**Abstract**

The purpose of this experiment was to test the effect of random noise in an image on the effectiveness of an edge detection algorithm. The hypothesis stated that if the amount of noise increases, the effectiveness of the algorithm will decrease because it will not be able to detect the true edges due to the reduced image contrast, which is reflected in the reduced variation in the contrast derivative. The edge detection algorithm was implemented using the Java programming language in the Eclipse development environment. The main procedures in the edge detection algorithm use loops, lists, derivatives, signal to noise ratio and other simple math functions. The data collected in the experiment was taken from calculating the signal to noise ratio after noise was added and the guessed edges were detected. An ANOVA test showed that each experimental group was significantly different. The means of effectiveness for each percent noise were found and graphed to clearly support the hypothesis and explain how the effectiveness of the edge detection algorithm decreased as the noise increased.

**Introduction**

Image processing is the manipulation and analysis of digital images by means of image blurring, image enhancement, color inversion, image filtering, image distortion, edge detection and many more methods (Day, 1998).Edge detection is a process that identifies boundaries in a digital image by detecting the change in brightness; enabling a computer to detect shapes and patterns used in shape recognition. The human mind is capable of differentiating shapes by drawing inferences about distance, direction, volume, and area consequently allowing humans to be able to “guess” the shape by summarizing the information in which they observe. Computers, on the other hand, have to be programmed to be able to “see” the way a human can. Shape recognition is used for games, biomedical and investigative applications, art and design software, and robotics. It is found on radar systems, surveillance and forensics, and is used to create robots more human-like (Levine, 1985). Image processing is implemented in machine vision, which is a process by which a computer-driven device optically senses external objects. From the analysis of the sensed data, the device infers information about objects it senses or “sees”. The device is usually a type of camera connected to a computer that digitizes the images and then analyzes them. Machine vision is used for small tasks such as checking size and shape for automobiles, x-rays, photographs, chromosome slides, and cancer smears ("Machine Vision", 1991).

For this project, an image is imported and processed through a computer. The computer uses image processing, manipulating data in the form of an image, in order for the computer to “read” or interpret the image. An image is usually interpreted as a two-dimensional array of numbers, which resemble the image’s brightness values. Each point, or pixel, has a number according to the color that is shows. A pixel is a number that represents the brightness value of the image at a particular location.

This shows the color values of a circle in matrix form

This shows the color values of a circle in matrix form with noise added

11 27 29 0 23 17 5 20 13 27

1 24 76 154 214 227 140 54 13 19

12 55 231 231 230 244 233 213 66 7

10 144 233 251 242 240 247 241 148 16

7 210 232 229 244 244 250 233 196 28

9 196 253 242 249 224 228 252 218 2

7 156 222 240 240 254 230 230 134 17

21 38 231 223 242 231 238 228 72 0

19 10 84 134 235 211 133 42 15 11

15 22 31 22 5 1 29 18 23 29

1 0 1 0 0 0 0 1 0 0

0 0 67 151 227 225 155 60 1 0

1 67 206 252 253 256 249 209 63 1

0 151 252 256 252 256 256 254 154 0

0 227 253 252 256 256 254 254 222 1

0 225 256 256 256 254 256 256 224 0

0 155 249 256 254 256 254 256 153 0

1 60 209 254 254 256 256 203 64 0

0 1 63 154 222 224 153 64 0 0

0 0 1 0 1 0 0 0 0 0

Another important aspect of this project is noise, which is the independent variable. Image noise is the random variation of brightness or color in images creating the image to look fuzzy and grainy. It affects pattern recognition by making it more difficult for the program to decipher the edges because of the random brightness values (McHugh, 2010). To create noise, a random number is generated from -1 to +1. This generated number is then multiplied by 256, which is the largest color value and represents the blackest black, because the color value should be between -256 and +256. This number is then multiplied by the desired percentage change and is added to the original value of the pixel (Smith, 2008).

For example:

Original color = 100

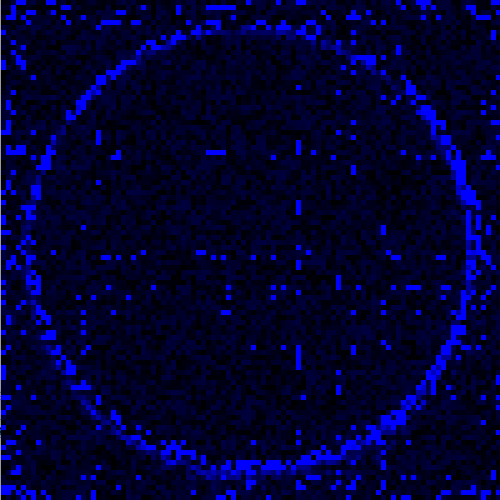
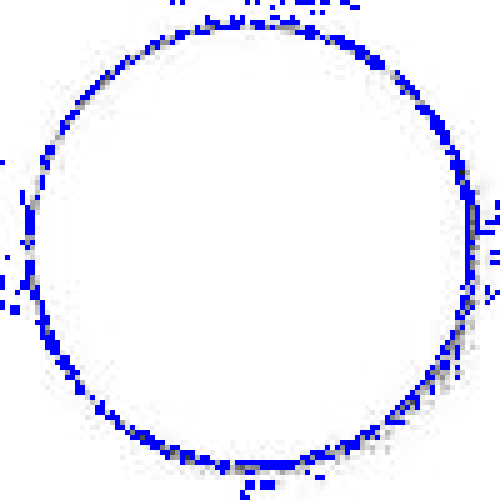
Random generated number = 0.5293719173

