Kinect Chapter 2.1. Charting the Depth Map

This chapter revisits the depth map example of Chapter 2 to illustrate a more numerical way of viewing the information. The first two versions of the ViewerPanel class in Chapter 2 convert the depth measurements into grayscale images. It's quite easy to insert a few lines of code to also display the bar chart shown in Figure 1.

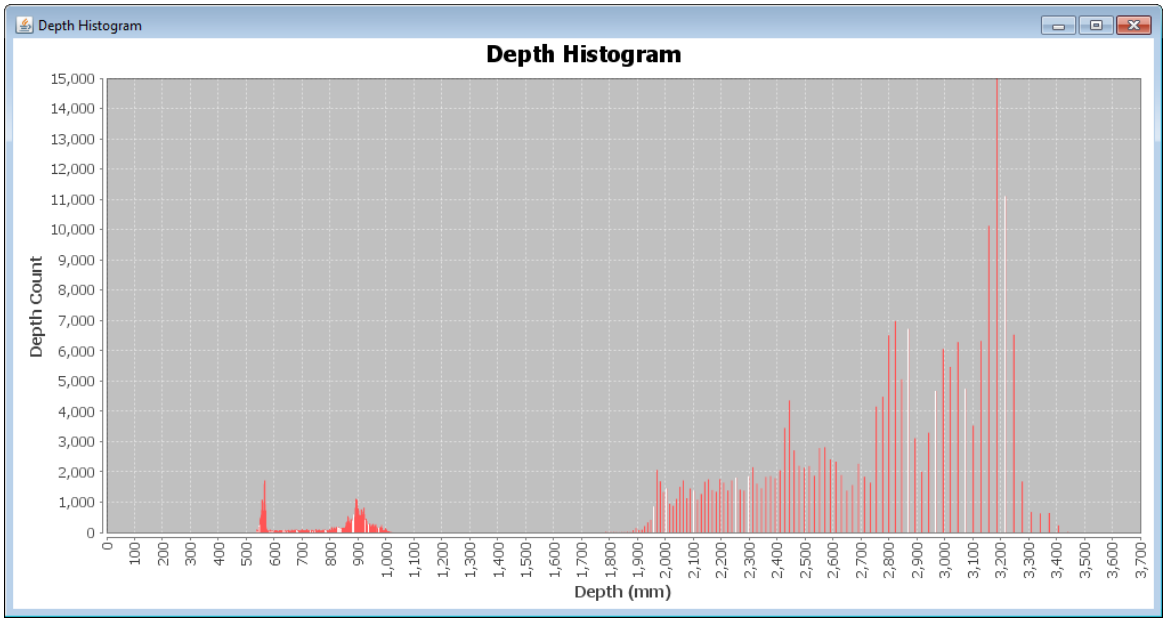

Figure 1. Bar Chart of Depths.

The graph's x-axis show the Kinect's depth measurements, which typically range from 500 to 3500 mm. The y-axis is for the number of measurements returned for a particular depth.

The corresponding grayscale depth image is shown in Figure 2. Recall that darker means further away, although black also means "too close" for a depth value to be calculated, or that no depth data was returned (e.g. for the edges of the user's arm).

Figure 2. The Depth Image Corresponding to Figure 1.

Figure 1 contains three main peaks, one at around 540 mm, another at 900 mm, and a wide spread of depths starting from 1900 mm and extending out to 3400 mm. A look at Figure 2 shows that the first peak is my raised hand and the spread is the back wall of my office.

The XY bar chart in Figure 1 is created using the JFreeChart library (http://www.jfree.org/jfreechart/), which allows the user to zoom-in (and out) on a chart. For instance, I used the mouse to select a region around the first peak, and the chart was redrawn as in Figure 3.

