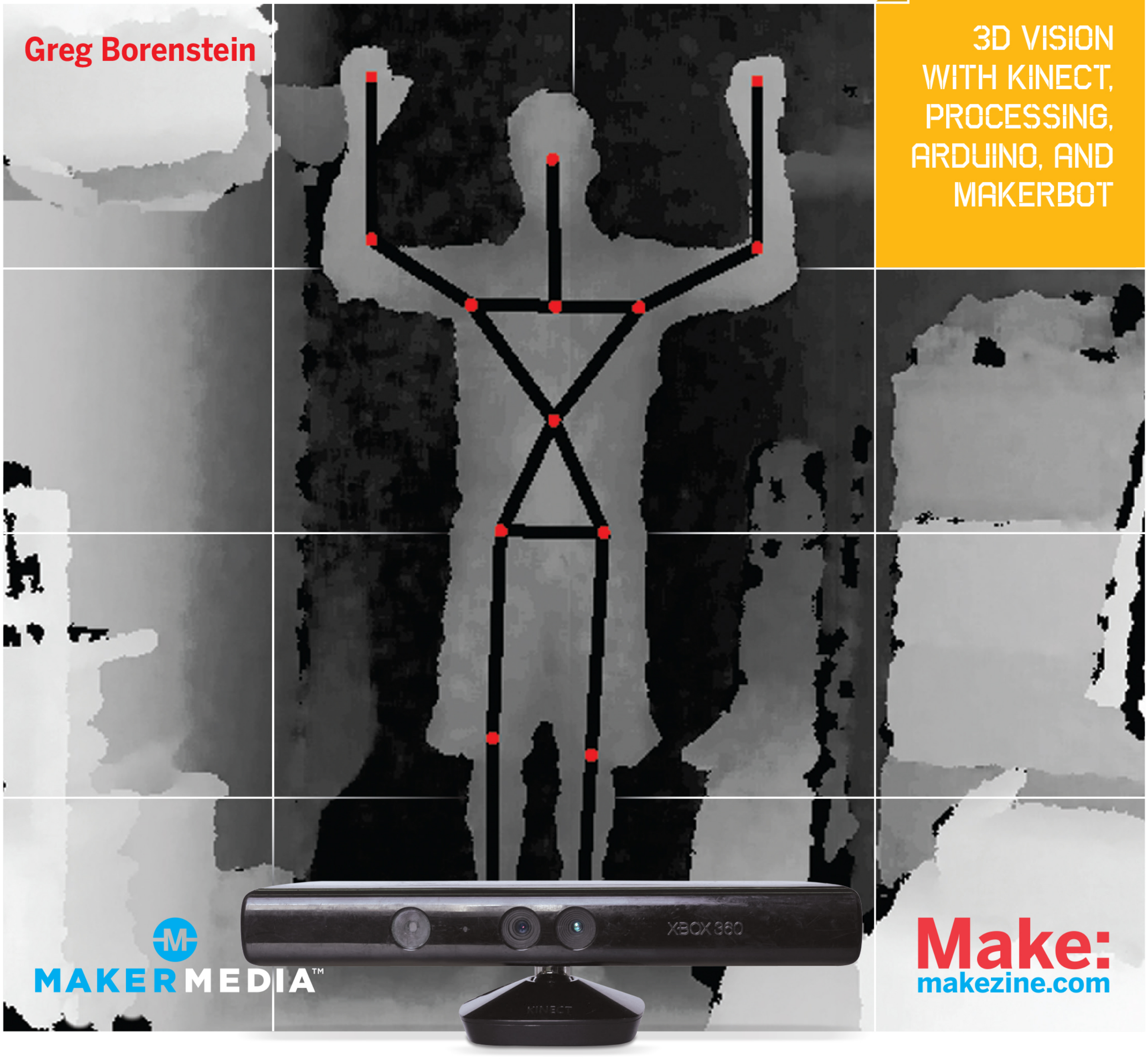
Make: Books

Making Things See

Learn by Discovery

Greg Borenstein

3D VISION WITH KINECT, PROCESSING, ARDUINO, AND MAKERBOT



MAKERMEDIA™

Make:
makezine.com

Making Things See

3D Vision with Kinect, Processing, Arduino, and MakerBot

Greg Borenstein

MAKER MEDIA
Sebastopol, CA

Making Things See

by Greg Borenstein

Published by Maker Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Maker Media books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*my.safaribooksonline.com*). For more information, contact O'Reilly Media's corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Andrew Odewahn, Brian Jepson

Production Editor: Holly Bauer

Proofreader: Linley Dolby

Indexer: Fred Brown

Compositor: Nancy Kotary

Cover Designer: Mark Paglietti

Interior Designer: Ron Bilodeau

Illustrator: Rebecca Demarest

January 2012: First Edition.