Noise Level = 25%

0.5293719173 x 256 = 135.519211

135.519211 x 0.25 = 33.8798028

100 + 33.8798028 = 133.8798028



The image on the left is an image of a circle that has its edge detected with no noise added. The image on the right is the same image of the circle with its edge detected and with noise added.

Edge detection is the process when a computer detects a change in brightness representing the edges in an image or of a particular shape. To detect edges in this experiment, derivatives are found to identify the edges.A derivative is the slope of a line that is tangent to a curve at a single point. The slope is calculated by taking two points close together on the curve and determining the slope of the line that passes through those two points. As the two points on the curve get very close together, the slope of the line becomes numerically equal to the derivative at a single point on the curve.“ To find all of these, a simple algorithm is written to scan the image horizontal and vertical finding the slope between every point that lay next to each other. To find the effectiveness of the algorithm, the signal to noise ratio is calculated. The signal to noise ratio (S/N ratio), a unitless ratio, is the contrast between the signal, which is what is being measured without variation, to the undesired signal known as the noise. More noise in the image results in a lower signal to noise ratio, correspondingly, less noise results in a higher signal to noise ratio (TopBits.com, 2010).

The purpose of this experiment was to test the effect of random noise in an image on the effectiveness of an edge detection algorithm. The independent variable was the noise level on the image; the dependent variable was the effectiveness of the algorithm. The effectiveness was calculated by the signal to noise ratio. The hypothesis reads that if the amount of noise increases, the effectiveness of the algorithm will decrease because it will not be able to detect the shape due to the high and low derivative becoming closer.

**Materials and Methods**

The only thing needed for this experiment is a computer with a Java development system, in this case Eclipse. An edge detection algorithm, written on a MacBook Pro, was used in this experiment. Three shapes were chosen to be tested, a circle, square, and triangle, each 100x100 pixels; refer to Appendix A for the shapes.Different methods are used in the algorithm for the program to work properly. The main methods, which will be explained later on, include: loading the image, adding noise, detecting the edges, and evaluating the effectiveness. For more algorithm details refer to Appendix B. For each shape a percent noise was chosen, the effectiveness was shown and recorded, then repeated five times for each percent noise. The effectiveness values for each percent noise for each shape were then averaged and graphed.

*Load Image*

To load the image a string file is sent to a method that reads the file and displays the image. It then sends the image to save each color value in the form of a two dimensional matrix so it can be manipulated later.

*Add Noise*

Noise is added randomly in the image. The original color can either be increased, to be brighter, or decreased, to be darker. To add noise, a random number is calculated from 0 to 1. 0.5 is then subtracted from this number in order for it to be positive or negative. Then, it is multiplied by 256 to correct the range to all visible colors and multiplied by the desired percentage to find the value that should be added to the pixel. This number is added to the value of a specified integer “color” which equals the original color at a specific point of the image.

*Detect Edges*

This method finds the edges of the image and turns those pixels light blue. To find the edges, it runs through the picture first horizontally and then vertically through each pixel finding the derivation of that pixel. To find the derivation, it grabs the pixels in one line of pixels and calls another method to find the derivatives. Two different methods were created for this. The “XDerivative”, which calculates the derivatives horizontally and the “YDerivative” which calculates the derivatives vertically. To find the derivative, the color value of the current pixel is subtracted by the last color value pixel. This number is then saved in the list “slope”. Each derivative value in the slope list is compared to the max multiplied by the “maxpercent”, which is .80. If the number is larger than the max times the “maxpercent”, then it is an edge. The coordinates that go with the derivative value are then added to a list called “edgepoints”

*Evaluate Effectiveness*

The second to last step is to find the effectiveness of the edge detection when a certain amount of noise is added. To find the effectiveness, the signal to noise ratio is calculated. To do this, the amount of edge pixels without noise, the signal, is divided by the current amount of edge pixels subtracted by the edge pixels without noise, the noise.

To keep the process clean and organized a window is set up for a user to choose a shape and a noise percentage (See Appendix I). The control for each shape will be the amount of edges the program finds with no noise added. Since the noise is random, it should change each time the program runs. The program will run 5 times for each shape for each noise level, finding its signal to noise ratio. These ratios are then averaged in each shape category to find the average signal to noise ratio for each shape in each noise percentage.

