TECHNOLOGY IN ACTION™

Hacking the Kinect

WRITE CODE AND CREATE INTERESTING PROJECTS INVOLVING MICROSOFT'S GROUND-BREAKING VOLUMETRIC SENSOR

Jeff Kramer, Nicolas Burrus, Florian Echtler, Daniel Herrera C., and Matt Parker

Hacking the Kinect

Jeff Kramer
Nicolas Burrus
Florian Echtler
Daniel Herrera C.
Matt Parker

Apress®

Hacking the Kinect

ISBN 978-1-4302-3867-6

ISBN 978-1-4302-3868-3 (eBook)

President and Publisher: Paul Manning
Lead Editor: Jonathan Gennick
Technical Reviewer: Curiomotion LLC
Editorial Board: Steve Anglin, Mark Beckner, Ewan Buckingham, Gary Cornell, Louise Corrigan, Morgan Ertel, Jonathan Gennick, Jonathan Hassell, 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: Anita Castro
Copy Editor: Heather Lang
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/.

I dedicate this book to Jennifer Marriott, the light of my life.

—Jeff Kramer

For my family and for Lauren, who helped me in every possible way.

—Matt Parker

Contents at a Glance

Contents

About the Authors

■ Jeff Kramer (@Qworg) is a builder, maker, hacker, and dreamer. Jeff is currently a research programmer and roboticist at the National Robotics Engineering Center (NREC) in Pittsburgh, PA (www.rec.ri.cmu.edu/). He also breaks things and rebuilds them all of the time. He's taught graduate courses in animal psychology, chaired international robotics conference sessions, made Bill Nye cry with a weapon of mass destruction, constructed high-powered arc lamps, written journal articles in robotics, devastated players with a robotic foozball table, and generally made a mess—a beautiful, strangely functional mess that is always evolving. Jeff holds a master's degree in robotics from the University of South Florida. You can check him out at http://mind-melt.com/ and http://about.me/JeffreyKramer/. Or just e-mail him at jeffkramr@gmail.com.

Nicolas Burrus is a researcher in computer vision at the Carlos III University of Madrid, with a special interest in 3D object model acquisition and recognition for robotic applications. He actively took part in the impressive wave of interest that followed the release of the Kinect by publishing RGBDemo, an open source software showcasing many applications of the Kinect. RGBDemo is being used by many research labs, companies, and hobbyists, and this success led him to co-found the Manctl startup with the ambition of developing a low-cost universal 3D scanner. He holds a PhD from Paris VI University and a master's of computer science from EPITA (Paris, France).

Florian Echtler is a researcher in human-computer interaction and information security, currently working at Siemens Corporate Technology in Munich, Germany. He wrote the very first Kinect hack, an interface in the style of *Minority Report*, in less than 24 hours after the first release of the open source drivers. He is the main author of libTISCH, a NUI development platform that supports the Kinect as an input device and is actively used in several research projects. He holds a diploma and a PhD in computer science from the Technical University of Munich (TUM).

Daniel Herrera C. works as a computer vision researcher at the University of Oulu, Finland. He is currently doing his PhD in the areas of image-based rendering and free viewpoint video. His early work with the Kinect led him to develop an open source calibration algorithm. The Kinect Calibration Toolbox is now being used by researchers around the world. Since then the Kinect has become one of the main tools for his research. He holds a master's in computer vision and robotics from the Erasmus Mundus program Vibot (UK, Spain, and France).

Matt Parker is a new media artist and game designer. As an artist, his interest lies in exploring the intersection of the physical and digital worlds. His work has been displayed at the American Museum of Natural History, SIGGRAPH Asia, the NY Hall of Science, Museum of the Moving Image, FILE Games Rio, Sony Wonder Technology Lab, and many other venues. His game Lucid was a finalist in Android's Developer Challenge 2, and his game Recurse was a finalist for Indiecade 2010. His project Lumarca won the "Create the Future" Prize at the World Maker Faire 2010.

Matt earned his BS in computer science from Vassar College and a master's from NYU's Interactive Telecommunications Program (ITP). He has served as a new media researcher and adjunct faculty member at NYU since 2009. He is currently a visiting professor at Sarah Lawrence College and an artist in residence at Eyebeam Art and Technology Center.