Revision History for the First Edition:

2012-01-04	First release
2012-03-16	Second release
2013-03-22	Third release

See *http://oreilly.com/catalog/errata.csp?isbn=0636920020684* for release details.

ISBN: 978-1-449-30707-3
[TI]

For Jacob and Ellie and Sophie and Amalia. The future is yours.

Contents

Preface

When Microsoft first released the Kinect, Matt Webb, CEO of design and invention firm Berg London, captured the sense of possibility that had so many programmers, hardware hackers, and tinkerers so excited:

> "WW2 and ballistics gave us digital computers.
> Cold War decentralization gave us the Internet.
> Terrorism and mass surveillance: Kinect."

Why the Kinect Matters

The Kinect announces a revolution in technology akin to those that shaped the most fundamental breakthroughs of the 20th century. Just like the premiere of the personal computer or the Internet, the release of the Kinect was another moment when the fruit of billions of dollars and decades of research that had previously only been available to the military and the intelligence community fell into the hands of regular people.

Face recognition, gait analysis, skeletonization, depth imaging—this cohort of technologies that had been developed to detect terrorists in public spaces could now suddenly be used for creative civilian purposes: building gestural interfaces for software, building cheap 3D scanners for personalized fabrication, using motion capture for easy 3D character animation, using biometrics to create customized assistive technologies for people with disabilities, etc.

While this development may seem wide-ranging and diverse, it can be summarized simply: for the first time, computers can see. While we've been able to use computers to process still images and video for decades, simply iterating over red, green, and blue pixels misses most of the amazing capabilities that we take for granted in the human vision system: seeing in stereo, differentiating objects in space, tracking people over time and space, recognizing body language, etc. For the first time, with this revolution in camera and image-processing technology, we're starting to build computing applications that take these same capabilities as a starting point. And, with the arrival of the Kinect, the ability to create these applications is now within the reach of even weekend tinkerers and casual hackers.

Just like the personal computer and Internet revolutions before it, this Vision Revolution will surely also lead to an astounding flowering of creative and productive projects. Comparing the arrival of the Kinect to the personal computer and the Internet may sound absurd. But keep in mind that when the personal computer was first invented, it was a geeky toy for tinkerers and enthusiasts. The Internet began life as a way for government researchers to access one anothers' mainframe computers. All of these technologies only came to assume their critical roles in contemporary life slowly as individuals used them to make creative and innovative applications that eventually became fixtures in our daily lives. Right now it may seem absurd to compare the Kinect with the PC and the Internet, but a few decades from now, we may look back on it and compare it with the Altair or the ARPAnet as the first baby step toward a new technological world.

The purpose of this book is to provide the context and skills needed to build exactly these projects that reveal this newly possible world. Those skills include:

- Working with depth information from 3D cameras
- Analyzing and manipulating point clouds
- Tracking the movement of people's joints
- Background removal and scene analysis
- Pose and gesture detection

The first three chapters of this book will introduce you to all of these skills. You'll learn how to implement each of these techniques in the Processing programming environment. We'll start with the absolute basics of accessing the data from the Kinect and build up your ability to write ever more sophisticated programs throughout the book. Learning these skills means not just mastering a particular software library or API, but understanding the principles behind them so that you can apply them even as the practical details of the technology rapidly evolve.