**Results**

Average Signal to Noise Ratio According to Noise Percentage for Each Shape

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **0** | **25% Noise** | **50% Noise** | **75% Noise** | **100% Noise** |
| **Circle** | ∞ | 6.34267178207346 | 2.72700963433947 | 0.904223256522195 | 0.528796696429223 |
| **Square** | ∞ | 3.1594824636076 | 3.10534638439943 | 2.74022693757858 | 1.06926545770273 |
| **Triangle** | ∞ | 1.49150683382595 | 0.690224258668032 | 0.526968100010168 | 0.407327687803635 |

After five trials, the mean effectiveness was found for each noise percentage and shape to determine which noise percentage allowed the algorithm to work the best. Each shape at 0% noise had an effectiveness of infinity because there was no noise to make the signal offset.

The graph clearly shows the effectiveness of each noise percentage for each shape and how when more noise is added, the effectiveness decreases.

Summary of ANOVA Test Results for The Effect of Noise on Edge Detection Algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Comparisons | Degrees of Freedom | Alpha Level of Significance | P Value | Significant or Not Significant |
| 25% vs. 50% vs. 75% vs. 100% | 3 | .05 | 2.5459888 X 10^-6 | Significant |

The variability was shown significant from the ANOVA because the P value was much less than the alpha level.

**Conclusion**

The purpose of this experiment was to test the effect of noise on the effectiveness of an edge detection algorithm. The hypothesis was that if the amount of noise increases, then the effectiveness of the algorithm will decrease. With 3 degrees of freedom and a 0.5 alpha level of significance, the differences in the mean effectiveness of each experimental group were statistically significant as shown in the results from the ANOVA test. Each mean effectiveness value was less than the control and continued to decrease as more noise was added for each shape used. There was a larger difference in effectiveness for the circle, than there was for the square or triangle. With only 25% noise, the algorithm was most efficient on the circle, however with 100% noise, the circle and triangle had the least effectiveness. **The research data supported the hypothesis that as noise increased, the effectiveness of the algorithm decreased.**

An increase of the amount of noise affected the algorithm because it increased the noise value while the signal remained the same, so the s/n ratio decreased. With more noise in the image, the change of the color value caused the non-edge points and the edge points to be less differentiated, reducing the contrast so that edges were more difficult to detect. Each time the program ran, a random number was used for the noise at each pixel; these results were averaged over five trials for the estimated s/n ratio. The increase in noise percentage reduced the distinction between edge and non-edge pixels. Adding random noise obscures true edges, and creates false ones. This happens when the random noise between two points causes an increase in color in one point and a decrease in the next, so the algorithm interprets that variation as an edge.

Other researchers have found it difficult to find edges in images with a high noise level. To solve this problem, complex algorithms are created to find the edges more efficiently or to reduce the amount of noise. Additional studies could include shape detection, where the program uses the edges to detect what shape it is. This experiment could be improved by using a more sophisticated algorithm that includes noise smoothing and convolution. Noise smoothing is a method that averages pixel by using surrounding pixels to guess the actual color of the center pixel. This would have improved the effectiveness as the noise increased because averaging cancels out the random noise. Convolution is a more complicated process that is used to find the edges, and would be used instead of simply finding the derivatives.

**Literature Cited**

Day, Bill, and Jonathan Knudsen. "Image Processing with Java 2D." *JavaWorld*. 01 Sept. 1998. Web. 10 Jan. 2011. <http://www.javaworld.com/javaworld/jw-09-1998/jw-09-media.html?page=1>.

Levine, Martin D. Vision in Man and Machine. New York: McGraw-Hill, 1985. Print.

"Machine Vision." How the New Technology Works: A Guide to High-tech Concepts. 2

ed. 18 Sep. 1991. eLibrary Science. Web. 09 Sep 2010.

McHugh, Sean. "Digital Camera Image Noise; Part 2." *Cambridge in Colour*. Web. 18

Oct. 2010. <http://www.cambridgeincolour.com/tutorials/image-noise2.htm>.

Smith, Warren E. , "Image processing," in AccessScience, ©McGraw-Hill Companies,

2008, http://www.accessscience.com

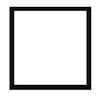
TopBits.com. "Signal to Noise Ratio." 2010. Web. 14 Dec. 2010.