About the Technical Reviewer

Curiomotion LLC is a technology company dedicated not to providing a specific product or service, but to bringing tomorrow's technology to today's applications. Currently, Curiomotion's team of software engineers is working with the latest advancements in motion sensing technology to enhance consumer engagement in retail scenarios. Curiomotion is one of the first companies to offer products for this new paradigm of technology-driven commerce, providing interactive marketing solutions compatible with any business strategy.

Acknowledgments

A book of this magnitude cannot be accomplished alone. First, I would like to thank everyone at Apress for giving me the opportunity to write about something truly exciting and game changing. I'm proud that we are supporting future creative and commercial endeavors using the Kinect and other 3D sensors. I would like to thank Jonathan Gennick, my lead editor, who first reached out to me and offered just a chapter, then half a book, and then a book. Without him, I would have never started. I would like to thank Anita Castro, my coordinating editor, who dealt with my fuzzy deadlines, last-minute changes, and never-ending shenanigans with grace and dignity. I know I put her through hell, and I simply cannot apologize enough. Without her, I would have never finished. Technical reviewer Max Choi's insight not only caught bugs and hammered on inconsistencies, but also truly turned the examples from mere toys to finished products. Copy editor Heather Lang's deft touch polished my oft-times clunky prose and refitted all of the text—a monumental task. Thank you both. Michelle Lowman, part of the editorial board, also deserves thanks for her role in refining the ideas that led to this book.

Working with genuinely amazing people never gets old. I would like to thank Ed Paradis (http://edparadis.com/) for his unending support, as well as his deft photography and Lego skills, especially on the final chapter. He was an excellent person to bounce ideas off of and helped me get my chapters finished. I would like to thank my coauthors—Daniel, Florian, Matt, and Nicolas. You guys took a chance on me, on this book, and it panned out! Like jumping into a very cold lake—you just have to start swimming. Thank you for your efforts and contributions. This book would not have been even a quarter as good without you.

I would like to thank Kyle Wiens over at iFixit (www.ifixit.com) for letting me use one of their teardown images, and for providing such an amazing resource for hardware hackers everywhere. If you can't fix it, you don't own it. I would like to thank the team building PCL (http://pointclouds.org/) for all of their crucial work—they are truly bringing 3D manipulation to the masses.

I would like to thank Dr. Abraham Kandel. As always, your support from afar carries me through rough spots, and your example inspires me to always do more, better. Thank you.

I would like to thank my mom and dad for providing such an excellent example and their support growing up. I would have never been in this position without them.

Last and most certainly not least, I would like to thank my soon-to-be wife, Jennifer Marriott, and our lovely daughter, Charlotte. Late nights, missed dinners, angst, misery, and snark—you both stood by me, supported me, and loved me, even when it was hard to do so. I love you both in ways I cannot express in words. Thank you!

Jeff Kramer

I wish to thank all those who have called me their friend in this cold city, for any endeavor is rendered empty if there is nobody to share it with. And a special mention to those who have fed me, for I often forget to do it myself.

Daniel Herrera C.

I'd like to thank NYU ITP, Eyebeam Art and Technology Center, and the Openframeworks community.

Matt Parker

CHAPTER 1

Introducing the Kinect

Welcome to *Hacking the Kinect*. This book will introduce you to the Kinect hardware and help you master using the device in your own programs. We're going to be covering a large amount of ground—everything you'll need to get a 3-D application running—with an eye toward killer algorithms, with no unusable filler.

Each chapter will introduce more information about the Kinect itself or about the methods to work with the data. The data methods will be stretched across two chapters: the first introduces the concept and giving a basic demonstration of algorithms and use, and the second goes into more depth. In that second chapter, we will show how to avoid or ameliorate common issues, as well as discuss more advanced algorithms. All chapters, barring this one, will contain a project—some basic, some advanced.

We expect that you will be able to finish each chapter and immediately apply the concepts into a project of you own; there is plenty of room for ingenuity with the first commercial depth sensor and camera!

