Matt Mueller
May 2 2009
CS 638 Final Project

Piximilar: Image by Color

Abstract

Piximilar: Image by Color is a project based on the work of Idée Labs, where you search for images by color rather than by text. Piximilar offers a unique experience that allows web designers to define the color scheme on the drawing board and write the cascading style sheet (CSS) before picking out the images. Piximilar leverages Flickr's vast free photo archive picking out only the images that Flickr deems "interesting". Piximilar analyzes these pictures using an in-house algorithm that maps each pixel to color palette options, allowing pictures to be retrieved based on their color contents. The processing is abstracted away from the user and built behind a beautiful user interface, which is user-friendly, scalable, and fast.

Introduction

One of the most important aspects of designing a website is finding the right images. Piximilar offers an innovative approach to finding these perfect images, allowing one to search through thousands of interesting images by clicking on a desired color. Piximilar can even combine up to four colors, providing designers with a new opportunity of finding interesting images that will match their website's theme. Piximilar also adheres to businesses that rely on color combinations for branding purposes. Furthermore, unlike Google Image Search, Piximilar will only search through images with non-restrictive licenses – that is free to use, free to modify, free to profit from. As we will see, Piximilar is a very handy tool that every web developer and graphic designer should be equipped with.

Motivation

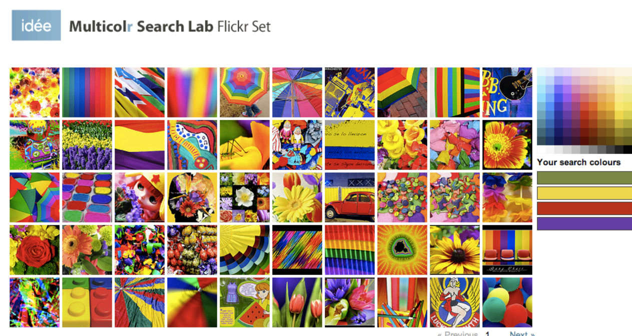
My main motivation for taking on this problem came after I saw the work of Idée Labs and how they addressed this problem. I found their tool extremely useful and innovative. This cool web application coupled with my aspirations to be a profession web developer were the main motivating factors for me to build my own web application that finds images by color.

Problem

The underlying problem that I will address in this paper is how to find images that are similar, not according to the text around them, but by the colors these images contain. I will go through the three main steps needed to solve this problem– how to find images to index and analyze, how to derive meaning and find similarities from images without a textual context, and how to wrap these results in an interface that makes it useful for everyone.

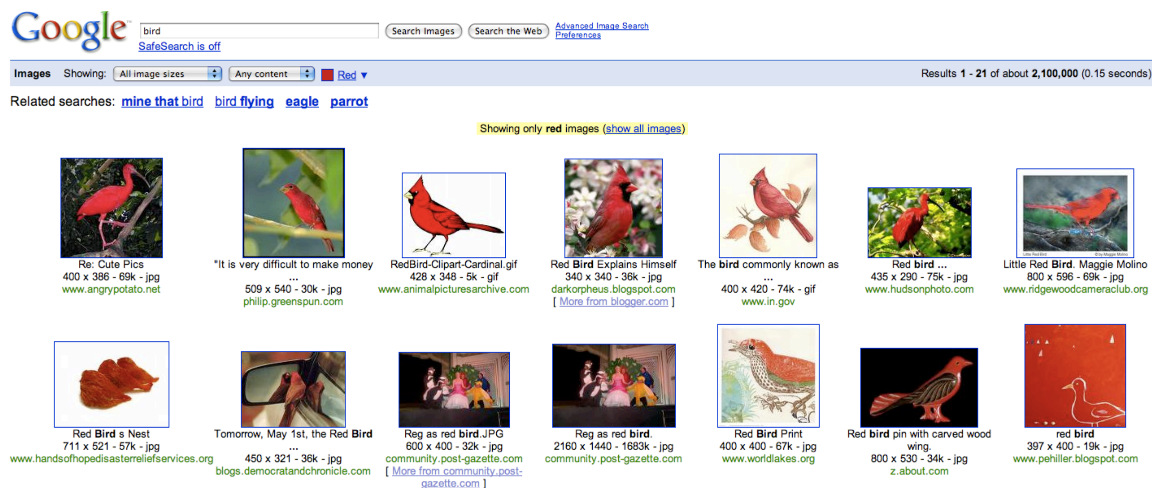
Related Work

As mentioned in the abstract– this project is based off the work of Idée Labs. Idée Labs created the Multicolr Search Lab, which extracts the colors from over 10 million of the "most interesting" free photos on Flickr and allows you to search through the images by color. The picture below shows the interface they designed to leverage their visual similarity technology.



