rename Your Ball

Rename your ball by selecting the Sphere game object in the Hierarchy and clicking it again after one second, just like you renamed your Plane in Step 4. Name your sphere Player.

Step 10: Reset the Ball's Position

Next, let's reset the position of our Player game object (our ball) to 0,0,0. In the Inspector panel, click the gear icon for the Transform element. After the context menu opens, select Reset, just as we did in Step 5 (see Figure 2-10.)

Step 11: Zoom to Your Ball

In the Hierarchy, select the Player game object and press *F* on your keyboard to zoom in or zoom out so that the ball fills your scene window.

Step 12: Raise the Ball's Position

As you can see in the scene, the ball is halfway into the Ground plane. Moving a game object's position is similar to scaling it (see Step 7). Figure 2-13 shows several options for moving an object's position. Raise the Player game object up 0.5 units. After doing so, the ball will rest perfectly on the surface of the Ground plane, as shown in Figure 2-13.

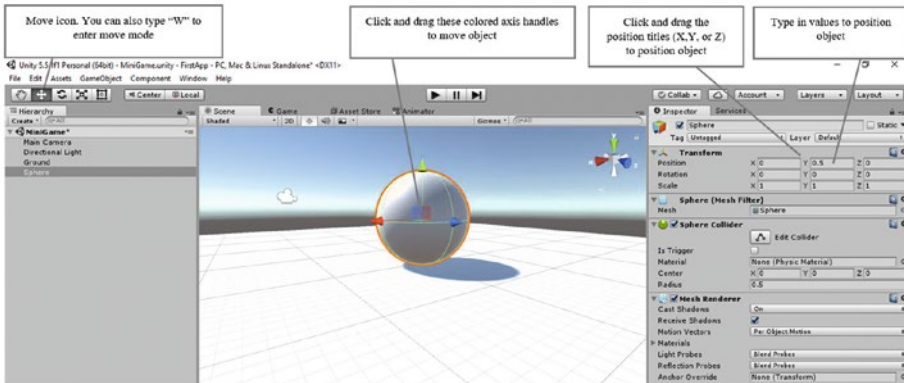


Figure 2-13. There are a variety of ways to position game objects within Unity

Step 13: Color the Ground Blue

So far, you've created the ground plane and a ball. The default color of the objects you've created is white. Let's change some colors so that we can distinguish better between the Player ball and the Ground plane. Within Unity, there are many options for modifying the appearance of your object. There's an intricate world of textures, materials, and shaders. This chapter won't cover these. For now, let's apply a simple material to our Ground plane. First, let's create a folder to organize our materials. It's always a good practice to keep your project files organized, which will help avoid confusion and speed up development work, especially in very large and complex projects. To create a folder, click the Create drop-down list in the Project panel and select Folder, as shown in Figure 2-14.

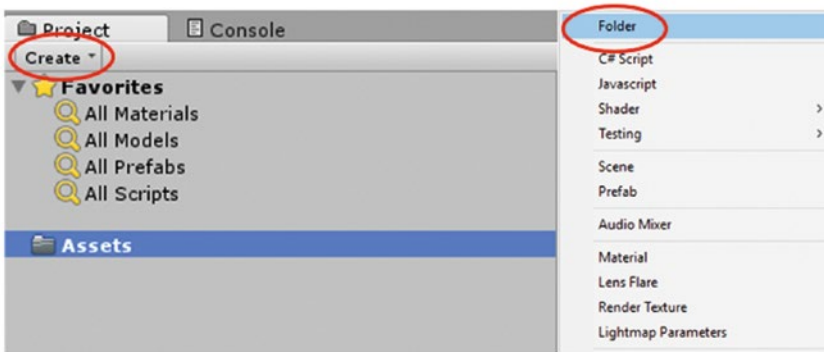


Figure 2-14. Creating a new folder in Unity

Once the folder is created, rename the folder to Materials, as shown in Figure 2-15.

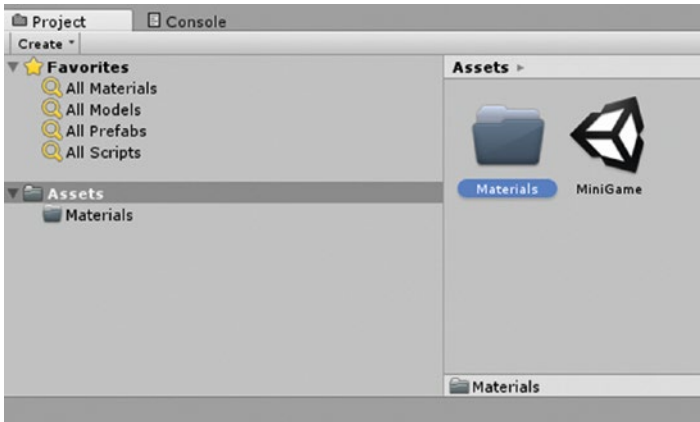


Figure 2-15. After renaming, you will now have a Materials folder in your project list

Next, we will create the material. Select the Materials folder you just created so that it's highlighted. Using the same Create drop-down menu that you used to create the materials folder, create a new material, as shown in Figure 2-16. The new material you created should now be inside the Materials folder (because the Materials folder was selected when you created the new material). Rename the Material to Background in the same way you've renamed other items during this tutorial.



Figure 2-16. Creating a new material

Next, select your new Background material and open the Albedo setting in the Inspector, as shown in Figure 2-17. To open the Albedo setting, click the colored box (the default color is white) to the right of the word Albedo. Don't click the eyedropper icon or the grey box on the left side of Albedo. A pop-up window will open where you can select various color options for your material, as shown in Figure 2-17.

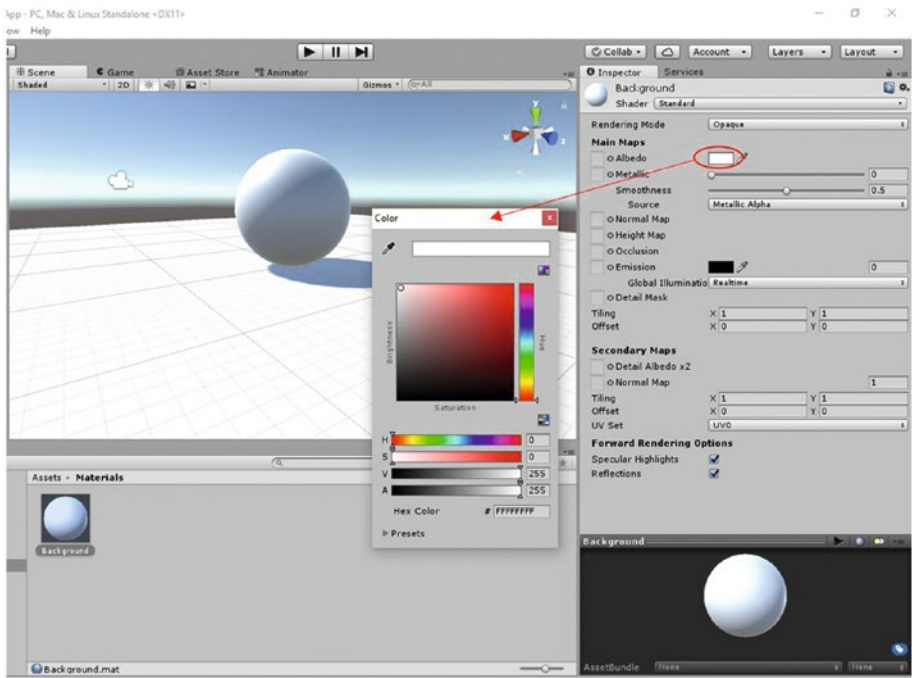


Figure 2-17. Select the Background material's Albedo setting to change its color

Within the pop-up color window, choose a dark blue color. You may need to drag the Hue bar to blue and then select a color within the square color box, as shown in Figure 2-18. Of course, you may choose any color you like.

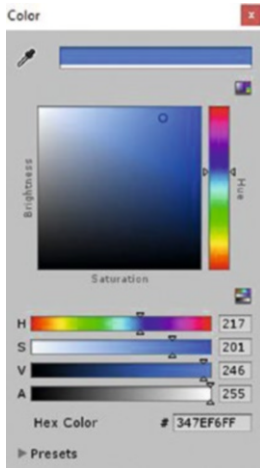


Figure 2-18. Choose a dark blue color by changing the Hue and Brightness/Saturation settings. You may also type in the specific numerical values shown

Next, apply the blue Background material to the Ground plane by dragging the material from the Project window to the Ground plane, as shown in Figure 2-19. The ground plane should become blue, allowing the ball to stand out better.

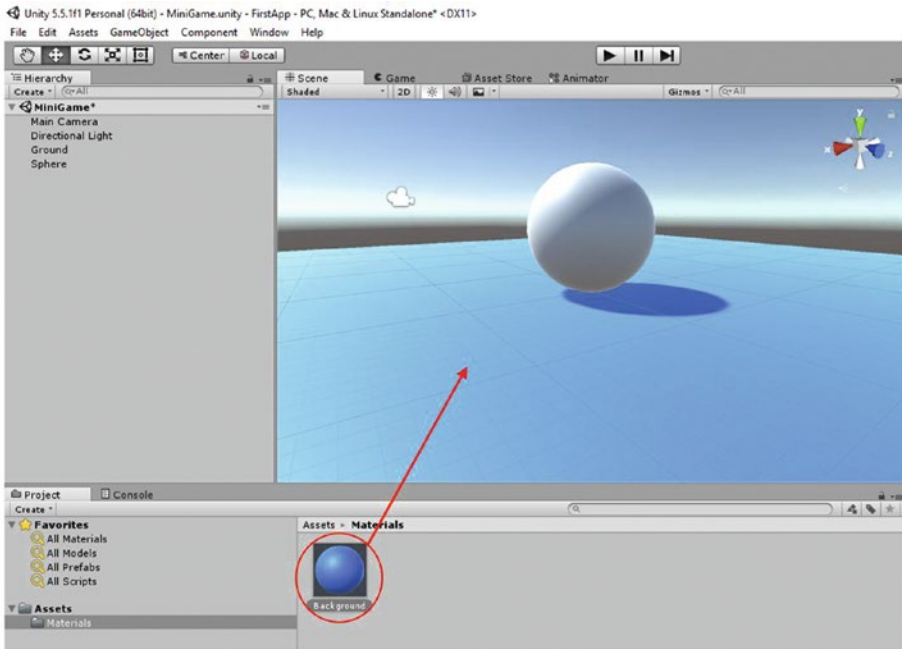


Figure 2-19. Drag and drop your blue Background material from the Project panel to the Ground plane, and the Ground plane will turn blue

Step 14: Add Physics to the Ball

Because this game will involve rolling our ball on our Ground plane, we want our ball to behave somewhat like a ball would in the real world. This means we want to use Physics. To use Physics, select our ball (which is the Player game object) in the Hierarchy, click the Add Component button in the Inspector, and select Physics ► Rigidbody. See Figure 2-20 for the location of these menu items. *Rigidbody*s are a fundamental component of the Unity physics engine and are responsible for storing various state variables needed for the equations of motion.

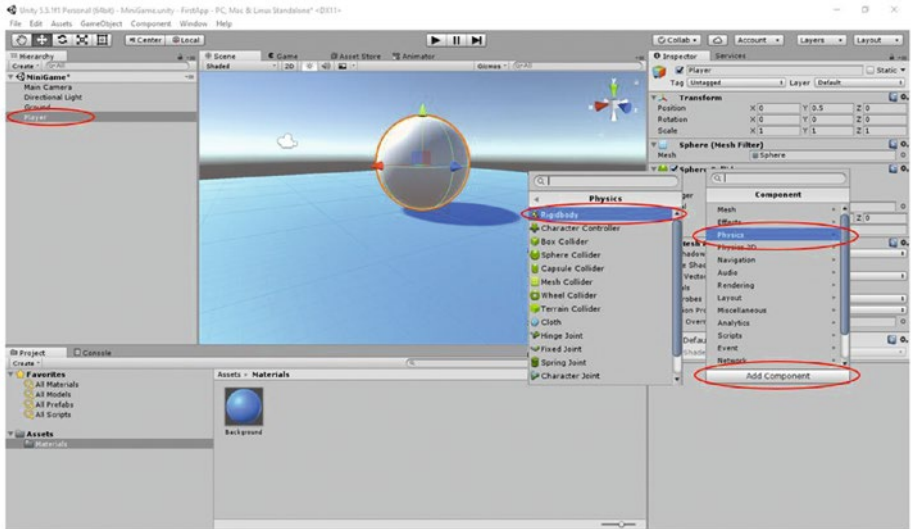


Figure 2-20. Add the Rigidbody component to the Player game object to apply Physics to our ball

Step 15: Enable Keyboard Control

We want to be able to move our ball using our keyboard as a game controller. To achieve this, we will need to apply some code to our ball. In Unity, our code documents are called *scripts*. Let's stay organized by creating a new folder to store our scripts, just as we created a new folder to store our materials in Step 13.

- To create the new folder, go to your Project panel and click Create ► Folder. Be sure that the scripts folder is created in the Assets folder and not inside the Materials folder.
- Rename your new folder Scripts.

Next, we want to add a new script to our ball:

- Select the Player game object.
- Click the Add Component button in the Inspector.
- Scroll to the bottom of the list and select New Script.
- In the next window that opens, be sure that the language of the script is set to C Sharp (usually spelled C#).
- Name your script PlayerController.
- Click the Create and Add button.

Figure 2-21 illustrates how to add and name your new script.

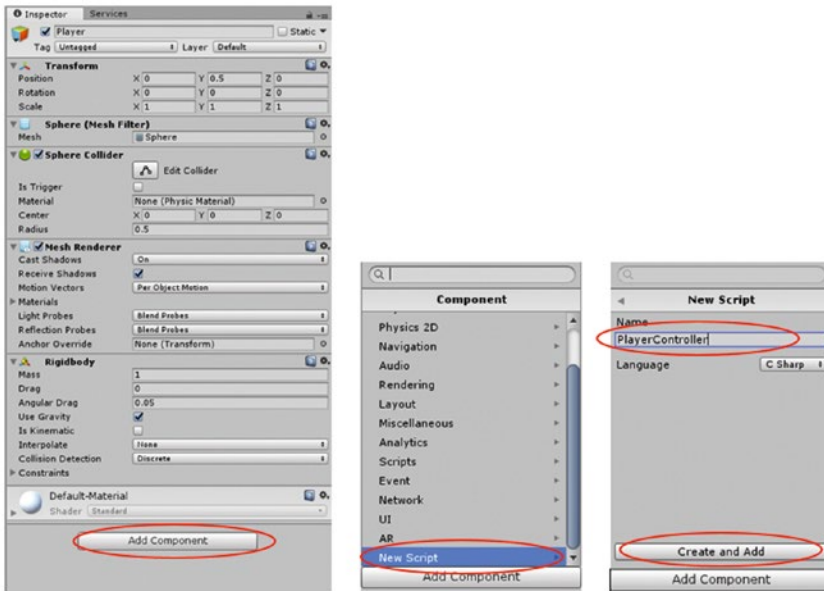


Figure 2-21. Create a new script for your Player game object called PlayerController

You'll notice in your Project panel that the new script you just created was not placed inside your Scripts folder. Drag and drop the PlayerController script inside your Scripts folder, as shown in Figure 2-22. Getting into the habit of staying organized is important.

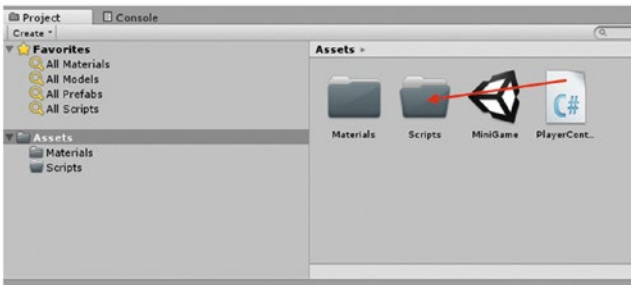


Figure 2-22. Drag and drop the PlayerController script inside your Scripts folder to stay organized

Next, we'll program our Player game object to respond to keyboard control:

- Select the Player game object in the Hierarchy.

- Double-click the `PlayerController` script in the Inspector, as shown in Figure 2-23. Visual Studio should now launch, so you can start editing your script.

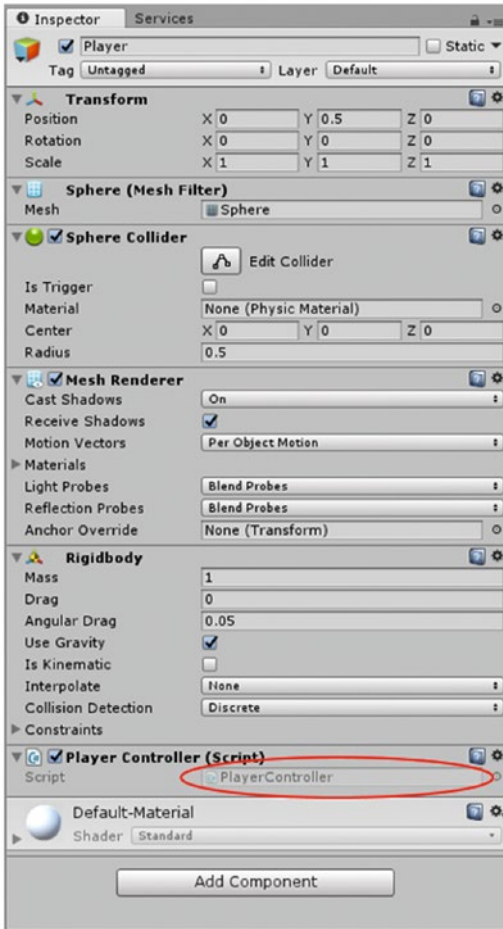


Figure 2-23. Double-click the `PlayerController` script to open it in Visual Studio for editing

You should have already downloaded, installed, and set up Visual Studio in Chapter 1. If you haven't already set up your account in Visual Studio, you will be asked to sign in or set up a new account. Once Visual Studio opens your `PlayerController` script, you should see a window similar to Figure 2-24.

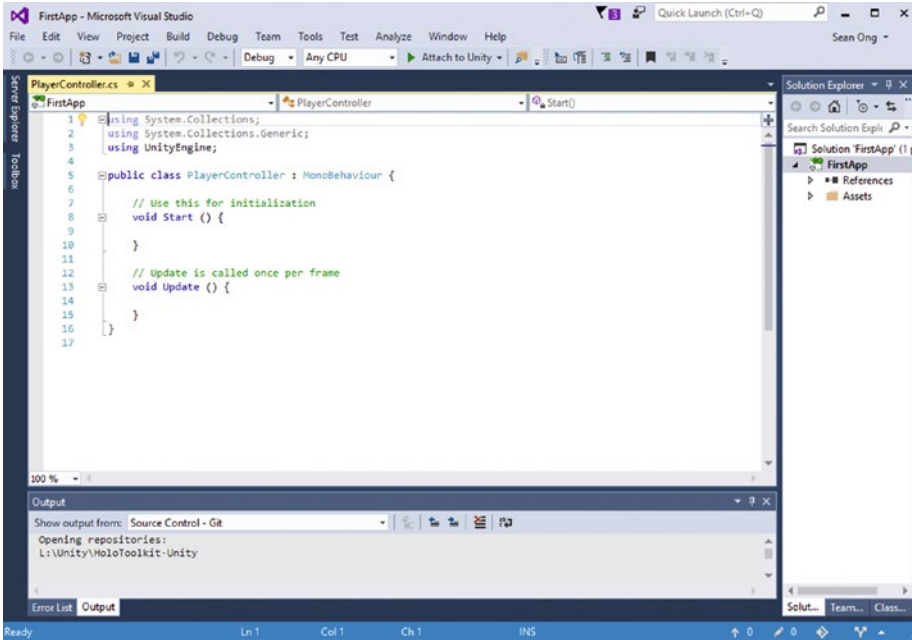


Figure 2-24. Example of what you should see after Visual Studio opens your *PlayerController* script

In Visual Studio, go ahead and erase all the code you see and replace it with the code in Listing 2-1.

Listing 2-1. Code for moving the ball

```

using UnityEngine;
using System.Collections;

public class PlayerController : MonoBehaviour {

    public float speed;

    private Rigidbody rb;

    void Start ()
    {
        rb = GetComponent<Rigidbody>();
    }
}
```



```

void FixedUpdate ()
{
    float moveHorizontal = Input.GetAxis ("Horizontal");
    float moveVertical = Input.GetAxis ("Vertical");

    Vector3 movement = new Vector3 (moveHorizontal, 0.0f, moveVertical);

    rb.AddForce (movement * speed);
}
}

```

Your Visual Studio window should now look like Figure 2-25 (the same code is shown in Figure 2-25 and Listing 2-1). Save your code by clicking the Save All icon, circled in Figure 2-25.

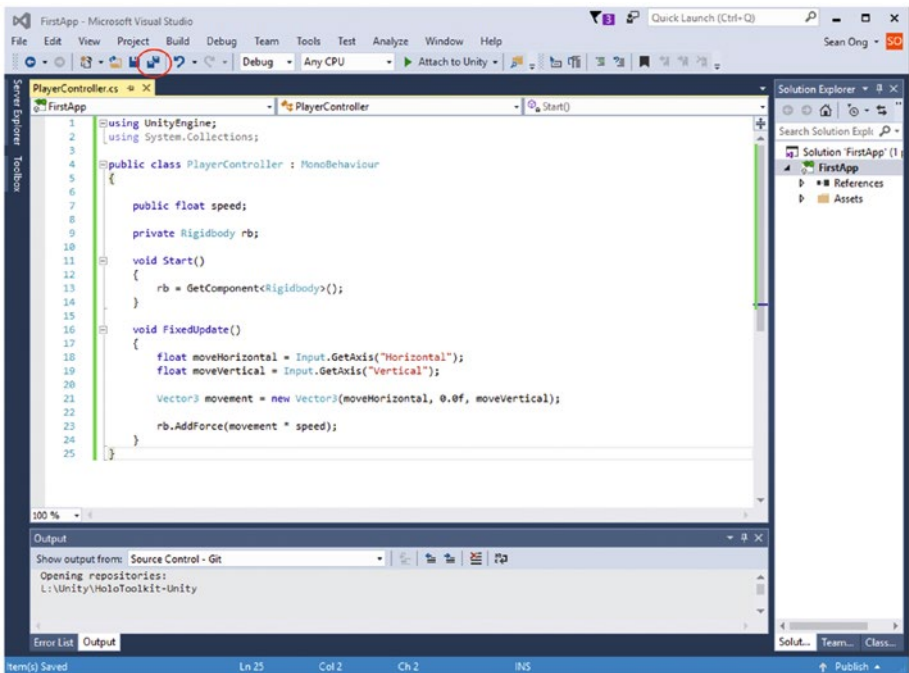


Figure 2-25. Replace all the default code in *PlayerController* with the code provided. Save your script by clicking the save all icon, circled in red.

I won't go into the explanation of this code in this tutorial, because this chapter is primarily intended to familiarize yourself with Unity's workflow. In later chapters, as we begin developing Mixed Reality applications, I will walk through scripts in detail so that you understand how the code works. If you are still curious how the previous code works, you'll find a detailed video walk-through of it at <https://unity3d.com/learn/tutorials/projects/roll-ball-tutorial/moving-player?playlist=17141>.

Step 16: Testing Your App

Next, we'll test our new app to ensure everything works as designed. Go back into the Unity editor and select the Player game object in the Hierarchy. You'll notice a new field has appeared under the Player Controller in the Inspector, as shown in Figure 2-26. This is due to the new script that we programmed. Specifically, the addition of the line `Public float speed` allows the value of Speed to be changed in the Inspector. For this to show up in the Inspector panel, you will need to save your script and allow Unity a few seconds to process the saved changes and update the Inspector panel. Change the value of Speed to 10 and click the play button (located above the scene panel), as shown in Figure 2-26.

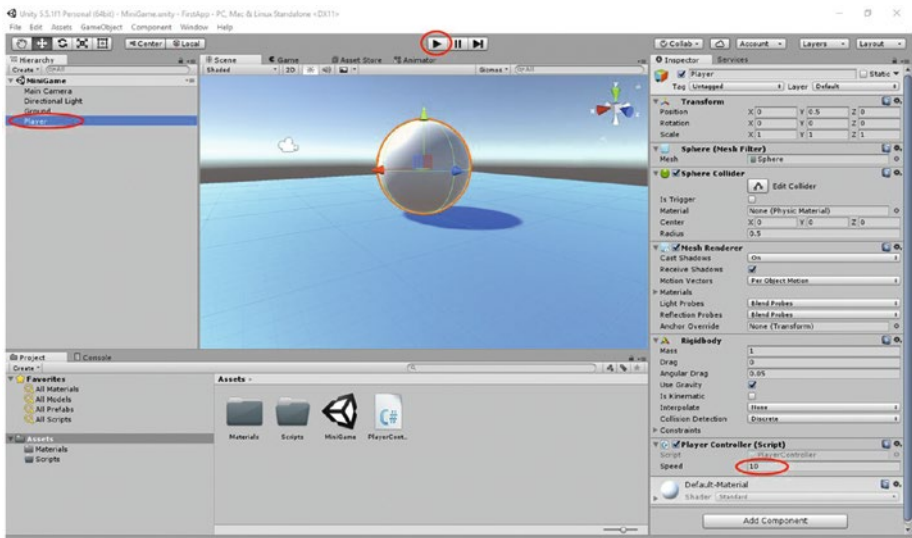


Figure 2-26. Set the Player's Speed to 10 in the Inspector and then click the play button (top) to test your new app

After clicking play, you will enter Game mode, where you will view the game through the scene's camera. Go ahead and try pressing the left, right, up, and down arrow keys on your keyboard and watch the ball move! If you fall off the edge of the Ground plane, you can always reset the game by clicking play again to stop the game, and then clicking play once more to enter the game. Feel free to explore how changing the speed of your Player impacts gameplay.

Summary

Congratulations! You have successfully completed your very first Unity application. Here are some things we've learned in this chapter:

- We discussed some basic information about what Unity is.
- We looked at Unity's pricing tiers.
- We learned about the Unity's editor interface.
- We learned how to create new game objects.
- We learned how to scale and move game objects.
- We learned how to apply materials to game objects.
- We learned how to apply physics to game objects.
- We learned how to create and apply scripts to game objects.
- We created and tested our first Unity app.

Now that you're familiar with the basics of Unity, we can start exploring how to create Mixed Reality experiences with this powerful platform. Unity is a very large and nuanced tool, and you will spend much of your time as a Mixed Reality developer refining your Unity skills. As we walk through Mixed Reality development in upcoming chapters, we'll also dive deeper into various Unity topics.

PART II



Building Holographic Experiences

CHAPTER 3



Creating Your First Hologram

In this chapter, you'll build your very first holographic experience. We'll begin by setting up Unity for Mixed Reality development using the HoloToolkit. After creating our first hologram, we'll test our app by deploying directly to our HoloLens using Visual Studio. We'll also test the app using Unity's holographic remoting and holographic simulation.

Getting Unity Ready for Mixed Reality Development

Before we begin creating our first hologram, we need to make sure Unity is ready for Mixed Reality development. There are several settings in Unity that need to be changed for our apps to work on the HoloLens and other Mixed Reality devices. For example, in Chapter 2's Unity tutorial, you may have noticed a grey/brown floor and blue sky in the scene. We usually won't want this digital floor and sky to appear in all our Mixed Reality experiences, so we need to black out our background so that it doesn't appear in our device. We also need to adjust our camera settings, so that each eye sees a slightly different perspective of our scene, which will allow users to perceive depth when wearing the headset. These are just a few examples of settings that need to change to prepare Unity for Mixed Reality development. All these settings can be changed manually, but it would be very tedious and time consuming to do this each time you create a new Mixed Reality project.

Fortunately, Microsoft provides a community resource called the HoloToolkit, which helps you automatically set up Unity for making Mixed Reality apps. There's an entire chapter on the HoloToolkit in this book, so I won't cover all it has to offer in this chapter. The following steps will walk you through preparing your scene for Mixed Reality development.

Note The HoloToolkit is updated regularly, and some elements may have changed since these instructions were written. Be sure to check the HoloToolkit documentation for updated instructions if you can't find the objects I refer to in this tutorial. You can find the HoloToolkit instructions at <https://github.com/Microsoft/HoloToolkit-Unity/blob/master/GettingStarted.md>.

Step 1: Import HoloToolkit to a New Unity Project

Before proceeding, be sure you've already downloaded and saved the HoloToolkit Unity package per the instructions in Chapter 1.

- Create a new Unity project (see Chapter 2 if you need a reminder on how to do this) and name it Holo World. *Important:* Save your scene and give it a name. If you don't save your scene, you won't be able to apply HoloLens settings in Step 2.
- From the menu bar, go to Assets ► Import Package ► Custom Package. In the pop-up window that appears, browse to the HoloToolkit that you downloaded in Chapter 1. See Figure 3-1 for an illustration of these menu items.

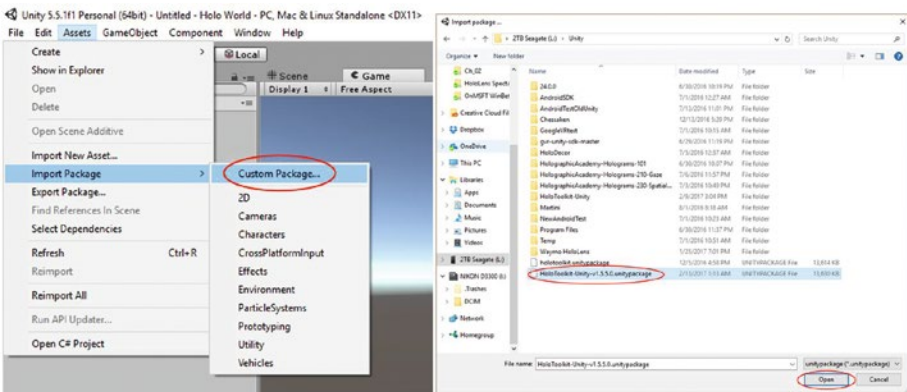


Figure 3-1. Import the HoloToolkit package you downloaded in Chapter 1

- Unity will take a minute to prepare the package you selected and then show you another pop-up window where you can select or deselect package items. Go ahead and leave everything checked (everything should be checked by default) and click the Import button, as shown in Figure 3-2.

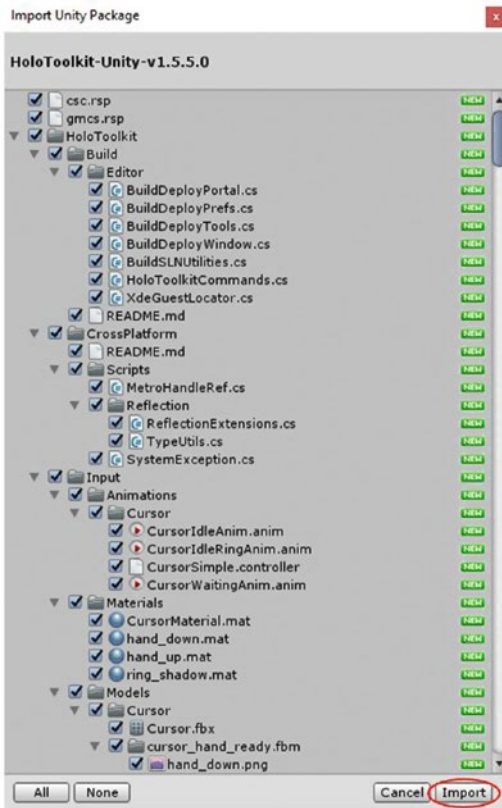


Figure 3-2. Click Import to import the HoloToolkit package

Step 2: Use HoloToolkit to Prepare Your Scene for Mixed Reality Development

After completing Step 1, you should now see a HoloToolkit menu item in your menu bar, as shown in Figure 3-3.

- From the menu bar, select HoloToolkit ► Configure ► Apply HoloLens Scene Settings. This will make the scene background black (when in the Game tab) and modify camera settings. Click the Apply button in the pop-up window that appears. *Important:* Save your scene.

- From the menu bar, select HoloToolkit ► Configure ► Apply HoloLens Project Settings. This will convert the Unity project to a Windows Direct 3D (D3D) project, optimize quality, and enable Virtual Reality support. Click the Apply button in the pop-up window that appears. Unity will require you to reload your project. *If you didn't save your scene from the previous step, you will lose all changes to the scene and need to apply scene settings again.*

■ **Tip** After choosing to apply settings from the HoloToolkit menu, a pop-up window appears showing settings to apply. Click each item to learn more about it.

- Remove the Main Camera and Directional Light game objects from the Hierarchy by right-clicking each item and choosing Delete from the context menu.
- To insert our new camera, go to the Project panel, and browse to HoloToolkit ► Input ► Prefabs. Drag and drop HoloLensCamera prefab into the Hierarchy, as shown in Figure 3-4.
- Save your scene.

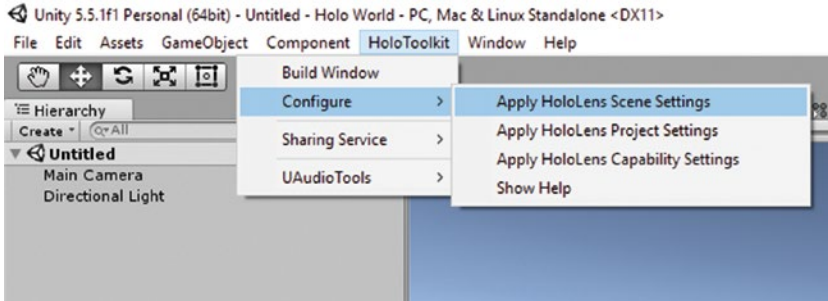


Figure 3-3. You now have a shiny new HoloToolkit menu item. Be sure to apply HoloLens scene settings and project settings

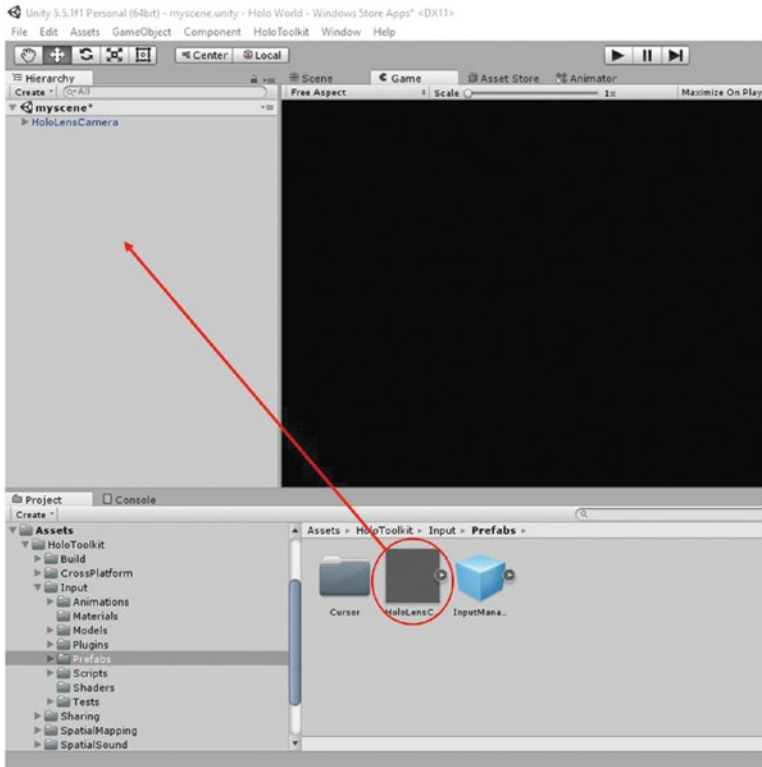


Figure 3-4. After deleting the Main Camera and Directional Light objects, insert the *HoloLensCamera* prefab into the Hierarchy

Congratulations! Your scene is now ready for holograms. Before starting each new Mixed Reality project in the future, you'll want to repeat these basic preparation instructions. And although you can keep using the same HoloToolkit package you downloaded, it's always a good idea to regularly check for updates to the HoloToolkit. New features are added all the time, and bugs are constantly being fixed. You can explore the HoloToolkit at <https://github.com/Microsoft/HoloToolkit-Unity>. Be sure to look around this page for reports of bugs, recent fixes, updates and more.

Your First Hologram

Your scene should now be completely black when in game view. You will still be able to see a floor grid and horizon when in the scene view. If you were to deploy this “app” to your HoloLens, you would see nothing. In this section, we'll create a simple cube object, which will serve as our first hologram.

Note In the context of Windows Mixed Reality, a *hologram* is any visible game object. To stay consistent with Microsoft's naming conventions, I generally call any visible game object or 3D model a hologram, but I may use these terms interchangeably throughout this book.

Step 1: Create a Cube

In Chapter 2, you created a plane and a sphere within Unity. Use the same approach to create a cube game object in your scene. In addition to the approach you learned in Chapter 2, you can also create a cube by right-clicking an empty place in the Hierarchy and in the pop-up context menu selecting 3D Object ► Cube, as shown in Figure 3-5.

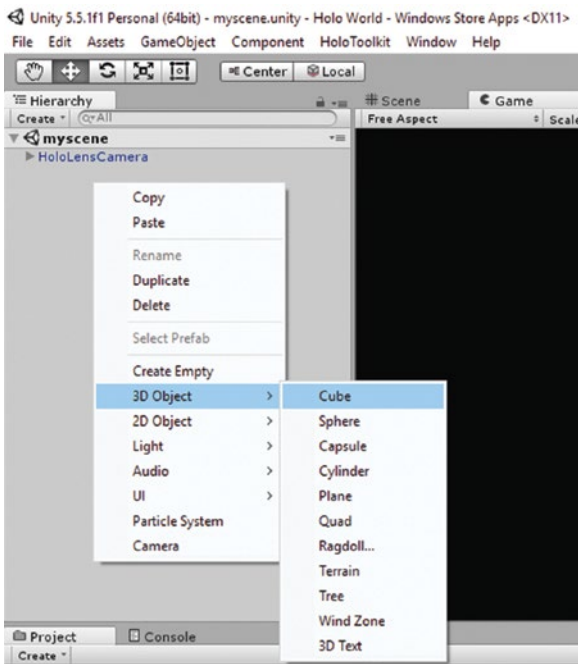


Figure 3-5. Create a cube in the Hierarchy

Step 2: Zoom to Your Cube

If you haven't already done so, be sure to switch to the scene view so that you can view objects in your scene. You can switch between views by clicking the tabs that are above the visualization window, as shown in Figure 3-6. Zoom to your cube by selecting the Cube game object in your Hierarchy and pressing the *F* key. Your scene should look similar to Figure 3-6.

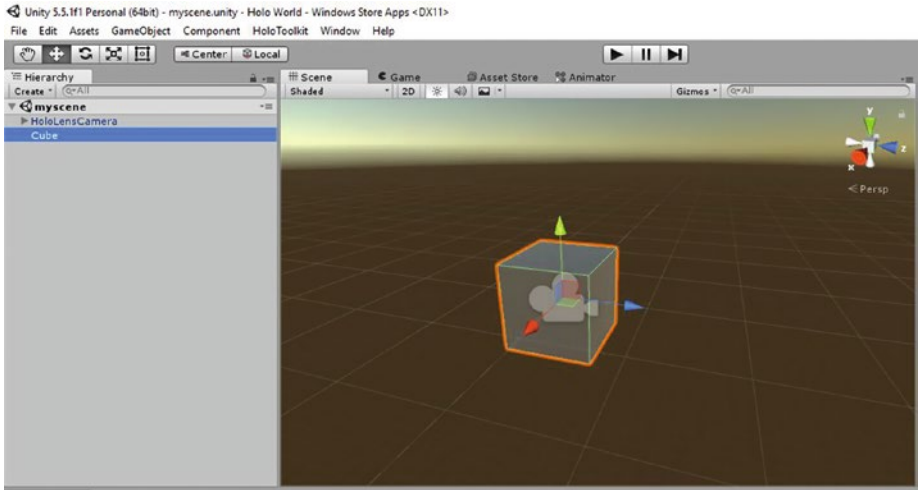


Figure 3-6. Zooming to your Cube

Step 3: Move the Cube Away from the Camera

Next, we'll want to move the Cube game object away from the camera. In Unity, *think of the camera as your eyes*. You view your app through the camera. You'll notice in the scene (refer to Figure 3-6) that both the cube and the camera are in the same place. This means that when you launch this application, you won't be able to see the cube because your eyes will be inside it! We need to move the cube a short distance in front of our face.

■ **Tip** When positioning objects in Unity, 1 unit represents approximately 1 meter (3.3 feet) in the real world.

Let's move the cube about 2 units in front of our face, which is about 2 meters in the real world:

- Select the Cube in the Hierarchy
- In the Inspector, change the Position to 2, as shown in Figure 3-7.

As you can see in Figure 3-7, the cube is now 2 units in front of the camera.

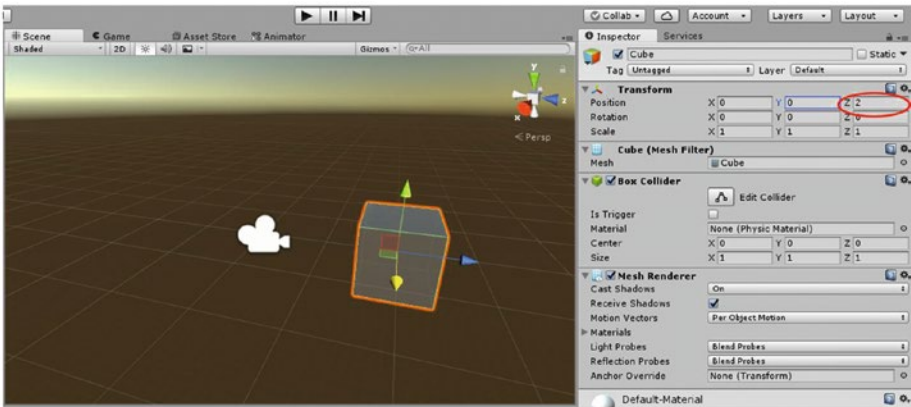


Figure 3-7. Moving the Cube 2 units in front of the camera

Step 4: Resize the Cube

Currently, our cube has a scale of 1 x 1 x 1, meaning it's approximately 1 meter on each side. Let's make it so we can view the entire cube within our field of view.

Use the approach in Step 7 of Chapter 2 to scale (resize) your cube down to 0.2 x 0.2 x 0.2, as shown in Figure 3-8.

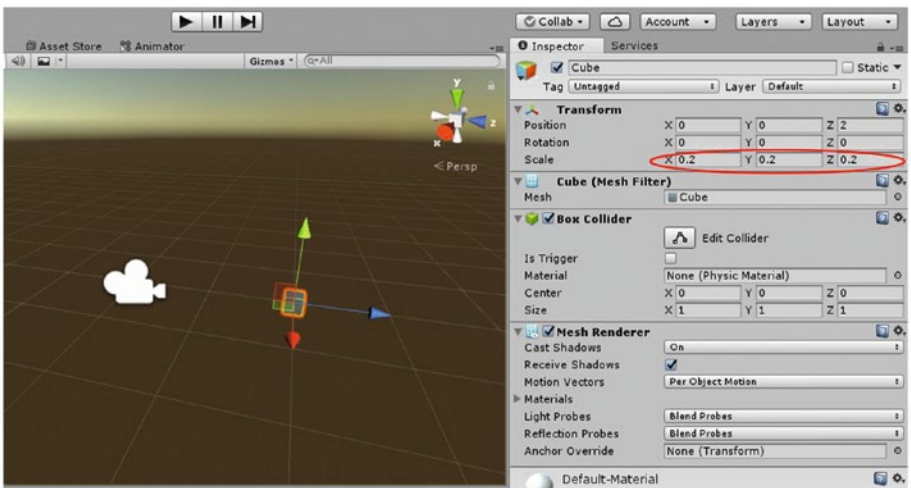


Figure 3-8. Scaling the cube down in size to make it easier to see

Step 5: Test Your App

It's important to regularly test your app as you develop it to ensure that it's behaving as you intend. Unity provides a very quick way of testing your app. To begin, simply click the play button located near the top of the Unity Editor, as shown in Figure 3-9. When you click Play, you should see your cube surrounded by a black background.

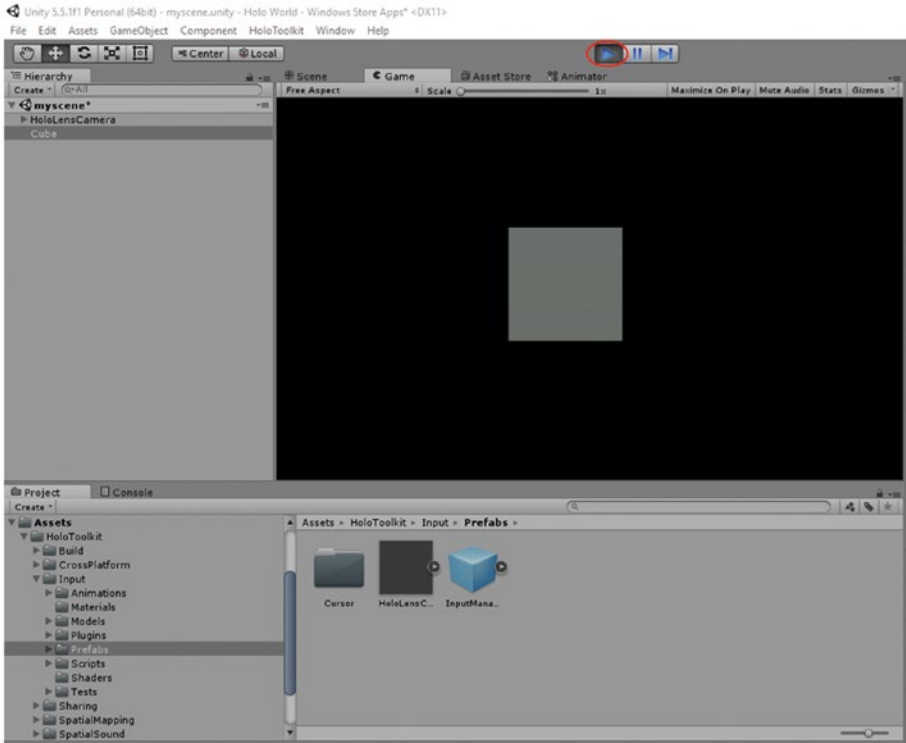


Figure 3-9. Clicking play to quickly test your app

The HoloLensCamera prefab that you imported into your scene near the beginning of this chapter also includes the ability to “move around” in your scene. Try it by pressing the left/right/up/down arrows on your keyboard while in Game mode (you may need to click the Game window before the keys work). You can also right-click your mouse to drag the camera's view. Be careful when pressing keys, as the movement can be very fast and you may lose sight of your cube. If that happens, just restart your app test.