Hardware Requirements and Overview

The Kinect requires the following computer hardware to function correctly. We'll cover the requirements more in depth in Chapter 3, but these are the basic requirements:

- A computer with at least one, mostly free, USB 2.0 hub.
    - The Kinect takes about 70% of a single hub (not port!) to transmit its data.
    - Most systems can achieve this easily, but some palmtops and laptops cannot. To be certain, flip to Chapter 2, where we give you a quick guide on how to find out.
- A graphics card capable of handling OpenGL. Most modern computers that have at least an onboard graphics processor can accomplish this.
- A machine that can handle 20 MB/second of data (multiplied by the number of Kinects you're using). Modern computers should be able to handle this easily, but some netbooks will have trouble.
- A Kinect sensor power supply if your Kinect came with your Xbox 360 console rather than standalone.

Figure 1-1 shows the Kinect itself. The callouts in the figure identify the major hardware components of the device. You get two cameras: one infrared and one for standard, visible light. There is an infrared emitter to provide structured light that the infrared camera uses to calculate the depth

image. The status light is completely user controlled, but it will tell you when the device is plugged into the USB (but not necessarily powered!) by flashing green.

Figure 1-1. *Kinect hardware at a glance*

Installing Drivers

This book focuses on the OpenKinect driver – a totally open source, low level driver for the Kinect. There are a few other options (OpenNI and the Kinect for Windows SDK), but for reasons to be further discussed in Chapter 3, we'll be using OpenKinect. In short, OpenKinect is totally open source, user supported and low level, therefore extremely fast. The examples in this book will be written in C/C++, but you can use your favorite programming language; the concepts will definitely carry over.

■ **Note** Installation instructions are split into three parts, one for each available OS to install to. Please skip to the section for the OS that you're using.

Windows

While installing and building OpenKinect drivers from source is fairly straightforward, it can be complicated for first timers. These steps will take you through how to install on Windows 7 (and should also work for earlier versions of Windows).

1. Download and install Git (`http://git-scm.com`). Be sure to select "Run git from the Windows Command Prompt" and "Check out Windows style, commit Unix-style line endings".
2. Open your command prompt; go to the directory where you want your source folder to be installed, and clone/branch as in Listing 1-1. See the "Git Basics" sidebar for more information.

Listing 1-1. *Git Commands for Pulling the Source Code*

```
C:\> mkdir libfreenect
C:\> cd libfreenect
C:\libfreenect> git clone https://github.com/OpenKinect/libfreenect.git (This will clone into
a new libfreenect directory)
C:\libfreenect> cd libfreenect
C:\libfreenect\libfreenect> git branch –track unstable origin/unstable
```

3. There are three major dependencies that must be installed for `libfreenect` to function: `libusb-win32`, `pthreads-win32`, and GLUT. Some of the options you select in the next section are dependent on your choice of compiler.
   a. Download `libusb-win32` from `http://sourceforge.net/projects/libusb-win32/`.
   b. Extract and move the resulting folder into `/libfreenect`.
   c. Download `pthreads-win32` from `http://sourceware.org/pthreads-win32/`. Find the most recent candidate with `release.exe` at the end.
   d. Extract and store the folder in `/libfreenect`. If you're using Microsoft Visual Studio 2010, copy `/Pre-built.2/lib/pthreadVC2.dll` to `/Windows/System32/`. If using MinGW, copy `/Pre-built.2/lib/pthreadGC2.dll` to `/Windows/System32/` instead.
   e. Download GLUT from `http://www.xmission.com/~nate/glut.html`. Find the most recent release ending in "`-bin.zip`".
   f. Extract and store the resulting folder in `/libfreenect`.
   g. Copy `glut32.dll` to `/Windows/System32/`. If you're using Microsoft Visual Studio 2010, copy `glut.h` to the `/include/GL` folder in your Visual Studio tree and `glut32.lib` library to `/lib` in the same tree. If the GL folder does not exist, create it. However, if you're using MinGW, copy `glut.h` to `/include/GL` folder in the MinGW root directory.
4. All of the dependencies are in place! Now we can install the low-level Kinect device driver.
   a. Plug in your Kinect. After a quick search for drivers, your system should complain that it cannot find the correct drivers, and the LED on the Kinect itself will not light. This is normal.
   b. Open Device Manager. Start Control Panel Hardware and Sound Device Manager.
   c. Double-click Xbox NUI Motor. Click Update Driver in the new window that appears.
   d. Select "Browse my computer for driver software", and browse to `/libfreenect/platform/inf/xbox nui motor/`.
   e. After installation, the LED on the Kinect should be blinking green. Repeat steps 3 and 4 for Xbox NUI Camera and Xbox NUI Audio.

