THE EXPERT'S VOICE® IN MICROSOFT

Beginning Kinect Programming with the Microsoft Kinect SDK

CREATE COOL KINECT APPLICATIONS USING THE MICROSOFT KINECT SDK

Jarrett Webb and James Ashley

Apress®

Beginning Kinect Programming with the Microsoft Kinect SDK

Jarrett Webb
James Ashley

Apress®

Beginning Kinect Programming with the Microsoft Kinect SDK

ISBN 978-1-4302-4104-1

ISBN 978-1-4302-4105-8 (eBook)

President and Publisher: Paul Manning
Lead Editor: Jonathan Gennick
Technical Reviewer: Steven Dawson, Alastair Aitchison
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hasselll, Robert Hutchinson, Michelle Lowman, James Markham, Matthew Moodie, Jeff Olson, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Dominic Shakeshaft, Gwenan Spearing, Matt Wade, Tom Welsh
Coordinating Editor: Annie Beck, Brent Dubi
Copy Editor: Jill Steinberg
Compositor: Bytheway Publishing Services
Indexer: SPI Global
Artist: SPI Global
Cover Designer: Anna Ishchenko

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary materials referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

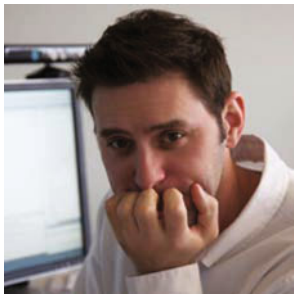
We dedicate this book to our families, Meredith, Tamara, Sasha, Paul and Sophia, who supported us, stood by us, cheered us on and were exceedingly patient with us through this process.

—Jarrett and James

Contents at a Glance

Contents

About the Authors



■ Jarrett Webb creates imaginative, dynamic, interactive, immersive, experiences using multi-touch and the Kinect. He lives in Austin, Texas.

■ James Ashley has been developing primarily Microsoft software for nearly 15 years. He maintains a blog at www.imaginativeuniversal.com. He helps run the Atlanta XAML user group and for the past two years has led the organizing of the reMIX conference in Atlanta. He is currently employed as a Presentation Layer Architect in the Emerging Experiences Group at Razorfish where he is encouraged to play with expensive technology and develop impossible applications. It is the sort of job he always knew he wanted but didn't realize actually existed.

James lives in Atlanta, Georgia with his wife, Tamara, and three children: Sophia, Paul and Sasha. You can contact him by email at jamesashley@imaginativeuniversal.com or contact him on twitter at @jamesashley.

About the Technical Reviewer

Steve Dawson is the Technology Director of the Emerging Experiences group at Razorfish. He collaborates with a cross-functional team of strategists and designers to bring innovative and engaging experiences to life using the latest technologies.

As the technical director of the Emerging Experiences group, Steve led the technology effort to launch the first Microsoft Surface solution worldwide and has deployed more than 4,000 experiences in the field for a variety of clients utilizing emerging technology.

In addition to supporting client solutions, Steve is responsible for R&D efforts within Razorfish related to emerging technologies – surface computing, augmented reality, ubiquitous computing, computer vision and gestural interface development using Kinect for Windows. His recent work with the Kinect platform has been recognized and praised by a variety of media outlets including Wired, FastCompany, Mashable and Engadget. Steve is active on the conference circuit, most recently speaking at E3 and SXSW.

Steve has been with Razorfish for 11 years and has had the pleasure of working with a variety of clients, some of which include Microsoft, AT&T, Dell, Audi, Delta, Kraft, UPS and Coca-Cola.

Acknowledgments

Many people have provided assistance and inspiration as we wrote this book. We would first like to thank our current and prior colleagues at Razorfish: Steve Dawson, Luke Hamilton, Alex Nichols, Dung Tien Le and Jimmy Moore for freely sharing their ideas and even, at times, their code with us. Being surrounded by knowledgeable and clever people has helped us to make this book better. We would also like to thank our employers at Razorfish, in particular Jonathan Hull, for creating an environment where we can explore new design concepts and bleeding edge technology in order to build amazing experiences.

We are indebted to Microsoft's Kinect for Windows team for providing us access to internal builds as well as assistance in understanding how the Kinect SDK works, especially: Rob Relyea, Sheridan Jones, Bob Heddle, JP Wollersheim and Mauro Giusti.

We would also like to thank the hackers, the academics, the artists and the madmen who learned to program for the Kinect sensor long before there was a Kinect for Windows SDK and subsequently filled the internet with video after inspiring video showing the versatility and ingenuity of the Kinect hardware. We were able to write this book because they lit the way.

–Jarrett and James

Introduction

It is customary to preface a work with an explanation of the author's aim, why he wrote the book, and the relationship in which he believes it to stand to other earlier or contemporary treatises on the same subject. In the case of a technical work, however, such an explanation seems not only superfluous but, in view of the nature of the subject-matter, even inappropriate and misleading. In this sense, a technical book is similar to a book about anatomy. We are quite sure that we do not as yet possess the subject-matter itself, the content of the science, simply by reading around it, but must in addition exert ourselves to know the particulars by examining real cadavers and by performing real experiments. Technical knowledge requires a similar exertion in order to achieve any level of competence.

Besides the reader's desire to be hands-on rather than heads-down, a book about Kinect development offers some additional challenges due to its novelty. The Kinect seemed to arrive *exnihilo* in November of 2010 and attempts to interface with the Kinect technology, originally intended only to be used with the XBOX gaming system, began almost immediately. The popularity of these efforts to *hack* the Kinect appears to have taken even Microsoft unawares.

