COMS W4115: Programming Languages and Translators

MATLIP: MATLAB-Like Language for Image Processing

Project Proposal

Pin-Chin Huang (ph2249@columbia.edu)
Shariar Zaber Kazi (szk2103@columbia.edu)
Shih-Hao Liao (sl2937@columbia.edu)
PoHsu Yeh (py2157@columbia.edu)

September 24, 2008

1. Introduction

A picture is worth a thousand words. Before any human language was invented, people used graphs and pictures to convey messages. Even nowadays, more and more people are using photos as an essential part of their diaries or blogs, thanks to the inexpensive digital cameras that gained popularity since year 2000. Computer graphics technology has also evolved so much that digital image processing has become an integral part of our daily lives.

We are proposing to develop a simple, portable, and easy-to-use image-processing language that is based on the succinct syntax of MATLAB and the portability of Java. We call it MATLIP: a MATLAB-Like Language for Image Processing.

2. Motivation

Our motivation to create MATLIP is to create a portable, low-cost, yet easy-to-use language for image processing. Most of the 2D image-processing languages in the market, such as MATLAB, are not portable from one platform to another. However, the popularity of PDAs, cell phones and digital cameras makes MATLIP, a portable language, desirable for those who would like to do simple pictures editing anywhere. They can write simple code to choose the suitable hues for the background, to rotate the pictures, or to adjust brightness and contrast right in a bus station or on a train with their own small electronic devices, instead of waiting until they have access to a PC at home.

The other problem with the current image processing languages out there is the high cost of obtaining the license and the complexity of syntax involved. For example, the license cost of MATLAB exceeds the budget of most individual users, and to use C/C++ or Java for image processing calls for sophisticated programming skills and even object-oriented concepts. Our goal is thus to design an easy-to-use language which includes all the basic functionalities of image processing so that people can use it with comfort and, most importantly, free of charge. They can easily download our language – MATLIP – from the Internet and can learn it in an intuitive way.

3. Language Features

Here we briefly describe the three most important features that constitute our language.

3.1 Image File Operations

Basic image file operations, such as read, write, show, print, and copy, should be supported by MATLIP. Read will load an image from disk; write will save an image to disk; show will display a graphical representation of the image onto the screen; print will print the image using the local printer, and copy will make a new copy of an image and assign it to another variable.

3.2 Image Processing

MATLIP should have some basic, built-in image processing functions, such as rotate, add, subtract, divide, multiply, and resize. In addition, we are also planning to make use of an existing collection of Java image processing filters and make them available to our language. Examples are color adjustment filters, distortion and wrapping filters, effect filters, texturing filters, blurring and sharpening filters, and edge detection filters.

3.3 Program Flow Control

MATLIP should also support some frequently-used program flow constructions, such as if-elseif-else, for-loop, and while-loop. The syntax should resemble that of MATLAB.

4. Examples of Syntax

We provide two simple code examples of our language. The first example illustrates how we will comment code (using %), read an image from disk, write an image to disk, and show an image on the screen. The second example illustrates how the if-else and for-loop flow control syntax will look like. The input/output images of the two examples are shown in Figure 1 and Figure 2, respectively.

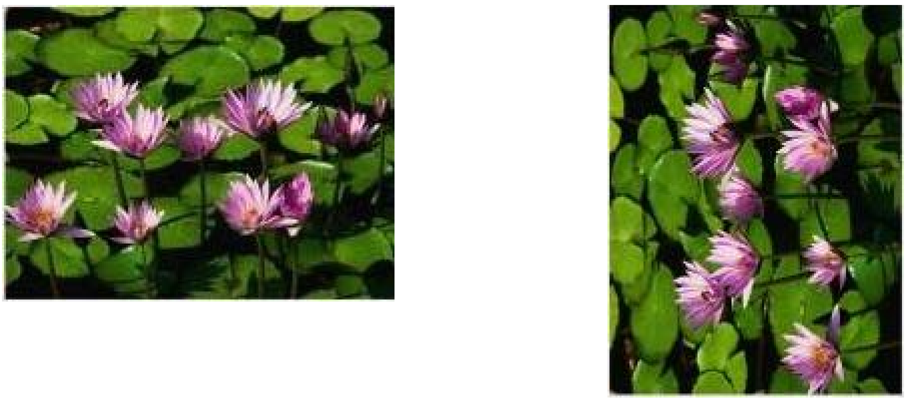

Figure 1: Rotating an image

```
% load an image from disk
I = imread('lotus.png');
% create a window to display the image
figure;
% show the matrix as image in the window
imshow(I);
% rotate the image for 90 degrees
I2 = rotate(I, 90);
figure;
imshow(I2);
% save the new image to disk
imwrite (I2, 'lotus2.png');
```

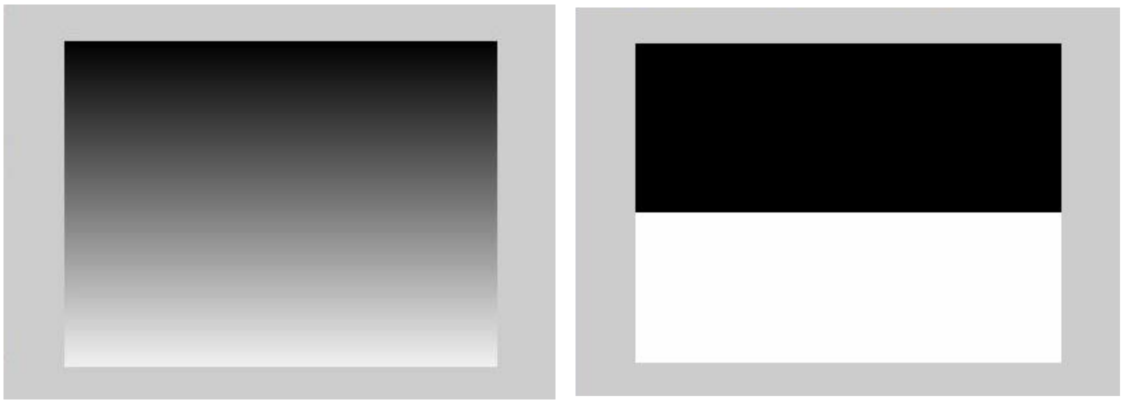

Figure 2: Generating continuous gradient and 2-step gradient

```
% The following nested for loop will generate a continuous gradient
```

```
for i=1:240
      for j=1:320
             zeros(i,j)=i;
      end
end
% create a window to display the image
figure;
% show the matrix as image in the window
imshow(zeros);
% The following nested for loop will generate a 2 step gradient
for i=1:240
      for j=1:320
             if zeros(i,j) < 128
                    zeros(i,j)=0;
             else
                    zeros(i,j)=255;
             end
      end
end
figure;
imshow(zeros);
```