karansher / **computer-graphics-raster-images** Public

forked from alecjacobson/computer-graphics-raster-images

Computer Graphics Assignment about Raster Images

2 stars 40 forks

Star Watch

Code Issues 47 Pull requests Actions Projects Security Insights

master

This branch is 7 commits ahead of alecjacobson:master.

abhimadan Another update ... on Sep 15 64

View code

Computer Graphics – Raster Images

> **To get started:** Clone this repository:
>
> ```
> git clone  https://github.com/karansher/computer-graphics-raster-images.git
> ```

Do not fork: Clicking "Fork" will create a *public* repository. If you'd like to use GitHub while you work on your assignment, then mirror this repo as a new *private* repository: https://stackoverflow.com/questions/10065526/github-how-to-make-a-fork-of-public-

README.md

Introduction

Welcome to Computer Graphics! The main purpose of this assignment will be to get you up and running with C++ and the cmake build setup used for our assignments.

For this assignment, and all future assignments, please read the following:

- this entire document (the README)
- the comments in the header files (the .h files located in the `include` folder)

Some articles linked here may also contain useful information that you should read, so use your judgement there.

Prerequisite installation

On all platforms, we will assume you have installed cmake and a modern c++ compiler on Mac OS X[1], Linux[2], or Windows[3].

We also assume that you have cloned this repository using the `--recursive` flag (if not then issue `git submodule update --init --recursive`).

Layout

All assignments will have a similar directory and file layout:

```
README.md
CMakeLists.txt
main.cpp
include/
  function1.h
  function2.h
  ...
src/
  function1.cpp
  function2.cpp
  ...
data/
  ...
...
```

The `README.md` file will describe the background, contents and tasks of the assignment.

The `CMakeLists.txt` file setups up the cmake build routine for this assignment.

The `main.cpp` file will include the headers in the `include/` directory and link to the functions compiled in the `src/` directory. This file contains the `main` function that is executed when the program is run from the command line.

The `include/` directory contains one file for each function that you will implement as part of the assignment. ***Do not change*** these files.

The `src/` directory contains *empty implementations* of the functions specified in the `include/` directory. This is where you will implement the parts of the assignment.

The `data/` directory contains *sample* input data for your program. Keep in mind you should create your own test data to verify your program as you write it. It is not necessarily sufficient that your program *only* works on the given sample data.

Compilation

This and all following assignments will follow a typical cmake/make build routine. Starting in this directory, issue:

```
mkdir build
cd build
cmake ..
```

If you are using Mac or Linux, then issue:

```
make
```

If you are using Windows, then running `cmake ..` should have created a Visual Studio solution file called `raster.sln` that you can open and build from there. Building the raster project will generate an .exe file.

Why don't you try this right now?

Execution

Once built, you can execute the assignment from inside the `build/` using

```
./raster
```

Background

> Every assignment, including this one, will start with a **Background** section. This will cite a chapter of the book to read or review the math and algorithms behind the task in the assignment. Students following the lectures should already be familiar with this material.

Read Chapter 3 of *Fundamentals of Computer Graphics (4th Edition)*.

The most common digital representation of a color image is a 2D array of red/green/blue intensities at pixels. Since each entry in the array is actually a 3-vector of color values, we can interpret an image as a 3-tensor or 3D array. Memory on the computer is addressed linear, so an RGB image with a certain `width` and `height` will be represented as `width*height*3` numbers. How these numbers are ordered is a matter of convention. In our assignment we use the convention that the red value of pixel in the top-left corner comes first, then its green value, then its blue value, and then the rgb values of its neighbor to the right and so on *across* the row of pixels, and then moving to the next row *down* the columns of rows.

> Q: Suppose you have a 767\times 772 rgb image stored in an array called `data`. How would you access the green value at the pixel on the 36th row and 89th column?
>
> A: `data[1 + 3*(88+767*35)]` (Remember C++ starts counting with `0`).

