HOLOLENS

SUCCINCTLY

BY LARS KLINT

SUCCINCTLY E-BOOK SERIES

Syncfusion

www.dbooks.org

HoloLens Succinctly

By
Lars Klint

Foreword by Daniel Jebaraj



Copyright © 2018 by Syncfusion, Inc.
2501 Aerial Center Parkway
Suite 200
Morrisville, NC 27560
USA
All rights reserved.

Important licensing information. Please read.

This book is available for free download from www.syncfusion.com on completion of a registration form.

If you obtained this book from any other source, please register and download a free copy from www.syncfusion.com.

This book is licensed for reading only if obtained from www.syncfusion.com.

This book is licensed strictly for personal or educational use.

Redistribution in any form is prohibited.

The authors and copyright holders provide absolutely no warranty for any information provided.

The authors and copyright holders shall not be liable for any claim, damages, or any other liability arising from, out of, or in connection with the information in this book.

Please do not use this book if the listed terms are unacceptable.

Use shall constitute acceptance of the terms listed.

SYNCFUSION, SUCCINCTLY, DELIVER INNOVATION WITH EASE, ESSENTIAL, and .NET ESSENTIALS are the registered trademarks of Syncfusion, Inc.

Technical Reviewer: James McCaffrey
Copy Editor: Courtney Wright
Acquisitions Coordinator: Tres Watkins, content development manager, Syncfusion, Inc.
Proofreader: John Elderkin

www.dbooks.org

Table of Contents

The Story behind the *Succinctly* Series of Books **9**

About the Author **11**

Chapter 1 Digital Reality Introduction **12**

Virtual reality 13

Augmented reality 15

Mixed reality 16

Genuine holographic experiences 17

HoloLens 17

Spatial mapping 19

Universal Windows Platform 19

Developers 19

Users 19

Businesses 20

Chapter 2 Tooling **21**

Unity 3D 21

Installing Unity 3D 21

Starting a new project 22

Creating 3D objects 23

3D modeling 24

Visual Studio 25

Installing Visual Studio 2017 25

Windows 10 SDK 26

HoloLens Emulator 27

HoloLens 29

Chapter 3 Spatial Mapping 30

What is it? 30

Spatial coordinate system 30

Spatial anchors 31

Hologram navigation 32

Occlusion 32

Holographic placement 32

Room visualization 32

Implementing spatial mapping 32

Mixed Reality Toolkit 33

Chapter 4 Gaze 36

Gaze uses 36

Implementing gaze 37

Cursor object 38

Visual design and behavior 39

Gaze targeting 40

Chapter 5 Gestures 42

Introduction 42

Gesture frame 42

Tap 43

Bloom 45

Advanced gestures 45

Hold 46

Manipulation 47

Navigation 48

Chapter 6 Voice 50

Built-in voice commands 50

www.dbooks.org

Select ... 50
Place ... 51
Face Me ... 51
Bigger/smaller ... 51
Hey Cortana ... 51
See it, say it ... 52
Custom voice commands ... 53
Voice command design ... 54
User education ... 54
User feedback ... 55
Dictation recognizer ... 55
Grammar recognizer ... 56
Chapter 7 Sound ... 58
Spatial sound ... 58
Head-related transfer function ... 59
Creating spatial sound ... 59
Sound design ... 61
Mixing sound ... 62
Interaction ... 62
Spatial mapping and sound ... 63
Chapter 8 Hologram Quality ... 64
Frame rate ... 64
Identifying issues ... 65
Accuracy ... 65
Color separation ... 65
Drift ... 66
Jitter ... 66

Judder 66
Jumpiness 67
Swim 67
Stabilization plane 67

Chapter 9 Testing HoloLens Experiences 69

Device Portal 69
3D view 70
Mixed Reality Capture 71
Performance tracing 71
Processes 73
System performance 73
System tools 74
Performance 75
Environmental testing 75
Light 75
Sound 76
Movement 76
Environment 76
Creating a testing environment 77
Perception Simulation 77
Comfort 80
Power management 80

Chapter 10 Sharing Holograms 81

Designing shared experiences 81
Sharing style 81
User location 82
Timing 82

www.dbooks.org

Interaction 82
Sharing service 82
Placement 84
Chapter 11 Next Steps 85

The Story behind the *Succinctly* Series of Books

Daniel Jebaraj, Vice President
Syncfusion, Inc.

Staying on the cutting edge

As many of you may know, Syncfusion is a provider of software components for the Microsoft platform. This puts us in the exciting but challenging position of always being on the cutting edge.

Whenever platforms or tools are shipping out of Microsoft, which seems to be about every other week these days, we have to educate ourselves, quickly.

Information is plentiful but harder to digest

In reality, this translates into a lot of book orders, blog searches, and Twitter scans.

While more information is becoming available on the Internet and more and more books are being published, even on topics that are relatively new, one aspect that continues to inhibit us is the inability to find concise technology overview books.

We are usually faced with two options: read several 500+ page books or scour the web for relevant blog posts and other articles. Just as everyone else who has a job to do and customers to serve, we find this quite frustrating.

The *Succinctly* series

This frustration translated into a deep desire to produce a series of concise technical books that would be targeted at developers working on the Microsoft platform.

We firmly believe, given the background knowledge such developers have, that most topics can be translated into books that are between 50 and 100 pages.

This is exactly what we resolved to accomplish with the *Succinctly* series. Isn't everything wonderful born out of a deep desire to change things for the better?

The best authors, the best content

Each author was carefully chosen from a pool of talented experts who shared our vision. The book you now hold in your hands, and the others available in this series, are a result of the authors' tireless work. You will find original content that is guaranteed to get you up and running in about the time it takes to drink a few cups of coffee.

Free forever

Syncfusion will be working to produce books on several topics. The books will always be free. Any updates we publish will also be free.

Free? What is the catch?

There is no catch here. Syncfusion has a vested interest in this effort.

As a component vendor, our unique claim has always been that we offer deeper and broader frameworks than anyone else on the market. Developer education greatly helps us market and sell against competing vendors who promise to “enable AJAX support with one click,” or “turn the moon to cheese!”

Let us know what you think

If you have any topics of interest, thoughts, or feedback, please feel free to send them to us at succinctly-series@syncfusion.com.

We sincerely hope you enjoy reading this book and that it helps you better understand the topic of study. Thank you for reading.

Please follow us on Twitter and “Like” us on Facebook to help us spread the word about the *Succinctly* series!





About the Author

Lars Klint is an author, trainer, Microsoft MVP, community leader, authority on all things Windows Platform, and part-time crocodile wrangler. He is heavily involved in the space of HoloLens and mixed reality, as well as a published Pluralsight author, freelance solution architect, and writer for numerous publications.

He has been a part of the software development community for the past 20 years and co-organizes the Developer Developer Developer (DDD) Melbourne community conference and developer events with Microsoft, and he also runs a part-time car restoration business. He has spoken at numerous technical events around the world and is an expert in Australian Outback Internet. He lives in Australia on a farm with his family, where they run a bed and breakfast.

www.dbooks.org

Chapter 1 Digital Reality Introduction

I had my first experience with virtual reality (VR) as a teenager, in 1993. I remember it very accurately, as it completely blew my mind. It was the game Dactyl Nightmare, and it was amazing. Looking at it today, it was a bunch of 8-bit blocks, some platforms, and a very uncomfortable headset you had to wear. I didn't care, though. That early experience of immersive computing has stayed with me since, and as the world is now entering a new era of virtual and augmented reality.

We have come a long way since 1993, and VR has developed into many shapes and forms. We now have augmented reality (AR) devices that overlay data on your physical world as viewed through your phone. VR devices are now much more accessible, such as the Oculus Rift, and there are devices that fit in neither or both categories, such as Microsoft's HoloLens, which exists in mixed reality (MR). In order to describe this whole segment of technology, regardless of what kind of reality, I have coined the term *digital reality* (no abbreviation though).

When discussing digital reality, it is important to understand the various segments, such as VR, AR, and MR. Not only is it important to know in which segment a potential project would fall, but you also want to target the right device with the right idea. For example, why create an experience for a $3,000 device that few people have access to, when the idea could work equally well on a $300 device, such as a smartphone? Understanding not only the development platform you are targeting, but also the users of this platform, as well as the other platforms in the same segment, is essential to creating a powerful and immersive experience that those users will buy into.

With that in mind, let's go through each digital reality segment to understand exactly where HoloLens applications fit in. As the following figure shows, digital reality has three main, defined areas: virtual, augmented, and mixed.

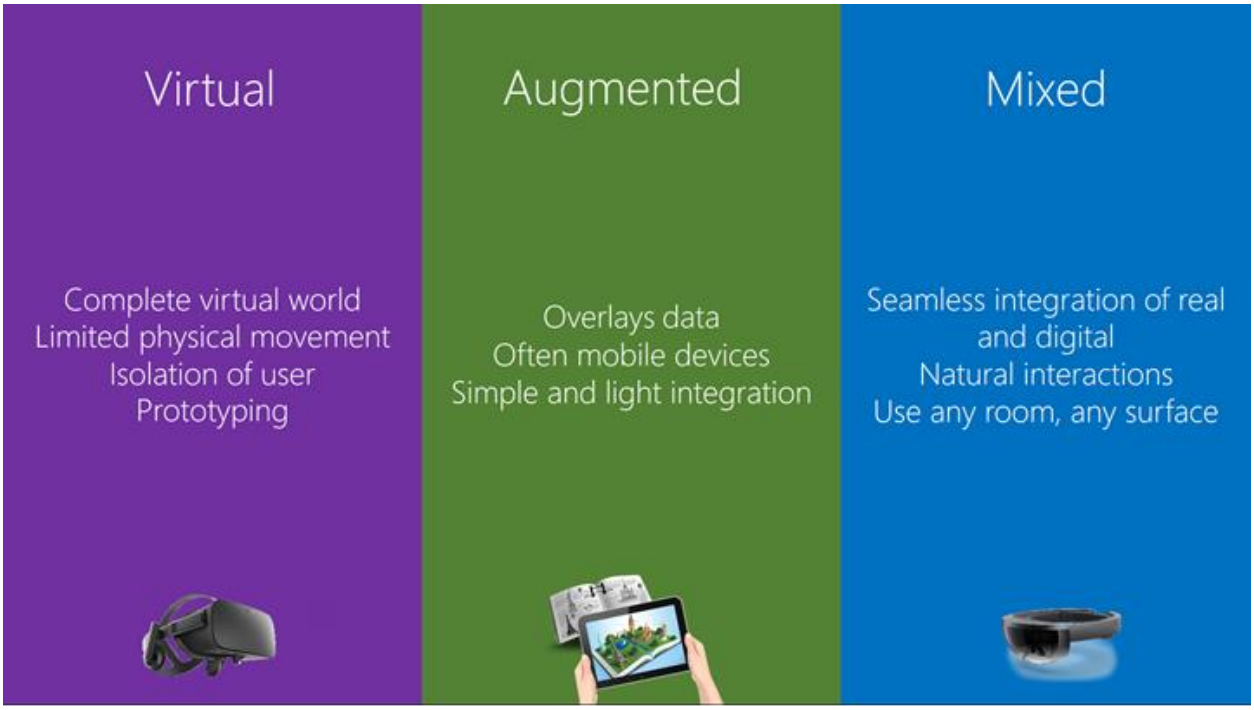


Figure 1: The Main Categories within Digital Reality

Virtual reality

Most people have heard the term virtual reality and associate it with either some form of gaming or a gimmick. The general public, and to some extent IT professionals, see it very much as the "flavor of the month" and pay it little attention. The truth is that VR was a billion-dollar industry in 2016[1] and is predicted to be a $150 billion industry by 2020.[2] It is a huge market, and the possibilities and opportunities will only get better and more elaborate.

Note: The first references to the concept of virtual reality came from science fiction. Stanley G. Weinbaum's 1935 short story "Pygmalion's Spectacles" describes a goggle-based virtual reality system with holographic recording of fictional experiences, including smell and touch.

[1] http://www2.deloitte.com/global/en/pages/technology-media-and-telecommunications/articles/tmt-pred16-media-virtual-reality-billion-dollar-niche.html

[2] http://www.digi-capital.com/news/2015/04/augmentedvirtual-reality-to-hit-150-billion-disrupting-mobile-by-2020/#.V8wFEih97gk

www.dbooks.org

The defining feature of VR is a complete virtual world created by the developer for the user. Every single component of the reality is virtual, and the user is removed from the physical reality they are in. This makes the experience incredibly immersive, as the developer is in control of the entire experience. It also means that developers must pay a lot of attention to all the little details. Managing motion sickness can be a real problem.



Figure 2: Oculus Rift VR Headset by Ats Kurvet (CC BY-SA 4.0)

Another limitation with VR is that the user can't physically move, because their entire reality has been replaced. Users moving about and crashing into objects is not a good look, so movement in VR experiences is generally done with handheld triggers. The movement must be simulated, and again, this leads back to managing motion sickness. If you make the movement too realistic, chances are you'll need a bucket for the user to throw up in. If you make it not realistic enough, users won't buy into the world you have created.

One strength of VR is the prototyping capabilities for scenarios that would be either expensive or dangerous to put inexperienced users through. Major motor manufacturing companies have been using VR for a long time to design and engineer their vehicles before having produced a single physical nut or bolt. You can also place users in locations they would never have dreamt of, such as in the center of the Melbourne Cricket Ground with 90,000 spectators.

VR is here to stay, no doubt about it. The technology has changed the landscape of digital reality, and it has come a long way from the early 90s and my first experience with it. More and more devices are being brought to market and the experiences are becoming more elaborate and immersive. Yet, by the sheer nature of blocking out the physical world, the VR experiences are limited in their reach. Although the applications will break boundaries and change the way software is viewed, VR does not cover every aspect of the digital reality space.

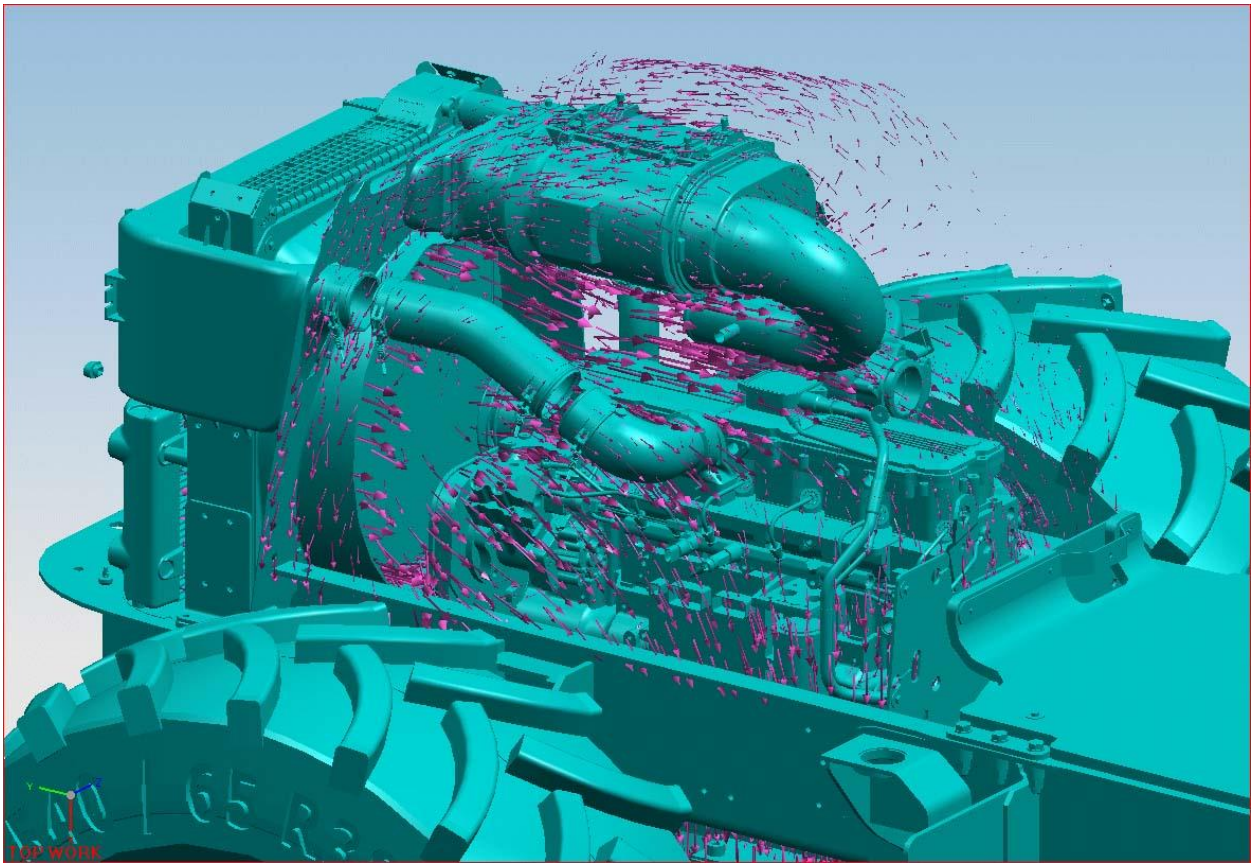


Figure 3: Engine Airflow Simulation (Screenshot of UGS NX 5, CC BY-SA 3.0)

Augmented reality

Whereas VR often requires dedicated hardware, AR can utilize any smartphone or tablet most of the time. It is a live direct or indirect view of a physical, real-world environment whose elements are augmented by computer-generated sensory input such as sound, video, graphics, or GPS data. Augmented reality experiences overlay data onto the physical experience, and thus enhance it (or at least, that's the idea).

Because the reality is an augmentation of the real world, experiences are often delivered through smaller handheld devices. When you're looking through this device, most commonly by using the camera and screen, the experience changes how you see the real world. This makes the technology very accessible for most people, and it's used in a large range of different ideas, from business cards that come to life to seeing through real walls in imaginary places.

www.dbooks.org



Figure 4: The Augmented Reality Game Pokemon Go (CC BY-SA 2.0)

Note: Pokemon Go is one of the most successful and popular augmented reality experiences.

Generally, AR experiences need a trigger point, which is a visual clue that will start the experience. Because this trigger point relies on a specific outline or figure, the experience is very limited and can't work just anywhere. That isn't to say that it doesn't enhance the user's world, but the limited "view" into the augmented reality combined with the trigger point makes for a sometimes clunky experience.

Mixed reality

The last digital reality we'll go through actually fits in between the first two. It's the newest, it is a combination of the first two to some extent, and it is the focus of this book. A term invented by Microsoft in conjunction with launching HoloLens, mixed reality (MR) is exactly what it sounds like: a mix of digital and real-world objects you can interact with in more-or-less natural ways. In a way, mixed reality is the best of both virtual and augmented realities, but it's also more complex. On the one hand, you can place any hologram you like in the real world, but then you also have to manage this hologram in both a digital reality and a physical reality.

Table 1: Digital Reality Comparison

	Augmented Reality	***Mixed Reality***	***Virtual Reality***
Real world augmentation	✓	✓	
Interactive Holograms		✓	
Virtual world experience		✓	✓
Replaces real world			✓

Genuine holographic experiences

Because of the futuristic connotations of placing and interacting with holograms in your physical space, designing and developing a true mixed-reality experience comes with its own set of pitfalls. It is easy to get caught up in the “wow” of the device and the capability, but often that results in either an experience that is lacking in longevity or an experience that is not uniquely mixed reality. Building apps that take over the entire reality are often better off being built as full-on VR apps, and apps that have a limited use case and simple “trigger and event” could be done with a smartphone as an AR app.

When designing for mixed reality, there has to be interaction between the holograms and the physical world. For example, you might make characters sit on the couch next to you, have robots enter the room through the actual walls, or create holograms sized to fit in whatever physical space they are placed in. If you place a holographic ball on a table, it should roll off if the table is moved.

HoloLens

The first commercially available mixed-reality device is the HoloLens from Microsoft. It is a device many years in the making, and it has managed to push the boundaries for digital reality, even in its prototype state. The HoloLens is a fully self-contained computer with bespoke hardware, such as the Holographic Processing Unit (HPU), which is in charge of calculating all of the 3D spatial computation in relation to holograms. The main difference, and distinct advantage, from other digital reality devices is that the HoloLens is untethered. There is no additional computing device required to run the mixed reality experience, which means the user is completely free to move about their environment. This is incredibly important, as the device is aware of its surroundings and physical position.

www.dbooks.org

If you place a hologram—be it on a wall, on a table, or floating in midair—it stays there pretty much forever until it's removed. Holograms persist in relation to the real world around them, and this is part of the experience. You can place a flat 2D hologram, such as a video clip, on a wall, walk to the next room, and when you come back it will be exactly where you left it. This is what the mixed reality experience is all about: a complete mix between digital and physical realities.

Some of the first demos we are seeing from Microsoft include Minecraft in 3D being placed in a physical space with interaction with 2D devices; a first-person shooter where robots come out of your walls and shoot at you; a full medical representation of the human body to allow medical students to learn about bones, muscles, and the nervous system; and much more. There are so many applications for the HoloLens that the imagination of the developers is the only boundary.

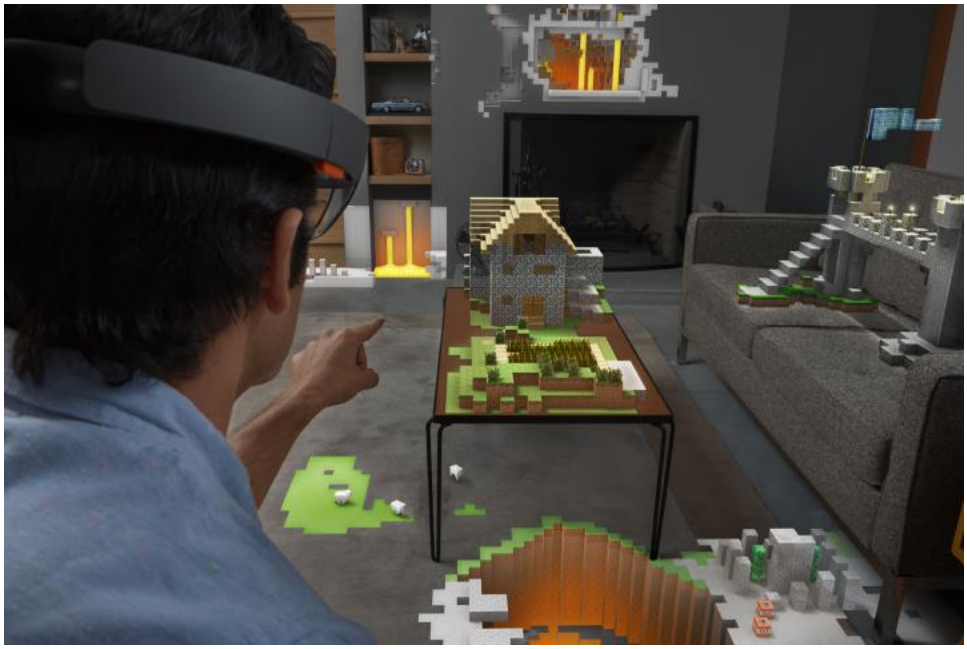


Figure 5: Minecraft on HoloLens (Microsoft Sweden—Win10-HoloLens-Minecraft, CC BY 2.0)

Although the hardware is cutting edge and unlike anything else, we aren't going to delve into the hardware side of things in this book. We will focus on how to create the mixed reality experiences from a software perspective.