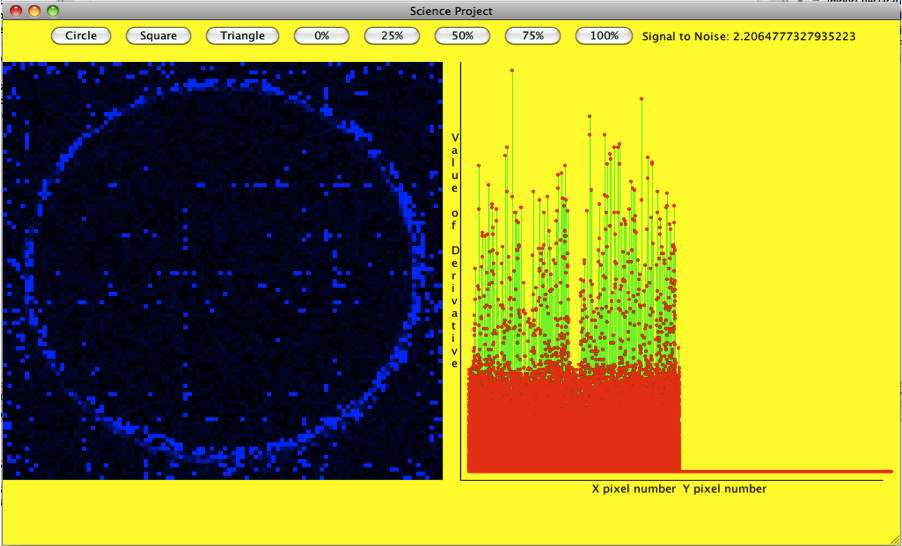
<http://www.tech-faq.com/signal-to-noise-ratio.html>.

**Acknowledgements**

The author would like to thank her father for helping her with the mathematics of the experiment and pushing her to make the algorithm as good as it could get in the time given. The author would also like to thank Mr. Bui, her Advanced Topics in IT teacher, for helping her with the Java techniques and allowing her to work on the experiment in his class.

**Appendix A**





Noise

Signal

This is an image of the window display showing buttons for choosing shape and noise percentage. It shows the shame with noise and edge detection as well as the value of the derivatives and the signal to noise ratio.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Shape** | **Trial** | **%Noise** | **SNR** | **Shape** | **Trial** | **%Noise** | **SNR** |
| **Circle** | C | 0 | ∞ | **Square** | C | 0 | ∞ |
| **Circle** | 1 | 25 | 4.2578125 | **Square** | 1 | 25 | 3.6989247311828000 |
| **Circle** | 2 | 25 | 5.923913043478260 | **Square** | 2 | 25 | 2.6259541984732800 |
| **Circle** | 3 | 25 | 6.337209302325580 | **Square** | 3 | 25 | 2.9783549783549800 |
| **Circle** | 4 | 25 | 9.39655172413793 | **Square** | 4 | 25 | 3.1705069124424000 |
| **Circle** | 5 | 25 | 5.797872340425530 | **Square** | 5 | 25 | 3.323671497584540 |
| **AVERAGE** |  |  | 6.34267178207346 | **AVERAGE** |  |  | 3.1594824636076 |
| **Circle** | 1 | 50 | 1.8664383561643800 | **Square** | 1 | 50 | 2.8666666666666700 |
| **Circle** | 2 | 50 | 3.40625 | **Square** | 2 | 50 | 2.991304347826090 |
| **Circle** | 3 | 50 | 3.1142857142857100 | **Square** | 3 | 50 | 2.8666666666666700 |
| **Circle** | 4 | 50 | 2.3491379310344800 | **Square** | 4 | 50 | 3.2 |
| **Circle** | 5 | 50 | 2.898936170212770 | **Square** | 5 | 50 | 3.602094240837700 |
| **AVERAGE** |  |  | 2.72700963433947 | **AVERAGE** |  |  | 3.10534638439943 |
| **Circle** | 1 | 75 | 0.9511343804537520 | **Square** | 1 | 75 | 2.8786610878661100 |
| **Circle** | 2 | 75 | 0.8346094946401220 | **Square** | 2 | 75 | 3.0442477876106200 |
| **Circle** | 3 | 75 | 0.9628975265017670 | **Square** | 3 | 75 | 2.5864661654135300 |
| **Circle** | 4 | 75 | 0.8790322580645160 | **Square** | 4 | 75 | 2.8907563025210100 |
| **Circle** | 5 | 75 | 0.8934426229508200 | **Square** | 5 | 75 | 2.301003344481610 |
| **AVERAGE** |  |  | 0.904223256522195 | **AVERAGE** |  |  | 2.74022693757858 |
| **Circle** | 1 | 100 | 0.4972627737226280 | **Square** | 1 | 100 | 1.0766823161189400 |
| **Circle** | 2 | 100 | 0.5444555444555440 | **Square** | 2 | 100 | 1.2531876138433500 |
| **Circle** | 3 | 100 | 0.5126999059266230 | **Square** | 3 | 100 | 1.1505016722408000 |
| **Circle** | 4 | 100 | 0.5079217148182670 | **Square** | 4 | 100 | 0.9234899328859060 |
| **Circle** | 5 | 100 | 0.5816435432230520 | **Square** | 5 | 100 | 0.9424657534246580 |
| **AVERAGE** |  |  | 0.528796696429223 | **AVERAGE** |  |  | 1.06926545770273 |