Alpha map

Natural images (e.g., photographs) only require color information, but to manipulate images it is often useful to also store a value representing how much of a pixel is "covered" by the given color. Intuitively this value (called alpha or α represents how opaque (the opposite of *transparent*) each pixel is. When we store rgb + α image as a *4*-channel rgba image. Just like rgb images, rgba images are 3D arrays unrolled into a linear array in memory.

.png files can store rgba images, whereas our simpler .ppm file format only stores grayscale or rgb images.

.ppm files

We'll use a very basic *uncompressed* image file format to write out the results of our tasks: the .ppm.

Like many image file formats, .ppm uses 8 bits per color value. Color intensities are represented as an integer between `0` (0% intensity) and `255` (100% intensity). In our programs we will use `unsigned char` to represent these values when reading, writing and doing simple operations. For numerically sensitive computations (e.g., conversion between rgb and hsv), it is convenient to convert values to decimal representations using double precision floating point numbers `0` is converted to `0.0` and `255` to `1.0`.

To simplify the implementation and to help with debugging, we will use the text-based .ppm formats for this assignment.

Grayscale Images

Surprisingly there are many acceptable and reasonable ways to convert a color image into a grayscale ("black and white") image. The complexity of each method scales with the amount that method accommodates for human perception. For example, a very naive method is to average red, green and blue intensities. A slightly better (and very popular method) is to take a weighted average giving higher priority to green:

$$i = 0.2126r + 0.7152g + 0.0722b.$$

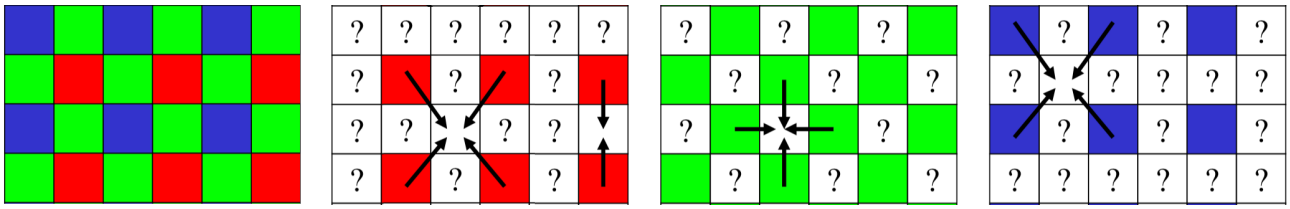
> Q: Why are humans more sensitive to green?
>
> Hint: 🐒

Mosaic images

The raw color measurements made by modern digital cameras are typically stored with a single color channel per pixel. This information is stored as a seemingly 1-channel image, but with an understood convention for interpreting each pixel as the red, green or blue intensity value given some pattern. The most common is the Bayer pattern. In this assignment, we'll assume the top left pixel is green, its right neighbor is blue and neighbor below is red, and its kitty-corner neighbor is also green.

> Q: Why are more sensors devoted to green?
>
> Hint: 🐒

To *demosaic* an image, we would like to create a full rgb image without downsampling the image resolution. So for each pixel, we'll use the exact color sample when it's available and average available neighbors (in all 8 directions) to fill in missing colors. This simple linear interpolation-based method has some blurring artifacts and can be improved with more complex methods.



Color representation

RGB is just one way to represent a color. Another useful representation is store the hue, saturation, and value of a color. This "hsv" representation also has 3-channels: typically, the hue or `h` channel is stored in degrees (i.e., on a periodic scale) in the range $[0°, 360°]$ and the saturation `s` and value `v` are given as absolute values in $[0, 1]$.

Converting between rgb and hsv is straightforward and makes it easy to implement certain image changes such as shifting the hue of an image (e.g., Instagram's "warmth" filter) and the saturation of an image (e.g., Instagram's "saturation" filter).

