Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

Read the guide

alibukhari6728 / **Lab1_Synthetic_Lighting** Private

master 1 branch 0 tags Go to file Add file Code

ZoraizQ fix: link re... 4c5c02b 1 hour ago 18 commits

img	Add files via upload	7 days ago
manual_im...	Add files via upload	7 days ago
.gitignore	fix: opencv dependency to ...	7 hours ago
README....	fix: link replace	1 hour ago
filters.cpp	Add files via upload	7 days ago
main.cpp	fix: opencv dependency to ...	7 hours ago
morphing....	Update morphing.cpp	6 days ago

README.md

About

No description, website, or topics provided.

Readme

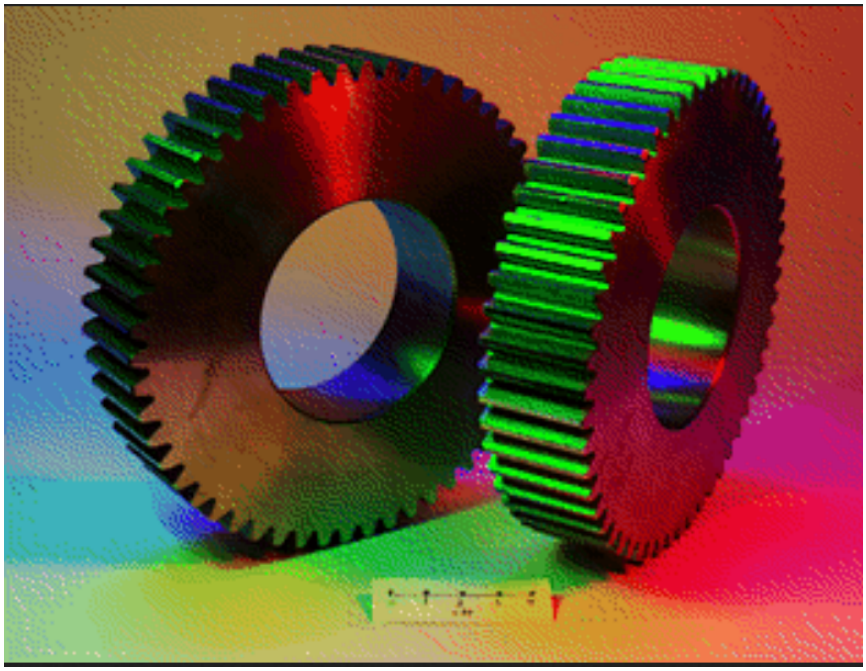
Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 2



Lab 1: Synthetic Lighting

In this lab we will learn how to (mathematically and programmatically) post-process photographs of scenes in which the contribution of each light source is known. We will be using image manipulation techniques to create interesting visual effects by changing the color and contribution of light sources in a scene.

Logistics

Submission

There's a 20 minute slot for each part. You need to copy and rename the relevant cpp file to *partX.cpp* where X is the part that you're trying to submit. Upload this file before the 20 minute slot expires. The solution will be released as soon as the allocated 20 minutes are over. You can copy and paste it into the working directory before proceeding to the next part (in case you're not sure about your solution).

Getting started

alibukhari6...

ZoraizQ Zor...

Languages

C++ 100.0%

You can either download the zipped assignment straight to your computer or clone it from GitHub using the command

```
https://github.com/cs452lums/lab0.git
```

Setup

You must have the latest version of OpenCV (C++) installed on your device.

Using the GUI

You can run the executable in the Lab1_Synthetic_Lighting directory with the following command

```
g++ main.cpp -o output `pkg-config --cflags
--libs opencv4`;./output X
```

where the final 'X' is to be replaced by 1,2,3 or 4 (the part you want to run).

Once you run a part, you can hover your cursor on the produced image window to see different variations.