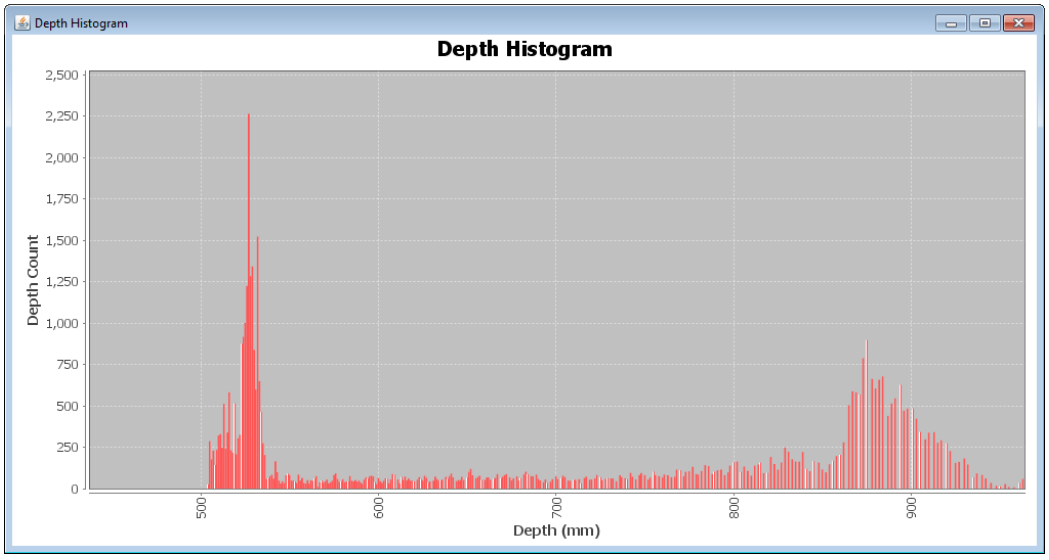

Figure 3. A Closer Look at the Left of Figure 1.

The necessary modifications to the ViewerPanel class to add this chart are quite modest since JFreeChart handles the complex details of chart updating and rendering.

One issue I had was chart update frequency: chart rendering takes over 0.5 seconds, and slows down OpenNIViewer by too much if done for every Kinect update. Instead, ViewerPanel imposes a 2 second delay between chart updates so the application isn't affected too often.

Before I describe the charting version of ViewerPanel, I'll briefly introduce JFreeChart with two short examples for drawing a 3D pie chart and a XY bar chart.

1. JFreeChart

JFreeChart (http://www.jfree.org/jfreechart/) can generate an enormous range of charts, including standard ones such as pie, bar, line, and Gantt charts, many variations and combinations of those charts, and many less standard graphs such as spider and vector plots. Every chart includes useful features such as tool tips, zooming, and automatic redrawing of dynamically changing data.

Despite this diversity, the API offers a fairly standard and simple approach to creating a chart, which I'll illustrate with two examples.

A limited number of chart screenshots can be found at the JFreeChart site (at http://www.jfree.org/jfreechart/samples.html), but the best collection of samples is in the demo JAR (jfreechart-1.0.14-demo.jar) which comes as part of the JFreeChart installation.

I downloaded jfreechart-1.0.14.zip, and unzipped the contents to a new location. The two important libraries, jfreechart-1.0.14.jar and jcommon-1.0.17.jar can be found in jfreechart-1.0.14\lib\, and must be added to the classpath when code is being compiled and run. The demo application, jfreechart-1.0.14-demo.jar, is in jfreechart-1.0.14\

Double-clicking on jfreechart-1.0.14-demo.jar starts the application shown in Figure 4.

Figure 4. The JFreeChart Demo.

Selecting a chart from the tree on the left, triggers it's display on the right. Unfortunately, JFreeChart doesn't include the source code for these examples, although its quite easy to decompile the JAR with a tool such as jd-gui (http://java.decompiler.free.fr/?q=jdgui).

Another excellent source of JFreeChart examples is the java2s site (http://www.java2s.com/Code/Java/Chart/CatalogChart.htm), which lists over 70 chart categories, usually with multiple examples for each type. Most employ JFreeChart, but a few utilize other charting libraries.

1.1. Creating a Chart

There are three main stages in creating a JFreeChart chart:

1. create a data set
2. create a chart using the data set
3. add the chart to a panel for rendering with Swing

The data set is managed by a subclass of the Dataset class, chosen to match the chart type that you'll create in stage 2 using a method from the ChartFactory class. The API documentation for these classes is at http://www.jfree.org/jfreechart/api/javadoc/.

JFreeChart includes a subclass of JPanel, called ChartPanel, for holding the chart. ChartPanel extends JPanel with useful zooming and property features, accessible via the right mouse button.

1.2. A Pie Chart

The following example, called SimplePie.java, creates a 3D pie chart containing three pieces of data, which is displayed in an application window (see Figure 5).

