Trabalho de Processamento de Imagens(PDI)

Profª Danielle D. B. Colturato.

Integrantes do grupo:

Ailton Borges dos Santos RA: C47GAF-0
Felipe Consentino Capelli RA: C42894-9
Vitor Sumiyoshi Lima Mukai RA: C0733F-8
Wagner Rafael Giarini RA: B58FCJ-1

Introdução

O sentido da visão humana, talvez seja nosso sentido mais importante para reconhecer o mundo a nossa volta, ele nos possibilita ver o mundo de uma forma que possamos processar a informação e tirar conclusões baseado em observações. Essa capacidade biológica de processar informações nos motivou aos estudos da computação gráfica e processamento de imagens, de modo que possa ser usado para "expandir" a nossa capacidade de visão e abstração da informação. Trabalhar com imagens digitais também nos possibilitou um grande avanço no tratamento de imagem, por ser processada por meios eletrônicos, tornou mais fácil a forma como manipulamos imagens sem se preocupar com reações químicas acontecendo na mesma, com isso há diversas formas possíveis de melhora-la afim descobrir informações ocultas com a aplicação de filtros de processamento e tratamento, ao contrário da analógica que nos traria limitações começando com o seu tempo de vida.

O estudo de processamento de imagem está presente em diversas áreas, sua aplicação inicia-se desde objetivos simples, como jogos de computador, publicidades, mercado fotográfico e até sistemas mais críticos como representação de modernas aeronaves espaciais ou área medica com intuito de fazer diagnósticos em órgãos para análise de uma possível doença.

O Processamento Digital de Imagens (PDI) não é um trabalho simples, na realidade envolve um conjunto de tarefas. Tudo se inicia com a captura de uma imagem, a qual normalmente, corresponde à captura dos reflexos da iluminação que é refletida na superfície dos objetos, realizada através de um sistema de aquisição. Após a captura por um processo de digitalização, uma imagem precisa ser representada de forma compatível para tratamento computacional. Imagens podem ser representadas em duas ou mais dimensões. O primeiro passo no processamento é conhecido como pré-processamento, o qual envolve passos como a filtragem de ruídos introduzidos pelos sensores e a correção de distorções geométricas causadas pelo sensor. Após isso os objetos precisam ser distinguidos do plano de fundo, distinguindo o que está à frente e atrás, feito isso através de um processo chamado de segmentação, nesse processo são usados modelos matemáticos para a identificação das bordas e definição do que faz parte do objeto e não faz parte. O mesmo processo se aplica a tratamentos com objetivo de realizar correções, são usados algoritmos para reconhecimento de granulações, distorções ou para realce de cores.

Computação gráfica x Processamento de Imagens

O conhecimento em ambas as áreas tem crescido consideravelmente, o que tem permitido a resolução de problemas cada vez mais complexos. Numa observação simplificada, CG busca imagens fotos-realísticas de cenas tridimensionais geradas por computador, enquanto PDI tenta reconstruir uma cena tridimensional a partir de uma imagem real, obtida através de uma câmera. Neste sentido, PDI busca um procedimento inverso ao de CG, então uma área depende da outra para a construção de uma imagem digital final.

Descrição do trabalho

O trabalho tem como o objetivo mostrar a utilização dos filtros Gaussiano, Sobel, Canny, escala de cinzas e aplicação da equalização do histograma, através de um programa computacional escrito em Java especialmente para este propósito. Com essa aplicação, é possível ter conhecimentos sobre a função dos filtros aplicados por meio dos resultados obtido dentre as imagens e sua variação de histograma. Foi usado biblioteca opencv de código aberto que também utilizada por empresas como a Google, Microsoft, Intel, IBM, que aplicam em sistemas como Street View, sistemas de segurança em câmeras para detecção de movimento, e principalmente destinada ao aprendizado em suas aplicações.