|  |  |  |  |
| --- | --- | --- | --- |
| **Shape** | **Trial** | **%Noise** | **SNR** |
| **Triangle** | C | 0 | ∞ |
| **Triangle** | 1 | 25 | 1.6015831134564600 |
| **Triangle** | 2 | 25 | 1.5524296675191800 |
| **Triangle** | 3 | 25 | 1.5251256281407000 |
| **Triangle** | 4 | 25 | 1.4050925925925900 |
| **Triangle** | 5 | 25 | 1.3733031674208100 |
| **AVERAGE** |  |  | 1.49150683382595 |
| **Triangle** | 1 | 50 | 0.6692392502756340 |
| **Triangle** | 2 | 50 | 0.7082847141190200 |
| **Triangle** | 3 | 50 | 0.7183431952662720 |
| **Triangle** | 4 | 50 | 0.640295358649789 |
| **Triangle** | 5 | 50 | 0.7149587750294460 |
| **AVERAGE** |  |  | 0.690224258668032 |
| **Triangle** | 1 | 75 | 0.4967266775777410 |
| **Triangle** | 2 | 75 | 0.4963205233033520 |
| **Triangle** | 3 | 75 | 0.6112789526686810 |
| **Triangle** | 4 | 75 | 0.5409982174688060 |
| **Triangle** | 5 | 75 | 0.48951612903225800 |
| **AVERAGE** |  |  | 0.526968100010168 |
| **Triangle** | 1 | 100 | 0.40066006600660100 |
| **Triangle** | 2 | 100 | 0.41746905089408500 |
| **Triangle** | 3 | 100 | 0.4118046132971510 |
| **Triangle** | 4 | 100 | 0.40683646112600500 |
| **Triangle** | 5 | 100 | 0.39986824769433500 |
| **AVERAGE** |  |  | 0.407327687803635 |

These tables represent the raw data. They show each shape, the trial number, the percent noise, and the signal to noise ratio.

Appendix B

**import** javax.swing.JFrame;

**public** **class** FinalScienceProject **extends** JFrame {

**public** FinalScienceProject() {

add(**new** FinalScienceProjectPart2()); //ShowPictures

setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

setSize(500\*2+20, 650);

setLocationRelativeTo(**null**);

setTitle("Science Project");

setResizable(**true**);

setVisible(**true**);

}

**public** **static** **void** main(String[] argv){

FinalScienceProject i = **new** FinalScienceProject();

}

}

**import** java.awt.Color;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** java.awt.RenderingHints;

**import** java.awt.Toolkit;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.image.BufferedImage;

**import** java.io.File;

**import** java.util.Random;

**import** javax.imageio.ImageIO;

**import** javax.swing.JButton;

**import** javax.swing.JLabel;

**import** javax.swing.JPanel;

**public** **class** FinalScienceProjectPart2 **extends** JPanel {

BufferedImage inputSourceImage;

String stringSource, status, stringShape;

Random generator = **new** Random();

**int**[][] processedImage;

**int**[] edgepoints, rgbs;

**int**[] totaldata, xDerivativeList,yDerivativeList; //derivatives

**int** wpixel, hpixel, average, numofaverages, numofedges,numNoNoisePoints;

**int** i = 0;

**int** derivativeCounter = 0;

**int** edgecounter = 0;

**double** efficiency;

**double** percentNoise; // Noise value. Total value is usually 256.

**int** edgeColor = 255;

**double** maxpercent = .70;

Boolean noiseOn = **false**;

//SCREEN SETTINGS

**int** titleHeight=20, buttonHeight=30, screenWidth, screenHeight, graphStartX, space=20, mag=5,

buttonWidth = 12, PAD = 10;

//buttons

JButton circle = **new** JButton("Circle");

JButton square = **new** JButton("Square");

JButton triangle = **new** JButton("Triangle");

JButton zero = **new** JButton("0%");

JButton twentyFive = **new** JButton("25%");

JButton fifty = **new** JButton("50%");

JButton seventyFive = **new** JButton("75%");

JButton oneHundred = **new** JButton("100%");

JLabel efficiencyLabel = **new** JLabel();

**public** FinalScienceProjectPart2(){

setFocusable(**true**);

setBackground(Color.*yellow*);

setDoubleBuffered(**true**);

setVisible(**true**);

generateButtons();

setInputImage("/Users/kyla/Desktop/SHAPES/blank.jpg",percentNoise);

setStringSource("/Users/kyla/Desktop/SHAPES/blank.jpg");

stringShape = "Circle";

}

**public** **void** setInputImage(String file, **double** n){

**try** {

inputSourceImage = ImageIO.*read*(**new** File(file));

}

**catch** (Exception e) {

e.printStackTrace();

