SIPL: Simple Image Processing Language

Shanshan Zhang, Yihan Zhao, Yuedong Wang, Ci Chen, Simon Zhai
{sz2648, yz2996, yw2931,cc4192, yz3116}@columbia.edu

September 26, 2017

Contents

1 Proposal

Simple Image Processing Language is a language specifically targeted for image processing. We intend on implementing a language that can deal with images in an effective and fast way. It simplifies the implementation of image processing algorithm compared to using languages like C, C++ by converting image into a **Image**, which is represented by channels, each of which is a primitive data type - **Mat**. Mat is a 2d matrix and all other operations are based on this data type, which makes SIPL simple and compact.

SIPL not only supports basic calculation for images, but also supports more advanced image manipulations. For example, users could ask to resize, flip and rotate the images. Also, our language allow users to change an image into grey, detect the edge of an image and separate the image into three-primary colors.

2 Feature

SIPL simplify the programming of image Processing. There is no need to define the size of image, SIPL would compute the size and allocate memory automatically.

3 Example Syntax

ThThe goal of SIPL language is to convert images into a 3d matrix and complete all the image operation on the 3d matrix using mathematical methods. For example, for every input image, we will save the image into a matrix like this: img = (width, height, n), where n = 1, 2, 3, which represents the default three channels (red, green and blue) of the image. And all the following operations we are dealing with are just simple mathematical operation on the image matrix.

3.1 Data Types

Image Represented by channels (e.g. Red, Green, Blue), each channel is a matrix.
Mat Matrix data type.

3.2 Image I/O

The following 3 functions describe the input and output of an image.
Image img = readImg('img1.png')//read an image and represented by RGB channels.
Image showImg(img) // display an image to the user.
Image writeImg(img, ' /directlry/FileName'); // write image to file.

3.3 Basic Calculation

3.3.1 Addition

Image plusedImage = img + n; // every pixel plus n, which will enhance brightness.
Image addedImage = img + img2; // the pixels addition between two images.

3.3.2 Subtraction

Image minusImage = img - n; // every pixel minus n, which will decrease brightness.
Image minusImage = img - img2; // the pixels subtraction between two images.

3.3.3 Multiplication

Image multipliedImage = img×n; // every pixel of the image multiply a value, e.g. n.
We create a multiplication operation here to make the operator system complete, though there is no reason for making multiplication on pixels at all.

3.3.4 division

Image dividedImage = img / n; // every pixel of the image divided a value, e.g. n.
Again, this operator barely has meaning for image processing, we just want to make the operator system complete.

3.4 Image Manipulation

3.4.1 Gray Conversion

img.grey := greyImg(img); // convert an image into grey picture.
Here we define a grey channel for this image using greyImg() built-in function.

3.4.2 Image Rotation

Image rotatedImage = rotateImg(img, n);
// counterclockwisly rotate an image by $\pm\frac{\pi}{2}$(n=1 for $\frac{\pi}{2}$, n =-1 for $-\frac{\pi}{2}$, otherwise cast an error).

3.4.3 Image Flipping

Image flipedImage = flipImg(img, n);
// flip an image(n=1 for horizontal flip and n=-1 for vertical flip, otherwise cast an error).

3.4.4 Matrix Construction

Mat kernel1 = $[a_{ij}]_{n\times n}$;
Mat kernel2 = $[b_{ij}]_{n\times n}$;
Mat kernel = kernel1 ** kernel2; convolution of two kernels.
Also we expected to create a 2 dimensional gaussian kernel for blurring:
Mat GaussianKernel = N(σ) //Since the Gaussian kernels used for image processing are always centered, so we only care about the variance in the kernel.

3.4.5 Image Filtering

Image filteredImage = convImg(img, kernel);
// provides a convolution on given image, kernels and their parameters TBD.

3.4.6 RGB Channelling

imgRed = img.red // the red channel of img
imgGreen = img.green // the green channel of img
imgBlue = img.blue // the blue channel of img
// separate the image into 3 red blue green(color = 'red', 'blue', 'green')

3.4.7 Image Resizing

Image resizedImage = resizeImg(img, width, height);
// resize the image by stretching it horizontally or vertically eg. by setting width = 2, height = 0.5, we can get an image with have of its original height and twice of its original width.

3.4.8 Edge Detection

Image edges = EdgeDetect(img);// detect the edges of an image.

4 Sample Code

```
Image Img = readImg('img1.png');
// import an image for operation
Img = rotateImg(Img, 1);
// rotate this image
Img = img + 2;
// enhance the brightness of the rotated image.
Mat kernel = N(2);
// create a gaussian kernel
Img = convImg(img, kernel);
// we convolve the image using gaussian kernel, which will blur the image
showImg(Img);
// here we want to check the modified img
writeImg(Img,'~/MyDirectory/modified image.png');
// save the modified image into a directory named 'modified image.png'.
```

5 Reference

linear filters (Convolution):
http://docs.opencv.org/master/d4/dbd/tutorial_filter_2d.html
Canny Edge Detector:
http://docs.opencv.org/master/da/d5c/tutorial_canny_detector.html
Basic Thresholding Operations:
http://docs.opencv.org/master/db/d8e/tutorial_threshold.html