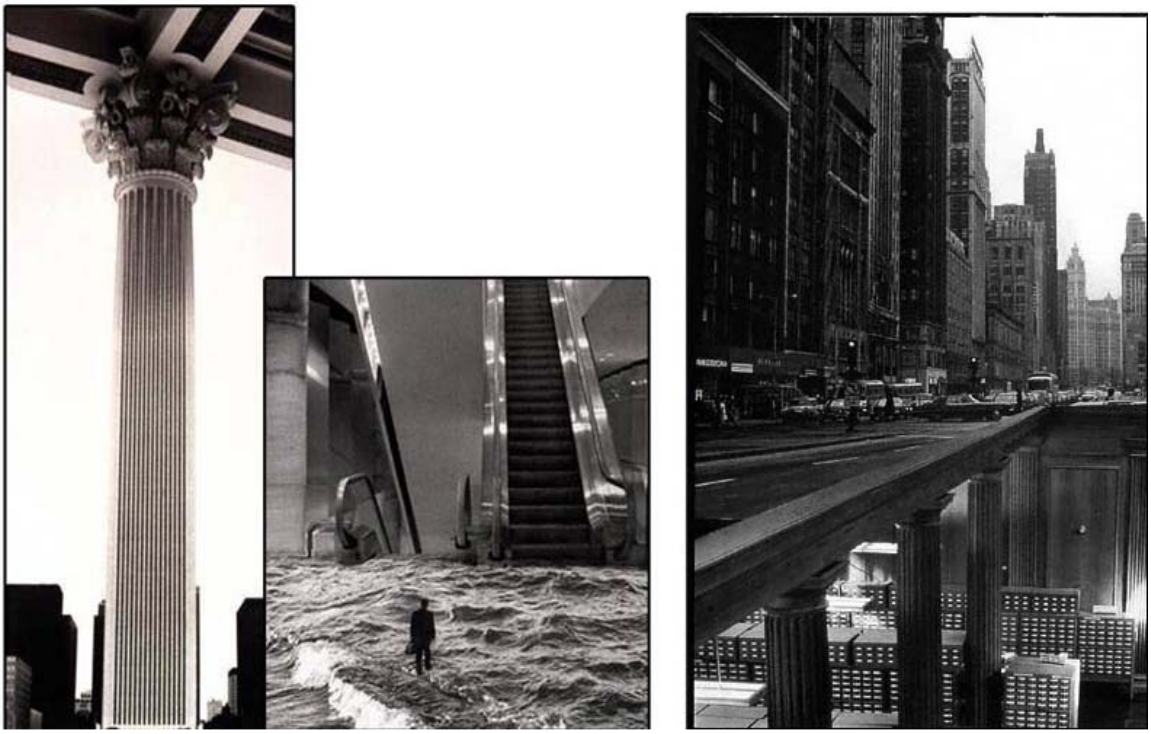

Column Escalator Library

Photogram

Programmable Photoshop® Language

Whitepaper

Ohan Oda **(oo2116@columbia.edu), Group Leader**
Neesha Subramaniam **(ns2295@columbia.edu)**
Richard Ng **(rjn2003@columbia.edu)**
Seikwon Kim **(sk2617@columbia.edu)**

Introduction

Adobe® Photoshop®, the professional image-editing standard and leader of the Photoshop digital imaging line, has been the most prominent and powerful image-editing tool over the years. Although the software is very powerful, due to its complexity and overwhelming graphic user interface, it is difficult for the users to achieve the results they want without being very experienced. Additionally, even though Photoshop® provides thousands of image-editing functions such as resizing and filtering, the capabilities of image manipulation are limited to what the software provides (e.g. the software provides Gaussian blur, motion blur, radial blur, and smart blur for blurring operations, but the user cannot use their own specified blurring filter). Further more, undoing or fixing previous undesired operations is a tedious task using Photoshop®. One common occurrence is that a user performs several different operations on an image and finds out later that the operation step 3 is undesirable after completing operation step 10. Now he/she is forced to cancel/undo all of operations down to step 3, and then redoing all of the steps from 4 to 10 after either fixing or eliminating step 3. Another example of a tedious task in Photoshop® is making multiple collages consisting of multiple images because applying the same operations on multiple images for multiple collages is extremely repetitive. For a collage, the user is required to cut and paste from many windows that contain images and it can easily reach the point where finding the desired image is a chore. A repetitive task such as applying the same blurring filter on an entire directory of images also requires the user to go through the same operation for each image which is an inefficient time sink.