Click the play button again to exit Game mode. This was a very quick and easy way to test your app. We didn't utilize holographic simulation or holographic remoting when we clicked play, so we weren't able to test all aspects of a full Mixed Reality experience, such as gestures, spatial mapping, object manipulations, and more. You'll learn about holographic remoting and simulation (collectively called *holographic emulation* in Unity) later in this chapter.

Step 6: Install Your App on the HoloLens

Now that we've tested our Cube app, let's install it on our HoloLens to experience our first hologram in person:

- Be sure that you've exited out of Game mode from the previous step. The play button should be black when you're not in Game mode; it's blue when you're in Game mode.
- In your menu bar, go to File ► Build Settings
- A pop-up window will appear, as shown in Figure 3-10.

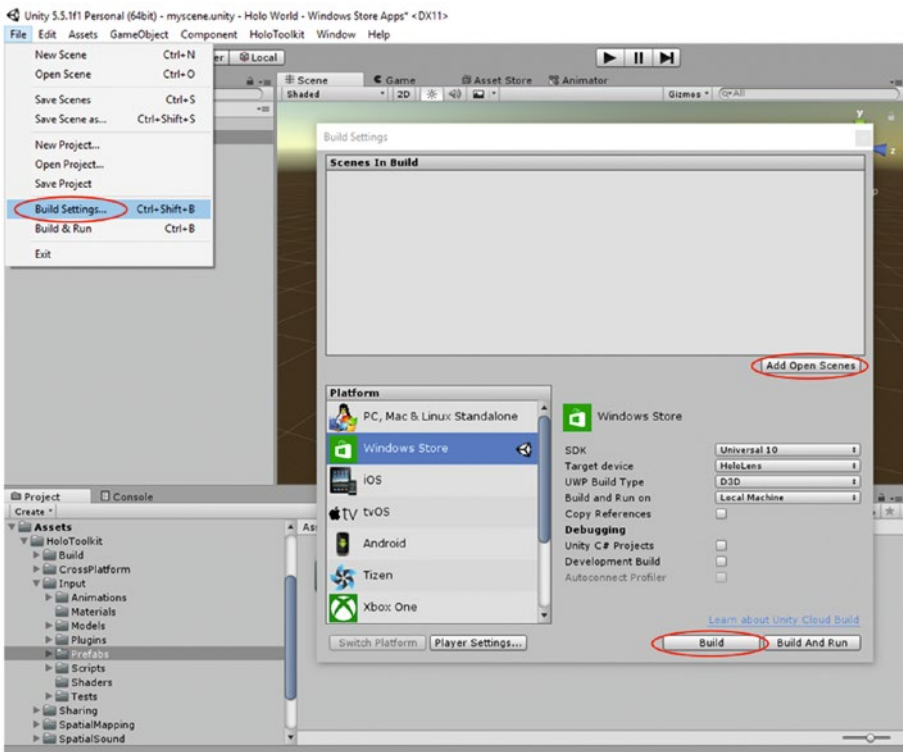


Figure 3-10. Export your first Mixed Reality app using the Build Settings window in Unity

- Be sure to click the Add Open Scenes button to add your current scene to the list of scenes to build.

- If you applied all the HoloLens projects settings at the beginning of this chapter correctly, then the remaining settings should not need to be modified. Review to make sure the Platform is set to Windows Store, the SDK is set to Windows 10, the Target device is set to HoloLens, the UWP Build Type is set to D3D, and that “Build and Run on” is set to Local Machine. See Figure 3-10 for how these settings should appear.
- Click the Build button.
- After you click Build, another pop-up window will appear. Create a new folder and name it. I typically name my folder App. Click your newly created folder and click the Select Folder button as shown in Figure 3-11.

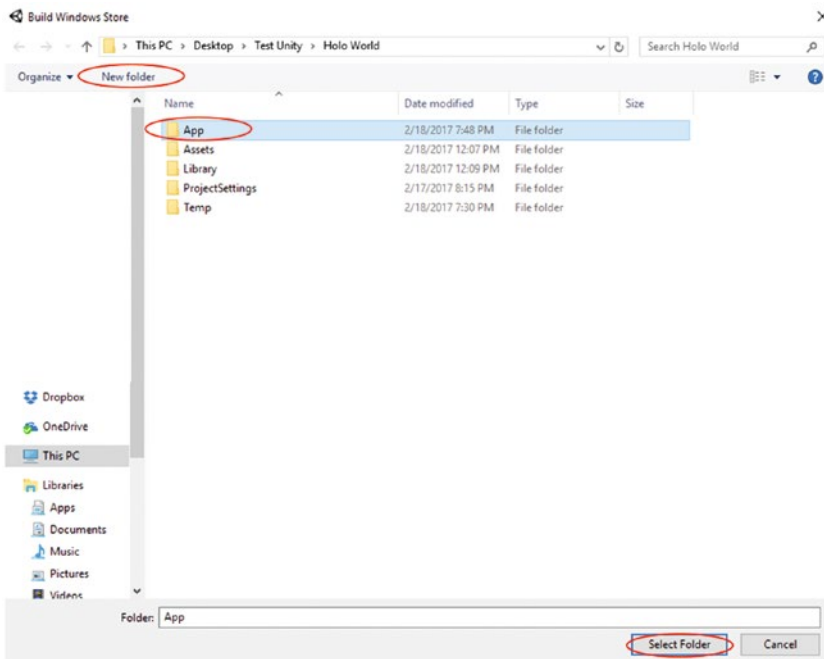


Figure 3-11. Create a new folder to contain your app

- Unity will spend a few seconds building your new app and placing project files in the new folder you created. After Unity has completed building your app, a pop-up window will appear, showing the new folder you created. Open this folder.
- Double-click Holo World.sln to open your project in Visual Studio. (Your file may be named differently if you didn't name your project Holo World).

- We will use Visual Studio to deploy (install) our app to our HoloLens. But first, we need to enable Developer mode on the HoloLens. Turn on your HoloLens and open the Settings app. Select the Update & Security menu item, as shown in Figure 3-12.

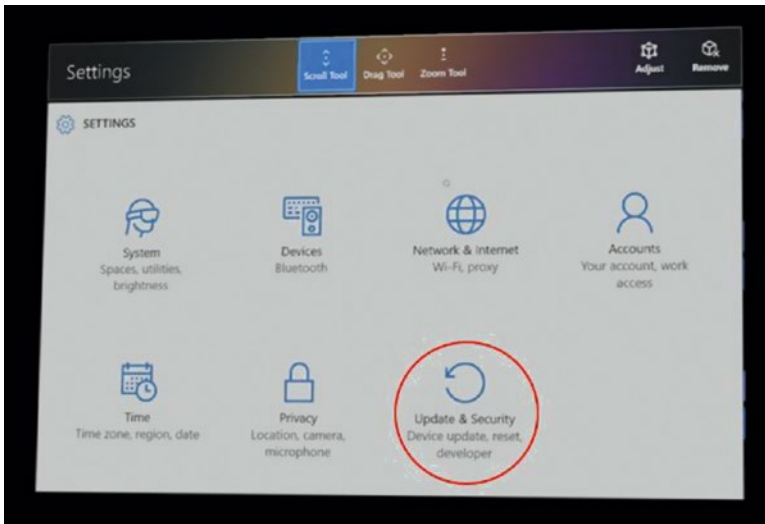


Figure 3-12. In the Settings app, go to the Update & Security menu item

- Once in the Update & Security menu, navigate to the “For developers” section and ensure that Developer mode is turned on, as shown in Figure 3-13.

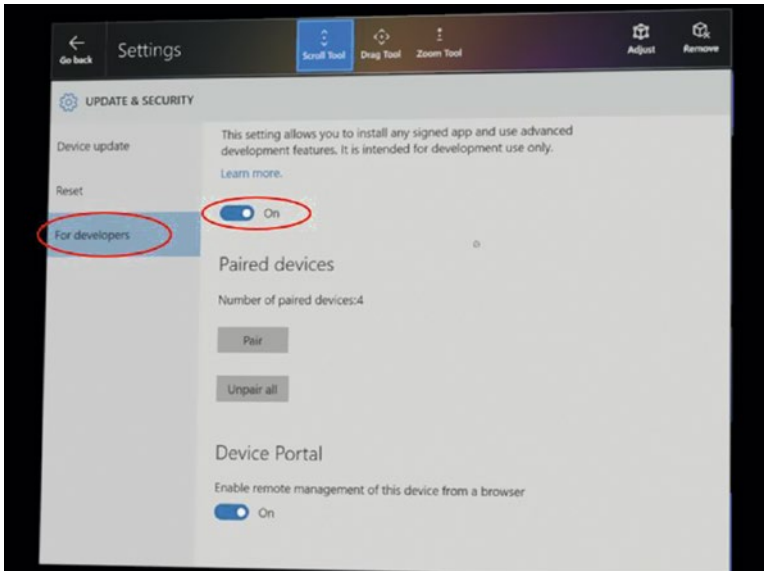


Figure 3-13. Navigate to the “For developers” section and enable Developer mode

- Make sure that your HoloLens is connected to the same WiFi network as your development PC.
- On your PC, set your configuration to Release and your platform to x86, as shown in Figure 3-14.

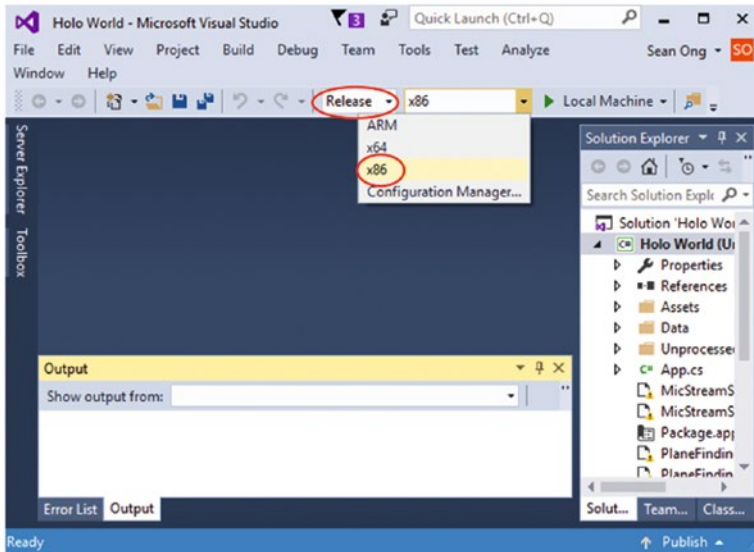


Figure 3-14. Set your configuration to Release and your platform to x86

- Set your target device to Remote Machine, as shown in Figure 3-15.

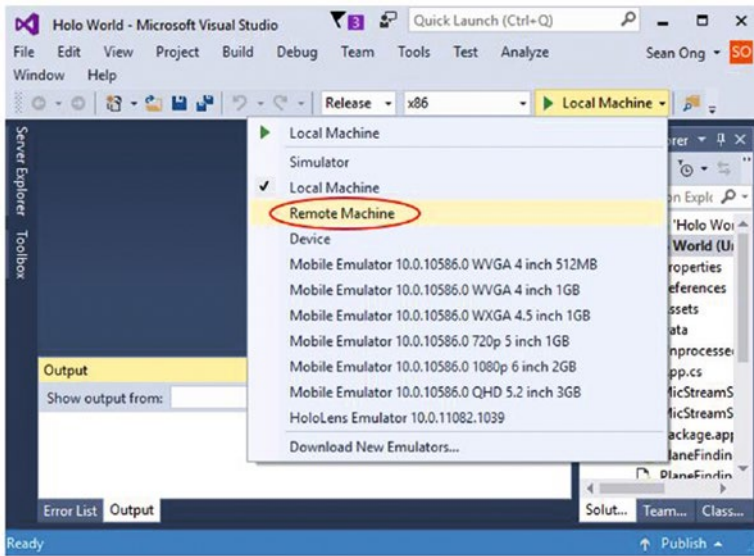


Figure 3-15. Set your target to Remote Machine

- A pop-up window will appear, where you can enter the IP address of your HoloLens. If you don't know your IP address, you can access it on your HoloLens by going to Settings ► Network & Internet ► Advanced Options. See Figure 3-16 for guidance on where to find the IP address in your HoloLens Settings app.

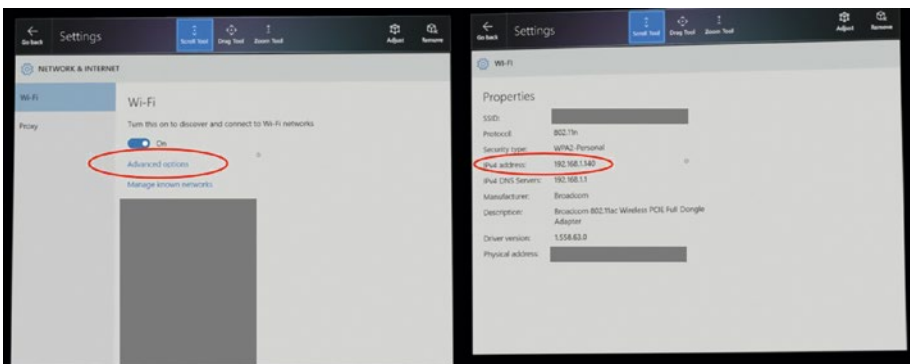


Figure 3-16. Where to find your HoloLens IP address in the Settings app

- Once you know your HoloLens IP address, enter it in the Remote Connections pop-up window in Visual Studio and click the Select button, as shown in Figure 3-17.

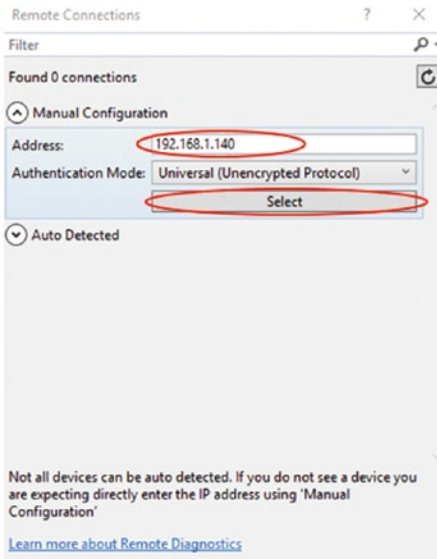


Figure 3-17. Enter your HoloLens IP address into Visual Studio's Remote Connections window

- You are now ready to deploy your app to your HoloLens. In the Debug menu, select Start Without Debugging, as shown in Figure 3-18.

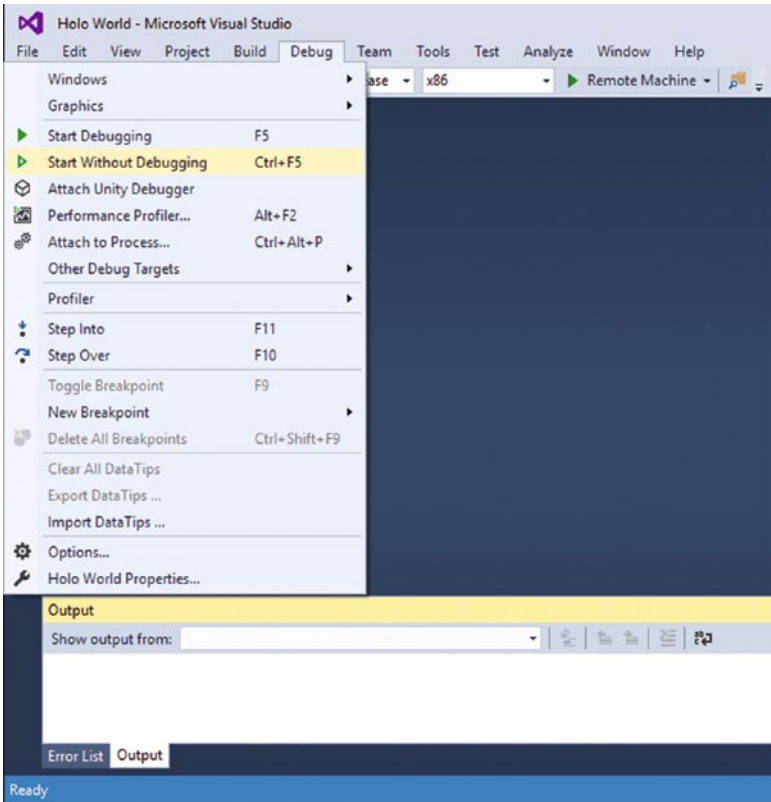


Figure 3-18. Start the app deployment process by going to **Debug** ➤ **Start Without Debugging**

- If this is your first time deploying to your HoloLens from this PC, you will be prompted to pair your HoloLens with Visual Studio, as shown in Figure 3-19. To pair your HoloLens, go back to the “For developers” section of your HoloLens Settings app and click the Pair button, as can be seen back in Figure 3-13. You will see numbers appear on your HoloLens (see Figure 3-20 for an illustration of this), which you can then type into the pop-up window in Visual Studio.

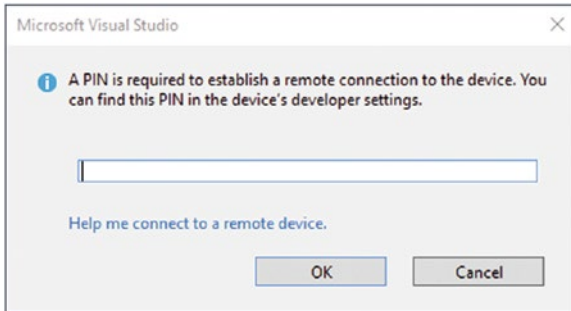


Figure 3-19. Visual Studio will prompt you for a PIN if this is the first time you're deploying to your HoloLens from this PC

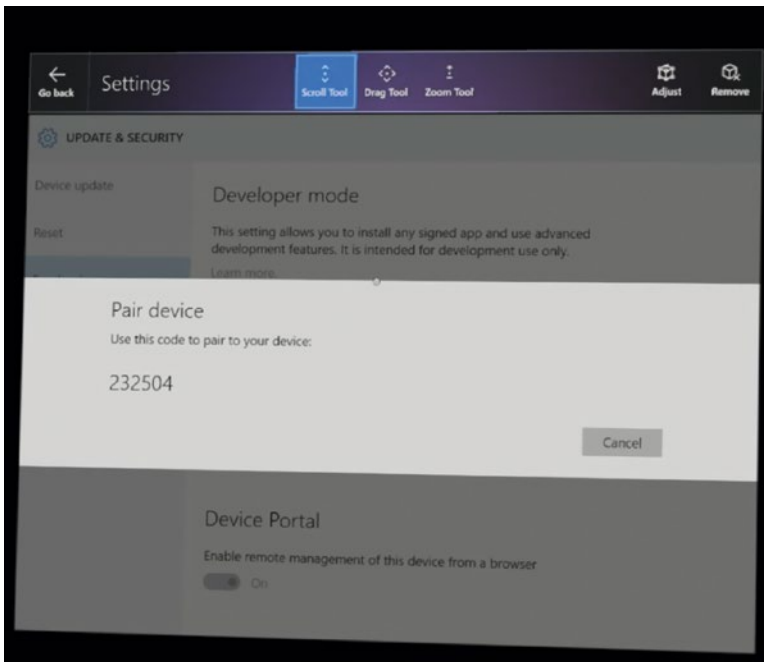


Figure 3-20. After clicking the Pair button, your HoloLens will display a PIN for you to enter into Visual Studio on your PC. Note: Of course, your PIN will be different than the one shown in this figure.

- After entering your HoloLens PIN into Visual Studio on your PC, you may dismiss the PIN pop-up on your HoloLens by clicking the Done button.

- Visual Studio will begin deploying your app to the HoloLens. You should see some output text in Visual Studio indicating that your app was successfully deployed to the HoloLens, similar to Figure 3-21. If you receive any error messages and a failed deployment, please check the output to see what the error messages are. Check to make sure you followed all the steps in this tutorial. Even if you followed all the steps correctly, there may be other reasons why a deployment may fail. For example, if you run out of disk space on your drive, or if Visual Studio wasn't installed correctly. If a mysterious error is preventing a successful deployment, I've found that restarting your computer and HoloLens often helps resolve the issue. Other potential solutions include rebuilding your App folder (deleting all contents and rebuilding from Unity) or re-entering your HoloLens IP address into Visual Studio.

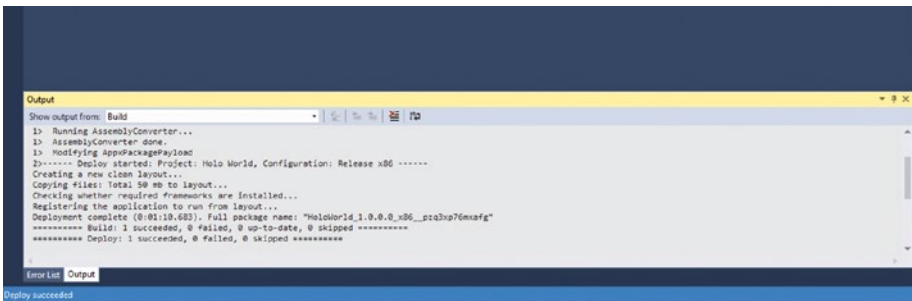


Figure 3-21. Example of text displayed by Visual Studio if deployment to the HoloLens is successful

- You should now be experiencing your very first Mixed Reality app. Congratulations! Feel free to walk around your hologram (cube) and look at it from various angles. What happens when you try to touch it? Is your hologram behind your computer monitor or wall? If so, restart your app while facing an open area.
- Your app is now installed on your HoloLens—which means you'll see it appear in your apps list as Holo World (if you named it Holo World, as I did in this tutorial).

Test Your App Using Holographic Remoting

This section discusses how to use Unity's holographic remoting feature to speed up development. In the previous step, we deployed our app to the HoloLens for testing. Although that's the most robust way to test your app, you may have noticed that it was time consuming and somewhat tedious. It's an inefficient way to test your app after each

change you make to your project. A much faster way to test your app on your HoloLens is to use Unity's holographic remoting. With holographic remoting, your app is streamed (via WiFi) from Unity to your HoloLens without needing to go through Visual Studio.

Step 1: Install and Run the Holographic Remoting Player to Your HoloLens

If you haven't done so already, you'll need to install the Holographic Remoting Player app on your HoloLens.

- On your HoloLens, open the Store app.
- In the Store's search bar, start typing in *remoting*. In the search results, you will see an app called Holographic Remoting Player, as shown in Figure 3-22.

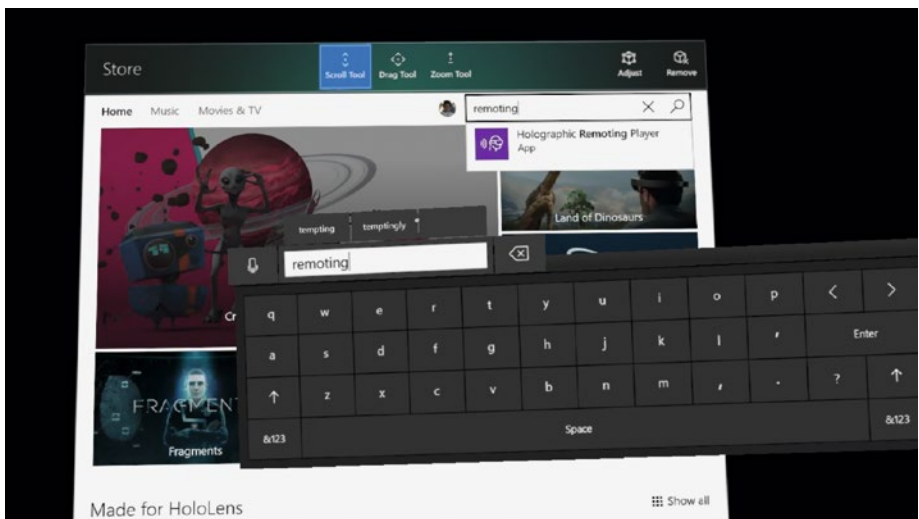


Figure 3-22. Search for the Holographic Remoting Player app and install it on your HoloLens

- After installing the Holographic Remoting Player app, launch the app. You will see a welcome screen showing your device's IP address, as shown in Figure 3-23.

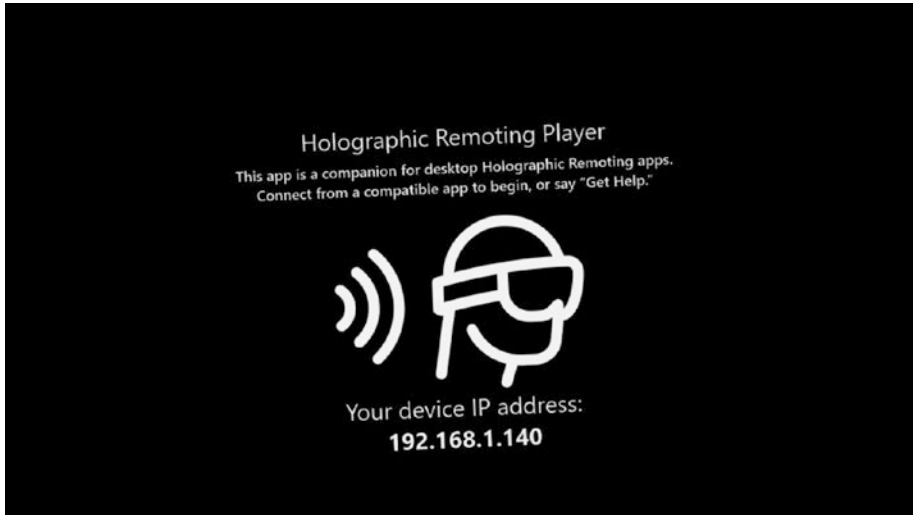


Figure 3-23. You'll see this screen upon launching the Holographic Remoting Player app on your HoloLens

Step 2: Connect to Your HoloLens with Unity's Holographic Remoting

- In Unity on your PC, go to Window ► Holographic Emulation, as shown in Figure 3-24.

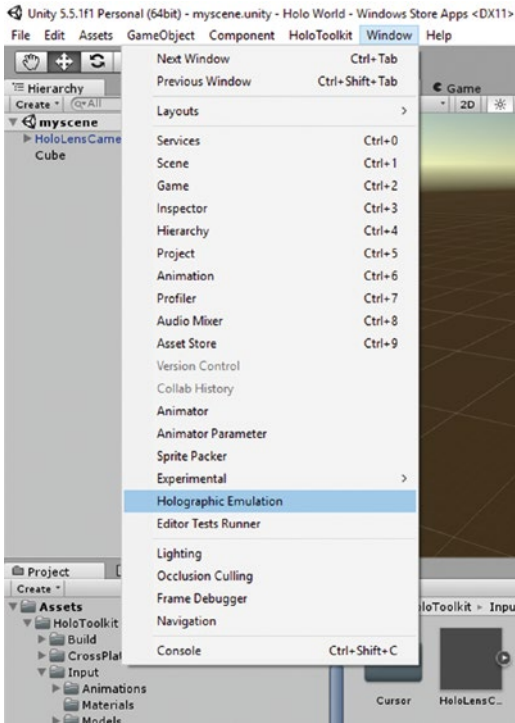


Figure 3-24. In Unity, open the Holographic Emulation window

- A pop-up window will appear where you can select your Holographic Emulation settings. For Emulation Mode, select Remote to Device. For the Remote Machine IP Address, enter the IP address that you see displayed on your HoloLens (be sure the Holographic Remoting App player is running). See Figure 3-25 for an example of the Holographic Emulation window in Unity. After entering your IP address, you may need to uncheck and check one of the checkboxes for the changes to take effect, due to a bug. Try that workaround, if you're unable to connect to your HoloLens.

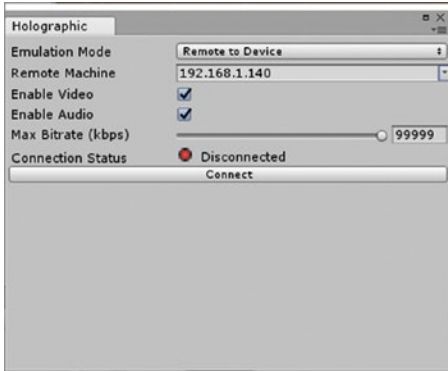


Figure 3-25. Example of Unity's Holographic Emulation window

- Click the Connect button to connect to your HoloLens. If the connection was successful, the Connection Status in Unity's Holographic Emulation window should indicate that it's connected. You should see a blank screen (nothing/transparent) in your HoloLens. If you have music or audio playing on your PC, you may hear it on your HoloLens speakers.

Step 3: Test Your App Using Holographic Remoting

To test your application, click the play button in the Unity editor (refer back to Figure 3-9, from a previous example). Your hologram will instantly appear in your HoloLens, as if it were installed on your HoloLens.

Tip For the best holographic remoting experiences, be sure to have a strong WiFi connection. A poor connection will cause lagging and/or pixilation issues. Using a PC with a powerful graphics card will also help with performance when streaming complex scenes to your HoloLens.

As you can see, holographic remoting is a very fast and efficient way to test your app on your HoloLens. Later, we'll see that holographic remoting also supports gesture recognition from your HoloLens, as well as voice commands.

Test Your App Using Holographic Simulation

In this section, we'll walk through how to use holographic simulation within Unity when testing your apps. So far, you've already learned how to deploy apps to your HoloLens via Visual Studio. You also learned a much faster approach for streaming apps to your HoloLens with holographic remoting. Another approach to quickly test your app

(when using your HoloLens device is not required) is called *holographic simulation*. As discussed in Chapter 1, holographic simulation allows you to use a controller, such as the Xbox One controller to walk around and control your app from within Unity.

You may be wondering how this is different from simply clicking the play button in Unity without holographic simulation enabled. Here are some key advantages of using holographic simulation over testing your app without emulation within the Unity editor:

- You can select between several rooms/areas in order to simulate spatial mapping.
- Holographic simulation supports voice and gesture simulation.
- You can easily use a wireless controller to walk around and control your app.

Let's walk through how to set up holographic simulation to test your app.

Step 1: Enable Holographic Simulation

- Open the holographic emulation window by clicking Window ► Holographic Emulation.
- In the Holographic Emulation pop-up window that appears, select *Simulate in Editor* for the Emulation Mode, as shown in Figure 3-26. You don't need to modify the Room or Gesture Hand for the Cube app.

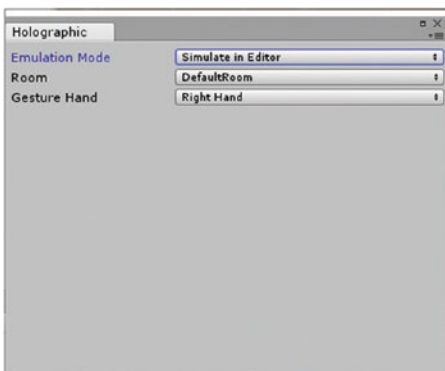


Figure 3-26. To enable holographic simulation, set Emulation Mode to *Simulate in Editor* within the Holographic Emulation window

Step 2: Connect Your Controller

In order to walk around, look around, and control your app, you'll need to connect a gamepad or game controller, such as the Xbox One controller. Please see instructions specific to your controller to connect it to your PC. Unity will automatically recognize your controller once connected. No setup is required in Unity for your controller.

Step 3: Test Your App Using Holographic Simulation

Click the play button in the Unity editor to begin testing your app using holographic simulation. If your app isn't responding to your controller, be sure to check that your game window has focus by clicking it. If you accidentally clicked another part of the Unity editor, or clicked out of Unity, the app won't respond to your controller.

Holographic emulation loads a virtual model of a room that your app can interact with. Figure 3-27 shows what the spatial mapping mesh looks like when an application has the ability to make the spatial map visible. Note that you won't be able to see the spatial map in the application we developed in this chapter. I cover spatial mapping extensively in Chapter 6.

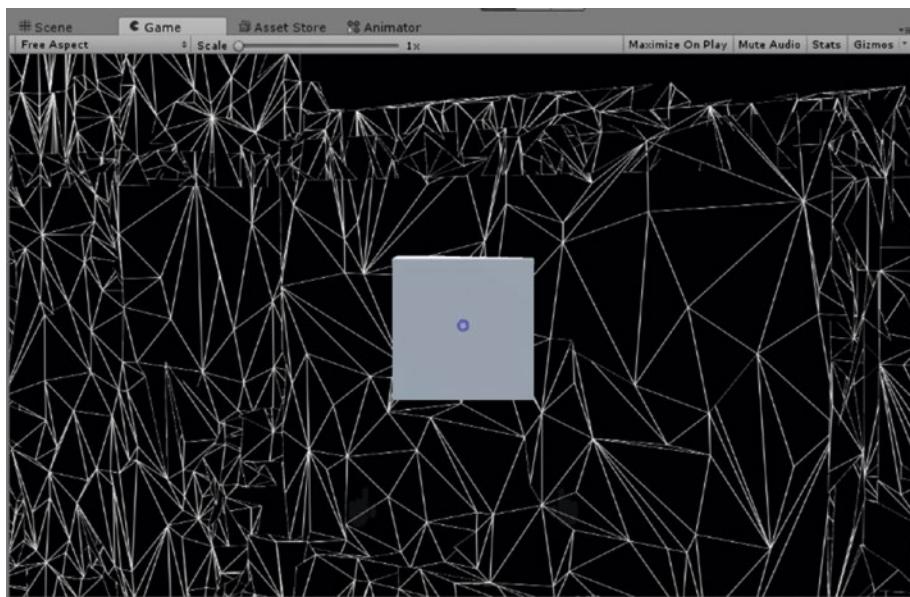


Figure 3-27. Holographic simulation loads a virtual room where you are able to test spatial mapping features without leaving Unity

Summary

Congratulations! Having created your very first hologram, you're well on your way to becoming a Mixed Reality developer. Creating and seeing your first hologram is a very satisfying experience. Let's recap what was covered in this chapter:

- You learned how to prepare Unity for holographic development using the HoloToolkit.
- You learned how to place a hologram into a scene.

- You learned how to install your app on a HoloLens by deploying it using Visual Studio.
- You learned how to stream your app to your HoloLens using Unity's holographic remoting.
- You learned how to test your app using Unity's holographic simulation.

The tutorials in this chapter serve as the building blocks for all Mixed Reality development workflows.

CHAPTER 4



Introduction to the HoloToolkit

This chapter covers more about the HoloToolkit and its importance for Mixed Reality development. We'll learn about the various components and test scenes included with the HoloToolkit, and how we can leverage this community resource for development.

What Is the HoloToolkit?

From Microsoft's HoloLens documentation, you might think that the HoloToolkit is an optional toolkit for enhancing your development experience. In fact, the HoloToolkit is an essential part of HoloLens development. The HoloToolkit provides developers with all the tools needed to get started developing Mixed Reality applications. It does everything from preparing Unity's settings to enabling gestures and spatial mapping. The HoloToolkit also contains many useful example scenes for developers to explore and understand how to use various parts of the HoloToolkit.

The HoloToolkit is a community resource that is overseen by Microsoft and other trusted individuals/groups. Anyone, including you, can contribute content to the HoloToolkit (it will first need to be vetted before being incorporated). As such, the HoloToolkit is constantly being updated and improved. In fact, it has already vastly improved between the time I starting writing this book and the moment I started writing this chapter. After learning of some of the useful HoloToolkit features in this chapter, I recommend exploring the HoloToolkit repository online to learn about any additional changes. Toward the end of this chapter, I walk you through navigating the online HoloToolkit repository.

HoloToolkit Setup

This section walks you through downloading and installing the HoloToolkit. To download the HoloToolkit Unity package, go to <https://github.com/Microsoft/HoloToolkit-Unity/releases>.

Make sure you download the latest release of the HoloToolkit, typically located near the top of the page. Be sure that the HoloToolkit version you download is compatible with the version of Unity you downloaded. Typically, compatibility is announced in the title of the HoloToolkit version, as shown in Figure 4-1. To download the HoloToolkit Unity package, click the download link with the file extension .unitypackage. For example, in Figure 4-1, the appropriate download link (circled) is named HoloToolkit-Unity-v1.5.5.0.unitypackage.

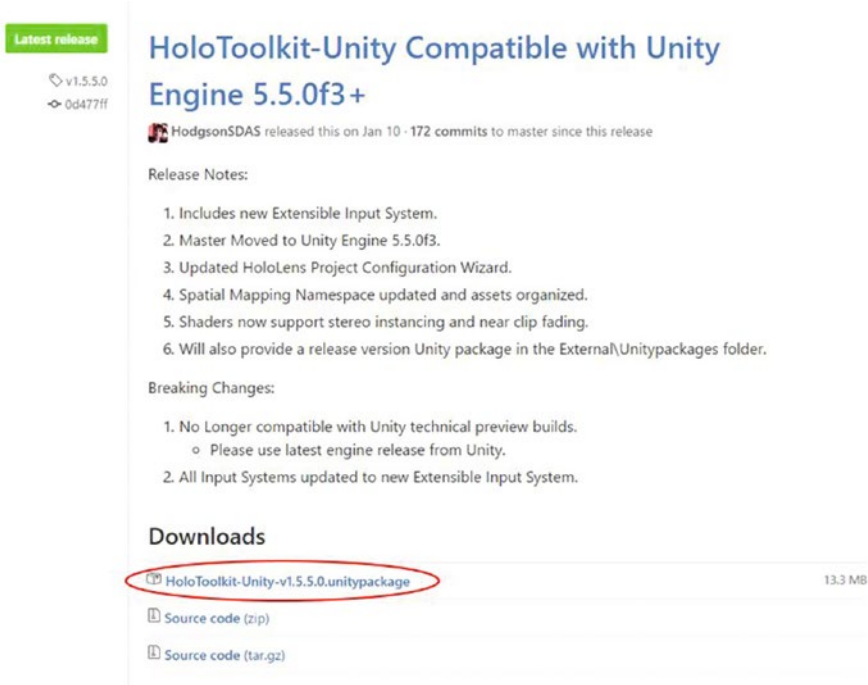


Figure 4-1. Browse to the HoloToolkit download page and download the HoloToolkit Unity Package, circled in this figure

Save the HoloToolkit to your PC. In your Unity project's menu bar, go to Assets ► Import Package ► Custom Package. In the pop-up window that appears, browse to the HoloToolkit you just downloaded. Figure 4-2 shows these menu items.

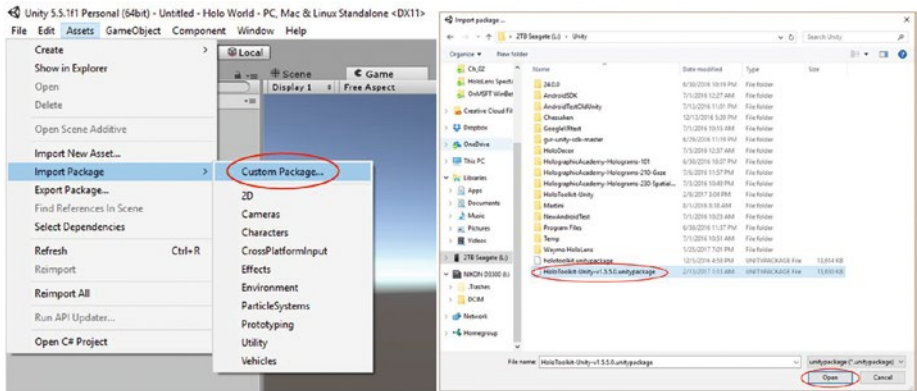


Figure 4-2. Import the HoloToolkit package you downloaded in Chapter 1

Unity will take a minute to prepare the package you selected and then show you another pop-up window where you can select or de-select package items. Go ahead and leave everything checked (everything should be checked by default) and click the Import button, as shown in Figure 4-3.

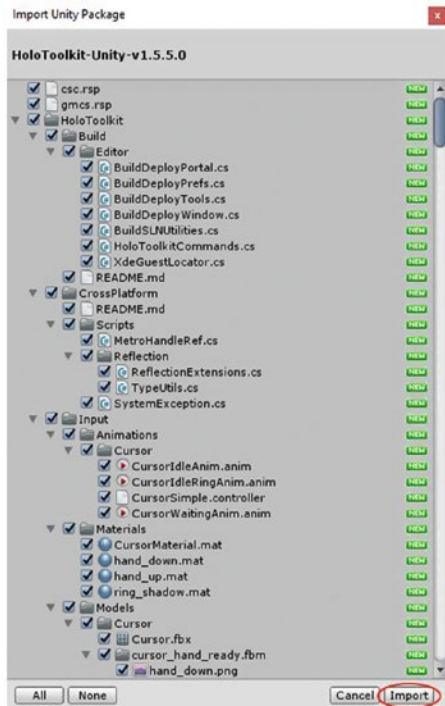


Figure 4-3. Click the Import button to import the HoloToolkit package

After completing these steps, the HoloToolkit will now be installed to your project.

■ **Note** You'll need to import the HoloToolkit every time you start a new Unity project.

HoloToolkit Components

We're now at the exciting part of the chapter where I walk you through the various components of the HoloToolkit. The HoloToolkit includes seven feature areas. Table 4-1 lists each feature area and provides a brief description.

Table 4-1. HoloToolkit Features and Descriptions

Feature Area	Description
Input	Allows developers to include input in their apps, such as gestures, clickers, gaze, and voice commands.
Sharing	Allows developers to make shared experiences. This allows many users to see and experience the same app together.
Spatial Mapping	Tools to enable and use spatial mapping. This allows your application to interact with the physical environment, such as walls, floors, and objects nearby.
Spatial Understanding	Allows your application to understand the physical environment. For example, it can differentiate between chairs, tables, and other common structures.
Spatial Sound	Allows developers to include spatial sound capabilities, so that objects sound as if they are physically in your environment.
Utilities	A collection of useful utilities, such as frame rate viewers, object locators, ability for objects to follow you, menu items for configuring Unity, and more.
Build	Time-saving features that enable the ability to quickly build and deploy projects directly from Unity without having to first go through Visual Studio.

The following subsections discuss highlights of each of the seven feature categories. Upcoming chapters cover some of these features in greater detail—in fact, some features have entire chapters devoted to them.

HoloToolkit: Input

The input features in HoloToolkit provide developers with the ability to interact with holograms. It includes a collection of scripts that allow your app to recognize gestures (such as the air-tap gesture and the hold gesture), inputs from devices such as clickers, gaze capabilities, and voice commands. Chapter 5 covers these input methods in much greater detail. Under the input features, you'll also find cursors that you can use for your Mixed Reality projects.

There are a wide range of test scenes where you can test various features included in the input feature set. Test scenes are an excellent way to explore HoloToolkit features and gain inspiration for your own projects. I often use test scenes as templates for my own projects. There are dozens of test scenes across all HoloToolkit features. Let's walk through how to explore one of these test scenes.

Tip Test scenes are a great way to see HoloToolkit items in action and learn how to implement them. You can also use a test scene as a template for your next project.

Running a Test Scene

Test scenes are typically located in a project folder called Scenes. Within the input feature area, you can find the scenes folder by going to Assets ► HoloToolkit ► Input ► Tests ► Scenes, as shown in Figure 4-4. Each of the seven feature groups may have a slightly different folder organization. Folder organization within the HoloToolkit is evolving over time, so be sure to explore the project folders or check the latest HoloToolkit documentation if you can't find the test scenes in your version of the HoloToolkit.

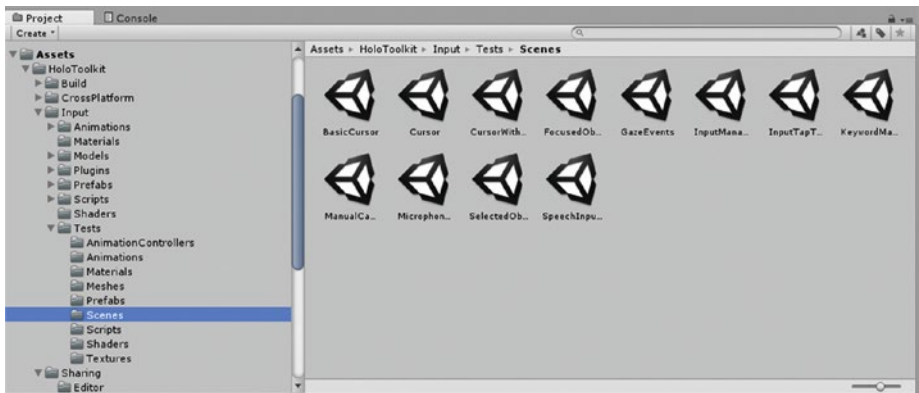


Figure 4-4. Navigate to the Scenes folder in each HoloToolkit feature to try out various test scenes

As shown in Figure 4-4, asset names in your project panel may be shortened (a partial name will be displayed, followed by ...). To see the full name, you may adjust the icon view by adjusting the slider, as shown at the bottom right corner of Figure 4-4.

Scene names are typically self-explanatory, but sometimes you may want to know additional details before trying a scene. The best place to learn more about each test scene is in the HoloToolkit documentation on GitHub. For example, to learn more about the scene called InputTapTest, you can start by going to the HoloToolkit-Unity page at <https://github.com/Microsoft/HoloToolkit-Unity>.

Scroll down to the README.md section as shown in Figure 4-5 and click Input, since our scene is located in the inputs section. You'll see a long page with detailed documentation on all the resources provided within the Input feature set.

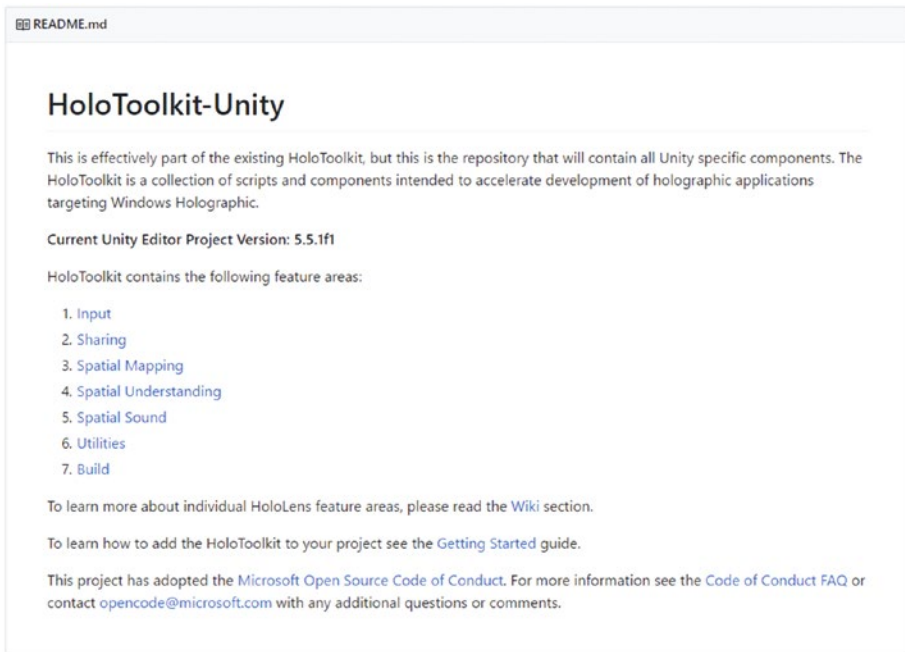


Figure 4-5. The HoloToolkit-Unity page on GitHub contains useful documentation on all features

Scroll down to the Tests section, where you can find the name of the test scene you're interested in and read more about it. As shown in Figure 4-6, the InputTapTest.unity scene shows how to respond to user's gaze using the Input module and how to respond to the user's tap gesture.