Several frameworks for interpreting the raw feeds from the Kinect sensor have been released prior to Microsoft's official reveal of the Kinect SDK in July of 2011 including libfreenect developed by the OpenKinect community and OpenNI developed primarily by PrimeSense, vendors of one of the key technologies used in the Kinect sensor. The surprising nature of the Kinect's release as well as Microsoft's apparent failure to anticipate the overwhelming desire on the part of developers, hobbyists and even research scientists to play with the technology may give the impression that the Kinect SDK is hodgepodge or even a briefly flickering fad.

The gesture recognition capabilities made affordable by the Kinect, however, have been researched at least since the late 70's. A brief search on YouTube for the phrase "put that there" will bring up Chris Schmandt's1979 work with the MIT Media Lab demonstrating key Kinect concepts such as gesture tracking and speech recognition. The influence of Schmandt's work can be seen in Mark Lucente's work with gesture and speech recognition in the 90's for IBM Research on a project called DreamSpace. These early concepts came together in the central image from Steven Speilberg's 2002 film *Minority Report*that captured viewers imaginations concerning what the future should look like. That image was of Tom Cruise waving his arms and manipulating his computer screens without touching either the monitors or any input devices. In the middle of an otherwise dystopic society filled with robotic spiders, ubiquitous marketing and *panopticon* police surveilence, Steven Speilberg offered us a vision not only of a possible technological future but of a future we wanted.

Although *Minority Report* was intended as a vision of technology 50 years in the future, the first concept videos for the Kinect, code-named Project Natal, started appearing only seven years after the movie's release. One of the first things people noticed about the technology with respect to its cinematic predecessor was that the Kinect did not require Tom Cruise's three-fingered, blue-lit gloves to function. We had not only caught up to the future as envisioned by*Minority Report* in record time but had even surpassed it.

The Kinect is only new in the sense that it has recently become affordable and fit for mass-production. As pointed out above, it has been anticipated in research circles for over 40 years. The

principle concepts of gesture-recognition have not changed substantially in that time. Moreover, the cinematic exploration of gesture-recognition devices demonstrates that the technology has succeeded in making a deep connection with people's imaginations, filling a need we did not know we had.

In the near future, readers can expect to see Kinect sensors built into monitors and laptops as gesture-based interfaces gain ground in the marketplace. Over the next few years, Kinect-like technology will begin appearing in retail stores, public buildings, malls and multiple locations in the home. As the hardware improves and becomes ubiquitous, the authors anticipate that the Kinect SDK will become the leading software platform for working with it. Although slow out of the gate with the Kinect SDK, Microsoft's expertise in platform development, the fact that they own the technology, as well as their intimate experience with the Kinect for game development affords them remarkable advantages over the alternatives. While predictions about the future of technology have been shown, over the past few years, to be a treacherous endeavor, the authors posit with some confidence that skills gained in developing with the Kinect SDK will not become obsolete in the near future.

Even more important, however, developing with the Kinect SDK is fun in a way that typical development is not. The pleasure of building your first skeleton tracking program is difficult to describe. It is in order to share this ineffable experience -- an experience familiar to anyone who still remembers their first software program and became software developers in the belief thissense of joy and accomplishment was repeatable – that we have written this book.

About This Book

This book is for the inveterate tinkerer who cannot resist playing with code samples before reading the instructions on why the samples are written the way they are. After all, you bought this book in order to find out how to play with the Kinect sensor and replicate some of the exciting scenarios you may have seen online. We understand if you do not want to initially wade through detailed explanations before seeing how far you can get with the samples on your own.At the same time, we have included in depth information about why the Kinect SDK works the way it does and to provide guidance on the tricks and pitfalls of working with the SDK. You can always go back and read this information at a later point as it becomes important to you.

The chapters are provided in roughly sequential order, with each chapter building upon the chapters that went before. They begin with the basics, move on to image processing and skeleton tracking, then address more sophisticated scenarios involving complex gestures and speech recognition. Finally they demonstrate how to combine the SDK with other code libraries in order to build complex effects. The appendix offers an overview of mathematical and kinematic concepts that you will want to become familiar with as you plan out your own unique Kinect applications.

Chapter Overview

Chapter 1: Getting Started
Your imagination is running wild with ideas and cool designs for applications. There are a few things to know first, however. This chapter will cover the surprisingly long history that led up to the creation of the Kinect for Windows SDK. It will then provide step-by-step instructions for downloading and installing the necessary libraries and tools needed to developapplications for the Kinect.

Chapter 2: Application Fundamentals guides the reader through the process of building a Kinect application. At the completion of this chapter, the reader will have the foundation needed to write

relatively sophisticated Kinect applications using the Microsoft SDK. Thisincludes getting data from the Kinect to display a live image feed as well as a few tricksto manipulate the image stream. The basic code introduced here is common to virtually all Kinect applications.

Chapter 3: Depth Image Processing
The depth stream is at the core of Kinect technology. This code intensive chapter explains the depth stream in detail: what data the Kinect sensor provides and what can be done with this data. Examples include creating images where users are identified and their silhouettes are colored as well as simple tricks using the silhouettes to determinine the distance of the user from the Kinect and other users.

Chapter 4: Skeleton Tracking
By using the data from the depth stream, the Microsoft SDK can determine human shapes.
This is called skeleton tracking. The reader will learn how to get skeleton tracking data, what that data means and how to use it. At this point, you will know enough to have some fun. Walkthroughs include visually tracking skeleton joints and bones, and creating some basic games.

Chapter 5: Advanced Skeleton Tracking
There is more to skeleton tracking than just creating avatars and skeletons. Sometimes reading and processing raw Kinect data is not enough. It can be volatile and unpredictable. This chapter provides tips and tricks to smooth out this data to create more polished applications. In this chapter we will also move beyond the depth image and work with the live image. Using the data produced by the depth image and the visual of the live image, we will work with an augmented reality application.

