EZIP: Easy Image Processing

Kevin Chiu | Tejas Nadkarni | Swati Kumar | Avanti Dharkar

{kgc2113, tgn2104, sak2144, ad2518}@columbia.edu

September 24, 2007

In this project, we are planning to design and implement a language for image processing, which can perform arbitrary convolutions and various types of image arithmetic.

1 Introduction

It does not come as a surprise that we are surrounded by images. We use images for communication, transmitting information, creation of fiction etc. . In short we need image processing to understand, analyze and even change the world around us. Computer Vision techniques are widely applied in television, medicine, and even Hollywood movies.

We have developed efficient technologies to process digital images; however, Computer Vision remains a challenging domain. A deep understanding of the difficulties in the existing best practices has led us to the development of a suitable language for image processing. We call it EZIP, or Easy Image Processing.

2 Background

Image processing operations like smoothing and edge detection, and many more are very widely used in areas like Computer Vision. These operations are generally performed on images represented as matrices. We need a language using which can perform matrix operations like addition, subtraction, multiplication and convolution simply and easily.

The inspiration for our language stems from our own experiences with writing computer vision algorithms in C and C++. We noticed that while implementing simple image processing algorithms like blurring, edge detection, smoothing etc. most of our time was spent in writing the code rather than developing effective algorithms. In C and C++, the traditional implementation language for Computer Vision algorithms, the user is left to handle matrix operations using loops and conditional statements that make the code convoluted and difficult to read and understand.

The goal of our language is to provide the user with easy matrix manipulation techniques required for image processing. Generally image processing algorithms have an image and a kernel that works on that image. Our language is specifically designed to elegantly handle various kernel operations on the image and deliver an output image in a user-friendly manner. The user no longer has to worry about numerous unintuitive and error prone constructs, he only has to think about the operations he wants to perform on the image.

3 Related Works

3.1 Matlab

Matlab is a high-level language and interactive environment that enables the user to program complicated algorithms faster than with traditional programming languages such as C, C++, and Fortran. The image processing add on provides powerful libraries for image processing. It has native support for the class of Computer Vision algorithms that involve kernel convolutions and is also adeptly suited to performing global image operations such as histogram equalization and contrast enhancement.

However, Matlab is targeted at users working in the commercial or research fields who have plenty of funds to afford it. The cost of an individual license of Matlab exceeds the budget for many individual users. For an average individual looking to perform some basic image processing, Matlab is not a feasible option. Our language provides a suitable alternative for such users. At no cost, the user can perform basic image processing using few lines of code. The user need not even have much programming background since the syntax is intuitive.[1]

3.2 C/C++

It is well established that C and C++ are extremely powerful and flexible languages for performing computationally intensive numerical operations. An experienced programmer can use C/C++ to implement complex image processing algorithms from first principles. However, implementation is not such a trivial matter for people with lesser programming skills. EZIP, although not as flexible as C/C++, is better suited to developers who need to quickly program Computer Vision algorithms without having to implement low level, elementary functions.

4 Goal

The goal of our language is to provide a simpler, more programmer-friendly way of doing image processing. We will make image processing accessible to the masses.

4.1 Ease-of-Use and Freedom

The most common use cases will be de-noising, edge detection, and image enhancement. Most of these complex operations will be accomplishable with only a few lines of code in our language. EZIP is a modular language that is consistent with standard methods and naming of matrix operations. It has a clear and concise way of defining matrix operations relevant to image processing and a user familiar with image processing can look at the code and understand the function of the program. Common data structures, such as kernels, will also be simple to create and straight forward to use.

Our language and implementation will be free to use and modifiable by anyone under an open source license, such as the BSD or MIT license. The combination of power and freedom will ensure that our language offers an attractive alternative to existing methods. EZIP is also platform agnostic. It is a translated language targeted to Java, which runs on the vast majority of platforms available today.

5 Main Language Features

In this section we describe some of main features in our language.

5.1 Image Operations

Basic image operations should be supported, e.g., convolution, addition, subtraction, division, multiplication, resizing.

5.2 Kernels

Our language will include a set of built-in kernels that are commonly used in image processing, such as the gaussian kernel, typically used for blurring images, and the Mexican hat kernel, typically used for finding edges. The user will also be able to define custom kernels.

5.3 Program Flow Control

The main statements of program flow control should be implemented, including if-else, while, and break.

5.4 Internal Functions

The minimum set of internal functions includes print, view, copy, load, and save. Print will print ASCII representations of objects to the command line. View will show a graphical representation of the image passed into it. Copy, load, and save will handle making copies of images in between variables, loading images from, and saving images to, the disk.

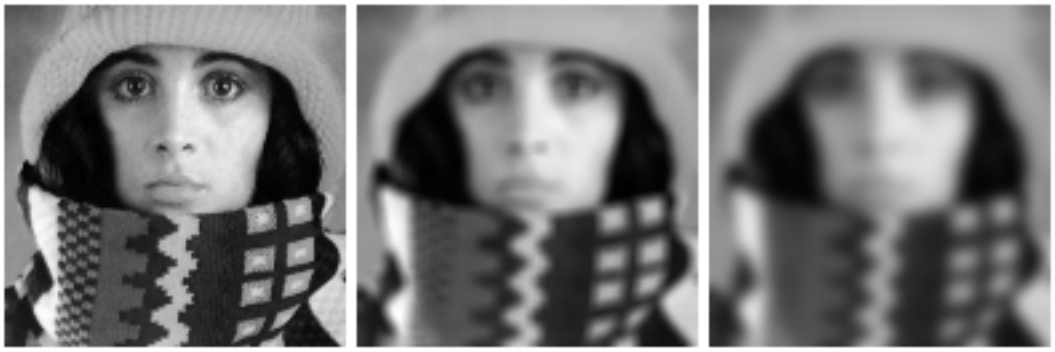

Figure 1: An example image blurred multiple times.[2]

5.5 Code Sample

This sample code implements a blurring function similar to the one illustrated in Figure 1. The code gives an example of how we will comment code, make image declarations, make kernel declaration, and perform convolution. The syntax may change during implementation.

```
// blur image three times and then display

Image i = load("myimage.jpg"); Kernel k = {0,1,0; 1,4,1; 0,1,0};

Image smoothed = i; int j = 0; while(j < 3){ smoothed = smoothed ** k; j = j+1; }

view(smoothed);
```

References

[1] Matlab, http://www.mathworks.com/

[2] Image Processing Fundamentals, http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip-Smoothin.html