Tests

Tests related to the input features. To use the scene:

1. Navigate to the Tests folder.
2. Double click on the test scene you wish to explore.
3. Either click "Play" in the unity editor or File -> Build Settings.
4. Add Open Scenes, Platform -> Windows Store, SDK -> Universal 10, Build Type -> D3D, Check 'Unity C# Projects'.
5. Click 'Build' and create an App folder. When compile is done, open the solution and deploy to device.

BasicCursor.unity

Shows the basic cursor following the user's gaze and hugging the test sphere in the scene.

Cursor.unity

Shows the cursor on holograms hugging the test sphere in the scene and cursor off holograms when not gazing at the sphere.

CursorWithFeedback.unity

Shows the cursor hugging the test sphere in the scene and displays hand detected asset when hand is detected in ready state.

FocusedObjectKeywords.unity

Example on how to send keyword messages to currently focused dynamically instantiated object. Gazing on an object and saying "Make Smaller" and "Make Bigger" will adjust object size.

InputTapTest.unity

Test scene shows you in a simple way, how to respond to user's gaze using the Input module. It also shows you how to respond to the user's tap gesture.

Figure 4-6. *The HoloToolkit provides a brief description for all test scenes*

In the Unity Editor, choose the scene you're interested in trying (in our case, the InputTapTest scene) and drag it from the Project panel to an empty area in your Hierarchy, as shown in Figure 4-7.

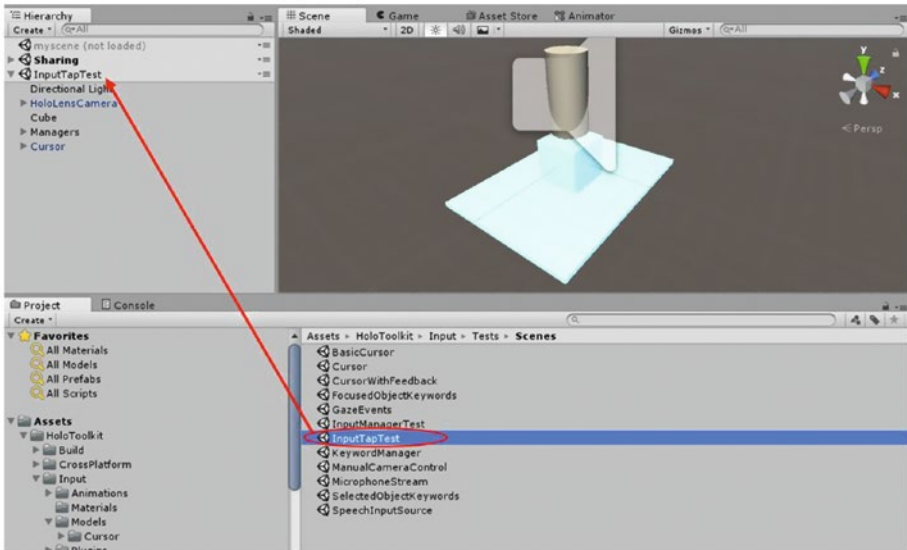


Figure 4-7. To explore a test scene, drag it into your Hierarchy

To avoid conflicts, disable any other open scenes by right-clicking the other scenes and selecting **Unload Scene** from the context menu. Unloading a scene will temporarily disable it, making it quick and easy to toggle scenes. You may also choose to **Remove Scene** if you no longer want to work with a scene. You can still reimport the scene from your Project panel if you didn't intend to remove the scene from your Unity project.

Now that you have your test scene loaded, feel free to try it out by clicking the play button. You may also deploy it to your HoloLens or stream it to your device using holographic emulation. This is a great opportunity to explore the code being used and see how the test project works. Many test scenes are included with the HoloToolkit, and I recommend you try as many of them as possible.

HoloToolkit: Sharing

One of the more powerful features included with the HoloToolkit is the sharing module. The sharing module allows multiple people to share the same Mixed Reality experience, locally or remotely. For example, several people wearing Mixed Reality headsets in the same room will be able to see and interact with the same holograms together. For remote users, the sharing module allows users to see avatars of each other, hear each other, interact, and collaborate. Figure 4-8 is an example of what such a share collaboration might look like.



Figure 4-8. *HoloToolkit's sharing module allows Mixed Reality experiences to be shared both locally and remotely (source: Microsoft)*

Several test scenes are included in this module to help you familiarize yourself with setting up a shared holographic experience.

HoloToolkit: Spatial Mapping

The HoloToolkit's spatial mapping module provides you with the resources you need to include spatial mapping capabilities into your project. Spatial mapping uses the sensors on a Mixed Reality headset to create a virtual map of the physical surroundings. The HoloToolkit provides resources to use this map or mesh to hide or occlude objects behind the mesh, interact with the mesh, and visualize the mesh. Chapter 6 covers spatial mapping in depth.

HoloToolkit: Spatial Understanding

Spatial understanding is a remarkable capability included with the HoloToolkit that enables our Mixed Reality experiences to “understand” the spatial environment. Based on the precise measurements taken with spatial mapping, this module interprets the spatial mesh and guesses which parts of the mesh are walls, tables, chairs, and more. Mixed Reality applications can use this feature in many ways. For example, you might have a holographic character or avatar sit on a chair in your room during a game. To achieve this, you'll need the spatial understanding module to find the part of a room that is a sitting surface, as shown in Figure 4-9. Chapter 6 covers spatial understanding.

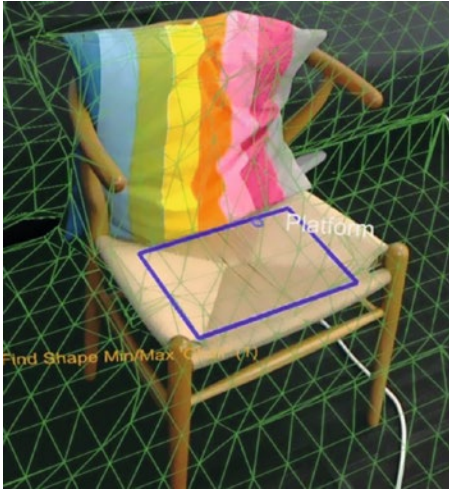


Figure 4-9. The spatial understanding module being used to find the sitting surface of a chair

HoloToolkit: Spatial Sound

The HoloToolkit's spatial sound module provides you with the resources you need to include spatial sound capabilities into your project. We rely heavily on our ears to precisely locate objects around us. In the context of Mixed Reality, this is called *spatial sound*. Spatial sound is essential for increasing the feeling of immersion and realism.

The HoloToolkit's spatial sound module includes capabilities such as *audio occlusion* (the ability to accurately reduce an object's audio when it's behind another hologram), the ability to tune the audio based on the size and shape of the physical room, 3D positioning of audio, and other advanced audio settings. Chapter 7 covers the details and implementation of spatial sound.

HoloToolkit: Utilities

The HoloToolkit's utilities module provides several useful utilities that can be used in your Mixed Reality applications. I recommend exploring this module in the HoloToolkit and reading the online documentation for the most up-to-date listing of utilities, because utilities are added to this module regularly. The following is a description of some of the most common and useful utilities included in the HoloToolkit. This is far from an exhaustive list, but it will give you a taste for the type of utilities included in this module:

- *FPSDisplay.prefab*: This is a “billboard” or floating screen that follows you in your application and displays your app's frames per second (FPS). When your device's performance struggles due to a complex scene or heavy processing requirements, the first noticeable effect is a reduction in FPS. Having a FPS billboard helps you keep an eye on your app's FPS so you can optimize it during development.

- *HeadsUpDirectionIndivator.prefab*: Sometimes you want to prompt users about where to look to find relevant holograms in your applications. This useful utility lets you include an arrow in your app that points the user to look in the right directions.
- *Toolbar configuration items*: Remember the HoloToolkit toolbar items you used to get Unity ready for Mixed Reality development in Chapter 2? That was a feature of the HoloToolkit's utilities module. This useful utility makes it easy for developers to quickly apply important settings to the project, camera, and scene.
- *Billboard.cs*: Apply this useful script to billboards in your app so that the billboards are always facing the user, even if the user moves.
- *Tagalong.cs*: Do you want a hologram to follow you in your application? Simply apply this tagalong script to the object you want to have follow you.

HoloToolkit: Build

The HoloToolkit's build module saves you time by allowing you to build, save, and deploy your app directly from Unity without needing to open Visual Studio. After importing the HoloToolkit, you'll gain access to the build window, located under the HoloToolkit menu item in the menu bar. As shown in Figure 4-10, the build window provides you with multiple options to build, save, or deploy your app.

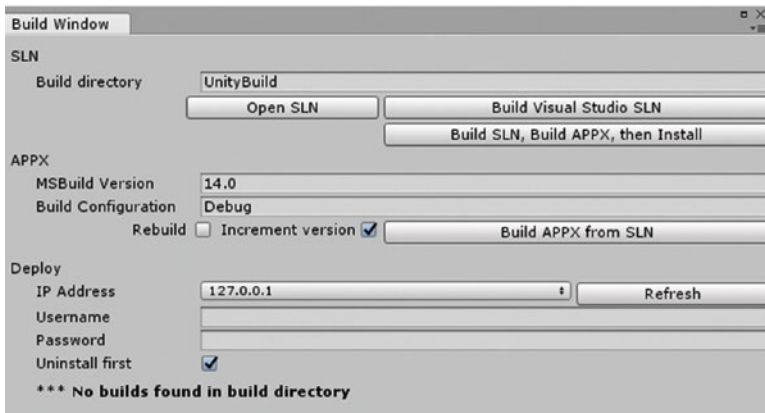


Figure 4-10. The build window included in the HoloToolkit lets developers quickly build and deploy apps to devices without first needing to go through Visual Studio

HoloToolkit Online

As mentioned previously, the HoloToolkit is constantly being updated and improved by its community of developers. In this section, you'll learn about the online HoloToolkit repository so that you can keep updated on the latest updates, issues, and improvements.

The Two HoloToolkit Repositories

There are actually two HoloToolkit repositories online. The first is called HoloToolkit, and the second is HoloToolkit-Unity. You can find them at the following links:

- *HoloToolkit*: <https://github.com/Microsoft/HoloToolkit>
- *HoloToolkit-Unity*: <https://github.com/Microsoft/HoloToolkit-Unity>

The “regular” HoloToolkit is a generalized version of the HoloToolkit that contains the core C++ code base that many of the Unity toolkit features are built on top of or are merely wrappers around. This version of the HoloToolkit is often used by developers that use platforms other than Unity for development.

The HoloToolkit-Unity repository contains Unity-specific components and will be what we focus on throughout this book.

What Is GitHub?

GitHub is a commonly used website among developers for storing and sharing software project files. It allows for careful monitoring and approval/rejection of changes to project files, making it an ideal platform when many developers are using and modifying a project at the same time.

HoloToolkit Help and Documentation

As of this writing, finding all documentation for HoloToolkit components is admittedly challenging and somewhat fragmented. I've included a few links here to help you quickly access documentation for the HoloToolkit:

- <https://github.com/Microsoft/HoloToolkit-Unity/blob/master/README.md>: The “readme” section of HoloToolkit-Unity contains detailed documentation on each of the seven features/modules included in the HoloToolkit.

- <https://github.com/Microsoft/HoloToolkit-Unity/wiki>: The wiki of HoloToolkit-Unity contains some additional context and background for several features. I recommend reading the material here first, before reading the detailed documentation from the readme sections. The HoloToolkit wiki includes some links in the home page and some additional links in the Pages bar on the right of the wiki's web page. Don't forget to check out <https://github.com/Microsoft/HoloToolkit-Unity/wiki/HoloToolkit-Menu> to learn more about the HoloToolkit menu.
- <https://github.com/Microsoft/HoloToolkit-Unity/blob/master/GettingStarted.md>: The "getting started" instructions for the HoloToolkit include some basic information to help you set up a new project.
- <https://github.com/Microsoft/HoloToolkit-Unity/issues>: The issues section of the HoloToolkit is important for understanding any outstanding issues that you may experience while using the HoloToolkit. If you discover any new issues, this is also where you can report them.

Summary

This chapter familiarized you with the seven HoloToolkit features or modules. You learned what the HoloToolkit is and got a sense of its importance when developing Mixed Reality experiences. You saw how to download and install the HoloToolkit, how to try test scenes that are included with the HoloToolkit, and how to navigate the online repository.

The HoloToolkit is an active community resource that's constantly changing and evolving. New features are added almost daily, and old features are depreciated (deleted or made obsolete). As such, I recommend you explore the HoloToolkit and discover any new and amazing features that may have been added. And as you continue your Mixed Reality journey, you will likely contribute new and exciting content to the HoloToolkit that will benefit other users too.



Interacting with Holograms

In this chapter, you'll learn about input mechanisms used with Mixed Reality development. There are several ways in which users can interact with holograms and other elements within an application. These include hand gestures, voice commands, gaze, and controllers. We'll walk through each input method and learn how to use input resources found in the HoloToolkit.

Input Methods

Each input method used with Windows Mixed Reality has its benefits and limitations. The following list provides a description for each input category:

- *Gaze*: The use of Gaze in Windows Mixed Reality is the primary method by which the user focuses on holograms and objects. In fact, *gaze is as essential to Mixed Reality as the mouse is to the PC*. You use the mouse to point at objects on your PC screen. In the same way, you use your gaze (the direction you look) to point to objects in 3D space. On a PC screen, the location of your mouse is represented by a cursor or arrow. The gaze cursor is typically represented by a small dot or donut-shaped object. Currently, gaze is controlled by the movements of your head and not the physical gaze of your eyes. There are many ways to use gaze in Mixed Reality, such as pointing at objects, pointing at far-away locations (for use in teleportation), and having objects follow your gaze.
- *Gestures*: Gestures involve the use of your hands to control and manipulate your experience within Mixed Reality. Windows Mixed Reality devices such as the HoloLens heavily rely on the use of gestures for interacting with holograms. As of this writing, there are a limited selection of gestures, including the select or air-tap gesture, home or bloom gesture, hold, manipulation, and navigation. You'll learn about how to use each of these in this chapter.

■ **Note** Some Mixed Reality headsets may not have the sensors to support gestures and instead rely on motion controllers as their primary form of input.

- *Voice:* Voice input is the use of voice commands to interact with your Mixed Reality experience. Voice commands are extremely useful when developing Mixed Reality applications because they allow a high-level of control and customization without creating a cluttered UI. Users can say a word or series of words to select objects, activate features, and enhance their experience. Voice can also be used to dictate words and sentences (speech-to-text) for fast text input instead of using a keyboard.
- *Motion controllers:* Motion controllers are the primary input method for immersive Windows Mixed Reality headsets. Motion controllers are handheld controllers with precise spatial positioning features for accurately interacting with virtual objects.
- *Other hardware:* In addition to the primary input methods listed already, there is a wide range of hardware options that can be used with Windows Mixed Reality headsets. These include devices such as Bluetooth keyboards and mice, Bluetooth gamepads, clickers, and other Bluetooth accessories.

Later in this chapter, I'll walk you through how to utilize each input method in your Mixed Reality application.

Gaze Tutorial

This section covers some key elements used with the gaze input method. I'll show you how to use cursors to represent your gaze and provide an overview of how gaze is implemented in code.

Step 1: Set Up the Unity Scene

For this tutorial, we'll use a test scene from the HoloToolkit. If you haven't already, be sure to set up Unity for Mixed Reality development as described in Chapter 4. You may also refer to Chapter 4 for a refresher on how to run HoloToolkit test scenes in Unity.

Find the Cursor test scene (or `cursor.unity`) in your project panel by using the search bar or locating it within the folder structure. Drag the test scene into your Hierarchy, as shown in Figure 5-1. Be sure to unload (disable) all other scenes that you might have open.

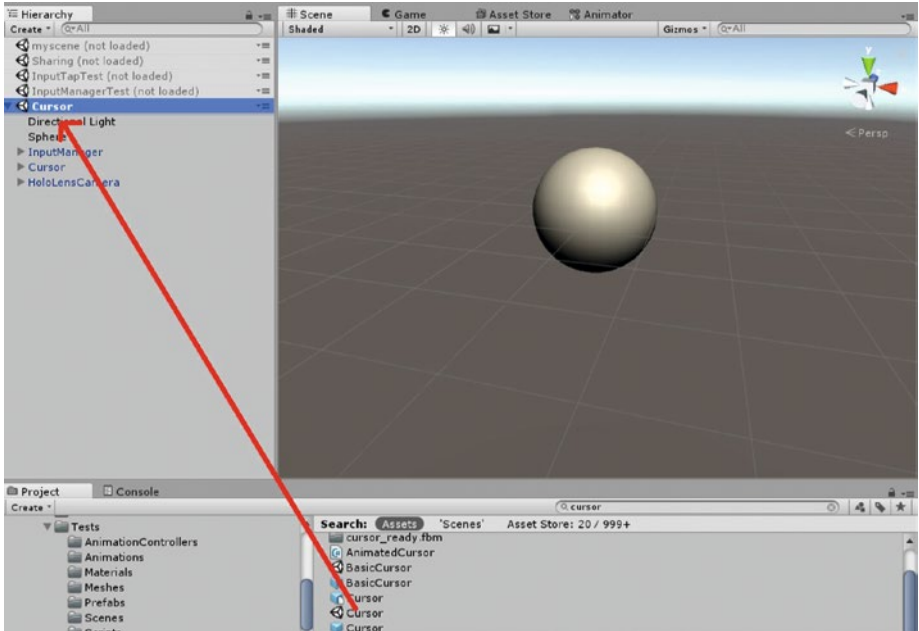


Figure 5-1. Open the *Cursor* test scene from the *HoloToolkit*

Step 2: Try the Scene

Within the *Cursor* scene, you should only see a sphere. Go ahead and try the scene by clicking the play button. Feel free to test within the Unity Editor—use holographic remoting to your device or deploy the application to your device.

When you're looking at the sphere (*gazing* at the sphere), your cursor becomes a torus (donut shape), as shown in Figure 5-2. When you gaze off the sphere, your cursor becomes a fuzzy dot, as shown in Figure 5-3.



Figure 5-2. When playing the scene, your gaze cursor will turn into a torus and follow the contour of the sphere

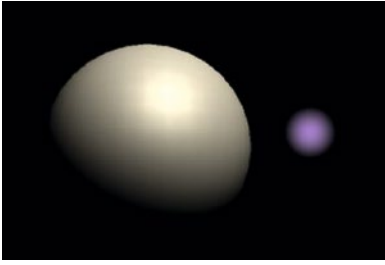


Figure 5-3. When your gaze is not pointed at the sphere, your cursor becomes a fuzzy dot

■ **Tip** By default, the test scene includes the ability for you to easily navigate your scene using your mouse and keyboard. This is useful if you want to do a quick test without your headset. Hold down the right mouse button and move the mouse to simulate head movements for gaze. Hold the Shift button or spacebar to simulate holding your hand in front of the HoloLens. Left-click to simulate an air-tap gesture (while holding Shift or spacebar). Use the keyboard arrow keys to walk around your environment.

Go ahead and stop the simulation when you're done testing the scene by clicking the play button again.

Step 3: Understand the Scene

Now that you've experienced the test scene and had some fun, it's time to understand how this scene works and how you can use elements from this scene in your own project. Let's start with the Hierarchy and work our way through the elements shown.

There are two primary objects in this scene that are important for gaze. The first is the `InputManager` object, which contains several important scripts, as shown in Figure 5-4. The second is the `Cursor` object, which contains two cursors: one for when your gaze is pointing to a hologram (the `CursorOnHolograms` object) and another for when your gaze is not pointing at a hologram (`CursorOffHolograms` object), as shown in the Hierarchy in Figure 5-4. Many of the other objects shown relate to gesture inputs, Unity Editor navigation options, and other features that we'll cover later.

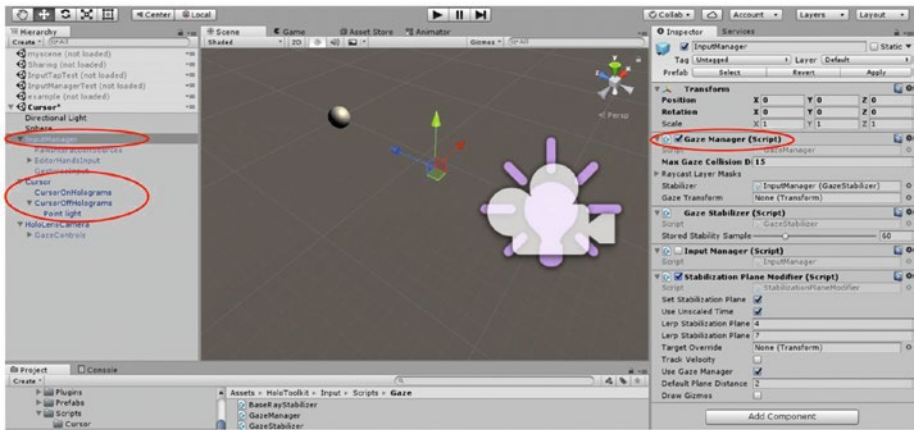


Figure 5-4. Important objects in the Cursor scene for the gaze input method

If you highlight the InputManager object in the Hierarchy, you'll see several scripts in the Inspector (see Figure 5-4). The most important of these is the `GazeManager.cs` script (circled). *The Gaze Manager manages everything related to a gaze ray that can interact with other objects.* The Gaze Manager creates an invisible ray or “beam” from the HoloLens, pointing in the direction the user is facing. The Gaze Manager script also captures information about the object that the ray hits and which objects to ignore.

The ray only extends as far out as the Max Gaze Collision Distance value that you can specify in the Inspector (see Figure 5-4). If a hologram is farther away than this distance, the ray won't touch the hologram, even if the user is looking directly at the hologram.

Stabilization scripts are important gaze-related scripts within the InputManager object that help reduce the jitter or shakiness of the experience.

Because the gaze ray is invisible, we want to use a cursor to represent where the user is gazing and whether the gaze ray is touching a hologram that's in front of the user. For this, we can see the *cursor prefab*, which is highlighted in the Hierarchy in Figure 5-5. The cursor prefab is a nice package that works seamlessly with the Gaze Manager script. It represents the end of your gaze ray with a fuzzy dot when the gaze ray is not touching a hologram. As mentioned, if the gaze ray touches a hologram, a donut-shaped cursor appears at the point where the gaze ray touches the hologram (refer to Figures 5-2 and Figure 5-3 for examples). The `ObjectCursor.cs` script obtains the gaze ray status from the `GazeManager.cs` script and controls the behavior of the cursor.

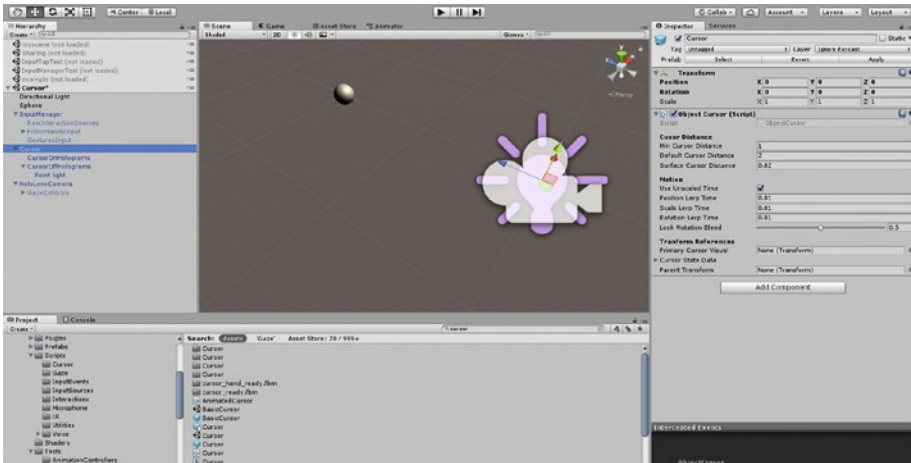


Figure 5-5. The cursor prefab (highlighted in blue) equips your project with a handy cursor

I encourage you to explore the various cursor settings in this prefab to see how the behavior of your cursor changes as you adjust values.

Step 4: Use Gaze in Your Project

Now that you have a basic understanding of the important components of the gaze input method and the cursor, you that some required scripts and objects are needed to create your own gaze and cursor functionality. Let's review what these are:

- *InputManager prefab*: Contains the GazeManager.cs script for gaze and stabilization scripts to reduce jitter and improve visual performance.
- *Cursor prefab*: Contains scripts and cursor objects to help users visualize the endpoint of the gaze ray and when the gaze ray touches a hologram.

One option for applying these settings to a new project scene is to search for (or navigate to) these prefabs and drag them into your Hierarchy. Often it's far more efficient to start with a pre-existing scene that already includes these core items and then use that “template scene” as a foundation for building your application. The Cursor test scene used in this section does not contains other core functionality (such as gesture and voice input capabilities) and is therefore too limited to be used as an effective template. As we walk through other input methods in this chapter, we will arrive at a full-featured test scene that you can repeatedly use as your template scene each time your start a new project.

Gestures Tutorial

In this section, I'll walk you through using gestures in your application. We'll explore the various classes of gestures and the use cases in which each are appropriate. We'll take a look at the code to see how gestures are implemented.

Step 1: Load the Test Scene

For this tutorial, we'll be loading the `InputManagerTest.unity` scene. As you've done with previous test scenes, search or browse for the `InputManagerTest` and drag it into your Hierarchy, as shown in Figure 5-6. Unload any other scenes that you might have open.

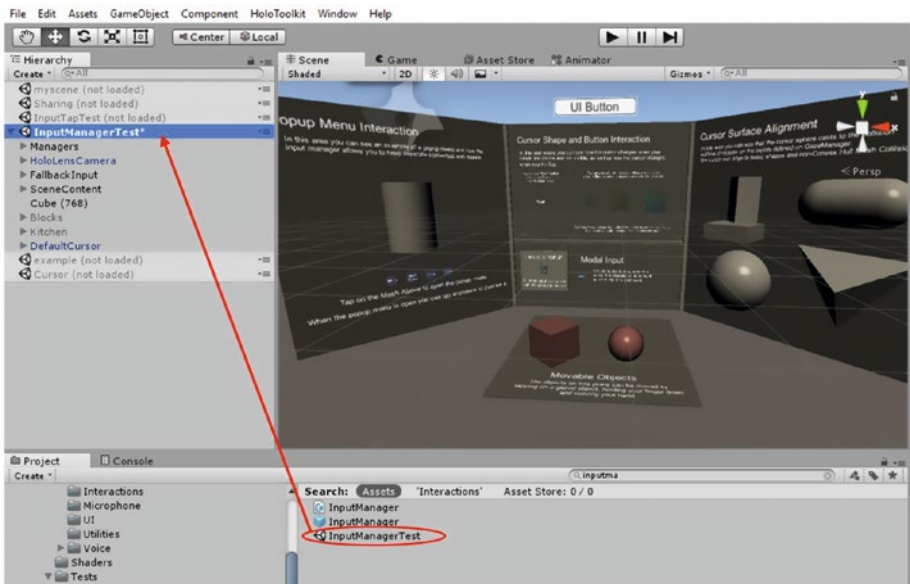


Figure 5-6. Open the `InputManagerTest` scene. This wonderful test scene is a playground for experimenting with gestures and inputs.

As shown in Figure 5-6, you'll immediately be greeted with a friendly display reminiscent of an intriguing science fair display.

Step 2: Try It Out

As before, go ahead and try this test scene out by clicking the play button in the Editor, remotely to your device, or deploying to your device. Some features highlighted in this demo include pop-up menus, cursor functionality, the ability to tap objects, the ability to drag/move objects, and more.

Are you interested in learning more about something you see in the demo? After you exit the demo (by clicking play again), you can select objects of interest within the scene window and view the Inspector to explore which scripts and components are responsible for the object's behavior. In Figure 5-7, I select the red cube in order to view the object's components in the Inspector panel on the right.

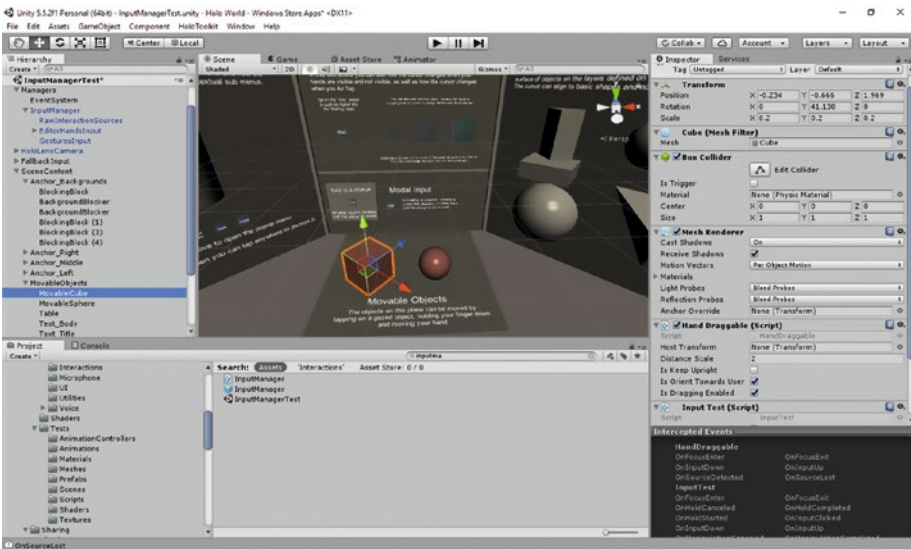


Figure 5-7. To learn more about how each object and feature works in this test scene, highlight the object of interest and explore the components in the Inspector panel on the right

There are many features of interest in this scene. The previous section already covered some of the cursor and gaze-related topics. We'll walk through some other key features in the next few steps.

Step 3: Use Air-Tap or Select Gesture

The most important gesture for Windows Mixed Reality devices is the *select* gesture. For the HoloLens, this is synonymous with the air-tap gesture. It's equivalent to the mouse click on a PC.

Let's explore one of the objects in our test scene that responds to the select gesture. As shown in Figure 5-8, there are three squares in the middle panel that change color when they're selected or air-tapped.

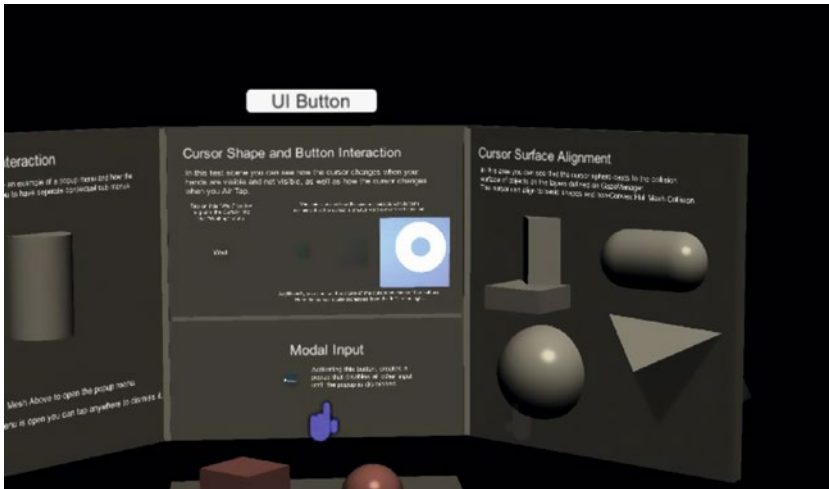


Figure 5-8. Each of the three squares in the middle panel responds to the air-tap or select gesture by changing color

When you select any of the three square objects, you will notice only one component in the Inspector panel—the TestButton.cs script, as shown in Figure 5-9.

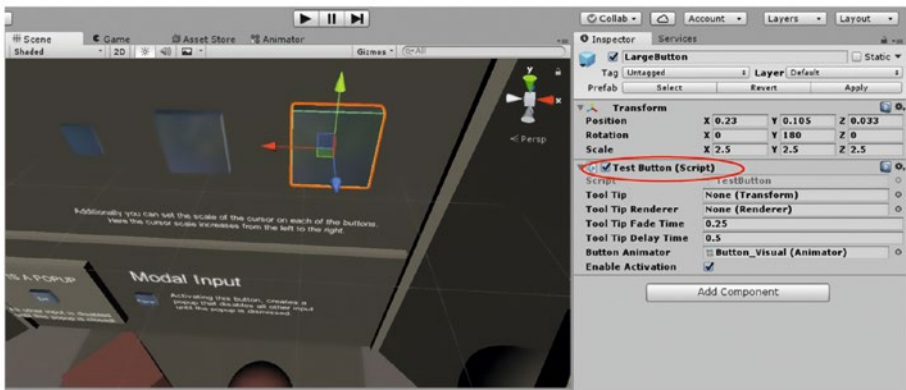


Figure 5-9. Select one of the three squares in the Unity editor to display the object's components in the Inspector panel

Open the TestButton.cs script in Visual Studio by double-clicking the script in the Editor or in the Project panel. As mentioned in the script's summary text, this script can be added to any object (including objects in your projects) in order to make it intractable.

■ **Note** Often, HoloToolkit scripts don't work independently. You'll need to ensure that the necessary prefabs and scripts are loaded into your scene. For example, any gesture-related script relies on the presence of the InputManager prefab in your scene.

There's a lot of C# code in the TestButton.cs script. Let's look at the core piece that activates upon detection of an air-tap or select gesture. Note that I've removed the code in the function in order for us to focus on the important pieces:

```
public void OnInputClicked(InputEventData eventData)
{
    //Do Stuff Here!
}
```

Add this piece to any of your scripts to perform tasks when you click/select/air-tap an object that contains your script. You'll also need the InputManager prefab in your scene as well as proper setup of your script (such as inclusion of appropriate HoloToolkit namespace and IInputClickHandler class, as seen at the beginning of the TestButton.cs script).

Step 4: Enter and Exit Focus

You'll notice that each of the three squares from Step 3 responds to the user's gaze by increasing in size when gazed upon and decreasing in size when no longer gazed upon. This behavior is made possible by the following pieces in the TestButton.cs script:

```
public void OnFocusEnter()
{
    // Do something when your gaze enters the object
}

public void OnFocusExit()
{
    // Do something when your gaze exits the object
}
```

The OnFocusEnter() function is activated when the user's gaze enters the object that this function references. The OnFocusExit() function is activated when the user's gaze exits the object.

There are limitless opportunities for these functions! To name a few examples:

- Play a noise when gazing on an object
- Change an object's color when gazing on it
- Make an object smaller or larger when gazing on it
- Make objects disappear when gazing on it

As with other input scripts in this section, these functions cannot be used in isolation. You will also need the `InputManager` prefab in your scene as well as proper setup of your script (such as inclusion of appropriate `HoloToolkit` namespace and `IInputClickHandler` class, as seen at the beginning of the `TestButton.cs` script).

Step 5: Move Objects

If you haven't already, try moving the red cube and sphere near the bottom of the test scene's exploratory area, as shown in Figure 5-10. You can move these objects by tapping on a gazed object, holding your finger down and moving your hand. Make sure that you're playing the scene in the Editor or remoting to your device, or that the application is deployed to your device.

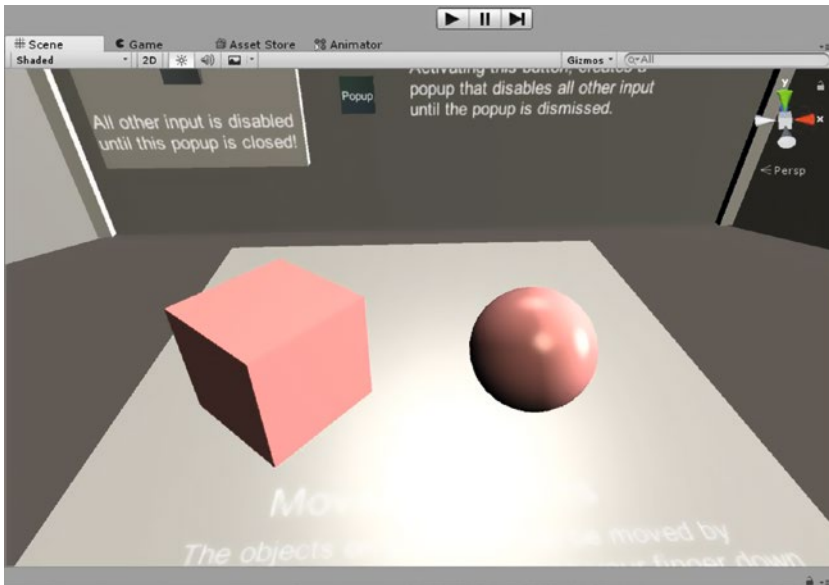


Figure 5-10. Movable objects in the test scene can be moved by tapping and holding with your finger and moving your hand

Objects are made movable by adding a script called `HandDraggable.cs`. Figure 5-11 shows the Inspector panel for the Movable Sphere object in the test scene. For your own project, be sure that the `InputManager` prefab is already added to your scene, because the `HandDraggable.cs` script relies on it. As shown in Figure 5-11, the script contains several customizable options in the Inspector panel. Let's walk through these:

- *Host Transform*: If you want to move an object other than the object that the script is attached to, you reference the object that you want to move in this field. An example application for Host Transform is a virtual joystick, where a user may control a joystick, but another object (perhaps a game character) may be the object that moves.
- *Distance Scale*: Think of this as a sensitivity factor. If the scale is very large, then small movements of your hand will be multiplied by this scale and mean very large movements for your object. For very fine manipulations, keep this scale small. For very large movements, increase the scale. You will need to explore various values to find a good range for your project.
- *Is Keep Upright*: Enable this to keep the object oriented upright as you move it around.
- *Is Orient Towards User*: Enable this to always orient the object toward the user as it is being moved.
- *Is Dragging Enabled*: Enable this to allow the object to move when dragged.

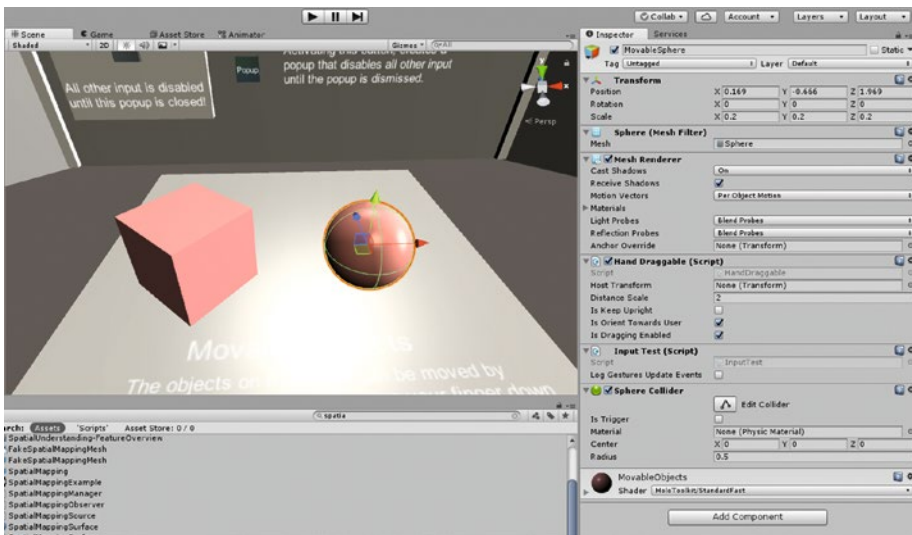


Figure 5-11. Making an object movable is easy. Just add the HoloToolkit's *HandDraggable*.cs script to any game object that you want to make movable.

It's important to ensure that a collider is attached to the object you want to control. The collider allows your gaze to “collide” with the object and lets gestures be directed to the object on which you're focused. In Figure 5-11, you see the selected sphere's collider as Sphere Collider in the Inspector panel.

Step 6: Implementing Gestures in Your Application

Now that you have a basic understanding of the gesture input methods, you know that required scripts and objects are needed to create your own gesture functionality. Let's review what these are:

- *InputManager prefab*: Contains the InputManager.cs script that tells your application how to direct gesture events. Contains the GazeManager.cs script for gaze and stabilization scripts to reduce jitter and improve visual performance.
- *Cursor prefab*: Contains scripts and cursor objects to help users visualize the endpoint of the gaze ray and when the gaze ray touches a hologram. Gaze and cursors are necessary for gestures to function and are useful for telling your application which object you're currently focusing on, which is generally the object that will also receive gesture events.
- *HandDraggable.cs*: If this script is attached to an object in the scene, it allows that object to become draggable with a gesture or to control another object's movements.
- *Gesture/gaze code elements*: These are elements that can be included in your code to respond to gesture and gaze events. A few examples are `OnInputClicked()` for responding to air-taps or select gestures, `OnFocusEnter()` for gazing on an object, and `OnFocusExit()` for gazing off an object.

The InputManagerTest scene that we've been exploring in this section is an excellent template scene to use when starting new projects. It contains the essential prefabs just listed and several unique examples that you can modify or expand for your project. As you gain experience, you may also consider building your own template scene that contains components you regularly use. If you want to start with one of the test scenes provided with the HoloToolkit, modify it, and save it as your own template scene, you may do so by selecting the scene in your Hierarchy and in the menu bar clicking File ► Save Scene As.

Voice Command Tutorial

This section walks you through key elements needed to enable voice commands in your Mixed Reality application. Voice commands are extremely useful when developing Mixed Reality applications because they allow a high level of control and customization without creating a cluttered UI. Users can say a word or series of words to select objects, activate features, and enhance their experience.

Often, my arms get fatigued when air-tapping for long periods of time. When this happens, I switch to using voice commands by saying "select" instead of air-tapping. This is one example of the power of voice commands. Well-thought-out use of voice in your application is key to making a good user experience.

As before, we'll start with a HoloToolkit test scene to explore how voice commands work for Windows Mixed Reality.

Step 1: Load the Test Scene

For this tutorial, we'll be loading the `FocusedObjectKeywords.unity` scene. As you've done with previous test scenes, search or browse for the `FocusedObjectKeyword` and drag it into your hierarchy, as shown in Figure 5-12. Don't forget to unload any other scenes that you might have open.

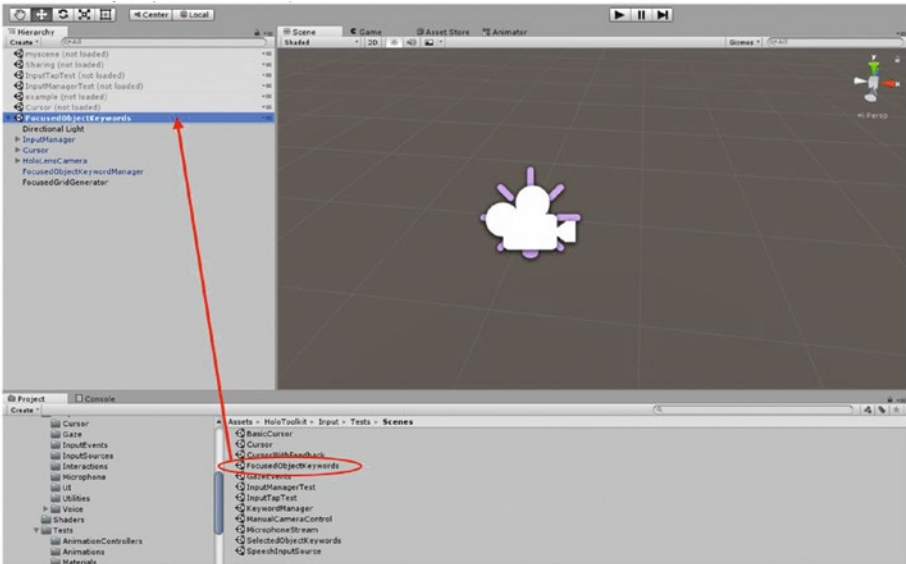


Figure 5-12. Open the `FocusedObjectKeyword` test scene to explore how to use voice commands in your application

Step 2: Try It Out

As before, go ahead and try out this test scene by clicking the play button in the Editor, remoting to your device, or deploying to your device.

Important Be sure to add the microphone capabilities in your app when using voice commands. In Unity, you can check the Microphone option at **Edit > Project Settings > Player > Settings for Windows Store > Publishing Settings > Capabilities**.

Upon starting the test scene, you should see a few game objects, such as cubes and spheres. When you gaze on an object, it will be highlighted in red, as shown in Figure 5-13. When your gaze is focused on an object, try saying out loud, “Make bigger,” or, “Make smaller,” and watch as the selected objects resize. If you’re testing via the Unity Editor, the voice commands should work if you have a microphone attached to your PC (or if you have build-in microphone or microphone integrated into your webcam).

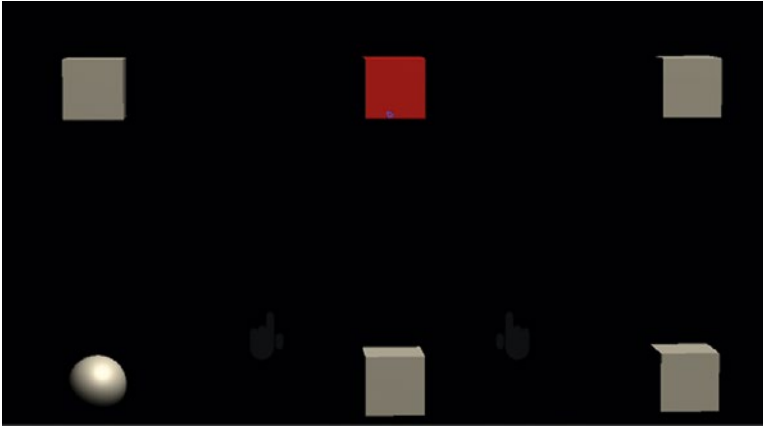


Figure 5-13. Upon playing the test scene, you’ll see several objects. Gaze on an object to highlight it and try saying out loud, “Make bigger,” or, “Make smaller”

Step 3: Understand the Scene

Now that you’ve had some fun trying out voice commands, let’s take a deeper look at what makes this experience possible. As shown in Figure 5-14, several items are included in the Hierarchy of the FocusedObjectKeywords test scene.

