Java OOP: Controlling an Edge Detector with a Slider*

R.G. (Dick) Baldwin

This work is produced by OpenStax-CNX and licensed under the
Creative Commons Attribution License 3.0†

Abstract

Learn how to use a slider to continuously change the threshold detection level of an edge detector and to draw the edge-detected image on the screen.

1 Table of Contents

- Preface (p. 1)
  - Viewing tip (p. 1)
    - Figures (p. 2)
    - Listings (p. 2)
- Preview (p. 2)
- Discussion and sample code (p. 5)
- Run the program (p. 9)
- Summary (p. 9)
- What's next? (p. 9)
- Miscellaneous (p. 9)
- Complete program listing (p. 10)

2 Preface

This module is one of a series of modules designed to teach you about Object-Oriented Programming (OOP) using Java.

The program described in this module requires the use of the Guzdial-Ericson multimedia class library. You will find download, installation, and usage instructions for the library at Java OOP: The Guzdial-Ericson Multimedia Class Library [1] .

2.1 Viewing tip

I recommend that you open another copy of this document in a separate browser window and use the following links to easily find and view the figures and listings while you are reading about them.

*Version 1.2: Sep 6, 2012 1:47 pm -0500

†http://creativecommons.org/licenses/by/3.0/

[1] http://cnx.org/content/m44148/latest/

2.1.1 Figures

- Figure 1 (p. 3) . Contents of the file named Prob08.jpg.
- Figure 2 (p. 4) . Output images.

2.1.2 Listings

- Listing 1 (p. 5) . Beginning of the class named Prob08Runner.
- Listing 2 (p. 6) . Beginning of the constructor.
- Listing 3 (p. 7) . Beginning of the edgeDetector method.
- Listing 4 (p. 7) . Detect using adjacent pixels in each column.
- Listing 5 (p. 8) . Register a ChangeEvent listener on the slider.
- Listing 6 (p. 10) . Complete program listing.

3 Preview

An earlier module *(see 3D Displays, Color Distance, and Edge Detection* [2] *)* explained how to implement an edge detection algorithm for a fixed detection threshold.

The primary objective of this module is to illustrate how to use a slider to continuously change the detection threshold of an edge detector and to draw the edge-detected image on the screen.

Brief program specifications

Write a program named **Prob08** that uses the class definition for the class named **Prob08** shown in Listing 6 (p. 10) and Ericson's media library along with the image file named Prob08.jpg [3] *(shown in Figure 1 (p. 3))* to produce the two graphic output images shown in Figure 2 (p. 4) .

[2] http://www.developer.com/java/other/article.php/3798646/3D-Displays-Color-Distance-and-Edge-Detection.htm
[3] http://cnx.org/content/m44913/latest/Prob08.jpg

Contents of the file named Prob08.jpg.

Figure 1: Contents of the file named Prob08.jpg.

Output images.

Figure 2: Output images.

Note that Figure 2 (p. 4) consists of two separate output images. The image containing the slider is positioned directly below the image of the butterfly.

New classes

You may define new classes as necessary to cause your program to behave as required, but you may not modify the class definition for the class named **Prob08** given in Listing 6 (p. 10) .

The output images

The top image shown in Figure 2 (p. 4) is an image of a butterfly to which an edge detection algorithm has been applied.

The bottom image in Figure 2 (p. 4) is a slider that is used to control the edge-detection threshold.

Detect by rows and by columns

The edge-detection algorithm performs edge detection on a **Picture** object by rows and also by columns.

All edges that are detected by processing adjacent pixels on a row are marked in red. All edges that are detected by processing adjacent pixels on a column are marked in black.

If a pixel is determined to be on an edge using both approaches, it ends up being black. If an edge is not detected, the corresponding pixel is marked in white.

Program behavior

At startup, the thumb on the slider is positioned at the 50-percent mark and the image has been edge-detected using a threshold value of 50.

As you move the slider to the right, the threshold value increases up to a value of 100, which in turn causes the amount of white area in the image to increase.

As you move the slider to the left, the threshold value decreases down to a value of zero, which in turn causes the amount of white area in the image to decrease.

The program must terminate and return control to the operating system when you click the large X in the upper-right corner of the GUI containing the slider.