Chapter 6: Gestures
The next level in Kinect development is processing skeleton tracking data to detect using gestures. Gestures make interacting with your application more natural. In fact, there is a whole fieldof study dedicated to natural user interfaces. This chapter will introduce NUI and show how it affects application development. Kinect is so new that well-established gesture libraries and tools are still lacking. This chapter will give guidance to help define what a gesture is and how to implement a basic gesture library.

Chapter 7: Speech
The Kinect is more than just a sensor that sees the world. It also hears it. The Kinect has an array of microphones that allows it to detect and process audio. This means that the user can use voice commands as well as gestures to interact with an application. In this chapter, you will be introduced to the Microsoft Speech Recognition SDK and shown how it is integrated with the Kinect microphone array.

Chapter 8: Beyond the Basics introduces the reader to much more complex development that can be done with the Kinect. This chapter addresses useful tools and ways to manipulate depth data to create complex applications and advanced Kinect visuals.

Appendix A: Kinect Math
Basic math skills and formulas needed when working with Kinect. Gives only practical information needed for development tasks.

What You Need to Use This Book

The Kinect SDK requires the Microsoft .NET Framework 4.0. To build applications with it, you will need either Visual Studio 2010 Express or another version of Visual Studio 2010. The Kinect SDK may be downloaded at http://www. kinectforwindows.org/download/ .

The samples in this book are written with WPF 4 and C#. The Kinect SDK merely provides a way to read and manipulate the sensor streams from the Kinect device. Additional technology is required in order to display this data in interesting ways. For this book we have selected WPF, the preeminant vector graphic platform in the Microsoft stack as well as a platform generally familiar to most developers working with Microsoft technologies. C#, in turn, is the .NET language with the greatest penetration among developers.

About the Code Samples

The code samples in this book have been written for version 1.0 of the Kinect for Windows SDK released on February 1st, 2012. You are invited to copy any of the code and use it as you will, but the authors hope you will actually improve upon it. Book code, after all, is not real code. Each project and snippet found in this book has been selected for its ability to illustrate a point rather than its efficiency in performing a task. Where possible we have attempted to provide best practices for writing performant Kinect code, but whenever good code collided with legible code, legibility tended to win.

More painful to us, given that both the authors work for a design agency, was the realization that the book you hold in your hands needed to be about Kinect code rather than about Kinect design. To this end, we have reined in our impulse to build elaborate presentation layers in favor of spare, workman-like designs.

The source code for the projects described in this book is available for download at http://www.apress.com/9781430241041. This is the official home page of the book. You can also check for errata and find related Apress titles here.

CHAPTER 1

Getting Started

In this chapter, we explain what makes Kinect special and how Microsoft got to the point of providing a Kinect for Windows SDK—something that Microsoft apparently did not envision when it released what was thought of as a new kind of "controller-free" controller for the Xbox. We take you through the steps involved in installing the Kinect for Windows SDK, plugging in your Kinect sensor, and verifying that everything is working the way it should in order to start programming for Kinect. We then navigate through the samples provided with the SDK and describe their significance in demonstrating how to program for the Kinect.

The Kinect Creation Story

The history of Kinect begins long before the device itself was conceived. Kinect has roots in decades of thinking and dreaming about user interfaces based upon gesture and voice. The hit 2002 movie *The Minority Report* added fuel to the fire with its futuristic depiction of a spatial user interface. Rivalry between competing gaming consoles brought the Kinect technology into our living rooms. It was the hacker ethic of unlocking anything intended to be sealed, however, that eventually opened up the Kinect to developers.

Pre-History

Bill Buxton has been talking over the past few years about something he calls the Long Nose of Innovation. A play on Chris Anderson's notion of the Long Tail, the Long Nose describes the decades of incubation time required to produce a "revolutionary" new technology apparently out of nowhere. The classic example is the invention and refinement of a device central to the GUI revolution: the mouse.

The first mouse prototype was built by Douglas Engelbart and Bill English, then at the Stanford Research Institute, in 1963. They even gave the device its murine name. Bill English developed the concept further when he took it to Xerox PARC in 1973. With Jack Hawley, he added the famous mouse ball to the design of the mouse. During this same time period, Telefunken in Germany was independently developing its own rollerball mouse device called the Telefunken Rollkugel. By 1982, the first commercial mouse began to find its way to the market. Logitech began selling one for $299. It was somewhere in this period that Steve Jobs visited Xerox PARC and saw the mouse working with a WIMP interface (windows, icons, menus, pointers). Some time after that, Jobs invited Bill Gates to see the mouse-based GUI interface he was working on. Apple released the Lisa in 1983 with a mouse, and then equipped the Macintosh with the mouse in 1984. Microsoft announced its Windows OS shortly after the release of the Lisa and began selling Windows 1.0 in 1985. It was not until 1995, with the release of Microsoft's Windows 95 operating system, that the mouse became ubiquitous. The Long Nose describes the 30-year span required for devices like the mouse to go from invention to ubiquity.

A similar 30-year Long Nose can be sketched out for Kinect. Starting in the late 70s, about halfway into the mouse's development trajectory, Chris Schmandt at the MIT Architecture Machine Group started a research project called Put-That-There, based on an idea by Richard Bolt, which combined voice and gesture recognition as input vectors for a graphical interface. The Put-That-There installation lived in a sixteen-foot by eleven-foot room with a large projection screen against one wall. The user sat in a vinyl chair about eight feet in front of the screen and had a magnetic cube hidden up one wrist for spatial input as well as a head-mounted microphone. With these inputs, and some rudimentary speech parsing logic around pronouns like "that" and "there," the user could create and move basic shapes around the screen. Bolt suggests in his 1980 paper describing the project, "Put-That-There: Voice and Gesture at the Graphics Interface," that eventually the head-mounted microphone should be replaced with a directional mic. Subsequent versions of Put-That-There allowed users to guide ships through the Caribbean and place colonial buildings on a map of Boston.