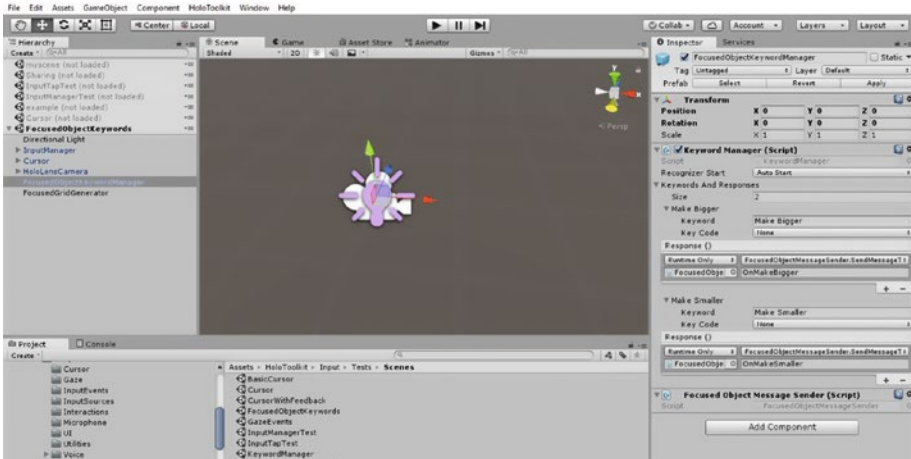


Figure 5-14. Voice commands are made possible by the *FocusedObjectKeywordManager* prefab

By now, you should be very familiar with some of the objects shown, such as the *InputManager* prefab and the *Cursor* prefab. You will also see *FocusedGridGenerator* in the Hierarchy, which is a prefab that generates the grid of cubes and spheres. Although that generator is not the focus of this section, I encourage you to explore the scripts and methods that make spawning objects possible.

The key prefab in this scene is the *FocusedObjectKeywordManager* prefab. As you can see in Figure 5-14, the prefab contains two scripts: *KeywordManager.cs* and *FocusedObjectMessageSender.cs*. Here's a basic overview of how this prefab works:

1. First, the *KeywordManager.cs* script starts the keyword recognizer, so that your microphone is constantly listening for the voice commands that you provide it in the Inspector. In this case, the pre-loaded voice commands are *Make Smaller* and *Make Bigger*.
2. When you say a voice command or phrase, *KeywordManager.cs* will check to see if it matches the provided word or phrase. If there's a match, the script will trigger a message. For example, the message associated with the *Make Bigger* voice command is *OnMakeBigger*.
3. The *FocusedObjectMessageSender.cs* script is responsible for sending the message triggered in #2 to the object that is currently being gazed upon.
4. Once the message is sent to the object, the object must have a script to respond to the message. In our case, each of the spawned objects has *OnMakeSmaller()* and *OnMakeBigger()* methods in its *ScaleObjectMessageReceiver.cs* script.

Step 4: Add Your Own Voice Command

The best way to learn how something works is to try it yourself. Let's add another voice command to the existing set. The objects currently grow when you say, "Make bigger," and they shrink when you say, "Make smaller." Let's add a voice command for Move Up to have the objects move upward.

In the `FocusedObjectKeywordManager` prefab, look for the `KeywordManager.cs` script and edit the `Size` field in the Inspector from 2 to 3, as shown in Figure 5-15. This means we're changing the number of voice commands from two to three.

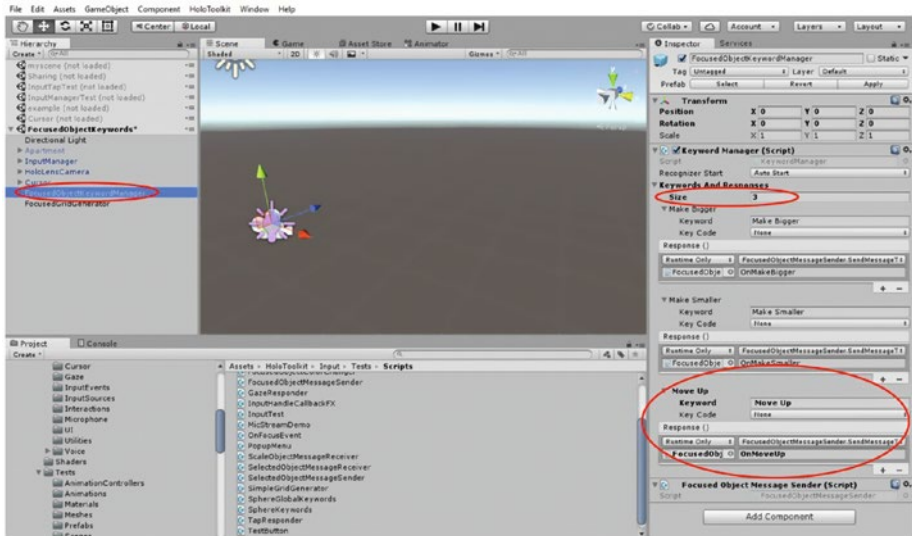


Figure 5-15. Add your own voice command by increasing the `Size` field and customizing keywords in the `KeywordManager.cs` script

Once you increase the `Size` to 3, you'll notice that a new keyword entry appears in the Inspector. You may need to expand the new entry to edit it. Go ahead and change the `Keyword` field to `Move Up`. The keyword is the voice command. Be careful not to put any spaces before the keyword, because that would prevent the code from recognizing your voice command.

Under the `Response` section, add the message `OnMoveUp`, as shown in Figure 5-15. Note that my use of `Move Up` and `OnMoveUp` are completely arbitrary—you could also write your own custom keywords and messages.

■ **Tip** If you wanted multiple voice commands to trigger the same action, you can keep adding keywords while keeping the same message. For example, you could add `Move Up`, `Move Upward`, and `Go Up`, and for each keyword use the same message of `OnMoveUp`.

Now every time you say, “Move up,” a message of `OnMoveUp` will be sent to the object you’re gazing upon. However, the objects in the scene won’t know what to do when they receive the `OnMoveUp` message. We need to add a method called `OnMoveUp()` to the script on the game object. Let’s take a look at the script attached to the game objects.

You can find the script by searching your Project panel for `ScaleObjectMessageReceiver.cs` or you can find it in the Inspector by going to the `FocusedGridGenerator` prefab in your Hierarchy and then double-clicking either `FocusedObjectCube` or `FocusedObjectSphere` in the Inspector panel. You’ll see `ScaleObjectMessageReceiver.cs` attached to the object.

When you open this script, you’ll see both the `OnMakeBigger()` and `OnMakeSmaller()` methods. Go ahead and create the `OnMoveUp()` method by typing in the following code:

```
public void OnMoveUp()
{
    transform.Translate(0.0f, 0.2f, 0.0f);
}
```

This small addition simply moves the object upward (in the Y-direction) by 0.2 units.

Go ahead and try out your shiny new voice command. Gaze on each object and say, “Move up,” and you’ll see the object move up.

Step 5: Use Voice Commands in Your Own Project

Congratulations! You’ve successfully uncovered how voice commands work with Windows Mixed Reality and how to add your own voice commands to the existing test scene. Adding voice commands to a new scene or a different scene is a simple process. Let’s review what you need to do to enable voice commands in your own project:

- Add the `FocusedObjectKeywordManager` prefab from the `HoloToolkit` to your scene.
- Within this prefab, change the `Size` variable to be the number of keywords or voice commands that you would like to have. A keyword can be a word or a phrase.
- Once you specify the number of keywords, specify what each keyword will be (the voice command that the user will say out loud).
- For each keyword, be sure to specify at least one message. The message will be sent to the object on which you gaze. You can also press the + button near the lower right corner of the keyword manager script responses in the inspector panel (See Figure 5-15) to add additional objects and messages—for example, if you wanted to implement a universal voice command, regardless of gaze.

Implementing voice commands in Windows Mixed Reality is relatively simple yet greatly enhances the user experience of your application. Next I discuss some best practices for using voice in your application.

Best Practices for Voice Commands

Voice commands are an excellent way for users to interact with Mixed Reality applications. Here are some best practices to keep in mind for implementing voice in your application:

- Use keywords that have two or more syllables. This helps voice recognition for a wide range of accents.
- Design your app for users accidentally triggering voice commands. Allow users to undo an action, where appropriate (for example, when deleting an object by accident).
- Make sure all voice commands are distinctive. If two or more commands sound similar, the voice recognizer may activate the wrong command.
- Make sure your voice commands are easily recognized across a range of accents whenever possible.
- Voice commands are a wonderful way to quickly access nested menus or other situations in which multiple gestures may be required.
- Consider providing the user with a list of voice commands if the UI doesn't already reveal what they are.
- Put a microphone icon next to buttons that can be tapped or activated with a voice command. If possible, make all buttons and UI elements voice-enabled.

Other Hardware Input

A wide range of hardware options can be used with Windows Mixed Reality headsets. These include devices such as motion controllers, Bluetooth keyboards and mice, Bluetooth gamepads, clickers, and other Bluetooth accessories.

Although it's considered a best practice to avoid using traditional PC hardware (keyboards and mice) for Mixed Reality experiences, there may be some applications where these input methods are appropriate.

Gamepads are an excellent choice for applications that may involve moving a third-person object, such as a game character, holographic helicopter, or holographic race car.

Motion controllers are wireless handheld controllers with six degrees of freedom for accurate placement and manipulation of 3D objects. They are an excellent form of input for immersive Mixed Reality Headsets. These controllers were first announced at the Microsoft Build conference in 2017 but were not yet available at the time of this writing.

Summary

Congratulations! This chapter covered the primary forms of input for Windows Mixed Reality headsets. We walked through several tutorials on how gaze, gestures, and voice commands work with Mixed Reality. We learned how to enable these features in our own application and how to leverage the HoloToolkit to easily implement powerful input features.

As you continue your Mixed Reality development journey, keep in mind that the industry is still in its infancy. Everyone generally agrees that input methods for Mixed Reality devices are somewhat clunky and awkward at times, lacking precision and elegance. Think about what an ideal input experience would be like, and don't be afraid to try out new ways of interacting with your virtual environment. You never know who will introduce an input method as relevant to Mixed Reality as the mouse was to the PC.

CHAPTER 6



Using Spatial Mapping

In this chapter, you'll learn how to use one of the most defining features of Windows Mixed Reality headsets like the HoloLens: spatial mapping. You'll learn how to apply spatial mapping in Unity using the HoloToolkit and unwrap some neat tricks that you can do with spatial mapping. You'll learn how to identify walls, floors, ceilings, and chairs. You'll also learn how to anchor digital objects to your physical environment, and how to save those anchors so that your digital objects will persist where you left them, even after closing and re-opening your app.

What Is Spatial Mapping?

Devices like the HoloLens are constantly tracking their environment and building a 3D model of the area that they're in. This is called *spatial mapping*. Without spatial mapping, holograms wouldn't be able to be set on floors and tables, or be pinned to walls. Objects in other rooms would still be visible, degrading the user's experience.

Spatial mapping is important for several reasons:

- *Occlusion*: This tells the HoloLens which holograms to hide from view. For example, if you place a hologram in your hallway and then walk into another room, the spatial map of that room's walls will prevent you from seeing the hologram in your hallway. If there were no spatial map, you'd see the hologram as if it were visible through your walls, causing an unrealistic experience.
- *Placement*: This allows users to interact with the spatial map—for example, to pin items to your walls, allow characters to sit on your sofa (as seen in Microsoft's Fragments app), or automatically decorate your surroundings.
- *Persistence*: This allows for *holographic persistence*, which is the ability for holograms to stay where the user left them, even after turning off the device. Your HoloLens will (remarkably) be able to remember your space and restore any holograms you had placed in that space.

- *Physics*: This allows objects to collide with or bounce off your walls, furniture, and floors, resulting in a more realistic experience.
- *Navigation*: Use gaze to allow game characters and other holograms to follow along mapped surfaces.

For more information on spatial mapping and the sensors involved, see Chapter 1.

Spatial Mapping Tutorial

In this section, I walk you through setting up some basic spatial mapping capabilities. I show which elements from the HoloToolkit are needed to enable spatial mapping and provide some tips for a good experience.

Step 1: Set Up Unity Scene

This tutorial uses a test scene from the HoloToolkit. If you haven't done so already, be sure to set up Unity for Mixed Reality development as described in Chapter 4. Refer to Chapter 4 for a refresher on how to run HoloToolkit test scenes in Unity.

Find the TapToPlace test scene (or TapToPlace.unity) in your Project panel by using the search bar or find it within the folder structure. Drag the test scene into your Hierarchy, as shown in Figure 6-1. Be sure to unload (disable) all other scenes that you might have open.

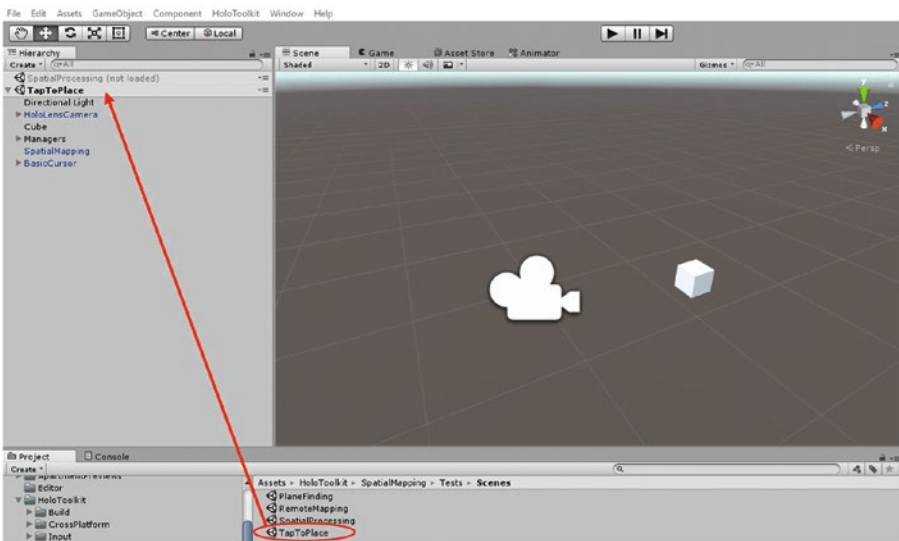


Figure 6-1. Open the TapToPlace scene from the HoloToolkit to explore a basic implementation of spatial mapping

Step 2: Try It Out

The next step is to try it out by clicking the play button.

If you have a HoloLens or similar Windows Mixed Reality device, I highly recommend using Unity's holographic remote to device feature (see Chapter 3 for a discussion and tutorial on holographic remoting) in order to experience spatial mapping of your physical environment. You may also deploy the app to your HoloLens.

If you don't have a device, or prefer not to use it for this test, be sure to use Simulate in Editor with Unity's holographic emulation (again, see Chapter 3 for more information on this) in order for spatial mapping to work.

■ **Tip** When using the Simulate in Editor mode of Unity's holographic emulation, Unity will load in a 3D model of a room or area that you can use to test your spatial mapping capabilities without using a headset. Unity provides several different rooms and spaces that you can use. To choose a 3D space, use the Room drop-down menu in the holographic emulation window.

After clicking the play button, you'll see the scene's cube in your area, but you won't be able to see the spatial map. After tapping the cube, the spatial map will appear, and the cube will follow your gaze, as shown in Figure 6-2. If wearing the HoloLens, you'll see the spatial map well aligned to your physical surroundings, as shown in Figure 6-3. When you tap a second time, the spatial map will become invisible again.

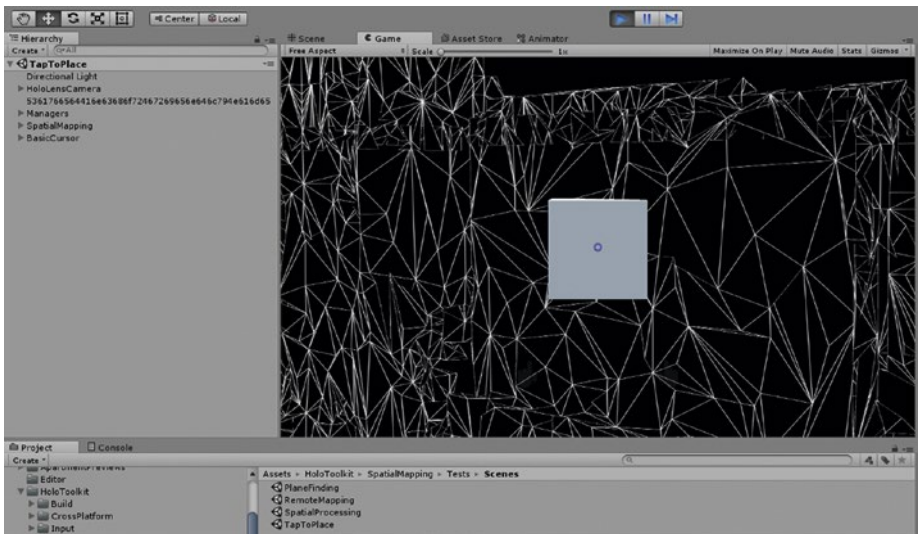


Figure 6-2. View of the spatial map, as seen through the Unity Editor

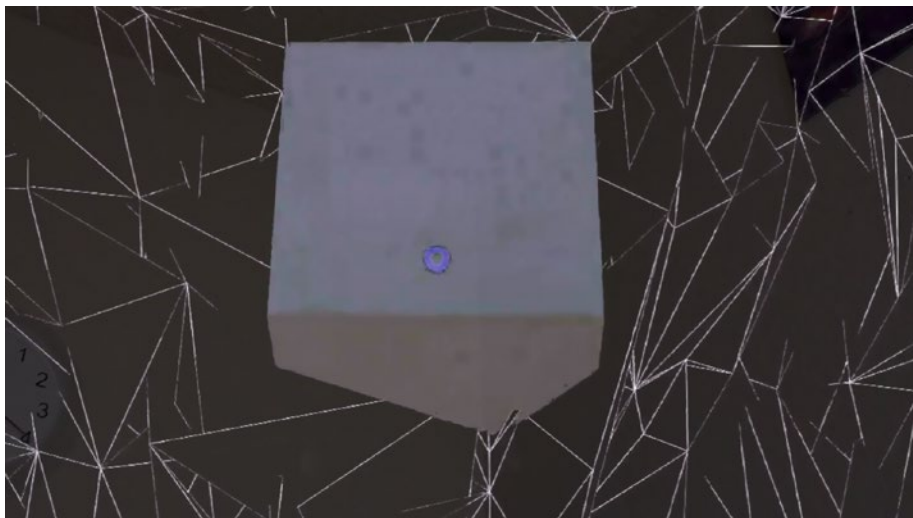


Figure 6-3. When viewed through the HoloLens, the spatial map will align well with your physical surroundings

As you can see, the rendering of the spatial map is a collection of vertices, edges, and faces. It looks like a net covering your surroundings (later we'll see how to change the spatial mapping appearance). The 3D object generated by spatial mapping is often called the *spatial mapping mesh*.

Step 3: Understand the Scene

Now that you've had the opportunity to experience spatial mapping, let's dig into our scene to learn about the key components that make spatial mapping possible.

Looking at the scene's Hierarchy, we see some familiar items that we've already learned about in Chapter 5, including the InputManager prefab and the BasicCursor prefab. There is one unfamiliar item in our Hierarchy: the SpatialMapping prefab, as shown in Figure 6-4.

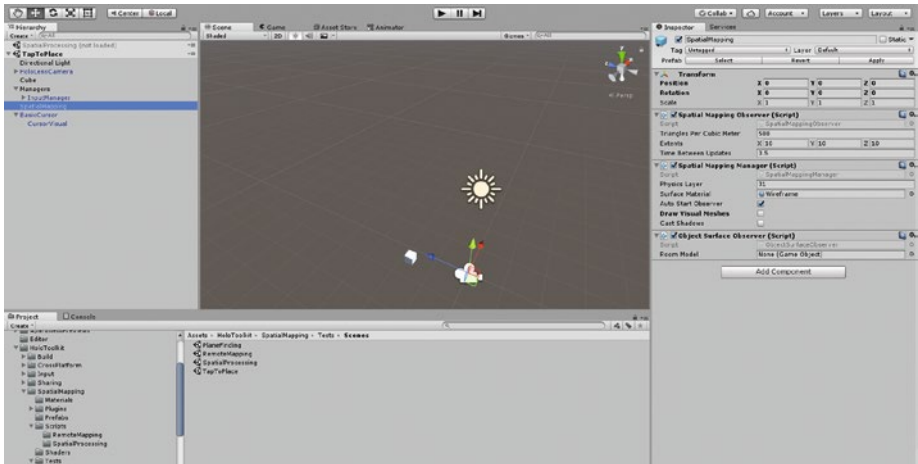


Figure 6-4. The *SpatialMapping* prefab is all that's needed to enable spatial mapping in your project

This small prefab, containing only three scripts, is all that's responsible for spatial mapping. You can easily find this prefab in the HoloToolkit and drag it into your project to enable spatial mapping. This is yet another example of how the HoloToolkit makes it easy for developers to quickly and efficiently set up a Windows Mixed Reality project.

Let's walk through each of the three scripts in the *SpatialMapping* prefab.

- SpatialMappingObserver.cs*:** This script is responsible for managing the surfaces observed on the HoloLens and renders them so they can be displayed in the scene. You can adjust the resolution of the spatial map in the Inspector panel using the Triangles Per Cubic Meter field. You can also adjust how far out from the HoloLens you want to observe by adjusting the Extents variables, and you can specify how often to process spatial mapping updates using the Time Between Updates field.
- SpatialMappingManager.cs*:** This script allows you to choose to load a saved spatial mapping mesh or collect data in real time from the HoloLens. To help with performance and avoid the processor-intensive task of constantly scanning a room, it can be beneficial to save the current room to memory and only scan occasionally or as needed. In the Inspector panel, you may also select the material to use for rendering the spatial mapping data.
- ObjectSurfaceObserver.cs*:** This script is used when you're not using a HoloLens device for spatial mapping but instead are using a pre-existing 3D model of a room or area within the Unity Editor. You can specify a custom 3D model in the Inspector panel.

In addition to the scripts in the SpatialMapping prefab, the Cube game object also has a script attached to it called TapToPlace.cs, which is responsible for making the Cube interactive and placeable on the spatial mapping mesh. There's also a new script called WorldAnchorManager.cs. If you click the Managers item in the Hierarchy, you'll see this script. I discuss world anchors and spatial anchors in greater depth later in this chapter.

Step 4: Use Spatial Mapping in Your Application

As mentioned in the previous step, enabling spatial mapping in your application is as easy as dragging the SpatialMapping prefab from the HoloToolkit to your project Hierarchy. Simply use the Project panel's search bar to find the SpatialMapping prefab or navigate to it in the directory, as shown in Figure 6-5.

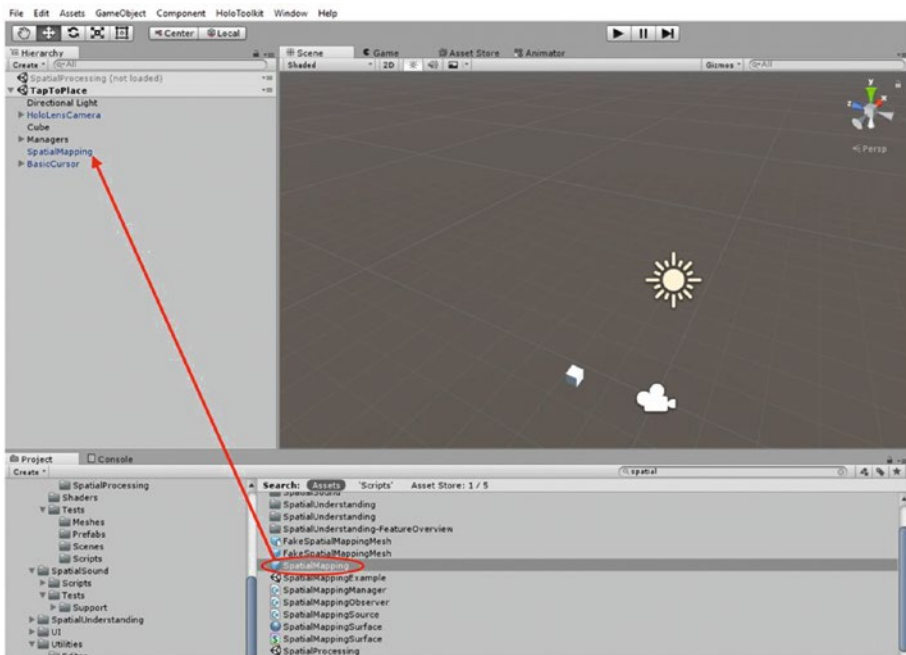


Figure 6-5. To apply spatial mapping to your application, simply apply the SpatialMapping prefab from the HoloToolkit to your scene's Hierarchy

You must also enable SpatialPerception to your Unity application by going to Edit ► Project Settings ► Player ► Settings for Windows Store ► Publishing Settings ► Capabilities. See Figure 6-6 for an illustration of this setting.

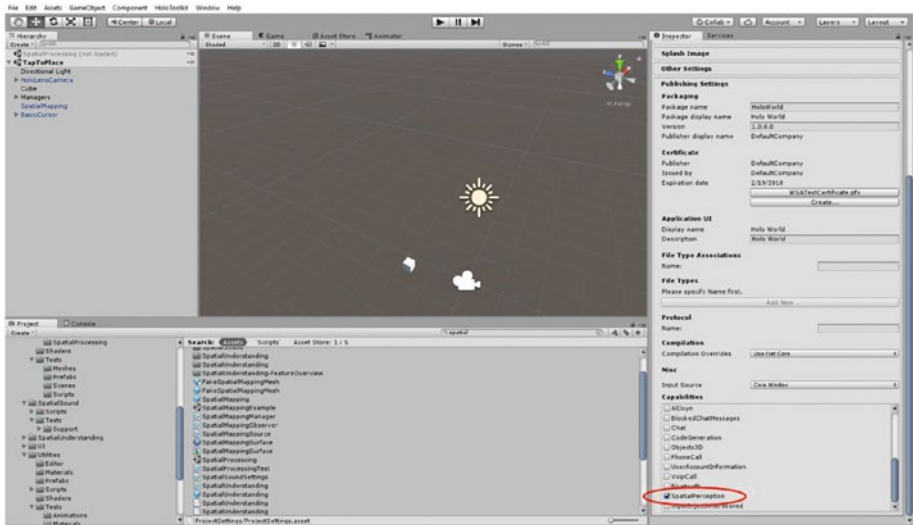


Figure 6-6. Be sure to enable *SpatialPerception* in Unity's Publishing settings for spatial mapping to work

Spatial Plane Finding Tutorial

Rather than just applying a digital mesh “blanket” over physical surfaces, we can leverage the HoloLens’ computational power to find planes in our environment. In this section, I walk you through enabling *plane finding* in your application and discuss why plane finding is important.

Step 1: Set Up the Unity Scene

This tutorial uses a test scene from the HoloToolkit. If you haven’t already done so, be sure to set up Unity for Mixed Reality development as described in Chapter 4. Refer to Chapter 4 for a refresher on how to run HoloToolkit test scenes in Unity.

Find the *PlaneFinding* test scene (or *PlaneFinding.unity*) in your Project panel by using the search bar or finding it within the folder structure. Drag the test scene into your Hierarchy, as shown in Figure 6-7. Be sure to unload (disable) all other scenes that you might have open.

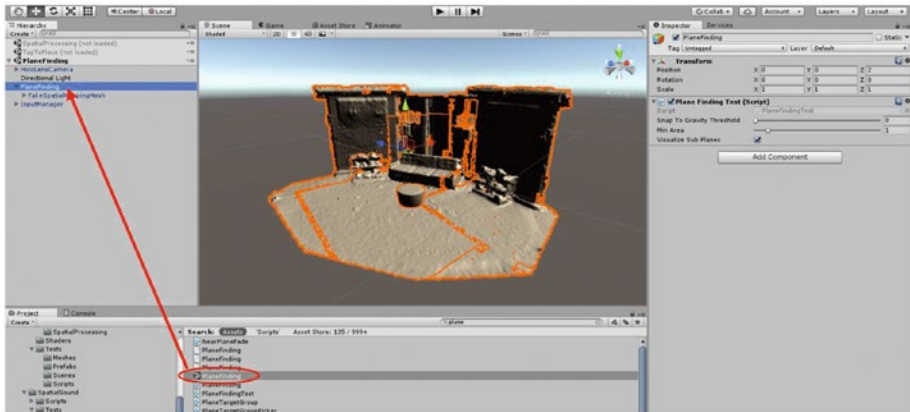


Figure 6-7. Open the *PlaneFinding* scene to explore the HoloToolkit's spatial mapping plane finding feature

Upon loading the scene, you should see a 3D model of a room. You may notice that this scene doesn't use spatial mapping but rather the pre-existing room model. We'll try out plane finding on a real spatial mapping mesh later in this section.

Step 2: Try It Out

Go ahead and click the play button to start exploring the scene. This scene is intended to be experienced within the Unity Editor, so we won't be using the headset.

To see the identified planes, *switch to the scene view while in Play mode*. You will be able to visualize the planes, as shown in Figure 6-8.

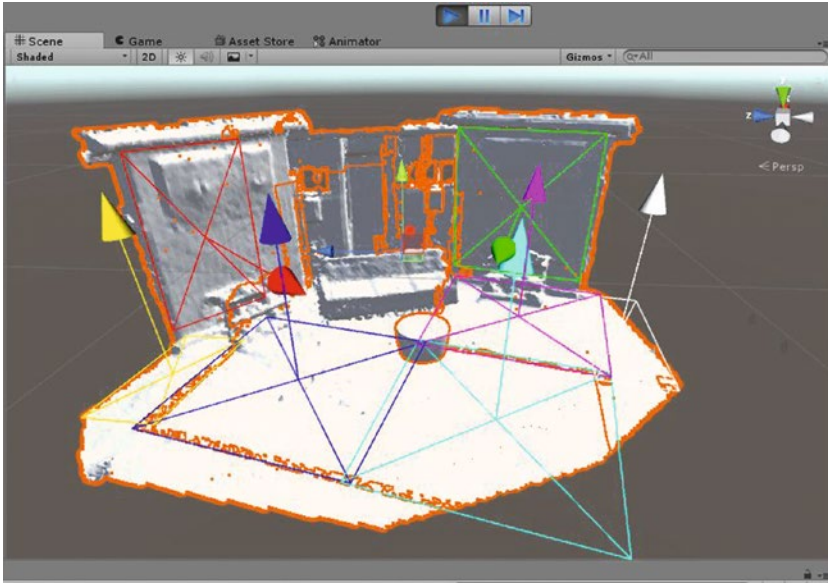


Figure 6-8. Identified planes can be seen while the scene is running and you're in the scene view

While in the scene view, feel free to adjust the parameters for `PlaneFindingTest.cs` in the Inspector panel (you'll need to select the `PlaneFinding` item in the Hierarchy to see the script).

This scene provides a controlled environment to test plane finding and allows you to carefully explore parameters.

Step 3: Load the Spatial Processing Scene

Now that you've had the opportunity to explore a basic plan finding scene and implementation, let's expand this capability to an actual spatial mapping mesh. Find and load the `SpatialProcessing` scene (`SpatialProcessing.unity`), as shown in Figure 6-9, and unload the `PlaneFinding` scene.

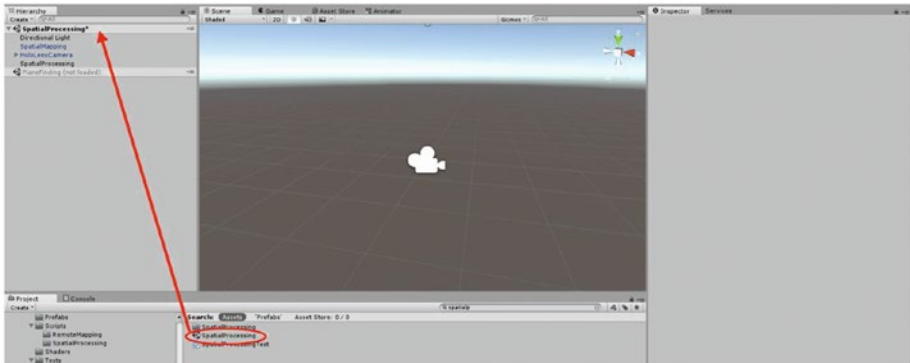


Figure 6-9. Load the SpatialProcessing scene and disable the PlaneFinding scene

Step 5: Try Out the SpatialProcessing Scene

Try out the spatial processing scene by clicking the blue play button while using Unity's holographic emulation or deploying to your device. At first you'll see the regular spatial mapping mesh, but after a few seconds the spatial mapping mesh will be replaced with large white rectangles representing the identified planes of the room, as shown in Figure 6-10.

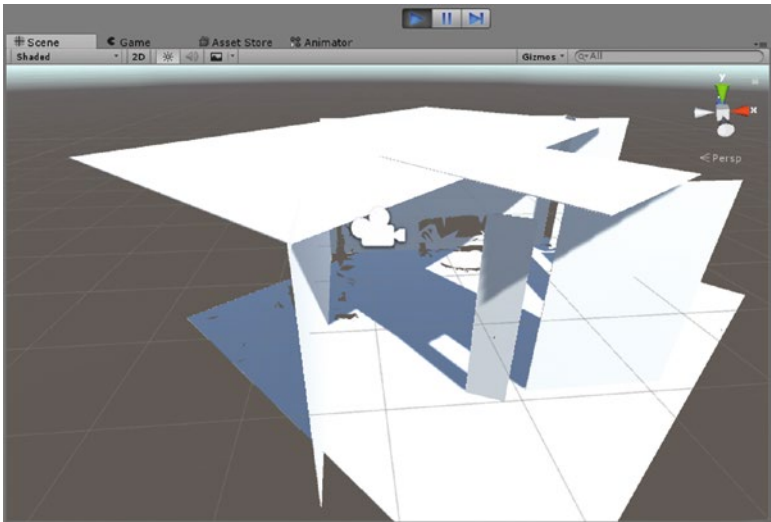


Figure 6-10. After a few seconds, the spatial processing feature will identify planes and replace the spatial mapping mesh with white planes

Step 6: Understand the SpatialProcessing Scene

When I first tried the SpatialProcessing scene while wearing the HoloLens, I was overjoyed at seeing how perfectly aligned the planes were to my walls. SpatialProcessing is made possible by the SpatialProcessing item in the Hierarchy, which includes three important scripts, as shown in Figure 6-11.

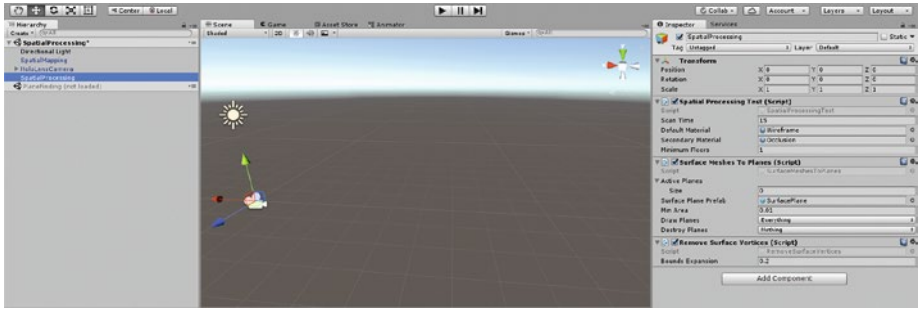


Figure 6-11. The SpatialProcessing object (highlighted in blue) contains three important scripts, as can be seen in the Inspector panel

Let's walk through each of these three scripts to see how they come together for visualizing planes in our environment:

- SpatialProcessingTest.cs:** This script allows you to control a variety of settings related to spatial processing. You'll notice that you can adjust these settings directly from the Inspector panel. The first field is the Scan Time, which is the number of seconds to allow the SurfaceObserver to scan the environment. The longer the scan time, the more time you'll have to build out a good spatial mapping mesh of your environment. After the time expires, this script will stop the SurfaceObserver (that is, stop scanning the environment) and start the mesh processing. This script also allows you to set the default material, which lets you visualize the spatial mapping mesh during scanning. The secondary material allows you to visualize the spatial mapping mesh after scanning is complete. The Minimum Floors field is the minimum number of floor planes needed to exit processing. If you set the minimum number of floors to 1, but no floors are detected, the script will continue trying to find a floor plane.

- *SurfaceMeshesToPlanes.cs*: This script is responsible for finding planes from the mesh and generating planes. In the Inspector, you may set the `MinArea`, which is the minimum area required before a plane will be created. A larger number means you'll have larger but fewer planes in your scene, whereas a smaller number means you'll create plans for smaller surfaces in your scene. There are also two drop-down lists in the Inspector where you can specify which types of surfaces to draw planes for and which types of surfaces to destroy. The `snapToGravityThreshold` variable (in the script but not shown in the Inspector) is used to align planes with gravity so that they appear more level. The *Surface Plane* prefab determines the appearance of the planes. Feel free to edit this prefab if you want to modify the plane appearances—for example, if you want to have the floor planes be a different color from the wall planes.
- *RemoveSurfaceVertices.cs*: This script is responsible for removing vertices (removing parts of the spatial mapping mesh) that fall within boundaries that you specify. You may want to remove vertices if you need holes in your spatial mapping mesh or if you want to reduce polygon count in order to improve the performance of your application. In our *SpatialProcessing* scene, the *SpatialProcessingTest.cs* script uses the planes generated by *SurfaceMeshesToPlanes.cs* as the boundaries. It sends these boundaries to the *RemoveSurfaceVertices.cs* script, which then removes the spatial mapping mesh near the generated planes. The *Bounds Expansion* parameter that is visible in the Inspector allows you to expand the boundaries you provide, to remove more vertices around the boundaries.

■ **Tip** Because the *SurfaceMeshesToPlanes* script allows you to specify plane types (walls, floor, ceiling, table, and so on), you can selectively send some of these planes to the *RemoveSurfaceVertices* script. This is useful if you'd like to selectively remove parts of your spatial mapping mesh—for example, if you'd like to remove only your ceiling.

Step 7: Use Spatial Processing in Your Application

To enable spatial processing in your application, you need to do the following:

- Make sure to enable spatial mapping, as described in the previous tutorial.
- Make sure to enable spatial perception, as described in the previous tutorial.

- Find and add the `SpatialProcessingTest.cs` script to your Hierarchy to control scan times and mesh visualization.
- Find and add the `SurfaceMeshesToPlanes.cs` script to your Hierarchy. Adjust settings in the Inspector panel as needed by your project.
- Optionally, you may also find and add the `RemoveSurfaceVertices.cs` script to your Hierarchy to remove parts of your meshes that you replace with a plane.

Spatial processing is excellent for scenarios where you need to identify planes without necessarily visualizing them. For example, if I need to place a hologram on the floor, I can use spatial processing to identify and create an invisible floor plane and then proceed to place holograms on the floor. Other examples include placing objects on walls, hanging holograms from the ceiling, calculating headset height above the floor, and more.

Occlusion Tutorial

In this section, I walk you through implementing occlusion to your spatial mapping mesh. As mentioned, occlusion allows parts of objects that are fully or partially behind walls and other surfaces to become invisible, just as they would in the physical world. This increases your holograms' realism and improves your user's experience.

Step 1: Load the TapToPlace Scene

Let's reload the TapToPlace scene that we started with at the beginning of this chapter, shown back in Figure 6-1. Feel free to try out the application again. You will notice that the cube is visible, regardless of whether it's in your space or behind a physical wall.

Step 2: Apply Occlusion

Next, we want to apply a new material that will allow our spatial mapping mesh to block objects behind it while appearing invisible when viewed through the HoloLens or other device. The HoloToolkit provides a useful item to achieve this. Browse or search for the Occlusion material in your HoloToolkit folders within the Project panel. Once you locate it, drag it to the Surface Material field of the `SpatialMappingManager.cs` script within the Inspector panel, as shown in Figure 6-12.

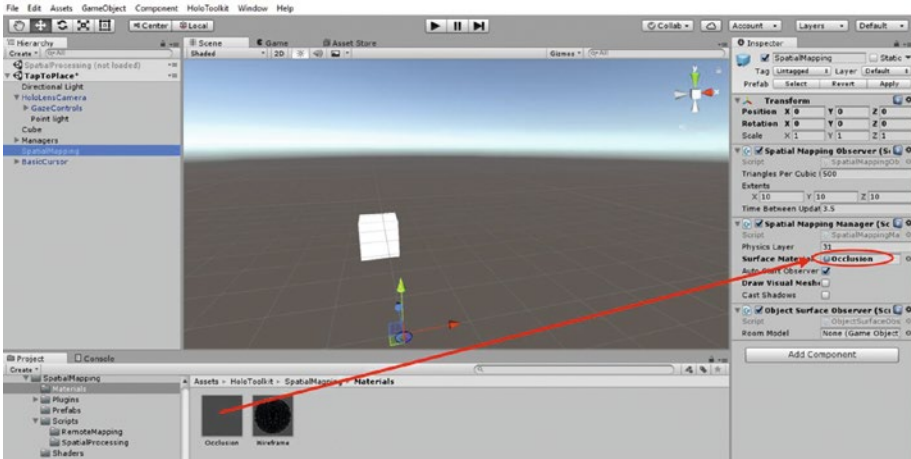


Figure 6-12. Apply the Occlusion material to the Surface Material field of the Spatial Mapping Manager script

Step 3: Try It Out

As before, use holographic remoting to test the app with the new Occlusion material. Initially, the occlusion material won't be rendered until you tap the cube. While in Placing mode (with the cube following your gaze), the occlusion material will be rendered. Because the material is transparent, you won't directly see the spatial mapping mesh, but you will see that part of the cube will be occluded by its environment.

As soon as you release the cube (by tapping it again), the spatial mapping mesh will no longer be rendered, and the cube will no longer be occluded by your surroundings. Go ahead and manually turn on occlusion while the application is running in Unity by checking the Draw Visual Meshes box for the SpatialMappingManager.cs script in the Inspector panel, as shown in Figure 6-13.

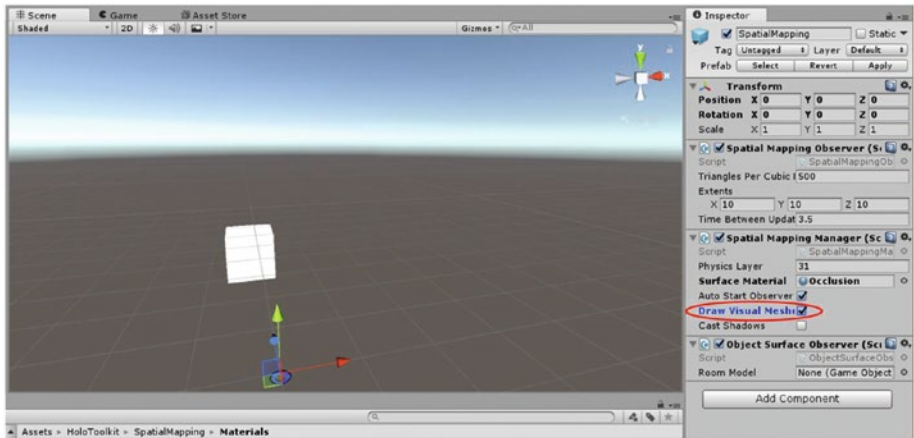


Figure 6-13. While the application is being simulated in Unity, check the Draw Visual Meshes checkbox to force the occlusion material to be rendered

As you can see from my tests, the cube was fully visible when there were no obstructions between me and the cube (Figure 6-14), but the cube was partially visible when obstructions were present (Figure 6-15).

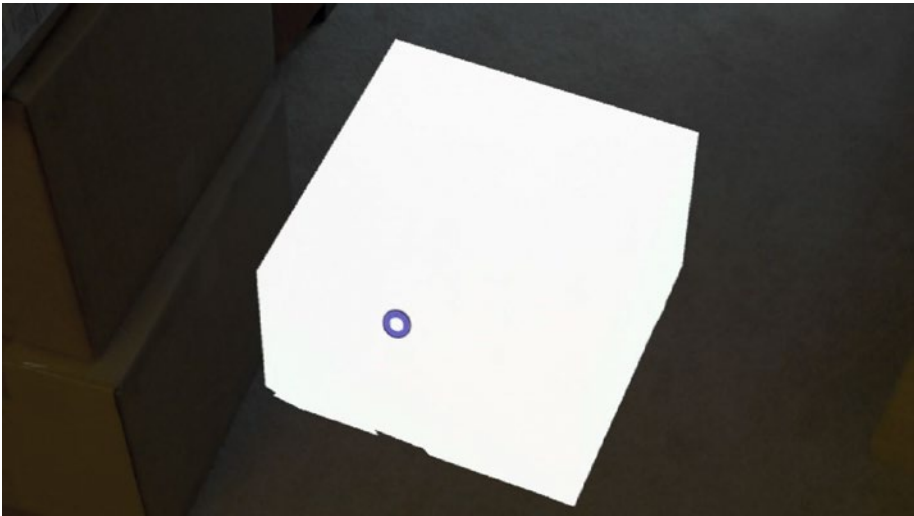


Figure 6-14. The white cube appears full when there are no obstructions



Figure 6-15. When obstructed (in this case by boxes), only the unobstructed part of the cube is visible

Because no processing or smoothing of the spatial mapping mesh occurs in this scene, you'll notice some imperfections in the spatial mapping mesh, as can be seen in Figure 6-15, with the sharp artifacts. Replacing the spatial mapping mesh with smooth planes and other smoothing techniques can help with occlusion and visualization.

Step 4: Use Occlusion in Your Application

Occlusion is an essential part of Mixed Reality development, especially for projects that utilize spatial mapping. Without occlusion, distant holograms in other rooms or behind objects would still be visible, causing the experience to be confusing and unnatural.

To apply occlusion to your project, you need to render your spatial mapping mesh using the Occlusion material found in the HoloToolkit, as shown in Step 2 of this section.

A more advanced treatment of the spatial mapping mesh may utilize multiple materials throughout your app. For example, your application may start out using one material that's visible (such as during a room-scanning phase of your app) and then later switch to the invisible occlusion material.

You may actively switch between materials for spatial mapping when scripting by using the `SpatialMappingManager.SetSurfaceMaterial()` function. See the following code for an example of this implementation:

```
if (condition == true)
{
    SpatialMappingManager.SetSurfaceMaterial(surfaceMaterial);
}
```

In the preceding code, `surfaceMaterial` is a previously assigned material. You may, for example, declare it as `public Material SurfaceMaterial` and drag and drop a material using the Unity Editor's Inspector panel, as we did earlier in this tutorial.

Spatial Understanding Tutorial

In this section, I walk through how to enable spatial understanding for your application. One of the most powerful examples that the HoloToolkit has to offer is the Spatial Understanding example. *Spatial understanding* can be thought of as a much more powerful version of the plane finding feature covered earlier in this chapter.

Here are a few things that spatial understanding allows you to do:

- Place objects on floors, ceilings, and walls
- Place objects in the air away from you or near you, without touching walls
- Place objects on the floor away from you or near you
- Find the largest wall and place objects on it
- Find sittable surfaces (so you can have characters sit on anyone's chair)
- Identify chairs and couches
- Identify large empty surfaces
- Allows users to “paint” their spatial mesh to limit the scanned area
- Smooth the spatial mapping mesh

Step 1: Set Up the Unity Scene

This tutorial uses a test scene from the HoloToolkit. If you haven't done so already, be sure to set up Unity for Mixed Reality development as described in Chapter 4. Refer to Chapter 4 for a refresher on how to run HoloToolkit test scenes in Unity.

