Microsoft

Programming for Mixed Reality with Windows 10, Unity, Vuforia, and UrhoSharp

Professional

Dawid Borycki

Microsoft

Programming for Mixed Reality with Windows 10, Unity, Vuforia, and UrhoSharp

Dawid Borycki

PROGRAMMING FOR MIXED REALITY WITH WINDOWS 10, UNITY, VUFORIA, AND URHOSHARP
Published with the authorization of Microsoft Corporation by:
Pearson Education, Inc.
Copyright © 2019 by Pearson Education, Inc.

ISBN-13: 9781509306879
ISBN-10: 1509306870

Library of Congress Control Number: 2018952317

1 18

Editor-in-Chief: Brett Bartow

Executive Editor: Laura Norman

Development Editor: Kate Shoup/ Polymath Publishing

Managing Editor: Sandra Schroeder

Senior Project Editor: Tracey Croom

Production Editor: Danielle Foster

Copy Editor: Kate Shoup/ Polymath Publishing

Indexer: Valerie Haynes Perry

Proofreader: Dan Foster

Technical Editor: John Ray

Cover Designer: Twist Creative, Seattle

Compositor: Danielle Foster

Graphics: Vived Graphics

Without hard work, nothing grows but weeds.

—Gordon B. Hinckley

Contents at a Glance

Contents

About the Author

DAWID BORYCKI is a software engineer and biomedical researcher experienced in Microsoft technologies. He has completed a broad range of challenging projects involving the development of software for device prototypes (mostly medical equipment), embedded device interfacing, and desktop and mobile programming.

Introduction

If you've ever watched any of the *Iron Man* movies starring Robert Downey Jr., you're probably familiar with Jarvis. Jarvis, short for "Just a Rather Very Intelligent System," is a highly advanced computerized AI system. Recently, Jarvis inspired Mark Zuckerberg to build his own AI-powered home-automation system (https://youtu.be/ZGLPxEv_EWo).

Incredibly, you can build your own wearable Jarvis-like system by combining the thrilling power of novel holographic devices like HoloLens with Microsoft Cognitive Services and Microsoft Mixed Reality. Such systems can quickly process visual content and describe it via spoken words or text displayed on screen. With just a few voice commands or air gestures, you can ask your Mixed Reality "Jarvis" system to look up what you see on the web and tell you more about it. You could use this system with AI to, say, develop a device to support visually impaired persons (https://youtu.be/R2mC-NUAmMk).

In this book you will learn how to develop AI-powered Mixed Reality apps using various strategies and technologies, including Universal Windows Platform (UWP), Microsoft Cognitive Services, Unity, Vuforia, and Xamarin UrhoSharp. First, you will learn how to set up the development environment, install the necessary tools, and use the HoloLens emulator and Mixed Reality simulators. Then you will start writing UWP two-dimensional apps that run across every Windows 10 device (including HoloLens and immersive headsets) and adjust their views and functionalities to specific devices. Subsequently, you will learn how to transfer images from the world-facing camera of the headset to the machine learning modules of the computer vision service to obtain image descriptions that are spoken by the device or displayed in text form. Afterward, you will use sensor readings from the mobile device to control content displayed on the Windows Mixed Reality headset.

After learning about 2D app development, you will move on to building 3D apps. You'll learn how to do this from the ground up. First, you'll explore the fundamental concepts of 3D graphics. You will then learn how to use Unity Editor to build 3D Mixed Reality apps, including setting up scenes, adding built-in and custom 3D objects, and formatting these objects with materials to create holograms. Afterward, you will learn how to make your holograms behave like real objects with various physics simulations and to interact with these holograms through air gestures and voice commands. Next, you will learn how to attach augmented reality to real objects with Vuforia. Finally, you will build a Mixed Reality app with UrhoSharp—a library for writing cross-platform 3D apps. This will enable you to transfer your skills to other platforms, such as iOS or Android.

Audience and Expected Skills

This book is devoted to developers, students, engineers, enthusiasts, designers, scientists, and researchers who would like to use their existing programming skills to develop software for Windows Mixed Reality (HoloLens and immersive headsets) with Unity, Vuforia, and UrhoSharp. I assume the reader knows fundamental aspects of C# programming and is experienced in Windows programming. I do not, however, assume any previous knowledge of Unity, Vuforia, or UrhoSharp.

Tools and Required Hardware

To implement all examples presented in this book, you will need a system that runs Windows 10 (Creators Update or later) and uses Visual Studio 2017 Community as the development environment.

Organization of This Book

The book is divided into three parts:

- **Part I: Fundamentals** Chapters 1 through 3 introduce Mixed Reality in Windows 10, define all related terms, show you how to prepare your development environment, and explain possible approaches to developing apps for Windows Mixed Reality.
- **Part II: Developing 2D UWP Apps** Chapters 4 through 7 cover 2D app development with a special focus on Universal Windows Platform (UWP) programming interfaces. Programmers familiar with UWP can easily port their existing apps to new holographic or Mixed Reality platforms and thereby extend the spectrum of their users.
- **Part III: Developing 3D Apps** Chapters 8 through 15 teach you how to create 3D Windows Mixed Reality apps with Unity, Vuforia, and UrhoSharp. These chapters assume no prior knowledge of those tools. The aim of this part is to guide you through all the steps and leave you with all the skills necessary to create exciting AI-powered 3D apps for Windows Mixed Reality.

Conventions

The following conventions are used in this book:

- **Boldface** type is used to indicate text that you type.
- *Italic* type is used to indicate new terms and file names.
- Code elements appear in a `monospaced` font.

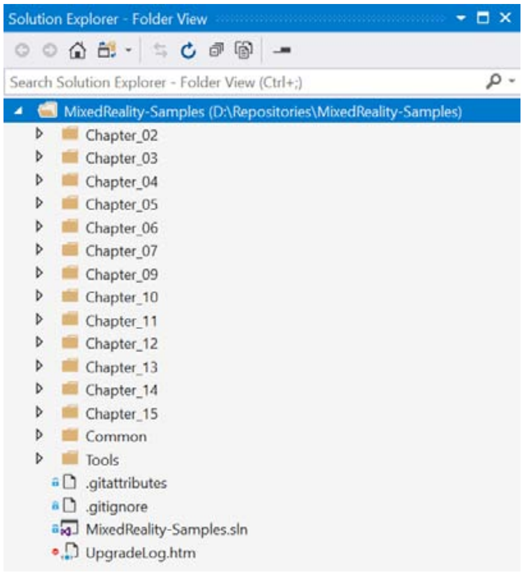
About the Companion Content

This book includes companion code to enrich your learning experience. The companion code for this book can be downloaded from the following page:

https://github.com/dawidborycki/MixedReality-Samples.

As shown in Figure I-1, the code is partitioned into subfolders, which correspond to particular chapters. To improve book readability, in many places I refer to the companion code rather than showing the full listings, so it is good to have the solution open while reading this book.

You can also find the companion files at *https://aka.ms/ProgMixedReality/downloads*.


Solution Explorer - Folder View
Search Solution Explorer - Folder View (Ctrl+;)

MixedReality-Samples (D:\Repositories\MixedReality-Samples)
Chapter_02
Chapter_03
Chapter_04
Chapter_05
Chapter_06
Chapter_07
Chapter_09
Chapter_10
Chapter_11
Chapter_12
Chapter_13
Chapter_14
Chapter_15
Common
Tools
.gitattributes
.gitignore
MixedReality-Samples.sln
UpgradeLog.htm

FIGURE I-1 The structure of the companion code.

Acknowledgments

The publication of this book would not have been possible without Loretta Yates and Laura Norman, who accepted my book proposal and provided initial feedback. I am grateful to John Ray for thoroughly checking every project presented in this book and providing useful, positive comments on the content. Many thanks, too, to Kate Shoup for copyediting the book. Finally, I appreciate ongoing support from my family: my wife, Agnieszka; my daughter, Zuzanna; and my son, Ksawery. I would achieve nothing without them.

Errata and Book Support

We have made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site at:

https://aka.ms/ProgMixedReality/errata

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

Stay in Touch

Let’s keep the conversation going! We’re on Twitter: @MicrosoftPress.