Another MIT Media Labs research project from 1993 by David Koonz, Kristinn Thorrison, and Carlton Sparrell—and again directed by Bolt—called The Iconic System refined the Put-That-There concept to work with speech and gesture as well as a third input modality: eye-tracking. Also, instead of projecting input onto a two-dimensional space, the graphical interface was a computer-generated three-dimensional space. In place of the magnetic cubes used for Put-That-There, the Iconic System included special gloves to facilitate gesture tracking.

Towards the late 90s, Mark Lucente developed an advanced user interface for IBM Research called DreamSpace, which ran on a variety of platforms including Windows NT. It even implemented the Put-That-There syntax of Chris Schmandt's 1979 project. Unlike any of its predecessors, however, DreamSpace did not use wands or gloves for gesture recognition. Instead, it used a vision system. Moreover, Lucente envisioned DreamSpace not only for specialized scenarios but also as a viable alternative to standard mouse and keyboard inputs for everyday computing. Lucente helped to popularize speech and gesture recognition by demonstrating DreamSpace at tradeshows between 1997 and 1999.

In 1999 John Underkoffler—also with MIT Media Labs and a coauthor with Mark Lucente on a paper a few years earlier on holography—was invited to work on a new Stephen Spielberg project called *The Minority Report*. Underkoffler eventually became the Science and Technology Advisor on the film and, with Alex McDowell, the film's Production Designer, put together the user interface Tom Cruise uses in the movie. Some of the design concepts from *The Minority Report* UI eventually ended up in another project Underkoffler worked on called G-Speak.

Perhaps Underkoffler's most fascinating design contribution to the film was a suggestion he made to Spielberg to have Cruise accidently put his virtual desktop into disarray when he turns and reaches out to shake Colin Farrell's hand. It is a scene that captures the jarring acknowledgment that even "smart" computer interfaces are ultimately still reliant on conventions and that these conventions are easily undermined by the uncanny facticity of real life.

The Minority Report was released in 2002. The film visuals immediately seeped into the collective unconscious, hanging in the zeitgeist like a promissory note. A mild discontent over the prevalence of the mouse in our daily lives began to be felt, and the press as well as popular attention began to turn toward what we came to call the Natural User Interface (NUI). Microsoft began working on its innovative multitouch platform Surface in 2003, began showing it in 2007, and eventually released it in 2008. Apple unveiled the iPhone in 2007. The iPad began selling in 2010. As each NUI technology came to market, it was accompanied by comparisons to *The Minority Report*.

The Minority Report

So much ink has been spilled about the obvious influence of *The Minority Report* on the development of Kinect that at one point I insisted to my co-author that we should try to avoid ever using the words

"minority" and "report" together on the same page. In this endeavor I have failed miserably and concede that avoiding mention of *The Minority Report* when discussing Kinect is virtually impossible.

One of the more peculiar responses to the movie was the movie critic Roger Ebert's opinion that it offered an "optimistic preview" of the future. *The Minority Report*, based loosely on a short story by Philip K. Dick, depicts a future in which police surveillance is pervasive to the point of predicting crimes before they happen and incarcerating those who have not yet committed the crimes. It includes massively pervasive marketing in which retinal scans are used in public places to target advertisements to pedestrians based on demographic data collected on them and stored in the cloud. Genetic experimentation results in monstrously carnivorous plants, robot spiders that roam the streets, a thriving black market in body parts that allows people to change their identities and—perhaps the most jarring future prediction of all—policemen wearing rocket packs.

Perhaps what Ebert responded to was the notion that the world of *The Minority Report* was a believable future, extrapolated from our world, demonstrating that through technology our world can actually change and not merely be more of the same. Even if it introduces new problems, science fiction reinforces the idea that technology can help us leave our current problems behind. In the 1958 book, *The Human Condition*, the author and philosopher Hannah Arendt characterizes the role of science fiction in society by saying, "… science has realized and affirmed what men anticipated in dreams that were neither wild nor idle … buried in the highly non-respectable literature of science fiction (to which, unfortunately, nobody yet has paid the attention it deserves as a vehicle of mass sentiments and mass desires)." While we may not all be craving rocket packs, we do all at least have the aspiration that technology will significantly change our lives.

What is peculiar about *The Minority Report* and, before that, science fiction series like the Star Trek franchise, is that they do not always merely predict the future but can even shape that future. When I first walked through automatic sliding doors at a local convenience store, I knew this was based on the sliding doors on the USS Enterprise. When I held my first flip phone in my hands, I knew it was based on Captain Kirk's communicator and, moreover, would never have been designed this way had Star Trek never aired on television.

If *The Minority Report* drove the design and adoption of the gesture recognition system on Kinect, Star Trek can be said to have driven the speech recognition capabilities of Kinect. In interviews with Microsoft employees and executives, there are repeated references to the desire to make Kinect work like the Star Trek computer or the Star Trek holodeck. There is a sense in those interviews that if the speech recognition portion of the device was not solved (and occasionally there were discussions about dropping the feature as it fell behind schedule), the Kinect sensor would not have been the future device everyone wanted.