5. Download CMake from `www.cmake.org/cmake/resources/software.html`. Get the most recent `.exe` installer, and install it.
6. Make sure you have a working C compiler, either MinGW or Visual Studio 2010.
7. Launch `CMake-GUI`, select `/libfreenect` as the source folder, select an output folder, and click the Grouped and Advanced check boxes to show more options.
8. Click Configure. You see quite a few errors. This is normal! Make sure that CMake matches closely to Figure 1-2. At the time of this writing, Fakenect is not working on Windows, so uncheck its box.

■ **Note** MinGW is a minimal development environment for Windows that requires no external third-party runtime DLLs. It is a completely open source option to develop native Windows applications. You can find out more about it at `www.mingw.org`.

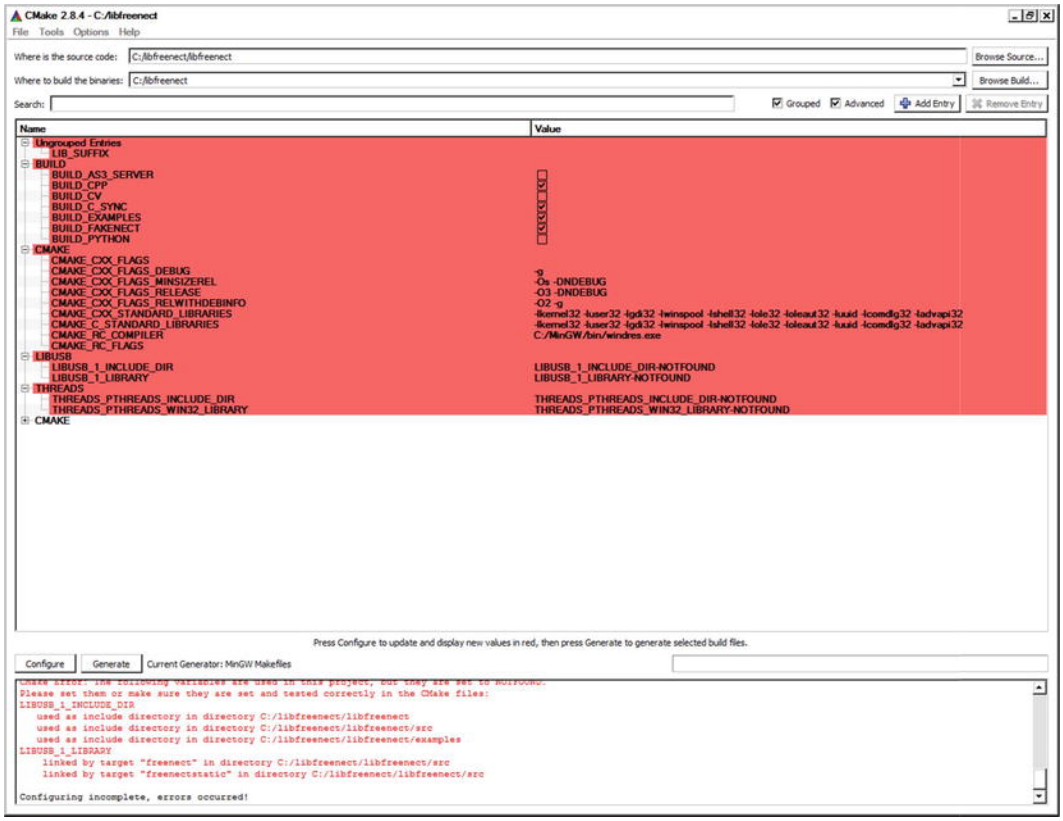

Figure 1-2. *CMake preconfiguration*

9. Here too, the following steps split based on compiler choices; this installation step is summarized in Table 1-1.

    a. For Microsoft Visual Studio 2010, `GLUT_INCLUDE_DIR` is the `/include` directory in your Visual Studio tree. `GLUT_glut_LIBRARY` is the actual full path to `glut32.lib` in your Visual Studio tree. `LIBUSB_1_LIBRARY` is `/lib/msvc/libusb.lib` in the `libusb` installation directory. `THREADS_PTHREADS_WIN32_LIBRARY` is `/Pre-built.2/lib/pthreadVC2.lib` in the `pthreads` installation directory.