In addition to the output images described above, your program must display your name on the command-line screen.

4 Discussion and sample code

4.1 The class named Prob08

You can view the driver class named **Prob08** at the beginning of the source code in Listing 6 (p. 10) . You are already familiar with the code in the **main** method of that class from earlier modules so I won't spend any time explaining it.

Brief description

Briefly, the **main** method instantiates a new object of the class named **Prob08Runner** and calls the **run** method on that object. When the **run** method returns, the GUI shown in Figure 2 (p. 4) has been displayed on the screen.

At that point, the program simply goes into an idle state and waits for the user to take some action that causes an event to be fired. When an event is fired, it is handled and the program goes idle again waiting for another event.

(Because there are images on the screen, the program does not terminate until the user forces it to terminate.)

4.2 The class named Prob08Runner

Will explain in fragments

I will explain this program in fragments. A complete listing of the program is provided in Listing 6 (p. 10) near the end of the module.

Beginning of the class named Prob08Runner

The class named **Prob08Runner** begins in Listing 1 (p. 5) .

Listing 1: Beginning of the class named Prob08Runner.

```
class Prob08Runner extends JFrame{

private JPanel mainPanel = new JPanel();
private JPanel titlePanel = new JPanel();
private JSlider slider = new JSlider();

private Picture butterfly =
                        new Picture("Prob08.jpg");
```

```
private int butterflyWidth =
                            butterfly.getWidth();
private int butterflyHeight =
                           butterfly.getHeight();

private Picture display =
     new Picture(butterflyWidth,butterflyHeight);

private Pixel pix1;
private Pixel pix2;
private Pixel displayPixel;

private double distance = 0;
```

There is nothing in Listing 1 (p. 5) that I haven't explained in earlier modules, so I won't repeat those explanations here.

Beginning of the constructor

The constructor begins in Listing 2 (p. 6) .

Listing 2: Beginning of the constructor.

```
  public Prob08Runner(){//constructor
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

slider.setMajorTickSpacing(10);
slider.setMinorTickSpacing(5);
slider.setPaintTicks(true);
slider.setPaintLabels(true);

mainPanel.setLayout(new BorderLayout());
titlePanel.add(new JLabel(
                  "Edge Detection Threshold"));
mainPanel.add(titlePanel,BorderLayout.NORTH);
mainPanel.add(slider,BorderLayout.CENTER);

getContentPane().add(mainPanel);

setSize(butterflyWidth + 7,97);
setLocation(0,butterflyHeight + 25);
setVisible(true);

//Produce the initial display with a threshold
// value of 50, which matches the initial
// position of the pointer on the slider.
display = edgeDetector(butterfly,50);

display.show();
```

Put the constructor on hold

The only thing that is new in Listing 2 (p. 6) is the call to the method named **edgeDetector** . I will put the explanation of the constructor on hold while I explain the method named **edgeDetector** .

Beginning of the edgeDetector method

The **edgeDetector** method begins in Listing 3 (p. 7) .

This method performs edge detection on a **Picture** object by rows and also by columns.

As mentioned earlier, all edges that are detected by processing adjacent pixels on a row are marked in red.

All edges that are detected by processing adjacent pixels on a column are marked in black.

If a pixel is determined to be on an edge using both approaches, it ends up being black.

Listing 3: Beginning of the edgeDetector method.

```
  private Picture edgeDetector(
                  Picture picture,int threshold){
for(int row = 0;row < butterflyHeight - 1;row++){
  for(int col = 0;col < butterflyWidth - 1;col++){
    pix1 = picture.getPixel(col,row);
    displayPixel = display.getPixel(col,row);

    //First process two adjacent pixels on the
    // same row.
    pix2 = picture.getPixel(col + 1,row);

    //Get and save the color distance between the
    // two pixels.
    distance = pix1.colorDistance(pix2.getColor());

    //Compare the color distance to the threshold
    // and set the color of the pixel in the
    // display picture accordingly.
    if(distance > threshold){
      displayPixel.setColor(Color.RED);
    }else{
      displayPixel.setColor(Color.WHITE);
    }//end else
```

A pair of nested for loops

Listing 3 (p. 7) shows the beginning of a pair of nested **for** loops that I used to detect the edges in the image of the butterfly. The code in Listing 3 (p. 7) detects edges using adjacent pixels on each row and sets the color of the corresponding pixel in the output picture to red if an edge is detected.