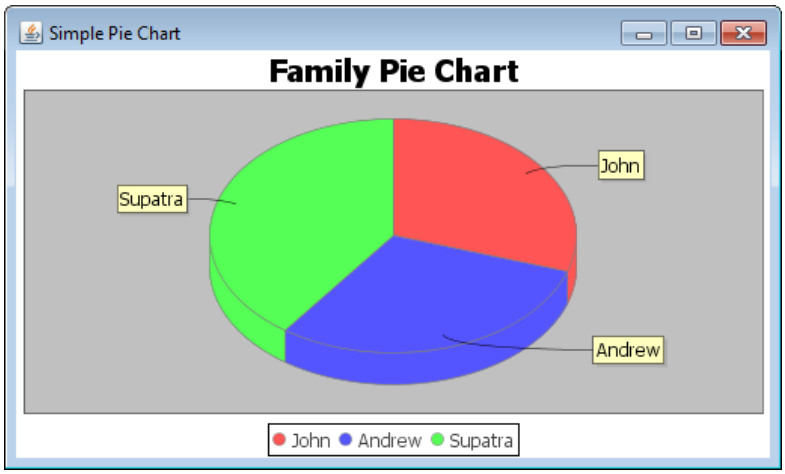

Figure 5. Creating a 3D Pie Chart.

The SimplePie class is:

```
public class SimplePie extends JFrame
{
  public SimplePie()
  {
    super("Simple Pie Chart");

    // (1) create the data set
    DefaultPieDataset dataset = new DefaultPieDataset();
    dataset.setValue("John", 30);
    dataset.setValue("Andrew", 30);
    dataset.setValue("Supatra",40);

    // (2) put the data into a chart
    JFreeChart chart = ChartFactory.createPieChart3D(
                   "Family Pie Chart", dataset, true, true, false);
                                          //legend?, tips?, urls?

    // (3) add the chart to a panel
    ChartPanel chartPanel = new ChartPanel(chart);
    chartPanel.setPreferredSize(new Dimension(500, 270));

    // add the panel to the window
    setContentPane(chartPanel);
    pack();
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setVisible(true);
  }  // end of SimplePie()
```

The DefaultPieDataset is filled with three data items, and then passed to the ChartFactory.createPieChart3D() call. The resulting chart is added to a ChartPanel object which is placed inside the application's JFrame.

1.3. A XY Bar Chart

The same three stages are used by my SimpleXYSeries class for creating the XY bar chart shown in Figure 6 .

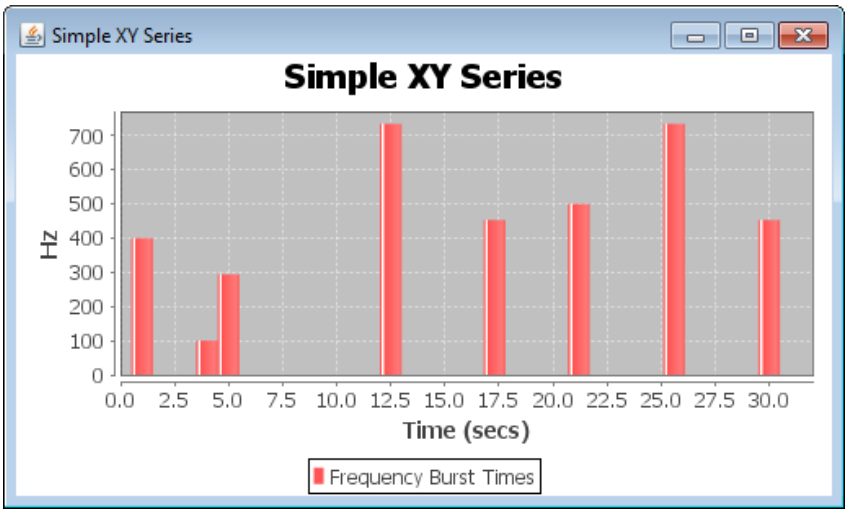

Figure 6. A XY Series Bar Chart.

The SimpleXYSeries constructor is defined as:

```
public SimpleXYSeries()
{
  super("Simple XY Series");

  XYSeriesCollection dataset = createDataset();   // stage (1)

  // (2) put the data into a chart
  JFreeChart chart = ChartFactory.createXYBarChart(
                   "Simple XY Series", "Time (secs)", false,
                                                 // is x a date axis?
                   "Hz", dataset, PlotOrientation.VERTICAL,
                   true, true, false);
               // legend?, tooltips?, urls?

  // (3) add the chart to a panel
  ChartPanel chartPanel = new ChartPanel(chart);
  chartPanel.setPreferredSize(new Dimension(500, 270));

  // add the panel to the window
  setContentPane(chartPanel);
  pack();
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  setVisible(true);
}  // end of SimpleXYSeries()
```