    b. For MinGW, the following choices must be set: `GLUT_INCLUDE_DIR` is the GLUT root directory. `GLUT_glut_LIBRARY` is the actual full path to `glut32.lib` in the GLUT root directory. `LIBUSB_1_LIBRARY` is `/lib/gcc/libusb.a` in the `libusb` installation directory. `THREADS_PTHREADS_WIN32_LIBRARY` is `/Pre-built.2/lib/pthreadGC2.a` in the pthreads installation directory.

c. For both, the following choices must be set: `LIBUSB_1_INCLUDE_DIR` is `/include` in the libusb installation directory. `THREADS_PTHREADS_INCLUDE_DIR` is `/Pre-built.2/include` in the pthreads installation directory.

Table 1-1. *CMake Settings for Microsoft Visual Studio 2010 and MinGW*

CMake Setting	Microsoft Visual Studio 2010	MinGW
GLUT_INCLUDE_DIR	<MSVSRoot>/VC/include	<GLUTRoot>/
GLUT_glut_LIBRARY	<MSVSRoot>/VC/lib/glut32.lib	<GLUTRoot>/glut32.lib
LIBUSB_1_INCLUDE_DIR	<LIBUSBRoot>/include	<LIBUSBRoot>/include
LIBUSB_1_LIBRARY	<LIBUSBRoot>/lib/msvc/libusb.lib	<LIBUSBRoot>/lib/gcc/libusb.a
THREADS_PTHREADS_INCLUDE_DIR	<PTHREADRoot>/Pre-built.2/include	<PTHREADRoot>/Pre-built.2/include
THREADS_PTHREADS_WIN32_LIBRARY	<PTHREADRoot>/Pre-built.2/lib/pthreadVC2.lib	<PTHREADRoot>/Pre-built.2/lib/pthreadGC2.a

10. Dependencies that have yet to be resolved are in red. Click Configure again to see if everything gets fixed.
11. As soon as everything is clear, click Generate.
12. Open your chosen output folder, and compile using your compiler.
13. Test by running `/bin/glview.exe`.

■ **Note** If you have problems compiling in Windows, check out the fixes in Chapter 3 to get your Kinect running.

Linux

Installing on Linux is a far simpler than on Windows. We'll go over both Ubuntu and Red Hat/Fedora. For both systems, you need to install the following dependencies; the first line in each of the listings below takes care of this step for you:

- `git-core`
- `cmake`
- `libglut3-dev`
- `pkg-config`
- `build-essential`

- libxmu-dev
- libxi-dev
- libusb-1.0.0-dev

Ubuntu

Run the commands in Listing 1-2. Follow up by making a file named 51-kinect.rules in /etc/udev/rules.d/, as shown in Listing 1-3, and 66-kinect.rules in the same location, as shown in Listing 1-4.

Listing 1-2. *Ubuntu Kinect Installation Commands*

```
sudo apt-get install git-core cmake libglut3-dev pkg-config build-essential libxmu-dev libxi-
dev libusb-1.0-0-dev
git clone https://github.com/OpenKinect/libfreenect.git
cd libfreenect
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig /usr/local/lib64/
sudo adduser <SystemUserName> video
sudo glview
```

Listing 1-3. *51-kinect.rules*

```
# ATTR{product}=="Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02b0", MODE="0666"
# ATTR{product}=="Xbox NUI Audio"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ad", MODE="0666"
# ATTR{product}=="Xbox NUI Camera"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ae", MODE="0666"
```

Listing 1-4. *66-kinect.rules*

```
#Rules for Kinect
SYSFS{idVendor}=="045e", SYSFS{idProduct}=="02ae", MODE="0660",GROUP="video"
SYSFS{idVendor}=="045e", SYSFS{idProduct}=="02ad", MODE="0660",GROUP="video"
SYSFS{idVendor}=="045e", SYSFS{idProduct}=="02b0", MODE="0660",GROUP="video"
#End
```

Red Hat / Fedora

Use Listing 1-5 to install, and then make the files in Listings 1-3 and 1-4 in /etc/udev/rules.d/.