Thus, it will be very beneficial if there is a language that can perform image-editing functionalities. The language will solve all of the existing issues on Photoshop addressed above. It will significantly save the time of people performing tedious image editions.

Goal

Photogram, a programmable Photoshop® language, is a language that enables the user to perform Photoshop operations using an iterative coding process. Photogram is meant to be easy to use, portable, powerful and expandable.

1. Ease-of-use
Photogram is a clear and intuitive language which allows users to edit photos and create animation by writing an algorithm. Like Java® or C Photogram uses a well-defined set of basic syntaxes similar to Java® and this makes Photogram programmer-friendly.

2. Portable
Java® is a cross-platform portable language due to its own interpreter, and since Photogram converts the user-created-program to Java code, it is also portable. You can execute the program on any platform where Java 1.5 Virtual Machine is installed.

3. Powerful

Photogram enables the user to edit the photo, for example sharpening, blurring, and changing colors, with just few lines of code. This power and efficiency allows work to be more productive as well as providing a clear outline of what was used to achieve the final result.

4. Expandable
Since Photogram is a programming language, it is a simple matter to add one's own functions or even import any other libraries in order to use new functions that the user needs.

Features

Our language, Photogram, will support the following features:

Syntaxes and Semantics: The syntaxes and semantics of Photogram will be very similar to the Java® language with one exception for declaring macros, in which case C++ syntax is used.

Functions: Photogram will support the implementation of functions with a return type and parameters. The syntax for writing a function will be the same as the Java® language.

Importing: Photogram will support importing functions written on other .pg (Photogram extension) files by using "#include" statement (see example under Sample Code section).

Non-Object Oriented: Photogram will NOT support implementation of classes/objects; however, Photogram provides several built-in classes such as Image, Font, Color, Pixel, Line, Rectangle, and Oval.

Arrays: Photogram will support arrays. The syntax for declaring an array will be the same as the Java® language (see example under Sample Code section).

Editor: We will provide a specialized editor for Photogram, which will open new picture-viewable windows so that users can easily find out the coordinates of the desired points in the input images. However, a programmer can choose not to use our editor and use any other word editors.

Image Processing: Photogram will provide some built-in image processing functions that are difficult to implement using Photogram code.

Caption: Photogram will support the ability to create captions, drawing a string of text on the image.

GUI Creation: Photogram will provide simple GUI creation functions which can be used to test image processing functionalities.

Sample Code

The following sample code generates an output image which is displayed on the top of the cover sheet of this whitepaper:

```
#include “library.pg” // collection of functions written in Photogram
                        // drawString(Image i, String s, Font f, int x_pos, int y_pos)
                        // resize(Image i, int width, int height, bool
                                reserve_aspect_ratio, int respect_to_what)
                        // paste(Image dest, Image src, int x_pos, int y_pos)
#define WIDTH 0

main(){
        // Opens images from a directory
        Image [] myImages = OpenDir(“C:/dir/images/”);

        String [] captions = new String[myImages.getNumImages()];
        captions[0] = “Column”;
        captions[1] = “Escalator”;
        captions[2] = “Library”;

        // Calling the function collage
        Image result = collage(myImages, captions, 790, 580, 3, 1);

        // Saving the output image file
        result.saveAs(“C:/dir/result/collage.jpg”);
}

// A user defined collage function
Image collage(Image [] images, String [] captions,  int width, int height, int row, int col){
        // Creating the target image.
        Image result = new Image(width, height);

        // Calculating the width and height of each image in the collage
        int x_dimension = width/col;
        int y_dimension = height/row;

        // Setting the font to be used for adding text onto the input images
        Font font = new Font(Font.ARIAL, 15, Font.EMBOSS);

        for(int i = 0, int x_pos = 10; i < images.getNumImages(); i++){
                if( i != 0) x_pos += images[i-1].getWidth();
                // Overlaying the text on each image
                drawString(images[i],captions[i], font, x_pos, images[i].getHeight() +10);

                // Resizing the input image while preserving its aspect ratio with respect to width
```

```
                resize(images[i],  x_dimension, 0, true, WIDTH);

                // Pasting the modified input image in the target image
                if ( i != 1 ) {
                        paste(result, image[i], x_dimension*(i%col), y_dimension*(i/col) + 20 );
                } else {
                        paste(result, image[i], x_dimension*(i%col) - 30, y_dimension*(i/col) + 20);
                }
        }
        return result;
}
```