**1. Xử lý ảnh không dùng các thư viện có sẵn**

Để xử lý một bức ảnh ta phải chuyển ảnh thành 1 mảng 2 chiều mà mỗi phần tử sẽ chứa giá trị màu của 1 pixel. Trong Java có cung cấp sẵn hàm getRGB(row, col), setRGB(row, col) để lấy, gán giá trị màu tại pixel có tọa độ x = row, y =col. Nhưng cách làm này sẽ làm cho chương trình của chúng ta chạy chậm vì hàm getRGB và setRGB sẽ thực hiện liên tục việc duyệt bức ảnh từ đầu đến cuối.

Để tránh việc duyệt đi, duyệt lại bức ảnh như vậy (đỡ tốn thời gian) chúng ta có thể chuyển ảnh thành 1 mảng byte, sau đó đọc mảng byte này và lưu thành từng giá trị int tương ứng với mỗi pixel rồi lưu vào 1 mảng 2 chiều tượng trưng cho bức ảnh để tiện xử lý.

Chương trình sau, chúng ta sẽ so sánh việc sử tốc độ của chương trình khi sử dụng hàm có sẵn và hàm do ta tự xây dựng.

- Sử dụng hàm getRGB và setRGB :

*private static int[][] convertTo2DUsingGetRGB(BufferedImage image) {*

*int width = image.getWidth();*

*int height = image.getHeight();*

*int[][] result = new int[width][height];*

*for (int row = 0; row < width; row++) {*

*for (int col = 0; col < height; col++) {*

*result[row][col] = image.getRGB(row, col);*

*}*

*}*

*return result;*

*}*

- Không sử dụng hàm getRGB và setRGB:

*private static int[][] convertTo2DWithoutUsingGetRGB(BufferedImage image) {*

*Raster raster = image.getRaster();*

*DataBufferByte dataBufferByte = (DataBufferByte) raster.getDataBuffer();*

*byte[] pixels = dataBufferByte.getData();*

*int width = image.getWidth();*

*int height = image.getHeight();*

*boolean hasAlphaChannel = image.getAlphaRaster() != null;*

*int[][] result = new int[height][width];*

*if (hasAlphaChannel) {*

*// Have alpha value*

*final int pixelLength = 4;*

*for (int pixel = 0, row = 0, col = 0; pixel < pixels.length; pixel += pixelLength) {*

*int argb = 0;*

*argb |= (((int) pixels[pixel] & 0xFF) << 24); // alpha*

*argb |= ((int) pixels[pixel + 1] & 0xFF); // blue*

*argb |= (((int) pixels[pixel + 2] & 0xFF) << 8); // green*

*argb |= (((int) pixels[pixel + 3] & 0xFF) << 16); // red*

*result[row][col] = argb;*

*col++;*

*if (col == width) {*

*col = 0;*

*row++;*

*}*

*}*

*} else {*

*// Don't have alpha value*

*final int pixelLength = 3;*

*for (int pixel = 0, row = 0, col = 0; pixel < pixels.length; pixel += pixelLength) {*

*int argb = 0;*

*argb |= 0xFF << 24; // alpha = FF*

*argb |= ((int) pixels[pixel] & 0xff); // blue*

*argb |= (((int) pixels[pixel + 1] & 0xff) << 8); // green*

*argb |= (((int) pixels[pixel + 2] & 0xff) << 16); // red*

*result[row][col] = argb;*

*System.out.println(row+"-"+col+"-"+result[row][col]);*

*col++;*

*if (col == width) {*

*col = 0;*

*row++;*

*}*

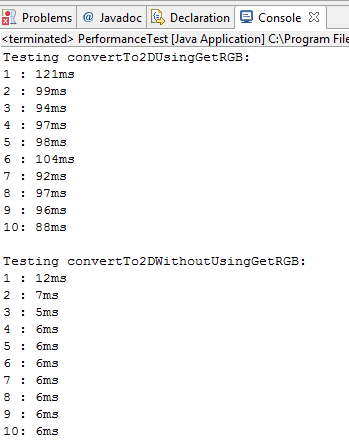
*}*

*}*

*return result;*

*}*

- Kết quả thu được khi ta tiến hành đọc dữ liệu của bức ảnh và lưu vào dưới dạng mảng pixel của bức ảnh có kích thước 1920 x 1080 bằng cách sử dụng hàm getRGB và không dùng hamg:

