

Lab: Part 2

The Need for Speed: Image Manipulation in C

2023/04/10 ~ 2023/05/01

Username

- Username not being the student ID will fail during grading

The screenshot shows a web browser at the URL `https://teaching.csap.snu.ac.kr/daon/part-1`. The browser's address bar and the GitLab interface are visible. The GitLab page is for a project named "Part 1 - Image Manipulation in Python" by Daon Park. The project ID is 861. It shows 6 commits, 2 branches, 0 tags, and 12.8 MB of project storage. A message indicates it was forked from an inaccessible project. A commit by Bernhard Egger is highlighted, with the message "Fix error introduced with previous commit" and a commit hash of d608f979. Below the commit, there are buttons for "Find file", "Web IDE", "Clone", and "Add Wiki". At the bottom, there is a table with columns for "Name", "Last commit", and "Last update".

Daon Park > Part 1 - Image Manipulation in Python

Part 1 - Image Manipulation in Python

Project ID: 861

6 Commits 2 Branches 0 Tags 12.8 MB Project Storage

Forked from an inaccessible project.

Fix error introduced with previous commit
Bernhard Egger authored 5 days ago d608f979

main part-1 / +

Find file Web IDE Clone

README Auto DevOps enabled Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster

Add Wiki Configure Integrations

Name	Last commit	Last update
------	-------------	-------------

Fork

- Set the visibility to private so that others cannot see your code
- Do not change your project name

The screenshot shows a web browser at the URL <https://teaching.csap.snu.ac.kr/daon/part-1>. The page is a GitLab project view for 'Part 1 - Image Manipulation in Python' by Daon Park. The project ID is 861. It shows 6 commits, 2 branches, 0 tags, and 12.8 MB of project storage. A message indicates it was forked from an inaccessible project. A recent commit by Bernhard Egger is shown with the message 'Fix error introduced with previous commit'. The interface includes buttons for 'Find file', 'Web IDE', 'Clone', and options to add a README, LICENSE, CHANGELOG, CONTRIBUTING, or Kubernetes cluster. A table at the bottom has headers for 'Name', 'Last commit', and 'Last update'.

Daon Park > Part 1 - Image Manipulation in Python

Part 1 - Image Manipulation in Python

Project ID: 861

6 Commits 2 Branches 0 Tags 12.8 MB Project Storage

Forked from an inaccessible project.

Fix error introduced with previous commit
Bernhard Egger authored 5 days ago d608f979

main part-1 / +

Find file Web IDE Clone

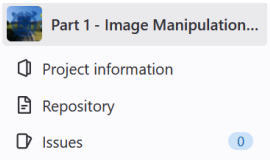
README Auto DevOps enabled Add LICENSE Add CHANGELOG Add CONTRIBUTING Add Kubernetes cluster

Add Wiki Configure Integrations

Name	Last commit	Last update
------	-------------	-------------

Tag

- Create "Submission" tag
- Check for spelling error(s) in the tag



Submission

Commit and push your work frequently to avoid data loss. When you are ready to submit your code for grading, create a tag called "Submission". The timestamp of the "Submission" tag is considered your submission time.

To create a tag, visit the repository on GitLab and navigate to Repository -> Tags. Enter "Submission" as the Tag name then hit "Create tag". You can leave the other fields empty.