The data set is sufficiently complex that I've moved it's creation to a createDataset() method:

```
private XYSeriesCollection createDataset()
{
  // a sequence of (x,y) data items
  XYSeries series = new XYSeries("Frequency Burst Times");
  series.add(1.0, 400.2);
  series.add(5.0, 294.1);
  series.add(4.0, 100.0);
  series.add(12.5, 734.4);
```



```
    series.add(17.3, 453.2);
    series.add(21.2, 500.2);
    series.add(21.9, null);     // null means missing value
    series.add(25.6, 734.4);
    series.add(30.0, 453.2);

    XYSeriesCollection dataset = new XYSeriesCollection(series);
    return dataset;
}  // end of createDataset()
```

The XYSeries object represents a sequence of zero or more (x, y) data items. By default, items in the series are sorted into ascending order by x-value, and duplicate x-values are permitted. The series is converted into a data set by being stored in an XYSeriesCollection object.

Back in the constructor, a different ChartFactory() creation method is called -- ChartFactory.createXYBarChart() – which includes arguments for the x- and y- axis labels and the plot orientation.

We'll shortly revisit these JFreeChart methods in my Kinect code since the depth chart shown in Figure 1 is also a XY bar chart.

2. Adding a Chart to the Depth Viewer

The OpenNIViewer application which displays the depth image in Figure 2 was described in the last chapter (primarily in section 7). Its classes are shown in Figure 7.

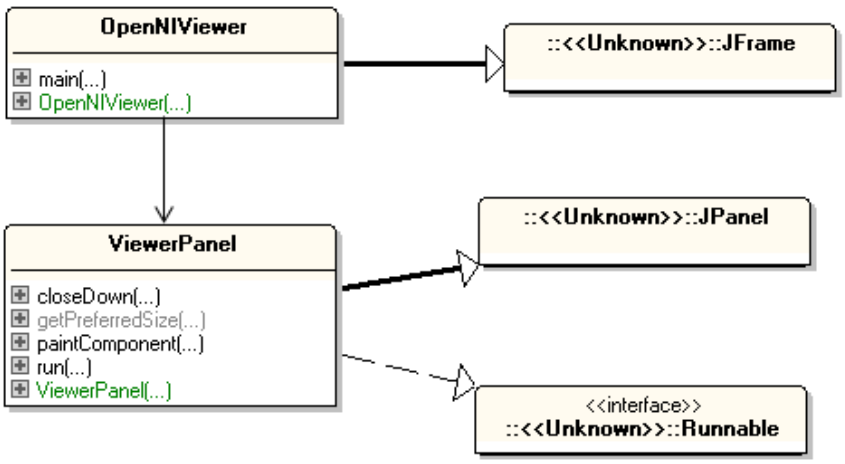

Figure 7. Class Diagrams for OpenNIViewer.

The charting code is added to ViewerPanel since it must access the depth data stored in the histogram[] array when the depth map is read from the Kinect.

The majority of the new code is located in two methods: initChart() which creates the bar chart and renders it in its own JFrame, and updateChart() which uses the current depths stored in histogram[] to update the chart's dataset.

2.1. Initializing the Chart

Since I'm using an XY bar chart again, initChart() looks a lot like the SimpleXYChart constructor from section 1.3. As before, there are three main stages: the creation of the data set (a collection of zero depths initially), the initialization of the chart, and its addition to a panel in a window. The initChart() method:

```
// globals
private static final int CHART_MAX_DEPTH = 3700;
                      // used for the chart's x-axis
private XYSeries series;


private void initChart()
{
  // create an initial data set
  series = new XYSeries("Depth Histogram");
  for (int i = 0; i <= CHART_MAX_DEPTH; i++)
    series.add(i,0);     // depth with zero count
  XYSeriesCollection dataset = new XYSeriesCollection(series);

  // put the data into a chart
  JFreeChart chart = ChartFactory.createXYBarChart(
           "Depth Histogram", "Depth (mm)", false, "Pixel Count",
           dataset, PlotOrientation.VERTICAL,
           false, true, false );    // legend, tooltips, urls

  // modify the chart's axes
  XYPlot plot = (XYPlot) chart.getPlot();

  NumberAxis domainAxis = (NumberAxis) plot.getDomainAxis();// x-axis
  domainAxis.setVerticalTickLabels(true);
  domainAxis.setRange(0, CHART_MAX_DEPTH);
  domainAxis.setTickUnit(new NumberTickUnit(100));

  ValueAxis rangeAxis = plot.getRangeAxis();    // y-axis
  rangeAxis.setRange(0,15000);  // a bit of a guess

  // add the chart to a panel
  ChartPanel chartPanel = new ChartPanel(chart);
  chartPanel.setPreferredSize(new Dimension(1000, 500));

  // add the panel to a window
  JFrame chartFrame = new JFrame("Depth Histogram");
  chartFrame.setContentPane(chartPanel);
  chartFrame.pack();
  chartFrame.setVisible(true);
}  // end of initChart()
```

