The production of three-dimensional data is the primary function of Kinect. It is up to you to create exciting experiences with the data. A precondition to building a Kinect application is having an understanding of the output of the hardware. Beyond simply understanding, the intrinsic meaning of the 1’s and 0’s is a comprehension of its existential significance. Image-processing techniques exist today that detect the shapes and contours of objects within an image. The Kinect SDK uses image processing to track user movements in the skeleton tracking engine. Depth image processing can also detect non-human objects such as a chair or coffee cup. There are numerous commercial labs and universities actively studying techniques to perform this level of object detection from depth images. There are many different uses and fields of study around depth input that it would be impossible to cover them or cover any one topic with considerable profundity in this book much less a single chapter. The goal of this chapter is to detail the depth data down to the meaning of each bit, and introduce you to the possible impact that adding just one additional dimension can have on an application. In this chapter, we discuss some basic concepts of depth image processing, and simple techniques for using this data in your applications.

Seeing Through the Eyes of the Kinect

Kinect is different from all other input devices, because it provides a third dimension. It does this using an infrared emitter and camera. Unlike other Kinect SDKs such as OpenNI, or libfreenect, the Microsoft SDK does not provide raw access to the IR stream. Instead, the Kinect SDK processes the IR data returned by the infrared camera to produce a depth image. Depth image data comes from a DepthImageFrame, which is produced by the DepthImageStream.

Working with the DepthImageStream is similar to the ColorImageStream. The DepthImageStream and ColorImageStream both share the same parent class ImageStream. We create images from a frame of depth data just as we did with the color stream data. Begin to see the depth stream images by following these steps, which by now should look familiar. They are the same as from the previous chapter where we worked with the color stream.

1. Create a new WPF Application project.
3. Add an Image element to MainWindow.xaml and name it “DepthImage”.
4. Add the necessary code to detect and initialize a KinectSensor object. Refer to Chapter 2 as needed.
5. Update the code that initializes the KinectSensor object so that it matches Listing 3-1.
Listing 3-1. Initializing the DepthStream

this._KinectDevice.DepthStream.Enable();
this._KinectDevice.DepthFrameReady += KinectDevice_DepthFrameReady;

6. Add the DepthFrameReady event handler code, as shown in Listing 3-2. For
the sake of being brief with the code listing, we are not using the
WriteableBitmap to create depth images. We leave this as a refactoring
exercise for you to undertake. Refer to Listing 2-5 of Chapter 2 as needed.

Listing 3-2. DepthFrameReady Event Handler

using(DepthImageFrame frame = e.OpenDepthImageFrame())
{
    if(frame != null)
    {
        short[] pixelData = new short[frame.PixelDataLength];
        frame.CopyPixelDataTo(pixelData);
        int stride = frame.Width * frame.BytesPerPixel;
        DepthImage.Source = BitmapSource.Create(frame.Width, frame.Height, 96, 96,
                                                PixelFormats.Gray16, null,
                                                pixelData, stride);
    }
}

7. Run the application!

When Kinect has a new depth image frame available for processing, the KinectSensor fires the
DepthFrameReady event. Our event handler simply takes the image data and creates a bitmap, which is
then displayed in the UI window. The screenshot in Figure 3-1 is an example of the depth stream image.
Objects near Kinect are a dark shade of gray or black. The farther an object is from Kinect, the lighter the
gray.