

Lab Assignment 2 – CSE 332/564, Spring 2008

Due: Thursday, February 26, 11:59pm

This lab will introduce you to global image manipulations. Continue to use the code you developed for lab 1. Upon completion of the assignment you will submit the following, via blackboard:

- the complete software (all that is needed to build the executable: source code, project files, etc)
- an executable (.exe file) of your work
- a comprehensive report that illustrates with images and narrative text all aspects of your work
- any images not part of the provided collection that you have used for testing your software

All of these components are equally important. Please note the policies posted on the class webpage. Also please note: While this assignment looks short, it will take time to do. If graphics programming is new to you, it will take some time to understand the mechanisms here. But you will need these concepts throughout the course, so it pays off to understand them now.

Check out the new demo on the lab page to see suggestions how the following will work in practice.

1. Add new items to the GUI

Use FLUID to add a row of 7 toggle buttons below the transfer function window. Read the FLTK manual how toggle buttons work. The idea is that only one button can be selected at a time, and an event is generated with the appropriate value when a button is selected. Of course, you need a routine to service this event and read the value. The 7 toggle buttons are: 'All', 'R', 'G', 'B', 'H', 'S', 'V'. Using, these, depending on what button has been selected, the user will be able to modify different properties of the image, using the transfer function, and also a different histogram is displayed.

Below the row of toggle buttons, add two independent check buttons: 'interactive' and 'log scale'.

On the menu bar, add a sub menu 'Image Processing' and add two menu items 'Grayscale' and 'Undo'.

2. Compute and draw the RGB histograms

After reading an image, the variable *nchan* will tell you if the image was grayscale or RGB color (*nchan*=1 or 3, respectively). But in any case, you will always have 3 bytes per pixel. In case of a grayscale image, R=G=B for each pixel (see the routine Grayscale() in Application.cpp). First, write a routine ComputeHistogram() that computes the RGB histogram for the image. For this, you will need a 3D array of length $2^8 = 256$ each to count the number of pixels that have a certain R, G, and B value (here the array index *i* will be given by the R,G,B value, so as you loop through the image you will increment hist[*i*] each time you encounter a value *i*). You should do this every time you read in a new image.

Once the histogram has been computed, draw it into the transfer function window. Which channel is drawn will depend on the set toggle button. When 'R'('G','B') is selected, draw the 'R' ('G','B') histogram in red (green, blue) color. When 'All' has been selected draw all such curves. The drawing itself is done similar than the way the transfer function is drawn (see code in TransFuncEditor.cpp), only now the histogram values will determine the vertex positions (here the x-axis is the R,G,B value {0,...,

255} and the y-axis is the number of pixels having this value). Connect these vertices to form a curve. To make optimal use of the display, find the maximum histogram count for each channel and scale the vertex y-coordinates such that the tallest point of the curve is close to the window height (given by the variable $h()$, check the code). The scale factor is $h()/\max(y)$, which will map the value $\max(\text{hist})$ when it occurs to $h()$.

You will see that sometimes one histogram range has a high number of counts, which will dwarf the remaining curve. Pushing the log scale button should convert the y-axis in the histogram to $\log(y)$. This is a common way to reduce the effect of large outliers in the visualization of a function. Note, if $\text{hist}[i]=0$ then simply map this value to 0. Again, find the maximum value of this converted series to map the $\max(\log(\text{hist}))$ to $h()$.

Each time the user selects a different toggle button, the corresponding histogram curve should be drawn.

2. Compute and draw the HSV histograms

Compute the HSV conversion of the image (see the algorithms in Foley/VanDam pp. 590-593 or look on the web for a suitable routine). If it is a gray-scale image just fill the HSV channels with the R channel. Do this also when you read in a new image. Compute the H,S,V histograms and display it in the same manner than the R,G,B histograms, if the corresponding toggle button has been selected by the user.

3. Convert a color image to gray-scale

This is simple – just fill the RGB channels with the V channel since V is the brightness channel. You could experiment with the other channels, H or S and see what you get.

4. Modify the image using the transfer function editor

Now we are getting to the interesting and potentially artistic part of the assignment. As discussed in class the transfer function serves as a way to translate the current pixel values into new values, that is, all values i will be assigned a new value $\text{transfunc}[i]$. If the button 'interactive' is not depressed, update the image and redraw it only when the mouse button is released. Else, update and redraw the image every time the mouse moves. In both cases, whenever you release the mouse button, you *commit* the change and the image array is updated with the mapped values. Before this commit-step, the transfer function will manipulate the 'old' image, that is, the one currently stored in the array, resulting from a previous commit or load.

The efficiency of your code, the size of the image, and the performance of your computer will determine if interactive is really interactive. The image channel updated (that is, R,G,B,H,S,V) will depend on the toggle button selected. Also redraw the histogram. For example, if you select H you will be able to modify the hues, which will give some interesting results. On the other hand, if you select V you will be able to modify brightness, which is useful to increase overall contrast or enhance contrast within some brightness ranges, while de-emphasizing other ranges. Try your luck at the low-contrast images provided to improve their appearance.

5. Undo function

Every good image processing package provides an ‘Undo’ utility, and so should yours. Simply save your current image array (the one resulting from previous commit or load) into a copy and swap it back into the active array when the undo menu item is selected. If you are ambitious you could also provide a ‘redo’ function and/or multiple levels of ‘undo’ by keeping a circular array of previous image manipulation results.

Extra credit (+10%)

Implement histogram equalization (add a corresponding submenu item to the Image Processing menu).