Assignment 3 - Software I, Spring 2004 (0368-2157-09/10/11/12)

`http://www.cs.tau.ac.il/~efif/courses/Software1_Spring_04`

Due: May 11, 2003

Required Knowledge arrays, malloc, free, getopt, glut, ppm
Before starting to answer the questions, please read very carefully the "Submission Guidelines"[1].

Ex 3 photo

In this assignment you are asked to implement a couple of operations on two-dimensional images, using the photo program shown in class as your framework. The first operation is to reflect an input image either horizontally or vertically, and the second operation is to scale down an input image using a specified method. You need to implement two methods as explained below. Before you continue, please download the source file of the photo program from the web practice-page at
`http://www.cs.tau.ac.il/~efif/courses/Software1_Spring_04/code/photo/photo.c`, and familiarize yourself with the code. Then, add the implementation of the command-line options below.

The photo program reads an image in binary Portable Pixmap File (PPM) format as input[2]. In this format each pixel is a triplet of red, green, and blue samples one character each, in that order. The program opens a window using the glut library, and displays the image on the window. You need to enhance the program to perform the required operation non interactively based on various command line options. For your convenience, you may enhance the program to perform the same operation interactively, as a response to various user key-strokes. The syntax of the command line is:

```
photo [options] filename
```

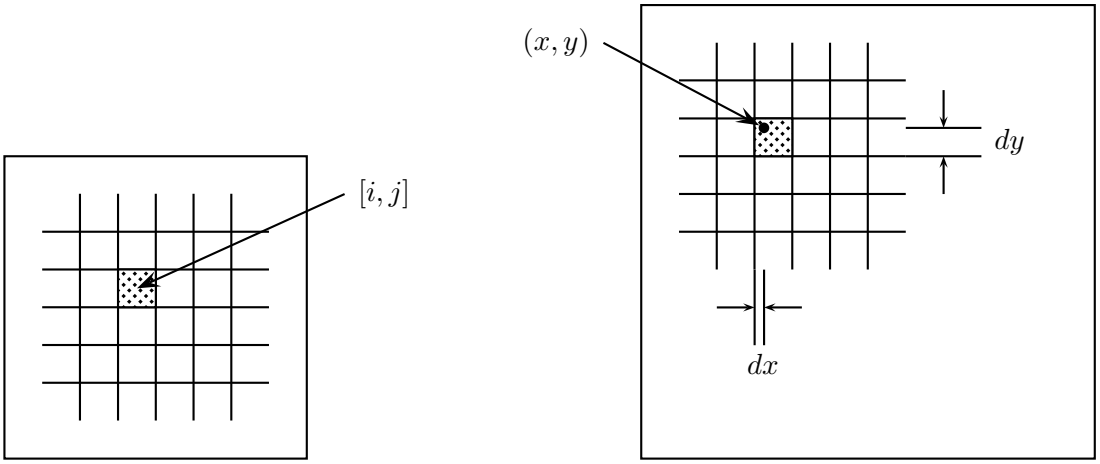
The various options are listed below and the `filename` is the name of a PPM-image file.
The source file `photo.c` that you downloaded contains already a function that reflects an imgae horizontally. You need to implement a function that reflects an image vertically and add the appropriate code to tie each of these functions with the user selection.

Terms and Definitions (required for the scale operations)

Let w_s and h_s be the width and the height of the source image respectively, and w_d and h_d be the width and the height of the destination image respectively. Let $C_s[i,j]$ and $C_d[i,j]$ denote the pixel colors of the source image and destination image respectively at position $[i,j]$. The formulae below for $C_d[i,j]$ must be applied to each of the red, green, and blue components separately.
Let (x,y) be the non-integer position in the source image that corresponds to the position $[i,j]$ in the destination image, and let dx and dy be the fractional part of x and y.

$$x = i \times w_s/w_d \qquad dx = x - \lfloor x \rfloor$$
$$y = j \times h_s/h_d \qquad dy = y - \lfloor y \rfloor$$

[1] `http://www.cs.tau.ac.il/~efif/courses/Software1_Spring_04/subgd.php`
[2] Binary PPM format implies that the image data is binary, but the header is ASCII



Command Line Options

-f factor

set the scale factor. The default is 1.0.

-h

print this help message.

-r direction

reflect the image about the indicated direction. Accepted directions are:

horizontal

reflect the image about horizontally (already implemented).

vertical

reflect the image vertically.