Find the `SpatialUnderstandingExample` test scene (or `SpatialUnderstandingExample.unity`) in your Project panel by using the search bar or find it within the folder structure. Drag the test scene into your Hierarchy, as shown in Figure 6-16. Be sure to unload (disable) all other scenes that you might have open.

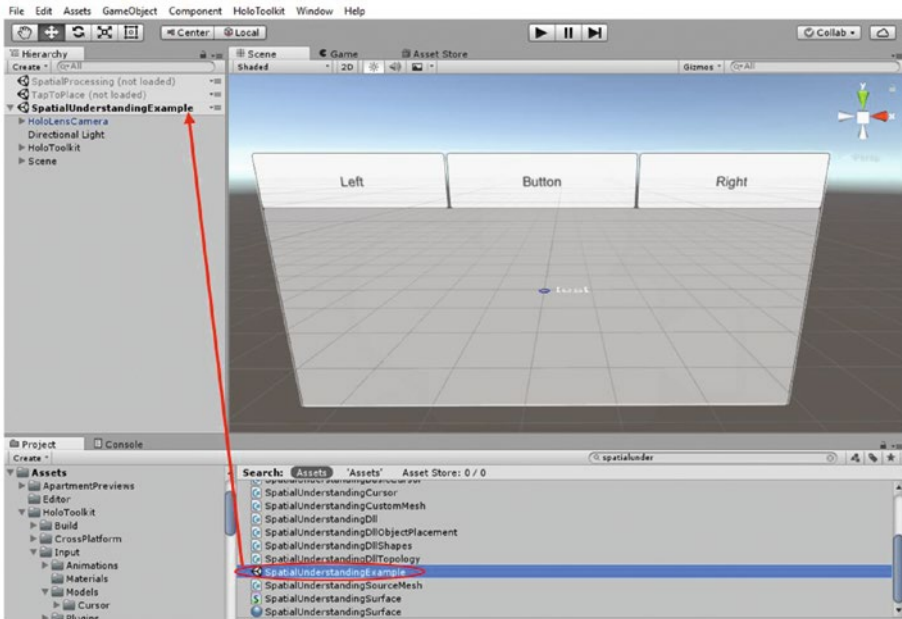


Figure 6-16. Open the *SpatialUnderstandingExample* scene to explore the HoloToolkit's spatial understanding features

Step 2: Try It Out

For this example scene, I highly recommend taking the time to build and deploy to your HoloLens or similar headset. Spatial understanding is an awe-inspiring example provided by the HoloToolkit and is best experienced on your device, without lag or limitation, and in a larger area of your home or office with interesting features nearby (table, chair, open area, walls, ceiling). If you prefer, you may still try out this scene from within the Unity Editor using holographic emulation or remoting to your device.

The application will first ask you to scan your environment. You'll immediately notice a very smooth implementation of the spatial mapping mesh, with well-leveled meshes on walls, floors, and ceilings. See Figure 6-17 for an example of the spatial mapping mesh used by spatial understanding.



Figure 6-17. The spatial mapping mesh processing by spatial understanding is remarkably smooth

Next, you may notice that a menu item has been placed on a wall near you, as shown in Figure 6-18. I recommend exploring all buttons and tabs in this menu to gain familiarity with what spatial understanding has to offer.

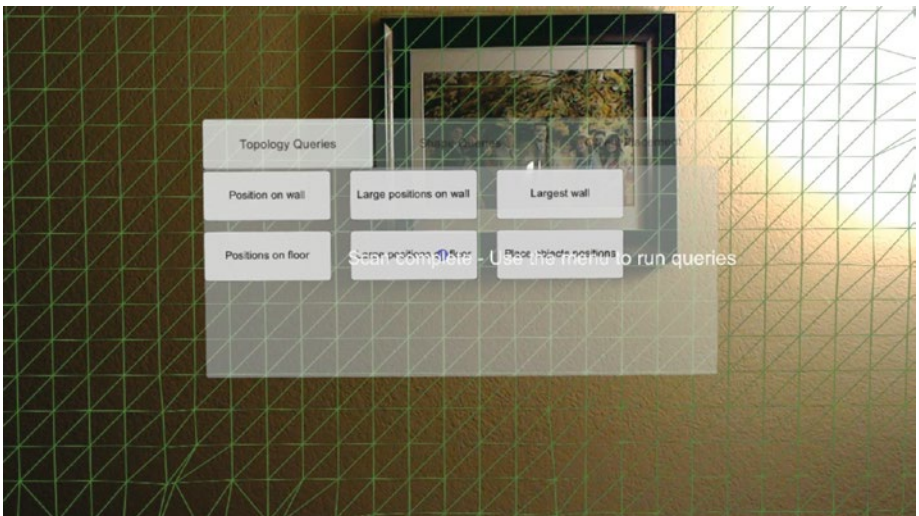


Figure 6-18. Explore all buttons and tabs offered in the spatial understanding menu. Prepare to be amazed!

When selecting buttons, a typical experience will involve rectangular objects flying from the menu to recognized surfaces around your mapped room.

Step 3: Use Spatial Understanding in Your Application

Entire chapters could be written about the inner workings of spatial understanding and all the ways developers could leverage it in their applications. In this step, I point out the key components needed to enable spatial understanding in your application and how to get started with it.

Enabling spatial understanding involves the following:

- Making sure spatial mapping is enabled, as shown previously in this chapter
- Locating the `SpatialUnderstanding` prefab and loading it into your scene's Hierarchy

The spatial understanding prefab contains three scripts, as shown in Figure 6-19.

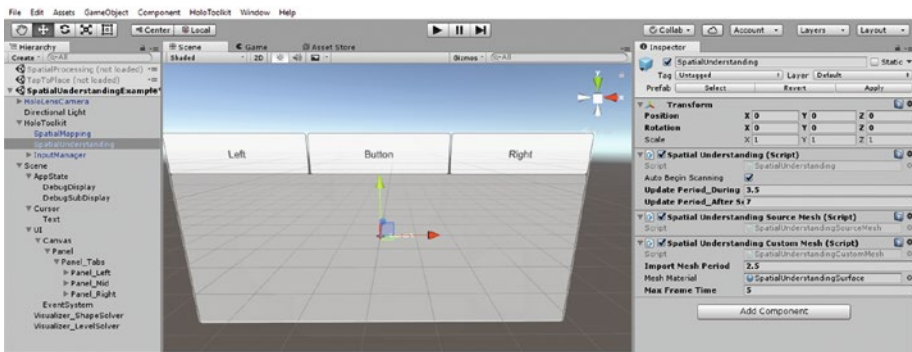


Figure 6-19. The `SpatialUnderstanding` prefab contains three important scripts

In simple terms, here is what each of the scripts are responsible for:

- *`SpatialUnderstanding.cs`*: This is responsible for managing the spatial understanding scanning process.
- *`SpatialUnderstandingSourceMesh.cs`*: This is responsible for providing the source (raw) spatial mapping mesh to the spatial understanding feature.
- *`SpatialUnderstandingCustomMesh.cs`*: This is responsible for generating the custom spatial mapping mesh, during spatial understanding processing and after it's complete.

To learn how to utilize all the data generated for the spatial understanding process, you'll want to dig into the following scripts: `SpaceVisualizer.cs` and `LevelSolver.cs`. These scripts are not part of the Spatial Understanding prefab or module, but they are part of the

Spatial Understanding Example scene, provided to show developers how to utilize spatial understanding data.

Let's look at one example code snippet from SpaceVisualizer.cs:

```
public void Query_Topology_FindLargeWall()
{
    ClearGeometry();

    // Only if we're enabled
    if (!SpatialUnderstanding.Instance.AllowSpatialUnderstanding)
    {
        return;
    }

    // Query
    IntPtr wallPtr = SpatialUnderstanding.Instance.UnderstandingDLL.
        PinObject(resultsTopology);
    int wallCount = SpatialUnderstandingDllTopology.QueryTopology_
        FindLargestWall(
            wallPtr);
    if (wallCount == 0)
    {
        AppState.Instance.SpaceQueryDescription = "Find Largest Wall (0)";
        return;
    }

    // Add the line boxes
    float timeDelay = (float)lineBoxList.Count * AnimatedBox.DelayPerItem;
    lineBoxList.Add(
        new AnimatedBox(
            timeDelay,
            resultsTopology[0].position,
            Quaternion.LookRotation(resultsTopology[0].normal, Vector3.up),
            Color.magenta,
            new Vector3(resultsTopology[0].width, resultsTopology[0].length,
                0.05f) * 0.5f)
        );
    AppState.Instance.SpaceQueryDescription = "Find Largest Wall (1)";
}
```

This function helps to find the largest wall in the room and places a pink rectangle over the position of the largest wall. The key pieces of code here are `resultsTopology[0].position` and `resultsTopology[0].normal`, where we can directly get the information about the largest wall and place objects on or near it. I encourage you to look through these scripts and use the code as inspiration for use in your own projects.

Interesting things to try:

- Have an avatar sit on a chair or sofa at your friend's house.
- Put a holographic clock on your wall.
- Place a holographic dinner on your dining room table, complete with food, plates, silverware, and more.
- Create Roman pillars that extend from your floor to the ceiling.

Spatial Anchors and Persistence

This section briefly discusses the importance of spatial anchors. I walk you through how to use spatial anchors in your Unity project and provide some best practices when using spatial anchors.

Spatial anchors are locations in your application that are anchored to the real world. This is part of what makes Mixed Reality possible. Without spatial anchors, the virtual experience viewed through your device would become increasingly disconnected with reality over time. The HoloLens and other similar Mixed Reality devices do their best to scan and recreate your physical world using spatial mapping. However, this process isn't perfect, and the spatial map may be improving and adjusting over time. These adjustments would cause your holograms to appear to be shifting away from where you placed them, unless you "anchored" the holograms to the spatial map using a spatial anchor.

How to Use Spatial Anchors

Attaching and removing spatial anchors is a relatively simple process. A spatial anchor is called a *world anchor* in Unity. To attach an anchor to your game object, use the following method:

```
WorldAnchor anchor = gameObject.AddComponent<WorldAnchor>();
```

To remove an anchor from a game object that you don't intend to move, use `Destroy`:

```
Destroy(gameObject.GetComponent<WorldAnchor>());
```

To remove an anchor from a game object that you intend to move, use `DestroyImmediate`:

```
DestroyImmediate(gameObject.GetComponent<WorldAnchor>());
```

We need to destroy anchors on game objects because (as the name implies) anchors prevent us from moving objects. To move an anchored object in your scene, you first need to immediately destroy the anchor, move it, then create the anchor again. See the following code example:

```

DestroyImmediate(gameObject.GetComponent<WorldAnchor>());
gameObject.transform.position = new Vector3(2, 2, 2);
WorldAnchor anchor = gameObject.AddComponent<WorldAnchor>();

```

Hologram Persistence

What if you could save a spatial anchor to your device's memory and load it the next time your application starts? If you did that, your holograms and objects would be exactly where you placed them in the physical world, even after closing and re-opening your application. The good news is that this feature, called *persistence*, is available and widely used with spatial anchors.

You can save spatial anchors to your device in a place called the `WorldAnchorStore`. Most of the hard work is done for us using a script in the HoloToolkit called `WorldAnchorManager.cs`.

Include the `WorldAnchorManager` in your scene by finding it in the Project panel and dragging it to an item in your Hierarchy, such as a Managers object, as shown in Figure 6-20.

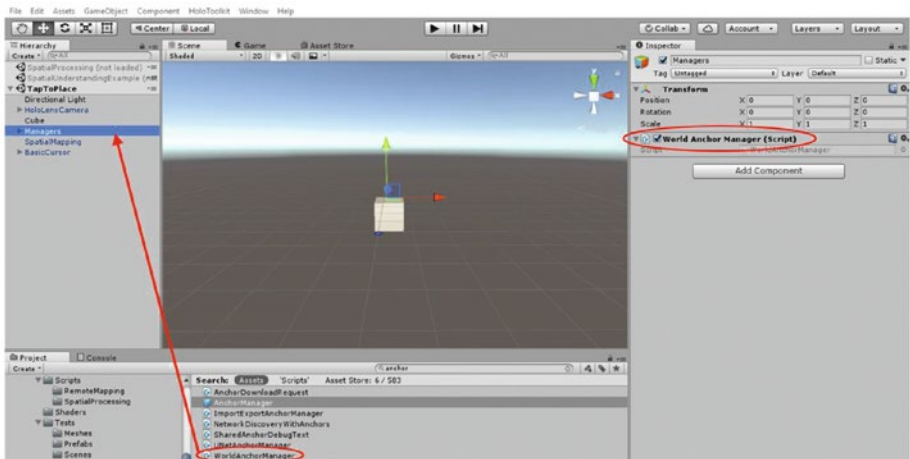


Figure 6-20. Include the `WorldAnchorManager.cs` script in your scene for a simplified spatial anchor and persistence experience

In Figure 6-20, you'll notice that I'm in the `TapToPlace` test scene from the HoloToolkit that we've been working with in this chapter. The `TapToPlace` scene already utilizes the `WorldAnchorManager.cs` script and the `TapToPlace.cs` script (attached to the cube). Feel free to dive into the scripts in this scene to explore how hologram persistence is set up in a working scene.

■ **Note** To experience persistence across app sessions, you need to deploy the application to your device or run your application in the HoloLens emulator outside of Unity. Persistence won't work when using the Unity Editor or holographic emulation within Unity.

With the `WorldAnchorManager.cs` script attached to your project, you'll be able to call its functions from other scripts. I include a few useful examples here. In the following examples, `anchorManager` refers to the `WorldAnchorManager` script:

```
anchorManager = WorldAnchorManager.Instance;
```

Use the following code to attach an anchor to your game object and save it to the `WorldAnchorStore` using a custom anchor name `SavedAnchorFriendlyName` that you can use to later retrieve the anchor in another app session:

```
anchorManager.AttachAnchor(gameObject, SavedAnchorFriendlyName);
```

To remove the anchor (perhaps while moving the object), you can use the following code:

```
anchorManager.RemoveAnchor(gameObject);
```

To load an existing anchor and attach it to a game object, use the same `AttachAnchor` code that we use to create anchors:

```
anchorManager.AttachAnchor(gameObject, SavedAnchorFriendlyName);
```

If the anchor store already has an anchor with the custom name you provided, it will load the anchor instead of creating a new one.

If you want to get all the existing anchor names in the anchor store and iterate through them, use the following code:

```
var ids = anchorManager.AnchorStore.GetAllIds();  
  
foreach (var id in ids)  
{  
    anchorManager.AttachAnchor(gameObject, id);  
}
```

A Note on Sharing Anchors

Not only do you have the ability to save anchors to your device, you can also transfer anchors to other devices. When two or more devices that are in the same physical room share anchors (and associated data), they see holograms and objects in the same place as everyone else. This allows for truly awe-inspiring shared sessions where multiple people can collaborate on the same project or view the same experience together.

Summary

Congratulations! You're now equipped with core knowledge about spatial mapping and can start taking advantage of some cool spatial mapping tools. Let's review what you learned in this chapter:

- What spatial mapping is
- All about the SpatialMapping prefab and all associated scripts
- How to enable spatial mapping in your application
- How to use spatial mapping to find and identify planes in your environment
- How to occlude objects using spatial mapping for a more realistic effect
- How to use spatial understanding to unleash the power of spatial mapping, identify objects and surfaces in your environment, and place objects on key surfaces in your environment
- All about spatial anchors and how to use them in your application
- How to persist objects and holograms across app sessions

You may not have thought there was so much to cover in a chapter about spatial mapping. Spatial mapping is extremely important for Mixed Reality. In fact, it's the headset's understanding of the physical environment that warrants the *mixed* in Mixed Reality, allowing our applications to mix the virtual and physical worlds together.

We've only touched the tip of the iceberg with regard to spatial mapping. Many untapped opportunities are waiting to be explored and implemented. Here are just a few examples of spatial mapping ideas I've heard mentioned:

- Expanding the spatial mapping mesh to make your room or area appear larger than it actually is
- Virtually painting your walls and furniture to see what various color options would look like
- Making holes in your walls to give the sensation that you can see through them

As you continue your developer journey, think about creative ways you can leverage spatial mapping and all associated tools. Be sure to think outside the box. Keep in mind that your spatial mapping mesh doesn't need to obey the laws of physics like your real walls and furniture do. The sky is the limit to what you can achieve.

CHAPTER 7



Spatial Sound

This chapter covers how to make the most of spatial sound in your applications. We rely heavily on our ears to precisely locate real objects around us. Our sense of hearing is able to detect extremely small differences in the arrival of sound waves at each of our ears in order to locate the position of the sound source in 3D space.

In the context of mixed reality, this is called *spatial sound*. Developers can leverage Mixed Reality audio tools that perform complex calculations in order to spatialize sound. These tools determine how sound waves should be modified and adjusted for each ear in order to “trick” our brains into hearing the sound as if it came from a specific point in 3D space and not from the speakers themselves. This vastly increases the feeling of immersion. Users will still be able to hear virtual objects (if the objects are intended to make noise) around them, even if they can’t see those objects. This is especially important for devices like the HoloLens, where the field of view is limited and users might not see holograms in their entire peripheral vision.

Here are different ways developers can leverage spatial sound:

- *Increase immersion:* Make the user feel like they are immersed in an experience where the holograms are all around them.
- *Call attention to holograms that are outside the field of view:* Play audio from a hologram that can’t be seen by users in order to prompt the user to look in the direction of that hologram.
- *Provide a better interactive experience:* When the user interacts with holograms or user interface elements in a Mixed Reality application, having spatial audio cues or effects coming from the point of interaction increases the sense of realism. Think of how satisfying it is to hear the “snap” of a light switch. You can now recreate this satisfaction in your application using spatial sound.

■ **Note** Spatial sound only works in Windows 10. If you’re developing your Mixed Reality application on a previous version of Windows, spatial sound won’t work.

Spatial Sound Tutorial

In this section, I walk you through a tutorial on how spatial mapping works in a Mixed Reality application.

Step 1: Set Up the Unity Scene

For this tutorial, we'll use a test scene from the HoloToolkit. If you haven't done so already, be sure to set up Unity for Mixed Reality development as described in Chapter 4. Refer to Chapter 4 for a refresher on how to run HoloToolkit test scenes in Unity.

Find the AudioOcclusionTest test scene (or AudioOcclusionTest.unity) in your Project panel by using the search bar or find it within the folder structure. Drag the test scene into your Hierarchy, as shown in Figure 7-1. Be sure to unload (disable) all other scenes that you might have open.

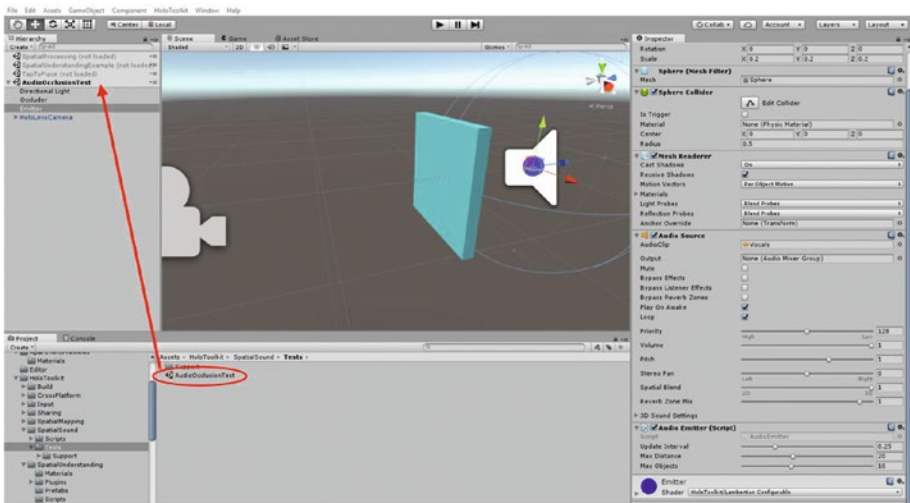


Figure 7-1. Load the AudioOcclusionTest scene from the HoloToolkit by dragging it into your Hierarchy

Upon loading the test scene in Unity, you'll notice a blue square (the Occluder) with a sphere (the Emitter) behind it. The Emitter is responsible for emitting the sound you will hear when you test this scene. The blue Occluder is responsible for occluding audio. More on that in Step 3.

Step 2: Try It Out

To experience the spatial sound from this test scene, you can use Unity's holographic emulation feature to remotely connect to your HoloLens or headset. You may also deploy the application to your headset using Visual Studio.

When you're ready, go ahead and click the play button to test the application. When you first start the application, you'll begin to hear a singing voice that is somewhat muffled.

■ **Note** If you're using Unity's holographic emulation and remoting to test this scene, be sure to turn down your PC's volume completely so that you only hear the audio through your headset. This allows you to fully experience the spatial sound effect without other speakers interfering with the experience.

The sound of the voice is muffled when the Occluder (the blue square) is between you and the sphere, as shown in Figure 7-2. When you walk around the square so that it's not between you and the sphere (as shown in Figure 7-3), you'll hear the singing voice loud and clear coming from the sphere.



Figure 7-2. When you first simulate the test scene, you'll see a blue square in front of you and hear a muffled singing voice

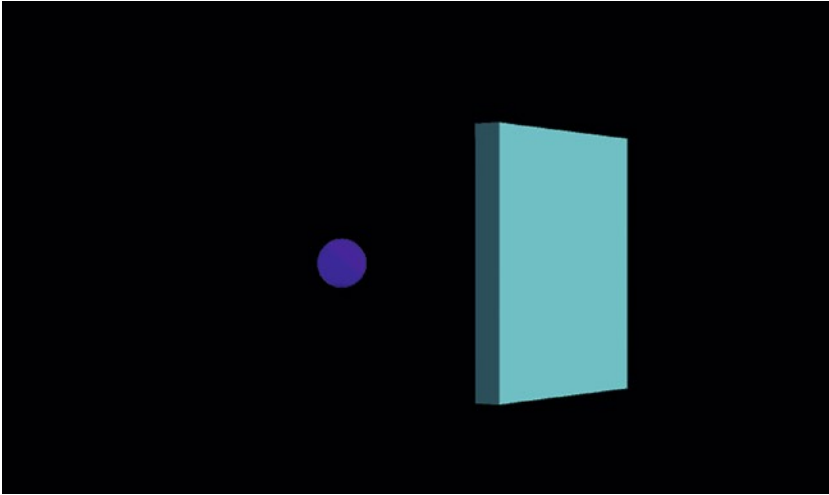


Figure 7-3. As you walk around the blue square, you'll notice that the source of the singing voice is a blue sphere. You'll hear the singing voice louder and clearer when the blue square is not obstructing the sound.

Try walking around to various parts of your room or area and try turning your head in various directions. Remarkably, you can hear the singing voices coming from the exact position of the sphere.

■ **Fun Experiment** While the app is running, try closing your eyes, walking around in a few circles, and then putting your fist exactly where you hear the audio source. Now open your eyes. Amazingly, your fist will be at the same exact spot as the sphere. This shows the amazing ability for the brain to spatialize sound, even without visual cues. It also shows the amazing ability for the HoloLens to digitally spatialize sound and sync the audio location to the hologram location.

Step 3: Understand the Scene

Now that you've had some time to try out spatial sound, let's dig a little deeper in the scene to understand all the components that make it work. The only two objects of interest in the Hierarchy are the Emitter object and the Occluder object, as shown in Figure 7-4.

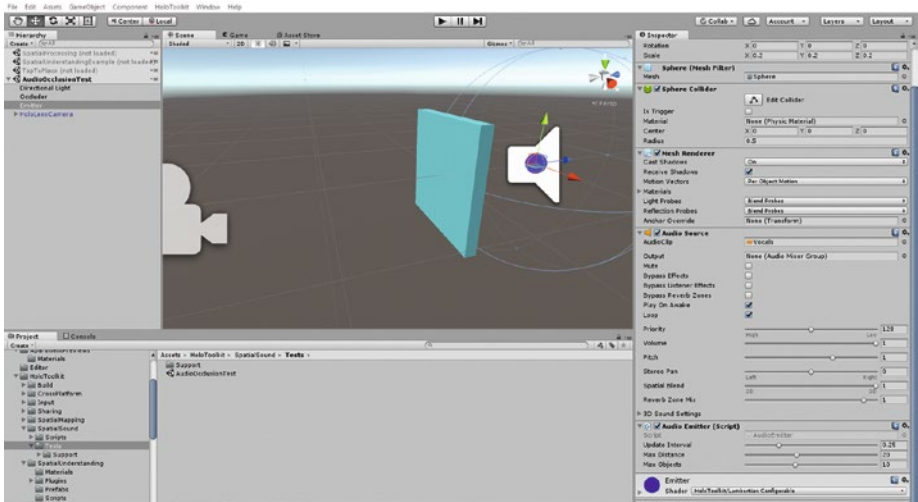


Figure 7-4. The *AudioOcclusionTest* scene has two objects that we focus on in this tutorial: the *Emitter* and the *Occluder*

First, let's look at the *Occluder* game object. After selecting the *Occluder*, you'll notice a script called *AudioOccluder.cs* in the Inspector panel, as shown in Figure 7-5. The *AudioOccluder.cs* script is a useful script provided in the HoloToolkit that allows objects to occlude spatialized audio sources.

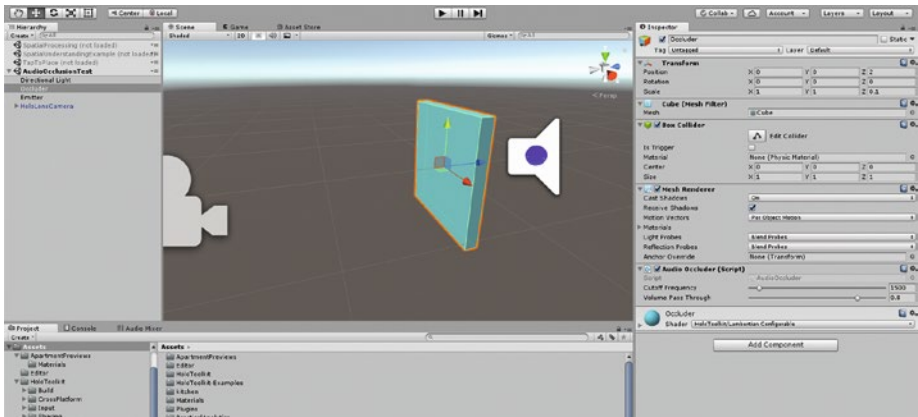


Figure 7-5. The *Occluder* object contains the *AudioOccluder.cs* script, allowing it to “occlude” any sound sources behind it

Let's see why audio occlusion is a useful feature of spatial sound. Think about a band playing music in a room. When you walk out of the room and close the door, you may still hear the band playing, but it will be muffled and a bit quieter. When you open the room's door again, the sound is loud and crisp. The `AudioOccluder.cs` script allows developers to mimic this behavior in Mixed Reality applications to increase realism for users.

When you attach the `AudioOccluder.cs` script to an object, the object will muffle and lower the volume on any spatialized audio sources, if the Occluder object is between you (the camera) and the audio sources or emitters. You can adjust a few of the script's parameters from the Inspector panel:

- The **Cutoff Frequency** parameter allows you to adjust the muffle of the occluded sound. This is essentially a low-pass filter.
- The **Volume Pass Through** parameter allows you to adjust how much volume to allow through the Occluder.

■ **Try It Out** If you're using holographic remoting in Unity, try to adjust the cutoff frequency and volume pass through in the Unity Editor to experience how the sound changes in the HoloLens.

The second key object in the hierarchy is the **Emitter**. The **Emitter** game object (the blue sphere) is the most important object in this scene, because the audio source is attached to this object and is where the sound is spatialized. After selecting this object in the Hierarchy, you'll notice a somewhat busy Inspector panel containing a few important components, as shown back in Figure 7-4.

The first vital component is the **audio source** component. When this component is attached to a game object, it causes it to behave as an audio source. It allows you to select the audio file, spatialize the audio source, adjust the volume, and add effects to your audio. Let's look a few key parameters (as shown in Figure 7-6) of the audio source that you can adjust in the Inspector panel:

- **AudioClip**: You may specify the audio file or asset—for example, a .wav or .mp3 file.
- **Mute**: Check this box to mute the audio. Useful for toggling within your project's scripts.
- **Spatialize**: Check this box to spatialize your audio.
- **Play On Awake**: Check this box to play the audio source when the scene loads.
- **Loop**: Check this box to loop the audio indefinitely .
- **Priority**: Allows you to set the priority of the audio file. A larger number means a lower priority, and a smaller number means a higher priority. If there are too many audio sources, then only sources with the highest priorities will be heard.

- *Volume*: Allows you to set the volume of your audio source.
- *Pitch*: Allows you to speed up or slow down your audio source.
- *Spatial Blend*: Allows you to set the degree to which your audio source is treated as a 3D spatial audio source. Set the value to 1 for spatial sound in the HoloLens.

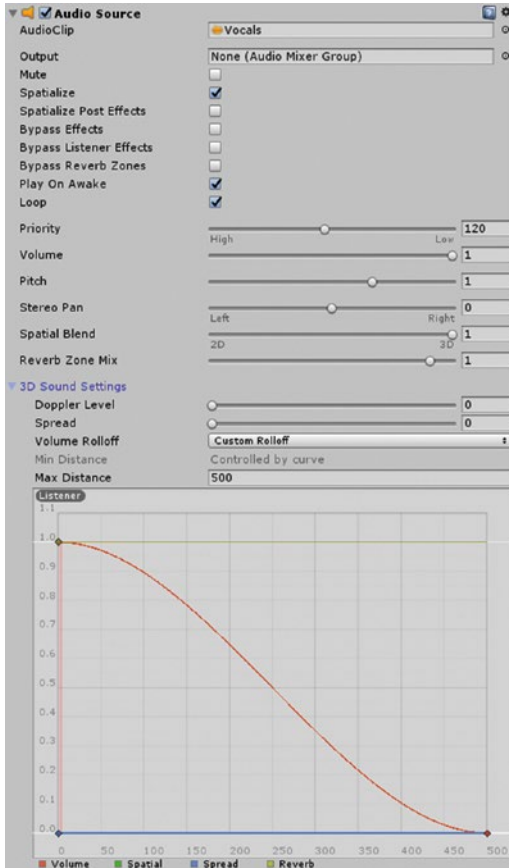


Figure 7-6. Adjustable parameters of the Audio Source component as shown in the Inspector panel of Unity's Editor

The second vital component attached to the Emitter game object is the `AudioEmitter.cs` script. This script allows the audio source to be influenced by other game objects in the scene. For example, the `Occluder` game object (containing the `AudioOccluder.cs` script) is able to influence this audio source because of the `AudioEmitter.cs` script. Let's look at some of the script's parameters (see Figure 7-7) that can be adjusted in the Inspector panel:

- *Update Interval*: The time, in seconds, between audio influence updates. To update each frame, set the value to 0. A longer time period provides better performance for your application, but also increases the time delay for activating the influencer.
- *Max Distance*: The maximum distance, in meters, for this object to look when finding the user or influencers.
- *Max Objects*: The maximum number of objects to consider when looking for influencers.



Figure 7-7. Adjustable parameters of the `AudioEmitter.cs` script, as seen in the Inspector panel of Unity's Editor

Step 4: Enable Spatial Sound in Your Application

Now that you've learned about some of the key elements of spatial sound and experienced a working example from the HoloToolkit, let's see how to implement spatial sound in your own application.

First, you need to enable spatial sound in Unity's settings. Go to **Edit ► Audio ► Spatializer** and select the Microsoft HRTF extension in the Spatializer Plugin drop-down, as shown in Figure 7-8. Set the System Sample Rate to 48000.

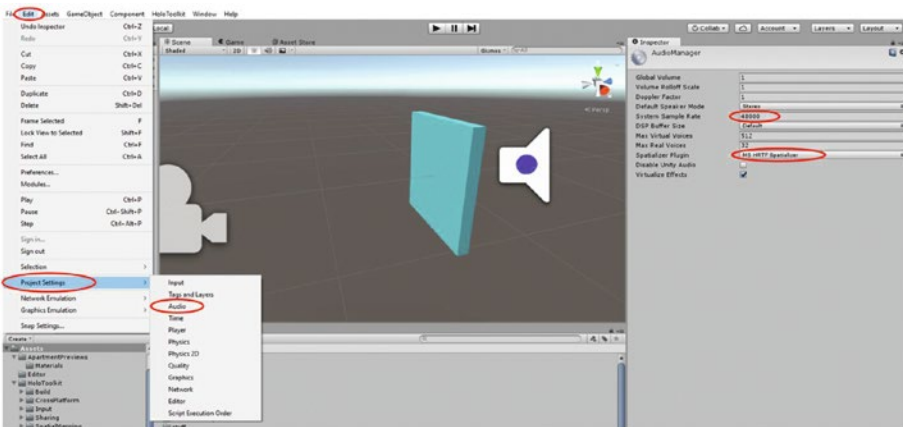


Figure 7-8. Enable spatial audio in Unity's Audio settings. Be sure to select the MS HRTF Spatializer and set the System Sample Rate to 48000.

Now you need to attach an audio source to any game object that you want to behave as an audio source. You can do this by selecting your game object, clicking the Add Component button at the bottom of your game object's Inspector panel, and searching for and attaching the audio source component, as shown in Figure 7-9. As you can see in that figure, I created a new Cube game object in the AudioOcclusionTest scene that we've been working with to illustrate creating spatial sound on a new game object.

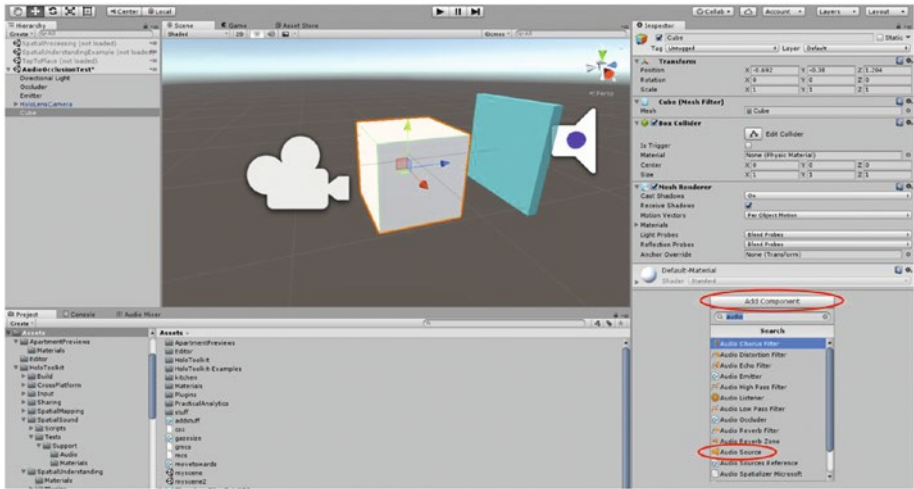


Figure 7-9. Attach an Audio Source component to the game object you want to behave as an audio source

Next, you need to configure the audio source for spatial sound. There are three parameters in the Audio Source component that you need to set, as shown in Figure 7-10. These are the changes you need to make:

- Enable the Spatialize checkbox
- Set the Spatial Blend to a value of 1
- Set Volume Rolloff to Custom Rolloff. You may need to expand the 3D Sound Settings item to see this parameter.

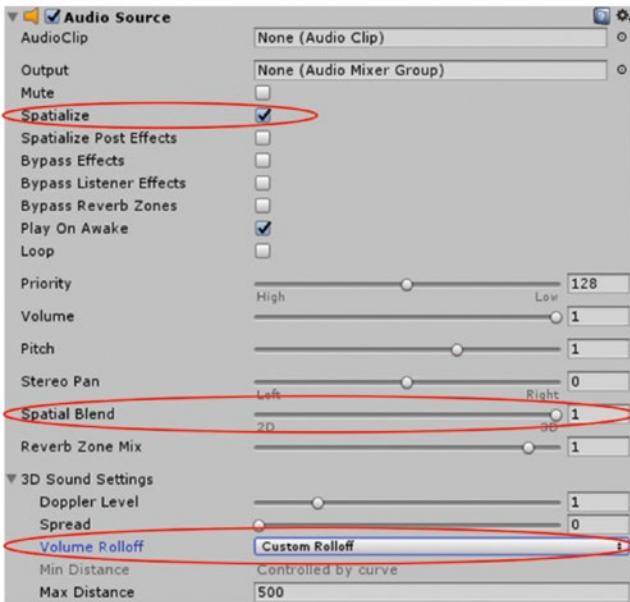


Figure 7-10. Modify the three parameters shown in the Audio Source component to spatialize your sound

That’s all it takes to enable spatial sound in your application. Feel free to drop an audio file from your assets into the AudioClip area and try it out with your headset.

To turn an object into an audio Occluder, simply attach the `AudioEmitter.cs` script to the game object that contains the audio source and attach the `AudioOccluder.cs` script to the object that you want to behave as the audio Occluder.

■ **Note** The audio listener component is also required to be attached to the camera for audio occlusion and spatial sound to work. By default, it’s included with the `HoloLensCamera` prefab in the `HoloToolkit`.

Spatial Sound Design Considerations

This section discusses some design considerations and best practices for using spatial sound in your application. I talk about when to use spatial sound and also things to avoid when using spatial sound.

When to Use Spatial Sound

Whenever possible, use spatial sound to help *guide users*. With such a small field of view, devices like the HoloLens can often frustrate users who are trying to find an object of interest. Although a visual arrow can be used to help a user find a hologram to look at, it's much better to leverage our instinctual ability to look in the direction of a sound we hear.

■ **Tip** When using spatial sound for guiding users or locating objects, use low- or mid-range audio frequencies. Have you ever tried locating a cricket by its chirp? It's extremely difficult because crickets chirp at a very high frequency. Our brains calculate the location of the sound based on how the sound waves arrive at each ear. Larger sound waves (lower frequencies) are easier to decipher than smaller sound waves (higher frequencies).

Increase the realism and immersion of a Mixed Reality experience by attaching appropriate spatial sounds whenever possible. I use the word *appropriate* because misuse of spatial sound can be annoying and jarring. Avoid loud, obnoxious noises. Add subtle audio effects to objects that collide, to buttons that are pressed, and to holograms that move. Think of spatial sound as a shadow. We don't really think about shadows on a regular basis, but when shadows are removed from real objects, a scene appears strange and "off." In the same way, the lack of appropriate audio effects in a virtual environment makes it seem inauthentic. The presence of spatial sound may go unnoticed, but like shadows, it's necessary to complete the experience. The intention behind spatial sound shouldn't be to call attention to the sound itself but rather to make the user comfortable and immersed in the experience.

Whenever possible, you should *spatialize all sound* in your Mixed Reality application. After all, in physical reality all sound is spatialized, meaning it comes from a source or series of sources. Mixed Reality applications should immerse users in a 3D experience—not just visually, but audibly too.

What to Avoid When Using Spatial Sound

As with digital holograms, you can do things with spatial sound in Unity that defy the laws of physics. Sometimes these effects can add a special edge to your application. But if not carefully tested and considered, they may create an unnatural or even uncomfortable experience for users.

Invisible audio sources or Emitters should rarely be used, if ever. When attaching an audio source to an invisible object, our sense of hearing can precisely locate the source of the audio. But when the user looks at the location where the sound is coming from and doesn't see any visible objects, it can be an unnerving experience.

Don't mix too many sounds together and *avoid overpowering spatial sounds with 2D sounds*. As discussed early in this chapter, devices use subtle modifications of sound waves to achieve the spatial sound effect. When these are masked or drowned out by ambient noises, such as background music, the spatial sound experience can be

degraded. When mixing multiple sound sources is needed, be sure the spatial sounds are louder than any ambient sounds.

Try to minimize the use synthesized or artificial sounds. When a sound is unnatural, users may not have a strong intuition on the source of the sound. Better to use natural sounds like the chirp of a bird, the click of a button, the voice of a person, and other recorded sounds. *Take advantage of human intuition or expectation* when designing your spatial sound experience. For example, a human voice is likely to be found at a human height range, so use a voice to guide the user to look at something that is approximately at eye level. Use the sound of rustling tree leaves or birds for objects above the user.

Summary

Congratulations! In this chapter, you learned how spatial sound in Mixed Reality development works. We walked through a working spatial sound example included in the HoloToolkit, learned how to implement spatial sound in your application, and talked about some best practices for designing spatial sound into your application. You're now equipped with the tools needed to start implementing a great spatial audio experience into your application.

When developing an application, it's easy for sound design to be deprioritized or forgotten altogether. However, I can't express enough the importance of good sound design when developing for Mixed Reality. I still remember a strong piece of advice I received from a professor during a course I took on video editing:

People will forgive bad video if there's good audio. People will not forgive bad audio, even if the video is excellent. —Fred Metzger

Sound design is extremely important and should be considered as part of your application design from the very beginning—not as an afterthought.

PART III



Growing as a Holographic Developer



Awe-Inspiring Experiences

This chapter goes over some tips and tricks on creating awe-inspiring Mixed Reality experiences. By now, you've learned all the basics you need to make compelling applications. Dozens of books could be written on advanced Mixed Reality development and techniques—and even more Mixed Reality techniques are yet to be discovered. In this chapter I give you a taste of some things you can do to enhance your applications and experiences by introducing you to some design concepts, HoloLens project samples, and some third-party tools you can use.

What Makes an App Awe-Inspiring?

There's a certain feeling of awe and magic when you experience a well-made application. The Microsoft published experiences do a great job of showcasing what's possible with the HoloLens. Applications like Fragments, RoboRaid, HoloStudio, HoloTours, Actiongram, and Galaxy Explorer provide an unforgettable experience for users. These apps run smoothly, take full advantage of spatial sound and spatial mapping, and make excellent use of shaders and colors.

Here are some key features of an awe-inspiring experience:

- An awe-inspiring app allows the user to forget about the headset and feel immersed in the experience. It fully takes advantage of spatial sound and spatial mapping so that holograms feel like they are really in the user's environment.
- It provides stunning visuals, as opposed to boring or plain visuals. It uses transparency, light, colors, and motion appropriately.
- It runs smoothly and maximizes frame rate. A choppy frame rate causes users to be disoriented and the experience to feel artificial. A smooth frame rate (60 frames per second) makes the application feel real and responsive.
- It provides an element of magic, or the ability to do something the user didn't think was possible. A few examples of this are using image recognition to identify objects in the user's scene, using spatial understanding to allow digital characters to sit on real sofas and chairs, and being able to use voice commands in unexpected ways.

Though Microsoft and a handful of Mixed Reality studios have published what I would consider awe-inspiring experiences, I want to reiterate one of the big assumptions of this book, which is that we're all still very early in determining best practices for Mixed Reality. There's still so much left to discover. The few tips and tricks I highlight in this chapter shouldn't be considered the best achievable but rather the minimum starting point from which to improve.

Many Mixed Reality developers (myself included) obtain inspiration from movies with a lot of special effects, such as science fiction movies. When watching these movies, you'll find many great ideas for what makes a hologram look amazing. You'll see colors and textures that elicit awe, excellent examples of fluid animations and transitions, and sound effects that work well.

Optimization and Performance

In this section, I walk you through best practices for optimizing your Mixed Reality application. The key metric in optimization is frame rate, or *frames per second* (FPS). Other crucial factors include CPU usage and impact on battery life, but it's the frame rate of your application that directly impacts how the user experiences your application. Frame rate can make the difference between a choppy, frustrating experience and smooth, wonderful one.

Optimizing for performance is one of those necessary but annoying aspects of development. When asked about my experiences developing for the HoloLens, I often tell people this:

I can make just about anything I want in Mixed Reality. That's the easy part. What's hard is optimizing for the best performance.

I've talked to countless HoloLens and Mixed Reality developers, and all developers, both the experienced and beginners, suffer with the pain of optimization. That's because all devices—whether it's a HoloLens, immersive headset, or powerful gaming PC—have limits to the amount of content they can render before running into hardware and performance limitations. As developers, it's easy to want to include high-detail 3D models, extravagant colors, extremely realistic lighting effects, and complex animations. But all these additions take processing power to render, and when you have too much content in your experience, your device responds with a drop in frame rate.

In essence, optimization of your application is a form of art. You need to learn the perfect balance between content and performance. Maximizing the look and feel of your content while maintaining 60 FPS is the ultimate goal of performance optimization in any Mixed Reality experience.

Although you should always strive to achieve 60 FPS, you may notice that lower frame rates (30–60 FPS) may be acceptable, depending on the use case. When you're out of optimization options, you may need to make a difficult decision to include certain features or objects at the expense of frame rate.

Note Because the HoloLens is a self-contained holographic computer (not tethered to a more powerful PC), it has more performance constraints than tethered Mixed Reality headsets. Because of this, the performance optimization conversation in this section will be focused on the HoloLens. Performance will be less of an issue when using tethered headsets connected to a powerful gaming PC. That said, all performance suggestions given can help when optimizing any experience, both small and large. Also, when developing for tethered headsets, it's important to consider individuals who may be running your application on a less powerful PC. I recommend testing on a minimum-specification PC for performance monitoring when developing for tethered Mixed Reality headsets.

How to Monitor for Performance

Before we dive into best practices, you should first know how to measure the performance of your applications. The best place to see all performance metrics of the HoloLens is through the Windows Device Portal. To access the Windows Device Portal, make sure your headset is turned on and either connected to the same WiFi network as your PC or connected via USB cable.

Ensure that you have already set up Developer mode on your device (as discussed in Chapter 3) and that the Device Portal option is enabled. You can find the Device Portal setting in your HoloLens at Settings ► Update and Security ► For Developers ► Device Portal.

If your device is connected to your PC via the included USB cable, you may type the address 127.0.0.1:10080 into your web browser. If you're using WiFi, you may type in the IP address of your headset. If you don't know your IP address, you can access it on your HoloLens by going to Settings ► Network & Internet ► Advanced Options. See Figure 8-1 for guidance on where to find the IP address in your HoloLens's Settings app.

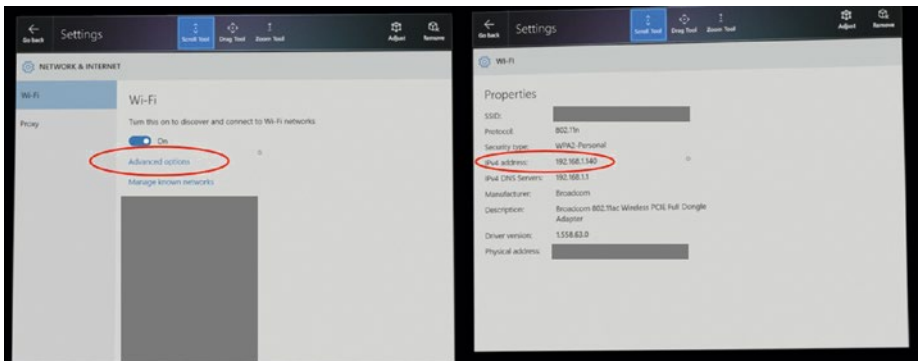


Figure 8-1. Where to find your HoloLens's IP address in the Settings app

Once you log in to the Windows Device Portal, navigate to System Performance in the menu bar on the left. You'll see real-time performance tracking and metrics, as shown in Figure 8-2.