Resultados

Filtros analisados:

- Gaussiano
- Sobel
- Canny
- Escala de cinzas

Operação:

- Equalização de Histograma

Gaussian (Gaussiano)

O filtro gaussiano, é um tipo de filtro muito usado em suavização de imagens, para remoção de ruídos ou granulações contidas nela, seu comportamento é de um passa-baixa com uma característica de ignorar arestas das imagens, por ele não reconhecer as diferenças de intensidades de cada pixel. Sua aplicação se dá pela definição:

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_\sigma(\|\mathbf{p} - \mathbf{q}\|)\, I_{\mathbf{q}},$$

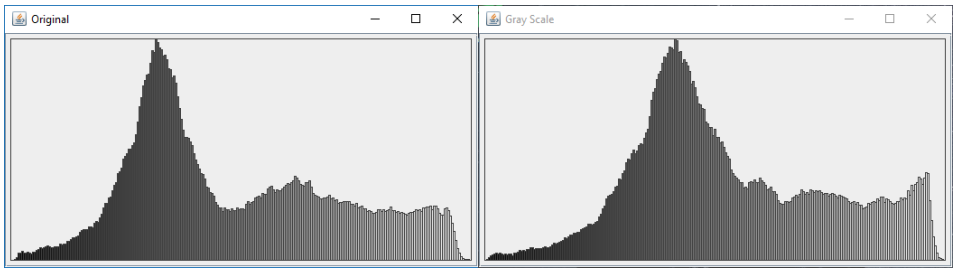
Onde Gσ

$$G_\sigma(x) = \frac{1}{2\pi\,\sigma^2}\ \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

Quanto maior o sigma, maior será a quantidade de pixels vizinhos cuja a suavização será influente.

Abaixo temos o resultado de uma imagem com Sigma = 5

É possível notar como a representação do histograma muda em relação as duas imagens, é feita uma distribuição maior dos tons de cinza em relação a original. Como nas imagens abaixo:



Original Gray Scale

Sobel

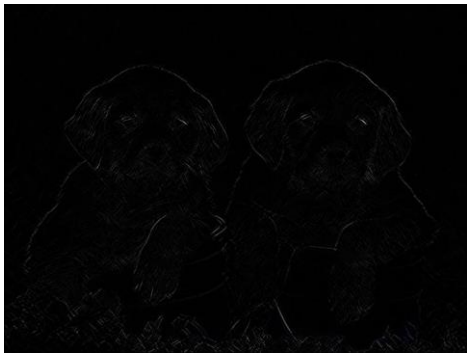
Sobel é um filtro passa-alta, do tipo não linear ou seja, é mais seletivo em relação ais pixels, bordas, linhas e pequenas estruturas nas imagens, então nesse caso ele dá uma grande ênfase nos traços da imagem, dando como resultado final um contorno dos principais objetos que se diferem pela alteração de intensidade do pixel, sendo usado em muitos casos como um “detector de bordas”.

São aplicados duas matrizes como mascaras para a detecção de bordas, as filtragens são feitas no eixo x, y , vertical e horizontal. A máscara é aplicada por duas matrizes

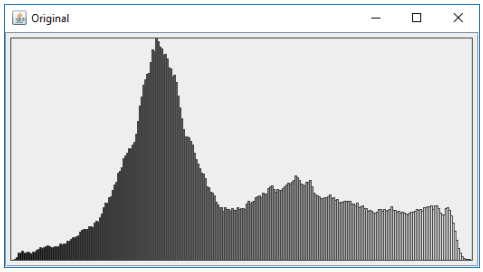
$$\mathbf{G_x} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * \mathbf{A} \quad \text{e} \quad \mathbf{G_y} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \mathbf{A}$$

E operador Sobel dado pela equação:

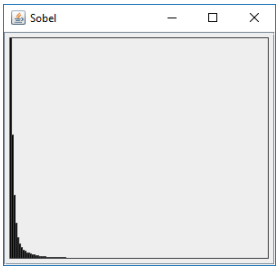
$$\mathbf{G} = \sqrt{\mathbf{G_x}^2 + \mathbf{G_y}^2}$$



Abaixo é possível notar o grande desvio do tom de cinza da original para modificada, o filtro usa poucos tons de cinza, aparentando ter apenas duas tonalidades como na imagem, preto e branco.



Original



Sobel

Canny

Assim como Sobel, Canny também é um filtro passa-alta para detecção de bordas, sua diferença é a aplicação de algoritmo que passa por vários estágios até o resultado final. Ele suaviza o ruído e localiza bordas, combinando um operador diferencial com um filtro Gaussiano. O filtro de Canny é um filtro que usa a primeira derivada de Gauss.