-s method

scale down the image using the indicated method. The accepted methods are:

nearest

this indicates the nearest-neighbor method, which is the default. The nearest-neighbor method is the most simple method to resize an image. This method finds the closest corresponding pixel in the source image for each pixel in the destination image.

$$C_d[i, j] = C_s[\lfloor x \rfloor, \lfloor y \rfloor]$$

cubic

this indicates the cubic B-Spline method. This method estimates the color of a pixel in the destination image by the average of the colors of the 16 pixels surrounding the closest corresponding pixel in the source image.

$$C_d[i, j] = \sum_{m=-1}^{2} \sum_{n=-1}^{2} C_s[\lfloor x \rfloor + m, \lfloor y \rfloor + n] R(m - dx) R(n - dy)$$

where

$$R(x) = \frac{1}{6}[P(x+2)^3 - 4P(x+1)^3 + 6P(x)^3 - 4P(x-1)^3]$$
$$P(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Assume that the border of the source image is replicated as many times as necessary, when computing the pixels near the border of the destination image. That is, $C_s[i,j] = C_s[0,j]$ for $i < 0$, and $C_s[i,j] = C_s[i,0]$ for $j < 0$.

-t

test the reflect and scale down operation.

If this option is not indicated, the program displays the input image, and awaits for instructions from the user in form of key strokes. When this is entered by the user however, the program executes in non-interactive mode. First, the image is read. Then, it is transformed according to the indicated options. If both reflection and scaling are required, reflect first then scale. Finally, its signature is calculated as explained below, and printed out in hexadecimal format with 8 digits padded with 0's. The program exits immediately after. This option will be used by the automatic testing script. The signature is the weighted[3] sum carried out in `unsigned int` given by:

$$\sum_{j=0}^{h}\sum_{i=0}^{w}(j \times w + i)(R[i,j] + G[i,j] + B[i,j])$$

where w and h are the image width and height respectively, and $R[i,j]$, $G[i,j]$, and $B[i,j]$ are the red, green, and blue components of pixel $[i,j]$ respectively.

Examples

Input:

```
photo -f 2.0 -s cubic -t photo.ppm
```

Output:

```
0x493da39b
```

This command tests the cubic B-spline scale method on the input image `photo.ppm`. The image is scaled down by a factor of 2.0.

Input:

```
photo photo.ppm
```

This command displays the input image `photo.ppm`.

[3]The weight is applied to have different signatures of reflected images in different directions.

Additional instructions

- Use `double` type to represent all real numbers in the program and to carry out all arithmetic operations on these numbers.
- For your convenience add the option below to the `keyboard` function, to allow the user to transform the image interactively.

  'h' - reflect the image horizontally

  'v' - reflect the image vertically

  'n' - scale down the image using the nearest-neighbor method

  'b' - scale down the image using the cubic B-Spline method
- The program must be linked with the glut library. Add the command line option `-lglut` to the compilation command line:

  ```
  gcc -Wall -lglut -o photo photo.c
  ```
- The program uses the glut library, which in turns uses the openGl library. This imposes a restriction on the image width — it must be a multiple of 4 to be displayed properly. Make sure to pad each line in the image with black (0 value) pixels accordingly.
- Make sure that every allocated memory block is deallocated when the program exists, or better yet, when the block is not needed any longer.
- If you want to practice on windows but still compile with gcc under cygwin, you need to issue the following compile-and-link command instead:

  ```
  gcc -Wall -lglut32 -lopengl32 -o photo photo.c
  ```
- If you want to compile with Visual Studio, you can download the cross-platform source-code file from:
  `http://www.cs.tau.ac.il/~efif/courses/Software1_Spring_04/code/photo/cp_photo.c`. (Notice that it is a different file.) If `cl` and `link` are accessible from the command line, you can issue the command:

  ```
  cl -nologo -c cp_photo.c
  link -nologo -subsystem:console -libpath:lib glut32.lib opengl32.lib cp_photo.obj
  ```

  Beware, the signatures produced by an executable compiled on windows with Visual Studio may deviate from those produced by an executable compiled with gcc on Linux.
- You can use IrfanView to convert more images to PPM format. IrfanView is free and can be downloaded from `http://www.irfanview.com/`.
- The `C` source file `photo.c` should be located under `~/Software1`.
- As usual, before submitting the solution set, please give permission to the files by executing the following command:

  ```
  chmod 705 ~ ~/software1 ~/software1/assign3 ~/software1/assign3/*
  ```