Listing 1-5. *Red Hat/Fedora Kinect Installation Commands*

```
yum install git cmake gcc gcc-c++ libusb1 libusb1-devel libXi libXi-devel libXmu libXmu-devel
freeglut freeglut-devel
git clone https://github.com/OpenKinect/libfreenect.git
cd libfreenect
mkdir build
cd build
cmake ..
make
sudo make install
sudo ldconfig /usr/local/lib64/
sudo adduser <SystemUserName> video
sudo glview
```

Mac OS X

There are several package installers for OS X, but we'll be focusing on MacPorts (Fink and Homebrew are not as well supported and are too new for most users). The maintainers of OpenKinect have issued a special port of `libusb-devel` that is specifically patched to work for the Kinect. Move to a working directory, and then issue the commands in Listing 1-6. If you want to build an XCode project instead of a CMake one, change the `cmake` line to `cmake -G Xcode . . .` . In `cmake`, configure and generate before exiting to run the remaining code.

■ **Note** MacPorts is an open source system to compile, install, and upgrade software on your OS X machine. It is the Mac equivalent of `apt-get`. Although it is almost always properly compiles new libraries on your system, it is extremely slow due to its "reinstall everything to verify" policy. Homebrew is up and coming and will likely be the package manager of the future. To learn more about MacPorts, please visit `www.macports.org`.

Listing 1-6. *Mac OS X Kinect Installation Commands*

```
sudo port install git-core
sudo port install libtool
sudo port install libusb-devel
sudo port install cmake
git clone https://github.com/OpenKinect/libfreenect.git
cd libfreenect/
mkdir build
cd build
ccmake ..
Run Configure to generate the initial build description.
Double check the settings in the configuration (this shouldn't be an issue with a standard
installation)
Run Generate and Exit
```

```
make
sudo make install
```

Testing Your Installation

Our driver of choice, `libfreenect`, helpfully ships with a small set of demonstration programs. You can find these in the `/bin` directory of your build directory. The most demonstrative of these is `glview`; it shows an attempt at fitting the color camera to the 3D space. Your `glview` output should look much like Figure 1-3.

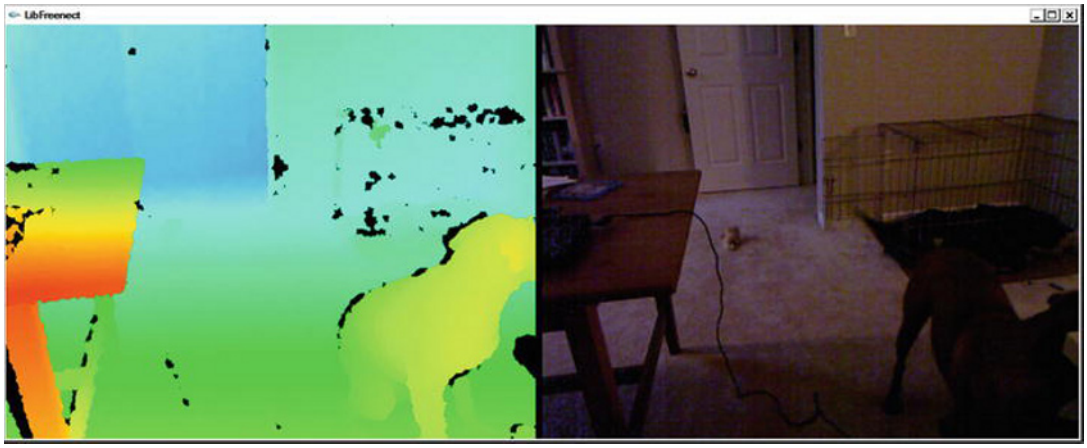

Figure 1-3. *glview capture*

Getting Help

While much of the information in this chapter should be straightforward, there are sometimes hiccups in the process. In that case, there are several places to seek help. OpenKinect has one of the friendliest communities out there, so do not hesitate to ask questions.

- `http://openkinect.org`: This is the home page for the OpenKinect community; it also has the wiki and is full of great information.
- `http://groups.google.com/group/openkinect`: This is the OpenKinect user group mailing list, which hosts discussions and answers questions.
- `IRC: #OpenKinect` on `irc.freenode.net`

Summary

In this first chapter, you installed the initial driver software, OpenKinect, and ran your first 3-D application on your computer. Congratulations on entering a new world! In the next chapter, we're going to dive deep into the hardware of the Kinect itself, discussing how the depth image is generated and some of the limitations of your device.