}

percentNoise = n;

wpixel = inputSourceImage.getWidth(**null**); //num of pixels

hpixel = inputSourceImage.getHeight(**null**); //num of pixels

screenWidth = wpixel\*mag;

screenHeight = (hpixel\*mag + buttonHeight + titleHeight);

setSize(screenWidth\*2+space, screenHeight+100);

graphStartX = screenWidth + space; //maybe \*mag or (screenWidth+space);

i=0;

edgecounter =0;

totaldata = **new** **int**[(wpixel\*hpixel)\*4]; //////////////

edgepoints = **new** **int**[(wpixel\*hpixel)\*4]; ////////////

processedImage = pixelArray();

FindEdge(processedImage);

//printPoints(edgepoints); THIS IS CALLED IN FINDEDGE()

//printDerivative(totaldata); THIS IS CALLED IN FINDEDGE()

// System.out.println("totaldatalength: " + totaldata.length);

// System.out.println("edgepoints: " + edgepoints.length);

// System.out.println("w: " + wpixel);

// System.out.println("h: " + hpixel);

efficiency = findEfficiency();

status = "Signal to Noise: " + efficiency;

efficiencyLabel.setText(status);

System.*out*.println("E:" + efficiency);

}

**public** **void** setStringSource(String s){

stringSource = s;

}

**public** **void** setNoiseOn(**boolean** o){

noiseOn = o;

}

**private** **void** generateButtons(){

circle.setBounds(buttonWidth, screenHeight, buttonHeight, buttonHeight);

circle.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent event) {

stringShape = "Circle";

System.*out*.println(stringShape + " w/ " + percentNoise + "%");

setInputImage("/Users/kyla/Desktop/SHAPES/circle\_jpg.jpg",percentNoise);

setStringSource("/Users/kyla/Desktop/SHAPES/circle\_jpg.jpg");

}

});

square.setBounds(buttonWidth\*2, screenHeight, buttonWidth, buttonHeight);

square.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent event) {

stringShape = "Square";

System.*out*.println(stringShape + " w/ " + percentNoise + "%");

setInputImage("/Users/kyla/Desktop/SHAPES/square7.jpg",percentNoise);

setStringSource("/Users/kyla/Desktop/SHAPES/square7.jpg");

}

});

triangle.setBounds(buttonWidth\*3, screenHeight, buttonWidth, buttonHeight);

triangle.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent event) {

stringShape = "Triangle";

System.*out*.println(stringShape + " w/ " + percentNoise + "%");

setInputImage("/Users/kyla/Desktop/SHAPES/goodtriangle.jpg",percentNoise);

setStringSource("/Users/kyla/Desktop/SHAPES/goodtriangle.jpg");

}

});

zero.setBounds(buttonWidth\*5, screenHeight, buttonWidth, buttonHeight);

zero.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent event) {

System.*out*.println(stringShape + " w/ " +" 0%");

setInputImage(stringSource,0);

setNoiseOn(**false**);

}

});

twentyFive.setBounds(buttonWidth\*6, screenHeight, buttonWidth, buttonHeight);

twentyFive.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent event) {

System.*out*.println(stringShape + "w/" + "25%");

setInputImage(stringSource,25);

setNoiseOn(**true**);

}

});

fifty.setBounds(buttonWidth\*7, screenHeight, buttonWidth, buttonHeight);

fifty.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent event) {

System.*out*.println(stringShape + " w/ " +" 50%");

setInputImage(stringSource,50);

setNoiseOn(**true**);

}

});

seventyFive.setBounds(buttonWidth\*8, screenHeight, buttonWidth, buttonHeight);

seventyFive.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent event) {

System.*out*.println(stringShape + " w/ " +" 75%");

setInputImage(stringSource,75);

setNoiseOn(**true**);

}

});

oneHundred.setBounds(buttonWidth\*9, screenHeight, buttonWidth, buttonHeight);

oneHundred.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent event) {

System.*out*.println(stringShape + " w/ " +" 100%");

setInputImage(stringSource,100);

setNoiseOn(**true**);

}

});

add(circle);

add(square);

add(triangle);

add(zero);

add(twentyFive);

add(fifty);

add(seventyFive);

add(oneHundred);

add(efficiencyLabel);

}

**public** **int**[][] pixelArray() {

rgbs = **new** **int**[wpixel\*hpixel];

// System.out.println("rgbs length: " + rgbs.length);

**int**[][] pixelArrayData = **new** **int**[wpixel][hpixel];

**int** colorcounter = 0;

/\* 24 bit color

16777216 = 2^24

16 bit color

65536 = 2^16

(2^24)/(2^24) = 1bit --> 0 or 1

(2^24) / (2^16) = 2^8bits --> 0-225

\*/

inputSourceImage.getRGB(0, 0, wpixel, hpixel, rgbs, 0, wpixel);