(Note that the output picture referred to by ***display*** *was instantiated as an instance variable in Listing 1 (p. 5) .)*

I explained an edge-detection algorithm in great detail in an earlier module titled 3D Displays, Color Distance, and Edge Detection [4] . The code in Listing 3 (p. 7) is essentially an implementation of that algorithm where the detection threshold is an input parameter to the method.

Detect using adjacent pixels in each column

Listing 4 (p. 7) detects edges using adjacent pixels on the columns.

Listing 4: Detect using adjacent pixels in each column.

[4] http://www.developer.com/java/other/article.php/3798646/3D-Displays-Color-Distance-and-Edge-Detection.htm

```
            //Now process two adjacent pixels in the same
        // column using the same approach.
        pix2 = picture.getPixel(col,row + 1);
        distance = pix1.colorDistance(pix2.getColor());
        //Compare the color distance to the threshold
        // and change pixel color accordingly.
        if(distance > threshold){
          displayPixel.setColor(Color.BLACK);
        }//end if

      }//end inner loop
    }//end outer loop

    return display;
  }//end edgeDetector
```

If an edge is detected in Listing 4 (p. 7) , the color of the corresponding pixel in the output picture is set to black.

Return the output picture and terminate

The **edgeDetector** method returns a reference to the output picture and terminates in Listing 4 (p. 7) .

(While writing this, I realized that because the variable named ***display*** *is an instance variable, the program would also work properly if the* ***edgeDetector*** *method were to return void.)*

Returning to the constructor...

Returning to where we left off in Listing 2 (p. 6) , the **show** method is called on the output picture referred to by **display** causing an image similar to that shown in Figure 2 (p. 4) to be displayed.

Register a ChangeEvent listener on the slider

Listing 5 (p. 8) registers an anonymous listener object on the slider. Each time the slider fires a **ChangeEvent** , the method named **stateChanged** is executed.

Listing 5: Register a ChangeEvent listener on the slider.

```
        slider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){

          display = edgeDetector(
                      butterfly,slider.getValue());
          display.repaint();

        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //-----------------------------------------------//
  }//end constructor
  //-----------------------------------------------//
}//end class Prob08Runner
```

Call the edgeDetector method

The **stateChanged** method calls the **edgeDetector** method to get a new edge-detected image for which the threshold is the current value of the slider.

Then the display is repainted showing the new image on the screen.

The end of the program

Listing 5 (p. 8) also signals the end of the constructor, the end of the class, and the end of the program.

5 Run the program

I encourage you to copy the code from Listing 6 (p. 10) . Compile the code and execute it. Experiment with the code, making changes, and observing the results of your changes. Make certain that you can explain why your changes behave as they do.

6 Summary

In this module, you learned how to use a slider to continuously change the threshold detection level of an edge detector and to draw the edge-detected image on the screen.

7 What's next?

In the next module, you will learn how to use a slider to continuously change the size of an image and to draw the scaled image onto a background image.

8 Miscellaneous

This section contains a variety of miscellaneous information.

NOTE: **Housekeeping material**

- Module name: Java OOP: Controlling an Edge Detector with a Slider
- File: Java3116.htm
- Published: 05/13/12
- Revised: 09/06/12

NOTE: **Disclaimers: Financial** : Although the Connexions site makes it possible for you to download a PDF file for this module at no charge, and also makes it possible for you to purchase a pre-printed version of the PDF file, you should be aware that some of the HTML elements in this module may not translate well into PDF.

I also want you to know that, I receive no financial compensation from the Connexions website even if you purchase the PDF version of the module.

In the past, unknown individuals have copied my modules from cnx.org, converted them to Kindle books, and placed them for sale on Amazon.com showing me as the author. I neither receive compensation for those sales nor do I know who does receive compensation. If you purchase such a book, please be aware that it is a copy of a module that is freely available on cnx.org and that it was made and published without my prior knowledge.

Affiliation : I am a professor of Computer Information Technology at Austin Community College in Austin, TX.

