EECS 22: Advanced C Programming
Week 8

Mihnea Chirila
mchirila@uci.edu

11/17/2017

Main Menu

```
--------------------------------
1: Load a PPM image
2: Save an image in PPM and JPEG format
3: Change a color image to Black and White
4: Make a negative of an image
5: Color filter an image
6: Sketch the edge of an image
7: Shuffle an image
8: Flip an image vertically
9: Mirror an image vertically
10:Add border to an image
11:Add noise to an image
12:Sharpen an image
13:Posterize an image
14:Blur an image
15:Crop
16:Resize
17:Brightness and Contrast
18:Watermark
19:Test all functions
20:Exit
Please make your choice:
```

This Week

Resize

This function resizes the image with the scale of percentage.

3 different cases:

- percentage == 100
  the size of the new image is the same as the original one.
- percentage < 100
  the size of the new image is smaller than the original one.
- percentage > 100, the size of the new image is larger than the original one.

Example:

Original

Resized to 80%

Resize

More specifically, we scale percentage as follows:

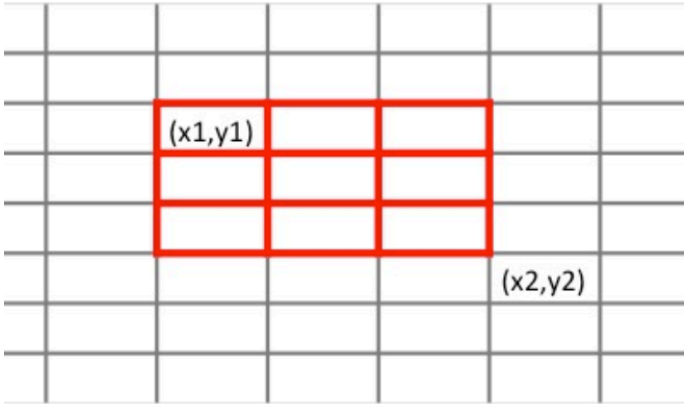
Percentage > 100:

- X_old = x_new * (100 / percentage)
- Y_old = Y_new * (100 / percentage)

Percentage < 100:

- X1_old = X_new * (100 / percentage)
- Y1_old = Y_new * (100 / percentage)
- X2_old = (X_new +1) * (100 / percentage)
- Y2_old = (Y_new +1) * (100 / percentage)

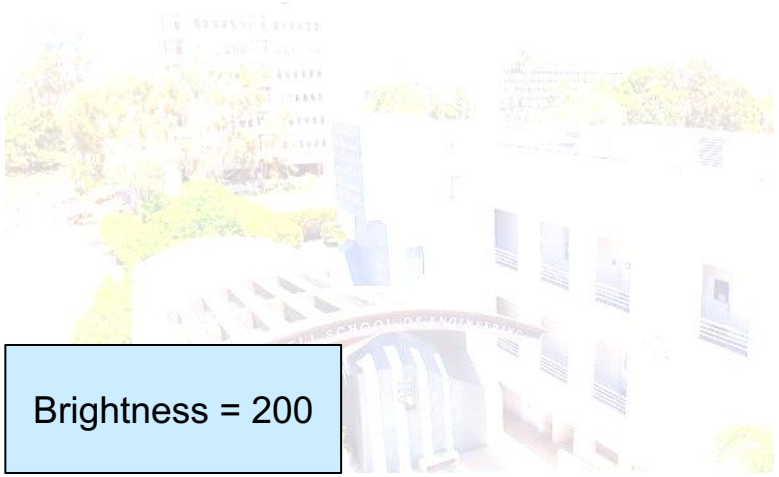
- Average over the pixels from X1_old to X2_old -1 and Y1_old to Y2_old -1



Brightness And Contrast

Original

Brightness = -100



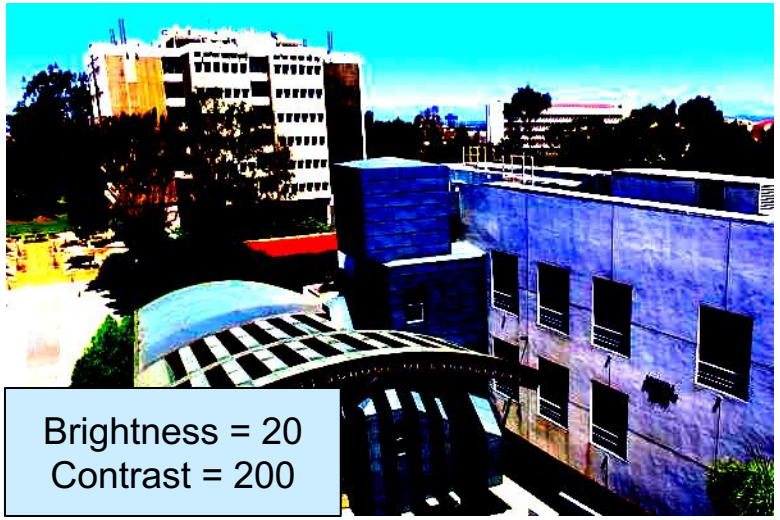
Brightness = 200

Brightness And Contrast

Original

Contrast = -128

Contrast = 128



Brightness = 20
Contrast = 200

Brightness And Contrast

```
IMAGE* BrightnessAndContrast(
        IMAGE *image, int brightness, int contrast);
```

- NewPixelValue = ContrastValue + BrightnessValue
- BrightnessValue
  - User defined between -255 and 255
  - Check proper input value
- ContrastValue
  - User defined contrast input between -255 and 255
  - Check proper input value
  - Formula:
    - $factor = \frac{259*(Contrast + 255)}{255*(259 - Contrast)}$
    - $ContrastValue = factor * (Value_{old} - 128) + 128$

Watermark

Original

With Watermark

```
IMAGE *Watermark(IMAGE *image, const IMAGE *watermark_image);
```

- If $pixel_{template}(x, y)$ is black (0, 0, 0), then

$$pixel_{watermark}(x, y) = pixel_{original}(x, y) * 1.45$$

- Don't forget to:
  - Check that the new pixel value ≤ 255;
  - Crop or tile the watermark to the image if necessary.

Extend the Makefile

- Extend the `Makefile` to compile the new module `image.c`
  - `PhotoLab`: with interactive menu and `DEBUG` mode off
  - `PhotoLabTest`: only calls `AutoTest` and enables `DEBUG` mode

Valgrind

- `Valgrind` is a multipurpose code profiling and memory debugging tool for Linux.
- `valgrind --leak-check=full program`
- Run your program in `Valgrind`'s environment and `Valgrind` will check for memory leaks in your program.
- You need to compile your program with `-g` option in `gcc` to enable detection of memory leak.
- For your final submission, your program should be free of warnings and errors reported by `Valgrind`.

Script File

- Start the script by typing the command: `typescript`
- Compile `PhotoLab` by using `Makefile` and run the executable `PhotoLab`
- Choose “Test all functions”
- Exit the program
- Compile `PhotoLabTest` by using `Makefile`
- Run `PhotoLabTest` under `Valgrind`
- Delete all object files, generated `ppm` files and executables by using `Makefile`
- Stop the script by typing the command: `exit`
- Rename the script file to `PhotoLab.script`

Submission

- Due: Wednesday Nov 22 at 6:00pm
  - `PhotoLab.c`
  - `PhototLab.script`
  - `PhototLab.txt`
  - `Image.c`
  - `Image.h`
  - `Constants.h`
  - `DIPs.c`
  - `DIPs.h`
  - `FileIO.c`
  - `FileIO.h`
  - `Advanced.c`
  - `Advanced.h`
  - `Makefile`