

Raster Graphics: Images From Pixels and Colors

Summary: We examine some basic operations for working with the raster graphics representation of images.

Contents:

- Introduction
- Representing Grid Points
- Relevant Procedures
- Quick Reference

Introduction

As you may have noted from our initial discussions of drawing, there are a number of ways to think about how one creates and how one represents an image. And, as we noted in our discussion of computer science, there is a strong relationship between the way we organize or represent information and the algorithms we write to manipulate that information. Hence, we begin by exploring *raster graphics*, one of the simplest ways to represent images on the computer.

In the raster graphics format, an image is represented as a grid of colors. That is, we segment the image into a large number of uniformly sized squares, which we arrange side-by-side and top-to-bottom, and we assign a color to each square. We call this grouping a “grid” because, well, it looks like the grid on graph paper.

Variations on the raster graphics format are used in JPG, Bitmap, GIF, and PNG images.

Representing Grid Points

When we describe a raster graphics image, we need to indicate the width and height of the image. Because raster images are a grid, we typically indicate the width and height in terms of the number of columns and rows, not in terms of inches. We also need to describe the color at each point in the grid. While there are a number of ways to indicate colors of different grid points, it is easiest if we specify the color of each one separately, particularly if we are going to write programs that process raster graphics images. We call one grid point in an image a *pixel*.

In essence, we need to assign an *index* to each point in the grid. In assigning these indices, we number the rows of the grid from top to bottom, and number the columns of the grid from left to right. We also start both column numbers and row numbers with 0. When we refer to one pixel on the grid, we do so in terms of its column number and its row number. Hence, in an image that is 9 pixels wide and 5 pixels high,

- The top-left pixel is indexed (0,0)
- The top-right pixel is indexed (8,0)
- The bottom-left pixel is indexed (0,4)

- The bottom-right pixel is indexed (8,4)
- The center pixel is indexed (4,2)

Why is (0,0) the top-left pixel, rather than the bottom-left position, as in the Cartesian plane? One possibility is that images were originally described for television tubes, and televisions scan from top to bottom. Another is that whoever was responsible for designing the notation was familiar with linear algebra, and was using the typical notation for elements of a matrix.

Why do we start counting rows and columns with zero rather than with one? It turns out that some computations are easier with such a numbering system. Computer scientists almost always start counting with 0, rather than with 1.

Relevant Procedures

Of course, in order to write programs that manipulate raster images, we need to know more than how images and colors are represented - we also need to know what operations are available.

DrFu provides a few core operations to build and analyze images.

`(image.new width height)` creates and returns an image of a specified width and height. It returns a number that DrFu uses to identify the image. You will find it easiest to assign a name to the image, as in the following.

```
(define my-first-image (image.new 9 5))
```

If you want to work with an existing image that is stored in the file, you can load the image with `(image.load name-of-file)`. For example, you can load a picture of one of the CS faculty with the following definition.

```
(define samr (image.load "/home/rebelsky/Desktop/samr.jpg"))
```

When you first create or load an image in DrFu, it is not visible on the screen. if you want to see the image (and you will, eventually), use `(image.show image)`.

If you're working with an image that someone else created, or if you have forgotten the size of your image, you can find out the width and height of that image with `(image.width image)` and `(image.height image)`.

What color is an image when it is first created? The `image.new` procedure makes all the pixels in the image the same color as the background color. Hence, you may want to set that color before creating the image, using `(envt.set-bgcolor! color)`.

`(image.get-pixel image column row)` extracts the color of a particular pixel in the image. We might get the initial color of the middle pixel of the previous image with

```
(define center-color (image.get-pixel my-first-image 4 2))
```

In contrast, `(image.set-pixel! image column row color)` changes the color of a particular pixel in the image. We might change the top-left pixel of the sample image to the same color as the center with

```
(image.set-pixel! my-first-image 0 0 center-color)
```

The exclamation point at the end of a procedure indicates that the procedure is intended to change something, instead of just computing a result. In this case, `image.set-pixel!` changes the image. As you'll see in a bit, `envt.set-bgcolor!` changes the environment by giving it a new default background color. Because changing things can be dangerous, Scheme programmers remind themselves that a procedure changes things by suffixing them with that exclamation point.

Of course, we also want to set pixels using existing colors. The `(cname->color color-name)` will find the internal representation of a color for the given name. (As you'll see in the next reading, those representations are typically RGB colors.) For example,

```
(define a-color-i-like (cname->rgb "deep purple"))
```

We can also use this technique to set colors.

```
(image.set-pixel! my-first-image 2 3 (cname->rgb "blood orange"))
```

What if we want to find out what color names are available to us? `(cname.list)` will give a list of every available color, and `(cname.list name)` will give a list of the colors that include *name*. For example, the following will list a number of colors whose name includes “red”.

```
(cname.list "red")
```

Finally, when we've obtained a color from an image, we can find the name of a similar color using `(rgb->cname color)`. Why is it a similar color, rather than exactly the same color? Because there are sixteen million, seven hundred seventy seven thousand, two hundred and sixteen different colors possible in the standard simple color scheme, and no one is mentally ill enough to try to name them all.

Quick Reference

- `(cname.list)` - list all the available named colors
- `(cname->rgb name)` - given the name of a color, convert it to a representation that can be used by `image.set-pixel!` and other procedures
- `(image.get-pixel image column row)` - get the color of the pixel at the given position
- `(image.height image)` - determine the height of the specified image
- `(image.load filename)` - load an image from the specified file
- `(image.new width height)` - create an image of the specified width and height
- `(image.set-pixel! image column row color)` - set the color of the pixel at the given point
- `(image.show image)` - display the specified image
- `(image.width image)` - determine the width of the specified image
- `(list-colors name)` - list all the available named colors that contain a particular name
- `(rgb->cname color)` - given a color obtained from an image, find a common name for that

Copyright © 2007 Janet Davis, Matthew Kluber, and Samuel A. Rebelsky. (Selected materials copyright by John David Stone and Henry Walker and used by permission.) This material is based upon work partially supported by the National Science Foundation under Grant No. CCLI-0633090. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc/2.5/> or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.