9 Complete program listing

A complete listing of the program discussed in this module is shown in Listing 6 (p. 10) .

Listing 6: Complete program listing.

```
    /*File Prob08 Copyright 2008 R.G.Baldwin
 *Revised 12/31/08

*******************************************************/
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.JLabel;
import javax.swing.event.ChangeListener;
import javax.swing.event.ChangeEvent;
import java.awt.BorderLayout;
import java.awt.Color;

public class Prob08{
  //DO NOT MODIFY THE CODE IN THIS CLASS DEFINITION.
  public static void main(String[] args){
    new Prob08Runner();
  }//end main method
}//end class Prob08
//=======================================================//

class Prob08Runner extends JFrame{

  private JPanel mainPanel = new JPanel();
  private JPanel titlePanel = new JPanel();
  private JSlider slider = new JSlider();

  private Picture butterfly = new Picture("Prob08.jpg");

  private int butterflyWidth = butterfly.getWidth();
  private int butterflyHeight = butterfly.getHeight();

  private Picture display =
              new Picture(butterflyWidth,butterflyHeight);

  private Pixel pix1;
  private Pixel pix2;
  private Pixel displayPixel;

  private double distance = 0;

  public Prob08Runner(){//constructor
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    slider.setMajorTickSpacing(10);
```

```
    slider.setMinorTickSpacing(5);
    slider.setPaintTicks(true);
    slider.setPaintLabels(true);

    mainPanel.setLayout(new BorderLayout());
    titlePanel.add(new JLabel(
                            "Edge Detection Threshold"));
    mainPanel.add(titlePanel,BorderLayout.NORTH);
    mainPanel.add(slider,BorderLayout.CENTER);

    getContentPane().add(mainPanel);

    setSize(butterflyWidth + 7,97);
    setLocation(0,butterflyHeight + 25);
    setVisible(true);

    //Produce the initial display with a threshold value
    // of 50, which matches the initial position of the
    // pointer on the slider.
    display = edgeDetector(butterfly,50);

    display.show();
    //----------------------------------------------------//
    //Register an anonymous listener object on the slider.
    //Each time the slider fires a ChangeEvent, this event
    // handler calls the edgeDetector method to get a new
    // edge-detected image for which the threshold is the
    // current value of the slider. Then the display is
    // repainted showing the new image.
    slider.addChangeListener(
      new ChangeListener(){
        public void stateChanged(ChangeEvent e){

          display = edgeDetector(
                              butterfly,slider.getValue());
          display.repaint();

        }//end stateChanged
      }//end new ChangeListener
    );//end addChangeListener
    //----------------------------------------------------//
  }//end constructor
  //----------------------------------------------------//
  /*This method performs edge detection on a Picture
   *object by rows and also by columns.
   *All edges that are detected by processing adjacent
   *pixels on a row are marked in red.
   *All edges that are detected by processing adjacent
   *pixels on a column are marked in black.
   *If a pixel is determined to be on an edge using both
   *approaches, it ends up being black.
```

```
  */

  private Picture edgeDetector(
                           Picture picture,int threshold){
    for(int row = 0;row < butterflyHeight - 1;row++){
      for(int col = 0;col < butterflyWidth - 1;col++){
        pix1 = picture.getPixel(col,row);
        displayPixel = display.getPixel(col,row);

        //First process two adjacent pixels on the same
        // row.
        pix2 = picture.getPixel(col + 1,row);

        //Get and save the color distance between the two
        // pixels.
        distance = pix1.colorDistance(pix2.getColor());

        //Compare the color distance to the threshold and
        // set the color of the pixel in the display
        // picture accordingly.
        if(distance > threshold){
          displayPixel.setColor(Color.RED);
        }else{
          displayPixel.setColor(Color.WHITE);
        }//end else

        //Now process two adjacent pixels in the same
        // column using the same approach.
        pix2 = picture.getPixel(col,row + 1);
        distance = pix1.colorDistance(pix2.getColor());
        //Compare the color distance to the threshold and
        // change pixel color accordingly.
        if(distance > threshold){
          displayPixel.setColor(Color.BLACK);
        }//end if

      }//end inner loop
    }//end outer loop

    return display;
  }//end edgeDetector
}//end class Prob08Runner
```

-end-