PART I

Fundamentals

CHAPTER 1

Introduction to Windows Mixed Reality

Windows Mixed Reality is a platform that delivers a completely new way of mixing digitally generated three-dimensional content with the real world or environment. To understand what this means, let's first discuss how people use traditional computing systems for work or fun. Typically, we work with two-dimensional screens, in which we use perspective drawing to depict three-dimensional objects. Although this approach is suitable for designing and implementing various elements, we cannot directly interact with digitally created objects. So, our ability to present or interact with what we create is limited. Windows Mixed Reality aims to suppress this barrier by providing a revolutionary way of computing. To that end, Windows Mixed Reality delivers software and hardware components that enable us to mix virtual and real worlds to accomplish much more than we ever could in traditional computing.

This chapter covers the concepts related to Windows Mixed Reality. It starts by defining virtual, augmented, and mixed realities. Then, it discusses Windows Mixed Reality headsets and the concept of holograms as it relates to Windows Mixed Reality. Afterward, the chapter discusses several Windows Mixed Reality–enabled utilities available in Windows 10 (with the Fall Creators update or above)—for example, Microsoft Paint 3D, which lets you create and interact with 3D objects through Windows Mixed Reality.

Virtual, Augmented, and Mixed Reality

Windows Mixed Reality (WMR) involves a broad range of software and hardware technologies, including virtual and augmented reality. There are also several devices supported by WMR. At the moment, only Microsoft HoloLens provides a true mixed reality experience. Other WMR headsets provide a virtual reality experience. To understand the key differences between these headsets, let's define virtual, augmented, and mixed reality.

Virtual Reality

Virtual reality (VR) refers to a realistic and immersive 3D simulation of an environment. This simulation is generated with the help of interactive software and hardware and is controlled by the user's voice, gestures, gaze, or body movements.

Predecessors of the current VR systems were called stereoscopes and were introduced during the 19th century by Sir Charles Wheatstone, David Brewster, and Oliver Wendell Holmes, who developed them independently (http://bit.ly/stereoscopes). During the early 20th century their ideas were extended by Sawyer Service Inc. to develop View-Master (http://bit.ly/viewmaster_sawyer). Nintendo made important advances in VR with its introduction of the VR-enabled Virtual Boy game console in 1995 (http://bit.ly/virtual-boy). The Virtual Boy came equipped with a controller and a viewer, which, like stereoscopes and the View-Master, generated a stereo pair of images. These images, called stereopair, created the illusion of depth in a 3D scene. In 1989, a company called VPL Research went even further. It developed a VR headset as part of a full-body outfit that contained sensors to measure body movement. This immersed the user even more in the VR simulation. Although these old VR devices did not succeed commercially, they are precursors of modern VR headsets.

Stereoscopy

Stereoscopy works by providing two two-dimensional images of the same environment. These images differ slightly from each other and are displayed independently for the left and right eye. The brain combines these two images to build a sense of depth in a 2D plane to create the virtual 3D scene.

Perhaps the easiest way to grasp this effect is to raise your thumb a few inches in front of your face and then quickly alternate blinking with your left and right eye. Notice how the location of your thumb appears to be different depending on which eye you use to view it, and how its correct location is restored when you have both eyes open. That's how stereoscopy works.

Currently, we are seeing a rapid revival of VR. In recent years, Facebook acquired a company called Oculus to further develop its VR headset technology; HTC released the Vive headset; Sony created PlayStation VR; and Google released the Cardboard, based on the View-Master, to enable users to cheaply convert nearly any smartphone into a VR headset. October 2017 brought the release of numerous new headsets dedicated for use with Windows Mixed Reality. (These are described in detail in the next section.)

Augmented Reality

Augmented reality (AR) is another technology that has recently taken off. In general, AR uses digital elements such as sound and images generated by a computer system to augment the human perception system.

The most exciting aspect of AR is its ability to overlay three-dimensional synthetic elements on real objects. AR can be further used to create an interaction between virtual and real objects. For example, with additional environment-understanding cameras, you can "touch" the synthetic objects and manipulate them with your hands or air gestures.

AR became well-known after the remarkable success of the AR-enabled game, Pokémon GO, which was itself preceded by Niantic's Ingress. These games inspired manufacturers of mobile devices to create several programming kits to speed up AR development. Although modern mobile phones are equipped with powerful processors and various inertial sensors, they do not exhibit the full potential of AR, as they are very rarely equipped with sensors to understand the environment, which precisely sense the phone's distance to various real objects in the scene. Moreover, displays on current mobile devices cannot generate stereoscopic 3D objects without the use of additional hardware (like Cardboard, for example). Instead, mobile devices overlay digital elements on real objects captured by the camera. Then, both the synthetic and the real objects are displayed on the same screen. In other words, mobile devices offer a relatively quick way to blend real with synthetic objects, but the AR experience is limited.

The Microsoft HoloLens takes a completely different approach. It uses a head-mounted AR module that creates stereoscopic 3D images of synthetic objects, which are displayed directly on the real scene. Because HoloLens does not use an opaque display, real and synthetic objects "live" in the same environment. Moreover, HoloLens is equipped with numerous cameras and sensors to probe your near environment. This enables HoloLens to understand the scene in which you are embedded, allowing real objects to occlude and interact with virtual ones. HoloLens can also recognize the user's hand gestures and voice input, and can generate stereoscopic sound to provide the best possible AR experience.

Virtuality Continuum

VR and AR are tightly coupled to the concept of the *virtuality continuum*, which was introduced in a paper by Paul Milgram and Fumio Kishino (http://bit.ly/mixed_reality). You can think of the virtuality continuum as a spectrum of real and virtual environments. The real environment (RE) is composed only of real objects. The virtual environment (VE) comprises only digitally created elements.

As shown in Figure 1-1, VR is closer to the VE extreme, though not all the way there. This is because VR relies primarily on digital elements but does not yet allow for a completely virtual environment and requires the use of real elements (like hand and body movements) to control the simulation. In contrast, AR is closer to the RE extreme because AR blends digitally created objects with reality. With AR, synthetic objects are displayed in the real environment, and the user can both interact with synthetic objects and occlude real ones.

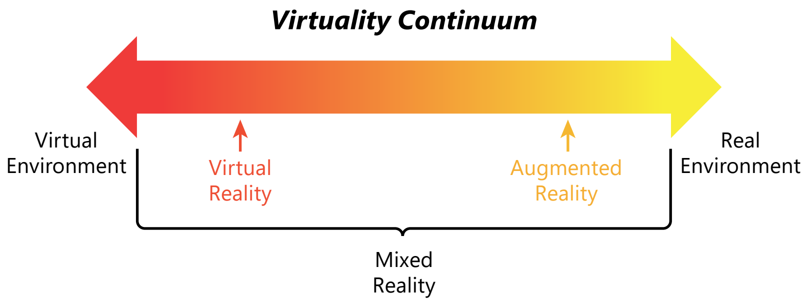


FIGURE 1-1 The virtuality continuum.

Mixed Reality

In Figure 1-1, VR and AR are shown as points on the virtuality continuum. These points represent only specific combinations (or blended ratios) of virtual and real environments. The set of all other combinations of blending virtual and real environments is denoted as mixed reality (MR). As shown in Figure 1-1, mixed reality is more general than VR and AR separately. MR includes both AR and VR as well as many more points on the virtuality continuum.

The blending of virtual and physical worlds enabled by MR advances human–device interaction to a whole new level, introducing a broad range of possibilities. With MR, developers are now equipped with all necessary tools to combine real environments, human input, and digital content to provide experiences restricted only by their imagination.

When desktops were the most popular computing systems, people mostly used a mouse and keyboard to control them. For most people, this became a convenient way to communicate with the device. However, touchscreens on mobile devices have significantly simplified human–device interaction. Every day, experiences show that even small children can easily unlock a smartphone to watch their favorite videos on YouTube. This is because touchscreens provide a more natural way to interact with a device. A similar significant step forward is now being made by MR.

MR provides an entirely new and exciting way to create apps that blend seamlessly and naturally with the real world. Users can now control apps using air gestures and head movements. More importantly, digital content can interact with the physical world because MR devices can be equipped with sensors that understand the user's near environment. In other words, both the device and the app can now perceive your near environment. Accordingly, developers can now create apps by placing virtual content in the user's surroundings. In practice, this means various communicator apps can provide virtual content (avatars) representing other people. As a result, users can talk to people located thousands of miles away as if they were in the same room!