Figure 6: Microsoft HoloLens (Microsoft Sweden—CC BY 2.0)

Spatial mapping

To allow the HoloLens to place holograms on physical surfaces and interact with the real world, spatial mapping is key. To understand how it works, see Chapter 3.

Universal Windows Platform

One of the key features of the HoloLens is that it runs on Windows 10. As amazing and futuristic as the holograms and the mixed reality experiences are, the fact that the device runs the latest Microsoft operating system means the device is also running the Universal Windows Platform (UWP). All Windows 10 devices—from Xbox, to PC, to HoloLens—conform to the contract with developers that is the UWP. This has great advantages, and not just for developers.

Developers

The UWP brings a common set of APIs and tools to creating apps for any device on the Windows 10 platform. As we will see in the next chapter, the tools are first class, and you can use the typical programming language from the Microsoft stack: C#. Developers can use the same approach to developing for any device that runs on Windows 10. There are of course platform-specific differences within the UWP, but as a base there is a Core API that is guaranteed between platforms.

This core API layer is also the reason it is so easy to publish apps across platforms. You will have to do the work according to each platform, such as managing mobile connections on smartphones, addressing a lack of a physical screen on IoT devices, and handling 3D on HoloLens. For HoloLens specifically, you can also publish 2D apps that will appear as flat apps in a 3D space when used.

Users

Although this book is primarily for developers, let us not forget that we are ultimately writing software for the users. The UWP is a big advantage for users as well. They can easily use the same profile across devices if the developer has set the service up, and users can also download the same app for any device that is supported. If they buy it once, they can download it anywhere.

In general, users don’t care how something works technically, but they do appreciate convenience and ease of use. The UWP creates a seamless experience across any Microsoft device that runs on Windows 10, and the user only logs in once on each device to set it all up. And of course, this works with HoloLens apps as well.

www.dbooks.org

Businesses

For businesses, the return on investment is greatly improved—mainly because of the reuse of code, but also the ease of development from using the same tooling for multiple platforms and form factors. Even the user interface is adaptive and can for the most parts be a single code base that works and conforms to the available screen space and platform-specific features.

The business use case is also not the focus of this book, but it is an important part of the decision to build UWP apps, whether they are for HoloLens or another Windows 10 device. Even existing apps that run on Windows 10 can be ported to HoloLens with little work, and businesses can profit on the UWP platform with little investment.

Chapter 2 Tooling

The tooling for creating MR experiences will be familiar to developers on the Microsoft stack, especially if you have done any mobile development. The tooling is based on Unity 3D for setting up the 3D space, and Visual Studio for managing code and business logic. There is also an as-real-as-possible emulator for developers who don't have a physical device.

The tooling is all free, which significantly lowers the barrier to entry. In fact, you don't *have* to have a physical HoloLens device to develop for it—although having one makes it a lot more fun! As you will learn in this chapter, the tooling can be downloaded and used instantly to create HoloLens apps for the UWP.

Unity 3D

Traditionally, a cross-platform game-development engine, Unity 3D is a tool for modeling a 3D experience, or series of experiences. It currently supports 27 different platforms and is a preferred development tool for companies like Nintendo and Microsoft HoloLens.

To develop experiences for HoloLens you will have to use Unity 3D to design what is known as the scene. Granted, you aren't required to use Unity to develop for HoloLens, but the way the integration and tooling is laid out, you will make your life a whole lot harder by not using it. For the remainder of this book, we will only use Unity to create and work on the scene in our app.

Installing Unity 3D

Getting ahold of Unity 3D is like any other piece of standard software. It is a simple installer for either Windows or Mac OS that you can download from the Unity 3D website.[3]

Select the default settings when going through the installation wizard and choosing the components to install. If you intend to develop Unity projects for other platforms, you will probably want to choose additional components.

[3] https://unity3d.com/

www.dbooks.org

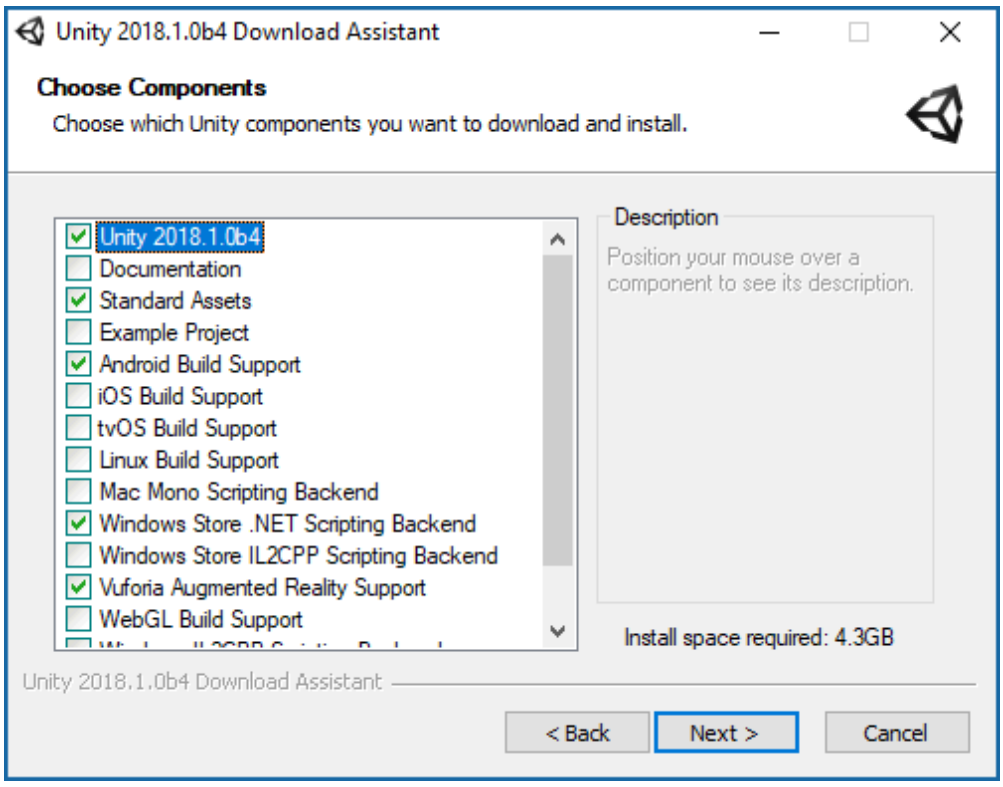


Figure 7: Unity 3D Installation Screen—Choose Components

The installer will download the required components, and once Unity is installed, it is ready to go.

Starting a new project

Creating a new project in Unity 3D is straightforward. You can either create a new project from the start screen, or you can do it from the **File** menu once Unity is loaded successfully, which in turn will bring up the start screen.

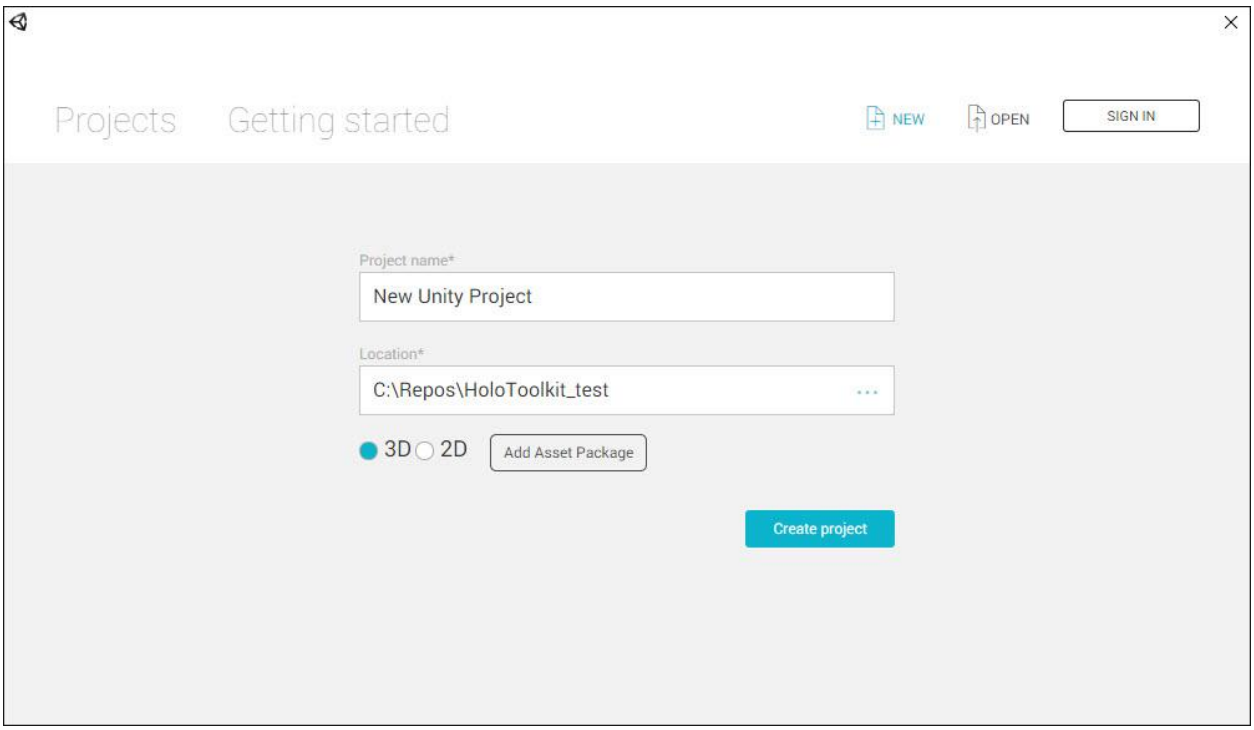


Figure 8: Unity Start Screen

If you have an online profile, you can sign in, and any projects will be linked to your profile.

Always choose the **3D** option for a new HoloLens project. This will set the scene view (your workspace) to 3D, set the lighting to conform to 3D, set the camera up correctly, and in general make sure the environment is 3D ready.

Creating 3D objects

One of the main aims of HoloLens development and creating mixed reality experiences, is to provide natural experiences for the end users. Part of this goal is achieved through realistic and believable 3D models. If you are anything like me (a developer with no design skills whatsoever), you will be dependent on 3D models from asset stores, designers, and other places. While you wait for artwork to arrive, use basic Unity 3D shapes as placeholders. Once you get the real asset, you can then replace it for the simple shape and write the scripts and behaviors for the real model.

www.dbooks.org

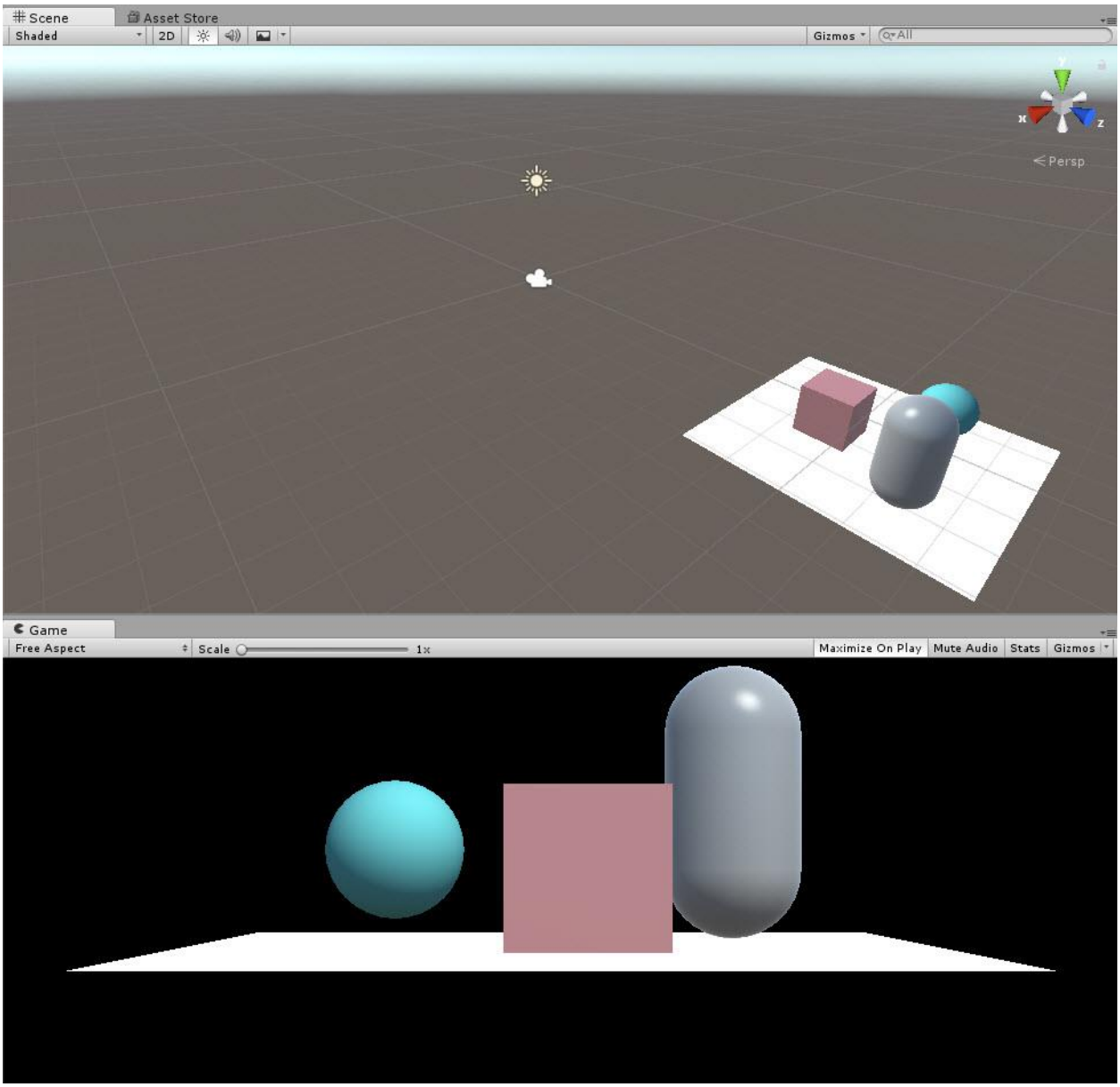


Figure 9: Unity 3D Objects

3D modeling

Building and designing 3D models is a job that goes well beyond software development. If you aren't a designer or someone with a good understanding of 3D objects and models, it is best to leave this to someone who is. There are many ways to get ahold of great 3D models for your projects, such as searching the Unity 3D asset store and other online 3D asset stores, using a freelance modeler, or just teaming up with someone who does 3D visual work.

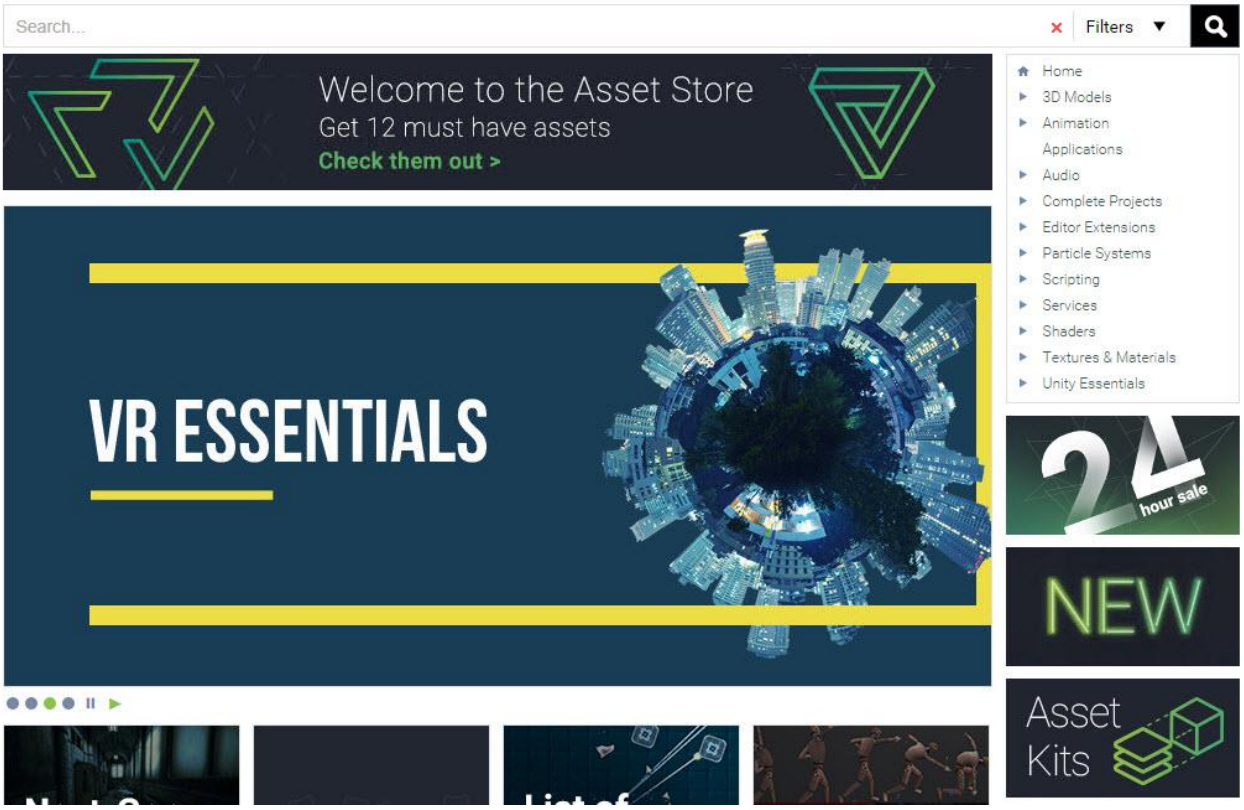


Figure 10: Unity Asset Store

Visual Studio

As with Unity, if you are doing HoloLens development, it is extremely difficult to get around Visual Studio as the primary editor and tool for development. If you haven't used Visual Studio before, you are in for a treat. At first glance, it is daunting and somewhat confusing, but the depth and thought put into the product are amazing. It is the tool for developing anything on the Microsoft stack (and other non-Microsoft technologies).

Note: This book describes developing mixed reality experiences with Visual Studio 2017.

Installing Visual Studio 2017

The community edition of Visual Studio 2017 is free and fully supports HoloLens development. You can download it from here, then install it in as little as 15 minutes and be on your way.

www.dbooks.org

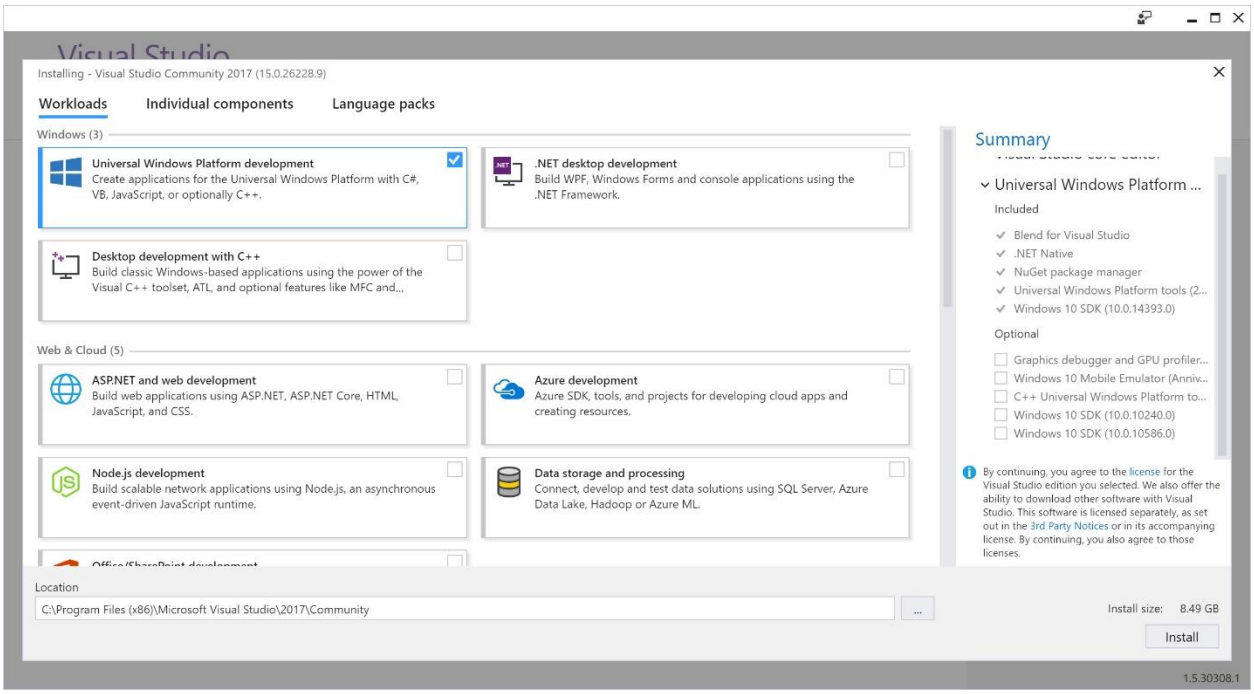


Figure 11: Visual Studio 2017 Installation

Windows 10 SDK

During the installation, select the **Universal Windows Platform development** package in the **Workloads** tab. This is what you need to build and compile Windows Holographic apps. Because the HoloLens itself runs on Windows 10, the apps you build are Universal Windows Platform apps (as mentioned in Chapter 1), and the Windows 10 SDK gives you the latest headers, libraries, and tools for building Windows 10 apps. If you already have VS 2017 installed, launch the Visual Studio Installer program, click **Installed** > **Modify**, and then select the **Universal Windows Platform development** package.