And yet even mastering these basic skills will not be enough to build the projects that really make the most of this Vision Revolution. To do that, you also need to understand some of the wider context of the fields that will be revolutionized by the cheap, easy availability of depth data and skeleton information. To that end, this book will provide introductions and conceptual overviews of the fields of 3D scanning, digital fabrication, robotic vision, and assistive technology. You can think of these sections as teaching you what you can do with the depth and skeleton information once you've gotten it. They will include topics such as:

- Building meshes
- Preparing 3D models for fabrication
- Defining and detecting gestures
- Displaying and manipulating 3D models
- Designing custom input devices for people with limited ranges of motion
- Forward and inverse kinematics

In covering these topics, our focus will expand outward from simply working with the Kinect to using a whole toolbox of software and techniques. The last three chapters of this book will explore these topics through a series of in-depth projects. We'll write a program that uses the Kinect as a scanner to produce physical objects on a 3D printer, we'll create a game that will help a stroke patient with physical therapy, and we'll construct a robot arm that copies the motions of your actual arm. In these projects, we'll start by introducing the basic principles behind each general field and then seeing how our newfound knowledge of programming with the Kinect can put those principles into action. But we won't stop with Processing and the Kinect. We'll work with whatever tools are necessary to build each application, from 3D modeling programs to microcontrollers.

This book will not be a definitive reference to any of these topics; each is vast, comprehensive, and filled with its own fascinating intricacies. This book aims to serve as a provocative introduction to each area—giving you enough context and techniques to start using the Kinect to make interesting projects and hoping that your progress will inspire you to follow the leads provided to investigate further.

Who This Book Is For

At its core, this book is for anyone who wants to learn more about building creative interactive applications with the Kinect, from interaction and game designers who want to build gestural interfaces to makers who want to work with a 3D scanner to artists who want to get started with computer vision.

That said, you will get the most out of it if you are one of the following: a beginning programmer looking to learn more sophisticated graphics and interactions techniques, specifically how to work in three dimensions, or an advanced programmer who wants a shortcut to learning the ins and outs of working with the Kinect and a guide to some of the specialized areas that it enables.

You don't have to be an expert graphics programmer or experienced user of Processing to get started with this book, but if you've never programmed before, there are probably other much better places to start.

As a starting point, I'll assume that you have some exposure to the Processing creative coding language (or can teach yourself that as you go). You should know the basics from *Getting Started with Processing* by Casey Reas and Ben Fry (*http://shop.oreilly.com/product/0636920000570.do*), *Learning Processing* by Dan Shiffman (*http://learningprocessing.com*), or the equivalent. This book is designed to proceed slowly from introductory topics into more sophisticated code and concepts, giving you a smooth introduction to the fundamentals of making interactive graphical applications while teaching you about the Kinect. At the beginning, I'll explain nearly everything about each example, and as we go I'll leave more and more of the details to you to figure out. The goal is for you to level up from a beginner to a confident intermediate interactive graphics programmer.

The Structure of This Book

The goal of this book is to unlock your ability to build interactive applications with the Kinect. It's meant to make you into a card-carrying member of the Vision Revolution I described at the beginning of this introduction. Membership in this Revolution has a number of benefits. Once you've achieved it, you'll be able to play an invisible drum set that makes real sounds, make 3D scans of objects and print copies of them, and teach robots to copy the motions of your arm.

However, membership in this Revolution does not come for free. To gain entry into its ranks, you'll need to learn a series of fundamental programming concepts and techniques. These skills are the basis of all the more advanced benefits of membership, and all of those cool abilities will be impossible without them. This book is designed to build up those skills one at a time, starting from the simplest and most fundamental and building toward the more complex and sophisticated. We'll start out with humble pixels and work our way up to intricate three-dimensional gestures.

Toward this end, the first half of this book will act as a kind of primer in these programming skills. Before we dive into controlling robots or 3D printing our faces, we need to start with the basics. The first four chapters of this book cover the fundamentals of writing Processing programs that use the data from the Kinect.

Processing is a creative coding environment that uses the Java programming language to make it easy for beginners to write simple interactive applications that include graphics and other rich forms of media. As mentioned previously, this book assumes basic knowledge of Processing (or equivalent programming chops), but as we go through these first four chapters, I'll build up your knowledge of some of the more advanced Processing concepts that are most relevant to working with the Kinect. These concepts include looping through arrays of pixels, basic 3D drawing and orientation, and some simple geometric calculations.

I will attempt to explain each of these concepts clearly and in depth. The idea is for you not to just to have a few project recipes that you can make by rote, but to actually understand enough of the flavor of the basic ingredients to be able to invent your own "dishes" and modify the ones I present here. At times, you may feel that I'm beating some particular subject to death, but stick with it—you'll frequently find that these details become critically important later on when trying to get your own application ideas to work.