Project structure

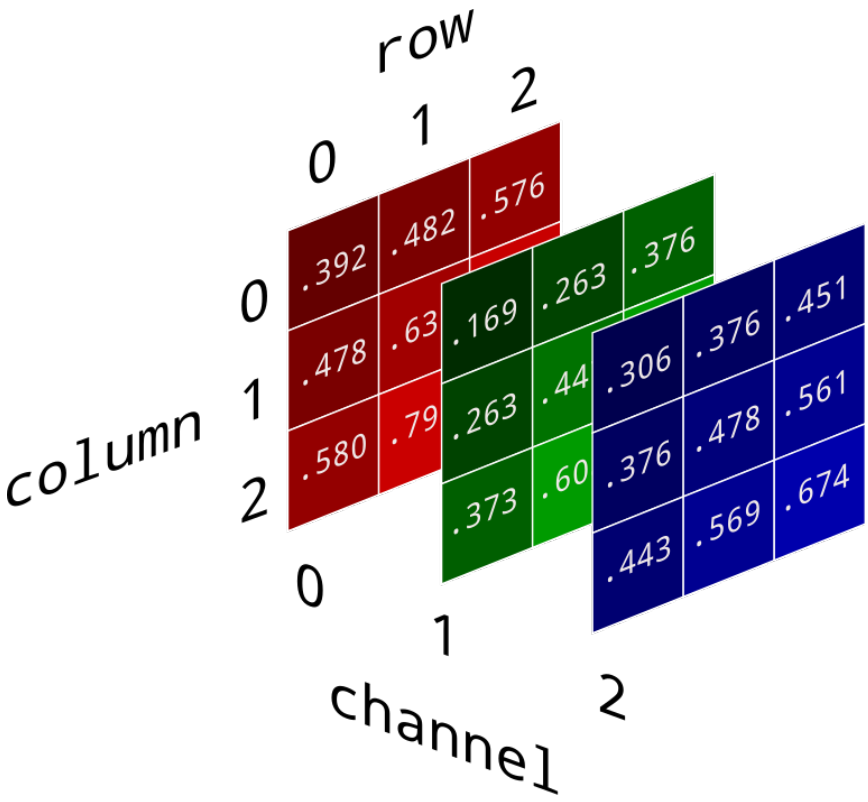
- Part 1: Light Morphing
- Part 2: Color Filters
- Part 3: Colored Lights
- Part 4: Colored Morphing

Here is a very brief sketch of what happens when you run the command. The number at the end of the command is collected as an argument which is then used to decide which one of the functions (each part corresponds to a function) should be called. The image is then rendered onto the window and the x coordinate of the cursor (when on the image window) is collected. This value of the x coordinate is then used to compute the weight(in morphing) or the type of filter(in filters).

Quick Background

Digital images are essentially large matrices that contain the color values of each pixel displayed on a screen. Each cell of the matrix, thus, represents a pixel and corresponds to 3 values (R,G,B), each of which varies between 0 and 255.

For a detailed look on how it works, you may want to take a look at the following article: RGB Model.



Part 1: Light Morphing

Morphing is a special effect that changes one image or shape into another through a seamless transition. In this part we will be morphing between the following two images:

Go to the `morphing()` function in *morphing.cpp*.

Given the two input images `inp1` and `inp2` and the contribution `w` of the first image, use the following equation to compute and return the morphed image.

```
It = (w*Ia) + ((1-w)*Ib)
```

where `It` is the morphed image, `Ia` and `Ib` are the input images and `w` is the weight that is varied between 0 and 1.

Mat operations:

- To take the element-wise sum of two matrices, you can simply use the `+` operator.

  Mat mat3 = mat1 + mat2;

- To multiply all elements of a matrix by a scalar, you can simply use the `*` operator.

  Mat mat2 = s * mat1;

To test your implementation, run the following command:

```
g++ main.cpp -o output `pkg-config --cflags
--libs opencv4`;./output 1
```

Now move your cursor horizontally on the image. If implemented correctly, you will observe the transition from one light to another as you move your cursor.

The transitional states should look something like:

Part 2: Color Filters

Color filters are widely used in applications ranging from professional post-processing (e.g. on Adobe Lightroom) to day to day image alterations (e.g. on instagram). In this part we will try to recreate a few basic image filters.

Go to `color_enhancement_filter()` function in *filters.cpp*.

Given an `image` and a `color`, loop over the entire image and use the following equation over each pixel to apply the filter on the image:

```
Px[color] = Px[color] + ((255 -
Px[color])/4)
```

where `Px` is an individual pixel in the given image (in Vec3b format) and `color` is an integer that represents the color of the filter to be applied (0,1,2 coorespond to B,G,R respectively).