Microsoft's Secret Project

In the gaming world, Nintendo threw down the gauntlet at the 2005 Tokyo Game Show conference with the unveiling of the Wii console. The console was accompanied by a new gaming device called the Wii Remote. Like the magnetic cubes from the original Put-That-There project, the Wii Remote can detect movement along three axes. Additionally, the remote contains an optical sensor that detects where it is pointing. It is also battery powered, eliminating long cords to the console common to other platforms.

Following the release of the Wii in 2006, Peter Moore, then head of Microsoft's Xbox division, demanded work start on a competitive Wii killer. It was also around this time that Alex Kipman, head of an incubation team inside the Xbox division, met the founders of PrimeSense at the 2006 Electronic Entertainment Expo. Microsoft created two competing teams to come up with the intended Wii killer: one working with the PrimeSense technology and the other working with technology developed by a company called 3DV. Though the original goal was to unveil something at E3 2007, neither team seemed to have anything sufficiently polished in time for the exposition. Things were thrown a bit more off track in 2007 when Peter Moore announced that he was leaving Microsoft to go work for Electronic Arts.

It is clear that by the summer of 2007 the secret work being done inside the Xbox team was gaining momentum internally at Microsoft. At the D: All Things Digital conference that year, Bill Gates was interviewed side-by-side with Steve Jobs. During that interview, in response to a question about Microsoft Surface and whether multitouch would become mainstream, Gates began talking about vision recognition as the step beyond multitouch:

> ***Gates**: Software is doing vision. And so, imagine a game machine where you just can pick up the bat and swing it or pick up the tennis racket and swing it.*
>
> ***Interviewer**: We have one of those. That's Wii.*
>
> ***Gates**: No. No. That's not it. You can't pick up your tennis racket and swing it. You can't sit there with your friends and do those natural things. That's a 3-D positional device. This is video recognition. This is a camera seeing what's going on. In a meeting, when you are on a conference, you don't know who's speaking when it's audio only … the camera will be ubiquitous … software can do vision, it can do it very, very inexpensively … and that means this stuff becomes pervasive. You don't just talk about it being in a laptop device. You talk about it being a part of the meeting room or the living room …*

Amazingly the interviewer, Walt Mossberg, cut Gates off during his fugue about the future of technology and turned the conversation back to what was most important in 2007: laptops! Nevertheless, Gates revealed in this interview that Microsoft was already thinking of the new technology being developed in the Xbox team as something more than merely a gaming device. It was already thought of as a device for the office as well.

Following Moore's departure, Don Matrick took up the reigns, guiding the Xbox team. In 2008, he revived the secret video recognition project around the PrimeSense technology. While 3DV's technology apparently never made it into the final Kinect, Microsoft bought the company in 2009 for $35 million. This was apparently done in order to defend against potential patent disputes around Kinect. Alex Kipman, a manager with Microsoft since 2001, was made General Manager of Incubation and put in charge of creating the new Project Natal device to include depth recognition, motion tracking, facial recognition, and speech recognition.

■ **Note** What's in a name? Microsoft has traditionally, if not consistently, given city names to large projects as their code names. Alex Kipman dubbed the secret Xbox project Natal, after his hometown in Brazil.

The reference device created by PrimeSense included an RGB camera, an infrared sensor, and an infrared light source. Microsoft licensed PrimeSense's reference design and PS1080 chip design, which processed depth data at 30 frames per second. Importantly, it processed depth data in an innovative way that drastically cut the price of depth recognition compared to the prevailing method at the time called "time of flight"—a technique that tracks the time it takes for a beam of light to leave and then return to the sensor. The PrimeSense solution was to project a pattern of infrared dots across the room and use the size and spacing between dots to form a 320X240 pixel depth map analyzed by the PS1080 chip. The

chip also automatically aligned the information for the RGB camera and the infrared camera, providing RGBD data to higher systems.

Microsoft added a four-piece microphone array to this basic structure, effectively providing a direction microphone for speech recognition that would be effective in a large room. Microsoft already had years of experience with speech recognition, which has been available on its operating systems since Windows XP.

Kudo Tsunada, recently hired away from Electronic Arts, was also brought on the project, leading his own incubation team, to create prototype games for the new device. He and Kipman had a deadline of August 18, 2008, to show a group of Microsoft executives what Project Natal could do. Tsunada's team came up with 70 prototypes, some of which were shown to the execs. The project got the green light and the real work began. They were given a launch date for Project Natal: Christmas of 2010.

Microsoft Research

While the hardware problem was mostly solved thanks to PrimeSense—all that remained was to give the device a smaller form factor—the software challenges seemed insurmountable. First, a responsive motion recognition system had to be created based on the RGB and Depth data streams coming from the device. Next, serious scrubbing had to be performed in order to make the audio feed workable with the underlying speech platform. The Project Natal team turned to Microsoft Research (MSR) to help solve these problems.

MSR is a multibillion dollar annual investment by Microsoft. The various MSR locations are typically dedicated to pure research in computer science and engineering rather than to trying to come up with new products for their parent. It must have seemed strange, then, when the Xbox team approached various branches of Microsoft Research to not only help them come up with a product but to do so according to the rhythms of a very short product cycle.

In late 2008, the Project Natal team contacted Jamie Shotton at the MSR office in Cambridge, England, to help with their motion-tracking problem. The motion tracking solution Kipman's team came up with had several problems. First, it relied on the player getting into an initial T-shaped pose to allow the motion capture software to discover him. Next, it would occasionally lose the player during motion, obligating the player to reinitialize the system by once again assuming the T position. Finally, the motion tracking software would only work with the particular body type it was designed for—that of Microsoft executives.