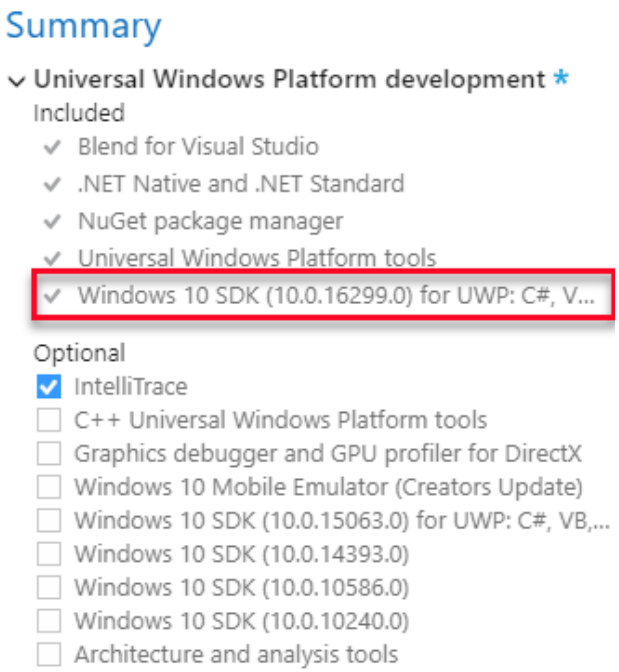


Figure 12: Select Windows 10 SDK

HoloLens Emulator

Chances are that you might not have a physical HoloLens. First of all, they aren't readily available in many countries, and second, they are very pricey. Unless you have a specific purpose or project for buying a HoloLens (or someone else is paying), it's not an item you "just get." Luckily, the HoloLens emulator is completely free. It is as close as you can get to having a real HoloLens without having one.

> ***Note: The HoloLens emulator is the main reason you need a version of Windows 10 with Hyper-V. If you have a physical HoloLens, Hyper-V is not needed, but does make development easier.***

www.dbooks.org



Figure 13: HoloLens Emulator

The emulator comes with tooling to help you emulate a real device as much as possible. Apart from the device portal (see Chapter 9 “Testing HoloLens Experiences”), there is a whole second tools screen for the emulator input and output. You can monitor the simulation coordinates of the actor in the emulation for body, head, and hands (gestures). You can also change the spatial mapping of the location (in other words change the room layout), and use a Microsoft account to log into the emulator and view network configuration values.

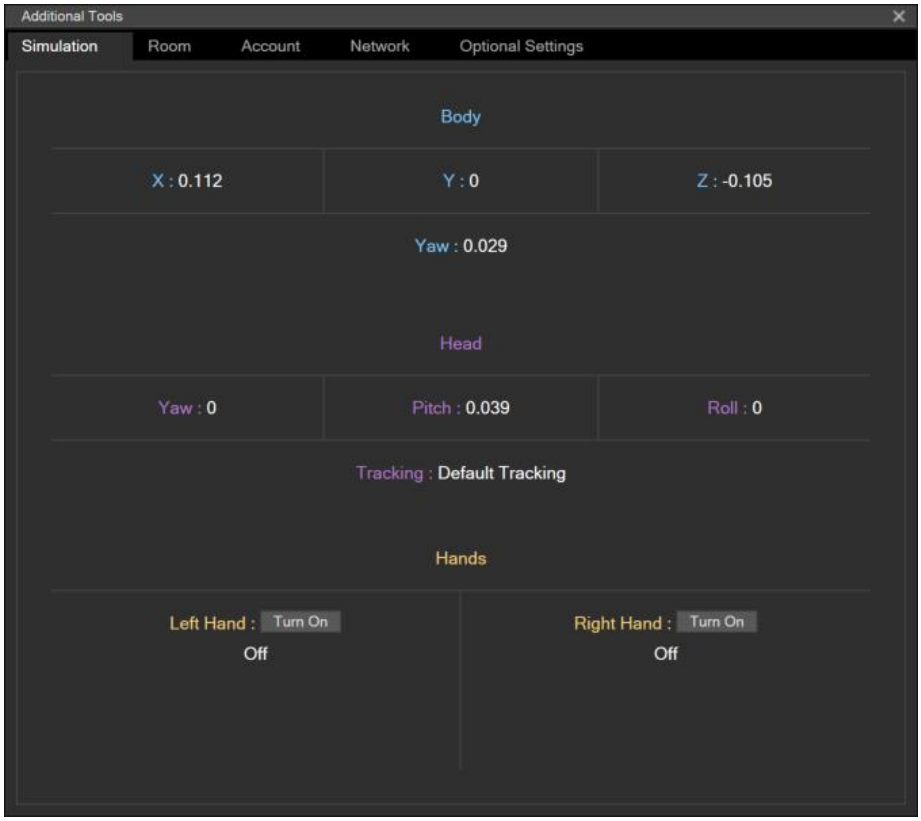


Figure 14: Emulator Tools—Simulation

HoloLens

We are talking about HoloLens development, and of course, having a physical HoloLens is the best part of it all. The device is groundbreaking, and experiencing the mixed reality in person is truly unique. There is no reason you can't develop HoloLens apps without a physical device, but having one available does make the whole experience a lot more enjoyable. If you do develop without using a device, I would recommend testing your app on a real device before releasing it to the wild.

www.dbooks.org

Chapter 3 Spatial Mapping

What is it?

Spatial mapping is what makes the HoloLens stand out from the other augmented reality devices as a mixed reality device. The HoloLens uses four environmental cameras on the front of the device to map the physical surroundings and build up a 3D model of the real world. As you use the device, it continually creates a spatial mapping model of the space you are in and updates any existing mapping. The cool thing is that the developer portal that comes with the developer tools allows you to see this 3D spatial mapping in real time, and it works for both physical HoloLens devices and the emulator.

Spatial coordinate system

In order to create a convincing illusion of digital and physical assets interacting, such as having digital characters sitting on physical furniture, holographic apps must be aware of both digital and physical realities (hence the name “mixed reality”). This requires very precise positioning of the holograms in the world in a way that makes completely natural sense to the user. Spatial coordinate systems are used to calculate this interaction.

To understand the complexity of spatial mapping for HoloLens, consider a similar coordinate system for virtual reality. Because you as the developer control the entire environment, you also know how far objects are from each other, as you have programmatic control over the objects themselves. You have a single master-coordinate system that all objects can map to and relate to. This means that you can very accurately relate the information to the user, and thus the experience is very stable and precise. The user expects the objects to relate to each other in the same way throughout the experience.

The user expectations are the same for mixed-reality apps, but because the digital assets are dependent on the spatial mapping done by the HoloLens to relate accurately to each other, there is a synchronizing challenge. Not only do the holograms position themselves in real time as the grid is updated, but the mapping might also be off between physical objects and digital assets. The units in the coordinate systems on HoloLens are always in meters, which makes it very easy to calculate where to place holograms in relation to each other and the real world. However, what if the spatial mapping is not accurate? If the initial environmental scanning is either inaccurate or incomplete, the holograms can be at different distances from each other or from physical objects as the spatial mapping is improved. This would cause holograms to start floating and shifting positions, which makes the experience uncanny and unnatural. To begin with, two objects are two meters from each other, and as the mapping updates, they are suddenly 1.8 meters from each other. At worst, they will make your users motion-sick, and your users will quickly abandon the experience. Luckily, there is a solution for this.

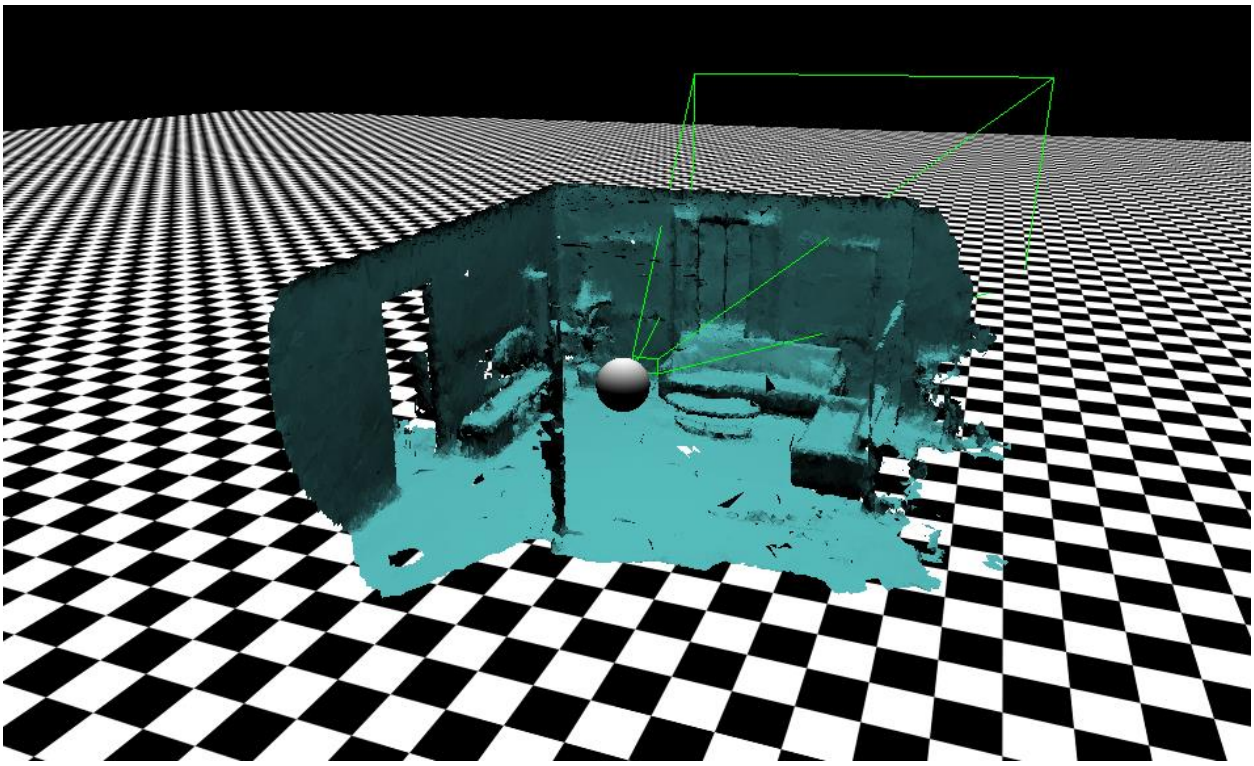


Figure 15: 3D View of Spatial Mapping.

Spatial anchors

To avoid drift and ensure that a hologram remains exactly at a specific spot in the world, even as the system discovers more about the world and updates the spatial map, you can place that hologram using a spatial anchor.

A spatial anchor represents an important point in the world that the system should keep track of over time. Each anchor has a coordinate system that adjusts as needed, relative to other spatial anchors or frames of reference, to ensure that anchored holograms stay precisely in place.

Rendering a hologram in a spatial anchor’s coordinate system gives you the most accurate positioning for that hologram at any given time. This comes at a performance cost of small adjustments over time to the hologram's position as the system continually moves it back into place relative to the real world.

Spatial anchors can be persisted between sessions and even shared between devices that share the same experience. This is done using the spatial anchor store for the specific app and providing a meaningful key to allow later identification of the anchor.

www.dbooks.org

Hologram navigation

Often, you want holograms to navigate your physical space as if they were part of it. This requires accurate spatial mapping to make sure you don't have people trying to walk through furniture and jump through walls. This also requires being able to identify surfaces, so people only walk on the floor and only sit on chairs and couches.

Occlusion

One of the simplest uses of spatial mapping, and one of the most important ones to create a natural interaction model, is to simply occlude holograms. By hiding holograms, or obscuring parts of them, you increase the perceived realism dramatically. You also create an expectation that holograms will collide with physical objects when they are solid. All together, this means simple occlusion is of great importance to effectively convey a realistic integration of physical and digital realities.

Holographic placement

To place holograms in a physical world, the HoloLens relies on spatial mapping to determine distance, space available, and usable surfaces. One of the concessions of using hand gestures to infer distance (the HoloLens interaction model is based on hand gestures) is that we normally only communicate direction with gestures. When this needs to be translated into distance as well, spatial mapping can help indicate to the user where holograms can be placed. Combining this with the shape of the hologram, placement adds another natural aspect to mixed reality.

Room visualization

At times, it may be necessary to visualize physical surfaces and structures to allow holograms to be placed on them accurately and realistically. You might also want to have holograms casting shadows on surfaces.

Implementing spatial mapping

Using spatial mapping in your HoloLens application is simpler than you might think. The first thing you do is enable the **`SpatialPerception`** capability for your Unity project. This will make 3D objects adhere to the 3D spatial-mapping model once you export the project from Unity to Visual Studio, and, in turn, to a HoloLens device.

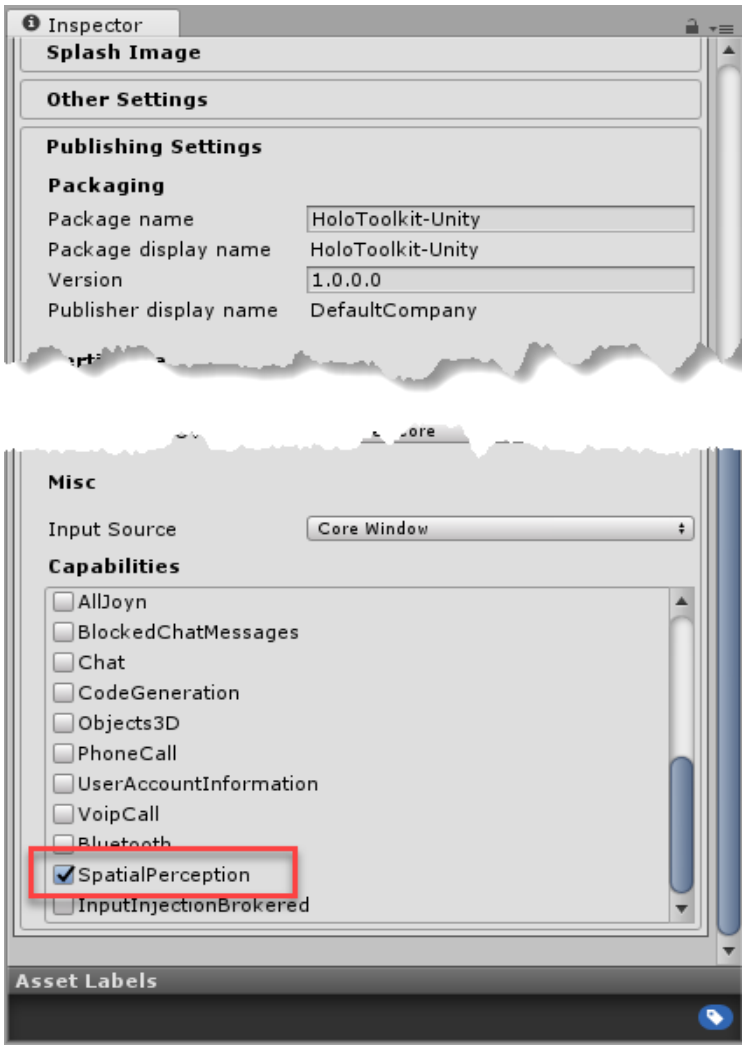


Figure 16: Unity Capabilities

Mixed Reality Toolkit

Created by the team that produced “Fragments” and “Young Conker,” the Mixed Reality Toolkit (formerly known as the HoloToolkit) is a collection of scripts and components intended to accelerate the development of holographic applications targeting Windows Holographic.



Figure 17: Mixed Reality Toolkit Logo

www.dbooks.org

The toolkit is free[4] and comes in two varieties: one for Visual Studio, and one for Unity 3D. Developing for HoloLens means spending a LOT of time in the Unity editor, and the HoloToolkit is a simple **`.unitypackage`** file you can download and import into your Unity project.

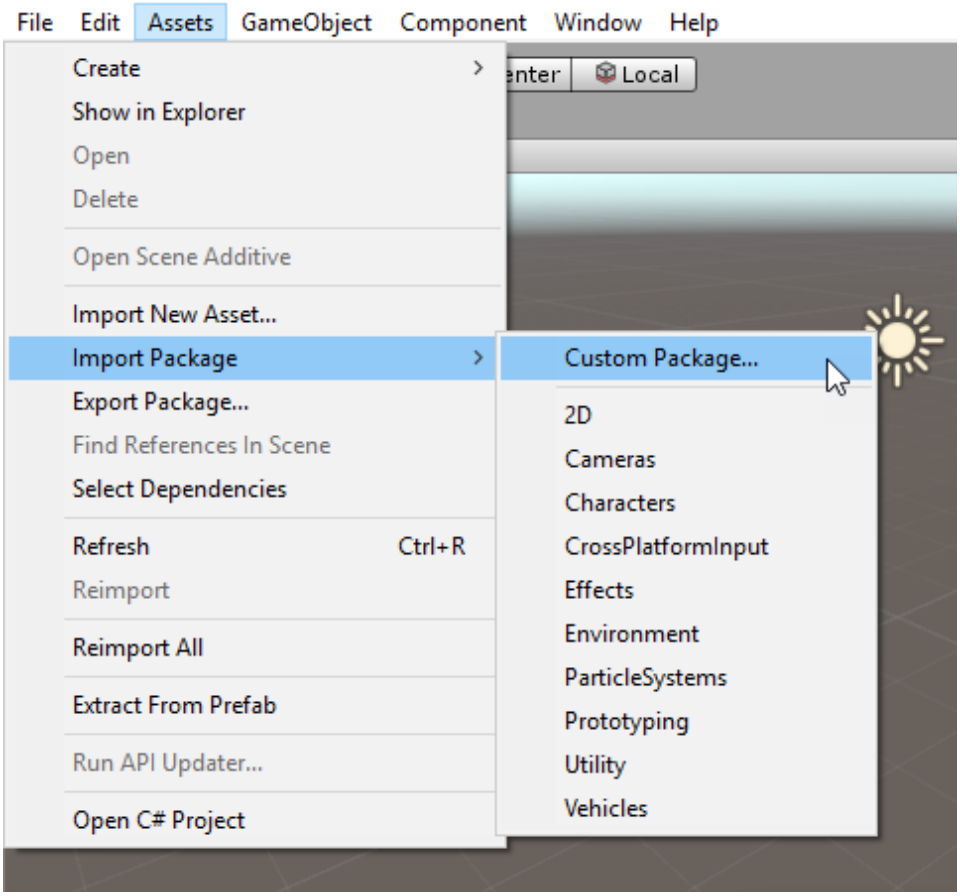


Figure 18: Mixed Reality Toolkit Install

This will show a new menu item, Mixed Reality Toolkit, in Unity.

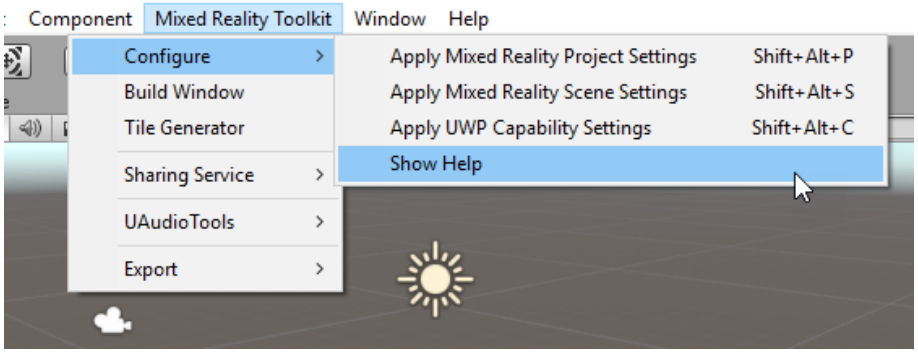


Figure 19: Mixed Reality Toolkit Menu

[4] Download from here.

These menu items form part of the visual change to Unity that HoloToolkit implements. Having menu items to perform the most trivial and basic configuration tasks for a new project gets you to the fun bits a lot quicker, and helps you make sure that you haven't missed anything.

Apart from the obvious menu item changes, the HoloToolkit also brings with it many scripts, prefabs, materials, shaders, and plugins you can use in Unity3D.