One nice side benefit to this approach is that these fundamental skills are relevant to a lot more than just working with the Kinect. If you master them here in the course of your work with the Kinect, they will serve you well throughout all your other work with Processing, unlocking many new possibilities in your work, and really pushing you decisively beyond beginner status.

There are three fundamental techniques that we need to build all of the fancy applications that make the Kinect so exciting: processing the depth image, working in 3D, and accessing the skeleton data. From 3D scanning to robotic vision, all of these applications measure the distance of objects using the depth image, reconstruct the image as a three-dimensional scene, and track the movement of individual parts of a user's body. The first half of this book will serve as an introduction to each of these techniques. I'll explain how the data provided by the Kinect makes these techniques possible, demonstrate how to implement them in code, and walk you through a few simple examples to show what they might be good for.

Working with the Depth Camera

First off, you'll learn how to work with the depth data provided by the Kinect. The Kinect uses an IR projector and camera to produce a "depth image" of the scene in front of it. Unlike conventional images in which each pixel records the color of light that reached the camera from that part of the scene, each pixel of this depth image records the distance of the object in that part of the scene from the Kinect. When we look at depth images, they will look like strangely distorted black and white pictures. They look strange because the color of each part of the image indicates not how bright that object is, but how far away it is. The brightest parts of the image are the closest, and the darkest parts are the farthest away. If we write a Processing program that examines the brightness of each pixel in this depth image, we can figure out the distance of every object in front of the Kinect. Using this same technique and a little bit of clever coding, we can also follow the closest point as it moves, which can be a convenient way of tracking a user for simple interactivity.

Working with Point Clouds

This first approach treats the depth data as if it were only two-dimensional. It looks at the depth information captured by the Kinect as a flat image when really it describes a three-dimensional scene. In the third chapter, we'll start looking at ways to translate from these two-dimensional pixels into points in three-dimensional space. For each pixel in the depth image, we can think of its position within the image as its x-y coordinates. That is, if we're looking at a pixel that's 50 pixels in from the top-left corner and 100 pixels down, it has an x-coordinate of 50 and a y-coordinate of 100. But the pixel also has a grayscale value. And we know from our initial discussion of the depth image that each pixel's grayscale value corresponds to the depth of the image in front of it. Hence, that value will represent the pixel's z-coordinate.

Once we've converted all our two-dimensional grayscale pixels into three-dimensional points in space, we have what is called a *point cloud*—that is, a bunch of disconnected points floating near each other in three-dimensional space in a way that corresponds to the arrangement of the objects and people in front of the Kinect. You can think of this point cloud as the 3D equivalent of a pixelated image. While it might look solid from far away, if we look closely, the image will break down into a bunch of distinct points with space visible between them. If we wanted to convert these points into a smooth continuous surface, we'd need to figure out a way to connect them with a large number of polygons to fill in the gaps. This is a process called *constructing a mesh*, and it's something we'll cover extensively later in the book in the chapters on physical fabrication and animation.

For now, though, there's a lot we can do with the point cloud itself. First of all, the point cloud is just cool. Having a live 3D representation of yourself and your surroundings on your screen that you can manipulate and view from different angles feels a little bit like being in the future. It's the first time in using the Kinect that you'll get a view of the world that feels fundamentally different that those that you're used to seeing through conventional cameras.

To make the most of this new view, you're going to learn some of the fundamentals of writing code that navigates and draws in 3D. When you start working in 3D, there are a number of common pitfalls that I'll try to help you avoid. For example, it's easy to get so disoriented as you navigate in 3D space that the shapes you draw end up not being visible. I'll explain how the 3D axes work in Processing and show you some tools for navigating and drawing within them without getting confused. Another frequent area of confusion in 3D drawing is the concept of the camera. To translate our 3D points from the Kinect into a 2D image that we can actually draw on our flat computer screens, Processing uses the metaphor of a camera. After we've arranged our points in 3D space, we place a virtual camera at a particular spot in that space, aim it at the points we've drawn, and, basically, take a picture. Just as a real camera flattens the objects in front of it into a 2D image, this virtual camera does the same with our 3D geometry. Everything that the camera sees gets rendered onto the screen from the angle and in the way that it sees it. Anything that's out of the camera's view doesn't get rendered. I'll show you how to control the position of the camera so that all of the 3D points from the Kinect that you want to see end up rendered on the screen. I'll also demonstrate how to move the camera around so we can look at our point cloud from different angles without having to ever physically move the Kinect.