On the other hand, the depth data provided by the sensor already solved several major problems for motion tracking. The depth data allows easy filtering of any pixels that are not the player. Extraneous information such as the color and texture of the player's clothes are also filtered out by the depth camera data. What is left is basically a player blob represented in pixel positions, as shown in Figure 1-1. The depth camera data, additionally, provides information about the height and width of the player in meters.

Figure 1-1. *The Player blob*

The challenge for Shotton was to turn this outline of a person into something that could be tracked. The problem, as he saw it, was to break up the player blob provided by the depth stream into recognizable body parts. From these body parts, joints can be identified, and from these joints, a skeleton can be reconstructed. Working with Andrew Fitzgibbon and Andrew Blake, Shotton arrived at an algorithm that could distinguish 31 body parts (see Figure 1-2). Out of these parts, the version of Kinect demonstrated at E3 in 2009 could produce 48 joints (the Kinect SDK, by contrast, exposes 20 joints).

Figure 1-2. *Player parts*

To get around the initial T-pose required of the player for calibration, Shotton decided to appeal to the power of computer learning. With lots and lots of data, the image recognition software could be trained to break up the player blob into usable body parts. Teams were sent out to videotape people in their homes performing basic physical motions. Additional data was collected in a Hollywood motion capture studio of people dancing, running, and performing acrobatics. All of this video was then passed through a distributed computation engine called Dryad that had been developed by another branch of Microsoft Research in Mountain View, California, in order to begin generating a decision tree classifier that could map any given pixel of Kinect's RGBD stream onto one of the 31 body parts. This was done for 12 different body types and repeatedly tweaked to improve the decision software's ability to identify a person without an initial pose, without breaks in recognition, and for different kinds of people.

This took care of *The Minority Report* aspect of Kinect. To handle the Star Trek portion, Alex Kipman turned to Ivan Tashev of the Microsoft Research group based in Redmond. Tashev and his team had worked on the microphone array implementation on Windows Vista. Just as being able to filter out non-player pixels is a large part of the skeletal recognition solution, filtering out background noise on a microphone array situated much closer to a stereo system than it is to the speaker was the biggest part of making speech recognition work on Kinect. Using a combination of patented technologies (provided to us for free in the Kinect for Windows SDK), Tashev's team came up with innovative noise suppression and echo cancellation tricks that improved the audio processing pipeline many times over the standard that was available at the time.

Based on this audio scrubbing, a distributed computer learning program of a thousand computers spent a week building an acoustical model for Kinect based on various American regional accents and the peculiar acoustic properties of the Kinect microphone array. This model became the basis of the TellMe feature included with the Xbox as well as the Kinect for Windows Runtime Language Pack used with the Kinect for Windows SDK. Cutting things very close, the acoustical model was not completed until September 26, 2010. Shortly after, on November 4, the Kinect sensor was released.

The Race to Hack Kinect

The release of the Kinect sensor was met with mixed reviews. Gaming sites generally acknowledged that the technology was cool but felt that players would quickly grow tired of the gameplay. This did not slow down Kinect sales however. The device sold an average of 133 thousand units a day for the first 60 days after the launch, breaking the sales records for either the iPhone or the iPad and setting a new Guinness world record. It wasn't that the gaming review sites were wrong about the novelty factor of Kinect; it was just that people wanted Kinect anyways, whether they played with it every day or only for a few hours. It was a piece of the future they could have in their living rooms.

The excitement in the consumer market was matched by the excitement in the computer hacking community. The hacking story starts with Johnny Chung Lee, the man who originally hacked a Wii Remote to implement finger tracking and was later hired onto the Project Natal team to work on gesture recognition. Frustrated by the failure of internal efforts at Microsoft to publish a public driver, Lee approached AdaFruit, a vendor of open-source electronic kits, to host a contest to hack Kinect. The contest, announced on the day of the Kinect launch, was built around an interesting hardware feature of the Kinect sensor: it uses a standard USB connector to talk to the Xbox. This same USB connector can be plugged into the USB port of any PC or laptop. The first person to successfully create a driver for the device and write an application converting the data streams from the sensor into video and depth displays would win the $1,000 bounty that Lee had put up for the contest.

On the same day, Microsoft made the following statement in response to the AdaFruit contest: "Microsoft does not condone the modification of its products … With Kinect, Microsoft built in numerous hardware and software safeguards designed to reduce the chances of product tampering. Microsoft will continue to make advances in these types of safeguards and work closely with law

enforcement and product safety groups to keep Kinect tamper-resistant." Lee and AdaFruit responded by raising the bounty to $2,000.

By November 6, Joshua Blake, Seth Sandler, and Kyle Machulis and others had created the OpenKinect mailing list to help coordinate efforts around the contest. Their notion was that the driver problem was solvable but that the longevity of the Kinect hacking effort for the PC would involve sharing information and building tools around the technology. They were already looking beyond the AdaFruit contest and imagining what would come after. In a November 7 post to the list, they even proposed sharing the bounty with the OpenKinect community, if someone on the list won the contest, in order look past the money and toward what could be done with the Kinect technology. Their mailing list would go on to be the home of the Kinect hacking community for the next year.