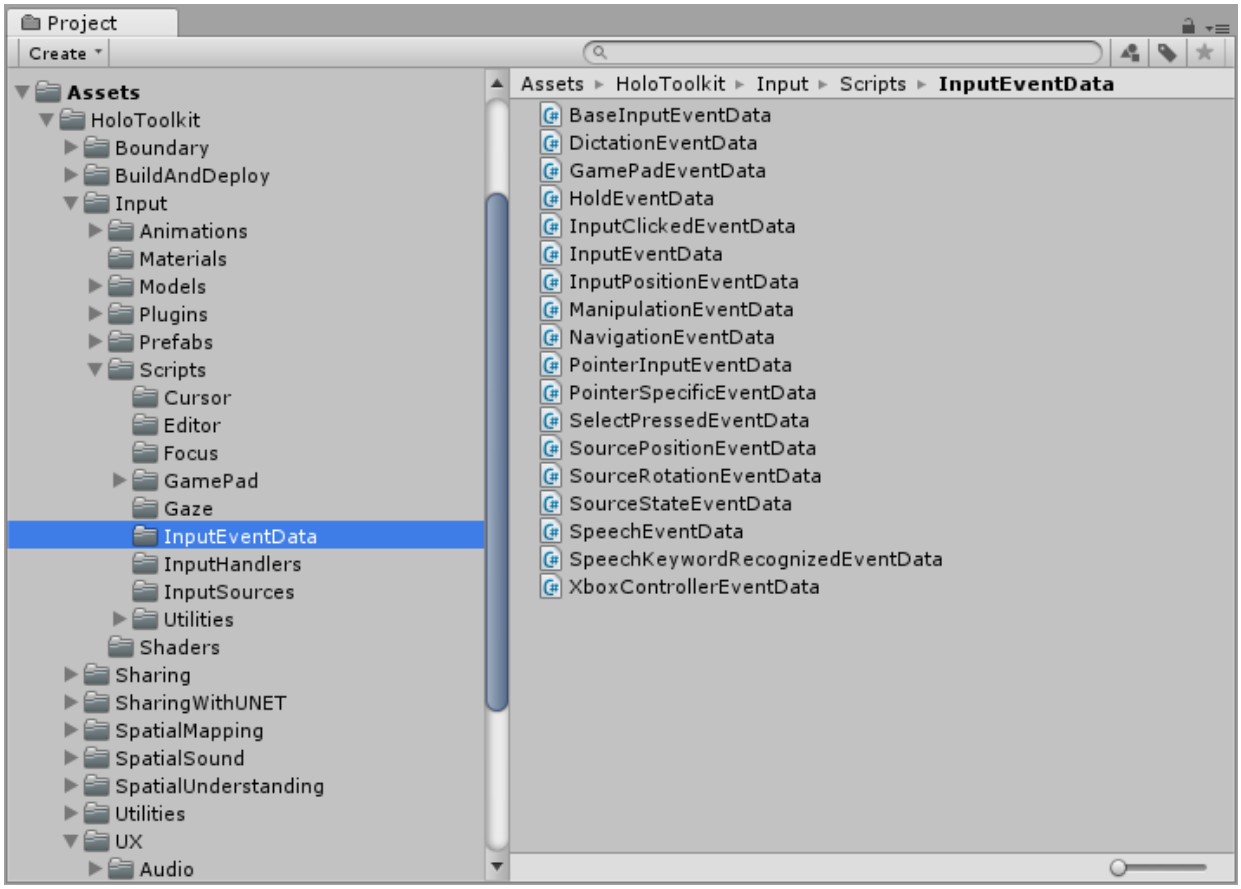


Figure 20: Mixed Reality Toolkit Contents

www.dbooks.org

Chapter 4 Gaze

The first of the main input methods for HoloLens apps is gaze. Gaze tells the app where the user is looking in the world, and similarly to the real world, users look at objects they intend to interact with. The HoloLens does not detect eye movement and eye activity; it is the position of the user's head that determines the direction, or vector, of the gaze. If you think in terms of traditional computing, you use your head as the mouse pointer to gaze at objects, a bit like if you had a laser pointer coming directly out of your head.

This "laser pointer" can intersect with the spatial mapping mesh described in the previous chapter, as well as any holograms you have already placed within your app. Interactions in any experience are based on the user's ability to target the element they want to interact with, and for HoloLens, this is most often done through gaze.

Gaze uses

Considering that gaze is one of the main inputs to a mixed-reality experience, it isn't surprising that there are several uses of gaze, some of which aren't immediately obvious.

- **Selection**: Users can select objects with gaze by intersecting the `RayCast` function.
- **Target**: Once a selection has been made, gaze assists in targeting the right gesture for the interaction.
- **Intersect**: You can intersect the user's gaze with the holograms in your scene to determine user attention. Holograms can react to the gaze when appropriate such as changing color on an object to identify it.
- **Combine**: Users can combine gaze with gestures (see Chapter 5) and allow more complex interactions.
- **Guide:** You can guide the user's gaze with visual clues to objects or areas they are *not* looking at, but should for the experience to be effective.

Gaze is the first of the standard input mechanisms for using the HoloLens, and the first way you are likely to interact with your users.

As the HoloLens user paradigm is very different from traditional computing, you are likely to have to hold the proverbial hand of your users for the first bit of your experience. Using your head as the mouse cursor is a shift from the way most people are used to interacting with computers and showing various tips to guide them along works wonders.

Implementing gaze

The code to implement gaze and gaze-handling is relatively simple. In fact, a lot of the code in this e-book is very basic, because if you have any experience with C# and Visual Studio, you are already halfway there. If you have no prior experience, the hardest part is getting the design right and mastering Unity 3D. With that in mind, Code Listing 1 shows the basic code to capture a gaze event. All the code in this book is meant to be part of a Unity 3D script that is referenced by or attached to a Unity object.

Code Listing 1: Simple RayCast Test (from Microsoft)

```
// Do a raycast into the world based on the user's
// head position and orientation.
var headPosition = Camera.main.transform.position;
var gazeDirection = Camera.main.transform.forward;

RaycastHit hitInfo;

if (Physics.Raycast(headPosition, gazeDirection, out hitInfo))
{
    // If the raycast hit a hologram...
    // Display the cursor mesh.
    meshRenderer.enabled = true;

    // Move the cursor to the point where the raycast hit.
    this.transform.position = hitInfo.point;

    // Rotate the cursor to hug the surface of the hologram.
    this.transform.rotation = Quaternion.FromToRotation(Vector3.up,
hitInfo.normal);
}
else
{
    // If the raycast did not hit a hologram, hide the cursor mesh.
```


www.dbooks.org

```
    meshRenderer.enabled = false;
}
```

Because the **camera** is the position of the user's eyes in Unity, you can use it to calculate the direction the user is pointing their head in. You then use the **RayCast** object to calculate if the gaze is intersecting any object, either in the spatial mapping 3D model, or another digital asset. If the **RayCast** returns **true**, then there is a hit, and you can get the 3D coordinates for the hit. In other words, you can get the location of the object the user is looking at.

RayCast is a construct often used in games in Unity to determine what a bullet might hit, where objects might land, if enemies are nearby, and more. It is a proven method to calculate relationships between assets, and for HoloLens this means you can determine with great accuracy where the user is turning their head to, and what they are looking at.



Note: HoloLens does not track a user's eyes—it only tracks head movement.

Cursor object

You aren't required to have a cursor object in your experience, but it is highly recommended. The cursor can be any 3D object in Unity, and you must manage it programmatically. You will typically position the cursor at the intersection between a real-world surface or hologram and the **RayCast** test. Not only is this a visual clue for users to let them know which element they are about to interact with, it also gives the user some confidence in the experience. By using the cursor, you are indicating to the user that you are reacting to their input.

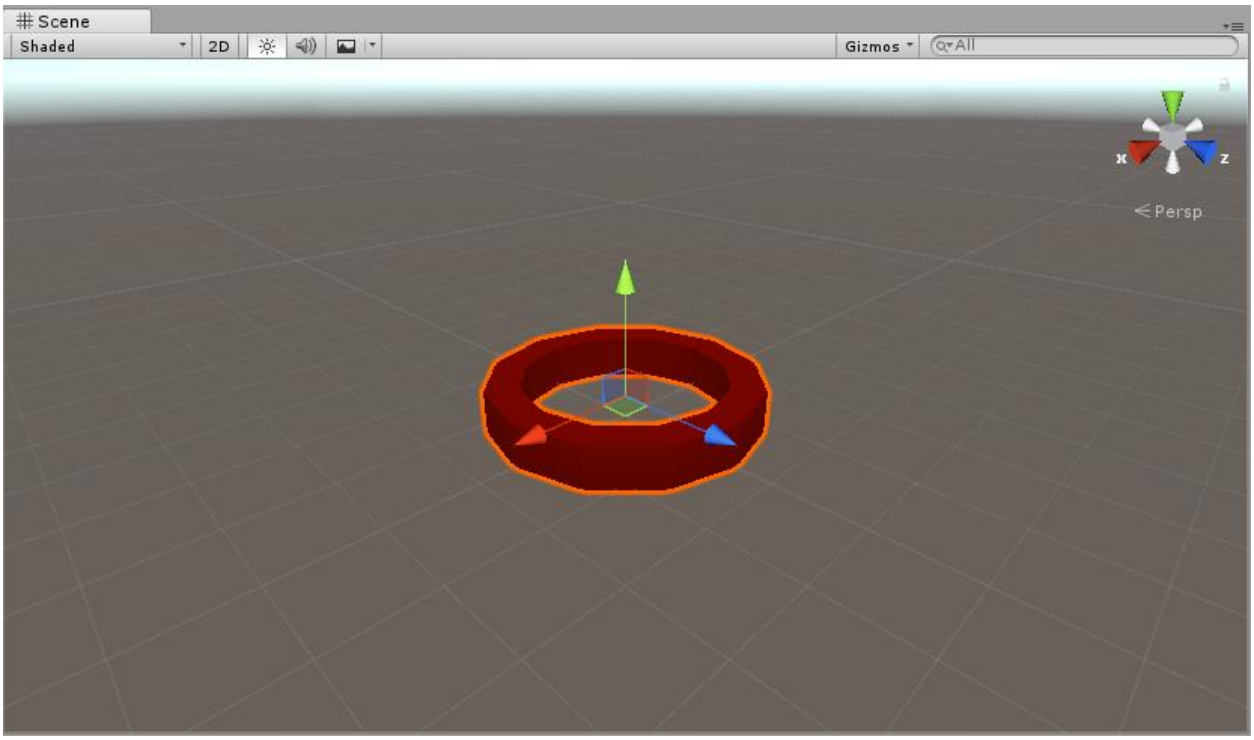


Figure 21: Cursor Object in Unity 3D

Although the cursor can be any 3D object, keep it simple to not detract from the parts of the experience that should be in focus. Once the user has targeted a hologram or real-world object with their gaze, you can visually indicate which actions are available for them.

Tip: If the cursor isn't needed at certain times in the experience, hide it. Less is more.

Visual design and behavior

The cursor can perform a lot more tasks than you might think. You can add feedback to the cursor and the holograms to let users know with more certainty which objects they are targeting with gaze. Changing the size of the cursor to indicate a specific target, or changing the color of an object when being targeted, are both great ways of guiding the user. When you code the experience, you can send events to the objects being gazed at by using the **`RayCast`** object information that is returned.

If you have an experience with a lot of smaller objects, it can often be difficult to accurately identify and target each of these objects individually. Using visual aids and helping the user select the appropriate object with gaze makes even small objects usable.

www.dbooks.org

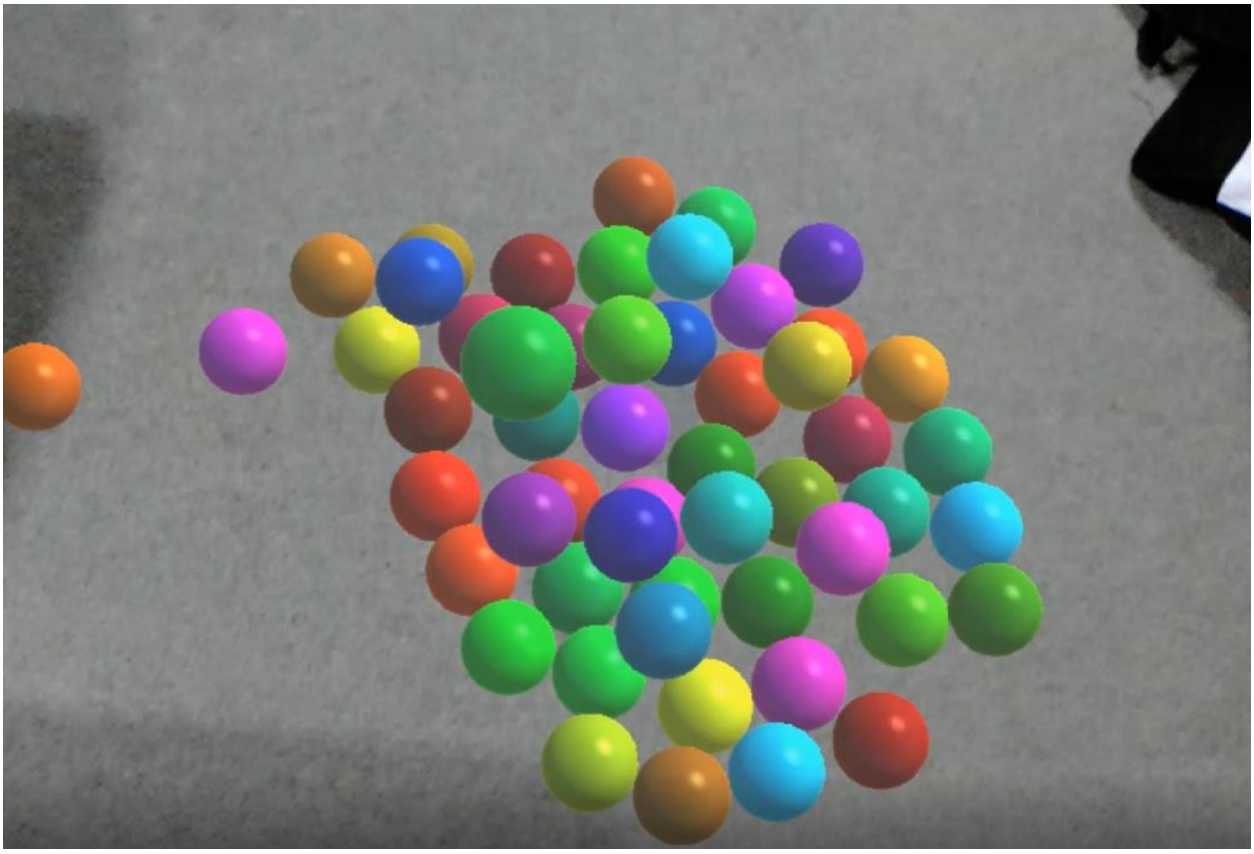


Figure 22: Targeting a Specific Ball with No Assistance

The main ways for users to interact with the object they are gazing at are by using either gestures or voice, which we will look at in the next two chapters.

You can also add audio hints when an important interaction occurs with the gaze. Audio in your HoloLens experience is covered in more detail in Chapter 7, “Sound.”

Gaze targeting

The aim of using gaze is to target objects that the user intends to interact with. When the targeting is accurate, meaning the expected outcome matches the real outcome, users will quickly and confidently get used to the gaze experience. This is where gaze targeting comes in.

Identifying what the user is gazing at does come with some limitations, but there are ways to ensure gazing is as accurate as possible. One of the issues is that users never hold their head still, and they never will (it is physically impossible for humans to do so). Another issue is that the further the distance to objects, the smaller they appear, and they harder they are to target.

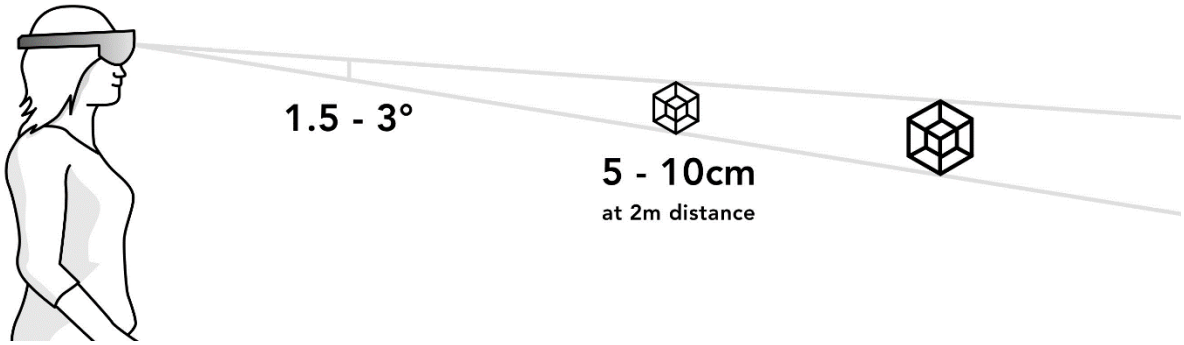


Figure 23: Gaze Target Optimal Distance and Size

You want targets to be within the 1.5–3 degree range of the gaze "cone" in front of the user. This obviously translates to larger objects regarding the further away they are, as illustrated in Figure 23. As mentioned previously, adding a cue in the form of audio or visual to help gaze targeting is extremely helpful.

There are many ways in which you can improve gaze targeting and help give users a great experience with your app.

- Place elements knowing this can have a big impact on the usability and, hence, gaze targeting of them. In general, elements placed very high or very low are harder to discover, as they are outside of the user's normal field of vision. Place important holograms in front of the user.
- Use gaze stabilization to negate the natural head and neck jerks and jitters a user may have. This technique implements "magnetism" to guide a user's gaze towards an object nearby. This is very effective and helps create a more reliable experience, too.
- Group related holograms together to form a natural way to progress through the experience.
- Assume intent based on what the user is doing. You can determine the closest hologram within a certain error margin, and, if appropriate, perform the necessary actions.
- Increase the size of the "hit boxes" that surround holograms. In Unity, the collider on an object determines when it is hit. If this area is slightly larger than the hologram, it makes for a smoother experience.

In general, gaze targeting is one of the most important aspects to get right in order to facilitate trust with the user.

www.dbooks.org

Chapter 5 Gestures

Introduction

If you have seen anyone use a HoloLens, you have probably noticed how users move arms and hands in the air. These gestures are the second type of input for HoloLens, and are the most recognized part of the Gaze-Gesture-Voice (GGV) input model used for mixed-reality experiences on HoloLens. As you'll see, the implementation of gestures is not difficult from a programming perspective, but you will have to be very particular about your design of the experience and at which points you allow certain gestures.

Gesture frame

When users perform gestures, they do so within a gesture frame. This frame defines the boundaries of where the HoloLens picks up gestures made by the user. It is roughly a couple of feet on either side of the user, as the following figure illustrates.

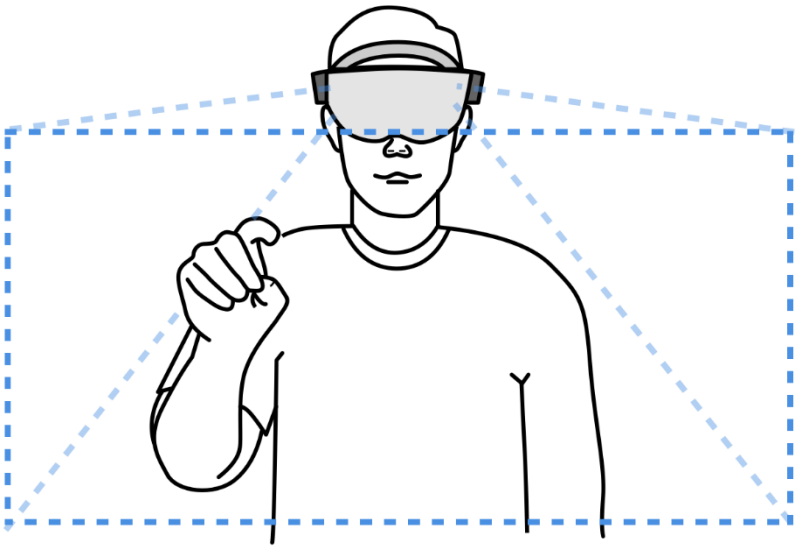


Figure 24: Gesture Frame

If the user performs gestures outside the boundaries of the frame, the gestures will stop working. If a gesture such as navigation takes place wholly or partly outside the gesture frame, the HoloLens will lose the input as soon as the gesture can't be seen. This can cause confusion, and the correct response in the experience should be managed.

To assist the user in knowing when they are performing a gesture inside or outside the frame, the HoloLens cursor will change shape (Figure 25). If no gesture is seen, it is merely a dot. If the ready state (refer to the following section) is seen, it is a circle or ring.

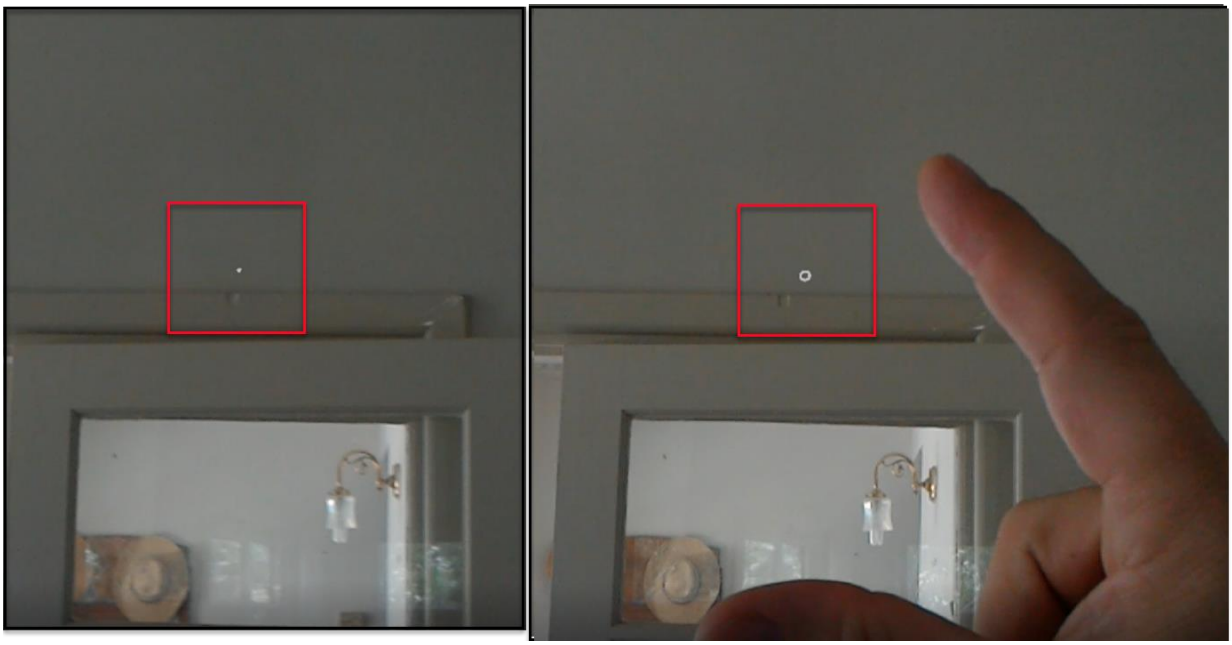


Figure 25: Hand Outside and Inside Gesture Frame

The gesture frame is completely managed by the framework, and no coding is required to take advantage of it. The interaction with gestures, however, needs to be considered, such as what happens to the experience if the gesture input is lost. Consequences of breaking the gesture frame boundaries should be minimized. In general, this means that the outcome of a gesture should be stopped at the boundary, but not reversed. Take the user on a journey and educate them on how you handle various exceptions.