Working with the Skeleton Data

The third technique is in some ways both the simplest to work with and the most powerful. In addition to the raw depth information we've been working with so far, the Kinect can, with the help of some additional software, recognize people and tell us where they are in space. Specifically, our Processing code can access the location of each part of a user's body in 3D: we can get the exact position of hands, head, elbows, feet, etc.

One of the big advantages of depth images is that computer vision algorithms work better on them than on conventional color images. The reason Microsoft developed and shipped a depth camera as a controller for the Xbox was not to show players cool looking point clouds, but because they could run software on the Xbox that processes the depth image in order to locate people and find the positions of their body parts. This process is known as *skeletonization* because the software infers the position of a user's skeleton (specifically, his joints and the bones that connect them) from the data in the depth image.

By using the right Processing library, we can get access to this user position data without having to implement this incredibly sophisticated skeletonization algorithm ourself. We can simply ask for the 3D position of any joint we're interested in and then use that data to make our applications interactive. In Chapter 4, I'll demonstrate how to access the skeleton data from the Kinect Processing library and how to use it to make our applications interactive. To create truly rich interactions, we'll need to learn some more sophisticated 3D programming. In Chapter 3, when working with point clouds, we'll cover the basics of 3D drawing and navigation. Then, we'll add to those skills by learning more advanced tools for comparing 3D points with each other, tracking their movement, and even recording it for later playback. These new techniques will serve as the basic vocabulary for some exciting new interfaces we can use in our sketches, letting users communicate with us by striking poses, doing dance moves, and performing exercises (among many other natural human movements).

Once we've covered all three of these fundamental techniques for working with the Kinect, we'll be ready to move on to the cool applications that probably drew you to this book in the first place. This book's premise is that what's truly exciting about the Kinect is that it unlocks areas of computer interaction that were previously only accessible to researchers with labs full of expensive experimental equipment. With the Kinect, things like 3D scanning and advanced robotic vision are suddenly available to anyone with a Kinect and an understanding of the fundamentals described here. But to make the most of these new possibilities, you need a bit of background in the actual application areas. To build robots that mimic human movements, it's not enough just to know how to access the Kinect's skeleton data, you also need some familiarity with inverse kinematics, the study of how to position a robot's joints in order to achieve a particular pose. To create 3D scans that can be used for fabrication or computer graphics, it's not enough to understand how to work with the point cloud from the Kinect, you need to know how to build up a mesh from those points and how to prepare and process it for fabrication on a MakerBot, a CNC machine, or 3D printer.

The final two chapters will provide you with introductions to exactly these topics: 3D scanning for fabrication and 3D vision for robotics.

3D Scanning for Digital Fabrication

In Chapter 5, we'll move from people to objects. We'll use the Kinect as a 3D scanner to capture the geometry of a physical object in digital form and then we'll prepare that data for fabrication on a 3D printer. We'll learn how to process the depth points from the Kinect to turn them into a continuous surface or mesh. Then we'll learn how to export this mesh in a standard file format so we can work with it outside of Processing. I'll introduce you to a few free programs that help you clean up the mesh and prepare it for fabrication. Once our mesh is ready to go, we'll examine what it takes to print it out on a series of different rapid prototyping systems. We'll use a MakerBot to print it out in plastic and we'll submit it to Shapeways, a website that will print out our object in a variety of materials from sandstone to steel.

Computer Vision for Robotics

In Chapter 6, we'll see what the Kinect can do for robotics. Robotic vision is a huge topic that's been around for more than 50 years. Its achievements include robots that have driven on the moon and ones that assemble automobiles. For this chapter, we'll build a simple robot arm that reproduces the position of your real arm as detected by the Kinect. We'll send the joint data from Processing to the robot over a serial connection. Our robot's brain will be an Arduino microcontroller. Arduino is Processing's electronic cousin; it makes it just as easy to create interactive electronics as Processing does interactive graphical applications. The Arduino will listen to the commands from Processing and control the robot's motors to execute them.