Figure 8-2. You can track real-time performance metrics for the HoloLens via the Windows Device Portal

When within the Device Portal, you'll see several metrics:

- **System Power:** This is the total power consumption of the HoloLens. Monitor this to ensure that power consumption is kept within the green and orange areas. Be sure to measure system power when the device isn't charging, because it won't measure while charging.
- **SoC Power:** The combined power consumption of only the CPU, GPU, and memory unit. As with System Power, be sure SoC power is kept within the green and orange areas.
- **Frame Rate:** The rendered frames per second of your 3D application. If you're developing a 2D application that will run within the Windows shell, the frame rate will be the frame rate of the shell, not your 2D application.
- **CPU:** Shows the load of your central processing unit.
- **GPU:** Shows the load of your graphics processing unit. It indicates the percentage of time that the GPU is active with tasks.
- **I/O:** Shows the disk usage for all processes.

- *Network*: Shows the network/WiFi usage for all processes.
- *Memory*: Shows the total memory committed by all processes on your device. To see memory usage for an individual process, see the Processes tab in the Windows Device Portal. *Note that if your application exceeds 900 MB of memory, it will be terminated.*

Although maintaining a high FPS is key, considering the preceding metrics when optimizing your application is also important. If you maintain 60 FPS, but your application is very demanding on these other metrics, your device may overheat, which will cause your application to be terminated. Here, in a nutshell, are the essential performance targets for the HoloLens:

- Maintain frame rate at 60 FPS
- Keep power consumption within orange and green areas
- Keep memory usage of your application under 900 MB

Another excellent tool for monitoring performance of your application during development is the FPSDisplay prefab included in the HoloToolkit. Search or browse for this prefab and drag it into your Hierarchy to have an always-on display of your 60-frame average FPS. Be sure to remove the prefab before publishing your app!

Best Practices for Performance

As mentioned earlier in this section, performance optimization is a big challenge, even for experienced developers. The guidelines provided here will help when optimizing, but you should be prepared to spend a good chunk of time on optimizing and troubleshooting performance issues.

Start Monitoring and Optimizing Early

As a rule of thumb, you should consider performance and optimization *very early* in your development cycle. If you start too late, you may have a difficult time identifying objects in your scene that are causing performance issues. If you start out with a smooth application at the beginning, you'll be able easily identify when additions to your project start to impact performance. You'll only be able to accurately test performance when deployed to your device. Playing your device in the Unity Editor won't give you many insights regarding the actual frame rate and hardware usage when your application is deployed to your headset. This might feel like you're spending more time deploying and testing up front, but it will save you time troubleshooting performance issues later in your development cycle.

Optimize Polygons and Textures

All 3D objects rendered by the HoloLens are made up of triangles or polygons, including the spatial mapping mesh. Although polygon count isn't the only factor affecting performance on the HoloLens, it's one of the easiest ways to begin optimizing your experience.

Models with many polygons are typically referred to as *high-poly*, whereas models with fewer polygons are *low-poly*. There's no strict definition for the number of polygons that constitute high- or low-poly models—it depends on the size and complexity of the model. In general, high-poly models appear smooth and curved, whereas low-poly models appear somewhat blocky.

High-poly 3D models are sure to slow down your frame rate. Low-poly models, when combined with other best practices, will allow your experience to perform well. In general, the HoloLens can render up to about 80,000 triangles while maintaining 60 FPS if optimal shaders are used. Frame rate drops to about 30 FPS with 200,000 triangles. In most cases, you'll want to stay safely in the 20,000–60,000 triangle range to give yourself some room to work with.

Figure 8-3 shows a few different views of a 3D model of an apple. The apple on the left is high-poly, and the apple on the right is low-poly. You can see that the high-poly apple looks much smoother and more realistic than the low-poly model. In the middle row of Figure 8-3, I turned on the wireframe so you can visualize the individual triangles that comprise each 3D model. In the top row, the high-poly apple uses the standard shader, allowing it to look shiny, but also impacting performance negatively. The low-poly apple in the top row uses the legacy diffuse shader, making it look dull but allowing for much better performance. In the bottom row, I swap the shaders between the two apples to illustrate that shaders can make low-poly models look better and vice versa.

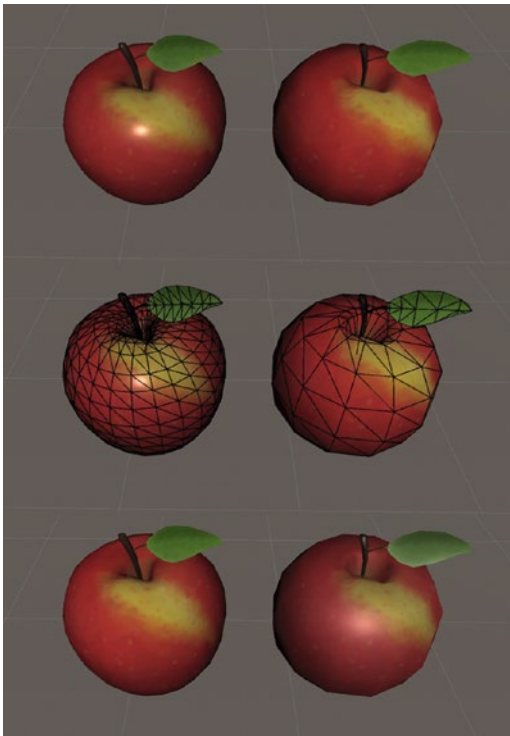


Figure 8-3. Showing how polygon count and shader can impact visual appearance of 3D models

As you can see, optimizing your HoloLens application sometimes means making difficult tradeoffs between visually appealing models and fast performance. There are tricks to making low-poly models look stunning, even without fancy shaders. 3D artists can color model textures to give them an illusion of appearing shiny or extremely detailed. For example, in Figure 8-4, the sofa appears to have a high level of detail, with folds in the cushion and light reflections at the edges. But the sofa model is actually a low-poly model, as shown in the lower image. The cushion folds and reflections came from the texture image applied to the 3D model. I was able to achieve both high performance and a stunning visual of a sofa in the HoloLens because I was able to minimize my polygon count and use a fast shader.



Figure 8-4. Low-poly models can be given the appearance of looking detailed with some creativity in the texture image applied

As much as possible, you should strive to use one texture per model, as opposed to creating a model comprised of many texture files. You should also minimize shader switching. Try to use the same type of shader for all objects in view.

Use Level of Detail Rendering

Level of detail (LOD) rendering is a performance technique whereby you reduce the polygon count and appearance of objects that are further away from view. It's a waste of computing power to render a high-quality model that's far from the user. Because the model will appear small when far away and the user won't be able to appreciate the model's detail. Unity provides a component called LOD group that allows you to change how an object is rendered at various distances. Figure 8-5 shows what this component looks like in Unity. You may add various levels of detail by right-clicking the colored bars, dragging each bar to specify distance from the camera, and adding models of varying polygon counts and shaders for each LOD. For more information, see <https://docs.unity3d.com/Manual/class-LODGroup.html>.

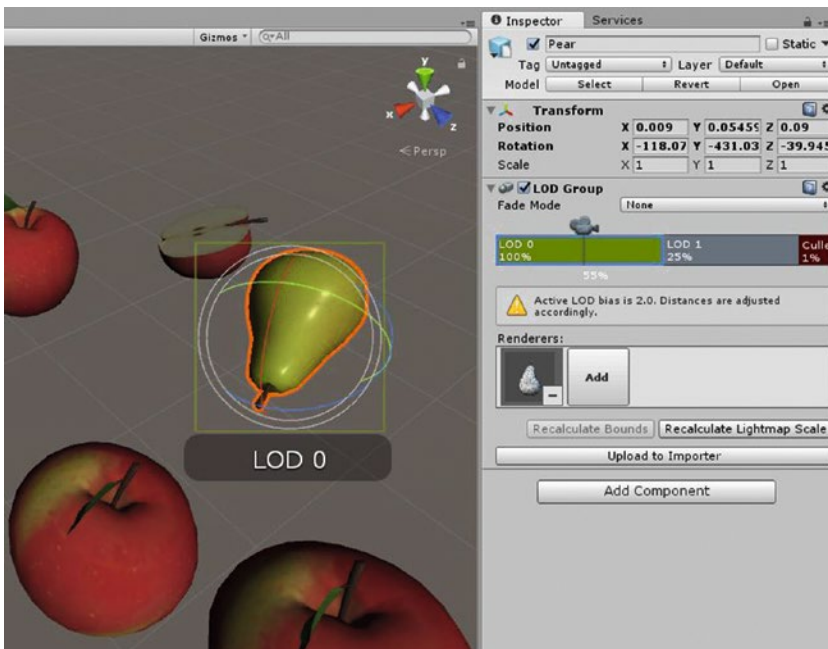


Figure 8-5. Unity's LOD group component allows you to specify how your model will be rendered based on distance from the camera. You can render a lower-quality model when the camera is far from the object.

Use Culling

Another effective performance strategy is called *culling*, which means to not render models that won't be seen. By default, Unity uses *frustum culling*, which means that anything outside of the user's view, or *frustum*, isn't rendered. Figure 8-6 illustrates the camera's frustum, shown with the white lines. Avoiding rendering objects outside of the camera's view limits what needs to be drawn and increases performance.

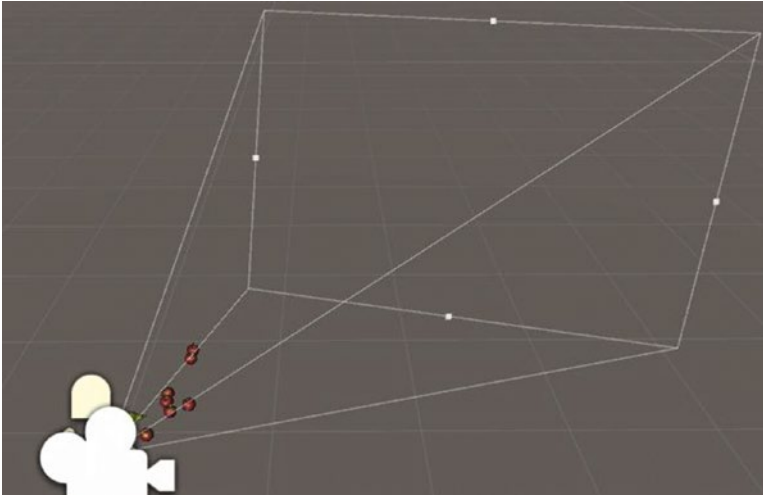


Figure 8-6. Frustum culling ensures that anything outside of the camera's frustum (shown in white here) isn't rendered

Another type of culling is called *occlusion culling*, which allows you to not render objects that are being obstructed by other objects in your scene, thereby reducing the load on your device and increasing performance. To enable occlusion culling, go to Window ► Occlusion Culling and click the Bake button in the window, as shown in Figure 8-7. You'll also need to ensure that the Occlusion Culling setting is enabled in your camera. For more on occlusion culling, see: <https://docs.unity3d.com/Manual/OcclusionCulling.html>.

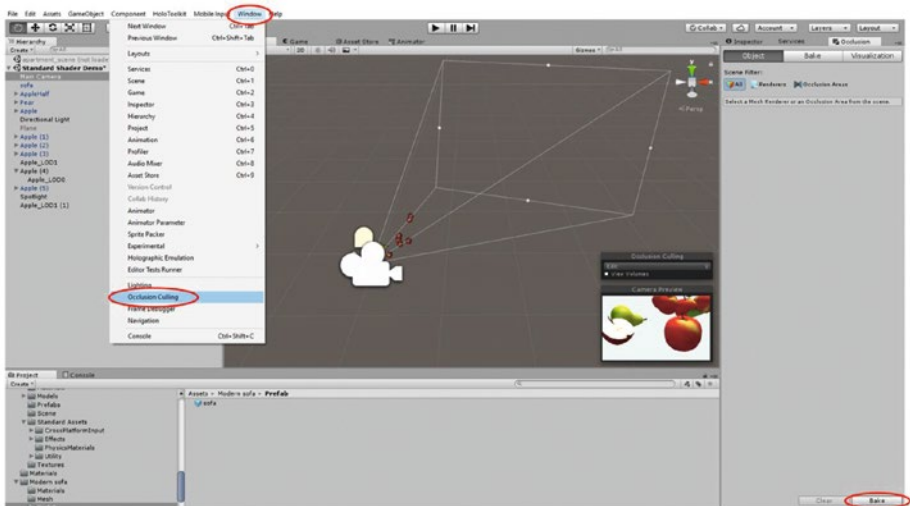


Figure 8-7. You may enable occlusion culling by going to Window ► Occlusion Culling. Occlusion culling increases performance by preventing obstructed objects from being rendered.

Enable Single-Pass Instanced Rendering

Hidden in your player settings (Edit ► Project Settings ► Player ► Other Settings), you'll find an option to change your stereo rendering method between Multi Pass and Single Pass Instanced. This option significantly reduces CPU processing time by rendering both eyes at the same time instead of one at a time. Figure 8-8 shows where you can find the stereo render setting. For more on Single Pass Instanced rendering, see <https://docs.unity3d.com/Manual/SinglePassStereoRendering.html>.

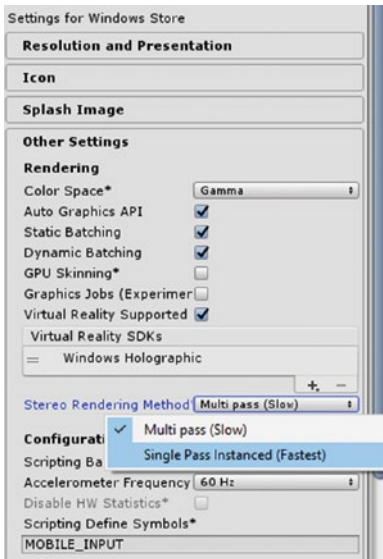


Figure 8-8. You can switch between stereo rendering modes in your player settings

Warning Not all shaders are compatible with Single Pass Instanced rendering. If you're using shaders outside of those included with the HoloToolkit, keep an eye out for differing views between your left and right eyes. If this happens, you may need to revert to Multi Pass rendering or update your shaders as described at <https://docs.unity3d.com/Manual/SinglePassStereoRendering.html>.

Optimize Shaders

Shaders are scripts or programs that perform calculations responsible for producing light levels, darkness, color, reflectivity, and special effects for objects. Shaders are an extremely important factor for how your object appears, but they can significantly contribute to performance issues in your application, arguably more so than your object's polygon count.

For best performance, several shaders are included with the HoloToolkit that are meant to focus on increased speed. Some of these shaders are FastConfigurable, LambertianConfigurable, StandardFast, VertexLitConfigurable, and more. Check the HoloToolkit documentation on GitHub for a complete description for each shader included in the HoloToolkit at <https://github.com/Microsoft/HoloToolkit-Unity/tree/master/Assets/HoloToolkit/Utilities>.

Typically, any special object visualization you would like to achieve in your experience will require a shader that decreases performance. Examples of special object visualizations include highly reflective metals or mirrored surfaces, glass-like or transparent objects, and glowing or neon-like effects on objects.

The topic of shaders and shader optimization is a big one in the 3D modeling and graphics industries. Many books have been written on the topic, and advanced developers will inevitably need to write custom shaders to perform specific visualizations while maintaining good performance. I won't go into the details of writing your own shaders in this book. For beginners, I recommend leveraging the various shaders offered in the HoloToolkit. For advanced tips and guidelines when working with or writing your own shaders, Microsoft's Mixed Reality documentation provides many good shader suggestions at https://developer.microsoft.com/en-us/windows/mixed-reality/performance_recommendations_for_hololens_apps.

Simplygon

What if you have a 3D model you'd like to use but it has too many polygons and contains large texture files? Simplygon is a tool (acquired by Microsoft in 2016) that allows users to reduce the number of polygons and reduce the complexity of textures of a 3D model without noticeably impacting the visual appearance of the model. Figure 8-9 shows an example of the Simplygon user interface. Simplygon also provides a plugin for Unity that allows you to optimize models from within Unity. To sign up for an account and download Simplygon for free, visit <https://account.simplygon.com/#/downloads>.



Figure 8-9. Simplygon allows you to convert large, complex 3D models into beautiful, low-poly models that performs well in the HoloLens

Holographic Remoting

Holographic remoting allows experiences to be streamed from a PC to your HoloLens. By now, you should be familiar with using holographic remoting when testing your apps using Unity's holographic emulation. In the same way, you can develop applications that run on your PC and stream to the HoloLens. In essence, this allows you to overcome the hardware and processing power limitations of the HoloLens and leverage the powerful graphics and processing capabilities of a good PC. Doing so lets you render in much greater detail, use beautiful shaders, and render many more objects than you would be able to with the HoloLens.

If you want to deliver a powerful desktop PC-level experience to the HoloLens, I recommend looking into adding holographic remoting capabilities to your application. For additional details on getting started, see https://developer.microsoft.com/en-us/windows/mixed-reality/add_holographic_remoting.

Stabilization Plane

The HoloLens performs special hardware-assisted stabilization to keep Holograms appearing stable in your environment. Because the HoloLens can't stabilize all holograms in your scene, an invisible plane in your scene called the *stabilization plane* is used to select objects that receive the maximum amount of stabilization. Figure 8-10 shows the stabilization plane with a white outline and also shows the script (stabilizationplanemodifier.cs) that manages this plane in the Inspector panel. By default, the stabilization plane will attempt to find appropriate holograms without requiring additional setup. But for best results, you can help the plane find appropriate objects to stabilize within your script and also within the stabilizationplanemodifier.cs script.

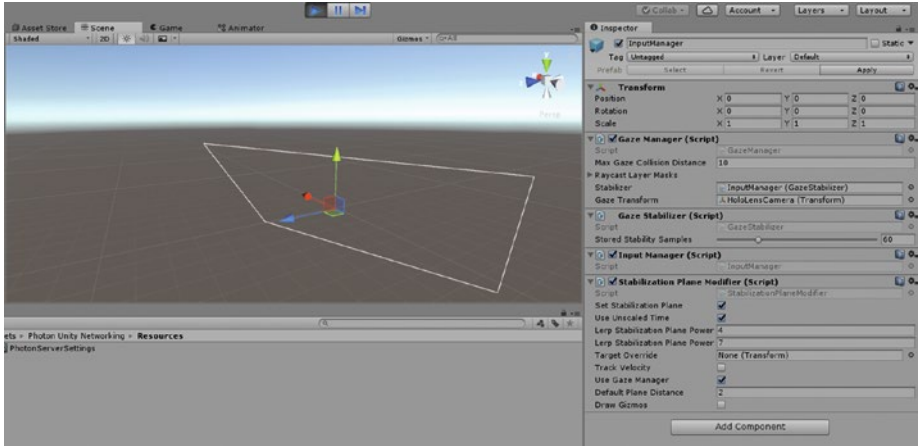


Figure 8-10. The stabilization plane helps the HoloLens prioritize which objects receive the greatest amount of stabilization for a better experience

The stabilization plane settings are included in the InputManager prefab of the HoloToolkit, as shown in Figure 8-10. The `stabilizationplanemodifier.cs` script is useful for adjusting the behavior of the stabilization plane. Here is a brief description of each setting in the script:

- *Set Stabilization Plane:* Allows the stabilization plane to be created and placed automatically, based on your gaze, a fixed distance, or a target that you provide.
- *Use Unscaled Time:* Modifies the way the plane moves by making it independent of frame rate, pausing, or other time manipulations.
- *Lerp Stabilization Plane Power Closer:* The speed at which the plane moves towards the camera.
- *Lerp Stabilization Plane Power Farther:* The speed at which the plane moves away from the camera.
- *Target Override:* Allows you to override the location of the stabilization plane. Use this when you want to stabilize specific holograms in your scene.
- *Track Velocity:* Keeps track of the velocity of your target object so that the plane can accurately follow and anticipate the target object's movement rather than react to it.
- *Use Gaze Manager:* Allows your plane to be set based on objects you gaze at. If this option isn't enabled, the plane will float at a fixed distance in front of the camera.

- *Default Plane Distance:* The distance from the camera that the plan will float if no target object is specified or identified.
- *Draw Gizmos:* Allows you to visualize the stabilization plane when testing or running your application.

To manually set the stabilization plane on a target object via scripting, you can use the `SetFocusPointForFrame()` function. Here's an example:

```
public GameObject objectToFocusOn;
void Update()
{
    var normal = -Camera.main.transform.forward;
    var position = focusedObject.transform.position;

    UnityEngine.VR.WSA.HolographicSettings.SetFocusPointForFrame(position, normal);
}
```

In the preceding code, you provide a normal and a point to define the plane. The normal is the direction in which your plane faces. The point (`position`) is the position of your target object (`objectToFocusOn`). Feeding these variables into the `SetFocusPointForFrame()` function allows the stabilization plane to be placed at your desired location.

Here are some general tips to get the most out of your stabilization plane:

- Try to use the plane to cover as much of your content as possible. For example, if your content is a plane (2D image, text, UI plane, or other flat surface), align the stabilization plane to your 2D surface.
- Do your best to have your plane cut through all relevant holograms in your scene, to stabilize as many objects as possible.
- Place the stabilization plane on objects that are further away from the camera, as those tend to be more unstable.
- Never place the plane outside your camera's view. There's no point wasting precious stabilization resources on objects that you can't see. Stabilize objects that the user will be looking at.
- Don't let the stabilization plane touch or cut through the user.

For more information and best practices for the stabilization plane, feel free to review the following resources:

- https://developer.microsoft.com/en-us/windows/mixed-reality/Hologram_stability.html
- https://developer.microsoft.com/en-us/windows/mixed-reality/focus_point_in_unity
- https://developer.microsoft.com/en-us/windows/mixed-reality/case_study_-_using_the_stabilization_plane_to_reduce_holographic_turbulence

Design and Magic

In this section, I focus on design elements, tools, and best practices that will help your application stand out visually and provide a good user experience. There are many ways in which developers can add 3D objects and holograms to a scene. However, when visuals feel dull or cheesy, it's a lost opportunity for a truly awe-inspiring experience. Incorporating good design and magical experiences into your application takes more thought and effort. In the long run, though, you'll have a Mixed Reality experience that you can be proud of and perhaps provide users with an excuse to keep coming back to your app.

To get started, I provide you with some best practices for Mixed Reality design, followed by some design-related tools and resources you can use. I also provide a brief tutorial on using Vuforia and talk about ways to incorporate some magic into your experiences.

Best Practices for Design

In the short period of time that the HoloLens and other Mixed Reality devices have been around, a few key best practices for design have emerged. Although there's still so much more to be learned by developers (that's you!), these best practices will help your design process in the short term.

Spatial Mapping

As much as possible, consider the use of spatial mapping in your application. It's easy to forego spatial mapping. You may think it's not worth the effort or adds little value to your project. I am also guilty of wondering why I should add spatial mapping when it would only limit where my holograms can go. However, a big benefit of devices like the HoloLens is its powerful ability to interact with the physical world. Take advantage of that by allowing your objects to be able to rest on floors and surfaces, be pinned to walls and ceilings, and be occluded by real objects in your scene. It's an easy way to add magic to your experience without trying to invent something new.

The HoloToolkit provides a very simple shader for visualizing your spatial mapping mesh. As shown in Figure 8-11, the default spatial mapping shader allow meshes to have white edges with transparent surfaces. It's functional but doesn't look very appealing. The HoloToolkit shader used with the spatial understanding mesh is slightly better, with a green tint, as shown in Figure 8-12.

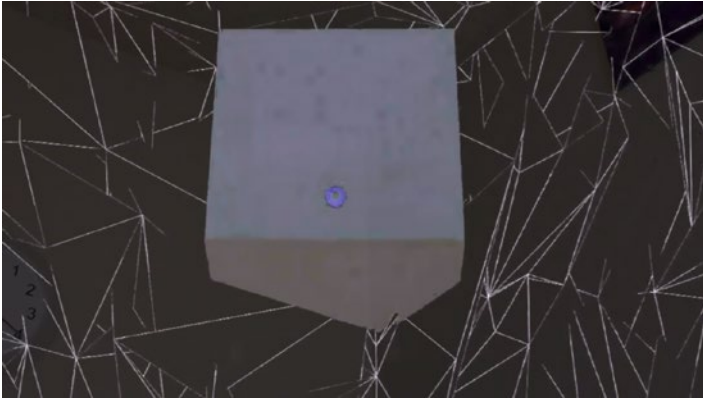


Figure 8-11. The default shader for visualizing the spatial mapping mesh is functional but not very pretty



Figure 8-12. Smoothing out the mesh and tinting the lines green, as is done with the spatial understanding mesh, improves the experience

Scanning your environment using spatial mapping is a technologically magical feat. As such, visualization of the mesh should be equally impressive. Microsoft-published applications like Fragments or Young Conker use vibrant colors and lighting effects to really breathe life into the spatial mapping mesh. Figure 8-13 shows how the mapping mesh builds out during the scanning phase of the Fragments app, using beautiful animated, sparkling, and dynamic squares. The specific shader used in Fragments and Young Conker are not available to developers as of this writing, but you can download similar shaders for use in your application. One of my favorite examples can be found at <http://smeenk.com/hololens-scanning-effect-in-unity/>, complete with a working sample you can download at GitHub.

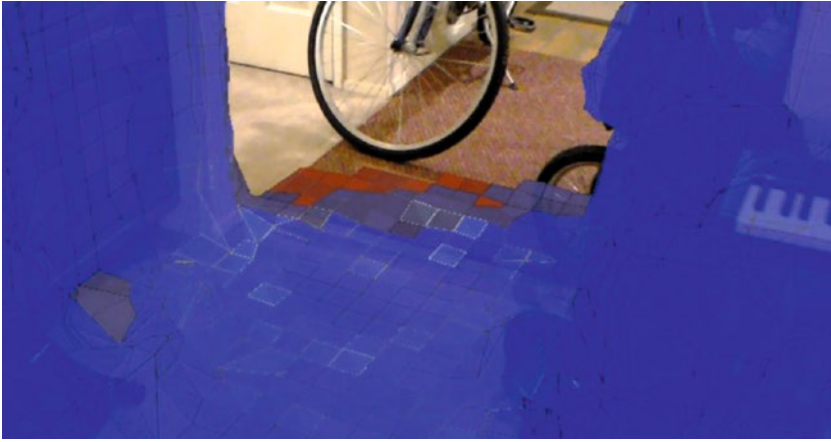


Figure 8-13. Adding regularity, color, texture, and animations to the spatial mapping mesh significantly boosts design

Distance from User

Microsoft recommends that optimal placement of holograms is between 1.25 m and 5 m, with 2 m being the best place. Holograms placed too close to users may result in discomfort or a jarring experience. By default, the HoloLens camera prefab in the HoloToolkit culls any holograms that are closer than 0.85 m (using frustum culling.) This is why you often can't view details when you approach a hologram too closely.

In my experience, reducing frustum culling from 0.85 m to 0.20 m is acceptable, depending on the use case of your application. This setting allows users to view intricate details close up without the model frustratingly disappearing. This value can be set by adjusting the Near Clipping plane in the Camera game object.

Shadows

Our brains heavily rely on shadows as a cue for where objects are in the world around us. Shadows are also an important part of 3D design. As such, the use of shadows can help increase realism and provide a greater sense that digital holograms are anchored in the real world.

Though Unity can easily create shadows in your scene, devices like the HoloLens are unable to display shadows because they can only add light to the display and cannot darken an area (displays that work in this manner are referred to as *additive displays*). The trick to overcoming this limitation is to use a technique called negative shadowing. *Negative shadowing* adds a small amount of light or glow around your object and then removes the light where the shadow would normally appear. The user won't perceive the light glow, but will perceive the shadow.

Voice

Whenever possible, add voice commands to your application. This is especially true for UI elements. If your application has a button that says Start, be sure to add a voice command for “start.” A best practice is to add a small microphone icon on or near your button to cue users that the button is voice-enabled. For bonus design points, you can have the icon of the microphone appear after the user has gazed on the object for a certain period of time (0.5–1 second should be sufficient).

Sharp Text

As of this writing, it’s very common to see blurry text in many HoloLens applications, especially those in the Windows Store. Using Unity’s default fonts causes letters to appear blurry in the HoloLens due to scaling. Figure 8-14 illustrates Unity’s default text (the upper *Testing*) and text written using the HoloToolkit’s UITextPrefab prefab (the lower *Testing*).

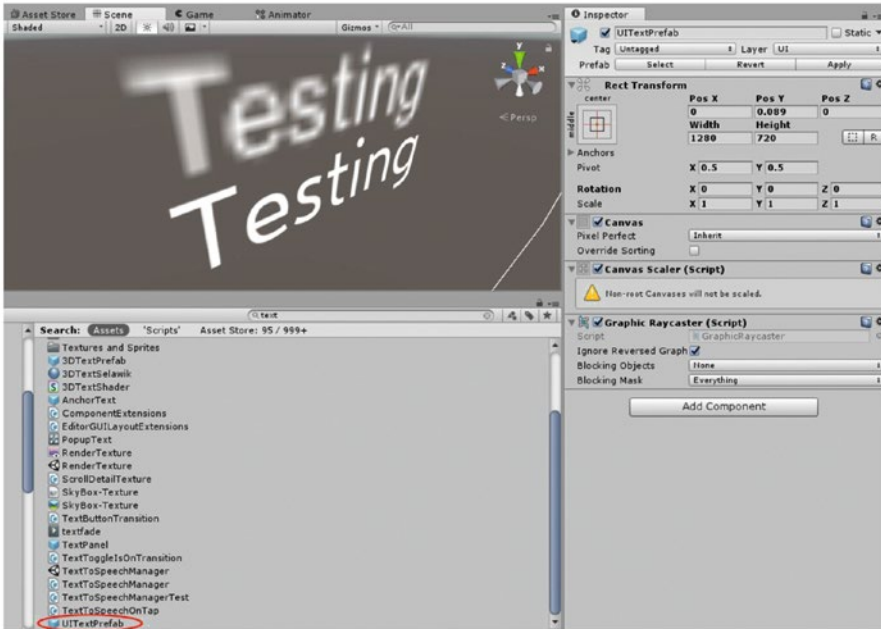


Figure 8-14. Use typography resources included with the HoloToolkit such as the *UITextPrefab* to achieve sharp-looking text in the HoloLens. Default Unity text resources appear blurry and unappealing.

As a rule of thumb, text also looks best in the HoloLens when the font is white against a colored or darker (but not black) backdrop. Avoid outlining your text.

Bounding Box

When users have the option of manipulating (moving, rotating, or scaling) a hologram, be sure to add a bounding box with “handles” that users can use for manipulation. The bounding box should appear invisible when the hologram isn’t selected, but appear when the hologram is selected.

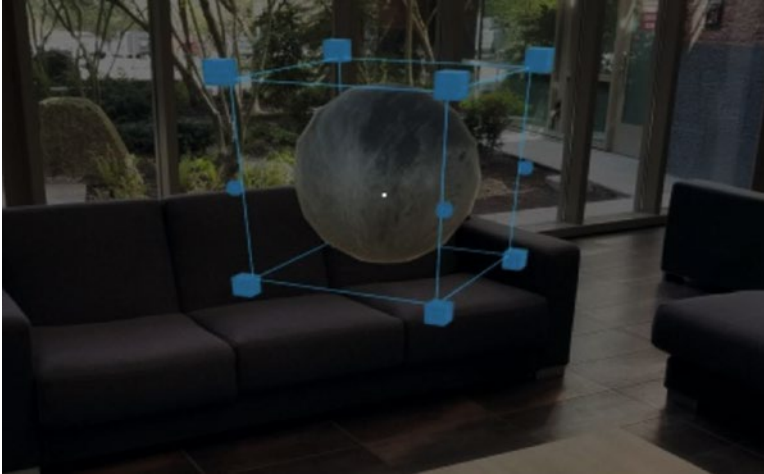


Figure 8-15. Bounding box around a selected hologram to assist with manipulation

Toolbars

I recommend using toolbars and app bars in Mixed Reality applications. Toolbars with minimalistic buttons evoke a feeling of cleanliness and make your experience feel modern. Figure 8-16 illustrates what a well-designed toolbar looks like. Each button should respond to being gazed at, such as the highlighted button shown in Figure 8-16.

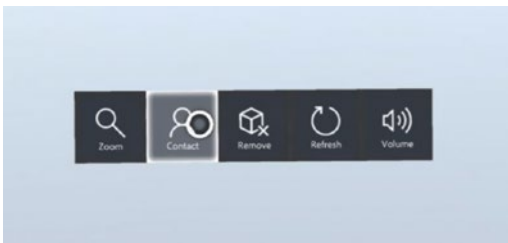


Figure 8-16. Use clean, modern-looking toolbars for your menu items

Colors

You may be tempted to make your application or experience shine vividly with all the colors that the HoloLens can render, but a great design strategy is to use only one color on any object that you want to appear as a traditional hologram. From sci-fi movies, we've come to expect holograms to have a certain monochromatic glow. Rendering an object with a single blue or green color evokes a special feeling and makes the hologram appear real.

Use only one or two colors as a theme for your application. You can see by the menu item in Figure 8-16 that the simple color choices result in a UI that appears modern.

Design Experiences to Avoid

Never head-lock content. Though heads-up display (HUD) UI may sound fun, it doesn't feel good in devices like the HoloLens. If an information panel or object needs to follow the user's gaze, be sure to give it a slight delay, so it appears to chase and catch up to your gaze. The `tagalong.cs` script included with the HoloLens already provides the delayed follow functionality.

Avoid using loud sound effects, especially when they're triggered by gaze. This may result in an annoying experience for users. And avoid making the user type information unless absolutely necessary. Text input on devices like the HoloLens is difficult and cumbersome.

Additional Resources

The HoloLens design team has provided a valuable resource for implementing many design best practices into your project. Separate from the HoloToolkit, you'll find another repository on GitHub called the Mixed Reality Design Labs. This is an excellent resource for finding prefabs and scripts for bounding boxes, app bars, progress bars, and more: https://github.com/Microsoft/MRDesignLabs_Unity.

Adding Magic: Vuforia

Devices like the HoloLens are already feats of modern engineering and magical in and of themselves. But there are many ways that developers can add more magic to their applications. It's difficult to define magic in this context, but I tend to define it as any experience that amazes the user due to being pleasantly unexpected or seemingly impossible.

The most powerful way to invoke the feeling of magic is to couple your Mixed Reality application with artificial intelligence or *cognitive services*. Services like IBM Watson, Microsoft Azure, and more allow applications to perform amazing tasks such as image recognition, custom speech/chat capabilities, real-time translation, and more.

Review Azure's list of cognitive services to get a feel for the type of resources available: <https://azure.microsoft.com/en-us/services/cognitive-services/>.

One of my favorite ways of adding magic to an application is to use a powerful tool called Vuforia. Vuforia allows your headset to recognize 2D images and 3D objects in your real world and place a hologram at or near the recognized image/object in real-time. This section gives a brief tutorial on how to download and set up an experience that's powered by Vuforia.

Step 1: Install the Vuforia sample

Download the Digital Eyewear sample for Unity from Vuforia's website, as shown in Figure 8-17, from <https://developer.vuforia.com/downloads/samples>.

Digital Eyewear

These samples demonstrate how to build apps for the different classes of digital eyewear devices.

The **HoloLens** sample shows how to attach an AR experience to an image and enable extended tracking.

The **Stereo Rendering** sample shows how to render a stereo view on the following optical see-through devices: ODG R-6, ODG R-7 and Epson BT-200.

The **AR/VR** sample shows how to create a mixed reality app for a video see-through device.



[Download for Android](#)

vuforia-samples-eyewear-android-6-2-10.zip (10.90 MB)



[Download for iOS](#)

vuforia-samples-eyewear-ios-6-2-11.zip (6.86 MB)



[Download for Unity](#)

vuforia-samples-eyewear-unity-6-2-10.zip (149.87 MB)

[Release Notes](#)

Figure 8-17. Download Vuforia's Unity package for the Digital Eyewear sample

After downloading and unzipping the Unity package, open up a new Unity project and import the HoloLens unity package by going to Assets ► Import Package ► Custom Package. Be sure to select the HoloLens package that you downloaded and not the ARVR or StereoRendering packages that are also included in the file you downloaded from Vuforia's website.

Step 2: Try It Out

To experience the application in action, you'll need to print out Vuforia's target image. You can download it from <https://i.imgur.com/ef26047.jpg>.

No additional settings are required with the app package you imported into Unity. You may test out the application using holographic remoting or by deploying to your device. I recommend deploying the application to your HoloLens for the smoothest experience.

When you launch the Vuforia application, you'll notice a 3D teapot appearing above the image of the rocks that you printed out. Figure 8-18 shows what this looks like. Try to move the paper around on your desk, hold it up, and rotate it. You'll see that the teapot is perfectly attached to the image of the rocks. A truly magical experience!



Figure 8-18. Like magic, a holographic teapot appears on the image of rocks with Vuforia's HoloLens sample

Be sure to sign up for a Vuforia account to gain access to the ability to add your own custom image targets and to learn more about the platform. For more information, visit <https://developer.vuforia.com>.

Capstone Project

In this section, we'll build a new project together that implements the best of what we've learned so far in this book. Over the last several chapters, you've learned a lot about HoloLens development. From creating your first app in Unity (Chapter 2) to understanding the best practices of Mixed Reality design (this chapter), you've equipped yourself with some handy tools for diving in as a developer. Let's put all this knowledge to work by building a new awe-inspiring app from scratch.

For our capstone project, we'll build an experience that requires you to control a ball to cross a river of Lava.

Step 1: Import HoloToolkit to a New Unity Project

Before proceeding, be sure you've already downloaded and saved the HoloToolkit Unity package per the instructions in Chapter 1.

- Create a new Unity project (see Chapter 2 if you need a reminder on how to do this) and name it Lava App.
- *Important:* Save your scene and give it a name. If you don't save your scene, you won't be able to apply HoloLens settings in Step 2.

- From the menu bar, go to Assets ► Import Package ► Custom Package. In the pop-up window that appears, browse to the HoloToolkit that you downloaded in Chapter 1. See Chapter 3 if you need a reminder on how to import the HoloToolkit.
- Unity will take a minute to prepare the package you selected and then show you another pop-up window where you can select or deselect package items. Go ahead and leave everything checked (everything should be checked by default) and click the Import button.

Step 2: Apply HoloLens Settings

After completing Step 1, you should now see a HoloToolkit menu item in your menu bar, as shown in Figure 8-19.

- From the menu bar, select HoloToolkit ► Configure ► Apply HoloLens Scene Settings. This will make the scene background black (when in the Game tab) and modify camera settings. Click the Apply button in the pop-up window that appears. *Important:* Save your scene.
- From the menu bar, select HoloToolkit ► Configure ► Apply HoloLens Project Settings. This will convert the Unity project to a Windows Direct 3D (D3D) project, optimize quality, and enable virtual reality support. Click the Apply button in the pop-up window that appears. Unity will require you to reload your project. *If you didn't save your scene from the previous step, you will lose all changes to the scene and need to apply scene settings again.*

■ **Tip** After choosing to apply settings from the HoloToolkit menu, a pop-up window appears showing settings to apply. Click each item to learn more about each item.

- Remove the Main Camera game object from the Hierarchy by right-clicking each item and choosing Delete from the context menu.
- To insert our new camera, go to the Project panel and browse to HoloToolkit ► Input ► Prefabs. Drag and drop HoloLensCamera.prefab into the Hierarchy, as shown in Figure 8-20.
- Save your scene.

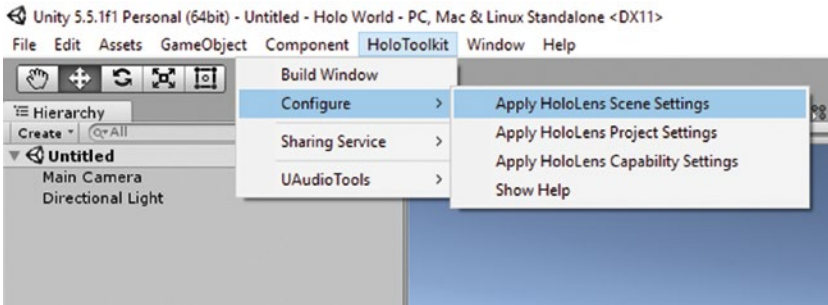


Figure 8-19. You now have a shiny new HoloToolkit menu item. Be sure to Apply HoloLens Scene Settings and Project Settings.

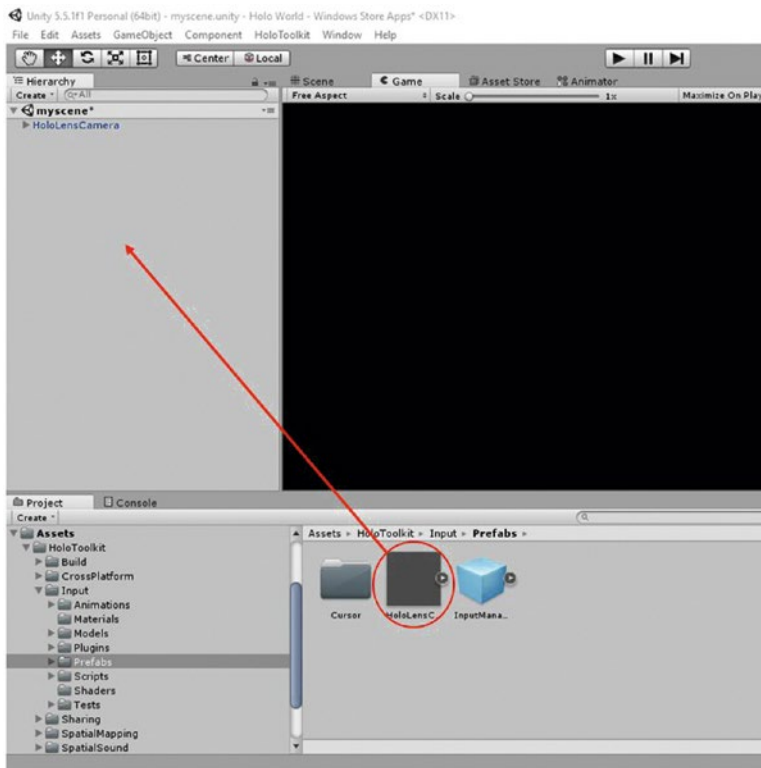


Figure 8-20. After deleting the Main Camera and Directional Light objects, insert the HoloLensCamera prefab into the Hierarchy

Step 4: Insert and Configure InputManager

Next, we want to insert the InputManager prefab into our Hierarchy. This will allow us to use gaze and gestures in our application.

To keep things organized, I like to create an empty game object in my Hierarchy to contain all my manager prefabs and scripts. To do this, right-click in an empty part of your Hierarchy and select Create Empty from the context menu. Right-click the newly created game object and select rename from the context menu. Rename it to Managers.

Next, search for *InputManager* in your Project panel and drag the InputManager prefab from your search results to your Managers game object in your Hierarchy, as shown in Figure 8-21.

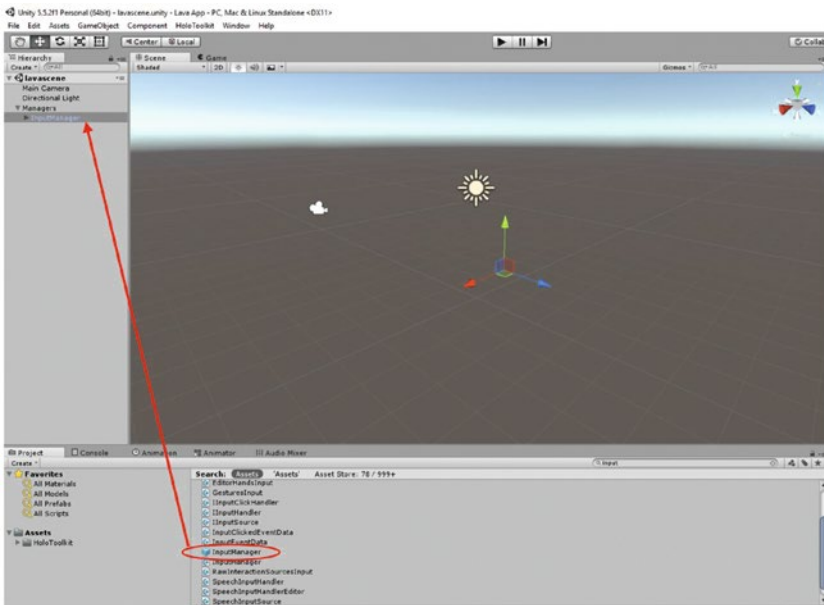


Figure 8-21. Drag the HoloToolkit's InputManager into your Hierarchy

Step 5: Add a Cursor

We'll need a cursor for our application so users can keep track of where they're gazing.

Search your project folder for *cursor* and drag the Cursor prefab into your Hierarchy.

There are several items named “cursor” with similar icons, so you may need to highlight each one to ensure you select the prefab. See Figure 8-22 for the proper prefab icon. Feel free to click the play button to test your app after each step to ensure you're not seeing any errors.

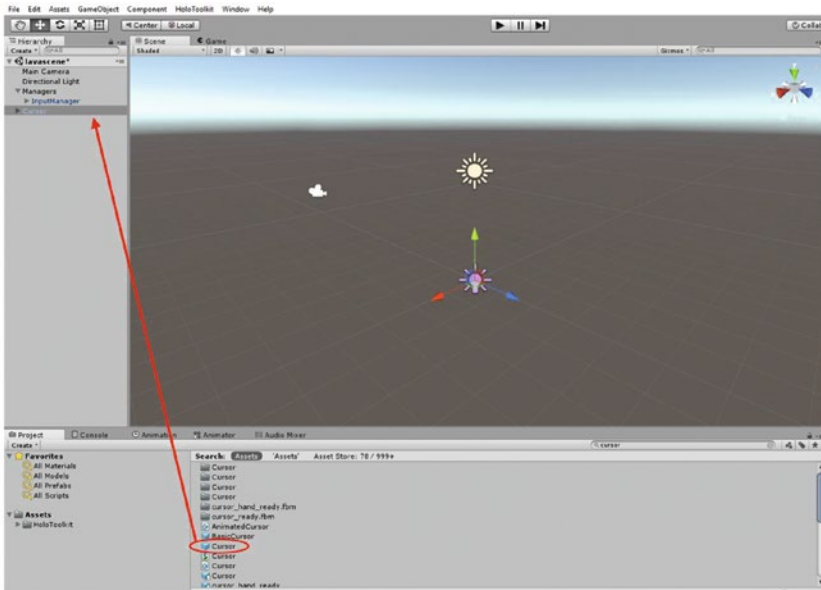


Figure 8-22. Drag the Cursor prefab into your Hierarchy to enable a cursor in your app

Step 6: Create Responsive Ball

Next, we'll create a small ball that responds to our gaze by following it. To achieve this, we'll need to create a ball, a surface (plane) from the ball to roll on, and a script that tells the ball to roll towards the gaze cursor. I'll assume that you're fairly comfortable with creating a plane and sphere by now. If you need a refresher, feel free to review the Roll-a-Ball tutorial in Chapter 2.