Tap

The most fundamental gesture of the HoloLens experience is the tap. This commonly used gesture is made up of two of the three basic gestures that HoloLens can recognize: ready and tap.

www.dbooks.org

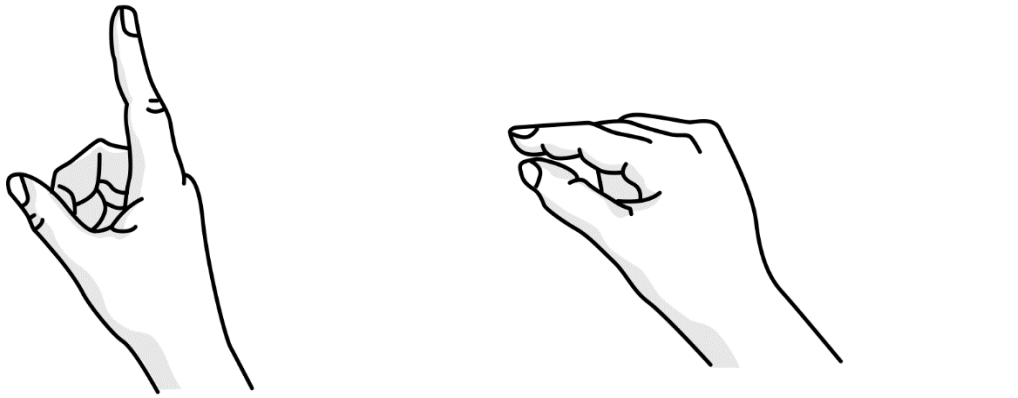


Figure 26: Ready and Tap Position

The tap is comprised of quickly moving the finger from the ready position to the pressed position and back again. Most of the time, the tap event is preceded by the user gazing at a hologram to select it. The tap is captured by the framework, and you can handle the event in code.

The key object to use is **`GestureRecognizer`**, which handles all the platform gesture events. To listen and handle the **`Tap`** event, use the following code.

Code Listing 2: Handling a Simple Tap Gesture.

```
var recognizer = new GestureRecognizer();

recognizer.Tapped += (args) =>

{

    // Do something when tap event occurs.


};

recognizer.SetRecognizableGestures(GestureSettings.Tap |
GestureSettings.Hold);

recognizer.StartCapturingGestures();
```

Of course, the code in Code Listing 2 is in a script, which is attached to the object that the user taps on. An important performance benefit is to use the **`SetRecognizableGestures`** to inform the **`GestureRecognizer`** about which gestures to raise events for.

Note:** `Args` **in Unity uses delegates for implementation, whereas** `EventArgs` **uses events.

Tip: Avoid using the .NET import of the** `System` **namespace, as it collides several places with the** `UnityEngine` **namespace.

Bloom

The third gesture that the HoloLens framework can recognize, apart from ready and pressed, is bloom. This gesture is the “home button” and equivalent to pressing the Windows key on a standard keyboard, and it t6rftconsists of holding out your palm and putting your fingertips together, then opening your hand (Figure 27).

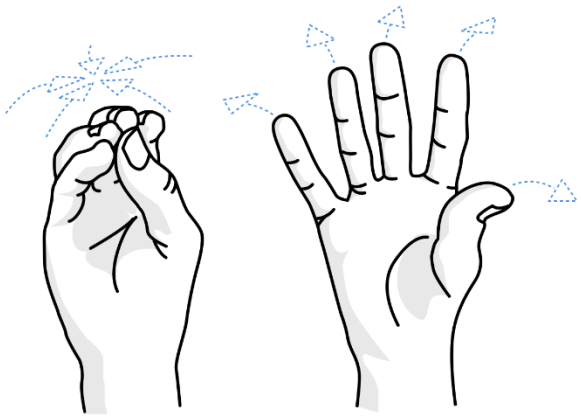


Figure 27: Bloom Gesture

The bloom gesture cannot be overridden or manipulated through code. When the user performs the gesture, one of two scenarios will happen: the start menu is opened or closed, or the user exits the currently running experience. These are fail-safe, and there isn’t any way to stop the user from using bloom.

Advanced gestures

Although the HoloLens currently supports only two basic gestures, you can use the ready and pressed states along with 3D space to perform more advanced gestures. These are also supported by the framework and come with their own set of events that you can handle in code.

These advanced gestures can be used at any time, and the setup for the HoloLens educates the user about them. They are a part of the framework, and as such, should be used in your experience where appropriate.

www.dbooks.org

Hold

The first gesture is hold, which is the user going from ready position to pressed, and then holding the finger pressed. This is useful for alternate tap actions, activating a relocation of a hologram, pausing a hologram, and so much more.

As with other built-in gestures, you don't have to identify when the gesture occurs, but only listen for the specific event. To handle the hold gesture, listen for these three events:

- **`HoldStarted`**: This fires when the system recognizes the user has started a hold gesture.
- **`HoldCompleted`**: This fires when the system recognizes the user has completed a hold gesture. This is generally when the user changes from the pressed to the ready state without moving the hand.
- **`HoldCanceled`**: This fires when the system recognizes the user has canceled a hold event. Generally, this happens when the user's hand moves outside the gesture frame of the HoloLens.



Figure 28: Hold Gesture

An example of how to handle the **`Hold`** event is shown in Code Listing 3.

Code Listing 3: HoldStarted Event Handling

```
var recognizer = new GestureRecognizer();

recognizer.HoldStarted += (args) =>

{

    // Do something when hold started event occurs.


};

recognizer.StartCapturingGestures();
```

Manipulation

Once they have performed a hold gesture, a natural action for users is to try and move the hologram they have targeted. Moving the hand either horizontally or vertically will trigger the manipulation gesture events. A manipulation gesture can be used to move, resize, or rotate a hologram when you want the hologram to react 1:1 to the user's hand movements.

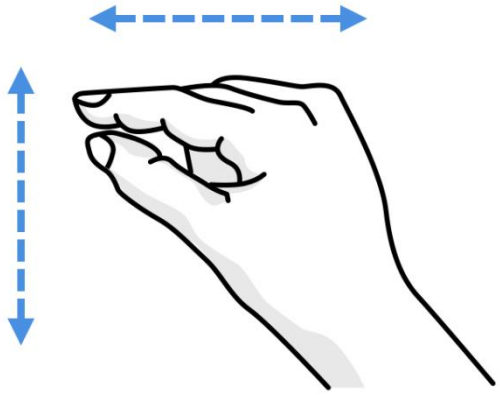


Figure 29: Manipulation Event

To use manipulation, listen and handle these events in your code:

- **`ManipulationStarted`**: This fires when the system recognizes the user has started a manipulation gesture.
- **`ManipulationUpdated`**: When the user moves their hand while performing the manipulation event, new coordinates for the destination of the gestures are generated and passed in the **`ManipulationUpdatedEventArgs`** object.
- **`ManipulationCompleted`**: This fires when the system recognizes the user has completed a manipulation gesture. This is generally when the user changes from the pressed to the ready state after moving the hand horizontally and/or vertically.
- **`ManipulationCanceled`**: This fires when the system recognizes the user has canceled a manipulation event. Generally, this happens when the user's hand moves outside the gesture frame of the HoloLens.

Code Listing 4: ManipulationStarted Event Handling

```
var recognizer = new GestureRecognizer();

recognizer.ManipulationStarted += (args) =>

{

    // Do something when the ManipulationStarted event occurs.


};
```


www.dbooks.org

```
recognizer.StartCapturingGestures();
```

The manipulation gesture is often used for scrolling pages using a hold gesture, and then moving the hand up and down, or for moving holograms in a 2D plane.

Navigation

The third (and most complex) gesture is navigation. It operates like a virtual joystick within a normalized 3D cube. The user can move their hand along the X, Y, or Z axis from a value of -1 to 1, with 0 being the starting point.

The navigation gesture is often used to move holograms from one location to another, resize objects, and to perform more complex tasks like velocity-based actions based on how much you move your hand. It is a very natural gesture as you are moving your hand in 3D space, and the actual experience is also 3D.

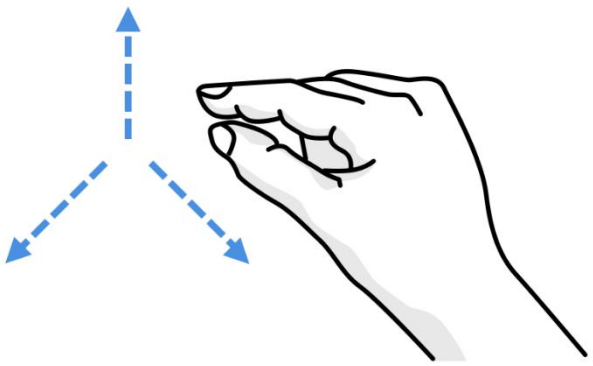


Figure 30: Navigation Event

To use navigation in your experience, you can handle four different events:

- **`NavigationStarted`**: This fires when the system recognizes the user has started a navigation gesture.
- **`NavigationUpdated`**: When the user moves their hand while performing the navigation event, new coordinates for the destination of the gestures are generated and passed to the **`NavigationUpdatedEventArgs`** object.
- **`NavigationCompleted`**: This fires when the system recognizes the user has completed a navigation gesture. This is generally when the user changes from the pressed to the ready state after moving the hand within the bounded 3D cube.
- **`NavigationCanceled`**: This fires when the system recognizes the user has canceled a navigation event. Generally, this happens when the user's hand moves outside the gesture frame of the HoloLens.

Code Listing 5: Handling a NavigationUpdated Event

```
var recognizer = new GestureRecognizer();
recognizer.NavigationUpdated += (args) =>
{
    // Set a position for something to the updated position when the
    // NavigationUpdated event occurs.
    NavigationPosition = args.normalizedOffset;
};
recognizer.StartCapturingGestures();
```



Note: Users often accidentally go from manipulation to navigation. You can control and assist users in code by managing which events to listen for.

www.dbooks.org

Chapter 6 Voice

The HoloLens is one of the best voice recognition devices on the market. Because you always have the microphone in the same position on the device, the device can very accurately decipher your mumbles and broken sentences. Voice is the third input method in the GGV paradigm for HoloLens, and it is as important as the first two.

It is expected that you can use most, if not all, of the actions and interactions with voice commands. If there is a gesture to move a hologram, there should be a voice command to do the same.

Voice commands are meant to make the interaction model simpler, make the experience more delightful, and enhance the immersion and engagement from the user. If this isn't the case, either your interaction model could be improved, or your voice design is lacking.

This chapter will explain how to get the most out of the voice component of the GGV input model, and where some of the pitfalls are.

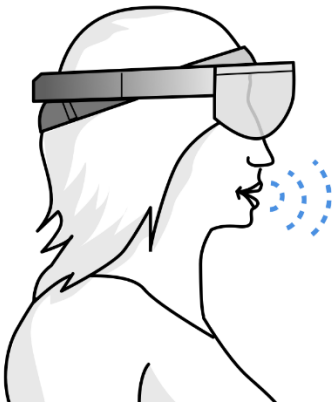


Figure 31: Voice Commands and the HoloLens Platform.

Built-in voice commands

First, there are voice commands provided by the platform. The HoloLens device has a number of commands that come for free and which users will get familiar with over time. As a developer or experience owner, you can use them to enhance the project.

Select

Instead of using the tap gesture, users can at any time say "select." This will trigger a **`Tapped`** event, identical to a tap gesture performed by a user. Your app won't know the difference, and in code, you will handle the event like any other **`Tapped`** event.

The user will hear a sound and see a tooltip with the word “select” appear as confirmation. Select is enabled by a low-power keyword detection algorithm, so it is always available to say at any time with minimal battery-life impact, even when the user has their hands at their side.

Place

Similar to the select command, users can say “place” to trigger a tap event to place a hologram. Where select is a broader command to execute a tap event, place is specifically designed for placing or putting down holograms.

Face Me

When the user is gazing at a hologram, they can say “face me,” and the hologram will turn towards the user. This requires the hologram to have a front identified to allow the turn to happen. This works really well on 2D apps that tend to always be at a funny angle relative to where the user is.

Bigger/smaller

If a hologram can be resized, use the “bigger” or “smaller” commands to resize it. This can often also be achieved with gestures or a combination of voice commands and gestures.

Hey Cortana

Cortana is Microsoft’s digital assistant, which works across devices running Windows 10. The HoloLens is no different. Cortana is always listening for the words “Hey Cortana,” which will trigger the blue lady.



Note: Cortana is named after the fictional intelligence character from the Halo video game series.

There are a lot of built-in commands you can use with Cortana to make her do your bidding, once you have her attention:

- *What can I say?*
- *Go home* or *Go to Start* (instead of bloom to get to Start Menu)
- *Launch <app>*
- *Move <app> here*
- *Take a picture*
- *Start recording*
- *Stop recording*
- *Increase the brightness*
- *Decrease the brightness*
- *Increase the volume*
- *Decrease the volume*
- *Mute* and *Unmute*

www.dbooks.org

- *Shut down the device*
- *Restart the device*
- *Go to sleep*
- *What time is it?*
- *How much battery do I have left?*
- *Call <contact>* (requires HoloSkype)

The full set of Cortana features and commands from Windows 10 on desktop aren't available on HoloLens, and, currently, the only supported language is English.

See it, say it

When buttons in a HoloLens experience have labels on them, users can use the "see it, say it" approach to trigger a button press. For example, when looking at a 2D app, a user can say the "remove" command, which they will then see in the app bar to close the app.



Figure 32: Say "Remove" to Trigger the Remove Button

It is highly recommended that you follow this rule, as users can easily understand what to say to control the system. To reinforce this, while gazing at a button, you will see a "microphone" tooltip that comes up after a second if the button is voice-enabled, which displays the command to speak to "press" it.

Custom voice commands

All the built-in commands are great, but the real power of the HoloLens platform comes in creating your own voice commands. While it is relatively trivial to create basic voice commands for your HoloLens experience, there are quite a few nuances to be aware of when building your system.

First, let's look at the code for implementing a basic voice command. The most commonly used feature is the **`KeywordRecognizer`** object. It listens for any speech input and matches any spoken phrases to a known list of keywords. These keywords are registered when the experience is started.

Code Listing 6: KeywordRecognizer Implementation

```
KeywordRecognizer recognizer = null;

Dictionary<string, System.Action> keywords = new Dictionary<string,
System.Action>();


    keywords.Add("Open e-Book", () =>

    {

        // Call the OpenBook method on every descendant object.

        this.BroadcastMessage("OpenBook");

    });


    // Tell the KeywordRecognizer about our keywords.

    recognizer = new KeywordRecognizer(keywords.Keys.ToArray());


    // Register a callback for the KeywordRecognizer.

    recognizer.OnPhraseRecognized +=
KeywordRecognizer_OnPhraseRecognized;

    recognizer.Start();

}
```

In Code Listing 6, we create a **`Dictionary`** to hold the **`Action`** associated with the keyword. The keyword is the phrase to be recognized by the framework. We create an event handler for when a phrase is recognized, and then start the **`KeywordRecognizer`**. (Don't forget to start it!)

www.dbooks.org

The next step would be to handle the **`OpenBook`** event being sent from the **`Action`**. There are other ways to create voice commands and actions, but they all revolve around the **`KeywordRecognizer`**.

Tip: Use the Mixed Reality Toolkit features to implement voice commands for more robustness and abstraction.

Voice command design

While the voice commands are technically straightforward to implement, creating effective and well-designed voice commands is much more difficult. There is a lot of theory and psychology to how users perceive, remember, and use voice commands for a given experience.

Table 2: Do's and Don'ts of Voice Commands

Do	Don't
Be concise and don't use unnecessary words in the commands. Concise is easier to remember.	Use single-syllable commands. These are more easily missed by the system and can be misinterpreted based on dialect and accent.
Use common words that are easy to learn and remember.	Use system commands. For example, the "select" command is already assigned to a system-wide function.
Be consistent. Use the same command for the same action in different places.	Use commands that sound similar. They are harder to remember correctly, and harder for the system to get right accurately.

It is important to create natural commands that are both easy to remember and that fit the context of the experience. For example, using the command "open hatch" is not as easy to remember as "open door."

Tip: The voice recognition APIs and tools provided by the HoloLens platform are all hardware accelerated. This uses a lot less power than if the CPU had to process the voice input, and you should avoid building your own voice recognition feature, unless necessary.

User education

A big part of the challenge when using voice commands on a platform that is mostly visual is knowing when to use a voice command, or even what they are! Educating your users on how and when to use voice commands in your experience is critical. Some of the ways to educate users include:

- Create visual clues such as a microphone icon with the command written next to it, when the command is available.
- Use a wizard process to guide users through any voice command learning.
- Have a short tutorial before the experience or level of the experience, to show any commands or gestures needed.

The education of users is critical to make your experience a success. If users don't "get" how to use the tools available, they will give up quickly and move on.

User feedback

Part of creating an effective experience, and effective voice commands is to have a two-way flow of communication. If you keep shouting at users, they will get annoyed and leave. Instead, make sure they know when the experience is listening for their command, give an audio or visual clue when you have processed the command, and keep them engaged.

Dictation recognizer

Another way to take advantage of the voice recognition part of the GGV input paradigm is the **`DictationRecognizer`**. This provides a great user experience if you have parts of your app that require significant user input. The HoloLens onscreen keyboard is cumbersome to use, mainly because you have to gaze at each letter and then tap.

Using the **`DictationRecognizer`** is very similar to the **`KeywordRecignizer`**. In fact, they are so linked that you can only register one or the other at any one time in your app.

Code Listing 7: Creating a DictationRecognizer and Setting It Up

```
DictationRecognizer recognizer = new DictationRecognizer();


// This event continually returns what has been heard so far.
recognizer.DictationHypothesis +=
DictationRecognizer_DictationHypothesis;

// This event fires at any pauses in dictation. It returns the full
recognized string.
recognizer.DictationResult += DictationRecognizer_DictationResult;


// This event fires when the recognizer stops.
recognizer.DictationComplete += DictationRecognizer_DictationComplete;

// Stop any KeywordRecognizers.
```


www.dbooks.org

```
PhraseRecognitionSystem.Shutdown();

Recognizer.Start()
```

By using the events in Code Listing 7, you can manage an elegant dictation experience for the user. Because the microphones on the HoloLens are always in the same position relative to the user's head, the voice recognition works extremely well. Combined with the hardware accelerated microphones and processing, it makes the HoloLens a very powerful voice recognition device. Voice dictation is especially good for something like URLs, where the user can spell out the whole address.



Note: You need a Wi-Fi connection to use the dictation feature.

Grammar recognizer

Closely related to the **`DictationRecognizer`** is the **`GrammarRecognizer`**. It uses a Speech Recognition Grammar Specification[5] (SRGS) file to define the rules for the grammar to check. The SRGS file specifies the words and patterns of words to be listened for by a speech recognizer.

Speech recognition is incredibly powerful, and using an SRGS file will create both structure and guidance to the experience. A speech recognition grammar is a container of language rules that define a set of constraints a speech recognizer can use to perform recognition.

Code Listing 8: Creating a GrammarRecognizer and Setting It Up

```
GrammarRecognizer recognizer = new GrammarRecognizer(<path to SRGS
file>);


grammarRecognizer.OnPhraseRecognized +=
GrammarRecognizer_OnPhraseRecognized;

grammarRecognizer.Start();

private void
GrammarRecognizer_OnPhraseRecognized(PhraseRecognizedEventArgs args)

{

    // Use this to build up the recognized message from each rule.

    StringBuilder messageHeard = new StringBuilder();
```

[5] See SRGS definition here.

```
    // This array contains the results of the SRGS rules the recognizer
heard.
    SemanticMeaning[] meanings = args.semanticMeanings;


    foreach (SemanticMeaning meaning in meanings)
    {
      messageHeard.Append(meaning);
   }
}
```

In Code Listing 8, when the **GrammarRecognizer** recognizes a phrase that matches a rule in the SRGS, the **GrammarRecognizer_OnPhraseRecognized** event handler is triggered. We then build up a rudimentary list of what was heard. In a real project, you would of course react to the individual semantic meanings and have your experience react accordingly.



Tip: Using a SGRS file makes the experience easier to port, as that is the standard for defining voice recognition phrases.

www.dbooks.org

Chapter 7 Sound

When a user moves through a HoloLens experience, they can't keep track of everything happening around them. This is partly because the human mind can only handle so many inputs at the same time, and partly because of the limited field of view on the device. Sound can provide a footing and reference for the experience that helps the user both navigate and trust it.

In a HoloLens experience, the sound simulates the environment by providing both distance and direction. The highly accurate and real-time speakers on the HoloLens delivers audio as if it was real in the experience. While the visuals are limited to where the user is looking and the physical frame of the lenses on the device, audio can come from all directions.

Spatial sound

Our human ears are already brilliant at managing sounds coming from all directions and distances. This spatial sound is part of our natural universe, and to create a HoloLens experience that is both natural and convincing, it is paramount that audio functions in a similar way. This is done through spatial sound, which means sound emanates from a particular location, and as the user moves around the space, the sound stays in the same location. This is an integral part of the HoloLens experience, and while not part of the GGV paradigm, is just as important as gaze, gesture, and voice.

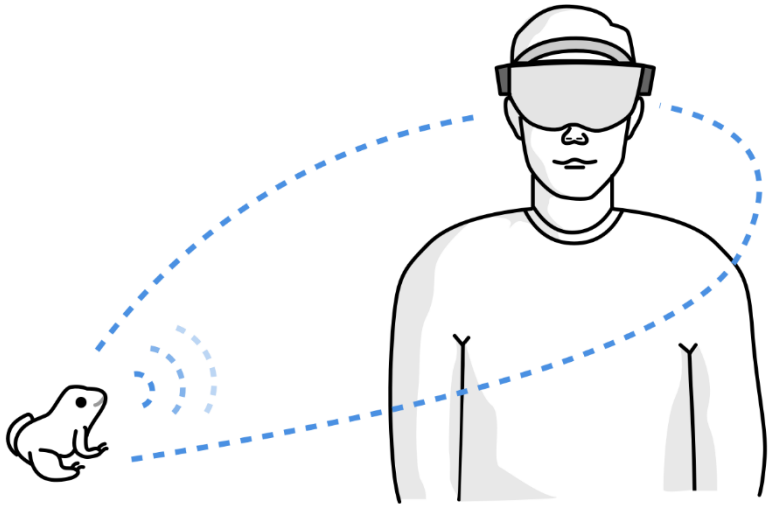


Figure 33: Spatial Sound

Head-related transfer function

To simulate and copy how human ears function, HoloLens implements a head-related transfer function (HRTF). This algorithm models the spectral response of how an ear receives sound from any point in space. The position of each sound, usually attached to a hologram, is calculated and simulated using the four speakers of the HoloLens.