Other currently explored MR applications include (but are not limited to) the following:

- **Tutorial or teaching** With these types of apps, students can see complex structures in 3D (such as images of the human body) and easily highlight or zoom selected parts to understand how they work. A good example is shown here: http://bit.ly/case_western.
- **Visualizations** These types of apps enable users to visualize objects on real physical surfaces in full size—useful when buying items like furniture, cars, and so on. Here's an example: http://bit.ly/HoloLens_Volvo.
- **Remote help** With these types of applications, users can employ an MR headset to transfer information about a problem in their near environment to a remote consultant. This consultant can then add digital content to provide precise instructions on how to fix the problem. Several companies now use this technology on an everyday basis, including thyssenkrupp (http://bit.ly/HoloLens_thyssenkrupp).

Windows Mixed Reality Hardware

To provide MR experiences, Windows Mixed Reality requires hardware components. One of these components is a head-mounted display (HMD). The other is either a clicker (for HoloLens; see Figure 1-2) or a motion controller (for Mixed Reality headsets; see Figure 1-3), which the user holds in his or her hands.

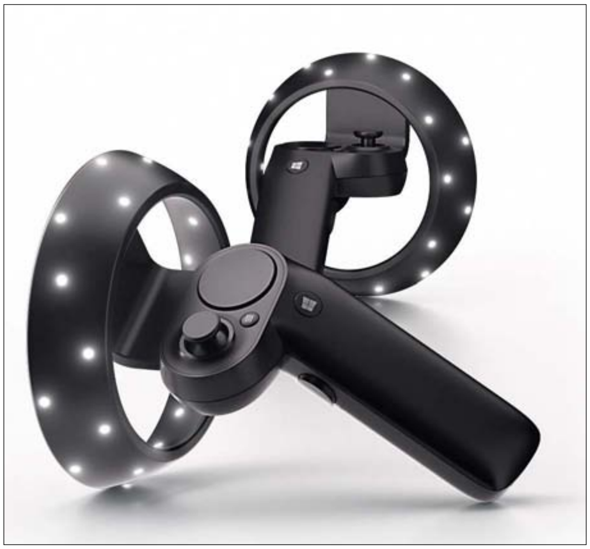


FIGURE 1-2 HoloLens clicker. (Courtesy of https://developer.microsoft.com.)

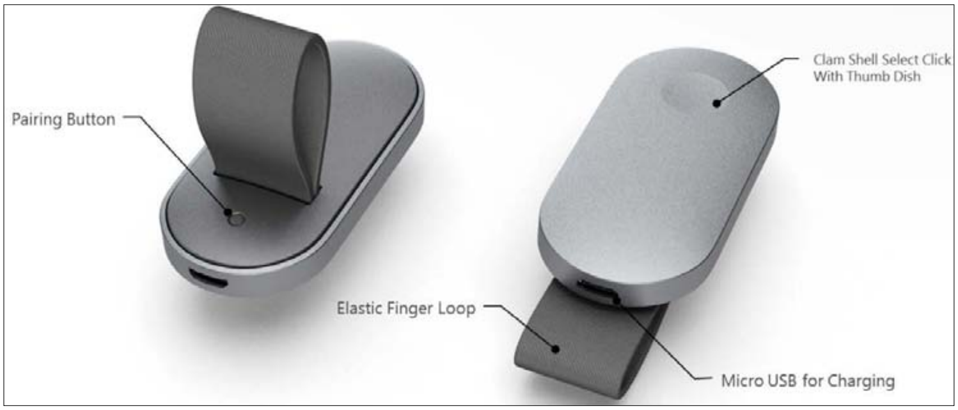


FIGURE 1-3 Windows Mixed Reality motion controllers. (Courtesy of https://developer.microsoft.com.)

There are two distinct groups of Windows Mixed Reality HMDs:

- **Holographic headsets** Holographic headsets use *see-through* displays, which embed synthetic 3D objects in the real environment. Accordingly, holographic headsets are on the AR side of the virtuality continuum.

- **Immersive headsets** Immersive headsets have *opaque displays*, which block the physical environment. As such, they are on the VR side of the virtuality continuum.

Both holographic and immersive devices fall under the class of MR devices. The following sections describe HMDs in a more detail.

Holograms

Microsoft defines a hologram as an "*object made of light and sound that can appear in the world around you*." This definition can give pause to people familiar with holography, however. In holography, the term hologram is reserved to describe a recording (stored in the appropriate medium) that encodes complex information (amplitude and phase) about the optical field in the form of an interference fringe pattern. This pattern can be then illuminated by a laser light to display a fully 3D image of the object without the help of any special headset or intermediate optics. Here, conforming to the Microsoft definition, I will use the term hologram to describe synthetic objects created by a holographic device, keeping in mind that this does not precisely adhere to the traditional meaning of the term.

Microsoft HoloLens

Microsoft HoloLens (see Figure 1-4) is the leading untethered, head-mounted device for providing Mixed Reality experiences.

Note *Untethered* means that the HoloLens is an independent device. It is a wearable, fully independent computing system. It does not require a separate PC.

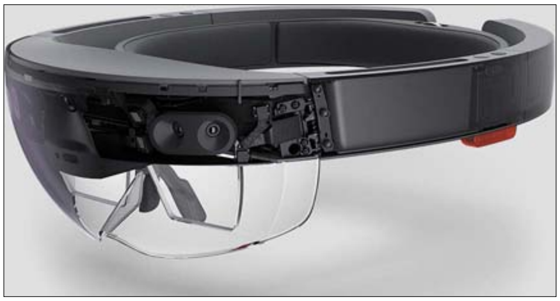


FIGURE 1-4 Microsoft HoloLens. (Courtesy of https://developer.microsoft.com.)

The HoloLens features see-through holographic lenses. These lenses enable the user to view his or her surroundings. They also double as a screen to enable the user to view stereo pairs of digital 3D objects, which are generated by two RGB LED projectors. (See Figure 1-5.) Although these projectors are much smaller than the digital projector you might use at work or at home, the principles behind their operation are similar.

Note The HoloLens screens, which are translucent, use waveguides. These waveguides reflect the light projected onto the screen by the RGB LED projectors while also transmitting light reflected from real objects contained in the surrounding scene. The screens then blend these images to provide an MR experience. (Of course, the technical implementation of the HoloLens optics is much more complex, as it relies on specific properties of light propagation. This book describes their operation only in general terms.)

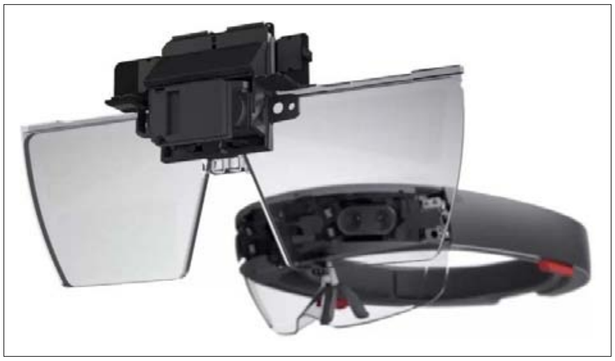


FIGURE 1-5 The see-through display of the Microsoft HoloLens. (Courtesy of https://developer.microsoft.com.)

The HoloLens provides an automatically adjustable element to optimize the creation of the stereo images based on the user's interpupillary distance (IPD). As for their clarity, these stereo images contain more than 2 million points and more than 2,500 points per radian. Some critique of HoloLens is related to its limited field of view (FOV). This, however, is more of a technical challenge than a conceptual one—meaning the FOV will likely be significantly increased in the near future.

The HoloLens is equipped with a custom processor to control the optical system. This processor, called the Microsoft Holographic Processing Unit (HPU), is a multiprocessor unit that processes information from various HoloLens sensors. (See Figure 1-6.) These include the following:

- **Four cameras** These help the HoloLens ascertain the near environment.
- **Depth or time-of-flight camera** This is used to determine the distance of various real objects in the environment from the HoloLens.
- **Inertial measurement unit (IMU)** This tracks head movements.
- **Ambient light sensor** This measures the intensity of ambient light for adjusting the brightness of holograms.

