Computer Vision
Assignment 2
German University in Cairo
Frederic-Gerald Morcos
4-1805
E13

Introduction

The user interface is designed in Glade, converted to GtkBuilder XML using the tool `gtk-builder-convert` and run using the Gtk+ user interface toolkit. Plotting the histograms is done using the Cairo graphics library. The Open Computer Vision library is used to get and set individual pixel values from and to the image.

As can be seen from the Illustrations, RGB histograms are drawn using curves instead of bars for the sake of visibility.

The command line interface to cv2 is as follows:

```
./cv2 <filename> [min-a] [max-b]
```

where if the given image is a gray-scale image, the program will apply contrast stretching using min-a as the desired minimum and max-b as the desired maximum. If no values are given, minimum will be assumed to be 0 and maximum will be assumed to be 255. If the given image is a color image, the program will apply histogram equalization.'

The program will output the histogram drawings to PNG files named histogram-*.png.

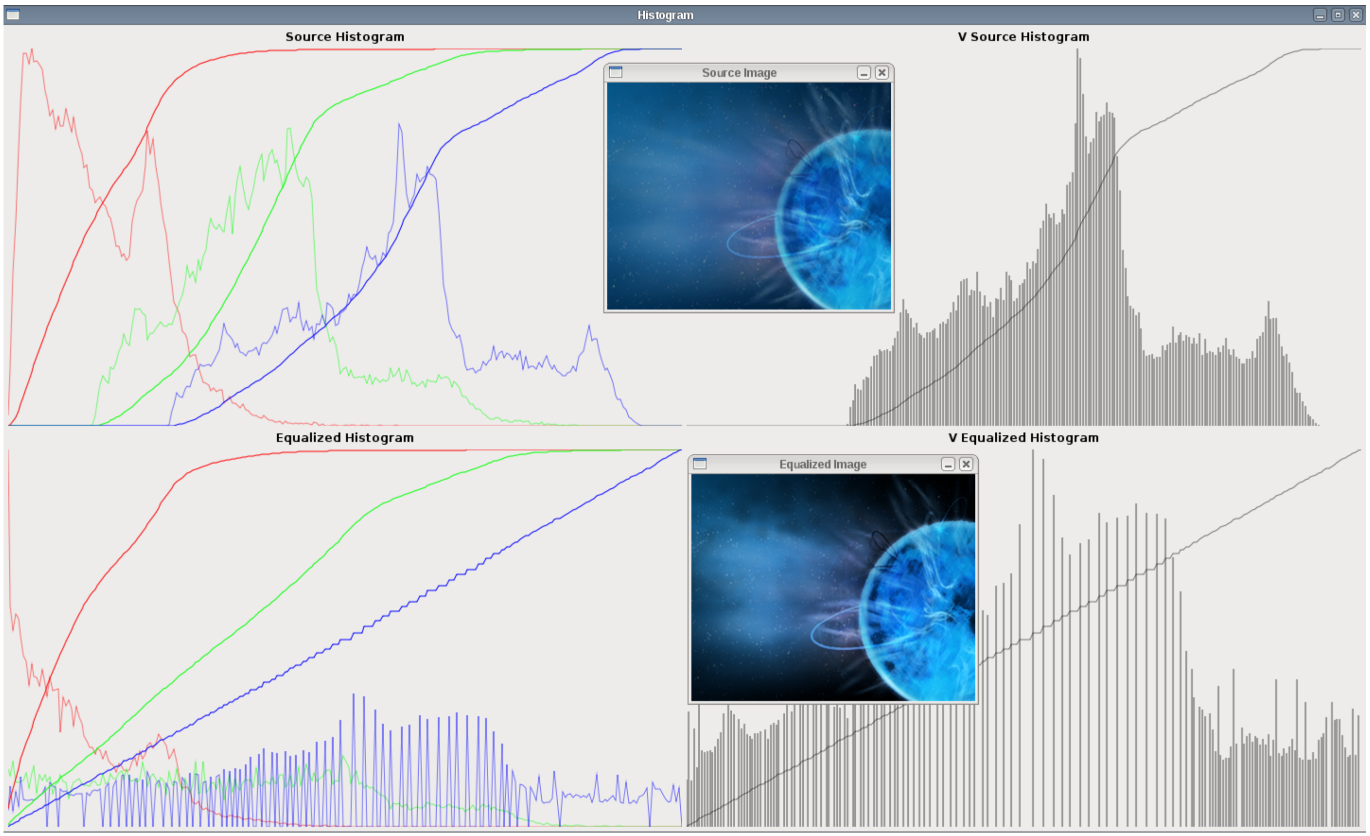


Illustration 1: Histogram Equalization of a Color Image

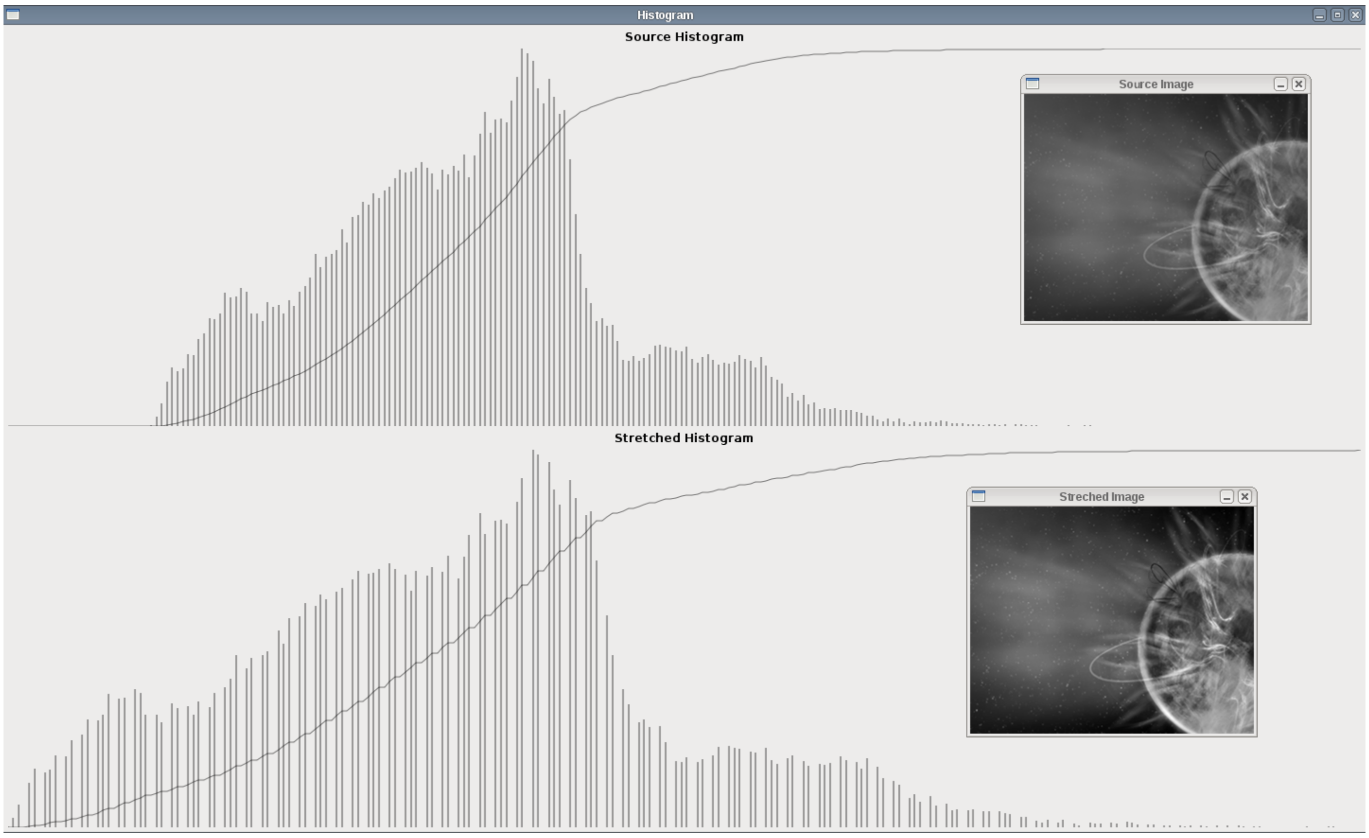


Illustration 2: Contrast Stretching from 0 to 255 of a Grayscale Image

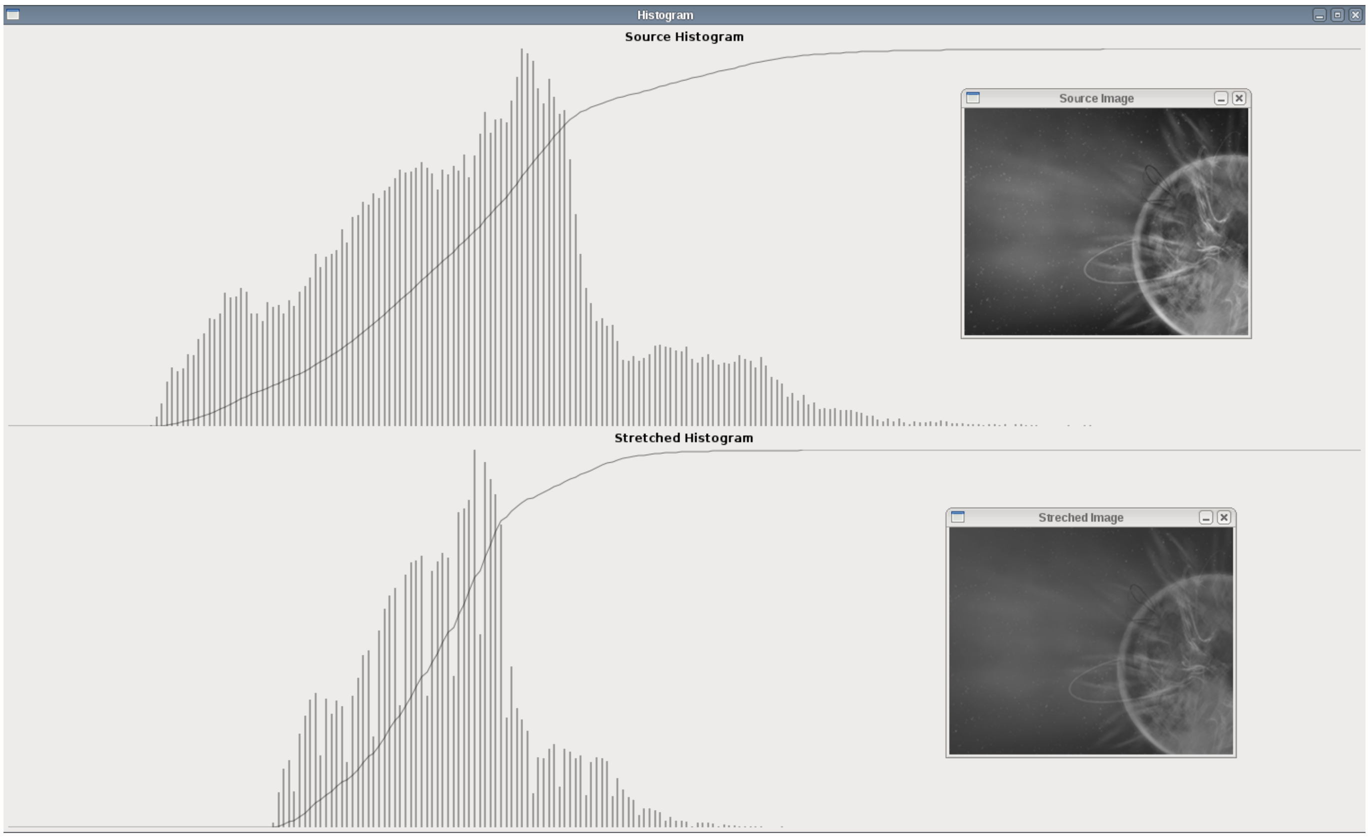


Illustration 3: Contrast Stretching from 50 to 150 of a Grayscale Image

Question 1

Histogram equalization is applied on RGB color images by converting the channels to HSV color-space, applying equalization on the V channel and converting the HSV image (with the new V values) back to RGB.