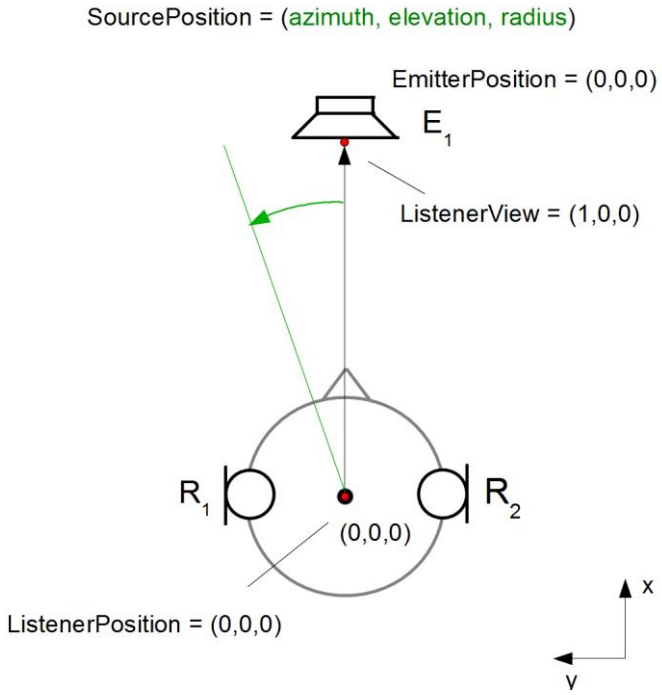


Figure 34: Head-related Transfer Function (User:Pitbub, CC BY 3.0)

The timing for a sound hitting each ear is different, and that is how we, as humans, perceive direction and distance of sound. The HRTF simulates this timing incredibly well, and the HoloLens hardware implements it. Humans have just two ears, but we can locate sounds in three dimensions: in range (distance); in direction above, below, in front of, and to the rear; and to either side. This entire range of binaural features is supported by the HoloLens.

While the HRTF is incredibly complex and varies from human to human, the HoloLens implementation is very accurate. The best part is that as a developer, you don't have to worry about it at all—implementing sound effects for your holograms is simple and fast.

Creating spatial sound

Using spatial sound in your HoloLens app is simpler than you might think. The easiest way to implement a sound design for a hologram is to use Unity to drag and drop a sound file onto a 3D object, and then configure it in the inspector panel.

www.dbooks.org

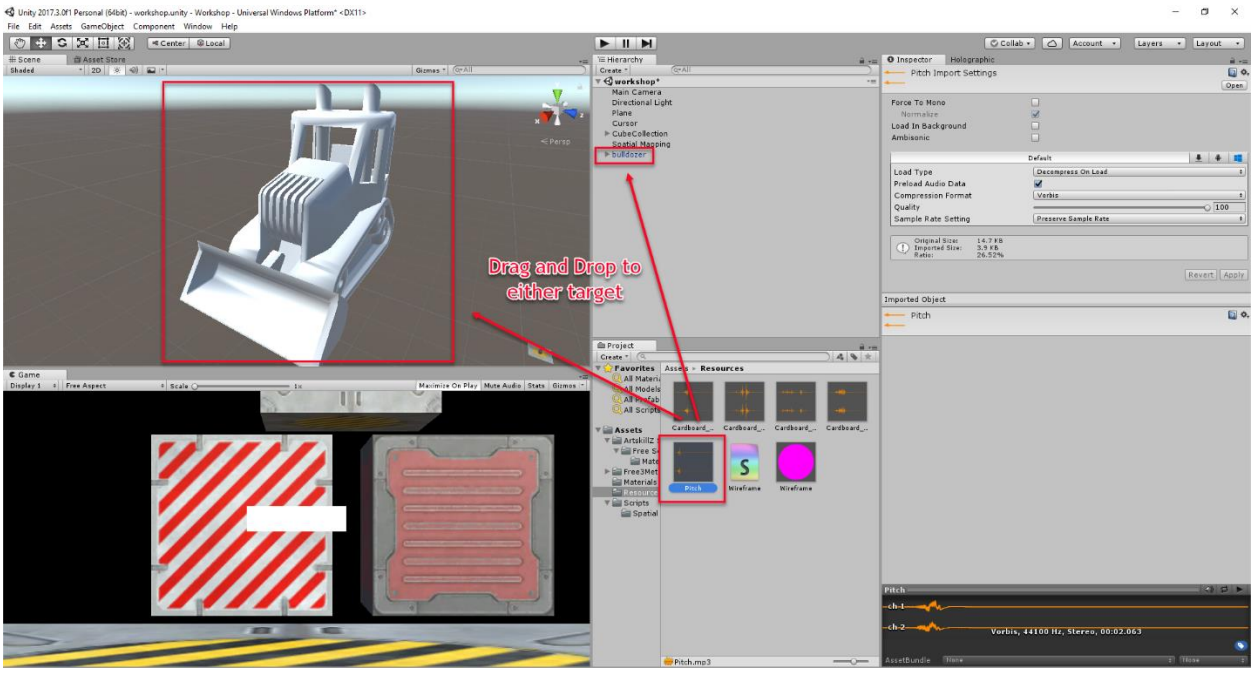


Figure 35: Spatial Sound Implementation in Unity 3D

That is all you need to do in order to create a basic spatial sound for a hologram. The framework will take care of calculating the HRTF and continuously making sure the sound is projected from the hologram using the HoloLens.

Having audio statically linked to a hologram doesn't cover all scenarios, of course. Often, you'd want to trigger sounds in code and manage it through interactions and actions. This is done through scripts, as with other parts of the platform, and makes use of the same properties that you used in the inspector panel by manipulating an **`AudioSource`** object.

Code Listing 9: Creating an Audio Source in Code and Triggering It

```
AudioSource audioSource = gameObject.AddComponent<AudioSource>();
audioSource.playOnAwake = false;
audioSource.spatialize = true;
audioSource.spatialBlend = 1.0f;
audioSource.dopplerLevel = 0.0f;
audioSource.maxDistance = 15.0f;
audioSource.rolloffMode = AudioRolloffMode.Logarithmic;


audioSource.clip = Resources.Load<AudioClip>("<some clip>");

// Occurs when this object starts colliding with another object.

void OnCollisionEnter(Collision collision)

{
    // Play an impact sound if there is an impact.
```

```
    if (collision.relativeVelocity.magnitude >= 0.2f)
    {
        audioSource.Play();
    }
}
```

Note: Unity 3D uses type `float` instead of `double`.

In Code Listing 9, a new **`AudioSource`** object is created, some of its properties are configured, and then in one of the built-in events, **`OnCollisionEnter`**, the audio is played.

The actual implementation of spatial sound in a HoloLens experience is not complex or difficult to program. As with most of the basic HoloLens development sections mentioned previously, you just need a knowledge of C# and some basic tooling.

Sound design

The difficult parts are designing the soundscape for your particular experience and understanding how various decisions will impact it. There are a ton of variables to consider when designing sound for your experience, but with a bit of careful planning and attention, it can make all the difference between an average and an excellent experience.

The list in Table 3 shows some of the common issues you must manage for a full and complete sound experience.

Table 3: Sound Design Tips

Area	Action
Normalizing	Create a standard normalized baseline volume for your sound files. This will keep them maintainable for future updates.
Movement	Be aware that users will move about in the space they occupy. Your sounds must match that experience.
Accuracy	Users can move about freely, which means they will get up close to holograms. The sound needs to be emitting from an accurate location on the hologram. For example, a dinosaur's roar shouldn't come from its foot.
Natural	The whole idea of HoloLens experiences is to convince the user that the holograms are part of the real world. Creating natural sounds is a great help in achieving this. An eagle should sound like an eagle, and tapping on wood should sound like that, too.

www.dbooks.org

Area	Action
Expectations	Every user will have expectations about the sounds they will hear. Based on familiarity, users will have “coded” responses to certain types of sounds. If they hear a bird, they will look up. If they hear their name, they will look at eye height for the source of the sound.
Hidden sounds	If sounds are unfamiliar, or they are sounds that would be recognized as coming from a visible source, don’t hide the emitter of the sound. It can be a very negative experience for the user if they can’t locate what is making the sound. Familiar sounds like wind that aren’t expected to be visible are exceptions.



Tip: Consider how you would expect real world scenarios to sound when designing sound for your HoloLens experience.

Mixing sound

One of the traps of being able to just drag and drop sound bites on 3D assets in Unity, or attach a new **`AudioSource`** in code, is that you can very quickly end up with a ton of sounds—all playing at the same time! Suddenly there are so many sounds that the user can’t hear any of them. It becomes unnatural, overwhelming, and frustrating.

Microsoft recommends that you aim for a volume mix of 70 percent on the HoloLens. This will allow real world sounds to still be heard, and you will create an appropriate mixed reality soundscape.

A common technique used for the HoloLens (and VR in general) is to dial down the volume on holograms that aren’t in the view of the user, or their gaze. Using the gaze APIs, you can turn the volume up and down for holograms, as they come in and out of the user’s vision. This will make it easier to focus attention on the visuals in front of them. This technique can also be used to dial up the volume on unseen holograms that you want to draw the user’s attention to.

Interaction

Part of a successful and convincing experience on HoloLens is the realism it brings over both AR and VR apps. As mentioned before, you want users to invest and immerse themselves to the point of not consciously distinguishing between real and digital.

When it comes to sound, you should use it to enhance the realism of your app. If you have a wooden chair hologram, then perhaps have some “wooden” sound effects for placing the chair or moving it. Let your users buy into the experience.

To further encourage interaction, combine gaze with sound. If there are important objects the user should be aware of in a given context or at a specific time, you can use a sound clip to guide and direct the user's gaze. It can be a sound that increases in volume as the user gets closer, it can be more of a beacon signal, or whatever suits the experience.

The same applies for gestures. Because a gesture is a non-haptic action, using sound to indicate an outcome of a gesture is very powerful. If a user correctly taps an object, let them know by using an audible cue. If the user tries to tap an object at an incorrect moment, let them know that, too.

Spatial mapping and sound

As mentioned previously, there is no HoloLens experience without spatial mapping. It is the glue that binds digital with real life, and it is the underpinning of all pure HoloLens experiences. Sound is no different. You want relevant sound effects to be part of the experience, regardless of whether the user is interacting with a part of the 3D model created by the spatial mapping, or a digital asset.

The main design decision when it comes to sound and spatial mapping is how to handle occlusion. When holograms are behind physical objects mapped in 3D, the sound design should handle this naturally.

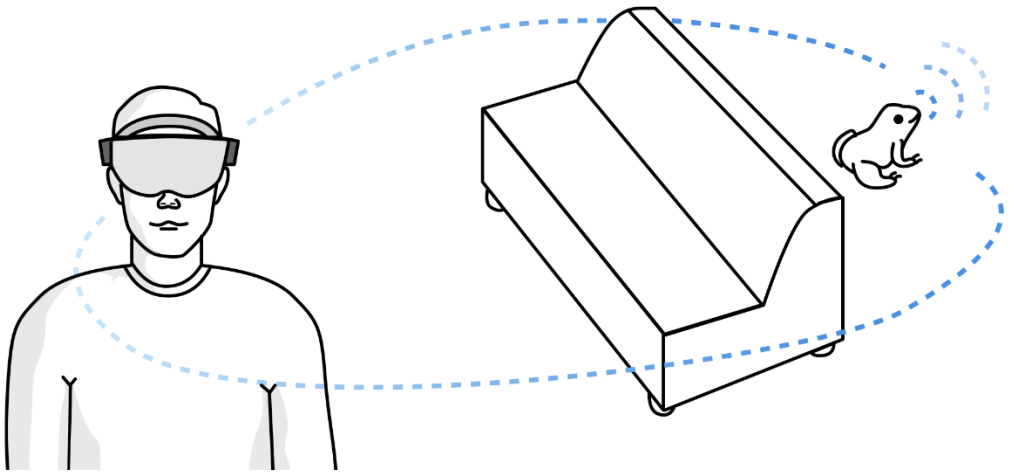


Figure 36: Occlusion of Spatial Sound

Programmatically, sound occlusion is done by identifying if there are objects between the hologram and the user, and if so, applying the appropriate effect to the hologram. The most common effect is to lower the volume and pitch. Imagine hearing sound behind a closed door—it's more muffled, and the volume is lower, than if the user were to open the door.

www.dbooks.org

Chapter 8 Hologram Quality

So far you have gone through how to implement and design your HoloLens experience. The focus has been on creating realistic and “world enhancing” experiences that provide value to the user. As much as you can optimize your code and experience, such as providing gaze guidance, audible clues, and intuitive and memorable voice commands, your holograms must be of great quality, too.

As you will learn, quality means a few things when it comes to holograms.

Frame rate

The absolute pillar of creating quality holographic experiences in mixed reality is the frame rate. Your app should run at 60hz, which means the frame is updated 60 times every second. For comparison, a standard movie you watch at the theater has 24 frames per second (fps). In other words, you must provide a new image from your app to the operating system approximately every 16 milliseconds.

Part of the illusion of having holograms interact with the real world is that they are rendered from the user’s position and perspective. The HoloLens is not fast enough to render the holograms in real time and uses predictive algorithms to estimate where a user’s head might be at any time. In addition, the HoloLens hardware will adjust any discrepancies between the predicted and the actual position of the user’s head.

By ensuring a constant experience at 60 frames per second, you are, as developer, doing your best to allow the hardware to perform optimally. By minimizing latency, you are also increasing hologram stability and decreasing prediction error margins. The issue of latency and frame rate is even higher on virtual reality devices that generally run at 90 fps.



Tip: Providing a frame rate lower than 60 fps will lose users fast.

Providing a lower frame rate from your app also strains the hardware, as well as the user’s acceptance of the experience. Imagine your app is running at 30 fps. The HoloLens is still rendering the display at 60 fps, so each image is rendered twice. The second image requires significant more error placement calculations before being rendered to the display.

Overall, providing a consistent frame rate of 60 fps will help alleviate judder (see the following section).



Tip: Remember that it is better to provide a consistently lower frame rate than an inconsistent one.

Here are some tools that can help you debug frame rate issues:

- Visual Studio's Graphics Debugger
- GPUView
- Profiler windows in Unity 3D

Identifying issues

The lack of quality for holograms can have a number of causes. It is important to be able to identify the various kinds of issues that can happen in a HoloLens experience. Some of the symptoms you can manage through better app development, and some the user will need to resolve on their device.

Accuracy

Holograms must stay in the exact same spot at all times. Even when the user makes a small head movement or their location changes, the view of the hologram has to be completely stable relative to the surrounding environment, as mapped by the spatial mapping. If the hologram shifts position, there is an issue of accuracy, which can cause the experience to fail or cause discomfort to the user.

Accuracy problems are often caused when two locations look similar. This is one of the pitfalls of creating a HoloLens experience, especially if users are in square rooms with no or little furniture. The spatial mapping becomes so similar that locations are mixed up and the accuracy suffers.

Color separation

The HoloLens device itself displays each of the colors red, green, and blue sequentially on the display the user sees. Each color is overlaid on the others to create a full color spectrum. This happens at 60hz per color, and as fast as that is, some separation of the colors can occur, either if the user moves their head rapidly, or if a hologram moves very fast in the experience. White colors can especially suffer from color separation, as they comprise all layers of color.

If you do want to move a hologram, keep the movement under 5 degrees per second if it requires either gaze or head movement from the user. Another really simple way of countering color separation is to make the object either red, green, or blue. You could also switch to a blurred version of the asset as it moves across the scene. This will make the user more accepting of color imperfections in the hologram.

www.dbooks.org

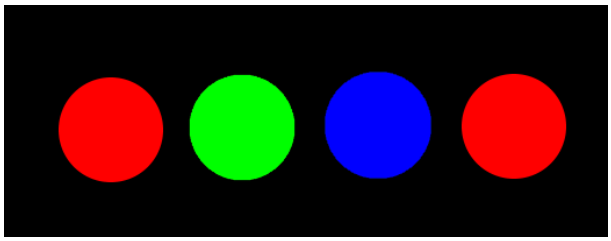


Figure 37: Color Perception of a White Circle as User Moves Gaze from Left to Right

Drift

When users experience holograms drifting away from their original placement, this is called drifting. Because holograms are anchored in relation to the real world using spatial anchors, it is not a perfect placement. The real world might change, or the initial estimate of location, placement, and distances might be off. Spatial anchors are needed to make sure holograms stay where you place them, and if the holograms are too far from the spatial anchor, they can drift from the anchor.

Note: How you manage drift is very often based on your experience. You don't want too many spatial anchors, as that affects accuracy, but you don't want holograms too far from them, either.

Jitter

If the tracking of the room starts degrading, or there are inconsistencies in the 3D model created by the HoloLens, the holograms might start to shake at a relatively high frequency. This is referred to as jitter.

To resolve jitter, it is most likely the user that needs to tune the sensors[6] on the device. As a developer, you don't have much control over the spatial mapping and its performance.

Judder

One of the symptoms of a low frame rate is juddering of images. When the frame rate jumps up and down, the device has a hard time presenting a consistent image to the user. This can result in ghosting of images, or sudden jumps in location. As mentioned in the "Frame " section of this chapter, it is paramount that you are delivering 60 fps, and if that isn't possible, at least be at a stable fps.

[6] Sensor tuning is done using the Sensor Tuning part of the Utilities in the Settings on the HoloLens device.

Jumpiness

Holograms may suddenly jump from one location to another. This effect is closely related to judder. However, where judder is related to poor frame rate performance, jumpiness generally happens as the device is updating the understanding of the surrounding environment. The new and updated understanding of the environment can cause remapping of where to project holograms, which makes them jump.

Swim

If a hologram is moving in sync with the motion of the user's head, it creates a weird detached sensation for the user. This is referred to as swimming. This can occur when holograms aren't correctly on the stabilization plane (see the next section). Developers must address this, as well as users, by running the calibration app on the device.

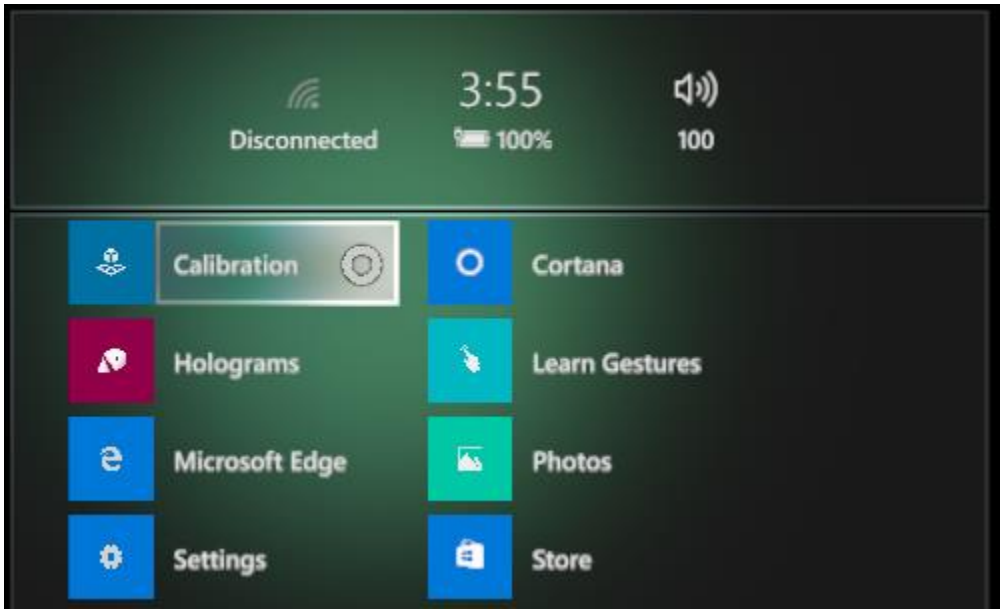


Figure 38: Calibration App on HoloLens

Stabilization plane

While you don't have to address or be concerned about the stabilization of holograms in general, there is a single responsibility for the developer. Each hologram is stabilized through the hardware, but it needs a single reference point to maximize this function. The stabilization plane is a single "slice" or window that is chosen in conjunction between the device and the application, or programmer. Holograms in the stabilization plane, which is generally the focus of the app, receive more accurate stabilization than other holograms.

www.dbooks.org

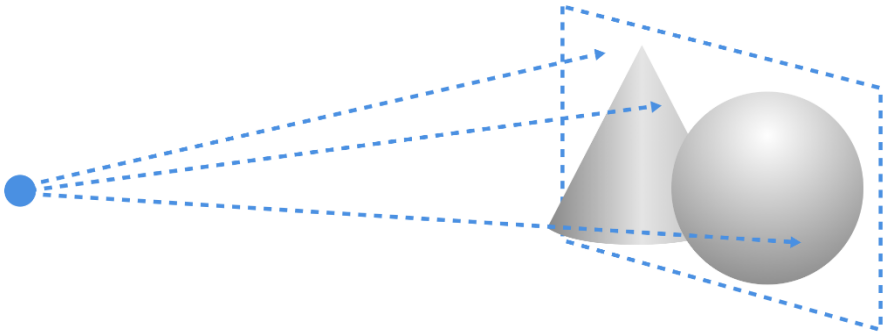


Figure 39: Stabilization Plane in a 3D Scene

The application can help the hardware identify the stabilization plane by setting the best current focus of the experience using the **`SetFocusPoint()`**. This is a purely subjective decision made by the app developer as to what they want the user of the app to observe and focus on.

Because all holograms on the stabilization plane are perfectly stabilized, you should place as much as possible on them. This might take some trial and error, but it is worth doing in order to avoid swimming holograms and to keep your users from having a visit from their lunch.

A few things to consider:

- If you are going to adjust the stabilization plane, don't just do it once. Users move around constantly, and the holograms will come and go.
- Avoid many changes of the focus point over a short period of time, especially between points relatively far apart.
- Never set the stabilization plane on top of or very close to the user. This is confusing and can cause discomfort to the user.

Chapter 9 Testing HoloLens Experiences

A HoloLens experience is a software project, regardless of how cool the hardware is, and how incredible the spatial mapping is. In the end, you are creating an application that will run on someone's machine that you don't have control over. And just like any other piece of software, you need to test your work.

As you will see, testing HoloLens development work is very different from traditional software testing. You suddenly have real-world interaction you need to manage, as well as your scripts and performance. This, of course, requires a different approach.

Device Portal

Your best friend, and tool, when it comes to testing a HoloLens experience, is the Device Portal. You access it using a web browser pointed to the IP address the HoloLens is assigned to on the local network. The portal provides insight into a range of stats, real time monitoring, and tools you can use when developing your experience.