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2}{2\sigma^2}}$$

Gauss

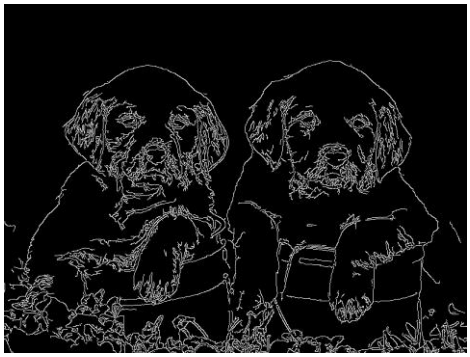
$$G'(x) = \frac{-x}{\sqrt{2\pi\sigma^3}} e^{\frac{-x^2}{2\sigma^2}}$$

primeira derivada de Gauss

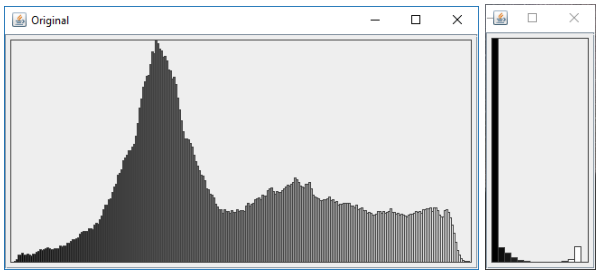
Além disso, é usado uma convolução para combinar o filtro com o operador da derivada I de gauss, é bem custosa em relação a outros filtros, essa fórmula pode ser expressada da seguinte maneira:

$$G(x,y) = \frac{1}{2\pi\sigma_x\sigma_y} e^{\frac{-x^2}{2\sigma_x^2} + \frac{-y^2}{2\sigma_y^2}}$$

convolução



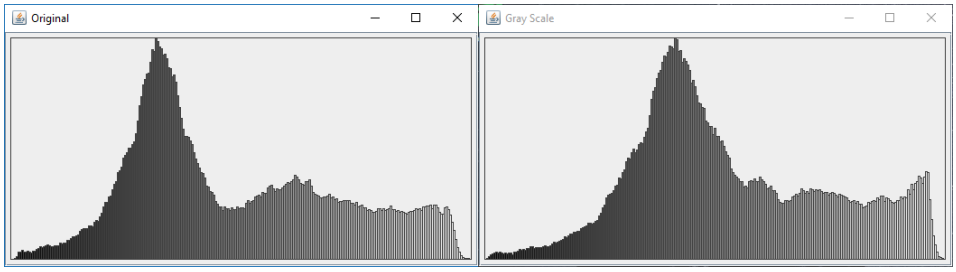
Abaixo é possível ver como o gráfico se assemelha ao de Sobel, por a quantidade e intensidade dos tons terem grande ênfase no espectro mais próximo de preto.



Gray Scale (Escala de Cinzas)

Em imagem digital, uma amostra de tons de cinza é ter os pixels com a mesma intensidade no RGB ex: #808080, essas imagens em nível de cinza têm enorme importância no PDI porque apesar de conterem apenas 1 tonalidade, elas conseguem preservar informações suficientes na imagem para obter resultados em uma grande quantidade de aplicações, como o uso da segmentação.

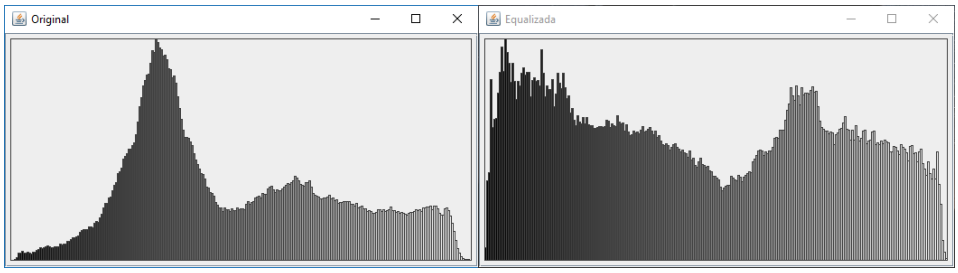
Abaixo está o histograma das duas imagens acima, há uma maior distribuição dos tons de cinza mais claros para definir cores em tons de cinza e relação a imagem original.



Equalização do Histograma

Na equalização de histograma, tem como objetivo uma maior distribuição na acentuação de cores, para que detalhes não exibidos anteriormente sejam destacados após a equalização. Na imagem abaixo, existe pouca diferença na acentuação de brilho, provável que já esteja equalizada, mas na segunda imagem, houve uma mudança na intensidade de cores, o que tornou a imagem com tons mais fortes dos que já existiam, sem dar muita ênfase no brilho e com maior nitidez em relação a anterior.