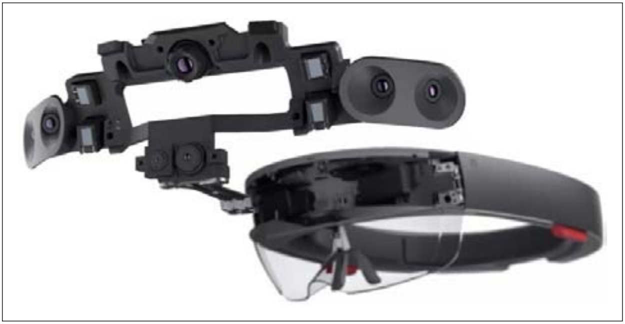


FIGURE 1-6 Sensors on the HoloLens. (Courtesy of https://developer.microsoft.com.)

Note The next generation of the HPU will also contain an integrated coprocessor to handle computations for implementing deep neural networks to support artificial intelligence. (For more information, see http://bit.ly/HPU_2.)

In addition to the HPU, HoloLens also has a 32-bit Intel central processing unit (CPU), which—along with 2 GB of RAM and 64 GB of flash memory—serves the hardware basis for the Windows 10 operating system. HoloLens also has built-in speakers, which generate spatial sound. All this hardware is powered by a battery, which lasts for about 2 to 3 hours of active use.

Immersive Headsets

Unlike the HoloLens, currently available immersive headsets are tethered, so they require a PC running Windows 10 with the Fall Creators update or above. Detailed requirements for PC hardware are given here: http://bit.ly/wmr_pc.

Like the HoloLens, however, immersive headsets use stereoscopy to create 3D objects. Therefore, one of the most important specifications is the display technology. Table 1-1 lists currently available immersive devices along with their key specifications. As shown, apart from the Samsung HMD Odyssey, they're all quite similar. Each one uses two liquid crystal displays (LCDs), each with a resolution of 1,440 x 1,440 and a field of view of 105°. In contrast, the Samsung HMD Odyssey uses an AMOLED display with the same horizontal resolution as the two LCDs in the other devices (2,880) and a higher vertical resolution (1600). Additionally, the Samsung headset provides a slightly larger field of view (110°). Figures 1-7 to 1-11 depict each of these headsets.

TABLE 1-1 A Summary of Immersive Mixed Reality Headsets

Device name	Display technology	Field of view
Acer Windows Mixed Reality headset	Two LCD displays at 1440 x 1440 points each	105º
Dell Visor Windows Mixed Reality headset	Two LCD displays at 1440 x 1440 points each	105º
HP Windows Mixed Reality headset	Two LCD displays at 1440 x 1440 points each	105º
Lenovo Explorer Windows Mixed Reality headset	Two LCD displays at 1440 x 1440 points each	105º
Samsung HMD Odyssey Windows Mixed Reality headset	AMOLED 2880 x 1600	110º

Note All immersive headsets come with the controllers shown in Figure 1-3. These controllers are not yet compatible with HoloLens, however. Each immersive headset also uses an HDMI cable for display purposes and USB for connectivity.

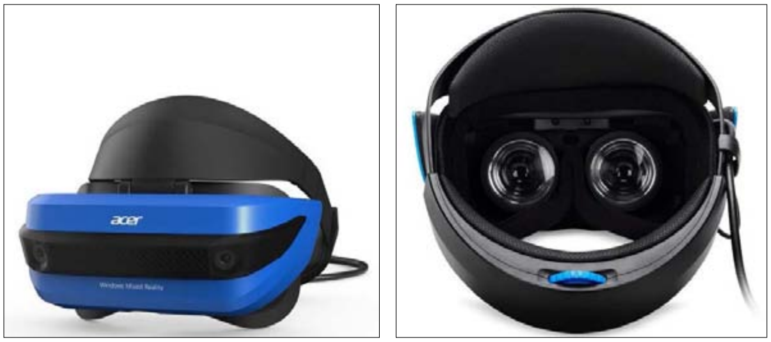


FIGURE 1-7 The Acer Windows Mixed Reality Headset. (Courtesy of https://www.microsoft.com/.)

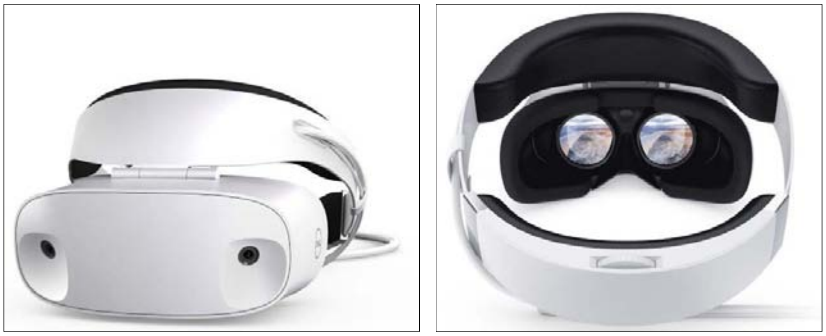


FIGURE 1-8 The Dell Visor Windows Mixed Reality headset. (Courtesy of https://www.microsoft.com/.)

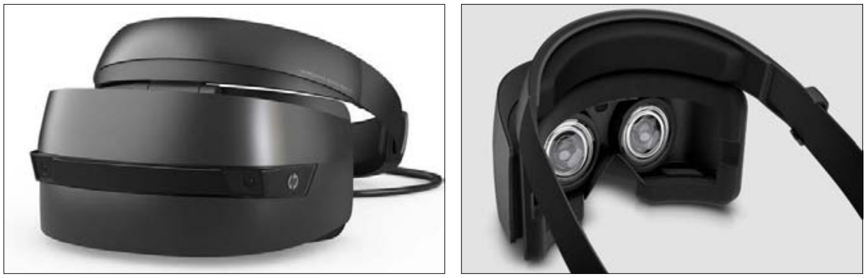


FIGURE 1-9 The HP Windows Mixed Reality headset. (Courtesy of https://www.microsoft.com/.)

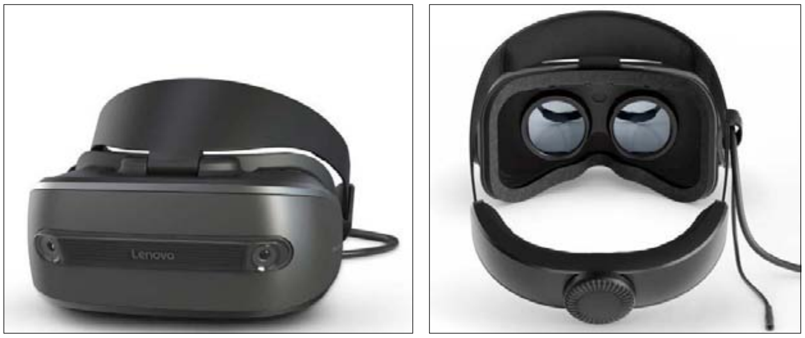


FIGURE 1-10 The Lenovo Windows Mixed Reality headset. (Courtesy of https://www.microsoft.com/.)

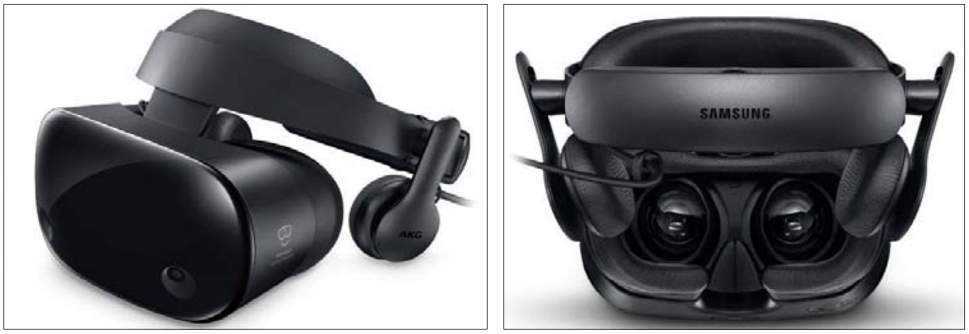


FIGURE 1-11 The Samsung HMD Odyssey Windows Mixed Reality headset.
(Courtesy of https://www.microsoft.com/.)