initChart() shows another new JFreeChart feature – the accessing of the chart's plot data so that its x- and y- axes can be adjusted. The x-axis and y-axis ranges are set to be 0-CHART_MAX_DEPTH and 0-15000 respectively. CHART_MAX_DEPTH is a reasonable upper bound for the maximum depth (3700 mm), although we'll see later that it's still necessary for the update code to test for larger depths. The y-axis maximum is more of a guess, but if the graph proves to be too small at run time, the y-axis view can be zoomed out.

Also, the x-axis tick labels are modified to be drawn vertically (to make them easier to read), and the tick interval is set to 100 (i.e. to steps of 100mm).

An example of the chart window created by initChart() is shown in Figure 1.

2.2. Updating the Chart

The ViewerPanel thread repeatedly iterates through a wait/update/render loop. Each update starts by building a histogram array of 8-bit depth values extracted from OpenNI's current depth map, such that histogram[i] holds the total number of i millimeter depths detected in the scene. For instance, if histogram[800] == 100 then it means that a hundred 800mm depths have been recorded by the Kinect.

histogram[] is subsequently converted into a cumulative count of the depths such that histogram[i] contains the number of depths detected at i millimeters or less. However, before this transformation is carried out, the array is passed to updateChart() which updates the chart's data set (stored in the series global).

```
// globals
private static final int CHART_MAX_DEPTH = 3700;
private XYSeries series;


private void updateChart(float histogram[], int maxDepth)
{
  if (maxDepth > CHART_MAX_DEPTH)
    System.out.println("Maximum depth (" + maxDepth +
                                        ") exceeds chart max depth");
  for (int i = 1; i <= CHART_MAX_DEPTH; i++) {
    // skipping histogram[0] which is for unknown depths
    try {
     series.update(((Number)Integer.valueOf(i)), histogram[i]);
    }
    catch(SeriesException e) {
      System.out.println("Problem updating (" + i + ", " +
                                          histogram[i] + ")");
    }
  }
}  // end of updateChart()
```

It's necessary to check that the depth data doesn't exceed the x-axis maximum (CHART_MAX_DEPTH), which happens occasionally when the Kinect miscalculates a depth. This can occur if the infrared beam is reflected in a mirror or glass so it appears to be displaced over a larger distance.

Another aspect of the code is that the XYSeries.update() loop starts at 1 rather than 0. This discards the depth counts stored in histogram[0] which record the number of missing reflections of the infrared light pattern.

One thing that updateChart() doesn't need to do is to force a repaint of the chart's JPanel (e.g. by calling repaint()) – the redrawing is managed by the JFreeChart API.

A drawback of this update approach is the length of time that updateChart() requires to modify all the chart data (typically nearly a second on my slow test machine). This

slows each iteration of the ViewerPanel loop to a crawl. I coded around this problem with a timer, stored in the chartTime global, that reduces the call frequency of updateChart().

The method that calls updateChart() checks the current chartTime value, and only calls it if the time has exceeded CHART_DELAY (2 seconds). When the update has been completed, the timer is reset to 0:

```
// globals
private static final int CHART_DELAY =  2000;
                        // ms time between chart updates

private int chartTime = CHART_DELAY;
                  // so no delay before first update


// code fragment in calcHistogram()
if (chartTime > CHART_DELAY) {
  updateChart(histogram, maxDepth);
  chartTime = 0;
}
```

The chartTime value is increased each time the update loop iterates in run(), using the duration of that iteration:

```
// code fragment in run()
long startTime = System.currentTimeMillis();
updateDepthImage();
  :
chartTime += (System.currentTimeMillis() - startTime);
```

The effect is that the rendering of the grayscale depth image is normally fast, but slows down roughly every 2 seconds (the CHART_DELAY value) when the depth chart is updated.