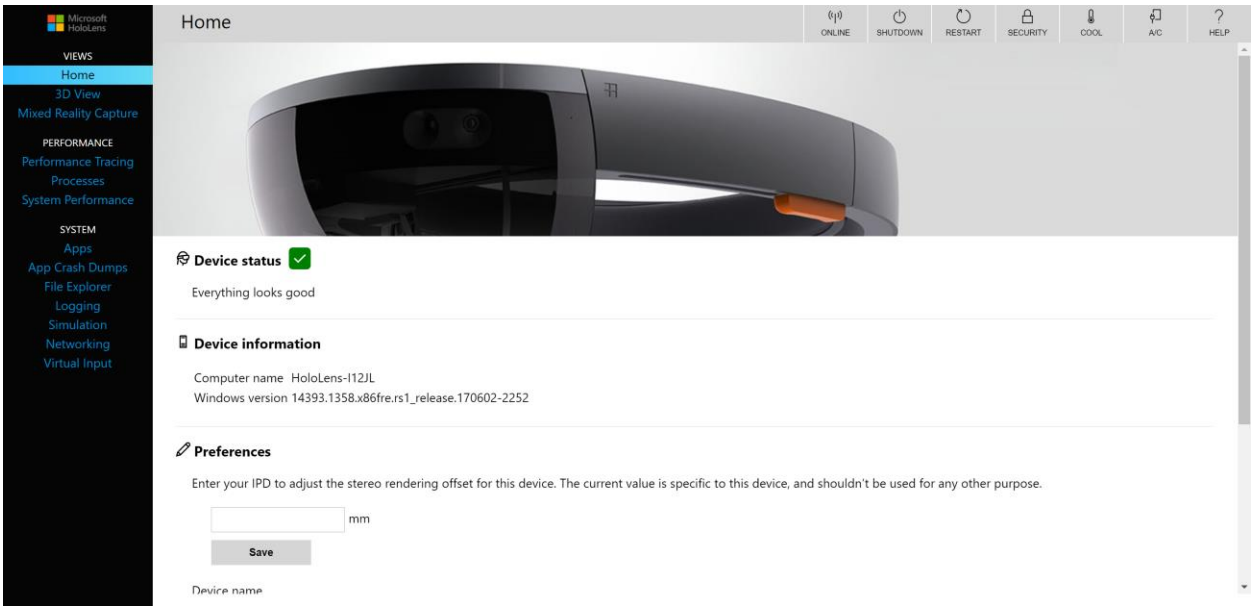


Figure 40: HoloLens Device Portal

Tip: The developer portal works on both the emulator and physical devices.

You will find that using the portal will speed up both your development and quality assurance phase, as it has tools aimed at both.

www.dbooks.org

3D view

Because the Device Portal is running off the device (or emulator) itself, you also have direct access to the 3D information from the device. Open the 3D view, and you can see exactly where the device is in relation to the current spatial mapping. This is really cool, and a great way to visualize where the user goes in the experience.

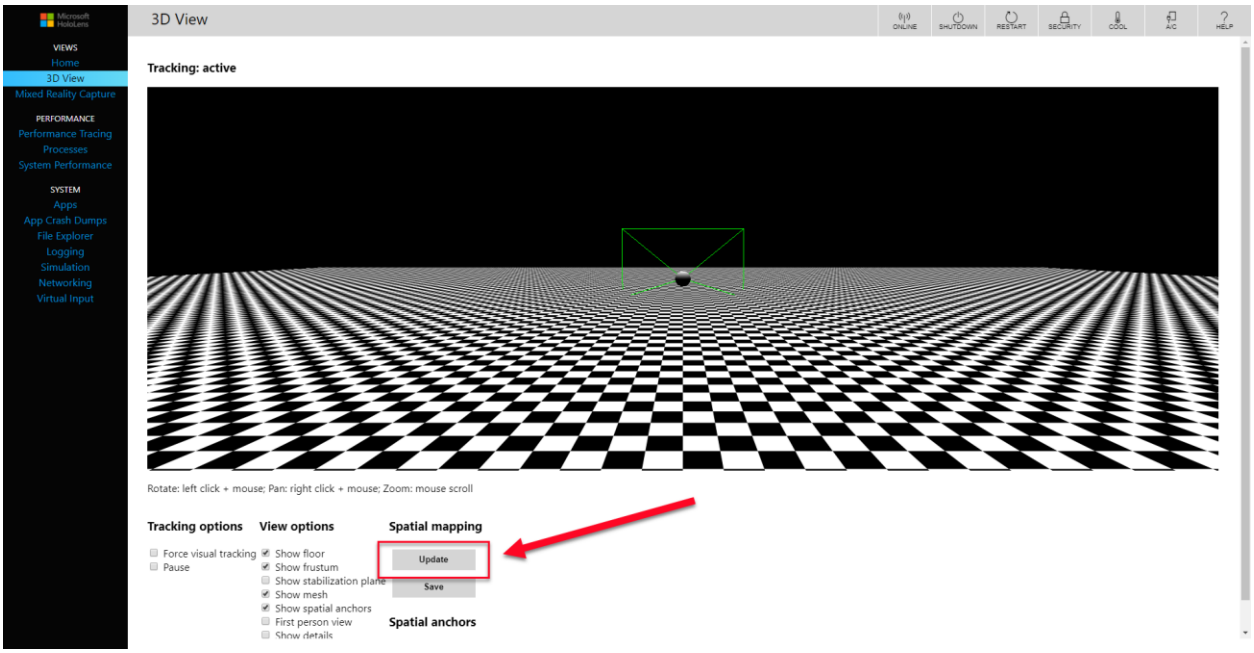


Figure 41: The Update Button

If you click the **Update** button on the **3D View** screen, the spatial mapping will be drawn in 3D on the plane in relation to the user. This is a representation of the instantaneous 3D model as created by the device.

Note: If you are using an emulator, the spatial mapping of the room is preloaded and will not update in real time.

The 3D view has a few more features that also come in handy. You can view the position of the stabilization plane in real time, which will help debug any stabilization issues. There is a checkbox for showing spatial anchors, which is very useful if holograms start drifting or you have issues with recognizing the room features. If you really want to experience the impact of the spatial mapping, you can use first person view to see the 3D environmental model as the user will see it.

Tip: Use the 3D view to update the spatial model until it is detailed enough for your use, then save it to disk. Other team members can then load the same room in their emulator, making team testing much more accurate.

Mixed Reality Capture

While the 3D view can show the polygons of the spatial mapping, you can also capture the whole experience as seen by the user. The Mixed Reality Capture lets you choose which parts of the experience you want to capture, including the holograms, the camera on the device, microphone audio, and application audio. You can also select the quality you want the recording to be in. Generally, you'd want the quality set to High, but if the bandwidth on the local network (from which you are receiving the signal) isn't great, it can work better at a lower resolution.

Being able to access the live feed of the mixed reality experience has a few benefits. You can record exactly what the user sees[7] in the experience, which can reveal any issues, such as drift and judder. Seeing exactly what the issue is makes debugging and performance improvements much easier. You can put your finger on exactly what the issue is and test a specific combination of inputs and movements, which are then recorded. You can share the recording with a team or developer that needs to analyze and resolve the error.

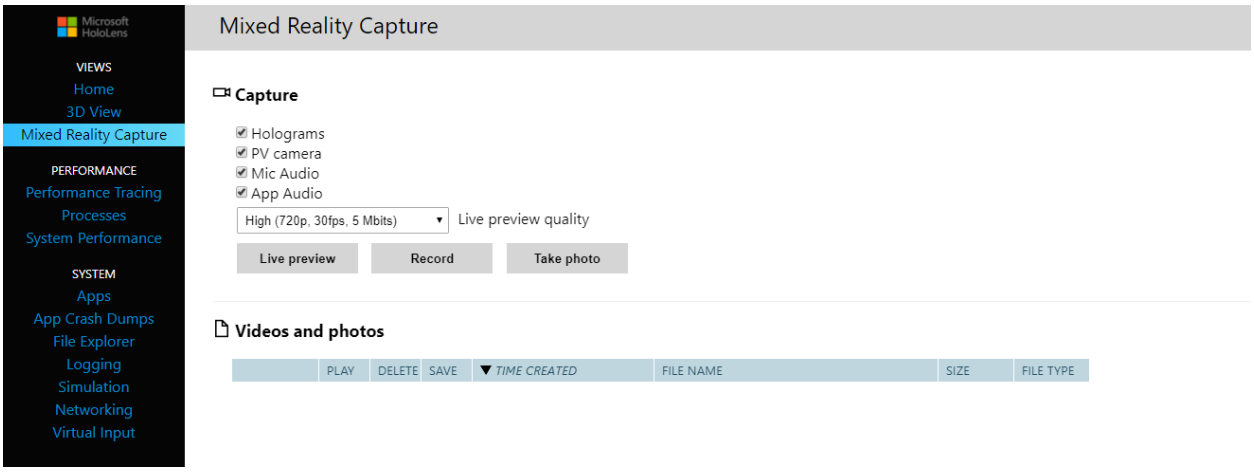


Figure 42: Mixed Reality Capture

The live preview is also very useful to show or reproduce a bug or symptom in real time for an audience. Being able to get input on actions and interactions to reproduce a particular scenario often results in a much more accurate approach.

Performance tracing

At times it can be difficult to pinpoint performance issues visually, and you might want more information for the investigation of the bug. The device portal has a very handy performance tracing function that can record a trace in an event trace log (ETL) file that holds a number of events.

[7] You record more than the user can see, due to the size of the camera feed on the HoloLens.

www.dbooks.org



Figure 43: Performance Tracing

You can open this file in the Windows Performance Analyzer[8] tool, which draws graphs and creates data tables based on the performance trace events in the file. This will provide more insight into the performance challenges of the app, but it does require a fair amount of digging, and there is a learning curve.

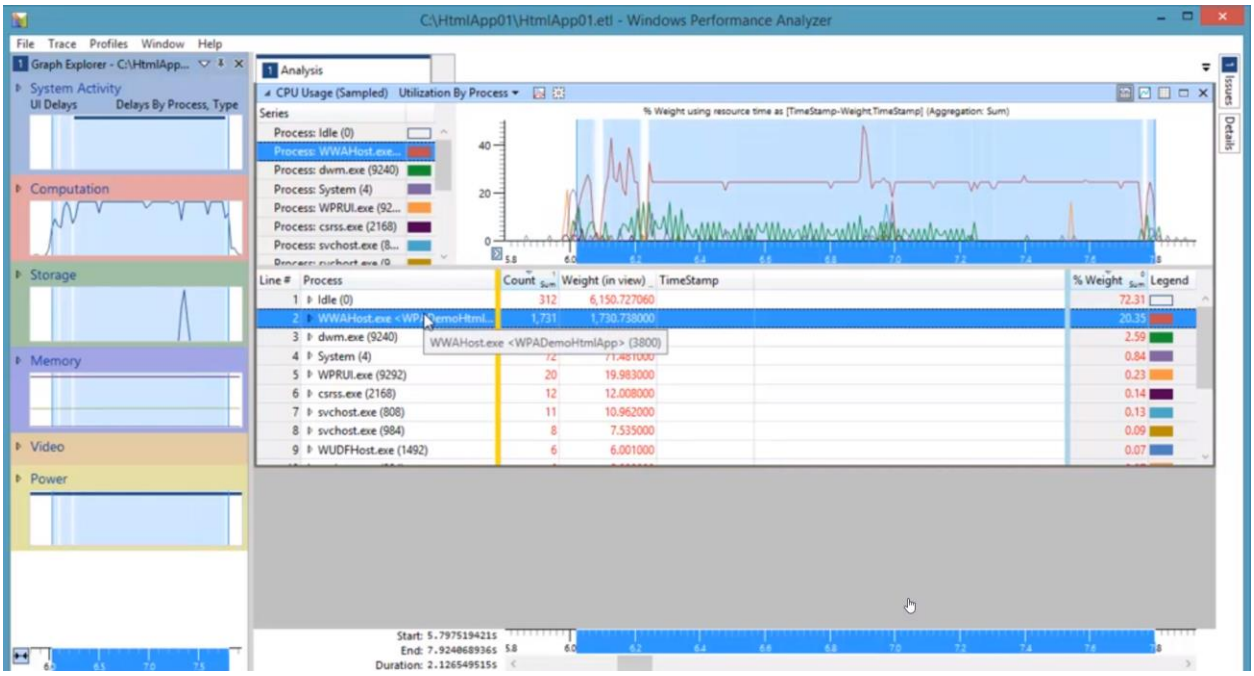


Figure 44: Windows Performance Analyzer

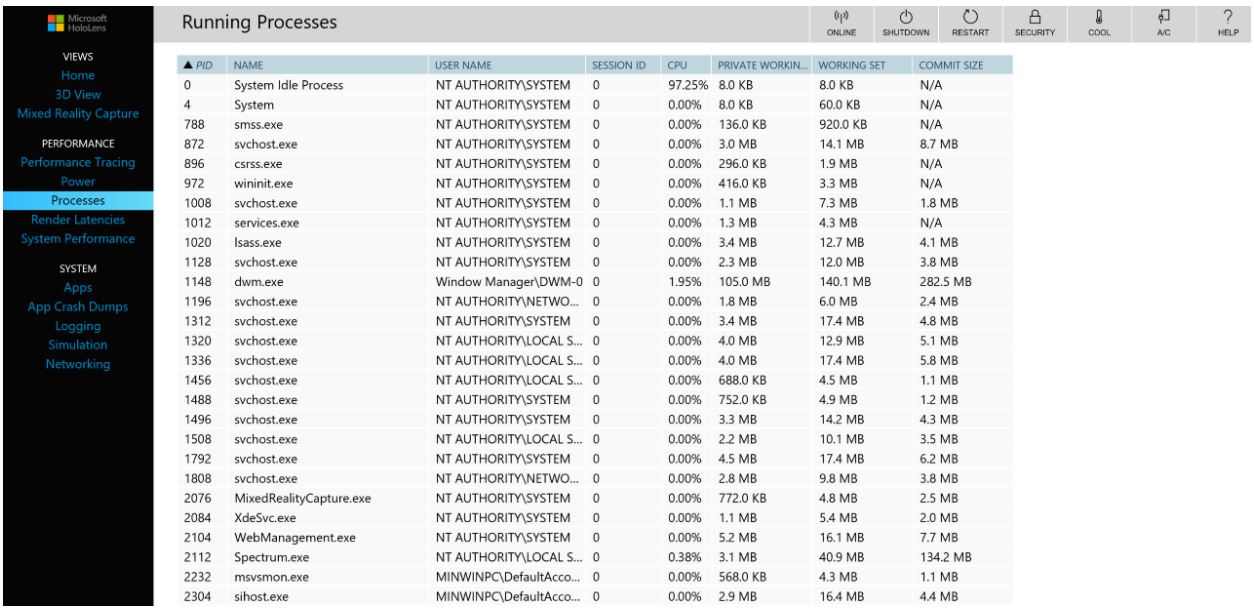
Tip: You will dramatically improve the success of your app if you become an expert in performance optimizing, as users respond extremely well to performing apps.

[8] The Windows Performance Analyzer is part of the Windows Assessment and Deployment Kit (Windows ADK).

Processes

The Processes screen is the equivalent of the Task Manager on a standard Windows installation. It shows the current running processes and, more importantly, the current CPU and memory usage. If there are performance issues with your experience, you can monitor the CPU and memory in real time.

This can also be a great tool to confirm that your app is running as well as it can, when the system is slow for one reason or another. The Processes screen shows both system and user applications running, so you can rule out one or the other.



▲ PID	NAME	USER NAME	SESSION ID	CPU	PRIVATE WORKIN...	WORKING SET	COMMIT SIZE
0	System Idle Process	NT AUTHORITY\SYSTEM	0	97.25%	8.0 KB	8.0 KB	N/A
4	System	NT AUTHORITY\SYSTEM	0	0.00%	8.0 KB	60.0 KB	N/A
788	smss.exe	NT AUTHORITY\SYSTEM	0	0.00%	136.0 KB	920.0 KB	N/A
872	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	3.0 MB	14.1 MB	8.7 MB
896	csrss.exe	NT AUTHORITY\SYSTEM	0	0.00%	296.0 KB	1.9 MB	N/A
972	wininit.exe	NT AUTHORITY\SYSTEM	0	0.00%	416.0 KB	3.3 MB	N/A
1008	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	1.1 MB	7.3 MB	1.8 MB
1012	services.exe	NT AUTHORITY\SYSTEM	0	0.00%	1.3 MB	4.3 MB	N/A
1020	lsass.exe	NT AUTHORITY\SYSTEM	0	0.00%	3.4 MB	12.7 MB	4.1 MB
1128	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	2.3 MB	12.0 MB	3.8 MB
1148	dwm.exe	Window Manager\DWM-0	0	1.95%	105.0 MB	140.1 MB	282.5 MB
1196	svchost.exe	NT AUTHORITY\NETWO...	0	0.00%	1.8 MB	6.0 MB	2.4 MB
1312	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	3.4 MB	17.4 MB	4.8 MB
1320	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	4.0 MB	12.9 MB	5.1 MB
1336	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	4.0 MB	17.4 MB	5.8 MB
1456	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	688.0 KB	4.5 MB	1.1 MB
1488	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	752.0 KB	4.9 MB	1.2 MB
1496	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	3.3 MB	14.2 MB	4.3 MB
1508	svchost.exe	NT AUTHORITY\LOCAL S...	0	0.00%	2.2 MB	10.1 MB	3.5 MB
1792	svchost.exe	NT AUTHORITY\SYSTEM	0	0.00%	4.5 MB	17.4 MB	6.2 MB
1808	svchost.exe	NT AUTHORITY\NETWO...	0	0.00%	2.8 MB	9.8 MB	3.8 MB
2076	MixedRealityCapture.exe	NT AUTHORITY\SYSTEM	0	0.00%	772.0 KB	4.8 MB	2.5 MB
2084	XdeSvc.exe	NT AUTHORITY\SYSTEM	0	0.00%	1.1 MB	5.4 MB	2.0 MB
2104	WebManagement.exe	NT AUTHORITY\SYSTEM	0	0.00%	5.2 MB	16.1 MB	7.7 MB
2112	Spectrum.exe	NT AUTHORITY\LOCAL S...	0	0.38%	3.1 MB	40.9 MB	134.2 MB
2232	msvsmon.exe	MINWINPC\DefaultAcco...	0	0.00%	568.0 KB	4.3 MB	1.1 MB
2304	sihost.exe	MINWINPC\DefaultAcco...	0	0.00%	2.9 MB	16.4 MB	4.4 MB

Figure 45: Processes in the HoloLens Device Portal

System performance

To get a real-time indication of how the device is performing, use the System Performance screen of the device portal. You can monitor the instantaneous system-on-chip (SoC) power utilization, averaged over one minute. If you have a drain on battery usage while using your app, this will most likely show up in the SoC metric or the overall system power consumption graph.

You can also monitor the live frame rate to understand if your app is either not delivering the necessary 60 frames per second or if the frame rate is fluctuating. As described in the section on Frame , not having a consistent frame rate can severely hinder the experience of the app and cause a bunch of issues.

You can also monitor GPU usage, CPU usage, I/O reads and writes, network usage, and memory used—all in real time. Having insight into the metrics, if there are issues with your app, is valuable. The system performance is part of your suite of tests for any HoloLens app.

www.dbooks.org

In general, follow these basic steps:

1. Create a test plan for your experience.
2. At a minimum, run the test plan for every update to the app or new feature.
3. Monitor system performance when testing on the device.
4. Repeat.

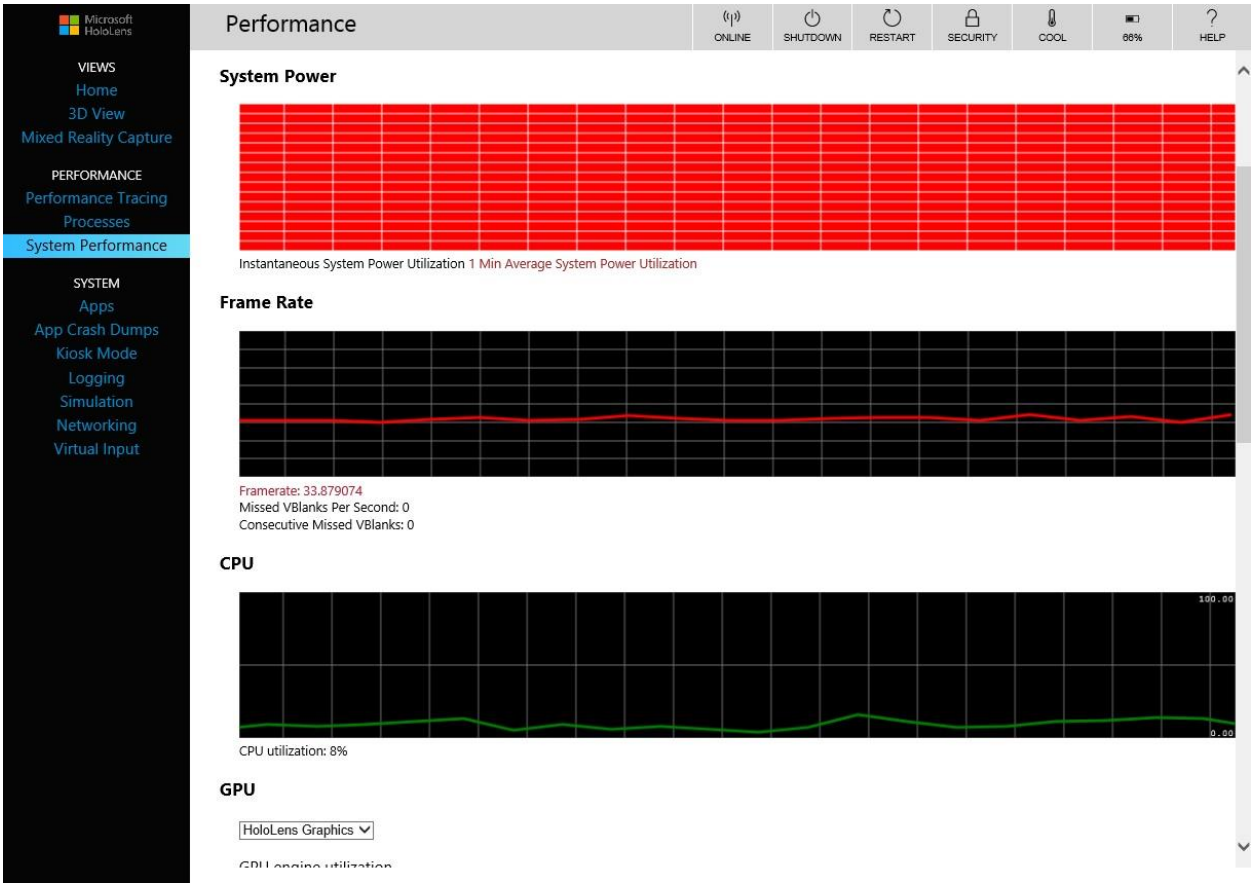


Figure 46: HoloLens System Performance Graphs

System tools

In addition to the specific tooling, there are also many extra system screens in the device portal. You can sideload[9] apps, get dumps for when apps crash (and they will), manage kiosk mode (which locks the device to a single app), set up logging integration with various frameworks, and monitor networking.