Placing Windows Mixed Reality Hardware on the Virtuality Continuum

Now that you've explored the available hardware for Windows Mixed Reality, you can place it on the virtuality continuum. (Refer to Figure 1-1.) Given the current capabilities of the HoloLens, you can place this device at the RE position on the continuum and moving across the spectrum to the midpoint. This covers a broad range of the virtuality continuum and defines a blending of real and virtual environments. (See Figure 1-12.) As for immersive headsets, their range on the continuum spans from the VE position to roughly the one-third point. Mobile devices cover only a small range of the virtuality continuum on both ends, as they offer a limited VR and AR experience.

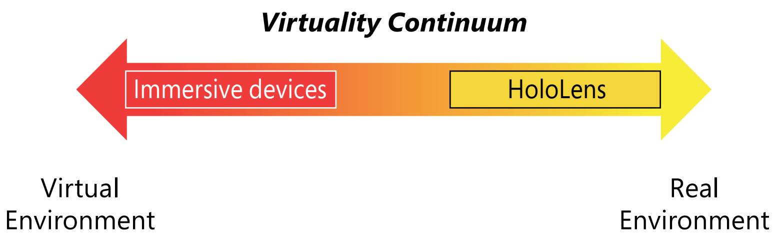
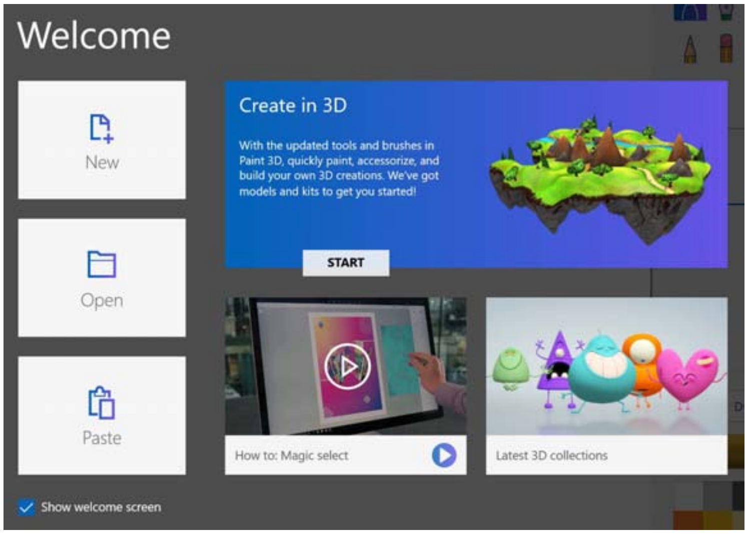


FIGURE 1-12 Immersive devices and the HoloLens on the virtuality continuum.

Mixed Reality in Windows 10

Windows 10 has several built-in apps that support Mixed Reality. One of the most exciting of these is Paint 3D. (See Figure 1-13.) Paint 3D complements the 2D functionality of Microsoft Paint, one of the best-known Windows apps.



With the updated tools and brushes in Paint 3D, quickly paint, accessorize, and build your own 3D creations. We've got models and kits to get you started!

FIGURE 1-13 Paint 3D in Windows 10.

With Paint 3D, you can quickly create 3D scenes using primitive 3D models like cubes, spheres, hemispheres, cones, and pyramids. You can also draw your own primitives using Doodle—a free drawing tool that turns your sketches into 3D objects. Finally, you can import 3D models of various scenes, people, or animals from the Remix 3D online repository. To import a 3D model into Paint 3D, follow these steps:

1. Click the **Remix 3D** tab in Paint 3D.
2. A collection of 3D models opens. Scroll this collection or type a description in the search box to find a model you like. (In Figure 1-14, I searched for a 3D model of a firefighter.)
3. Click the model you want to use. Then place it on the Paint 3D canvas by clicking the selected model.
4. To preview the scene containing the model, click the **View in Mixed Reality** icon above the Paint 3D canvas. (It's the one that features a rotated 3D cube in front of a screen.) This opens the Mixed Reality Viewer app, which overlays your scene on the real environment (captured by your device's camera).

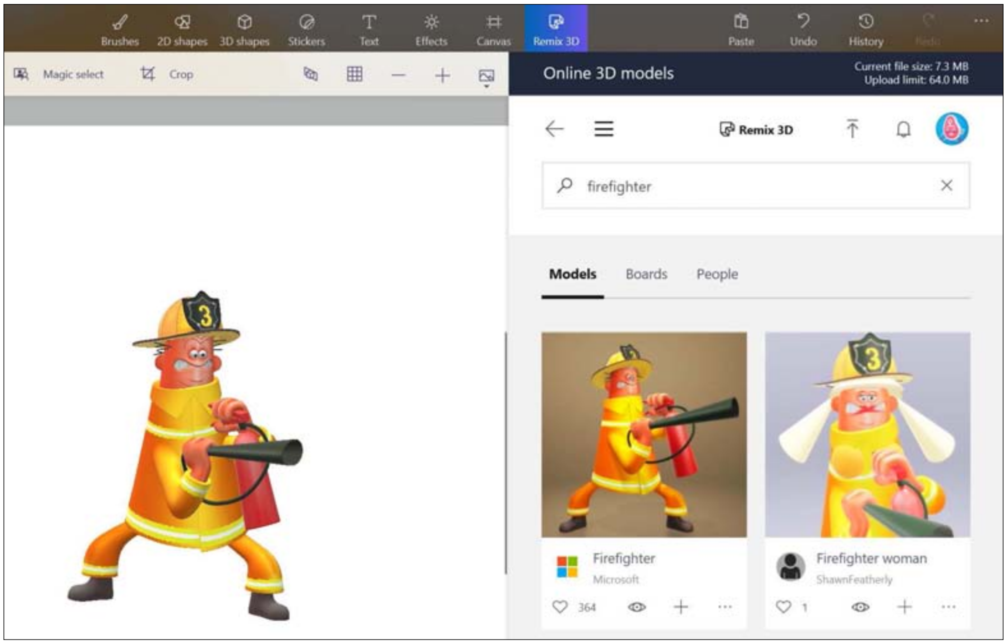


FIGURE 1-14 Importing 3D models in Paint 3D.

In addition to the Paint 3D (and Mixed Reality Viewer) app, several other Microsoft products are equipped with Mixed Reality features. These include the following:

- **Microsoft Word** Microsoft Word also enables you to import 3D models from Remix 3D.
- **3D Builder** With 3D Builder, you can create 3D scenes and then print them in 3D. You can also use 3D Builder to import any image and turn it into a 3D model. Figure 1-15 shows this capability in action. It contains a hexagon and a sphere, which I drew using primitives I accessed from the 3D Builder toolbar. Then, I added a photograph of the Samsung HMD Odyssey Windows Mixed Reality headset. (Refer to Figure 1-11.) With just a few mouse clicks, I created a 3D scene!