**for** (**int** r=0;r<wpixel;r++) {

**for** (**int** c=0;c<hpixel;c++) {

pixelArrayData[r][c] = -(rgbs[colorcounter])/65536; //greyscale

colorcounter++;

// System.out.print(pixelArrayData[r][c]+ " ");

}

// System.out.println();

}

**if** (noiseOn){

pixelArrayData = addNoise(pixelArrayData,percentNoise);

}

/\* System.out.println("WITH NOISE");

for (int r=0;r<wpixel;r++){

for (int c=0;c<hpixel;c++){

// System.out.print(pixelArrayData[r][c] + " ");

}

// System.out.println();

}\*/

**return** pixelArrayData;

}

**public** **int**[][] addNoise(**int**[][] data, **double** percent){

**double** random;

**int** color;

**int**[][] noiseImage = data;

**for** (**int** y=0;y<noiseImage.length;y++){

**for**(**int** x=0;x<noiseImage.length;x++){

random = generator.nextDouble() - 0.5; //256\*(percent/100)

random \*= 256\*(percent/100);

color = noiseImage[x][y];

**if** (color + (**int**)random > 256){

color -= (**int**) random;

noiseImage[x][y] = color;

inputSourceImage.setRGB(x,y,color);

}

**else** **if** (color + (**int**)random < 0){

color -= (**int**)random;

noiseImage[x][y] = color;

inputSourceImage.setRGB(x,y,color);

}

**else**{

color += (**int**)random;

noiseImage[x][y] = color;

inputSourceImage.setRGB(x, y, color);

}

}

}

**return** noiseImage;

}

**public** **void** FindEdge(**int**[][] list){ //list of whole image (may include noise)

// System.out.println("listSizeFindEdge(): " + list.length);

//each list of Z values in one row of Y need a derivative.

// System.out.println("YDERIVATIVE...");

**for** (**int** y = 0; y<list.length;y++){

yDerivativeList = **new** **int**[list.length];

**for** (**int** x=0;x<list.length;x++){

yDerivativeList[derivativeCounter] = list[y][x];

derivativeCounter++;

}

YDerivative(yDerivativeList,y);

derivativeCounter = 0;

}

// System.out.println("XDERIVATIVE...");

derivativeCounter = 0;

**for** (**int** x = 0; x<list.length;x++){

xDerivativeList = **new** **int**[list.length];

**for** (**int** y=0;y<list.length;y++){

xDerivativeList[derivativeCounter] = list[y][x];

derivativeCounter++;

}

XDerivative(xDerivativeList,x);

derivativeCounter = 0;

}

**if** (noiseOn == **false**){

numNoNoisePoints = edgecounter;

}

// System.out.println("numNoNoisePoints: " + numNoNoisePoints);

// System.out.println("edgecounter: " + edgecounter);

printPoints(edgepoints);

printDerivative(totaldata);

}

**public** **void** YDerivative(**int**[] list, **int** yvalue){

**int**[] slope = **new** **int**[list.length];

**int** counter = 0;

**int** sum = 0;

// System.out.println();

**for**(**int** x=0;x<list.length;x++){

// System.out.print(list[x] + ", ");

}

// System.out.println();

**for** (**int** y=1;y<list.length;y++){

slope[counter] = Math.*abs*(list[y]-list[y-1]);

counter++;

}

/\* System.out.println("SLOPE...");

for (int i =0; i<list.length;i++){

// System.out.print(slope[i]+ ", ");

}\*/

**for** (**int** k=0;k<list.length;k++){

sum += slope[k];

}

average += Math.*abs*(sum/slope.length);

numofaverages++;

// System.out.print("AverageD " + sum/slope.length);

**int** max = findMax(slope);

// System.out.println(" max: " + max);

counter = 0;

**for** (**int** j=0;j<slope.length;j++){

**if** (slope[j] > max\*maxpercent){

inputSourceImage.setRGB(j, yvalue, edgeColor);

edgepoints[edgecounter] = j;

edgecounter++;

edgepoints[edgecounter] = yvalue;

edgecounter++;

}

}

addToTotalData(slope);

}

**public** **void** XDerivative(**int**[] list, **int** xvalue){

**int**[] slope = **new** **int**[list.length];

**int** counter = 0;

**int** sum = 0;

// System.out.println();

**for**(**int** x=0;x<list.length;x++){

// System.out.print(list[x] + ", ");

}

// System.out.println();

**for** (**int** y=1;y<list.length;y++){

slope[counter] = Math.*abs*(list[y]-list[y-1]);

counter++;

}

/\* System.out.println("SLOPE...");

for (int i =0; i<list.length;i++){

System.out.print(slope[i]+ ", ");

}\*/

**for** (**int** k=0;k<list.length;k++){

sum += slope[k];

}

average += Math.*abs*(sum/slope.length);

numofaverages++;

// System.out.print("AverageD " + sum/slope.length);