[9] Sideloading apps happens when an app package is loaded directly onto a device, rather than through official channels like a store.

One cool feature is on the Simulation tab, on which you can capture the spatial mapping currently on the device. You can save this as a file that can then be reused by others on your team.

Performance

As we have seen, the most important performance measurement is making sure 60 frames are delivered to the operating system every second. This ensures that most of the symptoms described in Chapter 8 “Hologram Quality” don’t occur. To test frame rate performance for production, you need a real HoloLens device and the System Performance Tool. As good as the emulator is, system resources from other processes have an influence on the Hyper-V image that runs the emulator. The initial tests on the emulator are valuable to give an indication of performance, though.

Frame rate isn’t the only performance concern. If you have an excessive amount of processing to uphold the frame rate, then your power consumption is also going to go up. Use the System Performance tool in the developer portal to monitor power consumption.

The third major component in testing performance is memory usage. The HoloLens device only has 2 GB in total, so there isn’t a lot to use from the start. You will have to be smart about what you store in memory at any one time and take advantage of the hardware acceleration where possible.

Environmental testing

HoloLens apps are software applications written in familiar tools, using familiar approaches. What’s different is that they are running on a device that interacts with the physical environment surrounding the user. This is a whole new area of testing for traditional software developers.

Testing how a HoloLens application interacts with the physical environment is not done through unit or integration tests. You must get physical and put yourself in the shoes of your users.



Tip: Get both actual users and non-developers to test the experience. Tests with people who haven’t been part of the development process often result in more natural results.

Light

The HoloLens creates a spatial 3D model using infrared light, which in turn means that natural light can impact the mapping. When testing for light, these are some of the scenarios to check:

- **Dimly lit:** Turn off the light, either completely or mostly. The HoloLens sensors don’t work well in the dark, and it is worth knowing how your experience will react.
- **Different color**: Find various colors to test as light source. Red especially can be problematic.

www.dbooks.org

- **Dynamic light**: Change the color source and intensity at various speeds. How does your app react in flashing light, for example?
- **Digital light**: If you have your own light source defined in your experience, how does that work with the real world? Does it look out of place? Is it too bright?

Sound

Spatial sound is a key part of the HoloLens experience, and it is easy to get it confused with real-world sound. After all, that's the whole point! You want your digital sound to blend with the real world. What you don't want is overwhelming or "dirty" sound. Some scenarios to test are:

- **Ambient noise**: Loud background noise such as traffic or music can diminish the effect of your designed audio. It can also make it harder to use voice commands.
- **Conversation**: Any background conversation can potentially be recognized as voice commands, or it may pollute the intended voice commands.
- **Spatial audio**: Ensure spatial audio attached to holograms perform as expected. Move around the hologram and check the sound is 100 percent accurate in the spatial model.
- **Occlusion**: As discussed in the "Sound " section, in order to fully manifest the illusion of mixed reality, sound must behave naturally, including when it's hidden behind physical and digital objects.

Movement

People are individually different, which means so are your users. They will be different heights, have different mannerisms, and move about the room in very different ways. You have to make sure that your experience works consistently for as many of those users as possible. Include these scenarios in your testing where possible:

- **Head**: Users always move their head in different ways. Some people have a unique way of moving their head when they walk, when they talk, etc. Test as many movements of your head as you can think of.
- **Body**: Everybody is different. Different heights. Different body shapes. Different postures. Test for as many as you can.
- **Entry**: How do holograms behave when users interact with them? What happens if someone steps into a hologram?

Environment

What sets the HoloLens apart is the physical environment around you. When you design and develop your app, there is no way you can anticipate all of the locations users will be using your app, but you can take steps to anticipate some of them. When you test the environment around you, keep the following in mind:

- **Changes**: The real world changes constantly. Doors open, chairs are moved, curtains are drawn, etc. Test what happens if you place a hologram on a surface that moves or disappears.
- **Clutter**: What happens if a room doesn't have any empty or flat surfaces? If there are uneven objects everywhere, such as toys or boxes, will your app still function correctly?

- **Missing mapping**: The HoloLens will only map what it can see, so what happens if there are parts of the 3D model of the spatial mapping that is missing?
- **Materials**: Certain materials don't work well with the infrared mapping of the device and will create unexpected effects. Glass will almost always be invisible, and a mirror is a magic portal into another dimension (not really, it just kind of is invisible).

Creating a testing environment

How do you go about testing the environment in practice? If you have a suitable space, it is worth setting up a dedicated test location. It must be representative of who the users of the app are and how they interact with your app. If you know certain equipment or furniture will be present where the users will be, create a mockup of the shape. Mimic the expected lighting and the shape of walls and doorways, and get background audio as well. It could be the murmur of a crowd, the hum of machinery, or whatever fits the intended environment.

The more accurate you can be in the test location, the better off you will be when it's time to put it into production. For example, if you are designing and building an experience to help technicians fix an elevator, then set up the sections of the elevator that you can manage. This might just be the parts needing servicing, or if resources allow it, a more complete test location.

Perception Simulation

Automating testing for HoloLens is difficult because the real world changes all the time. Setting up a designated test location helps, but it doesn't solve the need to carry out tests that can be repeated. A unique tool (for HoloLens only) is Perception Simulation.

You can simulate any movement that the user can perform, but in code. The library sends fake human and world input data to your app so you can automate your tests. It might be strange to use at first, as it takes over the real device or emulator and any "real" input is ignored. You can simulate both head movements and gestures as well as voice commands.

Perception Simulation bypasses the live sensors on a device and sends simulated input to applications running on the device. The applications receive all the simulated (fake) input events through the same APIs they always use, and they can't tell the difference between running with real sensors versus running with Perception Simulation.

Here's how you set it up:

1. Make sure you have installed the HoloLens emulator.[10]

2. Add references to the following binaries:
    a. **`PerceptionSimulationManager.Interop.dll`** – Managed C# wrapper for Perception Simulation.

[10] https://developer.microsoft.com/en-us/windows/mixed-reality/install_the_tools

www.dbooks.org

b. **`PerceptionSimulationRest.dll`** – Library for setting up a web-socket communication channel to the HoloLens or emulator.
c. **`SimulationStream.Interop.dll`** – Shared types for simulation.

3. Add binary **`PerceptionSimulationManager.dll`** to your project.

Code Listing 10: Perception Simulation Example (based on sample from Microsoft)

```
RestSimulationStreamSink sink = await RestSimulationStreamSink.Create(
    // use the IP address for your device/emulator
    new Uri("http://169.254.94.136/"),
    // no credentials are needed for the emulator
    new System.Net.NetworkCredential("", ""),
    // normal priority
    true,
    // cancel token
    new System.Threading.CancellationToken());


IPerceptionSimulationManager manager =
PerceptionSimulationManager.CreatePerceptionSimulationManager(sink);


// Simulate Bloom gesture, which should cause Shell to disappear.
manager.Human.RightHand.PerformGesture(SimulatedGesture.Home);
Thread.Sleep(2000);


// Simulate Bloom gesture again... this time, Shell should reappear.
manager.Human.RightHand.PerformGesture(SimulatedGesture.Home);
Thread.Sleep(2000);


// Simulate a Head rotation down around the X axis.
// This should cause gaze to aim about the center of the screen.
manager.Human.Head.Rotate(new Rotation3(0.04f, 0.0f, 0.0f));
Thread.Sleep(300);
```

```
// Simulate a finger press & release.
manager.Human.RightHand.PerformGesture(SimulatedGesture.FingerPressed);
Thread.Sleep(300);
manager.Human.RightHand.PerformGesture(SimulatedGesture.FingerReleased);
Thread.Sleep(2000);

// Simulate a second finger press & release.
manager.Human.RightHand.PerformGesture(SimulatedGesture.FingerPressed);
Thread.Sleep(300);
manager.Human.RightHand.PerformGesture(SimulatedGesture.FingerReleased);
Thread.Sleep(5000);

// Simulate a Head rotation towards the upper-right corner.
manager.Human.Head.Rotate(new Rotation3(-0.14f, 0.17f, 0.0f));
Thread.Sleep(300);

// Simulate a third finger press & release.
manager.Human.RightHand.PerformGesture(SimulatedGesture.FingerPressed);
Thread.Sleep(300);
manager.Human.RightHand.PerformGesture(SimulatedGesture.FingerReleased);
Thread.Sleep(2000);

// Simulate Bloom gesture again... bringing the Shell back once more.
manager.Human.RightHand.PerformGesture(SimulatedGesture.Home);
Thread.Sleep(2000);
```


www.dbooks.org

Comfort

The issue of user comfort is up to you as a developer to deal with—there are no built-in API functions from Microsoft. Rather, it is important to keep the comfort of the mixed reality experience in mind. This is closely related to the section on identifying issues with hologram quality, such as jitter, judder, drift, and other issues that can cause discomfort to the user.

A small but important detail is managing the clip plane of the experience. The clipping plane is the distance from the user at which the hologram stops being rendered. For HoloLens applications, the distance recommended by Microsoft is 0.85 meters. This means that once the user is 0.85 meters or less from the hologram, it isn't rendered to the user's view, and disappears.

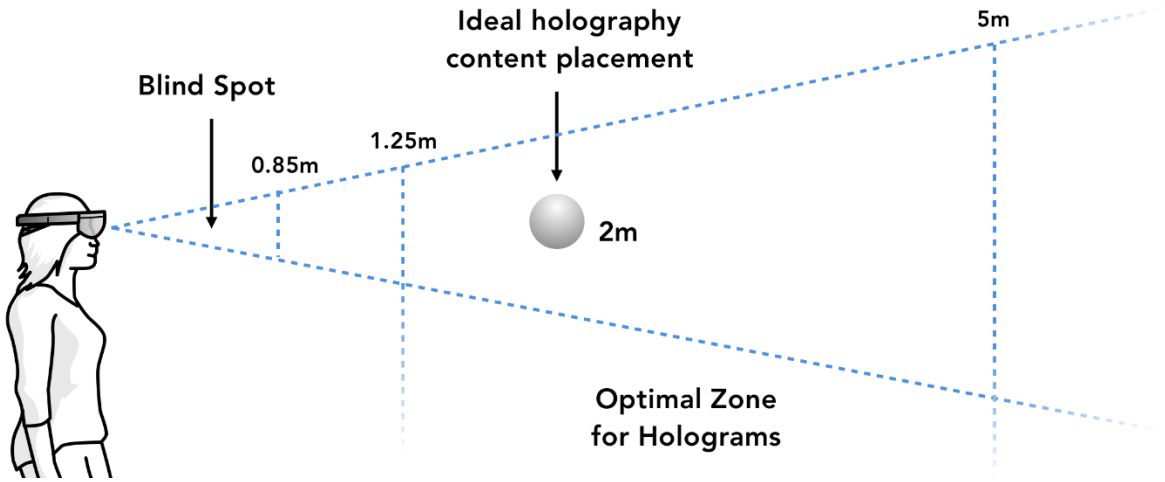


Figure 47: Recommended Distances for Holograms

There are scenarios where the clipping plane can be more or less than the recommended distance, but make sure you make a conscious decision to alter it. Users will expect the experience to be consistent with other HoloLens apps. Rendering the holograms in the user's blind spot (Figure 47) can result in a sense of uneasiness and lost confidence in the experience.

Power management

The battery time on a HoloLens is 2–3 hours on average. Your app should not affect this benchmark at all. Unless your experience only needs to run for a few minutes at the time, make sure to perform some stress tests on the battery consumption of the app.

You also need to ensure that powering the HoloLens off and turning it back on again will allow users to continue the experience in a way that makes sense. If your app relies on multiple sessions to complete an objective, don't make the user perform the same set of actions every time they start up. Instead, let them continue where they left off.

Chapter 10 Sharing Holograms

One of the most exciting features of the HoloLens is the collaboration aspect. It is exciting and groundbreaking to first create and manipulate holograms. Having interaction between digital and physical objects in a convincing way is inspiring and it opens up a lot of possibilities. However, when you introduce multiple users to the same HoloLens experience, the impact is exponentially increased. Having two or more users viewing the exact same experience is both powerful and persuasive.



Figure 48: Sharing the Same Hologram between Devices

Designing shared experiences

This e-book has touched on a lot of design decisions that need to be made for HoloLens experiences. The same thoughts and concerns apply when creating shared experiences, as well as a whole new dimension to consider.

Sharing style

Content can be shared in a few ways. You can be presenting an experience to onlookers, where you are in control of the content. This could be very useful for teaching students about the human body, for example. The teacher would go through the material, and all students would see it in real time. The teacher can share notes in real time that students can interact with and copy.

You can, of course, collaborate on the same experience between users. In the previous scenario, students could be invited to interact with the holograms and manipulate parts of it to engage in the learning.

Finally, you can have one user guide one or more users through an experience. For example, a teacher could assist with assignments that students are solving by guiding them through the experience.

www.dbooks.org

User location

Understanding where users will be located in a sharing experience is important. You can have co-located users that share the same physical space, users that are remote and each in their own physical space, or a combination of the two. Knowing where users will be when sharing an experience can help you design the communication model and consider how users are represented to each other, which parts of the experience is shared, whether you need to consider certain physical features that may not be present, and more.

Ideally, users are in the same location, as that gives by far the best shared experience on HoloLens. However, this isn't always the case, and you might be able to solve a business case by collaborating over a distance.

Timing

Typically, you would expect users to be collaborating at the same time in a shared experience. Two or more users will share an experience and engage each other at the same time. This means state is managed in real time, and there is usually a defined conclusion to the collaboration.

However, at times users will work asynchronously on a shared experience. This creates some hurdles to solve, as there is a need for managing state to ensure the collaboration is successful, the room might change between sessions, so the mapping will need updating and more.

Interaction

The interaction model is the same as for a single user experience, although you now need to manage multiple users potentially interacting with the same object at the same time. This requires a common reference point that is updated in real time to a common data repository. This is commonly done using spatial anchors communicated throughout a centralized server instance.

From a development point of view, you need to manage the changes that each user makes and then update the central collection of coordinates.

Sharing service

Luckily, you don't have to create your own service to share spatial anchor coordinates. Using the Mixed Reality Toolkit, you get a helping hand in creating and configuring a sharing service. Go to the **Mixed Reality Toolkit** menu and select **Configure** > **Apply Mixed Reality Project Settings** (Figure 49) to enable it.

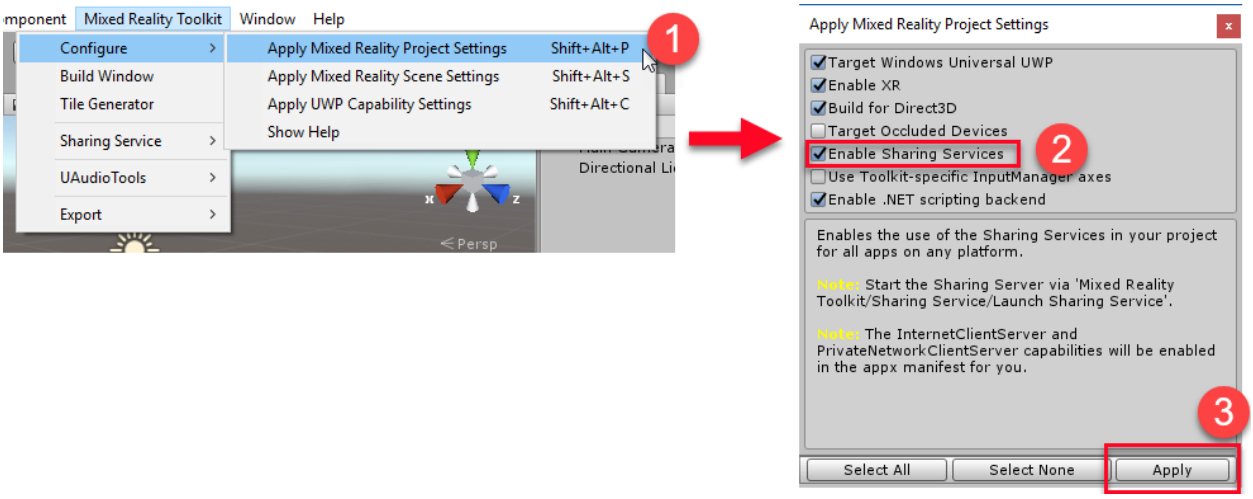


Figure 49: Enabling Sharing Service Using the Mixed Reality Toolkit

This will install the necessary components into your project, and when you want to start a sharing experience, you can again use the Mixed Reality Toolkit menu to launch the sharing service (Figure 50). You only have to do this once for one PC, per sharing experience.

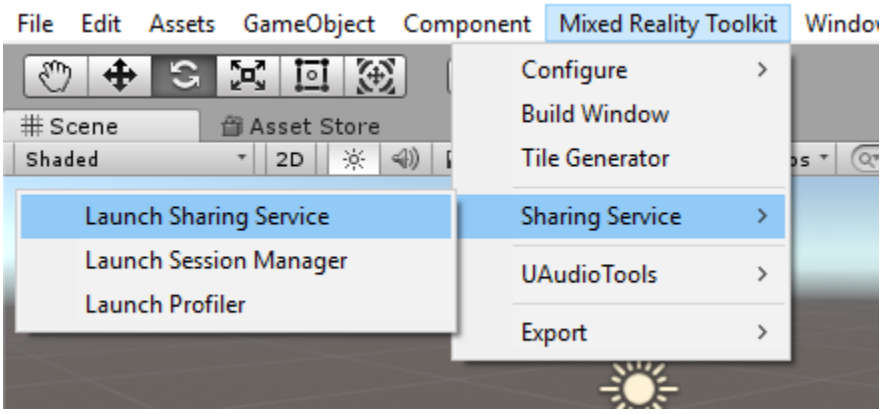


Figure 50: Launching Sharing Service Using Mixed Reality Toolkit

This will start the sharing service on the PC you launched it on, and this is the instance other HoloLens devices will connect to.

www.dbooks.org

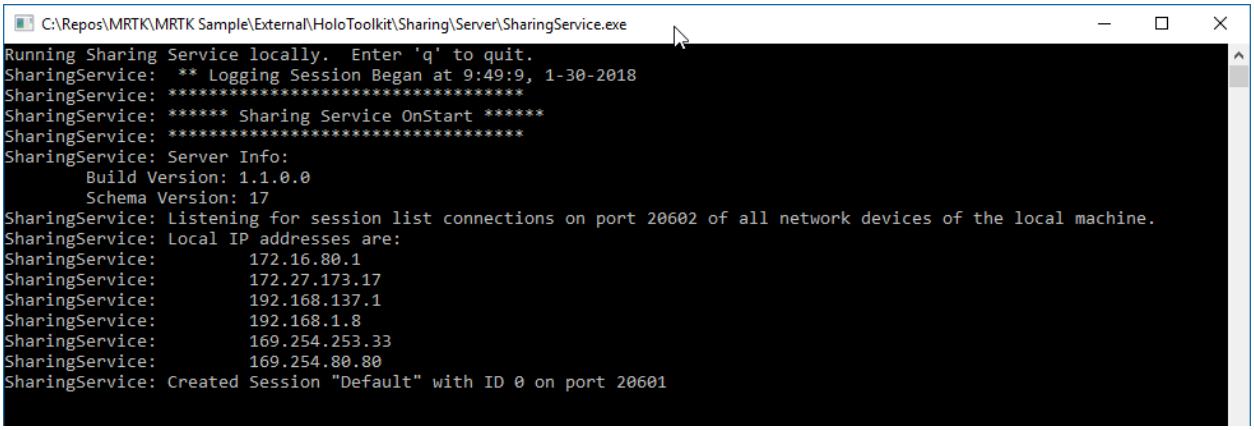


Figure 51: Sharing Service Running on Local Network

Placement

In a sharing experience, the mixed-reality illusion of placing digital assets in your real world must be maintained. As a developer, you must communicate any change to the placement of a hologram, so that all the other connected devices can update their experience accordingly. At the same time, you must also receive the updated placements from other users—and it all must happen in real time.

Code Listing 11: Receiving Updated Hologram Position

```
// Use the CustomMessages singleton to transmit messages between devices.
CustomMessages.Instance.MessageHandlers[CustomMessages.TestMessageID.StageT
ransform] = this.OnStageTransfrom;



void OnStageTransfrom(NetworkInMessage msg)

{
    // Update the position of the hologram.

    transform.localPosition = CustomMessages.Instance.ReadVector3(msg);

    transform.localRotation = CustomMessages.Instance.ReadQuaternion(msg);

}
```

Spatial anchors will persist when users exit your app using the bloom gesture or even if they turn the HoloLens off. Spatial anchors do not persist if the app is uninstalled, and you can't share them between different apps, either.

Chapter 11 Next Steps

HoloLens development, and mixed reality in general, is full of promise. With mostly existing knowledge and a few new tricks, you can get started quickly. This book has provided some of the first steps, both for code and design, and will give you a great start on your HoloLens development journey.

The next step is to start your own project. I encourage you to go through the tutorials at the HoloLens Academy. Some of the exercises there will be like the ones in this book, but you will get to building a real HoloLens app in no time.

Next, build your own experience, and use this book as well as the official documentation from both Microsoft and Unity to advance and excel.

I hope you have enjoyed the lessons and tips in this book, and maybe we'll meet soon in a new experience with holograms and magic.

"Oh, no! It's finally happened! They've disappeared into the simulation! Trapped! As my favorite sci-fi scenarios have foretold!"

– Dr. Disaster, Gunnerkrigg Court

www.dbooks.org