Simultaneously on November 6, a hacker known as AlexP was able to control Kinect's motors and read its accelerometer data. The AdaFruit bounty was raised to $3,000. On Monday, November 8, AlexP posted video showing that he could pull both RGB and depth data streams from the Kinect sensor and display them. He could not collect the prize, however, because of concerns about open sourcing his code. On the 8, Microsoft also clarified its previous position in a way that appeared to allow the ongoing efforts to hack Kinect as long as it wasn't called "hacking":

> *Kinect for Xbox 360 has not been hacked—in any way—as the software and hardware that are part of Kinect for Xbox 360 have not been modified. What has happened is someone has created drivers that allow other devices to interface with the Kinect for Xbox 360. The creation of these drivers, and the use of Kinect for Xbox 360 with other devices, is unsupported. We strongly encourage customers to use Kinect for Xbox 360 with their Xbox 360 to get the best experience possible.*

On November 9, AdaFruit finally received a USB analyzer, the Beagle 480, in the mail and set to work publishing USB data dumps coming from the Kinect sensor. The OpenKinect community, calling themselves "Team Tiger," began working on this data over an IRC channel and had made significant progress by Wednesday morning before going to sleep. At the same time, however, Hector Martin, a computer science major in Bilbao, Spain, had just purchased Kinect and had begun going to through the AdaFruit data. Within a few hours he had written the driver and application to display RGB and depth video. The AdaFruit prize had been claimed in only seven days.

Martin became a contributor to the OpenKinect group and a new library, libfreenect, became the basis of the community's hacking efforts. Joshua Blake announced Martin's contribution to the OpenKinect mailing list in the following post:

> *I got ahold of Hector on IRC just after he posted the video and talked to him about this group. He said he'd be happy to join us (and in fact has already subscribed). After he sleeps to recover, we'll talk some more about integrating his work and our work.*

This is when the real fun started. Throughout November, people started to post videos on the Internet showing what they could do with Kinect. Kinect-based artistic displays, augmented reality experiences, and robotics experiments started showing up on YouTube. Sites like KinectHacks.net sprang up to track all the things people were building with Kinect. By November 20, someone had posted a video of a light saber simulator using Kinect—another movie aspiration checked off. Microsoft, meanwhile, was not idle. The company watched with excitement as hundreds of Kinect hacks made their way to the web.

On December 10, PrimeSense announced the release of its own open source drivers for Kinect along with libraries for working with the data. This provided improvements to the skeleton tracking algorithms

over what was then possible with libfreenect and projects that required integration of RGB and depth data began migrating over to the OpenNI technology stack that PrimeSense had made available. Without the key Microsoft Research technologies, however, skeleton tracking with OpenNI still required the awkward T-pose to initialize skeleton recognition.

On June 17, 2011, Microsoft finally released the Kinect SDK beta to the public under a non-commercial license after demonstrating it for several weeks at events like MIX. As promised, it included the skeleton recognition algorithms that make an initial pose unnecessary as well as the AEC technology and acoustic models required to make Kinect speech recognition system work in a large room. Every developer now had access to the same tools Microsoft used internally for developing Kinect applications for the computer.

The Kinect for Windows SDK

The Kinect for Windows SDK is the set of libraries that allows us to program applications on a variety of Microsoft development platforms using the Kinect sensor as input. With it, we can program WPF applications, WinForms applications, XNA applications and, with a little work, even browser-based applications running on the Windows operating system—though, oddly enough, we cannot create Xbox games with the Kinect for Windows SDK. Developers can use the SDK with the Xbox Kinect Sensor. In order to use Kinect's near mode capabilities, however, we require the official Kinect for Windows hardware. Additionally, the Kinect for Windows sensor is required for commercial deployments.

Understanding the Hardware

The Kinect for Windows SDK takes advantage of and is dependent upon the specialized components included in all planned versions of the Kinect device. In order to understand the capabilities of the SDK, it is important to first understand the hardware it talks to. The glossy black case for the Kinect components includes a head as well as a base, as shown in Figure 1-3. The head is 12 inches by 2.5 inches by 1.5 inches. The attachment between the base and the head is motorized. The case hides an infrared projector, two cameras, four microphones, and a fan.

Figure 1-3. *The Kinect case*

I do not recommend ever removing the Kinect case. In order to show the internal components, however, I have removed the case, as shown in Figure 1-4. On the front of Kinect, from left to right respectively when facing Kinect, you will find the sensors and light source that are used to capture RGB and depth data. To the far left is the infrared light source. Next to this is the LED ready indicator. Next is the color camera used to collect RGB data, and finally, on the right (toward the center of the Kinect head), is the infrared camera used to capture depth data. The color camera supports a maximum resolution of 1280 x 960 while the depth camera supports a maximum resolution of 640 x 480.

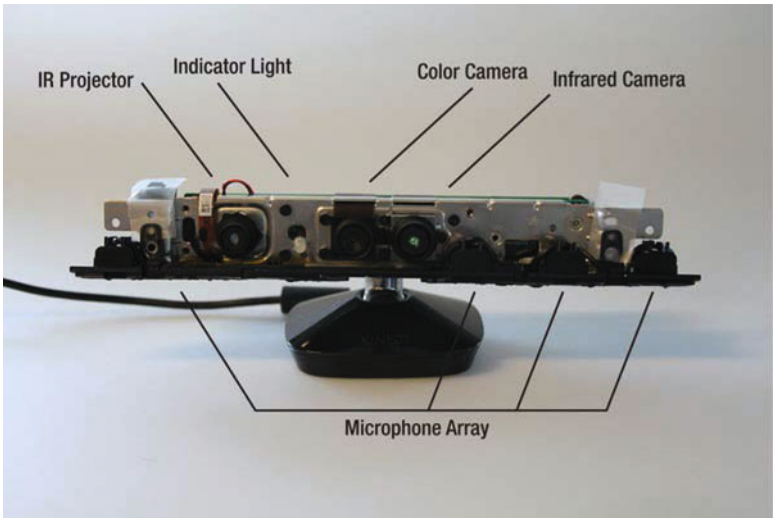


Figure 1-4. *The Kinect components*

On the underside of Kinect is the microphone array. The microphone array is composed of four different microphones. One is located to the left of the infrared light source. The other three are evenly spaced to the right of the depth camera.