**int** max = findMax(slope);

// System.out.println("slopemax: " + max);

counter = 0;

**for** (**int** j=0;j<slope.length;j++){

**if** (slope[j] > max\*maxpercent){

inputSourceImage.setRGB(xvalue, j, edgeColor);

edgepoints[edgecounter] = xvalue;

edgecounter++;

edgepoints[edgecounter] = j;

edgecounter++;

}

}

addToTotalData(slope);

}

**public** **void** addToTotalData(**int**[] data){

**for** (**int** x=0;x<data.length;x++){

totaldata[i] = data[x];

i++;

}

}

**public** **int** findMax(**int**[] list){

**int** max;

max = list[0];

**for** (**int** x=0;x<list.length;x++){

**if** (list[x] > max){

max = list[x];

}

}

**return** max;

}

**public** **int** findMin(**int**[] list){

**int** min;

min = list[0];

**for** (**int** x=0;x<list.length;x++){

**if** (list[x] < min){

min = list[x];

}

}

**return** min;

}

**public** **int** findAverage(){

**int** a = average/numofaverages;

**return** a;

}

**public** **void** printPoints(**int**[] list){

/\*System.out.println("PRINTEDGEPOINTS");

for (int x=0;x<list.length;x+=2){

System.out.print("(" + list[x] + "," + list[x+1] + ")");

}\*/

}

**public** **void** printDerivative(**int**[] d){

/\* System.out.println();

System.out.println("PRINTDERIVATIVE");

for (int x = 0;x<d.length;x++){

System.out.print(d[x] + ", ");

}\*/

}

**public** **double** findEfficiency(){

**double** efficiency;

// System.out.println();

// System.out.println("max: " + max);

// System.out.println("average: " + totalAverage);

// efficiency = (max-totalAverage)/totalAverage;

efficiency =(**double**) numNoNoisePoints/(edgecounter - numNoNoisePoints);

**return** efficiency;

}

**public** **void** paint(Graphics g){

**super**.paint(g);

Graphics2D g2 = (Graphics2D)g;

g.drawImage(inputSourceImage, 0, buttonHeight+titleHeight, screenWidth, screenHeight-buttonHeight-titleHeight, **this**);

g2.setRenderingHint(RenderingHints.*KEY\_ANTIALIASING*,

RenderingHints.*VALUE\_ANTIALIAS\_ON*);

// Draw Y axis.

g.drawLine(graphStartX,buttonHeight+titleHeight,graphStartX, screenHeight);

//draw X axis

g.drawLine(graphStartX,screenHeight,screenWidth\*2, screenHeight);

// Draw labels.

// Y label.

**int** xCounter = 115;

String[] s = "Value of Derivative".split("");

**for** ( **int** x=0; x<s.length; x++) {

xCounter +=15;

g.drawString(s[x], graphStartX-10, xCounter);

}

// g.drawString(s, graphWStart, i);

// X label.

String s2 = "X pixel number Y pixel number";

g.drawString(s2, graphStartX+150, screenHeight+15);

// Draw lines.

// The space between values along the x axis.

// System.out.println("totaldatalength: " + totaldata.length);

**double** xInc = (**double**)(screenWidth - 2\*PAD)/(totaldata.length-1);

/////// double xInc = (double)(screenWidth-PAD-space)/(totaldata.length-1);

// System.out.println("xInc: " + xInc);

// Scale factor for y/data values.

**double** scale = (**double**)(((screenHeight-buttonHeight-titleHeight)) - 2\*PAD)/findMax(totaldata);

//drawLines

g.setColor(Color.*green*);

// g2.setPaint(Color.green.darker());

**for**(**int** i = 0; i < totaldata.length-1; i++) {

**double** x1 = graphStartX+ PAD + i\*xInc;

**double** y1 = (screenHeight) -PAD- scale\*totaldata[i];

**double** x2 = graphStartX +PAD+ (i+1)\*xInc;

**double** y2 = (screenHeight)- PAD-scale\*totaldata[i+1];

g.drawLine((**int**) x1, (**int**)y1, (**int**)x2, (**int**)y2);

// g2.draw(new Line2D.Double(x1, y1, x2, y2));

}

// Mark data points.

g.setColor(Color.*red*);

// g2.setPaint(Color.red);

**for**(**int** i = 0; i < totaldata.length; i++) {

**double** x = graphStartX +PAD+ i\*xInc;

**double** y = (screenHeight)-PAD- scale\*totaldata[i];

g.fillOval((**int**) x-2,(**int**) y-2, 4, 4);

// g2.fill(new Ellipse2D.Double(x-2, y-2, 4, 4));

}

Toolkit.*getDefaultToolkit*().sync();

g.dispose();

requestFocusInWindow(**true**);

}

**public** **void** actionPerformed(ActionEvent e) {

FindEdge(processedImage);

add(efficiencyLabel);

repaint();

}

}