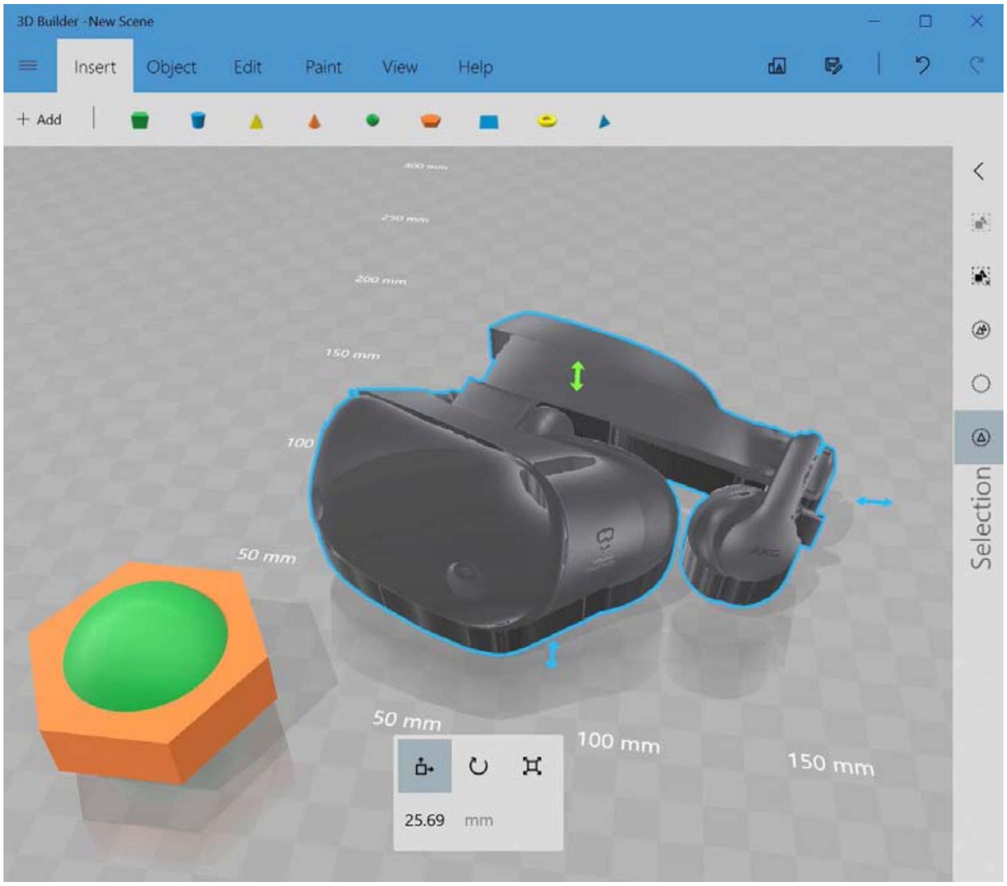


FIGURE 1-15 A simple 3D scene created with 3D Builder.

Note You can use a 3D modeling app (such as Paint 3D, 3D Builder, or a more advanced tool) to quickly create models or scenes. Then, you can use 3D Builder to export them to a file. Finally, you can import the model or scene contained in that file into the app you are developing to quickly create a virtual environment. Part III of this book explores these capabilities in a more detail.

Summary

The aim of this chapter was to further engage you in Windows Mixed Reality app development. It started by defining important terms relating to Windows Mixed Reality. Then it reviewed the concept of the virtual continuum, which describes the blending of real and mixed environments. After that, the chapter discussed where VR, AR, and MR exist within the continuum. You then learned about currently available hardware, including the HoloLens and immersive devices. Finally, you got an overview of Windows 10 apps that can support 3D modeling for Windows Mixed Reality apps.

At present, the Microsoft HoloLens is the leading Mixed Reality headset, being the only device capable of providing a true MR experience. Immersive headsets are oriented more toward VR and less toward blending with physical world than the HoloLens, as they are not equipped with see-through stereoscopy displays. However, all apps for Mixed Reality headsets can be implemented in a very similar way.

Chapter 2 discusses the development tools and models you can use to create apps for Windows Mixed Reality.

CHAPTER 2

Development Tools

Now that you're familiar with the concepts behind mixed reality and the various Microsoft Mixed Reality hardware components, let's review the development tools. Because Windows Mixed Reality headsets run Windows 10—either directly (as with the HoloLens) or indirectly through an associated PC (as with immersive headsets)—they are also a part of the Universal Windows Platform (UWP). UWP delivers a common API. Therefore, the easiest and most straightforward way to start developing apps for Windows Mixed Reality is through the UWP API.

UWP apps, however, are usually 2D—meaning that although you can view the 2D digital content they generate using a HoloLens or immersive device, the apps cannot achieve the full potential of Windows Mixed Reality. To achieve this full potential, you must instead develop 3D apps. This requires the use of Microsoft DirectX, SharpDX, Unity (supported by the Mixed Reality Toolkit and Vuforia), or UrhoSharp. This chapter summarizes and reviews all these tools.

Universal Windows Platform

The Universal Windows Platform (UWP) was introduced along with Windows 10. As noted, it provides a common API that you can use to develop apps with a consistent set of programming tools. You can also use UWP to distribute apps via the Microsoft Store.

UWP apps can target various device families, including Mixed Reality, Xbox, mobile, Internet of Things (IoT), desktop, and Surface Hub. (See Figure 2-1.) UWP provides a common core API that applies for each device family as well as access to tools to support device family–specific hardware functionality. To access these tools, you simply obtain an SDK extension dedicated to a specific device family. Then, to use the extension, you simply select it in the Reference Manager. An app that uses a device-specific API can still run across all other Windows 10 device families. However, an attempt to access the device-specific API on a device that does not support it will result in an exception of type `System.TypeLoadException`.

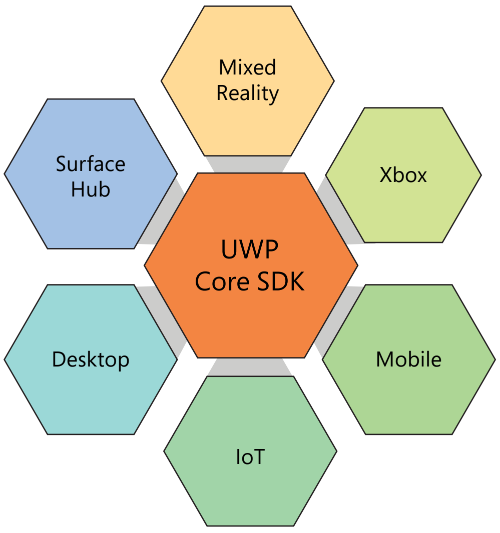


FIGURE 2-1 A sketch of the UWP API.

Adaptive Coding and Target Device Families

Both Windows 10 and UWP are still evolving, with new features frequently added to both. As a result, there may be times when an API you want to use might not be available for a particular device. To avoid this problem, you can use adaptive coding. Alternatively, you can make your UWP app non-universal by indicating the device families and minimum Windows 10 versions supported by your app.

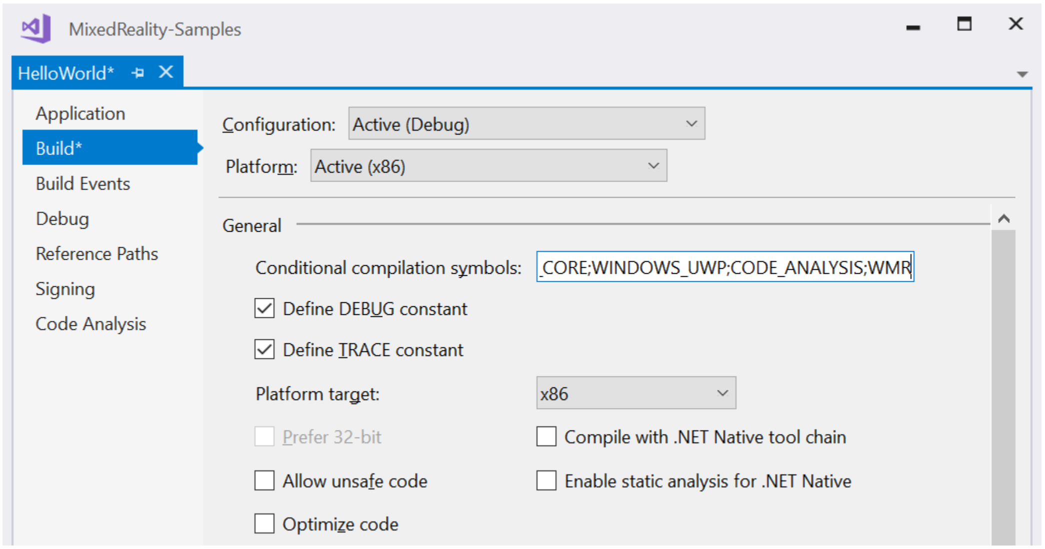
In an adaptive coding, you use the methods of the `Windows.Foundation.Metadata.ApiInformation` class to infer whether a type (`IsTypePresent`), a method (`IsMethodPresent`), an event (`IsEventPresent`), a property (`IsPropertyPresent`), an enumeration value (`IsEnumNamedValuePresent`), or an API contract (`IsApiContractPresent`) is available for the device on which an app is running. For instance, the following statement checks whether the `Windows.ApplicationModel.Preview.Holographic.HolographicApplicationPreview` class can be used:

```
var isTypePresent = Windows.Foundation.Metadata.ApiInformation.IsTypePresent(
    "Windows.ApplicationModel.Preview.Holographic.HolographicApplicationPreview");
```

Adaptive coding is a little like the conditional preprocessor directives (`#if`, `#else`, `#elif`, and `#endif`), which you use to compile the same code for various platforms. With these, you typically enable or disable specific code snippets depending on the compilation constants representing the particular platform. For instance, you could write something like the following:

```
public sealed partial class MainPage : Page
{
    public MainPage()
    {
        InitializeComponent();
#if WMR
        if (Windows.ApplicationModel.Preview.Holographic.HolographicApplicationPreview.
            IsCurrentViewPresentedOnHolographicDisplay())
        {
            // Do something with holographic preview
        }
#endif
    }
}
```

In this case, the code between the `#if` and `#endif` clauses will be compiled only when the WMR symbol is defined. You can define symbols using the #define directive (for example, `#define WMR`), provided it is the first clause in the file, or by entering them in the Conditional Compilation Symbols box in the project properties. (See Figure 2-2.)



