Lab 2 - RGB to Grayscale Conversions

Consider a color image, given by its red, green, blue components ***R, G, B.*** We present some methods for converting the color image to grayscale. Implement all these methods. (http://www.tannerhelland.com/3643/grayscale-image-algorithm-vb6/)

1. **Simple averaging**

$$\boldsymbol{Gray = (R + G + B) / 3}$$ (better use R/3+G/3+B/3) .

2. **Weighted average**

$$\boldsymbol{Gray = 0.3R + 0.59G + 0.11B}$$

or

$$\boldsymbol{Gray = 0.2126R + 0.7152G + 0.0722B}$$

or

$$\boldsymbol{Gray = 0.299R + 0.587G + 0.114B}$$

3. **Desaturation**
   In the HSI color model the S is the saturation component of the color. Saturation describes the intensity (purity) of that hue. When colour is fully saturated, the colour is considered in purest (truest) version. Primary colours red, blue or yellow are considered truest version colour as they are fully saturated.
   Desaturating an image means setting the S component to 0. This can be done by averaging the minimum and the maximum values or R, G, and B:

$$\boldsymbol{Gray = (\min(R,G,B) + \max(R,G,B)) / 2}$$.

4. **Decomposition**
   - Maximum $\boldsymbol{Gray = \max(R,G,B)}$
   - Minimum $\boldsymbol{Gray = \min(R,G,B)}$

5. **Single colour channel**

$$\boldsymbol{Gray = R}$$

or

$$\boldsymbol{Gray = G}$$

or

$$\boldsymbol{Gray = B}$$

6. **Custom number of grey shades**
   Usually a grayscale image has intensities that range between 0 to 255. If one wants to reduce the number of shades, one can divide the interval [0, 255] in a number of subintervals equal with the number of grey shades that we want to use. Let $p < 256$ be the number of shades we want to use. We consider the intervals:

$$[a_0 = 0,\ a_1),[a_1,\ a_2),\ldots,[a_{p-1},\ a_p = 255],\quad .$$

All the pixels with intensities in interval $[a_{i-1}, a_i)$ will be set to a common value (the average value of the pixels with intensities in the given interval, for example). If we have a colour image, first compute one of the weighted average of the colour components (see 2.) and then apply the above described algorithm. Generate the numbers $\{a_i; i = 1, ..., p\}$ randomly, $a_0 = 0 < a_1 < a_2 < \cdots < a_{p-1} < a_p = 255,$ or such that all the intervals have the same length, that is $a_i - a_{i-1} = 255 / p$.

7. Custom number of grey shades with error-diffusion dithering

Image dithering: http://www.tannerhelland.com/4660/dithering-eleven-algorithms-source-code/

Implement both the Floyd-Steinberg and the Stucki Dithering algorithms.

We explain error diffusion using an example. Consider the case of transforming a grayscale image into a binary one. The first pixel in the image is dark grey, assume it has 65 intensity, being closer to 0 (black) than to 255 (pure white) the pixel will be assigned to 0. Error diffusion works by spreading the error of each calculation to neighboring unvisited pixels. If it finds a pixel of 85 gray, it too determines that 85 is closer to 0 than to 255 and so it would make the pixel black. The error diffusion algorithm makes note of the "error" in its conversion, specifically, that the first gray pixel we have forced to black was actually 65 steps away from black. When it moves to the next pixel, the error diffusion algorithm adds the error of the previous pixel to the current pixel. If the next pixel is 85 gray, instead of simply forcing that to black as well, the algorithm adds the error of 65 from the previous pixel. This results in a value of 150, which is actually closer to 255, and thus closer to white.
The Floyd-Steinberg dithering algorithm is using the following diffusion mask:

$$\begin{bmatrix} \# & * & \frac{7}{16} \\ \frac{3}{16} & \frac{5}{16} & \frac{1}{16} \end{bmatrix}$$

The * marks the position of the currently processed pixel. The error is diffused only on pixels that were not already visited (the pixels marked # are left unchanged, because they were already processed).
The mask for the Stucki dithering method is:

$$\frac{1}{42}\begin{bmatrix} \# & \# & * & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{bmatrix}$$

Basic Floyd-Steinberg dithering algorithm

$$
\begin{aligned}
&I = \textit{Input Image}\\
&\textit{OutputImage} = 0;\\
&\textit{for}\ \ i = 1\ \textit{to}\ m\\
&\quad \textit{for}\ \ j = 1\ \textit{to}\ n\\
&\qquad \textit{OutputImage}[i,j] = \textit{NearestColor}(I[i,j]);\\
&\qquad \textit{err} = I[i,j] - \textit{OutputImage}[i,j];\\
&\qquad I[i,j+1] \mathrel{+}= \textit{err} * (7/16);\\
&\qquad I[i+1,j-1] \mathrel{+}= \textit{err} * (3/16);\\
&\qquad I[i+1,j] \mathrel{+}= \textit{err} * (5/16);\\
&\qquad I[i+1,j+1] \mathrel{+}= \textit{err} * (1/16);\\
&\quad \textit{end for } j\\
&\textit{end for } i
\end{aligned}
$$

Documentation exercise: **What about the "inverse" problem: grayscale to RGB?**

Try to find a very simple method that transforms a grayscale image (a medical image, for example) in a colour one (don't consider the 'making sense' problem).