Equalization of the V channel happens on each pixel as follows:

new_V_value = (cumulative_frequency_of_old_V_value – minimum_cumulative_frequency) * (255 / (total_number_of_pixels_in_image - minimum_cumulative_frequency))

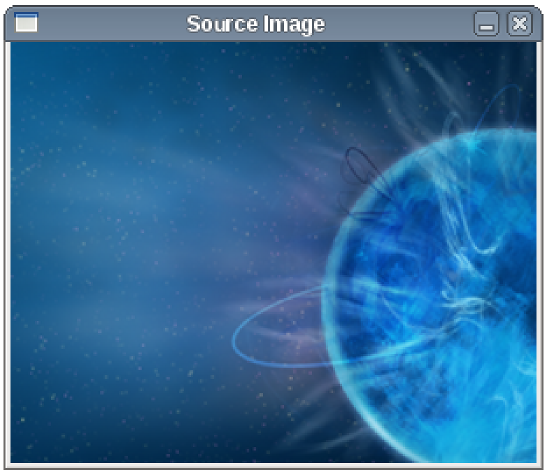


Illustration 4: Before

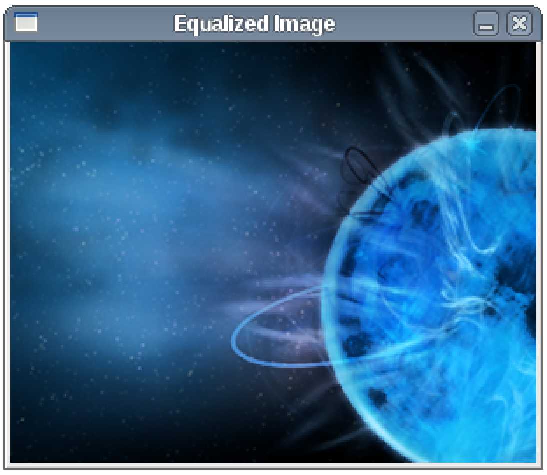
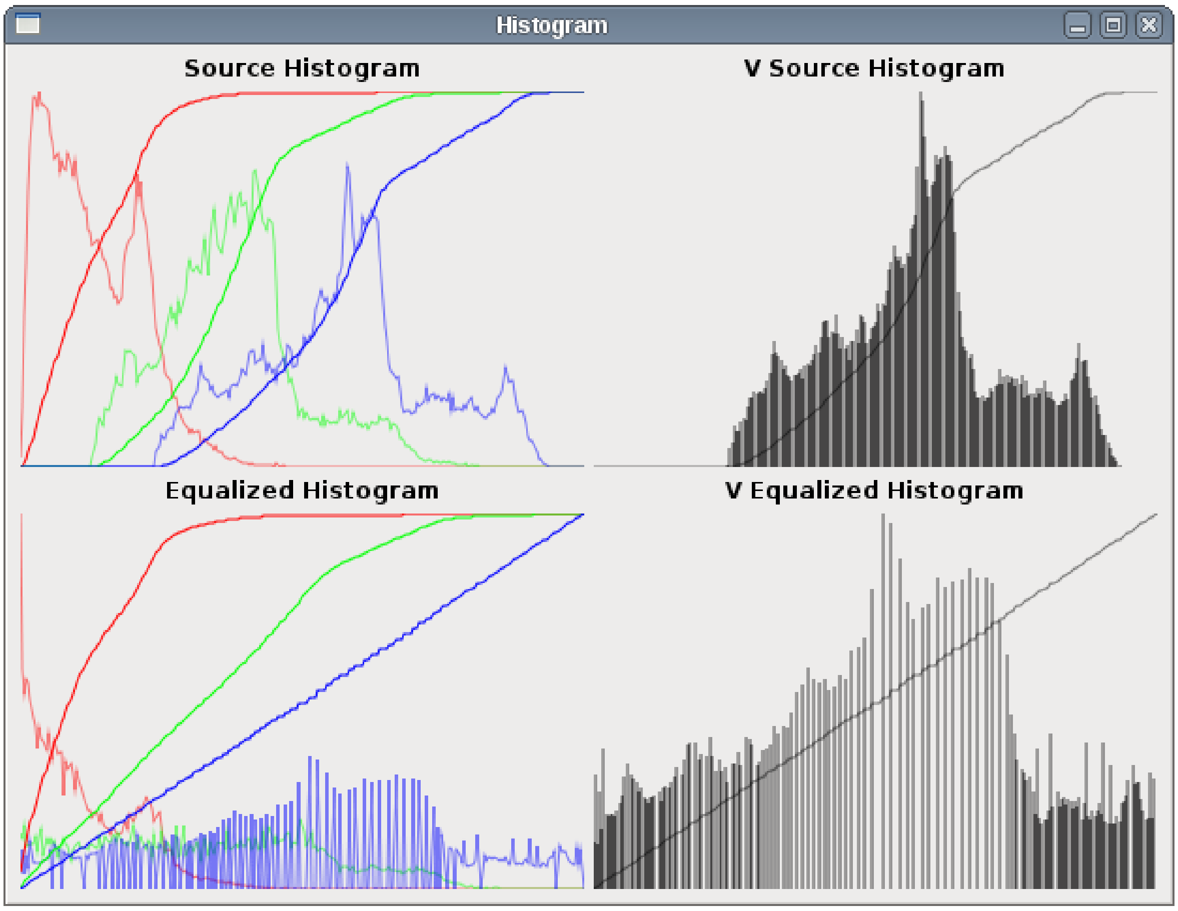