First, let's create our surface plane by right-clicking the Hierarchy and selecting 3D Object ► Plane from the context menu. Make sure the plane was not created as a child of any other game object. Reset the plane's position to 0,0,0.

Next, create the ball by right-clicking the Hierarchy and selecting 3D Object ► Sphere from the context menu. Again, make sure it wasn't created as a child of another game object. Position the sphere so that it's resting on the surface of the plane, or slightly above it. Resize the ball to 0.1,0.1,0.1.

Let's keep things organized as we proceed. Go ahead and rename the plane to floor and rename the sphere to ball. Create an empty game object and name it Holograms. Put both game objects into Holograms, as shown in Figure 8-23.

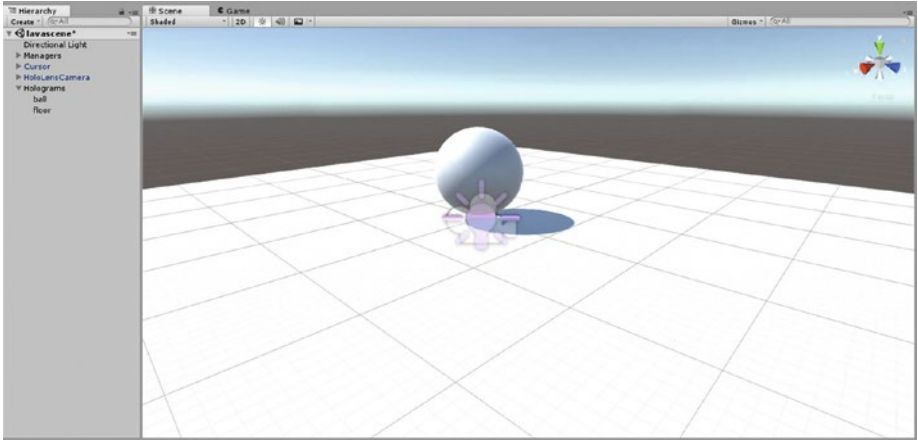


Figure 8-23. Insert a plane and a sphere as the first holograms of your app. Rename them *floor* and *ball* and make them children of a new empty game object called *Holograms*.

Now that we have a floor and a ball, let's add some physics to the ball by clicking the ball and adding the Rigidbody component from the Inspector panel. This allows the ball to respond to gravity and forces.

The force of gravity will keep the ball on the floor plane. Next, we want to add a force on the ball that always tries to pull the ball toward the cursor. Create a new script called *ballmanager.cs* by selecting the ball and then selecting Add Component in the ball's Inspector panel. Type *ballmanager* in the search bar and then click New Script, as shown in Figure 8-24. In the next screen, ensure that the script is a C# script and click Create and Add to create and add the script.

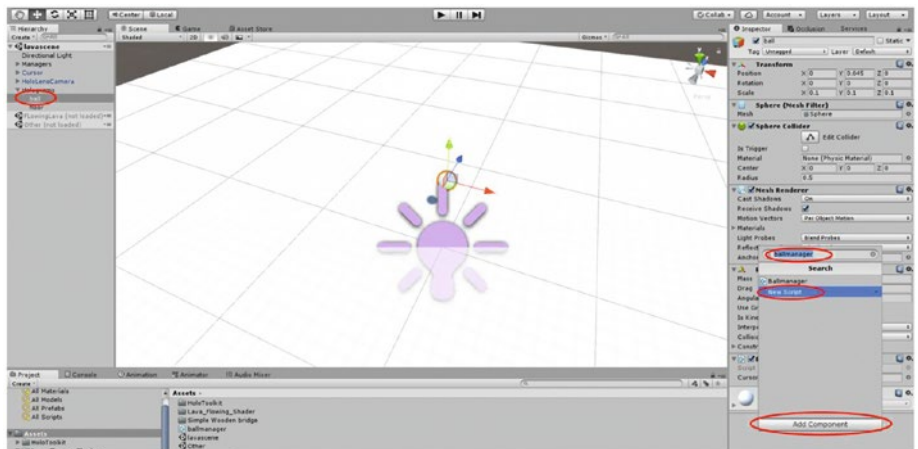


Figure 8-24. Create the *ballmanager.cs* script

Double-click the script to open the `ballmanager.cs` script in Visual Studio for editing. I'll walk through code additions in the text, explaining each addition of code we'll add to our new script. I include the full code at the end of this segment so you can ensure your `ballmanager.cs` script is properly set up.

We need to let our script know about our cursor object, because that's the object that will be pulling our ball towards itself. To do this, let's add this line of code before our `Start()` section:

```
public GameObject cursorObject;
```

This line is saying there's a public `GameObject` that we're calling `cursorObject` (our own made-up name). By making this `GameObject` public, it allows an empty field to appear in the Inspector panel, where we can drag and drop (or search for) a game object to include. In Figure 8-25, you can see that I included our cursor object in that field. I did this by dragging and dropping the `CursorOnHolograms` game object from the Hierarchy.

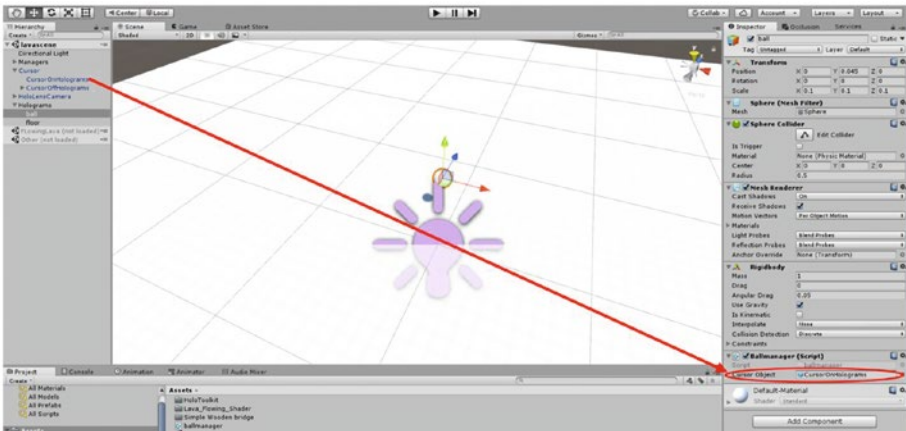


Figure 8-25. Drag and drop the `CursorOnHolograms` game object to your newly created cursor object field

The drag-and-drop method is not necessarily a best practice for development. You may also use the ability to find objects by name within the script using the `GameObject.Find` and `Transform.Find` methods. In general, I like to write code with simplicity and clarity in mind so that you may easily follow along. I highly recommend reading up on advanced C# programming in Unity as you continue developing your skills.

Next, we'll add these two lines of code into the `Update()` section:

```
Vector3 forceDirection = cursorObject.transform.position - transform.position;
gameObject.GetComponent<Rigidbody>().AddForce(forceDirection * 1);
```

The first line of code creates a new vector called `forceDirection`. That vector is created by subtracting the 3D position of the ball (`transform.position`) from the `cursorObject`. If you could visualize this vector, it would look like an arrow pointing from the ball directly to the cursor.

The second line of code adds a force to the ball in the direction of the vector we just created (`forceDirection`). We can multiply the force direction by a number (in this case 1) to adjust the strength of the applied force. Feel free to adjust this number during your tests. Other ways to enhance this code (which I won't show here to keep things simple) could be to add some friction or *dampening*, changing the force based on distance, and eliminating the force when the cursor is far enough away from the ball.

The complete code of `ballmanager.cs` should now look like Listing 8-1.

Listing 8-1. Ballmanager.cs Complete Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ballmanager : MonoBehaviour {

    public GameObject cursorObject;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

        Vector3 forceDirection = cursorObject.transform.position -
            transform.position;

        gameObject.GetComponent<Rigidbody>().AddForce(forceDirection * 1);