We'll approach this project in two different ways. First we'll reproduce the angles of your joints as detected by the Kinect. This approach falls into *forward kinematics*, an approach to robotics in which the robot's final position is the result of setting its joints to a series of known angles. Then we'll reprogram our robot so that it can follow the movement of any of your joints. This will be an experiment in *inverse kinematics*. Rather than knowing exactly how we want our robot to move, we'll only know what we want its final position to be. We'll have to teach it how to calculate all the individual angle changes necessary to get there. This is a much harder problem than the forward kinematic problem. A serious solution to it can involve complex math and confusing code. Ours will be quite simple and not very sophisticated, but will provide an interesting introduction to the problems you'd encounter in more advanced robotics applications.

None of these chapters are meant to be definitive guides to their respective areas, but instead to give you just enough background to get started applying these Kinect fundamentals in order to build your own ideas.

Unlike the first four chapters, which attempt to instill fundamental techniques deeply, these last three are meant to inspire a sense of the breadth and diversity of what's possible with the Kinect. Instead of proceeding slowly and thoroughly through comprehensive explanations of principles, these later chapters are structured as individual projects. They'll take a single project idea from one of these topic areas and execute it completely from beginning to end. In the course of these projects, we'll frequently find ourselves moving beyond just writing Processing code. We'll have to interview occupational therapists, work with assistive technology patients, clean up 3D meshes, use a 3D animation program, solder a circuit, and program an Arduino. Along the way, you'll gain brief exposure to a lot of new ideas and tools, but nothing like the in-depth understanding of the first four chapters. We'll move fast. It will be exciting. You won't believe the things you'll make.

Every step of the way in these projects, we'll rely on your knowledge from the first half of the book. So pay close attention as we proceed through these fundamentals, they're the building blocks of everything else throughout this book, and getting a good grasp on them will make it all the easier for you to build whatever it is you're dreaming of.

Then, at the end of the book, our scope will widen. Having come so far in your 3D programming chops and your understanding of the Kinect, I'll point you toward next steps that you can take to take your applications even further. We'll discuss other environments and programming languages besides Processing where you can work with the Kinect. These range from creative coding libraries in other languages such as C++ to interactive graphical environments such as Max/MSP, Pure Data, and Quartz Composer. And there's also Microsoft's own set of development tools, which let you deeply integrate the Kinect with Windows. I'll explain some of the advantages and opportunities of each environment to give you a sense of why you'd want to try it out. Also, I'll point you toward other resources that you can use to get started in each area.

In addition to exploring other programming environments, you can take your Kinect work further by learning about 3D graphics in general. Under the hood, Processing's 3D drawing code is based on OpenGL, a widely used standard for computer graphics. OpenGL is a huge, complex, and powerful system, and Processing only exposes you to the tiniest bit of it. Learning more about OpenGL itself will unlock all kinds of more advanced possibilities for your Kinect applications. I'll point you toward resources both within Processing and outside of it that will enable you to continue your graphics education and make ever more beautiful and compelling 3D graphics.

Acknowledgments

It's a cliché of acknowledgments to say that all books with solo bylines are really collaborative efforts. In this case, I'll go further and say that I myself am one. Specifically, my possession of the necessary knowledge and abilities to write this book was the direct product of the excellent and patient work of a series of amazing teachers I had at NYU's Interactive Telecommunications Program. This book would have been inconceivable without them.

Dan Shiffman's passion spurred my initial interest in the Kinect; his tireless aid as a professor, technical editor, and friend got me through learning and writing about it; and his inspiring abilities as a teacher and writer gave me a goal to aspire to.

Kyle McDonald and Zach Lieberman taught a short, seven-week class in the spring of 2011 that changed my life. That course introduced me to many of the techniques and concepts I attempt to pass on in this book. I hope my presentation of this material is half as clear and thorough as theirs. Further, Zach came up with the idea for the artist interviews, which ended up as one of my favorite parts of this book. And Kyle was invaluable in helping me translate his work on 3D scanning for fabrication, which makes up the soul of Chapter 5.

Dan O'Sullivan, the chair of ITP, and Red Burns, its founder and patron saint, gave me the space and institutional support to take on this intimidating project and created an environment that gave me the confidence and connections to complete it.

Lily Szajnberg was my first student and ideal reader. The best explanations in this book were forced out of me by her hunger to understand and honesty about when I wasn't making sense.

I'd like to thank Andrew Odewahn and Brian Jepson from O'Reilly. Andrew was the first person—even before me—to believe I could write a book. His early feedback helped turn this project from a very long blog post into a book. Brian's constant and continuous work has made this book better in a thousand ways I'll never be able to fully recount.