Illustration 5: After



```
- (Image *) equalize: (Image *) vImage {
        Image *img = [[Image alloc] initRGBWithWidth: image->width
                                       andHeight: image->height];
        HSVColor *hsv;

        for (int i = 0; i < image->width; i++)
                for (int j = 0; j < image->height; j++) {
                        hsv = [self HSV: POINT(i, j)];
                        hsv->V = equalizeV(hsv->V, [[vImage histogram] getCF: hsv->V],
                                       [[vImage histogram] getMinCF], [vImage pixels]);
                        [img HSV: hsv toPoint: POINT(i, j)];
                }
        return img;
}

- (unsigned int) getMinCF {
        unsigned int min = cG[0];
        for (int i = 0; i < 256; i++)
                if (cG[i] < min && cG[i] != 0)
                        min = cG[i];
        return min;
}

- (unsigned int) getCF: (unsigned char) intensity {
        return cG[intensity];
}

- (HSVColor *) HSV: (Point *) point {
        RGBColor *tmp = [self RGB: point];
        HSVColor *tmp2 = RGBtoHSV(tmp);
        free(tmp);
        return tmp2;
}

- HSV: (HSVColor *) color toPoint: (Point *) point {
        RGBColor *tmp = HSVtoRGB(color);
        [self RGB: tmp toPoint: point];
        free(tmp);
        return self;
}

unsigned char equalizeV (unsigned char i, unsigned int f, unsigned int fmin,
                                                    unsigned int numPixels) {
        return (f - fmin) * (255.0 / (numPixels - fmin));
}

- (Image *) vImage {
        Image          *img = [[Image alloc] initGrayWithWidth: image->width
                                                    andHeight: image->height];
        for (int i = 0; i < image->width; i++)
                for (int j = 0; j < image->height; j++) {
                        [img GS: (RGBtoHSV([self RGB: POINT(i, j)]))->V toPoint: POINT(i, j)];
                }
        return img;
}

HSVColor *RGBtoHSV(RGBColor *color) {
        HSVColor       *tmp = HSV(0.0, 0.0, 0.0);
        double         max = MAX(color->R, MAX(color->G, color->B)),
                               min = MIN(color->R, MIN(color->G, color->B));

        if (max == min)
                tmp->H = 0.0;
        else if (max == color->R && color->G >= color->B)
                tmp->H = (60 * (color->G - color->B) / (max - min)) + 0;
        else if (max == color->R && color->G < color->B)
                tmp->H = (60 * (color->G - color->B) / (max - min)) + 360;
        else if (max == color->G)
                tmp->H = (60 * (color->B - color->R) / (max - min)) + 120;
        else if (max == color->B)
                tmp->H = (60 * (color->R - color->G) / (max - min)) + 240;

        if (max != 0)
                tmp->S = (max - min) / max;

        tmp->V = max;

        return tmp;
}

RGBColor *HSVtoRGB(HSVColor *color) {
        double         hi = floorf(color->H / 60.0),
                              f = (color->H / 60.0) - hi,
                              p = color->V * (1.0 - color->S),
                              q = color->V * (1.0 - f * color->S),
                              t = color->V * (1.0 - ((1.0 - f) * color->S));
        RGBColor       *tmp = RGB(0.0, 0.0, 0.0);

        if (hi == 0.0) {
                tmp->R = color->V;
                tmp->G = t;
                tmp->B = p;
        }
        else if (hi == 1.0) {
                tmp->R = q;
                tmp->G = color->V;
                tmp->B = p;
        }
        else if (hi == 2.0) {
                tmp->R = p;
                tmp->G = color->V;
                tmp->B = t;
        }
        else if (hi == 3.0) {
                tmp->R = p;
                tmp->G = q;
                tmp->B = color->V;
        }
        else if (hi == 4.0) {
                tmp->R = t;
                tmp->G = p;
                tmp->B = color->V;
        }
        else if (hi == 5.0) {
                tmp->R = color->V;
                tmp->G = p;
                tmp->B = q;
        }

        return tmp;
}
```

