The IML *Language: An Overview*

Travis J. Galoppo (tjg2107@columbia.edu)
COMS 4115 – Spring 2006

Introduction

IML (Image Manipulation Language) is an interpreted programming language, built on top of the Java environment, for the rapid and easy development of image transformation tools. The language provides easy to use mechanisms for creating, loading, saving, and per-pixel processing of RGB raster images.

Motivation

Programmatically working with raster graphics can be challenging on many levels; the variety of file formats alone can keep a developer busy for years. Dealing with the complexities of bit masking, width vs. pitch, and colorspace issues can be very confusing. Although a number of libraries exist to help make the task easier, none seem to strike the ideal balance between ease of use and functionality.

Furthermore, while a number of very comprehensive software packages exist for image manipulation (i.e. Photoshop, GIMP), writing custom plug-ins for these packages remains a daunting task; also, batch processing of images with these packages, while possible, is limited.

The goal of IML is to provide a means of creating image transformation tools that can be used to batch process images.

IML

The crux of the language is the built-in "image" type, which allows for the easy loading, saving, creation, and per-pixel manipulation of RGB raster graphics. Images are broken down into pixels (addressable by (x,y) coordinate), which are further broken down into the individual color channels. The pixel type also has a built-in conversion to a 32-bit integer type, allowing for traditional bit-masking of color components, should the developer have a use for such.

By simplifying the details of accessing the raster pixel data, the programmer can concentrate on the implementation of image transforms.

Considerations

Special Purpose

IML is a special purpose language; as such, it provides only the constructs essential to the task of image manipulation, and limited, if any, support for general application development.

Simplicity

The special purpose nature of IML means that the language itself is quite small, making it relatively easy to learn. Furthermore, the built-in image and pixel types make the normally tricky task of raster manipulation quite easy.

Portability

Being rooted in the Java language, IML programs are architecture neutral and completely portable. Write once, run everywhere! (Well, ok, everywhere there is a Java runtime installed)

Sneak Peek

Syntax

IML syntax is a hybrid of Pascal and C++. Variable and function declarations take a Pascal-like form; for example:

To define a function, foo, which takes an integer, a, as an argument and returns an integer, you would use the following declaration:

```
function foo(a : integer) : integer
```

To define a function, bar, which takes an image, img, as an argument and returns nothing, you would use the following declaration:

```
function bar(img : image) : null
```

Variables defined within a function are preceded by the 'var' keyword:

```
var i, j : integer;
```

Most statements, however, are more familiar to C and C++ programmers:

```
for(i=0; i<10; i++){
  if(i % 2){
    j = i * 5;
  } else {
    j = i / 2;
  }
}
```

A function to perform a 50% alpha blend of two pixels would look like this:

```
function alphablend(p1 : pixel, p2 : pixel) : pixel {
  var p3 : pixel;
  p3.red = (p1.red + p2.red) / 2;
  p3.green = (p1.green + p2.green) / 2;
  p3.blue = (p1.blue + p2.blue) / 2;
  return p3;
}
```

As noted earlier, however, the language supports an implied conversion between the pixel type and the integer type, allowing code such as:

```
/* Invert pixel - for each color channel, c, c = 255-c */
function invert(p1 :  pixel) : pixel {
  var p2 : pixel;
  p2 = ~p1 & 0x00FFFFFF;
  return p2;
}
```

Furthermore, while user defined classes are not permitted in IML, the built in image type is a class with both static and instance methods. For example:

To load an image, we can say:

```
var myImage : image;
myImage = image::open("someimage.jpg");
```

To create a new image, we can say:

```
var newImage : image;
newImage = image::create(640, 480);
```

we could then save that image:

```
newImage.save("myimage.jpg");
```

Pixel data within the image is accessed by treating the image object as a two-dimensional array of pixels:

```
var myPixel : pixel;
myPixel = newImage[0,0]; /* x,y pixel coordinates */
```

Hopefully, this brief introduction to IML syntax has sparked your interest!

Summary

The IML language provides a simple platform for implementing image transform routines, which can subsequently be used for batch processing of image files. Due to its special purpose nature, it is easy to learn and simplifies the task of working with raster graphics.