Abaixo é possível notar essa acentuação nos tons mais claros, a imagem



Conclusão

O efeito da aplicação dos filtros, variam de acordo com a imagem, porém a maioria dos testes realizado pelo programa com outras imagens, foi possível notar alguma mudança significativa julgando o objetivo do filtro aplicado.

Referencias:

LEONARDO VIDAL BATISTA. Introdução ao Processamento Digital de Imagens.2005.
http://users.ecs.soton.ac.uk/msn/book/new_demo/sobel/
https://stackoverflow.com/questions/27086120/convert-opencv-mat-object-to-bufferedimage
http://www.inf.pucrs.br/~pinho/CG/Aulas/Intro/intro.htm
http://lvelho.impa.br/ip09/demos/jbu/filtros.html
http://www2.ic.uff.br/~aconci/G-LoG.PDF
https://pt.wikipedia.org/wiki/Ru%C3%ADdo_gaussiano
https://webserver2.tecgraf.puc-rio.br/~mgattass/fcg/trb13/RaquelGuilhon/trabalho1.html
http://www.dpi.inpe.br/~carlos/Academicos/Cursos/Pdi/pdi_filtros.htm
https://pt.wikipedia.org/wiki/Filtro_Sobel
https://rhaylsonsilva.wordpress.com/2012/03/02/processamento-de-imagens-deteccao-de-bordas-de-imagens/
http://erikasarti.net/html/tabela-cores/
https://fperrotti.wikispaces.com/Imagens+em+n%C3%ADveis+de+cinza