If you bought a Kinect sensor without an Xbox bundle, the Kinect comes with a Y-cable, which extends the USB connector wire on Kinect as well as providing additional power to Kinect. The USB extender is required because the male connector that comes off of Kinect is not a standard USB connector. The additional power is required to run the motors on the Kinect.

If you buy a new Xbox bundled with Kinect, you will likely not have a Y-cable included with your purchase. This is because the newer Xbox consoles have a proprietary female USB connector that works with Kinect as is and does not require additional power for the Kinect servos. This is a problem—and a source of enormous confusion—if you intend to use Kinect for PC development with the Kinect SDK. You will need to purchase the Y-cable separately if you did not get it with your Kinect. It is typically marketed as a Kinect AC Adapter or Kinect Power Source. Software built using the Kinect SDK will not work without it.

A final piece of interesting Kinect hardware sold by Nyco rather than by Microsoft is called the Kinect Zoom. The base Kinect hardware performs depth recognition between 0.8 and 4 meters. The Kinect Zoom is a set of lenses that fit over Kinect, allowing the Kinect sensor to be used in rooms smaller than the standard dimensions Microsoft recommends. It is particularly appealing for users of the Kinect SDK who might want to use it for specialized functionality such as custom finger tracking logic or productivity tool implementations involving a person sitting down in front of Kinect. From

experimentation, it actually turns out to not be very good for playing games, perhaps due to the quality of the lenses.

Kinect for Windows SDK Hardware and Software Requirements

Unlike other Kinect libraries, the Kinect for Windows SDK, as its name suggests, only runs on Windows operating systems. Specifically, it runs on x86 and x64 versions of Windows 7. It has been shown to also work on early versions of Windows 8. Because Kinect was designed for Xbox hardware, it requires roughly similar hardware on a PC to run effectively.

Hardware Requirements

- Computer with a dual-core, 2.66-GHz or faster processor
- Windows 7–compatible graphics card that supports Microsoft DirectX 9.0c capabilities
- 2 GB of RAM (4 GB or RAM recommended)
- Kinect for Xbox 360 sensor
- Kinect USB power adapter

Use the free Visual Studio 2010 Express or other VS 2010 editions to program against the Kinect for Windows SDK. You will also need to have the DirectX 9.0c runtime installed. Later versions of DirectX are not backwards compatible. You will also, of course, want to download and install the latest version of the Kinect for Windows SDK. The Kinect SDK installer will install the Kinect drivers, the Microsoft Research Kinect assembly, as well as code samples.

Software Requirements

- Microsoft Visual Studio 2010 Express or other Visual Studio 2010 edition: `http://www.microsoft.com/visualstudio/en-us/products/2010-editions/express`
- Microsoft .NET Framework 4
- The Kinect for Windows SDK (x86 or x64): `http://www.kinectforwindows.com`
- For C++ SkeletalViewer samples:
  - DirectX Software Development Kit, June 2010 or later version: `http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=6812`
  - DirectX End-User Runtime Web Installer: `http://www.microsoft.com/download/en/details.aspx?displaylang=en&id=35`

To take full advantage of the audio capabilities of Kinect, you will also need additional Microsoft speech recognition software: the Speech Platform API, the Speech Platform SDK, and the Kinect for Windows Runtime Language Pack. Fortunately, the install for the SDK automatically installs these additional components for you. Should you ever accidentally uninstall these speech components,

however, it is important to be aware that the other Kinect features, such as depth processing and skeleton tracking, are fully functional even without the speech components.

Step-By-Step Installation

Before installing the Kinect for Windows SDK:

1. Verify that your Kinect device is not plugged into the computer you are installing to.
2. Verify that Visual Studio is closed during the installation process.

If you have other Kinect drivers on your computer such as those provided by PrimeSense, you should consider removing these. They will not run side-by-side with the SDK and the Kinect drivers provided by Microsoft will not interoperate with other Kinect libraries such as OpenNI or libfreenect. It is possible to install and uninstall the SDK on top of other Kinect platforms and switch back and forth by repeatedly uninstalling and reinstalling the SDK. However, this has also been known to cause inconsistencies, as the wrong driver can occasionally be loaded when performing this procedure. If you plan to go back and forth between different Kinect stacks, installing on separate machines is the safest path.

To uninstall other drivers, including previous versions of those provided with the SDK, go to Programs and Features in the Control Panel, select the name of the driver you wish to remove, and click Uninstall.

Download the appropriate installation msi (x86 or x64) for your computer. If you are uncertain whether your version of Windows is 32-bit or 64-bit, you can right click on the Windows icon on your desktop and go to Properties in order to find out. You can also access your system information by going to the Control Panel and selecting System. Your operating system architecture will be listed next to the title System type. If your OS is 64-bit, you should install the x64 version. Otherwise, install the x86 version of the msi.

Run the installer once it is successfully downloaded to your machine. Follow the Setup wizard prompts until installation of the SDK is complete. Make sure that Kinect's extra power supply is also plugged into a power source. You can now plug your Kinect device into a USB port on your computer. On first connecting the Kinect to your PC, Windows will recognize the device and begin loading the Kinect drivers. You may see a message on your Windows taskbar indicating that this is occurring. When the drivers have finished loading, the LED light on your Kinect will turn a solid green.

You may want to verify that the drivers installed successfully. This is typically a troubleshooting procedure in case you encounter any problems as you run the SDK samples or begin working through the code in this book. In order to verify that the drivers are installed correctly, open the Control Panel and select Device Manager. As Figure 1-5 shows, the Microsoft Kinect node in Device Manager should list three items if the drivers were correctly installed: the Microsoft Kinect Audio Array Control, Microsoft Kinect Camera, and Microsoft Kinect Security Control.