Max Rheiner created the SimpleOpenNI library I use throughout this book and acted as a technical editor making sure I got all the details right. This book would have been more difficult and come out worse without his work.

I'd also like to thank all the artists who agreed to be interviewed: Robert Hodgin, Elliot Woods, blablablLAB, Nicolas Burrus, Oliver Kreylos, Alejandro Crawford, Kyle McDonald (again), Josh Blake, and Phil Torrone and Limor Fried from Adafruit. Your work, and the hope of seeing more like it, is why I wrote this book.

Huge thanks to Liz Arum and Matt Griffin from MakerBot as well as Catarina Mota, who helped me get up to speed on making good prints, and Duann Scott from Shapeways, who made sure my prints would arrive in time to be included.

And finally, my family and friends and fellow ITP Residents who put up with me while I was writing: I love you.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from MAKE books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Making Things See* by Greg Borenstein (MAKE). Copyright 2012 Greg Borenstein, 978-1-449-30707-3."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, email addresses, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values or by values determined by context.

This box signifies a tip, suggestion, or general note.

Warning

This box indicates a warning or caution.

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

Maker Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from Maker Media and other publishers, sign up for free at *http://my.safaribooksonline.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

> Maker Media, Inc.
> 1005 Gravenstein Highway North
> Sebastopol, CA 95472
> 800-998-9938 (in the United States or Canada)
> 707-829-0515 (international or local)
> 707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

> *http://www.oreilly.com/catalog/9781449307073*

To comment or ask technical questions about this book, send email to:

> *bookquestions@oreilly.com*

For more information about our publications, events, and products, see our website at *http://makermedia.com*.

Find us on Facebook: *https://www.facebook.com/makemagazine*

Follow us on Twitter: *https://twitter.com/make*

Watch us on YouTube: *http://www.youtube.com/makemagazine*

1 What Is the Kinect?

We've talked a little bit about all the amazing applications that depth cameras like the Kinect make possible. But how does the Kinect actually work? What kind of image does it produce and why is it useful? How does the Kinect gather depth data about the scene in front of it? What's inside that sleek little black box?

IN THIS CHAPTER

How Does It Work? Where Did It Come From?

In the next few sections of this introduction, I'll provide some background about where the Kinect came from as well as a little info on how the device works. This issue of the Kinect's provenance may seem like it's only of academic interest. However, as we'll see, it is actually central when deciding which of the many available libraries we should use to write our programs with the Kinect. It's also a fascinating and inspiring story of what the open source community can do.

What Does the Kinect Do?

The Kinect is a *depth camera*. Normal cameras collect the light that bounces off of the objects in front of them. They turn this light into an image that resembles what we see with our own eyes. The Kinect, on the other hand, records the distance of the objects that are placed in front of it. It uses infrared light to create an image (a *depth image*) that captures not what the objects look like, but where they are in space. In the next section of this introduction, I'll explain how the Kinect actually works. I'll describe what hardware it uses to capture this depth image and explain some of its limitations. But first I'd like to explain why you'd actually want a depth image. What can you do with a depth image that you can't with a conventional color image?

First of all, a depth image is much easier for a computer to "understand" than a conventional color image. Any program that's trying to understand an image starts with its pixels and tries to find and recognize the people and objects represented by them. If you're a computer program and you're looking at color pixels, it's very difficult to differentiate objects and people. So much of the color of the pixels is determined by the light in the room at the time the image was captured, the aperture and color shift of the camera, and so on. How would you even know where one object begins and another ends, let alone which object was which and if there were any people present? In a depth image, on the other hand, the color of each pixel tells you how far that part of the image is from the camera. Since these values directly correspond to where the objects are in space, they're much more useful in determining where one object begins, where another ends, and if there are any people around. Also, because of how the Kinect creates its depth image (about which you'll learn more in a second) it is not sensitive to the light conditions in the room at the time it was captured. The Kinect will capture the same depth image in a bright room as in a pitch black one. This makes depth images more reliable and even easier for a computer program to understand.

We'll explore this aspect of depth images much more thoroughly in Chapter 2.