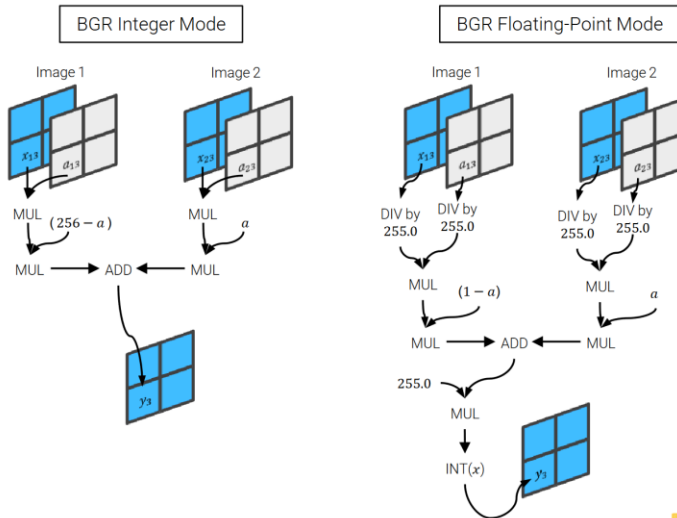
Fixed-Point Multiplication

- The implementations are completely up to you

Image Blend

- Merge

- Merge two input pixels with its alpha-channel values and the alpha parameter
- Integer mode will require fixed-point arithmetic



Fixed-Point Multiplication

- The handout literally told you everything

- **Floating-point code**

The images store the intensities as 8-bit integers ranging from 0 to 255. You need to normalize all channels of a pixel by dividing the integer value by 255 before performing your calculations. The alpha parameter is provided as a float in the range 0.00 to 1.00, so you do not have to normalize it anymore.

- **Fixed-point code**

To avoid the expensive floating point operations, many image manipulation libraries offer a faster but slightly inaccurate fixed-point implementation.

In our fixed-point variant, we use 1.8 bit fixed-point values to represent a number range from 0.00000000_2 to 1.00000000_2 . To avoid expensive scaling operations, we interpret the 8-bit pixel values directly as 1.8-bit fixed-point numbers, yielding a range from 0.00000000_2 to 0.11111111_2 . Since the first bit is always zero, we can drop the leftmost bit and store the fixed-point values of a pixel in four 8-bit characters. Since we are unable to represent 1.0, we lose a little bit of intensity when calculating with fixed-point numbers. As a result, the result of the floating-point code and the integer fixed-point variant will not be bit-identical. When calculating with fixed-point numbers, remember what you have learned about fractional binary numbers, in particular, what happens to the radix point when multiplying numbers. Make sure to add/subtract only values that have the radix point in the same bit position and convert the number back to our .8-bit format before storing the values in the output pixel.

Lab 2

- Project structure

```
part-2
|-> blend_driver.c
|-> blend_float.c
|-> blend.h
|-> blend_int.c
|-> blur_driver.c
|-> blur_float.c
|-> blur.h
|-> blur_int.c
|-> imlib.c
-> imlib.h
```

- Commands

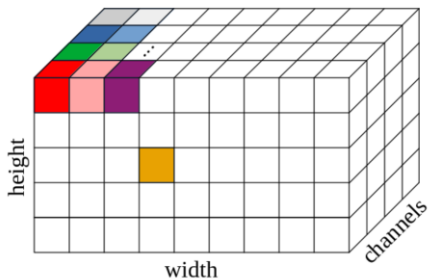
```
// compile
make all

// blur
./blend_driver [-h]
                    [--type {int,float}]
                    [--mode {overlay,merge}]
                    [--alpha ALPHA]
                    [--output OUTPUT]
                    image1
                    image2

// blend
./blur_driver [-h]
                    [--type {int,float}]
                    [--kernel {3x3,5x5,7x7}]
                    [--output OUTPUT]
                    image
```

Raw Format

- **Raw image input(s) only**
 - Reads in raw image as the 1D format instead of 3D format



↓ flatten



- **Macros**
 - `PIXEL()`
 - `INDEX()`

Dynamic Allocation

- **Dimension calculations**

- First, need to know the dimensions to properly calculate how much to allocate

- **Memory allocation**

- Allocate memory with `malloc()` or `calloc()`

- **Error checking**

- Always check if there was an error after memory allocation

- **Memory freeing**

- Once the data structure is not in use anymore, free the memory

Performance

- **Time measurements**

- Warm-ups (usually for GPUs, but why not?)
- Multiple iterations (at least 10 times)
- Leave out the best and the worst performance