To extract an individual pixel pixel from the image, use the following command:

```
Vec3b pixel = Im.at<Vec3b>(Point(x,y));
where Im is the given image, and (x,y) are
the coordinates of the pixel to be
extracted.
```

To test your implementation, run the following command:

```
g++ main.cpp -o output `pkg-config --cflags
--libs opencv4`;./output 2
```

Now move your cursor horizontally on the image. If implemented correctly, you will observe three different filters as you move your cursor.

The filtered images should look something like:

Bonus1:

If you're done early, use your visual perception to create the following filters:

Did you notice something unintuitive?

Part 3: Colored Lights

Now we will attempt to change the color of the light sources. We can do this by removing all other colors from the image.

Go to `colored_light()` function in *filters.cpp*.

Given an `image` and a `color`, loop over the entire image and apply the following scheme over each pixel to change the color of the light source.

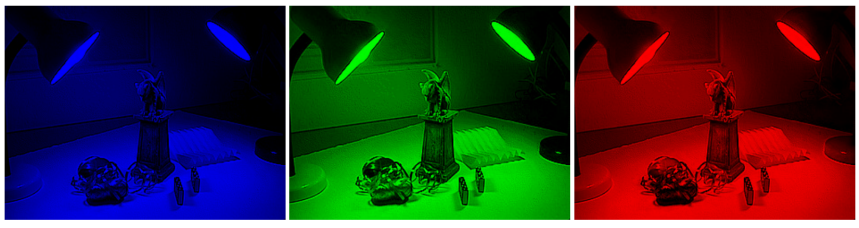
```
If the given color is c, change the value
of the other two color components to zero,
while keeping the values at c the same.
```

To test your implementation, run the following command:

```
g++ main.cpp -o output `pkg-config --cflags
--libs opencv4`;./output 3
```

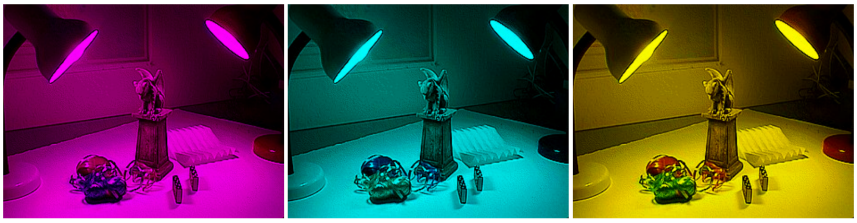
Now move your cursor horizontally on the image. If implemented correctly, you will observe three different colors of light sources as you move your cursor.

The colored lights should look something like:



Bonus2:

If you're done early, use your visual perception to create the following filters:



Part 4: Colored Morphing

Finally, we can put it all together to morph between two differently colored light sources.

Go to `filtered_morphing()` function in *morphing.cpp*.

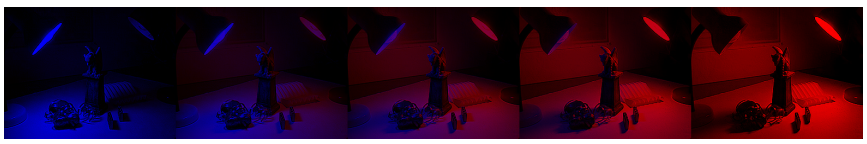
Given the two input images and a weight, change the color of the light sources to **red** and **blue** and perform morphing on the resulting images. You are encourged to use the funtions that you've already created (i.e. `colored_light()` and `morphing()`).

To test your implementation, run the following command:

```
g++ main.cpp -o output `pkg-config --cflags
--libs opencv4`;./output 4
```

Now move your cursor horizontally on the image. If implemented correctly, you will observe a transition between two differently colored light sources as you move your cursor.

The transition should look something like:



In case you face compilation errors (having a different opencv version than 4):

Open main.cpp and press Ctrl+H (Replace All), replacing cv::IMREAD_COLOR with CV_LOAD_IMAGE_COLOR

Then, for all of the commands you run use 'opencv' instead of 'opencv4' e.g.

```
g++ main.cpp -o output `pkg-config --cflags
--libs opencv`;./output 3
```

What's next?

A short graded assignment will be released in a few hours.