MixedReality-Samples

HelloWorld*

Application
Build*
Build Events
Debug
Reference Paths
Signing
Code Analysis

Configuration: Active (Debug)
Platform: Active (x86)

General

Conditional compilation symbols: _CORE;WINDOWS_UWP;CODE_ANALYSIS;WMR

[✓] Define DEBUG constant
[✓] Define TRACE constant

Platform target: x86

[ ] Prefer 32-bit
[ ] Compile with .NET Native tool chain
[ ] Allow unsafe code
[ ] Enable static analysis for .NET Native
[ ] Optimize code

FIGURE 2-2 Setting conditional compilation symbols.

Although preprocessor directives are widely used for developing multi-platform apps, they require you to recompile the code for each platform independently by manually changing conditional compilation symbols. In contrast, adaptive coding allows you to check API availability during runtime. As a result, you compile the source code only once for all platforms and then poll for the particular API, when the app is being executed.

To set target device families for your app, you edit the app package manifest (Package.appxmanifest). This XML file, contained in every UWP project, describes your app to the operating system. To edit this file, follow these steps:

1. Open the package-manifest file in the text or XML Editor. To do this quickly in Visual Studio, right-click the **Package.appxmanifest** file in Solution Explorer and choose **View Code** from the menu that appears.

2. In the **<Dependencies>** node, declare which device families your app should support. For example, to make your UWP app compatible only with the HoloLens, you would modify the node as follows:

```
<Dependencies>
    <TargetDeviceFamily Name="Windows.Holographic" MinVersion="10.0.0.0"
                        MaxVersionTested="10.0.0.0"/>
</Dependencies>
```

Each target device family appears within a `TargetDeviceFamily` tag. This tag has three attributes:

- **`Name`** This attribute references the name of the device family that your app is targeting. (For a complete list of device family names, see http://bit.ly/target_device_family.)
- **`MinVersion`** This attribute references the minimum version number of the device family with which your app is compatible.
- **`MaxVersionTested`** This attribute references the maximum version number of the device family against which your app was tested.

Being able to implement universal 2D apps is very important when you want to apply new or existing UWP apps to Windows Mixed Reality. In such cases, you just need to use adaptive coding to ensure that the needed API is accessible. Later, you can tailor the views of your apps to HoloLens or Windows Mixed Reality headsets. You'll learn how in Part II of this book. For now, I simply want to demonstrate how the same UWP app looks on various Windows 10 devices without making a single change to the source code (again, assuming the code contains the necessary APIs). Figure 2-3 shows a simple HelloWorld UWP app (which you'll explore further in subsequent chapters) in the Windows Mobile emulator. Figure 2-4 shows the same app on a Windows desktop. Finally, Figure 2-5 shows the app in a HoloLens emulator. As you can see, the app automatically adjusts to the device.

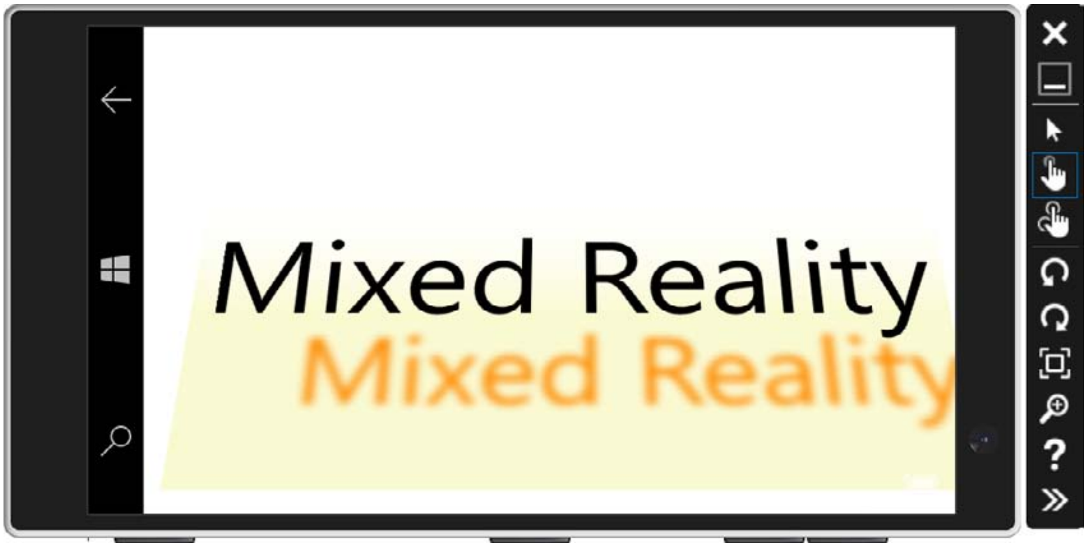


FIGURE 2-3 The HelloWorld UWP app executed in the Windows Mobile emulator.

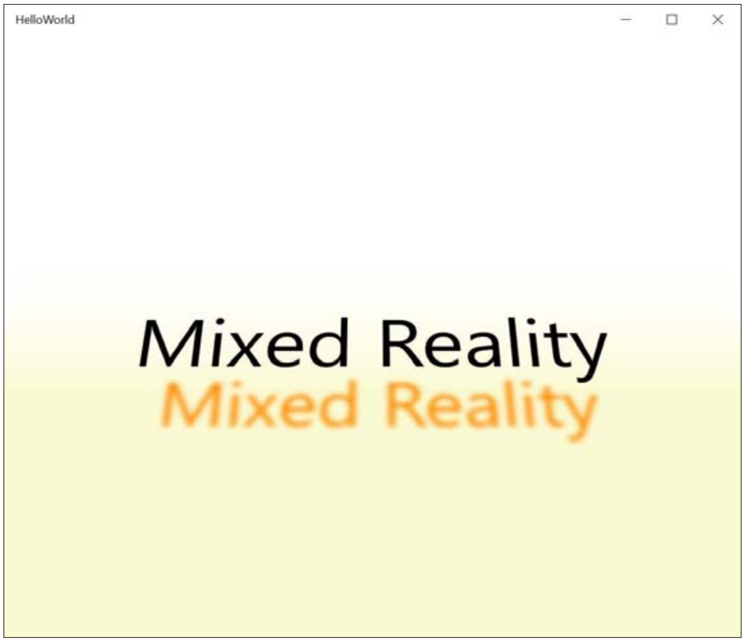


FIGURE 2-4 The HelloWorld UWP app executed in a Windows desktop environment.

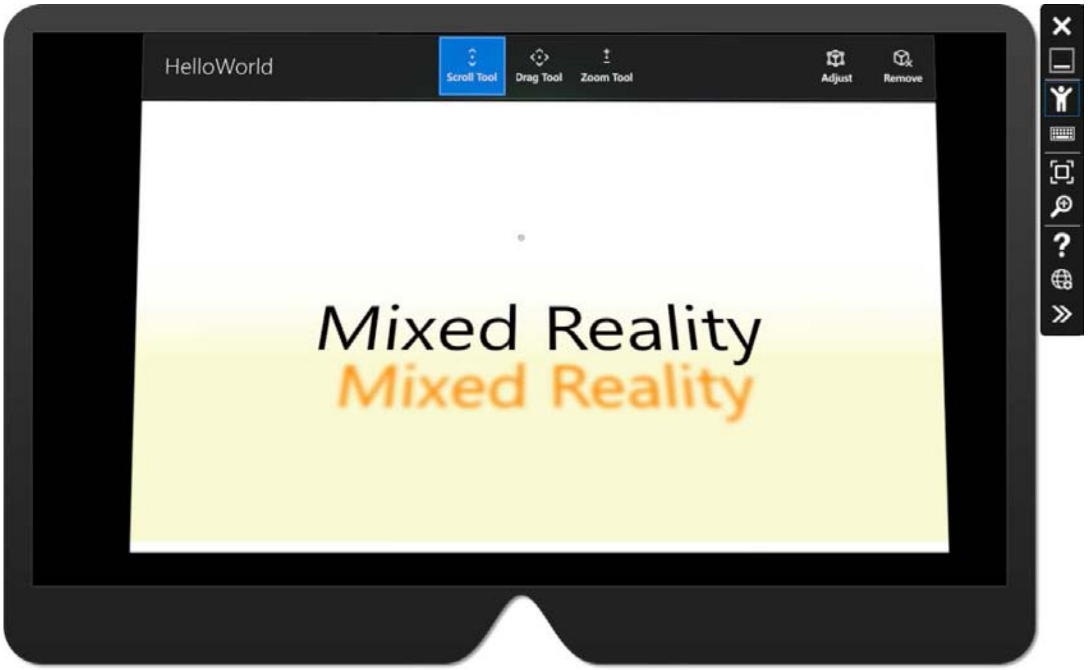


FIGURE 2-5 The HelloWorld UWP app executed in the HoloLens emulator.