Tasks

> Every assignment, including this one, will contain a **Tasks** section. This will enumerate all of the tasks a student will need to complete for this assignment. These tasks will match the header/implementation pairs in the `include/` / `src/` directories.

Groundrules

Implementations of nearly any task you're asked to implemented in this course can be found online. Do not copy these and avoid *googling for code*; instead, search the internet for explanations. Many topics have relevant wikipedia articles. Use these as references. Always remember to cite any references in your comments.

White List

Feel free and encouraged to use standard template library functions in `#include <algorithm>` and `#include <cmath>` such as `std::fmod` and `std::fabs`.

`src/rgba_to_rgb.cpp`

Extract the 3-channel rgb data from a 4-channel rgba image.

`src/write_ppm.cpp`

Write an rgb or grayscale image to a .ppm file.

At this point, you should start seeing output files:

- `bayer.ppm`
- `composite.ppm`
- `demosaicked.ppm`

- `desaturated.ppm`
- `gray.ppm`
- `reflected.ppm`
- `rgb.ppm`
- `rotated.ppm`
- `shifted.ppm`

`src/reflect.cpp`

Horizontally reflect an image (like a mirror)

`src/rotate.cpp`

Rotate an image 90^\circ counter-clockwise

`src/rgb_to_gray.cpp`

Convert a 3-channel RGB image to a 1-channel grayscale image.

src/simulate_bayer_mosaic.cpp

Simulate an image acquired from the Bayer mosaic by taking a 3-channel rgb image and creating a single channel grayscale image composed of interleaved red/green/blue channels. The output image should be the same size as the input but only one channel.

src/demosaic.cpp

Given a mosaiced image (interleaved GBRG colors in a single channel), created a 3-channel rgb image.

`src/rgb_to_hsv.cpp`

Convert a color represented by red, green and blue intensities to its representation using hue, saturation and value.

`src/hsv_to_rgb.cpp`

Convert a color represented by hue, saturation and value to its representation using red, green and blue intensities.

`src/hue_shift.cpp`

Shift the hue of a color rgb image.

Hint: Use your `rgb_to_hsv` and `hsv_to_rgb` functions.

`src/desaturate.cpp`

Desaturate a given rgb color image by a given factor.

Hint: Use your `rgb_to_hsv` and `hsv_to_rgb` functions.

`src/over.cpp`

Compute C = A Over B, where A and B are semi-transparent rgba images and "Over" is the Porter-Duff Over operator.

Submission

Submit your completed homework on MarkUs. Open the MarkUs course page and submit all the `.cpp` files in your `src/` directory under Assignment 1: Raster Images in the `raster-images` repository.

Frequently Asked Questions (FAQ)

This section will be updated if there are any important questions brought up in the GitHub issues that can be helpful to other students.

> Q: Can I write helper functions?

A: Yes, as long as you do not make any new header files or source files. The helper functions can only be in the provided .cpp files, since these are the only files you submit.

> Q: Does my output need to be pixel perfect?

A: No. Although you should get pixel-perfect results for the simpler questions, don't worry if, for example, your demosaic output has some minor pixel differences from ours. As long as it looks like the expected image, you should get full marks. That said, differences that you can see without an image diff tool or examining pixel values likely indicate a bug in your code.

> Q: I'm on Windows. Can I use a compiler other than Visual Studio?

A: Although something like WSL or cygwin can work for this assignment, we strongly recommend setting up Visual Studio in order to make your life easier for future assignments. See the notes for Windows users at the bottom of this page.

> Q: I'm on Windows. Do I have to use the Visual Studio IDE?

A: No. You need it for its compiler, so you can theoretically use any text editor you want. Then, you can compile the project from the command line. That being said, compiling Visual Studio on the command line is much less convenient than from the IDE, so if you don't have a strong preference for your text editor we recommend to use the IDE as well.

> Q: I'm on Mac. Do I have to use the XCode IDE?

A: No. We only ask you to install XCode to access its command line tools, not the IDE itself. In fact, by default CMake will not produce an XCode project.