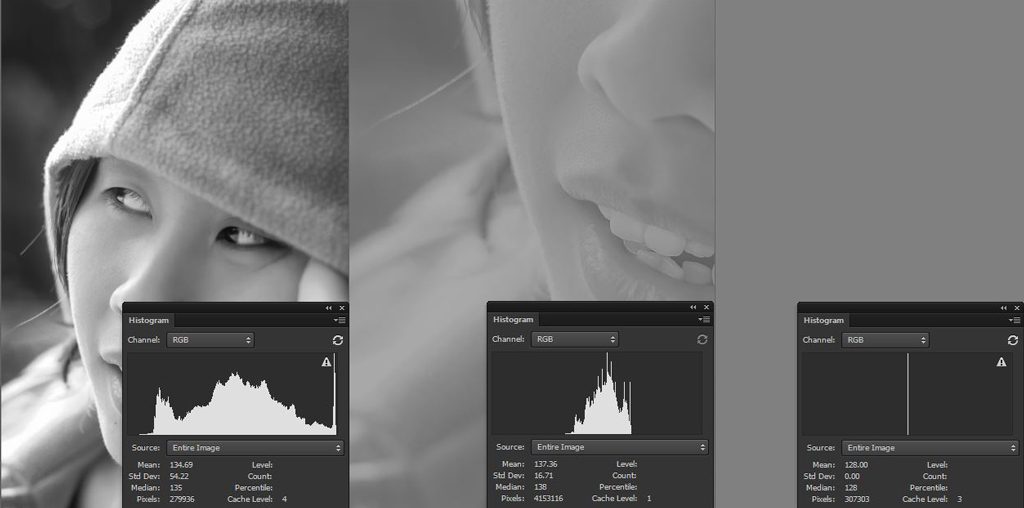


Rõ ràng việc không sử dụng hàm getRGB cho thời gian thực hiện nhanh hơn rất nhiều. Ở đây ta chỉ mới sử dụng hàm getRGB, nếu ta sử dụng thêm hàm setRGB thì chênh lệch thời gian xử lý sẽ còn lớn hơn nữa.

**2. Histogram Equalization**

**a) Khái niệm Histogram**

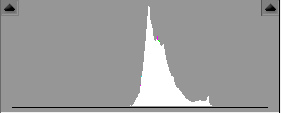
- Histogram là một biểu đồ thể hiện số lượng điểm ảnh và mức độ sáng tối của ảnh chụp. Histogram gồm có 2 trục: trục x có giá trị từ 0 -> 255 biểu hiện mức độ sáng của điểm ảnh, trục y là số điểm ảnh trong bức ảnh có độ sáng là i với 0 <= i <= 255.

- Một bức ảnh được tạo thành từ nhiều điểm ảnh có mức độ sáng tôí khác nhau, càng có nhiều mức độ sáng tối càng có nhiều chi tiết trong ảnh, nếu tất cả các điểm ảnh đều có cùng một độ sáng, ảnh sẽ không có một chi tiết nào hết.

Như vậy Histogram càng rộng, càng trải dài thì bức ảnh càng chi tiết và ngược lại nếu histogram càng hẹp thì càng ít chi tiết. Ta chú ý rằng nếu histogram nghiêng về bên trái (mức xám 0) hay bên phải (mức xám 255) thì ảnh sẽ bị quá tối hoặc quá sáng

**b) Cân bằng Histogram**

- Bức ảnh sau ta nhìn thấy có khá là ít chi tiết, nó được thể hiện rõ ràng qua Histogram của bức ảnh chỉ là một dải hẹp.





- Muốn làm cho bức ảnh trở nên chi tiết hơn ta phải kéo dãn Histogram sao cho trải dài từ 0 -> 255 để tăng độ chi tiết cũng như mức độ tương phản của bức ảnh.

Cách làm:

- Tính xác suất tương ứng với mức xám i (0 <= i < L) trong bức ảnh:

C:\Users\Hop\Pictures\f51c6c10db082a4ee8e7da15c042e157.png

Trong đó: i là mức sáng

ni là số điểm ảnh có mức xám i trong bức ảnh

n là số lượng điểm ảnh có trong bức ảnh

L số lượng mức xám sử dụng trong bức ảnh, thường là 256

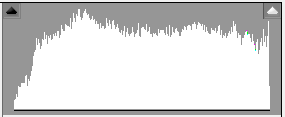
- Tính hàm phân phối xác suất (Cumulative Distribution Function) bằng cách cộng xác suất của mức xám hiện tại với tất cả các xác suất của các mức xám trước nó:

C:\Users\Hop\Pictures\latex.png

- Tạo Lookup Table: ta tiến hành nhân cdf(i) với (L - 1), L là số lượng mức xám sử dụng trong ảnh, thường là 256 để thu được mức xám mới để thay thế cho các mức xám cũ.

- Tiến hành thay thế các mức xám cũ băng các mức xám mới thu được ở bước trên ta được ảnh như hình dưới:





Bức ảnh đã trở nên chi tiết và tương phản hơn rất nhiều. Bên cạnh là histogram của ảnh sau khi cân bằng, được trải dài trên cả 256 mức xám.

* Problem1 Cân bằng histogram cho ảnh xám đã được nhưng khi làm đối với ảnh màu thì bị sai. Nguyên nhân là do RGB là hệ tọa độ màu còn phương pháp mà ta đang dùng để cân bằng histogram dựa trên mức xám của 1 điểm ảnh.

Giải pháp: Chuyển hệ tọa độ từ RGB sang HSV (Hue Saturation Value).

RGB to HSV conversion formula

The R,G,B values are divided by 255 to change the range from 0..255 to 0..1:

R' = R/255

G' = G/255

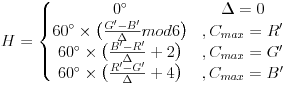
B' = B/255

Cmax = max(R', G', B')

Cmin = min(R', G', B')

Δ = Cmax - Cmin

Hue calculation:



Saturation calculation:

C:\Users\Hop\Downloads\sat-calc.gif

Value calculation:

V = Cmax

**3. Hough Transform**

**Note:** Trong class ComponentSampleModel, có thuộc tính protected int scanlineStride, thuộc tính này cần được truyền vào khi khởi tạo SampleModel.

scanlineStride phụ thuộc vào type của bức ảnh xuất ra, chứ ko liên quan gì đến bức ảnh đầu vào. Ta có thể get scanlineStride sau khi khởi tạo BufferdImage:

BufferedImage bufImg = **new** BufferedImage(width, height,

BufferedImage.***TYPE\_3BYTE\_BGR***);

WritableRaster writableRaster = bufImg.getRaster();

ComponentSampleModel coModel = (ComponentSampleModel) writableRaster

.getSampleModel();

**int** lineStride = coModel.getScanlineStride();