Programa

```
import java.awt.Color;

import java.awt.image.BufferedImage;

import java.io.ByteArrayInputStream;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import org.opencv.core.Core;

import org.opencv.core.Mat;

import org.opencv.core.MatOfByte;

import org.opencv.core.Size;

import org.opencv.imgcodecs.Imgcodecs;

import org.opencv.imgproc.Imgproc;

import java.util.ArrayList;

import java.util.List;


public class Filtros {

        public static final Filtros f = new Filtros();

        public static void main(String args[]) throws Exception {

                Mat  imagemM;
```

```
        BufferedImage imagemB;
        //carrega imagem
        imagemM = Filtros.carregaToMat();
        imagemB = Filtros.carregaToBuf();

        //aplica filtros
        Filtros.gaussiano(imagemM);
        Filtros.sobel(imagemM);
        Filtros.canny(imagemM);
        Filtros.equaHistograma(imagemM);
        Filtros.original(imagemB);
        Filtros.grayScale(imagemB);
}

public static Mat carregaToMat(){
        System.loadLibrary(Core.NATIVE_LIBRARY_NAME);
        String file ="imagens/original/exe1.jpg";
        Mat img = Imgcodecs.imread(file);
        return img;
}
public static void original(BufferedImage imag){
        //histograma
        JanelaHistograma.setTitulo("Original");
        new JanelaHistograma(imag);
}
public static void gaussiano(Mat src) throws Exception{
        // cria uma matriz vazia para guardar o resultado
        Mat nova = new Mat();
        //aplica gaussiano
        Imgproc.GaussianBlur(src, nova, new Size(45, 45), 5);
        // escreve nova imagem
```

```
                System.out.println("gaussiano aplicado");
                //converte para buff
                BufferedImage img;
                img = Filtros.mat2Img(nova);
                //salva
                try {
                        ImageIO.write(img, "jpg", new File("imagens/pos-
proc/Gaussiano.jpg"));
                } catch (IOException e){
                        System.out.println("Erro de armazenamento!"+e.getMessage());
                }
                //histograma
                JanelaHistograma.setTitulo("Gaussiano");
                new JanelaHistograma(img);
        }

        public static void sobel(Mat src) throws Exception{
                Mat nova = new Mat();
                Imgproc.Sobel(src, nova, -1, 1, 1);
                //Imgcodecs.imwrite("imagens/pos-proc/Sobel.jpg", nova);
                //converte para buff
                BufferedImage img;
                img = Filtros.mat2Img(nova);
                //salva
                try {
                        ImageIO.write(img, "jpg", new File("imagens/pos-proc/Sobel.jpg"));
                } catch (IOException e){
                        System.out.println("Erro de armazenamento!"+e.getMessage());
                }
                //histograma
                JanelaHistograma.setTitulo("Sobel");
```

```
            new JanelaHistograma(img);
            System.out.println("sobel aplicado");
    }
    public static void canny(Mat src) throws Exception{
            Mat nova = new Mat();
            //Imgproc.cvtColor(img, nova, Imgproc.COLOR_BGR2GRAY);
            // Detecting as bordas da imagem
            Imgproc.Canny(src, nova, 60, 60*3);
            //converte para buff
            BufferedImage img;
            img = Filtros.mat2Img(nova);
            //salva
            try {
                    ImageIO.write(img, "jpg", new File("imagens/pos-proc/Canny.jpg"));
            } catch (IOException e){
                    System.out.println("Erro de armazenamento!"+e.getMessage());
            }
            //histograma
            JanelaHistograma.setTitulo("Canny");
            new JanelaHistograma(img);
    }
    public static void equaHistograma(Mat src) throws Exception{
            Mat nova = new Mat();
            src.copyTo(nova);

            // Applying blur
            Imgproc.blur(nova, nova, new Size(3, 3));

            // Applying color
            Imgproc.cvtColor(nova, nova, Imgproc.COLOR_BGR2YCrCb);
            List<Mat> channels = new ArrayList<Mat>();
```

```
                // Splitting the channels
                Core.split(nova, channels);

                // Equalizing the histogram of the image
                Imgproc.equalizeHist(channels.get(0), channels.get(0));
                Core.merge(channels, nova);
                Imgproc.cvtColor(nova, nova, Imgproc.COLOR_YCrCb2BGR);

                Mat gray = new Mat();
                Imgproc.cvtColor(nova, gray, Imgproc.COLOR_BGR2GRAY);
                Mat grayOrig = new Mat();
                Imgproc.cvtColor(src, grayOrig, Imgproc.COLOR_BGR2GRAY);

                //Imgcodecs.imwrite("imagens/pos-proc/Equalizada.jpg", nova);
                //converte para buff
                BufferedImage img;
                img = Filtros.mat2Img(nova);
                //salva
                try {
                        ImageIO.write(img, "jpg", new File("imagens/pos-
proc/Equalizada.jpg"));
                } catch (IOException e){
                        System.out.println("Erro de armazenamento!"+e.getMessage());
                }
                //histograma
                JanelaHistograma.setTitulo("Equalizada");
                new JanelaHistograma(img);
                System.out.println("equalização aplicado");
        }
        public static void grayScale(BufferedImage imag){
```

```
                BufferedImage img = imag;
                //aplica filtro gray
                for(int x = 0; x < img.getWidth(); x++){
                        for(int y = 0; y < img.getHeight(); y++){
                                int rgb = img.getRGB(x, y);//recebe o rgb de 1 pixel
                                Color c = new Color(rgb);//criar cor opaca
                                //aplicação do comp. luma R299 G587 B114
                                int grey = (int) (0.299 * c.getRed() + 0.587 * c.getGreen() +
0.114*c.getBlue());
                                Color c2 = new Color(grey, grey, grey);
                                img.setRGB(x, y, c2.getRGB());
                        }
                }
                //salva imagem
                try {
                        ImageIO.write(img, "jpg", new File("imagens/pos-proc/Gray.jpg"));
                } catch (IOException e){