.NET Core and .NET Native

You can use a variety of programming languages to develop UWP apps. This book, however, focuses on C#. UWP apps built with C# use the Microsoft .NET Core Framework, which is a general-purpose and modular implementation of the Microsoft .NET Framework. In C# UWP apps, you access the Microsoft .NET Core Framework through the `Microsoft.NETCore.UniversalWindowsPlatform` NuGet package. This package is referenced by default whenever you create a UWP project.

The .NET Core Framework contains two tools to manage program execution—one for debug build configurations and one for release build configurations. (See Figure 2-6.)

- **Core Common Language Runtime (CoreCLR)** Core CLR is a .NET Core-tailored version of the CLR for the .NET Framework. It compiles source code by first converting it to intermediate language (IL) code. Then, before a particular piece of this code is executed for the first time, Core CLR uses a just-in-time (JIT) compiler to transform the IL code into the native code for a given device. At this point, the actual compilation happens. (The JIT compiler, as well as additional tools for memory management, exception handling, and garbage collection, are provided by the CoreCLR.) CoreCLR is used by default for debug build configurations.
- **.NET Native** .NET Native uses an ahead-of-time (AOT) compiler to automatically compile your app's code into the native code for a specific device. The native code is then executed by the Minimal CLR runtime (MRT.dll), which is similar to the C runtime (CRT.dll) used to execute C/C++ apps. Because .NET Native uses similar tools as C/C++ compilers, it offers clear performance benefits. .NET Native is used by default in release build configurations.

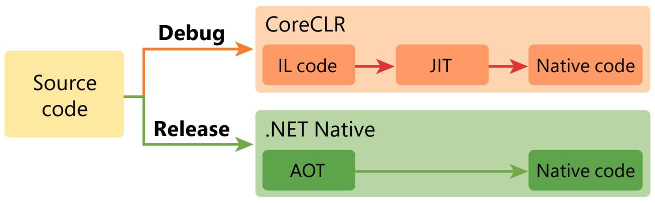


FIGURE 2-6 A sketch of UWP app compilation. Notice that the compilation mode differs depending on whether the compilation configuration is set to a debug build or a release build.

Microsoft DirectX

Microsoft DirectX is a set of APIs for creating high-performance 2D and 3D games or multimedia apps. The C++ UWP toolkit for Visual Studio 2017 includes several project templates for developing universal apps with DirectX. Like C# templates, these project templates enable you to create an app that can be deployed across various Windows 10 devices.

One template extends the capabilities of the default C++ DirectX project templates and is tailored specifically for Windows Mixed Reality: the Holographic DirectX 11 App (Visual C++) template. The default app created by this template features a holographic spinning cube positioned 2 meters along the gaze direction (that is, the direction the user is looking). This default app consists of the following elements. (See the companion code in the Chapter_02/HolographicAppDirectXMain folder.)

- **An entry point** This is the `main` method. It creates an instance of the `AppViewSource` class (discussed in an upcoming bullet).
- **Holographic content processor** The class name of this processor depends on the project name. (Here, I will call it `HolographicAppDirectXMain`.)
- **Shaders** These elements transform vertices. (See Chapter 8 for more information.)
- **`AppViewSource`** This class runs the UWP app using a static `Run` method from the `CoreApplication` class. It has one method, `CreateView`, which creates the `AppView` class. (See the next bullet.)
- **`AppView`** This manages the app window and handles the app lifecycle. It is similar to the App class from the UWP project template. The `AppView` class has several methods. Two are of particular importance for creating and rendering holographic content:
  - **`Initialize`** This method creates an instance of the holographic content processor.
  - **`Run`** This method continuously updates and renders holographic content with the help of methods from the `HolographicAppDirectXMain` class (discussed in the next bullet).
- **`HolographicAppDirectX`** This class manages and updates holographic content. `HolographicAppDirectX` uses two objects:
  - **`SpinningCubeRenderer`** This is a sample implementation of the rendering pipeline (defined in Chapter 8). Here, it renders the spinning cube.
  - **`SpatialInputHandler`** This is the gesture handler.

Figure 2-7 shows this app running in the HoloLens emulator.

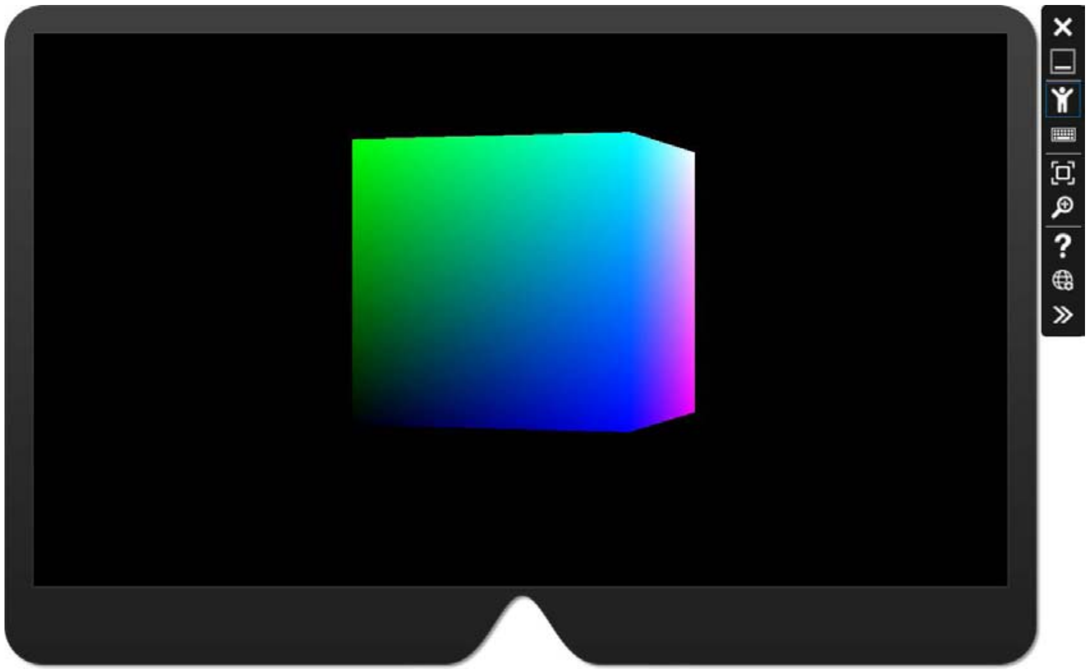


FIGURE 2-7 A HoloLens app created with the Holographic DirectX 11 App (Visual C++) template.

Note The holographic template allows access to various Windows Mixed Reality features and enables you to fully control the rendering pipeline. This comes at the cost of increased app complexity. For instance, a quick glance at the source code of the `SpinningCubeRenderer` class reveals that it handles a lot of low-level stuff to create the spinning cube, which is then programmatically added to the scene.

SharpDX

SharpDX is a .NET wrapper for the DirectX API. DirectX holographic templates include a C# project template that uses SharpDX to implement the same holographic app as the C++ DirectX template discussed in this section. The SharpDX template has the same structure as the DirectX template. The only difference is that everything is implemented with C# instead of C++. (See the companion code in the *Chapter_02/HolographicAppSharpDX* folder.)