    }
}
```

Feel free to test your application. You'll be pleasantly surprised to see that you can now control the ball with your gaze.

Step 7: Download Assets

Our ball and floor look somewhat dull. Let’s add some design and magic to our scene. When developing applications, the Unity Asset Store is your friend because it offers plenty of free and low-cost assets you can download for your project. Beware that some assets might not be optimized for the HoloLens (too many polygons, computationally expensive shaders, and so forth). Look for assets that say *low-poly* or that are built for VR or AR for best results.

To open the Asset Store, go to Window ► Asset Store in your menu bar, as shown in Figure 8-26.

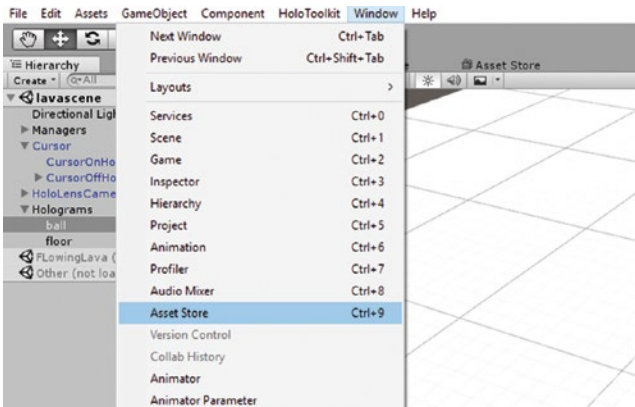


Figure 8-26. Open the Unity Asset Store

There are a few assets to download for our application. First, let’s download a flowing lava shader, which will be our river of lava that the ball needs to cross. Search the Asset Store for *Lava Flowing Shader*. See Figure 8-27 for an illustration of this free asset’s page. Click the Import button to import this asset into your project.

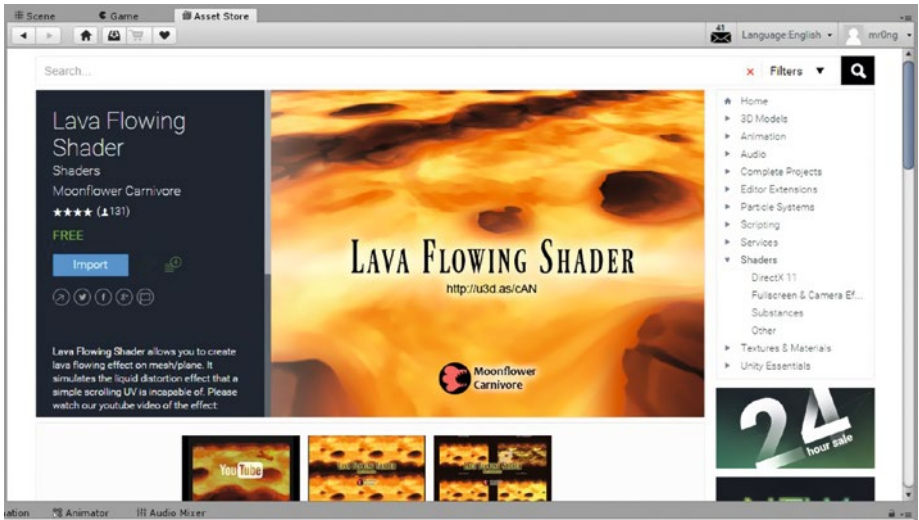


Figure 8-27. Download and import the *Lava Flowing Shader*

When you import an asset, the files will appear in your Assets folder. From there, you can typically drag assets into your scene. We'll apply the lava asset in the next step. For now, we'll continue with installing assets.

The next asset we'll install is the bridge for our ball. Search the asset store for *Simple Wooden Bridge*, another free asset. Download and import this asset into your project. Figure 8-28 shows this asset's store page.

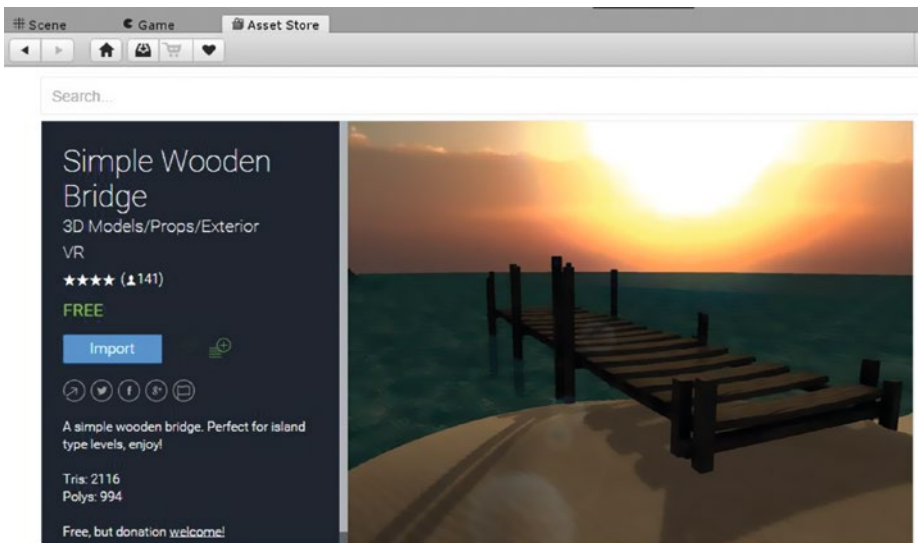


Figure 8-28. Download and import the *Simple Wooden Bridge* asset

Finally, let's make our ball look a little more like a rock. Find, download, and import the “Cracked stone filled with lava” asset from the Unity Asset Store. Figure 8-29 shows this asset's store page.

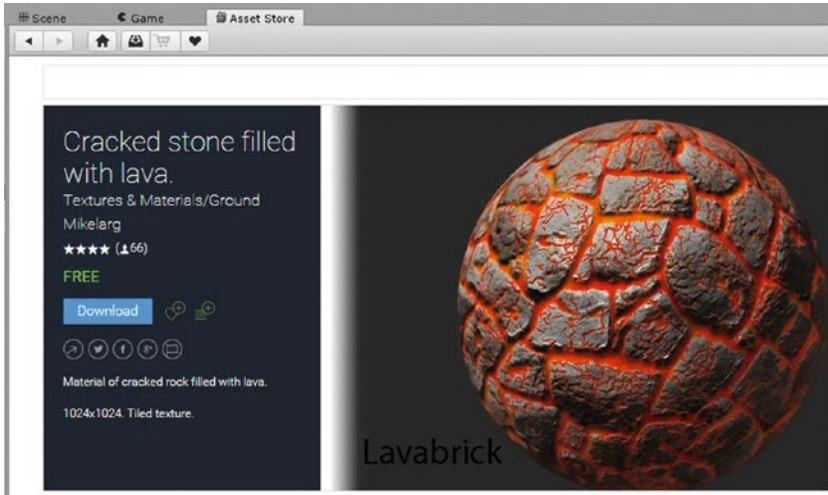


Figure 8-29. Download and import the “Cracked stone filled with lava” asset

Step 8: Create Your Lava Scene

Let's make our lava scene look beautiful. First, let's apply the lava shader to our floor. In your Project panel, navigate to `Lava_Flowing_Shader` ► `Materials` ► `mtl_lava_simple` and drag that material to your floor. You should see the lava material applied, as shown in Figure 8-30. Feel free to explore how the other material options differ. Each will have a slightly different visual appearance.

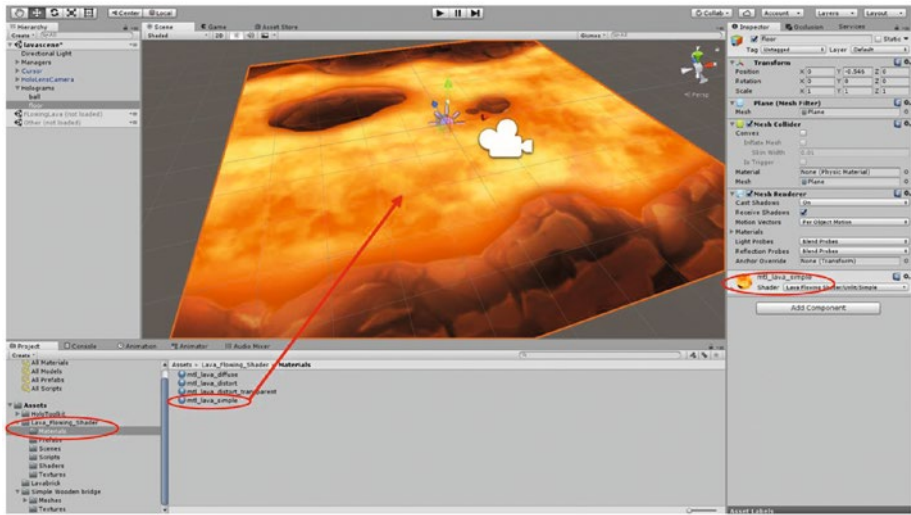


Figure 8-30. Apply the lava texture to your floor object

If you play your scene, you'll notice that the shader is static and not flowing. We need to apply a script to the floor. Navigate to `Lava_floating_shader ► scripts` and drag the `ScrollingUVsLayers.cs` script to the floor's Inspector panel. Change the following values in the script's Inspector panel fields:

- X: 0.06
- Y: 0
- *Texture Name:* `_LavaTex`

See Figure 8-31 for what this script should look like. Now when you play the scene you should see a nice flowing river of lava and a white ball that follows your gaze cursor.

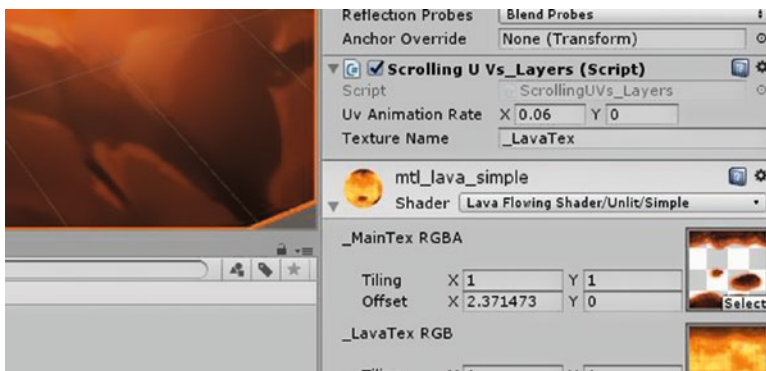


Figure 8-31. Add the `ScrollingUVsLayers.cs` script for the flowing lava effect

Next, let's add the rock texture to our ball. In your Project panel, navigate to the Lavabrick folder and drag the Lavabrick material to your ball, as shown in Figure 8-32. When you play the scene, you'll notice that the rolling ball with the rocky texture has a very nice appearance. The scene is really starting to come together.

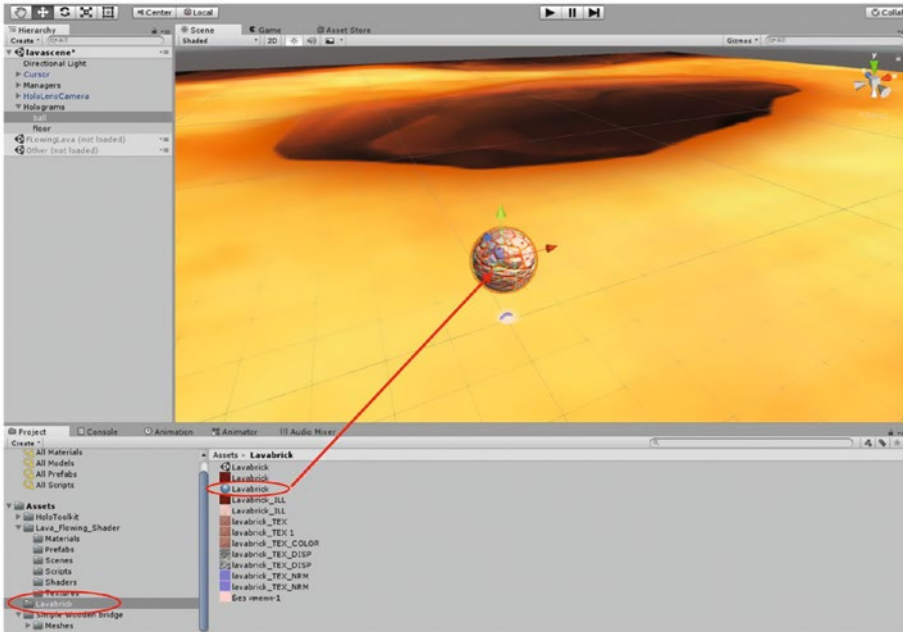


Figure 8-32. Drag the Lavabrick material to the ball to give it a rocky texture

The final addition to our scene will be the bridge that we'll use to cross the lava. In your Project panel, navigate to Simple Wooden Bridge ► Meshes and drag the Brudge.fbx model into your scene. You will likely need to resize and reposition your bridge so that it spans either size of the lava river, as shown in Figure 8-33. *Important:* Ensure that the bridge is a child of the Holograms game object.

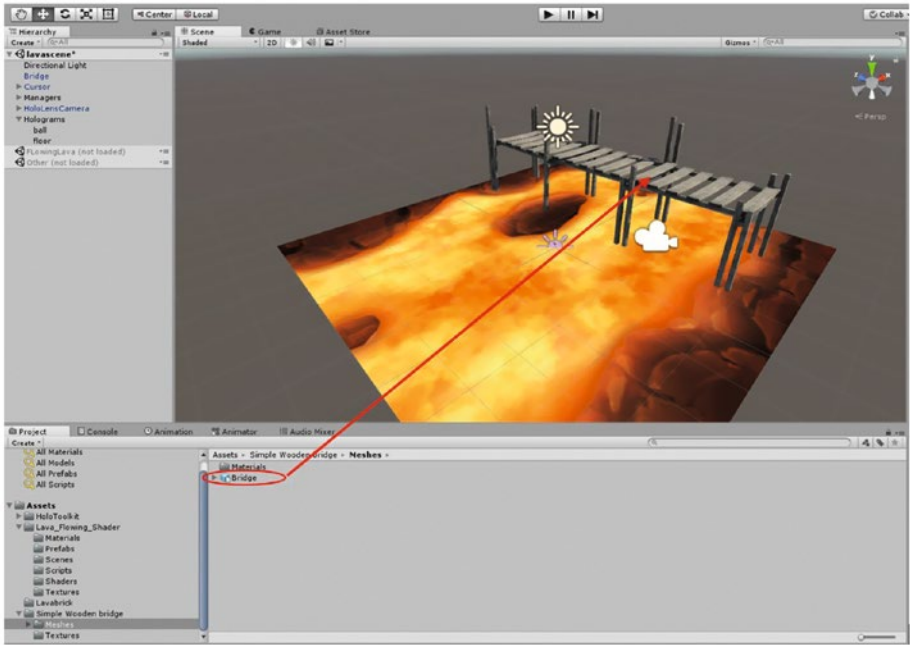


Figure 8-33. Add the bridge to your scene. Be sure to resize and position it appropriately.

Congratulations! We now have all the physical components we need in our scene. As you can see, without too much effort we’ve already created a stunning scene consisting of a gaze-responsive rolling lava rock, a flowing river of lava, and a rustic wooden bridge. From here, we’ll apply additional code logic, sound, and effects for a complete experience.

Step 9: Add the Ability to Move Lava Scene

Let’s add the ability for us to move the entire lava scene by dragging it with our hands. To do this, select the Holograms game object. In the Inspector panel of the Holograms object, click Add component, search for *Hand Draggable*, and click the result to add *handdraggable.cs* to this object, as shown in Figure 8-34. Feel free to adjust the parameters for various ways to grab and orient your scene. I prefer locking rotation when grabbing, and keeping the scene upright. Go ahead and test this addition in the Unity Editor or using holographic remoting to ensure the entire model moves as expected when you grab and move it.



Figure 8-34. Add the *handdraggable.cs* script to the *Holograms* game object to enable gesture-based grabbing and moving of the lava scene.

Step 10: Add and Configure Spatial Mapping

For realism, let's enable spatial mapping in our application. This will allow physical objects in the room to occlude our lava scene so that it doesn't appear to penetrate walls and objects. To enable spatial mapping, search for the *SpatialMapping* prefab included with the HoloToolkit and drag it into your *Managers* game object, as shown in Figure 8-35.

You must also enable *Spatial Perception* to your Unity application by going to **Edit > Project Settings > Player > Settings for Windows Store > Publishing Settings > Capabilities**.



Figure 8-35. Enable spatial mapping by dragging the *SpatialMapping* prefab into your *Hierarchy*. Don't forget to also turn on spatial perception in your player settings.

Depending on the size of your area, you may need to adjust the size of your lava scene to fit within your area. Use holographic remoting to quickly test how your lava scene interacts with the spatial mapping mesh of your environment. I reduced the size of my scene to be roughly 1 meter on each side and increased the size of the ball to be roughly the size of a tennis ball. Note that if your lava scene starts below the surface of your floor, you won't be able to see it due to the occlusion of the spatial mapping mesh. After resizing and adjusting the position of your scene, I recommend unchecking Draw Visual Meshes in the SpatialMappingManager.cs script. Doing this will disable visualization of the mesh's white outlines but still maintain occlusion.

Step 11: Add Spatial Sound Effects

Let's breathe some life into our lava scene by enabling spatial sound. First, you need to enable spatial sound in Unity's settings. Go to Edit ► Audio ► Spatializer and select the Microsoft HRTF extension in the Spatializer Plugin drop-down, as shown in Figure 8-36. Set the System Sample Rate to 48000.

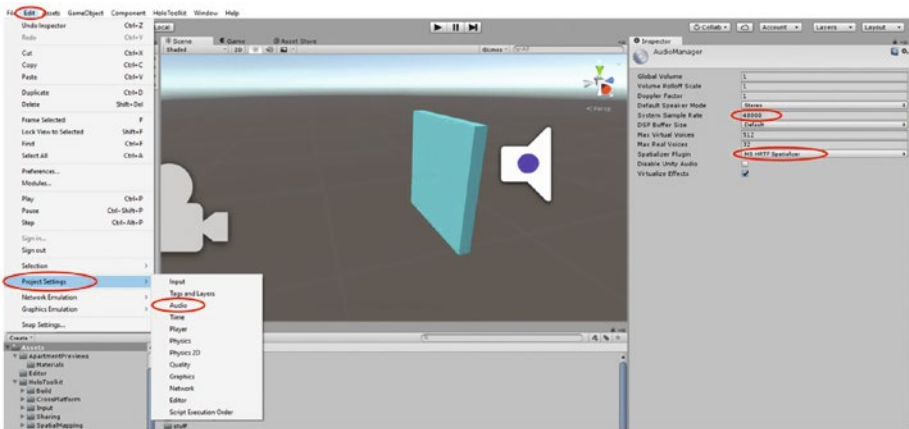


Figure 8-36. Enable spatial audio in Unity's Audio settings. Be sure to select the MS HRTF Spatializer and set the System Sample Rate to 48000.

Second, you need to attach an audio source to the Holograms game object (your lava scene). You can do this by selecting the Holograms game object in your Hierarchy, clicking the Add Component button at the bottom of your game object's Inspector panel, and searching for and attaching the Audio Source component, as shown in Figure 8-37.

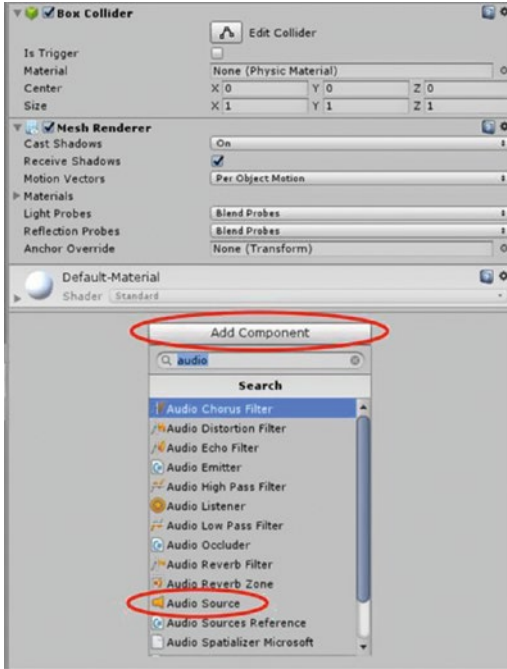


Figure 8-37. Attach an Audio Source component to the game object you want to behave as an audio source

Third, you need to configure the audio source for spatial sound. There are three parameters in the Audio Source component that you need to set, as shown in Figure 8-38. These are the changes you need to make:

- Enable the Spatialize checkbox.
- Set the Spatial Blend to a value of 1.
- Set Volume Rolloff to Custom Rolloff. You may need to expand the 3D Sound Settings item to see this parameter.

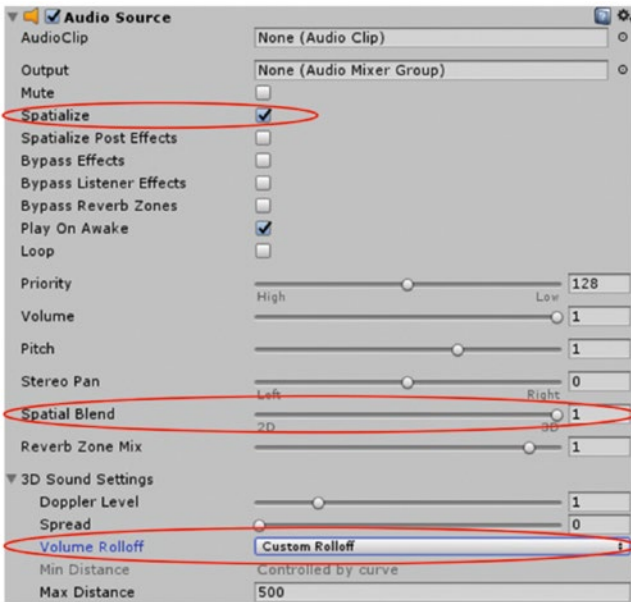


Figure 8-38. Modify the three parameters shown in the Audio Source component to spatialize your sound

Finally, you'll need to drop an audio file from your assets into the AudioClip area and try it out with your headset. For my project, I chose a looping audio file of what sounded like a burning river of lava. Try searching the asset store or looking online for free or paid sound effects to download for your application.

Step 12: Next Steps and Beyond

Congratulations! Together, we've quickly put together a prototype application using just a few of the skills we've learned so far. Let's review what we've learned and developed with our lava application so far:

- How to set up a Mixed Reality project from an empty project
- Download assets from the Unity Asset Store
- Work with advanced shaders
- Apply physics to game objects
- Enable gaze, gestures, spatial mapping, and spatial sound

We've developed a movable lava scene, complete with flowing lava, a red-hot rocky ball that follows your gaze, spatial sound, and occlusion from real-world objects.

There are many ways that you can proceed with this application. Here are a few ideas:

- Put a collider across the lava river. Use `OnCollisionEnter` to detect if the ball collides with the lava colliders so that you can destroy the ball if the ball “falls into the lava.” Find an explosion asset from the Unity Asset Store and trigger an explosion upon the ball’s demise for extra effect.
- Add the ability to tap on the bridge to lower or raise it. Add a mesh collider to the bridge so that the ball can roll across the bridge to the other side of the lava river. Play celebratory music once the ball makes it to the other side.
- Add additional levels and complexity with various arrangements of lava rivers and bridges. Turn the game into an interesting puzzle.

As you can see, there are many ways to improve upon what we’ve built. There’s no limit to what we can imagine and implement in Mixed Reality.

Summary

Creating awe-inspiring applications for Mixed Reality is ultimately about providing a great experience for users. In this chapter, I gave you a few guidelines and resources that will help you on your journey toward creating your own awe-inspiring experiences. I talked about the importance of optimizing the performance of your application, provided you with some insights to speed up your application, walked you through several best practices for good design, and showed you an example of how to add a little magic to your application.

There’s really no limit to how creative you can get when designing for Mixed Reality. There’s not much art you can do on a 1-dimensional line, but vast possibilities are unlocked when painting and developing on a 2D canvas. Bigger yet is the jump to the 3D stage of Mixed Reality, where infinite possibilities are waiting to be discovered and created.

I encourage you to build upon the best practices and tools provided in this Chapter. Try new design experiences—test what works and what doesn’t. The best way to gauge whether or not your application is awe-inspiring is to closely observe the reactions of others when they try your experience. You will quickly perceive if your app invokes a sense of wonder and amazement. Finally, don’t forget to share your lessons learned with the broader Mixed Reality development community (more on this in [Chapter 10](#)).



Turning Holograms into Money

In this chapter, I'll introduce you to several methods by which you can monetize your Mixed Reality development activities. From the app model to offering your services as a freelancer, today's developers can earn money from Mixed Reality in many ways. Developing for Mixed Reality is fun and rewarding, but plenty of business opportunities have also arisen from this revolutionary new medium.

When I bought my first HoloLens unit in the spring of 2016, I started developing for the platform as a hobby and for side projects. I worked on a few applications, both for myself and also others (at no cost). As the year progressed, I started observing that there was a fairly active market for HoloLens developers. I responded to several project bids, and quickly transitioned to becoming a full-time HoloLens developer.

The biggest financial opportunities for Mixed Reality, however, will come from the sheer fact that these devices will replace the way we do computing today. There's an immense amount of financial potential, and we don't need to wait for a mythical forthcoming device or form factor to start dipping into that potential. This chapter will provide you with plenty of resources and discussion for turning holograms into money today.

Publishing Your App to the Windows Store

This section covers publishing and monetizing your applications through the Windows Store, Microsoft's online shop for apps and media, which is perhaps the most direct way to monetize your Mixed Reality experiences. Figure 9-1 shows just a few of the 232 HoloLens applications on the Windows Store as of this writing.

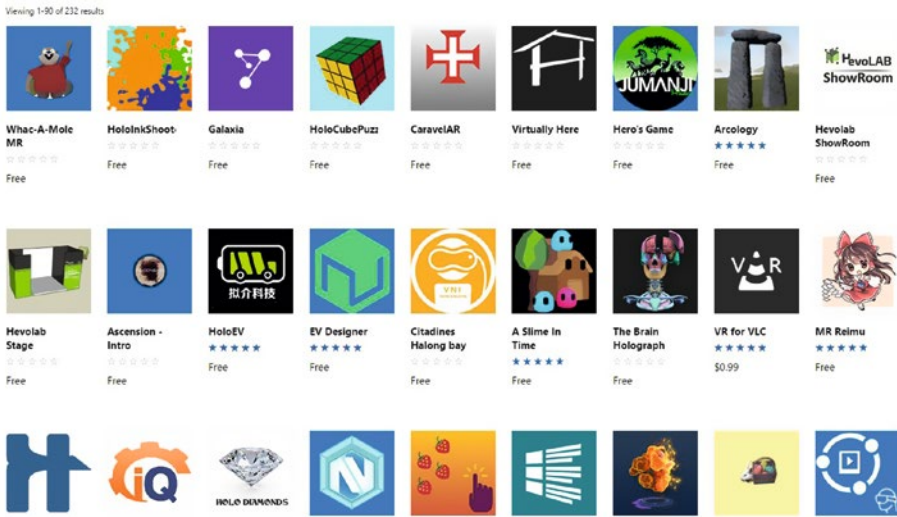


Figure 9-1. The Windows Store contains 232 HoloLens apps as of this writing, with more being added each month

When you publish to the Windows Store, you have several options for monetization:

- *Free:* These are apps that users can download at no cost. You won't receive any direct revenue from this approach, but it's a fantastic way to build your brand and reputation and may lead to other financial opportunities. Developers may also have a portfolio of applications, where free applications may lead users to downloading other paid applications from the same developer.
- *Free with Ads:* Applications that users can download at no cost but that include in-app advertising. Revenue comes from ad clicks and impressions. This is a fairly popular approach among mobile applications, but it's still unproven in Mixed Reality as of this writing.
- *Freemium:* Apps that users can download for free but may opt to pay a premium for additional features, in-app resources, or to remove advertising.
- *Paid:* Apps that users pay for before downloading. Developers have the option of offering a limited trial period before requiring users to pay. According to Practical Analytics, 14 percent of all HoloLens apps use the paid app model.

I've not found any statistics from Microsoft regarding Mixed Reality app downloads, user engagement, and revenue. However, the general consensus among developers is that monetization opportunities on the Windows Store are fairly low. Reports indicate that the number of HoloLens devices currently active on the market are in the thousands, or at

most, tens of thousands. My most popular application on the HoloLens only had a total of 1,215 downloads from June 2016 to June 2017. However, activity on the Windows Store is expected to increase with the introduction of lower-cost headsets to the market.

■ **Note** Several lower-cost Windows Mixed Reality headsets are expected to become available to consumers starting in late 2017. At the time of this writing, these headsets have not yet been released for consumers. Many in the Windows Mixed Reality community anticipate that the lower cost (\$299 for the Acer Headset compared to \$3,000 for the HoloLens) will contribute to a significant growth in the user base, resulting in significantly more download activity in the Windows Store. As such, keep a close eye on Windows Store statistics for Mixed Reality applications. It's difficult for anyone to forecast adoption of these headsets or user engagement in the Windows Store. Because the primary source of Mixed Reality applications will be through the Windows Store, I anticipate that a large consumer adoption of Mixed Reality headsets will translate to huge monetization opportunities in the Windows Store.

Freelancing

This section talks about being an independent Mixed Reality developer and shares best practices for successfully finding and securing opportunities. I share a little bit of my story to illustrate how you can be fully supported by Mixed Reality contracts.

A key turning point in my career was in January of 2017, which marked the point at which I started developing full-time for the HoloLens and was fully supported financially by Mixed Reality freelancing opportunities. It wasn't until after this transition that people around me began to take notice that the HoloLens wasn't just an expensive toy—it had real financial opportunities and use cases for businesses today. To date, I've completed projects for clients in Shanghai, Sydney, Dublin, New York, and Dubai. Earning a living using Mixed Reality is rewarding and exciting and gives you a grounded perspective on how Mixed Reality applications are used in business.

Finding Mixed Reality Freelance Opportunities

I was first made aware of opportunities by monitoring requests through forums and online communities (discussed in detail in Chapter 10). I typically see about three to five freelance and job opportunities per week in the virtual reality and Mixed Reality communities I follow online.

The benefit of finding opportunities through these online communities is that you interact directly with individuals who need help, and a hiring decision can be made swiftly. If you're active in these communities, frequently help others, showcase your work, and contribute to the community, you'll have a good chance of being recognized and sought out, and you may well land the freelance opportunity. It's fairly common for companies to need quick-turnaround help, lasting anywhere from a week to two months.

In addition to online communities, job boards and freelance websites are also an excellent source of contracts that you can bid on. It's harder to stand out on freelance sites, though, and you often don't have the opportunity to connect directly with the manager. However, there is usually a much larger pool of projects to bid to, with new projects being added daily. One website I monitor closely is VREX.io, which has relatively few opportunities overall but a big concentration of HoloLens and Mixed Reality opportunities. Figure 9-2 illustrates what to expect when looking for opportunities on VREX.io.

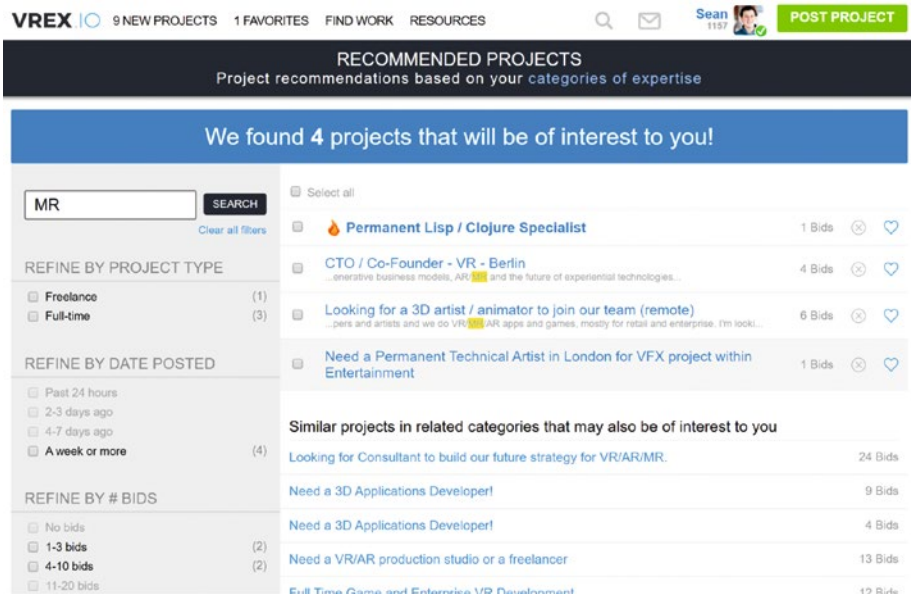


Figure 9-2. Job and freelancer websites like VREX provide many Mixed Reality opportunities to bid on

Increasing Your Chances of Winning a Contract

In winning and losing many HoloLens opportunities, I've observed a few patterns that can help increase your chances of landing a Mixed Reality job.

- *Have a portfolio:* Managers are interested in seeing your portfolio of apps. Try to have at least two or three good applications that you can refer prospective clients to. Have at least one stunning application that you can point to first. Many software development firms claim to have Mixed Reality development capabilities, but lack any real qualifications or prior experience. Managers have a very difficult time trying to filter out true Mixed Reality developers from those with no experience. A portfolio of applications quickly lets managers know that you have hands-on experience with the technology.

- *Submit a proposal:* Sometimes it's easy to think that a casual conversation or e-mail bid will suffice. When I take the time to develop a thought-out proposal, I have a much higher chance of winning the contract. I recommend developing a proposal template that you can use to quickly put together for projects you bid on.
- *Keep in touch:* My most successful projects have come out of lost bid opportunities. Regardless of whether you lost a bid or weren't called back after an initial phone call, keep a record of your various opportunities and follow up from time to time. It keeps you fresh on your client's minds and lets them know you're a dedicated Mixed Reality developer.

You'll always have plenty of competition, regardless of what industry you freelance or consult in. It's no different with Mixed Reality. However, because Mixed Reality is an emerging platform, there are many ways that you can stand out as a developer. Making unique contributions to the community, creating a great application, and making a splash on social media with something cool you did are all ways that can make you stand out and help you start freelancing in Mixed Reality.

Future Opportunities Today

By now, I hope that I've instilled in you a sense of where Mixed Reality is headed. A world in which everyone interacts with holograms instead of 2D screens means that a lot of industries will be disrupted, many new businesses will be created, and many financial opportunities will arise.

The truth is that this future is not a far-off science fiction prediction. It is something already here today. We have the technology and the resources to start building out this future, and there's no reason we shouldn't. Yes, technology will continue to improve (when does it ever stop?), and yes, devices will get smaller, lighter, and less expensive. The HoloLens, however, is a revolutionary device that's fully capable of ushering in our holographic future.

Imagine taking a PC back in time, to several decades before computers were widely used in businesses. Do you think you could walk into just about any business and show them how valuable a computer would be for their company—from document and spreadsheet editing, digital art, record keeping, audio recording, and much more? I suspect you'd be able to sell them on buying a computer without much persuasion. In the same way, Mixed Reality is here to usher in the next computing paradigm and start disrupting the status quo. Whenever you visit or drive by any office or business, start thinking about how that business could leverage holographic technology. Start thinking about the types of applications and experiences you could build that would add value to those companies.

There's nothing stopping you from scheduling a free demo to several companies local to you and pitching them on transitioning to the holographic age. If you're equipped with a compelling solution that will add value, I doubt many will turn down the offer for a fun Mixed Reality demo. In this way, you could create new and larger opportunities for yourself (and the industry) rather than merely respond to freelance and other Mixed Reality job opportunities.

Summary

In this chapter, I provided several insights into ways you can earn money as a Mixed Reality developer. I discussed various monetization models for the Windows Store, provided several ideas for successfully securing freelance and contract opportunities, and offered some inspiration for creating new opportunities all around you.

Ultimately, a technology platform will be widely accepted and embraced if it adds real value. So far, Mixed Reality is living up to its promise by demonstrating that it does indeed add value across a wide range of sectors. As you and other developers continue to build new experiences, discover new ways to interact with holograms, and find new ways this platform can add value to businesses, the financial opportunities of Mixed Reality will grow exponentially.



Community Resources

In this chapter, I introduce you to some valuable online and community resources that will help you on your journey as a Mixed Reality developer. Examples of these resources include relevant community forums, online groups, notable events, and other information that will help during the development process.

I cannot overstate the importance of leveraging community resources during application development of any kind. This is especially true for Mixed Reality development, where the platform is still new and developers everywhere are exchanging valuable lessons learned. As you've heard me echo throughout this book, the world has yet to unlock a good user experience when it comes to immersive computing. I anticipate dozens of "Eureka!" moments over the next few years as we (Mixed Reality developers) come to grips with this revolutionary technology. As such, it would be extremely advantageous to be plugged in to the community to exchange ideas, help one another, and build upon others' successes.

Microsoft's Official Mixed Reality Forum

Microsoft's Mixed Reality forum is an important community resource. Officially called the Windows Mixed Reality Developer Forum, you can find it at <https://forums.hololens.com>.

Note The Windows Mixed Reality Developer Forum was formerly known as the Windows Holographic Developer Forum. As of this writing, Microsoft is making a branding transition from Windows Holographic to Windows Mixed Reality. There is also a transition focusing less on the HoloLens name and more on the broader Mixed Reality platform, so it's possible that the URL <https://forums.hololens.com> may soon be changed or redirect to a revised URL, due to the HoloLens name.

Upon first visiting the forum website, you will be greeted with a page similar to Figure 10-1. The format of this forum has already changed several times, so don't be alarmed if the website you see looks somewhat different from what is shown.

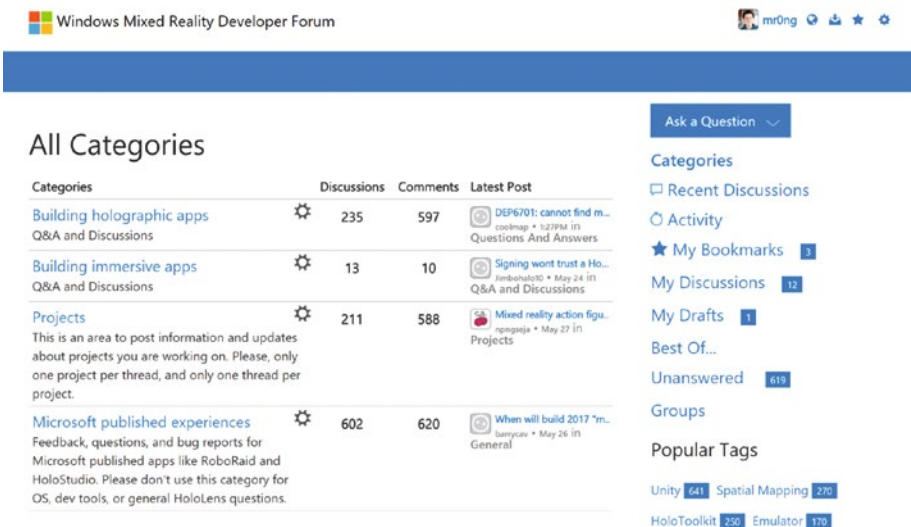


Figure 10-1. The main page of the Windows Mixed Reality Developer Forum, as it appeared in May 2017

As of this writing, the forums are organized into “Building holographic apps” and “Building immersive apps.” There are also additional sections for projects and Microsoft published experiences:

- *Building holographic apps* is a section dedicated to Q&A and discussions surrounding the development of applications for Mixed Reality headsets with transparent displays, such as the HoloLens.
- *Building immersive apps* is a section dedicated to Q&A and discussions surrounding the development of applications for Mixed Reality headsets with opaque or immersive displays, such as the Acer, HP, and Lenovo Mixed Reality headsets.
- *Projects* is a section where developers can showcase and share information about projects they are working on.
- *Microsoft published experiences* is a section pertaining to Mixed Reality applications published by Microsoft, such as Galaxy Explorer, RoboRaid, HoloStudio, Actiongram, and more.

The official forum is very useful for a few reasons:

- It’s closely monitored by members of the Microsoft HoloLens and Mixed Reality teams. Team members regularly respond to questions. It’s a great place to receive expert advice from the creators of HoloLens themselves.

- Major HoloLens and Mixed Reality announcements are posted here.
- Updates to Microsoft’s published Mixed Reality experiences and applications (such as Actiongram, RoboRaid, and others) are posted here.

The forum also has some shortcomings, which has contributed to the rise of many other Windows Mixed Reality community resources online. Common shortcomings that developers experience include the following:

- *Lack of responses:* Although many questions receive answers, many go unanswered.
- *Delayed interaction:* The forum structure doesn’t promote real-time communication between developers. The lack of instant notifications, live chat, and other similar messaging features make this forum less desirable for developers needing urgent answers to questions.
- *Lack of community:* Similar to the preceding bullet point, the forum’s lack of instant messaging features make it somewhat cumbersome for Mixed Reality developers to carry out casual group conversations that promote a sense of community and friendship.

It’s possible that some or all of these issues may be addressed in the future (or perhaps by the time you are reading this). I highly encourage you to sign up for an account on this forum and actively participate in discussions when appropriate.

Overall, Microsoft’s official Windows Mixed Reality Developer Forum is a fantastic community resource that all Mixed Reality developers should be part of. Access to the teams at Microsoft also makes this forum a key resource for developers. The shortcomings are filled in by other online community resources that I cover in this chapter, such as the HoloDevelopers Slack Team.

HoloDevelopers Slack Team

The HoloDevelopers Slack Team is my personal favorite HoloLens and Mixed Reality online community and by far what I consider the most helpful for new developers. In this section, I introduce the HoloDevelopers Slack Team, including information on how to join the group and the best way to participate in this community.

What Is Slack?

For those not familiar with Slack (<https://slack.com>), it is a fantastic collaboration and communication tool for groups. It can be thought of as a big chat room platform, where a community can discuss ideas and share content across multiple chatrooms. The power of Slack is its cross-platform compatibility (web, iOS, Android, Mac, Windows, Windows Phone, HoloLens, and more) as well as the ability to chat with and interact with large groups of people across multiple chat rooms (called *channels*), where each channel is

dedicated to specific topics of discussion. Anyone can create a Slack team, and thousands of Slack teams exist for a wide range of topics. Slack is also popular among businesses, which use it as an employee communications tool.

What Is the HoloDevelopers Slack Team?

The HoloDevelopers Slack team is the “semi-official” developer Slack team for all things HoloLens and Windows Mixed Reality. It’s a place where developers can share experiences, ask questions, and talk about Windows Mixed Reality. I say *semi-official* because this Slack team was not founded by Microsoft, yet it has become so foundational that Microsoft now recognizes it on its website (<https://developer.microsoft.com/en-us/windows/mixed-reality/community>). Dozens of Microsoft employees from the HoloLens team regularly contribute and participate in the HoloDevelopers Slack team.

The HoloDevelopers Slack team was founded by Jesse McCulloch of Roarke Software (<http://roarkesoftware.com>) after being frustrated by some of the shortcomings of Microsoft’s official Windows Mixed Reality developer forums. The Slack team was intended to provide Mixed Reality developers with a greater sense of community and quicker, more interactive feedback when asking questions.

The HoloDevelopers Slack team contains an evolving list of many relevant discussion channels, each active with lively conversations. Figure 10-2 shows one conversation on this Slack team. As of this writing, the HoloDevelopers Slack contains 900 users and is growing at a rate of about 25 to 30 new users per week. You can find the Slack team at <https://holodevelopers.slack.com>.

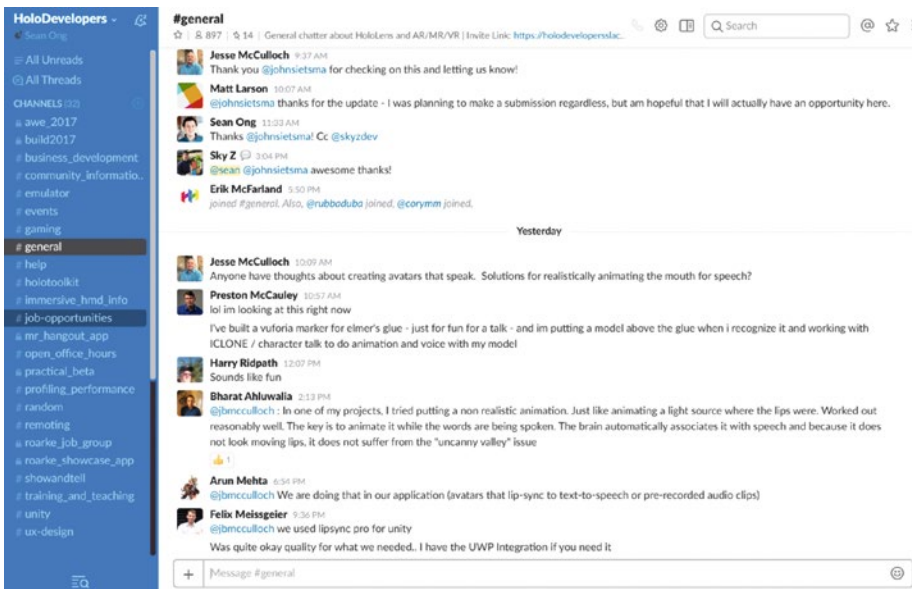
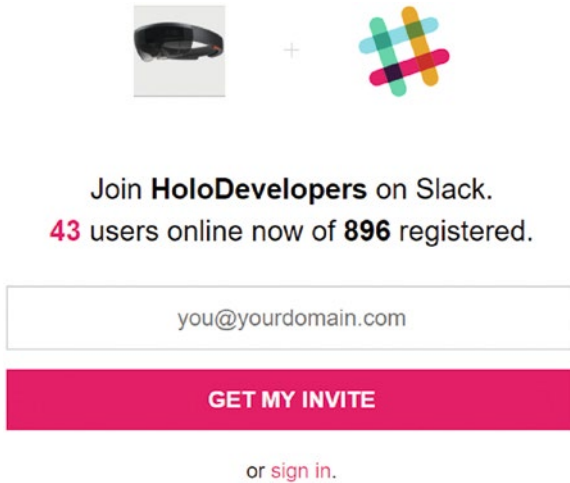


Figure 10-2. The HoloDevelopers Slack team is a lively group of HoloLens and Windows Mixed Reality developers

How to Join the HoloDevelopers Slack Team

Joining the HoloDevelopers Slack team is easy. Enter your e-mail address at <https://holodevelopersslack.azurewebsites.net>, as shown in Figure 10-3. You'll instantly receive an invitation to join the Slack team, at which point you can sign up for your account.



Join **HoloDevelopers** on Slack.

43 users online now of **896** registered.

you@yourdomain.com

GET MY INVITE

or [sign in](#).

Figure 10-3. Use the signup link to get an instant invite to join the HoloDevelopers Slack Team

Participating in the HoloDevelopers Slack Team

Once you're a member of the HoloDevelopers Slack team, introducing yourself to the community is a great way to kick off a conversation.

I recommend using the *#help* channel for any project-related questions you may have. Use the *#general* channel for general Windows Mixed Reality discussions, and use the *#random* channel for anything unrelated to Mixed Reality or whenever you're unsure if your content fits within the *#general* discussion.

Here are some general tips to make the most of being part of this Slack community:

- Microsoft has HoloLens employees actively participating in this community. Be sure to reach out to them whenever appropriate.
- Don't be afraid to ask tough questions. This Slack team boasts some amazing talent, and there's always someone happy to help. If your question goes unanswered, be persistent in asking the community.
- Make some money. Check the *#job-opportunities* channel regularly for fun employment and contract opportunities.

- Be sure to install the Slack app on your phone and PC to get notifications and easily follow discussions you're interested in.
- Use the direct messaging feature to have one-on-one conversations with individuals.
- Share your work. Everyone in the Mixed Reality community loves to see each other's progress and accomplishments. Share your work and share lessons learned.

■ **Note** As of this writing, the HoloDevelopers Slack Team is operating on a free Slack plan. As such, Slack limits searching of messaging to the last 10,000 messages. Though this may seem like a lot, it only takes a few weeks to cycle through 10,000 messages due to the activity of this Slack team. To view and search for the vast history of Mixed Reality posts shared on Slack, be sure to check out the Slack team's archive at <https://holodevelopers.slackarchive.io>.

Overall, if there's only one Windows Mixed Reality community to be part of, I would definitely choose the HoloDevelopers Slack team—even above Microsoft's official forum. The community, level of engagement, and quality of developers in this group make it second to none. I highly recommend joining and checking in regularly on this community.

Other Online Communities and Resources

In this section, I introduce other online HoloLens and Windows Mixed Reality communities and groups that you can participate in.

HoloLens Developers Facebook Group

As expected with the Internet, there are hundreds (and possibly thousands) of online groups, forums, and communities that you can join and participate in as a Mixed Reality developer. That said, I consider there to be three primary online communities. We've already talked about the first two: Microsoft's official Windows Mixed Reality Developer Forum and the HoloDevelopers Slack Team. The third is the HoloLens Developers Facebook group, located at www.facebook.com/groups/winhololographicdevs/.

From the description on this Facebook group, the HoloLens Developers group is an “open group to share thoughts, information, everything you want about the Microsoft HoloLens, Mixed Reality, and how to develop with these technologies.”

As of this writing, it boasts over 4,500 members and is the largest Windows Mixed Reality developer group on Facebook. Figure 10-4 shows an example of what you'll see when visiting this group. Facebook will prompt you to become a member of this group before you're allowed to post or comment in this group. One of seven administrators will grant you access, typically within a few hours of requesting to join this group.

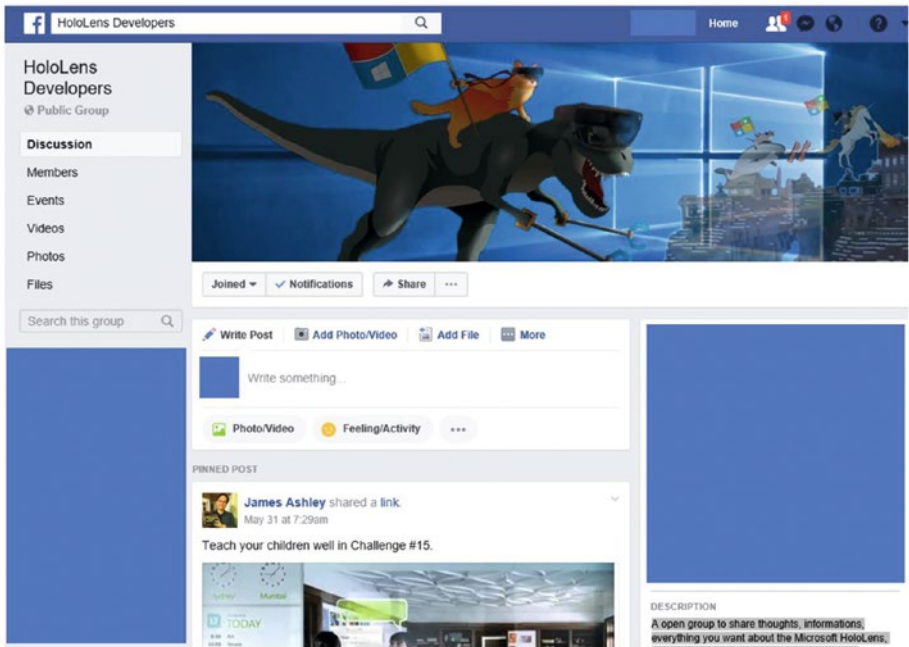


Figure 10-4. The HoloLens Developers group is the largest Windows Mixed Reality Developer group on Facebook

There's a certain degree of user overlap between this group and the other popular online communities we've covered so far. Most days, each group has different sets of active contributors and different content is shared and covered across each of these groups. For this reason, I typically monitor these (and other) communities on a weekly basis.

The Facebook group is generally more useful for the sharing and consumption of Mixed Reality news and experiences. New users can see the group's photos, links, and history easier than on Slack or the Forums. It's also convenient for developers who are comfortable on Facebook and frequently use the platform. This group is not ideal, however, for real-time chat and discussion. Extended developer discussions may also be difficult to follow on Facebook.

There are dozens (if not hundreds) of HoloLens and Mixed Reality related groups on Facebook. Figure 10-5 shows a small sampling of groups that appear when I searched for *Windows Mixed Reality* Facebook groups. Some of these groups have several thousand members. I've not had the opportunity to explore each of them, but if you're looking for a certain niche Windows Mixed Reality community, you're bound to find something relevant on Facebook.

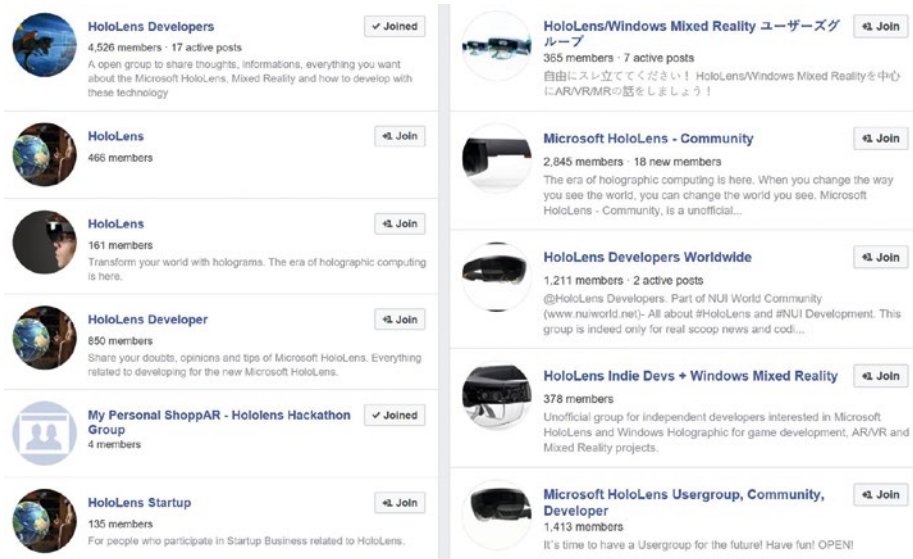


Figure 10-5. There are many HoloLens and Windows Mixed Reality groups on Facebook to choose from

Unity and Unity HoloLens Forum

Some of the most powerful development resources for any Unity-based application (including Windows Mixed Reality applications) are the Unity Forums. You can find the Unity Forums at <https://forum.unity3d.com>.

When asking your favorite search engine any Unity-related question, you'll most likely be taken to the Unity Forums for your answer. Outside of the Mixed Reality world, Unity is widely used for game development. This is excellent news, because it means that there are years of tutorials, resources, and forum discussions to help answer almost any question you may have as you're developing your Mixed Reality applications.

In addition to the broader Unity Forums community, there is also a HoloLens-specific Unity Forum at <https://forum.unity3d.com/forums/hololens.102>. In the HoloLens-specific forum, you'll find a great community of people helping each other and discussing HoloLens and Windows Mixed Reality development in Unity.

HoloLens Subreddit

If you're not familiar with Reddit (www.reddit.com), it is the seventh most popular website in the world (at the time of this writing). Reddit is popular because users "vote" relevant news and content to the top of users' feeds, instead of being presented curated content by unknown search engine algorithms or hand-picked by media agencies.

There are countless topic groups on Reddit, called *subreddits*. The HoloLens subreddit (www.reddit.com/r/HoloLens/) is the most popular subreddit for HoloLens and Windows Mixed Reality, boasting roughly 6,000 subscribers as of this writing. Figure 10-6 shows what you can expect to see when visiting the HoloLens subreddit.

The HoloLens subreddit is an excellent resource for filtering out relevant Windows Mixed Reality news from irrelevant or unimportant content. Naturally, any important or relevant posts will receive a higher number of upvotes and rise to the top of your feed.

Reddit also has a useful feature to sort by most upvoted posts for various time periods. As you can see in Figure 10-6, I've listed the top posts during the past month by clicking Top in the upper menu and then Past Month in the lower menu bar. This allows infrequent visitors to check in every few days/weeks/months and make sure that they didn't miss any big Windows Mixed Reality news or content.

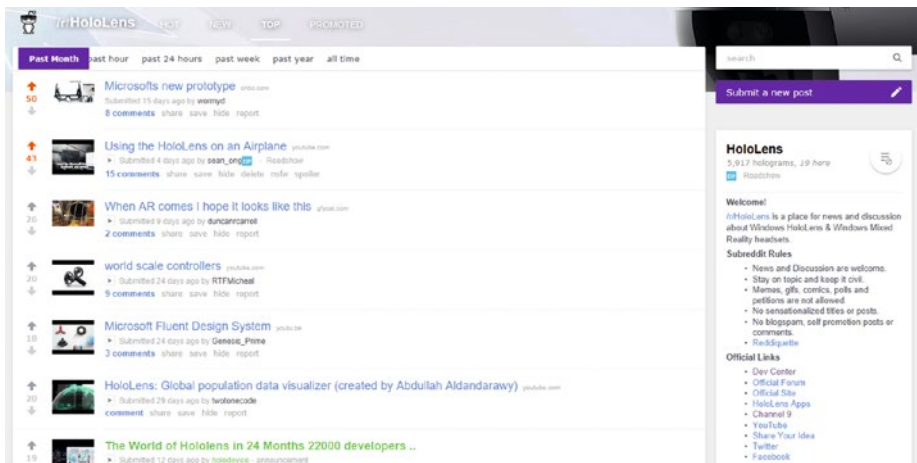


Figure 10-6. Subscribe to the HoloLens subreddit to stay up to date on the most relevant and exciting HoloLens and Windows Mixed Reality news

Don't forget to read the comments section of important posts. Reddit boasts an active community of commenters who share opinions and valuable insights, adding rich context and humor to most submitted posts.

Next Reality News

Of the vast number of technology-related news websites available, I've found that Next Reality News (<https://next.reality.news>) consistently provides the best coverage of Windows Mixed Reality headsets and software. It also regularly publishes tutorials that are helpful for HoloLens and Mixed Reality developers.

The special focus given to Windows Mixed Reality can be partially attributed to Jason Odom, a well-known Windows Mixed Reality developer and author. He is on the Next Reality News team and frequently publishes in-depth HoloLens articles. You'll also see Jason Odom actively participating in community groups such as the HoloDevelopers Slack team and the HoloLens Developers Facebook group.

Next Reality News is a great place to read in-depth coverage and hear opinions on Windows Mixed Reality (and other augmented/virtual reality) news with a special developer perspective that you'll be hard pressed to find at any other news source. The authors are also very accessible and always happy to interact with readers via the comments section or social media. Figure 10-7 shows what you can expect to see when you visit Next Reality News.

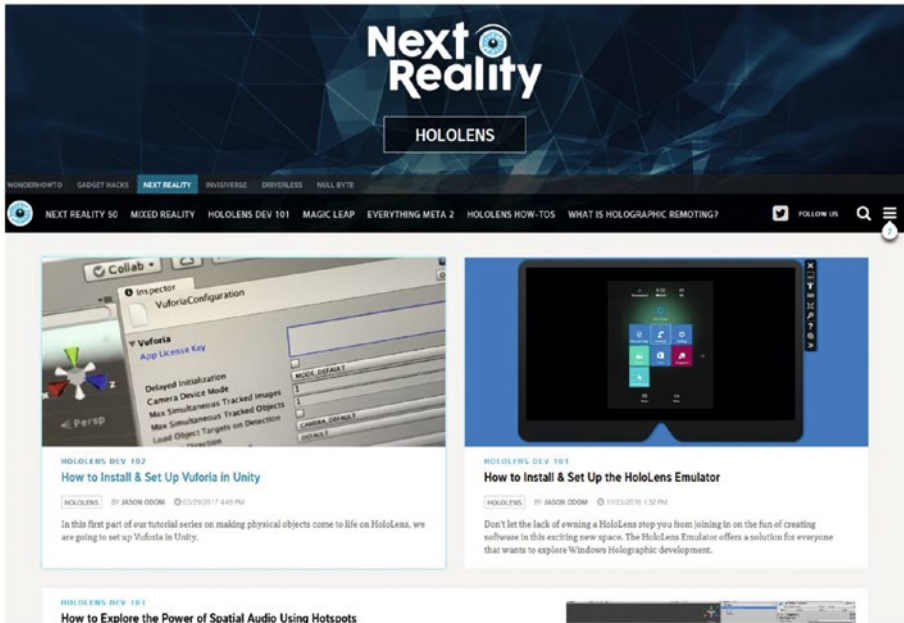


Figure 10-7. Next Reality News is an excellent community and source for Windows Mixed Reality and other VR/AR news written by HoloLens developers

YouTube

The best way to quickly visualize and learn about another Mixed Reality experience is to watch a video. This is why YouTube has been a valuable platform for developers to share their Windows Mixed Reality apps to the world. Here are a few YouTube channels worth subscribing to in the HoloLens and Windows Mixed Reality space:

- **Sean Ong's YouTube Channel:** A shameless plug for my YouTube channel, where you can find relevant content on my latest Windows Mixed Reality projects. With over 36,000 subscribers as of this writing, my channel is best known for technology-related tutorials, tips, tricks, and news with a special focus on Windows Mixed Reality and Microsoft products. Find me at www.youtube.com/c/seanong.

- *Official HoloLens YouTube Channel:* Follow Microsoft's official HoloLens YouTube channel for tutorials, app features, and inspiring examples for your next project at www.youtube.com/channel/UCT2rZIAL-zNqeK10mLLUa6g.
- *Matrix Inception's YouTube Channel:* With 540 subscribers as of this writing, Matrix Inception's rising YouTube channel features some of the most innovative concepts in Windows Mixed Reality, including beaming lasers through portals, a keyboard you can use in your own application, room scanning tricks, reviews, and more. See this channel at www.youtube.com/channel/UC5WLFKmv6BPFTBz0cZQzVag.
- *The Holo Herald:* Another up and coming YouTube channel dedicated to covering HoloLens and Mixed Reality content. The Holo Herald is best known for thorough coverage of reviewing Windows Mixed Reality applications in the Windows Store. Follow The Holo Herald at www.youtube.com/channel/UCTCokLfwnQN-zvbKFE96Lja>

Local Events and Meetups

This section introduces some ways you can get involved with local Mixed Reality events near you. Although online communities and groups provide a quick and easy way to communicate with a large number of developers across the world, there's still immense value in meeting fellow Windows Mixed Reality enthusiasts and developers face-to-face at a venue.

One popular resource for finding local meetups is the Meetup website, found at www.meetup.com. For example, I attend my local HoloLens/Windows Mixed Reality meetup group for the Seattle area, called the Windows Holographic User Group Redmond (WinHUGR). Figure 10-8 shows this group's page on Meetup.com. This particular meetup group boasts about 900 members, with an average attendance of about 60–80 people for the monthly meetings. Some meetup groups, such as the Austin HoloLens meetup may only have 5–10 attending on a weekly basis. Be sure to search Meetup.com to see if there are any Windows Mixed Reality groups near you. If not, use some of the other online communities mentioned earlier in this chapter to find and gather a few individuals near you and form a Meetup group.

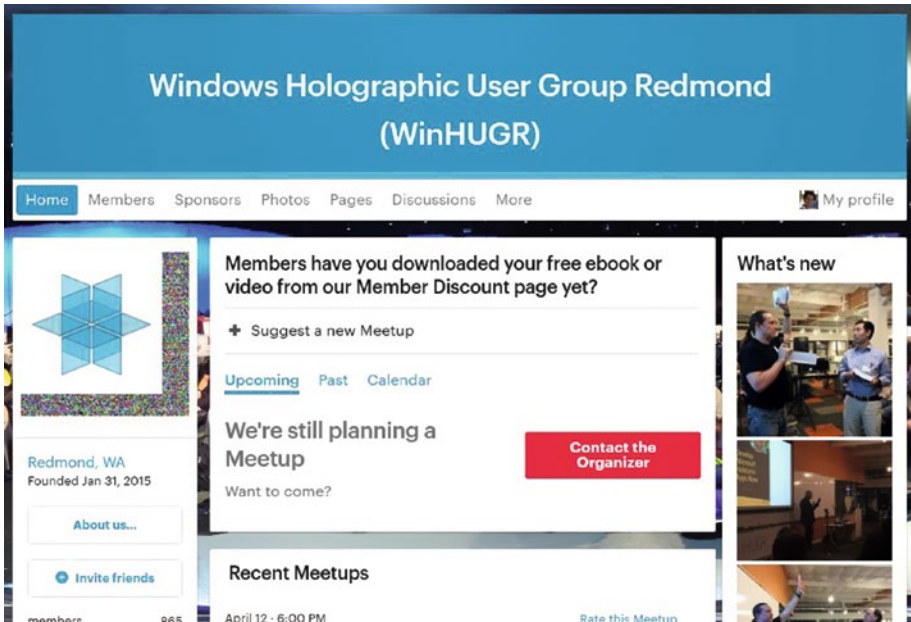


Figure 10-8. Meetup.com web page for a local HoloLens/Windows Mixed Reality meetup group in the Seattle area

In the next section you'll find non-comprehensive lists of HoloLens and Windows Mixed Reality meetups for cities around the world. Some of these may be broader developer or VR/AR groups, but are known to have a one or more Windows Mixed Reality developers in the community.

Europe Meetups

- Finland, Espoo [HoloLens Meetup](#)
- France, Paris [NUI Day](#)
- France, Paris [Mixed Reality Paris](#)
- Germany, Berlin [AR-VR Tools & Tech](#)
- Germany, München [HoloLens Meetup Germany](#)
- Ireland, Dublin [3DCamp Dublin](#)
- Ireland, Galway [3DCamp Galway](#)
- Netherlands, Amsterdam [Virtual Reality Amsterdam Meetup](#)
- Russia, Moscow [Moscow HoloLens Meetup](#)
- Sweden, Stockholm [Coding After Work](#)

- Switzerland, Zürich [NUI World](#)
- Turkey, Istanbul [HoloLens Development Meetup](#)
- UK, Belfast [Immersive Tech NI](#)
- UK, Brighton [VR Brighton – Digital Catapult Brighton](#)
- UK, London [Mixed Reality London](#)
- UK, London [Augmenting Reality](#)
- UK, London [VR London](#)
- UK, London [London HoloLens User Group](#)
- UK, London [London Virtual Reality Developer Meetup](#)
- UK, London [Let's Get Real! The Future of Augmented & Virtual Reality](#)
- UK, London [Unfold UK \(Women/Diversity in VR\)](#)

North America Meetups

- Canada, Toronto [HoloLens Meetup](#)
- USA, Austin [Austin Microsoft Developers](#)
- USA, Austin [Austin HoloLens meetup](#)
- USA, Boston [VRARA Boston](#)
- USA, Boston [Boston Unity Group](#)
- USA, Boston [Boston VR](#)
- USA, Boston [Boston AR/VR](#)
- USA, Dallas [Dallas AR/VR/MR UX & Development](#)
- USA, Knoxville [The Virtual/Augmented Reality Developers Network \(VARDNet\)](#)
- USA, Los Angeles [Los Angeles HoloLens Meetup](#)
- USA, Iselin [Microsoft Makers & App Devs of New Jersey](#)
- USA, New York [NYVR](#)
- USA, New York [NYC HoloLens Developers Meetup](#)
- USA, New York [Microsoft Makers & App Devs of New York City \(#MMADNYC\)](#)
- USA, Palo Alto [HoloLens Developer Meetup](#)
- USA, Philadelphia [VR Philly](#)

- USA, Philadelphia [Mid-Atlantic Mixed Reality User Group](#)
- USA, Portland [Portland HoloLens Meetup](#)
- USA, Reston [DC Holographic](#)
- USA, San Francisco [SFVR](#)
- USA, Seattle [Seattle VR Meetup](#)
- USA, Seattle [Windows Holographic User Group Redmond \(WinHUGR\)](#)
- USA, Silicon Valley [SVVR](#)
- USA, Washington DC [DC Metro Devs](#)

Asia Pacific Meetups

- Australia, Adelaide [Adelaide HoloLens Meetup](#)
- Australia, Sydney [Microsoft Events in Australia](#)
- Australia, Sydney [Women in Augmented Reality and Virtual Reality](#)
- China, Beijing [HoloLens User Group China](#)
- Japan, Tokyo [HoloMagicians](#)
- Malaysia, Kuala Lumpur [Microsoft Developer Malaysia](#)

Again, if you don't see your city or region represented in the preceding lists, be sure to check Meetup.com or perform a search on your favorite search engine to find meetups near you. Microsoft also maintains an updated list of community resources and meetups at <https://developer.microsoft.com/en-us/windows/mixed-reality/community>.

You may also consider joining or following the local chapter of the VR/AR Association. You may see a list of local chapters and chapter leaders at www.thevrara.com/team/.

Hackathons

A *hackathon* is an event where people come together for one or more days to quickly develop an application. A hackathon forces you to solve problems, leverage team members' expertise, and ask for help from experts. Hackathons are dear to my heart because a HoloLens hackathon I attended in the spring of 2016 was the catalyst that really propelled me forward as a HoloLens developer. Figure 10-9 shows a picture I took of the Seattle HoloHacks HoloLens hackathon I attended in 2016.



Figure 10-9. *The Seattle HoloLens hackathon was the event that propelled my career as a Windows Mixed Reality developer*

A hackathon will typically give you access to volunteers and experts who can help you out of tricky situations and show you optimal solutions to challenging development problems. Getting out of a coding problem that might take you a few hours of searching and reading online typically only takes a few minutes when you're able to have an expert show you what to do in person.

Attending a hackathon requires some commitment (typically a weekend) and stamina, but it's an extremely valuable experience that you won't get anywhere else. I highly recommend that you find a relevant hackathon, even if you must travel to attend it. Relevant hackathons include HoloLens, Mixed Reality, Virtual Reality, and Augmented Reality hackathons. VR and AR hackathons will typically include a healthy number of HoloLens and Windows Mixed Reality devices and developers.

Hackathons are often planned a few months in advance. The best place to find a hackathon is on a community calendar of your local Meetup group. You may also find them occasionally advertised on any of the online community groups I mention in this chapter. You can always ask members of your local or online community groups if they are aware of any upcoming hackathons, and you'll be sure to get several responses for a range of hackathons.

If you'd like to read more about my Seattle HoloLens hackathon experience to get a feel for what to expect, feel free to look at my blog post on the event at www.mrseanong.com/video-blog/my-experience-at-the-holohacks-seattle-hololens-hackathon.

Notable Industry Events

Industry events and conferences are an excellent way to keep a pulse on the Mixed Reality industry. Conventions and expos give you a chance to educate yourself during informative sessions, experience countless demos in person, build your network, and have a chance to meet up with people you'd typically communicate with online in your community groups.

There's no shortage of conferences, conventions, expos, and other events locally, nationally, and globally. As I mentioned with hackathons in the previous section, you can find out about upcoming events through community calendars and through your local or online-based groups.

I've also compiled a list of notable industry events, recognized for being especially important for HoloLens and Windows Mixed Reality developers:

- *Unity Vision Summit*: The annual Unity Vision Summit is a popular event focusing on VR/AR content creation in Unity. Because Unity is the preferred platform for Windows Mixed Reality development, this conference is highly relevant to developers. Microsoft and Unity have worked together to ensure that Windows Mixed Reality is a big area of focus at this conference. A link to the 2017 Unity Vision Summit can be found at <https://visionsummit2017.com>.
- *Augmented World Expo*: The Augmented World Expo (AWE) is the largest AR and VR event in the world. AWE includes a range of Mixed Reality technologies, with a focus on augmented reality. I attended the 2017 AWE and was surprised to see that a vast majority of booths in the expo featured HoloLens experiences. AWE is also known for being more enterprise- and commercial-focused. Many other VR/AR conferences tend to be social- and gaming-focused. Learn more about AWE at www.augmentedworldexpo.com.
- *Microsoft Build*: Microsoft Build is Microsoft's premier event for developers of Microsoft's various software and hardware products. Microsoft typically includes academy sessions on Windows Mixed Reality, makes major Mixed Reality announcements, and provides in-depth sessions on a wide range of topics. Microsoft Build tickets typically sell out within minutes, but keynote sessions are streamed online, and all sessions are made available on demand for free after the event. Find out more about Build at <http://build.microsoft.com>.

Summary

Congratulations! Not only have you made it to the end of this chapter, you've also completed the book. In this chapter, I walked you through ways you can keep informed and stay connected as a developer. I introduced the best online communities to participate in, the most informative sources of Windows Mixed Reality news, ways to get involved in person with local groups, and notable events and hackathons that you can attend. As mentioned at the beginning of this chapter, being plugged into a community of developers is imperative, especially because Mixed Reality is still an emerging field with a lot of best practices still being learned by development community.

Our journey with this book may have come to an end, but your journey as a developer is just beginning. There are countless ways to sharpen your skills as a developer, from mastering Physics in Unity to being a shader optimization pro. There's still so much to learn and so much that's yet to be discovered in the world of Mixed Reality.

I wish you the best on your new adventure and cannot wait to see all that you will create. Now, let's start building our holographic future together!

Index

■ A

- Air-tap gesture, [102–104](#)
- Albedo setting, [42](#)
- Asia Pacific meetups, [214](#)
- Audio occlusion, [90](#)
- Augmented World Expo (AWE), [216](#)
- Awe-inspiring experience
 - capstone project, [176–177](#), [179–194](#)
 - culling, [162–163](#)
 - design
 - additional resources, [174](#)
 - avoid, [174](#)
 - bounding box, [173](#)
 - colors, [174](#)
 - distance from user, [171](#)
 - shadows, [171](#)
 - sharp text, [172](#)
 - spatial mapping, [169–171](#)
 - toolbars, [173](#)
 - voice, [172](#)
 - Vuforia, [174–176](#)
- Device Portal settings
 - CPU, [158](#)
 - frame rate, [158](#)
 - GPU, [158](#)
 - I/O, [158](#)
 - memory, [159](#)
 - network, [159](#)
 - SoC power, [158](#)
 - system power, [158](#)
- features, [155](#)
- holographic remoting, [166](#)
- HoloLens's Settings app, [157](#)
- LOD, [162](#)
- monitoring and optimizing, [159](#)

- polygons and textures, [159–161](#)
- shaders, [165](#)
- Simplygon, [165](#)
- single-pass instanced, [164](#)
- stabilization plane, [166–168](#)

■ B

- Billboard script, [91](#)
- Bounding box, [173](#)

■ C, D

- Capstone project
 - add cursor, [179–180](#)
 - add spatial sound
 - effects, [191–193](#)
 - ballmanager.cs, [181–183](#)
 - configure InputManager, [179](#)
 - create responsive ball, [180–183](#)
 - download assets, [184–186](#)
 - HoloToolkit
 - settings, [177–178](#)
 - Unity package, [176](#)
 - ideas, [194](#)
 - lava scene
 - add ability to move, [189–190](#)
 - create, [186–189](#)
 - spatial mapping
 - add and configure, [190–191](#)
- Channels, [203](#)
- CPU, [4](#)
- Culling
 - frustum, [162](#)
 - occlusion, [163](#)
- Cursor test scene, [100](#)

■ E

Emitter game object, [146–147](#)
Europe meetups, [212–213](#)

■ F

Facebook group, [206–207](#)
Field-of-view (FOV)
 limitation, [12](#)
Fragments applications, [170](#)
Frame rate, [156, 158](#)
Frames per second (FPS), [90, 156, 159](#)
Freelancing
 finding opportunities, [197–198](#)
 winning contract
 keep in touch, [199](#)
 portfolio, [198](#)
 submit proposal, [199](#)
Freemium, [196](#)
Frustum culling, [162](#)

■ G

Game objects, [35](#)
Gamepads, [113](#)
Gaze
 code elements, [107](#)
 input method, [95](#)
 tutorial
 try scene, [97](#)
 understand scene, [98–99](#)
 use, [100](#)
 Unity scene, [96](#)
Gestures
 input method, [95–96](#)
 tutorial
 cursor prefab, [107](#)
 distance scale, [106](#)
 enter and exit, [104–105](#)
 gaze code elements, [107](#)
 HandDraggable.cs, [107](#)
 host transform, [106](#)
 implement, [107](#)
 InputManager.cs, [107](#)
 load test scene, [101](#)
 move objects, [105–106](#)
 OnFocusExit(), [104](#)
 select gesture, [102–104](#)
 try test scene, [101–102](#)
 use air-tap, [102–104](#)

GitHub, [92](#)

Graphics card, [5](#)

Graphics processing unit (GPU), [5](#)

Guide users, [151](#)

■ H

Hackathons, [214–215](#)
HandDraggable.cs, [105](#)
Headsets
 transparent *vs.* immersive, [11, 12](#)
Heads-up display (HUD), [174](#)
High-poly models, [160](#)
HoloDevelopers slack team
 contains, [204](#)
 founder, [204](#)
 general tips, [205–206](#)
 HoloLens and Windows
 Mixed Reality, [204](#)
 joining, [205](#)
 participating, [205–206](#)
 semi-official, [204](#)
Hologram
 cube
 create, [60](#)
 move from camera, [61–62](#)
 resize, [62](#)
 zoom, [60–61](#)
 HoloLens, [64–68, 70–72](#)
 persistence, [10, 137–138](#)
 test, [63](#)
Holographic
 building apps, [202](#)
 computer, [157](#)
 emulation, [5, 7](#)
 remoting, [6, 166](#)
 connect to HoloLens, [74–76](#)
 install on HoloLens, [73–74](#)
 test, [76](#)
 simulation, [7](#)
 advantages of, [77](#)
 connect controller, [77](#)
 enable, [77](#)
 test, [78](#)
Holo Herald channel, [211](#)
HoloLens
 cameras, [9](#)
 connect with holographic
 remoting, [74–76](#)
 emulator, [8](#)
 system requirements, [5](#)

- hologram, 64–72
- install holographic
 - remoting, 73–74
- Seattle hackathon, 214
- subreddit, 208–209
- testing with, 6
- Unity Forums, 208
- YouTube channel, 211
- HoloLens Developers Facebook
 - Group, 206–207
- HoloToolkit, 13
 - build, 91
 - defined, 55
 - downloading, 25–26
 - features and descriptions, 84
 - input, 85
 - Mixed Reality development, 57–59
 - new Unity project, 56
 - online
 - help and documentation, 92
 - repositories, 92
 - overview, 81
 - setup, 81, 83–84
 - sharing, 88–89
 - spatial mapping, 89
 - spatial sound, 90
 - spatial understanding, 89–90
 - test scene, 85–88
 - utilities, 90–91
 - version, 26, 82
- HoloToolkit-Unity, 25
- Host Transform, 106

■ I, J, K

- Immersive apps, 202
- Immersive headsets, 11
- InputManagerTest scene, 107
- Input methods, 95
 - gaze, 95
 - gestures, 95–96
 - hardware, 96
 - motion controllers, 96
 - voice, 96
- Inside-out tracking, 9
- Integrated graphics, 5

■ L

- Level of detail (LOD) rendering, 162
- Low-poly model, 160–161

■ M

- Matrix Inception's channel, 211
- Meetup.com
 - Asia Pacific, 214
 - Europe, 212–213
 - group's page, 212
 - North America, 213–214
- Microphone, 108
- Microsoft Build, 216
- Microsoft's Mixed Reality forum. *See*
 - Windows Mixed Reality Developer Forum
- Mixed Reality
 - freelancing, 197–199
 - opportunities, 199
 - Windows Store, 195–196
- Motion controllers, 113
 - input method, 96
- Multiple voice commands, 111

■ N

- Negative shadowing, 171
- Next Reality News, 209–210
- North America meetups, 213–214

■ O

- Occluder, 142–143, 145–146
- Occlusion
 - apply, 127–128
 - culling, 163
 - load TapToPlace scene, 127
 - try it out, 128–130
 - use in application, 130–131
- Operating system (OS), 4
- Outside-in tracking, 9

■ P, Q

- Persistence, 137–138
- Polygons
 - high-poly models, 160
 - low-poly models, 160–161
- Processor, 4

■ R

- Random Access Memory (RAM), 4
- Readme section, 92

Reddit, [208–209](#)
 RemoveSurfaceVertices script, [126](#)
 Roll-A-Ball tutorial, [30, 38](#)

■ S

Scenes, [85–88](#)
 Scripts, [45–46](#)
 Semi-official, [204](#)
 Shaders, [165](#)
 Shadows, [171](#)
 Sharing anchors, [138](#)
 Simplygon, [165](#)
 Single-pass instanced, [164](#)
 Slack, [203, 206](#)
 Spatial anchors, [136](#)
 Spatial mapping, [89](#)
 awe-inspiring
 experience, [169–171](#)
 hologram persistence, [10](#)
 HoloLens, [10](#)
 mesh, [118](#)
 navigation, [116](#)
 occlusion, [115](#)
 persistence, [115](#)
 physics, [116](#)
 placement, [115](#)
 spatial map, [10](#)
 tutorial
 set up Unity scene, [116](#)
 try it out, [117](#)
 understand scene, [118, 120](#)
 use in application, [120](#)
 Spatial plane
 set up Unity scene, [121](#)
 try it out, [122–123](#)
 Spatial processing scene
 load, [123–124](#)
 removing vertices, [126](#)
 SurfaceMeshesToPlanes.cs, [126](#)
 try it out, [124](#)
 understand, [125–126](#)
 use in application, [126](#)
 Spatial sound, [11, 90](#)
 avoid 2D sounds, [151](#)
 design considerations, [150](#)
 guiding users, [151](#)
 holograms, [141](#)
 increase immersion, [141](#)
 interactive experience, [141](#)

minimize artificial sounds, [152](#)
 tutorial
 audio file, [146](#)
 enable, [148–150](#)
 loop, [146](#)
 max distance, [148](#)
 max object, [148](#)
 mute, [146](#)
 pitch, [147](#)
 play on awake, [146](#)
 priority, [146](#)
 set up Unity scene, [142](#)
 spatial blend, [147](#)
 spatialize, [146](#)
 test scene, [142–144](#)
 understand scene,
 [144, 146–148](#)
 update interval, [148](#)
 volume, [147](#)
 Windows [10, 141](#)
 Spatial understanding
 set up Unity scene, [131–132](#)
 SpaceVisualizer.cs, [135](#)
 SpatialUnderstandingCustomMesh.cs,
 [134](#)
 SpatialUnderstandingSourceMesh.cs,
 [134](#)
 try it out, [132–133](#)
 use in application, [134–136](#)
 Stabilization plane
 default plane distance, [168](#)
 defined, [166](#)
 draw gizmos, [168](#)
 general tips, [168](#)
 lerp power
 closer, [167](#)
 farther, [167](#)
 resources, [168](#)
 set, [167](#)
 SetFocusPointForFrame(), [168](#)
 target override, [167](#)
 track velocity, [167](#)
 use gaze manager, [167](#)
 use unscaled time, [167](#)
 Stabilization scripts, [99](#)
 Stereo rendering method
 multi pass, [164](#)
 single pass instanced, [164](#)
 Subreddit, [208–209](#)
 SurfaceMeshesToPlanes script, [126](#)

■ T

Tagalong script, 91
 Test scene, 85–88, 98
 gestures, 101–102
 voice, 108

■ U

Unity, 5
 ball
 add physics, 44–45
 create, 38–39
 raise position, 40
 rename, 39
 reset position, 39
 zoom, 39
 create new project, 30, 32–33
 defined, 29
 enable keyboard
 control, 45–49
 free *vs.* paid tiers, 29
 ground plane
 color ground blue, 40–44
 create, 34, 36
 rename, 36
 reset position, 36
 scale, 37
 zoom, 37
 holographic remoting, 6
 holographic simulation, 7
 HoloLens emulator, 8
 HoloLens Forum, 208
 installation, 18–25
 save scenes, 33
 testing, 50
 Vision Summit, 216
 Windows Mixed
 Reality, 12, 29
 Unity Editor, 32
 Unity scene
 spatial mapping, 116
 Unity Vision Summit, 216

■ V

Visual Studio
 defined, 13
 installation, 13–18
 Visual Studio 2015, 16–17

Visual Studio 2017
 Game development with
 Unity, 16
 Universal Windows Platform
 development, 16
 Visual Studio Community, 14
 Voice
 command tutorial
 add, 111
 best practices, 113
 load test scene, 108
 OnMakeSmaller(), 112
 OnMoveUp(), 112
 own project, 112
 try test scene, 108
 understand scene, 109–110
 input method, 96
 VREX.io, 198
 Vuforia
 defined, 175
 install, 175
 try it out, 175

■ W, X

Wiki, 93
 Windows 10, 4
 Windows Holographic Developer Forum.
 See Windows Mixed Reality
 Developer Forum
 Windows Holographic User Group
 Redmond (WinHUGR),
 211–212
 Windows Mixed Reality
 hardware, 9–12
 hardware options, 113
 Unity, 12, 29
 Visual Studio, 13
 Windows Mixed Reality
 Developer Forum
 advantages, 202
 disadvantages, 203
 Facebook groups, 207
 holographic apps, 202
 immersive apps, 202
 main page, 202
 Microsoft published
 experiences, 202
 projects, 202
 Windows Mixed Reality development, 4

■ INDEX

Windows Store

- free apps, [196](#)

- freemium apps, [196](#)

- free with ads apps, [196](#)

- HoloLens apps, [196](#)

- Mixed Reality applications, [197](#)

- paid apps, [196](#)

World anchor. *See* Spatial anchor

■ Y, Z

Young Conker applications, [170](#)

YouTube

- Holo Herald, [211](#)

- Matrix Inception's channel, [211](#)

- official HoloLens channel, [211](#)

- Sean Ong's channel, [210](#)