                        System.out.println("Erro de armazenamento!"+e.getMessage());
                }
                //histograma
                JanelaHistograma.setTitulo("Gray Scale");
                new JanelaHistograma(img);
        }
        public static BufferedImage carregaToBuf(){
                BufferedImage imag;
                imag = null;
                //carrega imagem
                try{
                        imag = ImageIO.read(new File("imagens/original/exe1.jpg"));
                } catch (IOException e) {
                        System.out.println("Erro de carregamento!"+e.getMessage());
```

```
                }
                //f.mat2Img(in)
                return imag;
        }
//conversão de tipos Mat para Buffredimage
        public static BufferedImage mat2Img(Mat in)throws Exception {
                MatOfByte mob=new MatOfByte();
                Imgcodecs.imencode(".jpg", in, mob);
                byte ba[]=mob.toArray();

                BufferedImage bi=ImageIO.read(new ByteArrayInputStream(ba));
                return bi;
        }
}

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.util.Map;
import java.util.TreeMap;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;

@SuppressWarnings("serial")
public class JanelaHistograma extends JFrame {
```

```
public static String titulo;

public static String setTitulo(String titu){

        titulo=titu;
        return titulo;
}
public static String getTitulo(){
        return titulo;
}

// valores do mapa do histograma
private Map<Integer, Integer> mapa;

public JanelaHistograma(BufferedImage imagem)
{
        // carrega os dados da imagem para gerar o histograma
        carregarDados(imagem);
        // cria e exibe a janela e seus componentes com o mapa do histograma
        criarJanela();
}

// criacao basica da janela
private void criarJanela()
{
        setTitle(JanelaHistograma.getTitulo());
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setLayout(new BorderLayout());
        add(new JScrollPane(new Histograma(mapa)));
        pack();
        setLocationRelativeTo(null);
```

```
            setVisible(true);
        }

        // nao usado
        // recarrega os dados do histograma de acordo com a imagem atual
/*      private void recarregarDados()
        {
            carregarDados(imagem);
            repaint();
        }*/

        // converter o valor int do rgb para escala de cinza
        private int converterParaCinza(BufferedImage imagem, int x, int y)
        {
            int rgb = imagem.getRGB(x, y);
            int r = (rgb >> 16) & 0xFF;                 // vermelho
            int g = (rgb >> 8) &  0xFF;                 // verde
            int b = (rgb &              0xFF); // azul

            int cinza = (r + g + b) / 3;          // gerar nivel de cinza
            return cinza;
        }

        private void carregarDados(BufferedImage imagem)
        {

            int largura = imagem.getWidth();
            int altura = imagem.getHeight();

            int[][] dados = new int[largura][altura];
```

```
// converter cada pra pixel para escal de cinza (preto e branco)
for (int y = 0; y < altura; y++)
{
	for (int x = 0; x < largura; x++)
	{
		dados[x][y] = converterParaCinza(imagem, x, y);
	}
}

mapa = new TreeMap<Integer, Integer>();

for (int y = 0; y < dados.length; y++)
{
	for (int x = 0; x < dados[y].length; x++)
	{
		// pega o valor e quantidade de cada pixel da imagem
		int valor = dados[y][x];
		int quantidade = 0;

		if (mapa.containsKey(valor))
		{
			quantidade = mapa.get(valor);
			quantidade++;
		} else
		{
			quantidade = 1;
		}

		// guarda esse valor no mapa
		mapa.put(valor, quantidade);
	}
```

```
        }
    }


    protected class Histograma extends JPanel {

        // parametros do grafico do histograma
        protected static final int LARGURADA_DA_LINHA = 2;
        protected static final int ALTURA_MINIMA = 128;
        protected static final int ALTURA_PADRAO = 256;

        private Map<Integer, Integer> histograma;

        public Histograma(Map<Integer, Integer> mapa)
        {
            // parametros da janela em si
            this.histograma = mapa;
            int largura = (mapa.size() * LARGURADA_DA_LINHA) + 11; // 11 ==
tamanho da borda
            Dimension tamanhoMinimo = new Dimension(largura,
ALTURA_MINIMA);
            Dimension tamanhoPadrao = new Dimension(largura,
ALTURA_PADRAO);
            setMinimumSize(tamanhoMinimo);
            setPreferredSize(tamanhoPadrao);
        }

        protected void paintComponent(Graphics g)
        {
            super.paintComponent(g);
            if (histograma != null)
```

```
                {
                        int xOffset = 5;
                        int yOffset = 5;
                        int largura = getWidth() - 1 - (xOffset * 2);
                        int altura = getHeight() - 1 - (yOffset * 2);
                        Graphics2D g2d = (Graphics2D) g.create();
                        g2d.setColor(Color.DARK_GRAY);
                        g2d.drawRect(xOffset, yOffset, largura, altura);
                        int barWidth = Math.max(LARGURADA_DA_LINHA, (int)
Math.floor((float) largura / (float) histograma.size()));

                        int valorMaximo = 0;
                        for (Integer key : histograma.keySet()) {
                                int valor = histograma.get(key);
                                valorMaximo = Math.max(valorMaximo, valor);
                        }
                        int xPosicao = xOffset;
                        for (Integer key : histograma.keySet()) {
                                int value = histograma.get(key);
                                int alturaBarra = Math.round(((float) value / (float)
valorMaximo) * altura);
                                g2d.setColor(new Color(key, key, key));
                                int yPosicao = altura + yOffset - alturaBarra;

                                Rectangle2D bar = new Rectangle2D.Float(xPosicao,
yPosicao, barWidth, alturaBarra);
                                g2d.fill(bar);
                                g2d.setColor(Color.DARK_GRAY);
                                g2d.draw(bar);
                                xPosicao += barWidth;
                        }
                }
```

```
                }

            }

        }
```