NOTE: Source code contains comments.

Question 2

Contrast stretching is applied (to a gray-scale image, in this case) by stretching every pixel value individually as follows:

new_value = (old_value – c) * ((b – a) / (d – c)) + a

where c = the first intensity with non-zero frequency in the original histogram, d = the last intensity with non-zero frequency in the original histogram, a = the desired minimum value and b = the desired maximum value.

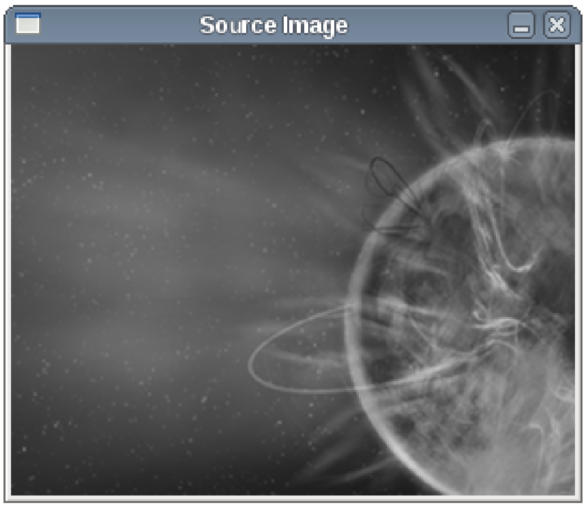


Illustration 7: Before

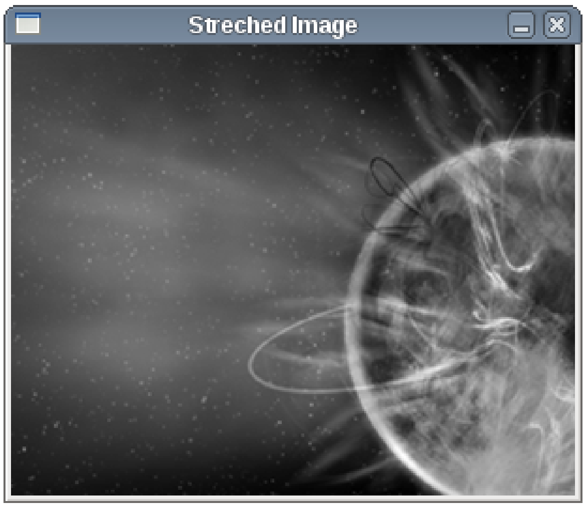


Illustration 8: After

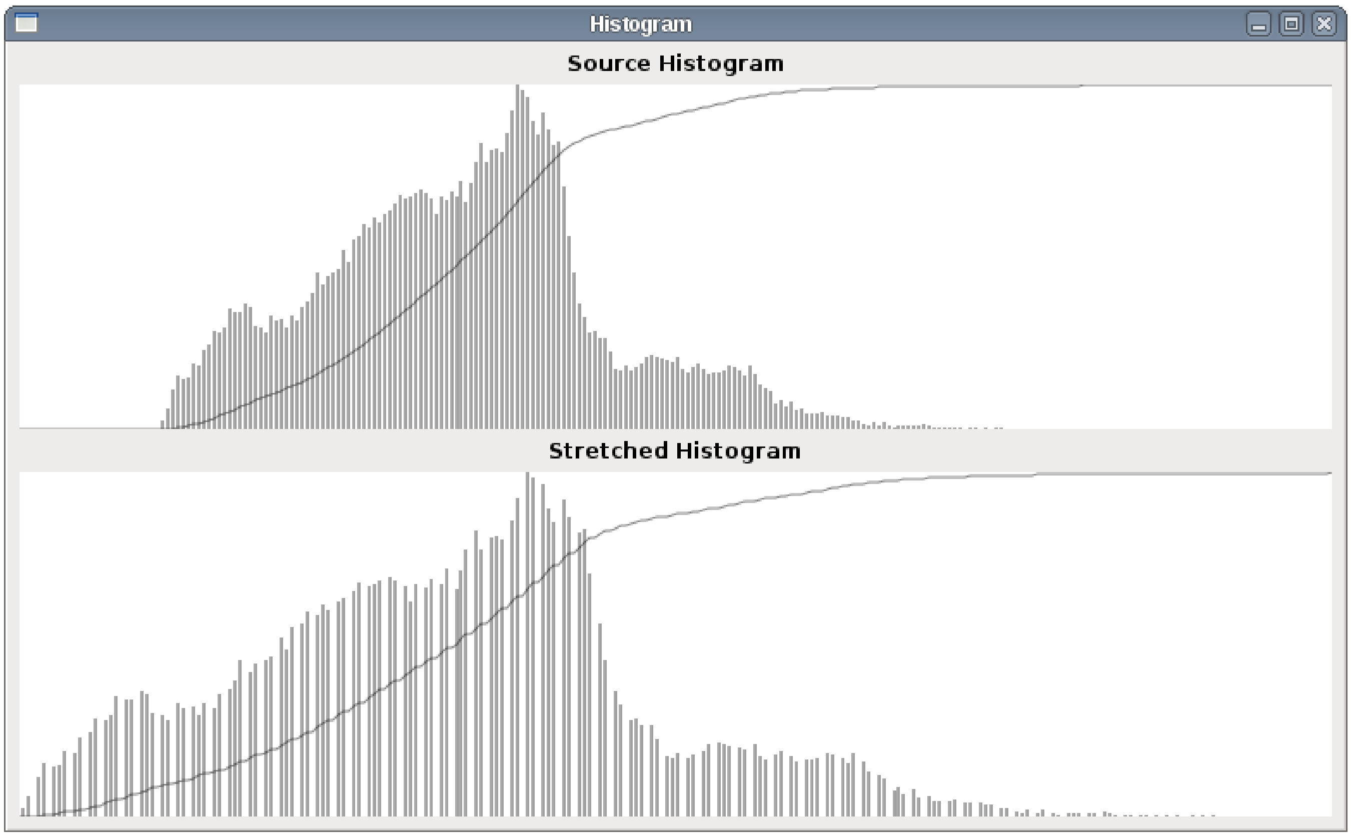


Illustration 9: Histograms

```
- (Image *) stretchWithMin: (int) a andMax: (int) b {
	Image		*img = [[Image alloc] initGrayWithWidth: image->width
						    andHeight: image->height];
	IplImage	*tmp = [img image];
	int			c = [histogram stretchGetC],
				d = [histogram stretchGetD];

	for (int i = 0; i < image->width * image->height; i++) {
		tmp->imageData[i] = stretchV(image->imageData[i], a, b, c, d);
	}
	return img;
}

unsigned char stretchV (unsigned char i, double a, double b, double c, double d) {
	return (i - c) * ((b - a) / (d - c)) + a;
}

- (unsigned char) stretchGetC {
	/* get first intensity with non-zero frequency */
	for (int i = 0; i < 256; i++)
		if (G[i] != 0)
			return i;
	return 0;
}

- (unsigned char) stretchGetD {
	/* get last intensity with non-zero frequency */
	for (int i = 255; i >= 0; i--)
		if (G[i] != 0)
			return i;
	return 255;
}
```

NOTE: Source code contains comments.