A depth image also contains accurate three-dimensional information about whatever's in front of it. Unlike a conventional camera, which captures how things *look*, a depth camera captures where things *are*. The result is that we can use the data from a depth camera like the Kinect to reconstruct a 3D model of whatever the camera sees. We can then manipulate this model, viewing it from additional angles interactively, combining it with other preexisting 3D models, and even using it as part of a digital fabrication process to produce new physical objects. None of this can be done with conventional color cameras.

We'll begin exploring these possibilities in Chapter 3 and then continue with them in Chapter 5 when we investigate scanning for fabrication.

And finally, since depth images are so much easier to process than conventional color images, we can run some truly cutting-edge processing on them. Specifically, we can use them to detect and track individual people, even locating their individual joints and body parts. In many ways, this is the Kinect's most exciting capability. In fact, Microsoft developed the Kinect specifically for the opportunities this body-detection ability offered to video games (more about this in "Who Made the Kinect?" on page 6). Tracking users' individual body parts creates amazing possibilities for our own interactive applications. Thankfully, we have access to software that can perform this processing and simply give us the location of the users. We don't have to analyze the depth image ourselves in order to obtain this information, but it's only accessible because of the depth image's suitability for processing.

We'll work extensively with the user-tracking data in Chapter 4.

What's Inside? How Does It Work?

If you remove the black plastic casing from the Kinect, what will you find? What are the hardware components that make the Kinect work, and how do they work together to give the Kinect its abilities? Let's take a look. Figure 1-1 shows a picture of a Kinect that's been freed from its case.

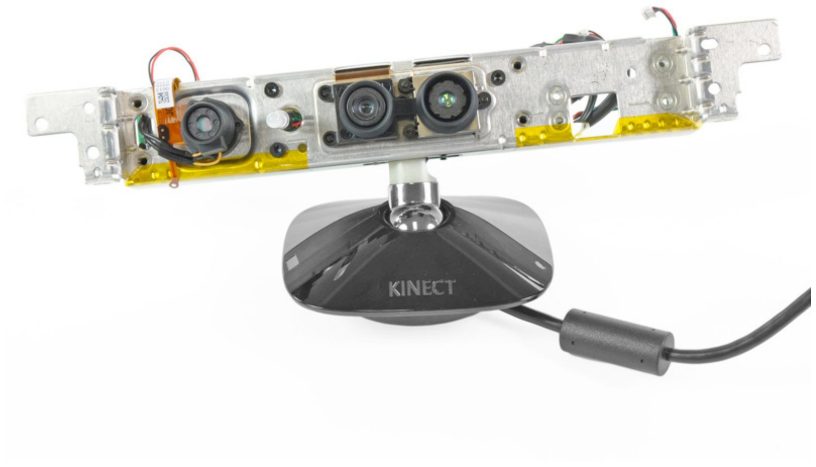


Figure 1-1. *A Kinect with its plastic casing removed, revealing (from left to right) its IR projector, RGB camera, and IR camera. (Photo courtesy of iFixit.)*

The first thing I always notice when looking at the Kinect *au natural* is its uncanny resemblance to various cute movie robots. From *Short Circuit*'s Johnny 5 to Pixar's WALL-E, for decades movie designers have been creating human-looking robots with cameras for eyes. It seems somehow appropriate (or maybe just inevitable) that the Kinect, the first computer peripheral to bring cutting-edge computer vision capabilities into our homes, would end up looking so much like one of these robots.

Unlike these movie robots, though, the Kinect seems to actually have three eyes: the two in its center and one off all the way to one side. That "third eye" is the secret to how the Kinect works. Like most robot "eyes," the two protuberances at the center of the Kinect are cameras, but the Kinect's third eye is actually an infrared projector. Infrared light has a wavelength that's longer than that of visible light so we cannot see it with the naked eye. Infrared is perfectly harmless—we're constantly exposed to it every day in the form of sunlight.

The Kinect's infrared projector shines a grid of infrared dots over everything in front of it. These dots are normally invisible to us, but it is possible to capture a picture of them using an IR camera. Figure 1-2 shows an example of what the dots from the Kinect's projector look like.

I captured this image using the Kinect itself. One of those two cameras I pointed out earlier (one of the Kinect's two "eyes") is an IR camera. It's a sensor specifically designed to capture infrared light. In Figure 1-1, an image of the Kinect naked without its outer case, the IR camera is the one on the right. If you look closely, you can see that this camera's lens has a greenish iridescent sheen as compared with the standard visible light camera next to it.