> Q: I clicked on raster.exe in File Explorer on Windows and it crashed! What happened?

A: Clicking on the exe on Windows doesn't work due to a subtle discrepancy with how CMake projects work on Windows vs Mac/Linux. See the notes for Windows users at the bottom of this page.

> Q: I tried running my code in Visual Studio and I got an error message. What's going on here?

A: CMake generates an `ALL_BUILD` target that, when compiled, compiles everything else in the project, but cannot be run. You instead need to change the startup project to the one matching the name of the exe you want to run (see here for instructions). For this assignment you just need to change the startup project to `raster`, but future assignments can have multiple executables, so you may need to change the startup project several times while testing your code.

> Q: My image dimensions are zero / strange numbers. What's going on?

A: This is actually the same issue as the previous question --- you're not running your binary from the correct location. Make sure you're running your code from the `build` folder.

> Q: I opened the project folder in Visual Studio, and I can build the project, but it still crashes. What's going on?

A: Again, this is a relative path issue. Visual Studio is probably reading CMakeLists.txt and setting things up accordingly, but its slightly different from running cmake from the command line. To avoid any confusion, I'd recommend using cmake from the command line and opening the generated .sln file in Visual Studio.

> Q: I can't find cmake from the command line!

A: This means your `PATH` variable isn't set correctly. I think Windows does this properly but some Mac installations don't seem to include it. If you downloaded cmake from their website, it should be located in `/Applications/CMake.app/Contents/bin/cmake`, so use this instead of just `cmake`. For info on setting your PATH variable on Mac/Linux see this link though note that newer Macs use zsh as the default shell so you may need to update your `.zshrc` instead. This shouldn't be a problem on Linux systems if you use your distribution's package manager (see the note for Linux users at the bottom of this page).

Questions?

Direct your questions to the Issues page of this repository.

Answers?

Help your fellow students by answering questions or positions helpful tips on Issues page of this repository.

> [1] Mac Users
>
> You will need to install Xcode if you haven't already to be able to use the C++ compiler.
>
> [2] Linux Users
>
> Many linux distributions do not include gcc and the basic development tools in their default installation. On Ubuntu, you need to install the following packages (more than needed for this assignment but should cover the whole course):

```
sudo apt-get install git
sudo apt-get install build-essential
sudo apt-get install cmake
sudo apt-get install libx11-dev
sudo apt-get install mesa-common-dev libgl1-mesa-dev libglu1-mesa-dev
sudo apt-get install libxinerama1 libxinerama-dev
sudo apt-get install libxcursor-dev
sudo apt-get install libxrandr-dev
sudo apt-get install libxi-dev
```

```
sudo apt-get install libxmu-dev
sudo apt-get install libblas-dev
```

[3] **Windows Users**

Our assignments only support the Microsoft Visual Studio 2015 (and later) compiler in 64bit mode. It will not work with a 32bit build and it will not work with older versions of visual studio. Please do not use WSL/cygwin/other Windows subsystems or compilers, as these all cause issues in later assignments that require OpenGL. Follow the instructions here to install the latest version of Visual Studio, selecting the "Desktop Development with C++" workload.

CMake project conventions are somewhat different for Visual Studio-based builds compared to make-based builds. Rather than the output executables being in the `build` folder, they are instead in the `build\Release` or `build\Debug` folder, depending on which configuration you are currently using. As a result, the image file paths baked into the executable (which assume the executable is run from `build`) will be broken if you run the exe from the `Release` or `Debug` folders (which is what happens when you click on it in the File Explorer). Please either run your code from Visual Studio directly, clicking on the green arrow button that says something like "Local Windows Debugger", or run your exe from the command line in the `build` directory (e.g., `.\Debug\raster`).

Releases

No releases published

Packages

No packages published

Languages

C++ 73.3% **C** 25.1% **CSS** 1.5% **CMake** 0.1%