CHAPTER 2

Hardware

In this chapter, you will extensively explore the Kinect hardware, covering all aspects of the system including foibles and limitations. This will include the following:

- How the depth sensing works
- Why you can't use your Kinect outside
- System requirements and limitations

Let's get started.

Figure 2-1. *Kinect external diagram*

Depth Sensing

Figure 2-1 will serve as your guidebook to the Kinect hardware. Let's start with the depth sensing system. It consists of two parts: the IR laser emitter and the IR camera. The IR laser emitter creates a known noisy pattern of structured IR light at 830 nm. The output of the emitter is shown in Figure 2-2. Notice the nine brighter dots in the pattern? Those are caused by the imperfect filtering of light to create the pattern. Prime Sense Ltd., the company that worked with Microsoft to develop the Kinect, has a patent (US20100118123) on this process, as filters to create light like this usually end up with one extremely bright dot in the center instead of several moderately bright dots. The change from a single bright dot to

several moderately bright dots is definitely a big advancement because it allows for the use of a higher powered laser.

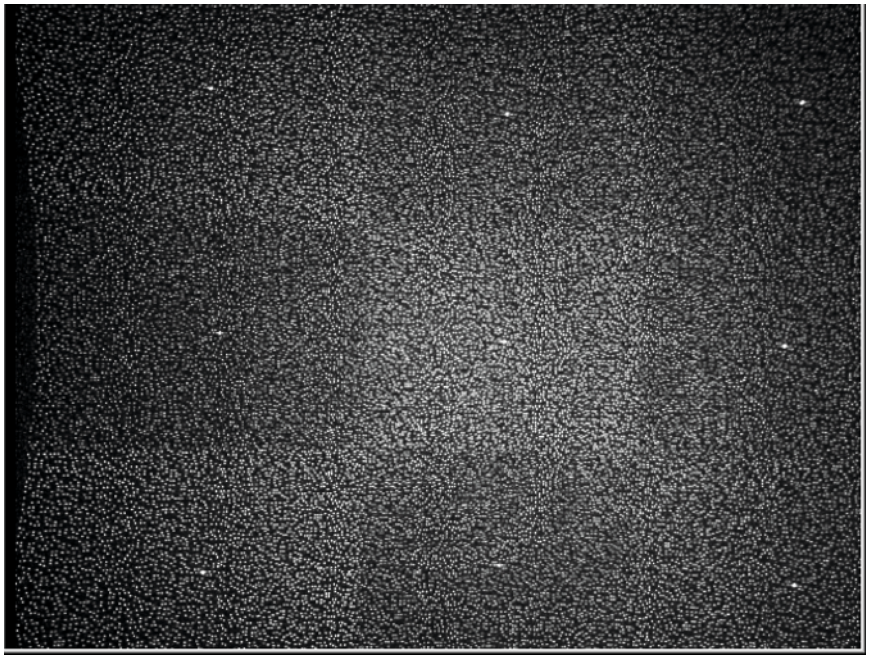

Figure 2-2. *Structured light pattern from the IR emitter*

The depth sensing works on a principle of **structured light**. There's a known pseudorandom pattern of dots being pushed out from the camera. These dots are recorded by the IR camera and then compared to the known pattern. Any disturbances are known to be variations in the surface and can be detected as closer or further away. This approach creates three problems, all derived from a central requirement: light matters.

- The wavelength must be constant.
- Ambient light can cause issues.
- Distance is limited by the emitter strength.

The wavelength consistency is mostly handled for you. Within the sensor, there is a small peltier heater/cooler that keeps the laser diode at a constant temperature. This ensures that the output wavelength remains as constant as possible (given variations in temperature and power).

Ambient light is the bane of structured light sensors. Again, there are measures put in place to mitigate this issue. One is an IR-pass filter at 830 nm over the IR camera. This prevents stray IR in other ranges (like from TV remotes and the like) from blinding the sensor or providing spurious results. However, even with this in place, the Kinect does not work well in places lit by sunlight. Sunlight's wide band IR has enough power in the 830 nm range to blind the sensor.

The distance at which the Kinect functions is also limited by the power of the laser diode. The laser diode power is limited by what is eye safe. Without the inclusion of the scattering filter, the laser diode in