As you can see, you click on colors in the swatch palette to select the colors. In this case, I clicked on a green, yellow, red, and purple. The results represent interesting images that have the largest sampling of our selected colors from their ten million image database.

Another company that has implemented searching images by color is Google. This feature is very new– in fact, they updated their Image Search to include this feature while I was working on this project. Google implements a search by color differently from Idée Labs. They offer the feature more as an additional filter for a textual search. So typically you would search for a word, say "bird" and then select a color, say "red", to find only birds that are red. The picture below represents the results of this query.



Google operates on a much larger scale, analyzing billions of images indexed from the web. A "Find Image by Color" option is better as a filter for a text search in this case because their database is so vast that it would be nearly impossible to find interesting images given a set of colors.

Theory

The theories and assumptions I made going into the project were invaluable in producing the final product. I assumed that there would be a way of mining Flickr for free images that were also interesting to look at. This was a big assumption, but after

looking at the Multicolr Search tool, I knew it had to be possible. The theory was that I would design a palette of colors or options for the user to choose from. Now these options were just a small subset of the possible RGB colors, but provided a good sampling of the entire set. Next I needed to go through each pixel of each image and map it to one of the colors in the color palette. I assumed that PHP, the scripting language I was using, would be capable of extracting RGB values from images and would be powerful enough to do this mapping. Next I needed a place to store these mappings, so that they could be leveraged in a web application. I assumed that MySQL would be a capable of storing over 20 million entries. I also assumed that MySQL would be able to perform calculations on these entries quickly in order to keep the interface interactive for the user. The theory was that since MySQL uses powerful set theory algorithms, it would be able to sort and multiply a vast number of rows very quickly, which was crucial for my project to function correctly. Furthermore if MySQL was capable of doing the heavy lifting, I assumed that the PHP script that is responsible for sending updates to the web application would run swiftly. My final theory was that if I obtained over 100,000 images, my application would have enough pictures to choose from that selecting images by color would work effectively.

Method

This section will be divided into the three crucial steps required for this project: Getting the images from Flickr, analyzes the images, and displaying the results.

The first step involved mining Flickr for images. Luckily, Flickr provides a way of doing this, by directing your code to interact with their API*. Flickr's API provides a method called *flickr.interestingness.getList*, which returns a list of interesting photos for the most recent day or a user-specified date. This method has some options including the

date, how many items per page, and which page number. I designed my script to grab the XML generated from the API using URL:

*http://api.flickr.com/services/rest/?method=flickr.interestingness.getList&api_key=**API_KEY**&date=**DATE**&per_page=500&page=1*

The ***API_KEY*** is the key every Flickr developer has to apply for, and ***DATE*** is the date I specified. One of the limitations of the Flickr API is that you can only retrieve a maximum of 500 images per page for a specified date. I needed more than just 500 images— I needed thousands of images. The work-around was to write a script that would go through each date and grab the XML the API generated. So I wrote a script that went through January 1st 2004 to March 1st 2009 and grabbed the XML from 500 of the interesting photos on each of those dates. Since this approach can lead to a lot of unexpected results- I tested my script extensively to ensure that it could handle anomalies such as days that do not have 500 interesting images as well as when the script tries to access the XML from a day that does not exist (ie. February 30th). Getting the XML for each day is only the first step in acquiring the images. The XML allows you to generate a URL that directly links to the image. So for each day, after I generated the XML, the script would generate this URL and then save the image to the desktop. The XML generated looked like this:

```
<photos page="1" pages="5" perpage="100" total="500">
<photo id="3493249123" owner="9137439@N08" secret="ea5fa34246"
server="3550" farm="4" title="Poolside Reflections" ispublic="1"
isfriend="0" isfamily="0" />
```

I then used an XML parser to obtain each photo's farm number, server number, id, and secret code. I plugged in the values into this URL:

*http://farm**farm**.static.flickr.com/**server**/**id_secret_s**.jpg*

To obtain:

*http://farm**4**.static.flickr.com/**3550**/**3493249123_ea5fa34246_s**.jpg*

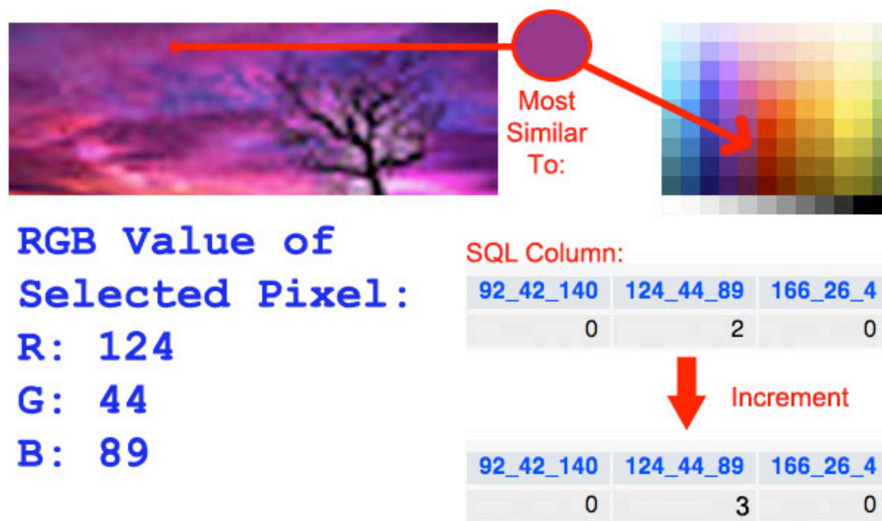
This URL provides a direct link to the image, so that it can be downloaded. Note the **_s**, right at the end of the URL. This tells Flickr to give back the small version of the photo, namely a 75 by 75 pixel image. This was crucial in saving space and speeding up the image analysis. Using this method, I downloaded 177,181 images to my computer, taking up a total of 1.3 Gb. The process took two full days to finish. Once the images were saved into a folder, I was ready to analyze these images.

The next step involves mapping each pixel of each image to a color in the color palette. I tried a few equal interval methods, to space out the colors evenly (ie. RGB(0,0,0), RGB(10,0,0), RGB(0,10,0), etc.), but the colors were very dull and uninteresting. I decided to map the colors to the palette used at Idée Labs. To do this, I first needed the RGB values of each of the colors in the palettes. I started to put each of these colors in a PHP array by hand using Photoshop's Color Picker on the image, but this quickly became tedious over all 120 color options. Since the color palette was arranged in a grid format, I wrote a script that incrementally moved through an image of the palette, and found the RGB values of each color, and put them in an array automatically. Now that I had the RGB values in an array, I was ready to write a function that would map the pixels in my images to colors in the color palette. The resulting equation was:

$$\sqrt{(red_img - red_palette)^2 + (green_img - green_palette)^2 + (blue_img - blue_palette)^2}$$

This was computed for each element in the RGB palette array, and the minimum was deemed the appropriate mapping. All in all, I simply found the minimum distance from the given pixel and a RGB pixel in the color palette. To find the RGB values of a given pixel, I used a PHP add-on feature called GD Library 2.0. This allowed me to use the built-in function imageColorAt(x, y), which returned an associative array of RGB values at a given x, y location in the image. Now that we know how to map the pixels of one

image to the pixels of the palette, we need a place to save this information. I made a database that had 122 columns, one for a unique ID, one for the name or the way of referencing the image, and the other 120 columns were for each of the colors in the color palette. I wrote a PHP script to generate the necessary SQL to create these columns. Once I had the database set up, I was ready to save the information from the mappings. As we went through each pixel of each image, the pixel would be mapped to a pixel in the color palette. At this point the column that corresponds to that color in the SQL database would be incremented. So images with a large amount of a certain color would all be mapped to the same color in the color palette, resulting in a large value in that color column in the SQL database. This is how you determine which images are rich with certain colors. The picture below illustrates the process:



The above example suggests that 3 pixels have so far been mapped to RGB value 124, 44, 89. We will continue this process until all 5,625 (75 x 75) pixels of each of the 177,181 images have been analyzed and saved to the database. This pre-processing algorithm took about three full days to run, but was essential to keep our web application fast and scalable.

The final step was to build an interface that would query the database and display the results of the color selections. Much of this is standard to all web sites- I build an HTML skeleton, styled it with some CSS and made some images with Photoshop. I also used jQuery, a lightweight Javascript framework to make AJAX calls easier and add effects. Designing the interface was fairly straightforward. The crucial step was to build the backend to query the database. When a user clicks on an item in the color palette, AJAX sends the RGB information of that color to a PHP script for processing. The PHP script then generates a query that finds the images with the largest number of pixels with that color. An example query looks like this:

```
SELECT * FROM `images` ORDER BY `images`.`124_44_89` DESC
LIMIT 0 , 32
```

This query sorts the selected color column in descending order then returns the top 32 images with RGB color: 124, 44, 89. This allows for you to pick a single color, but what if you want to find images that contain two colors, or three colors? To find images with a combination of colors, I used the equation:

```
Result_Column = First_Color_Column * Second_Color_Column
```

By multiplying two rows together, you promote images that have a combination of colors. If you would like to combine more colors together, you simply continue multiplying columns together. Once you have the result column, you do exactly what was done before – return top 32 images in the sorted result column. The following is an example query that shows this process:

```
SELECT 65_126_218, 199_34_5, name, (65_126_218*199_34_5) AS
sum FROM images ORDER BY sum DESC LIMIT 0 , 32
```

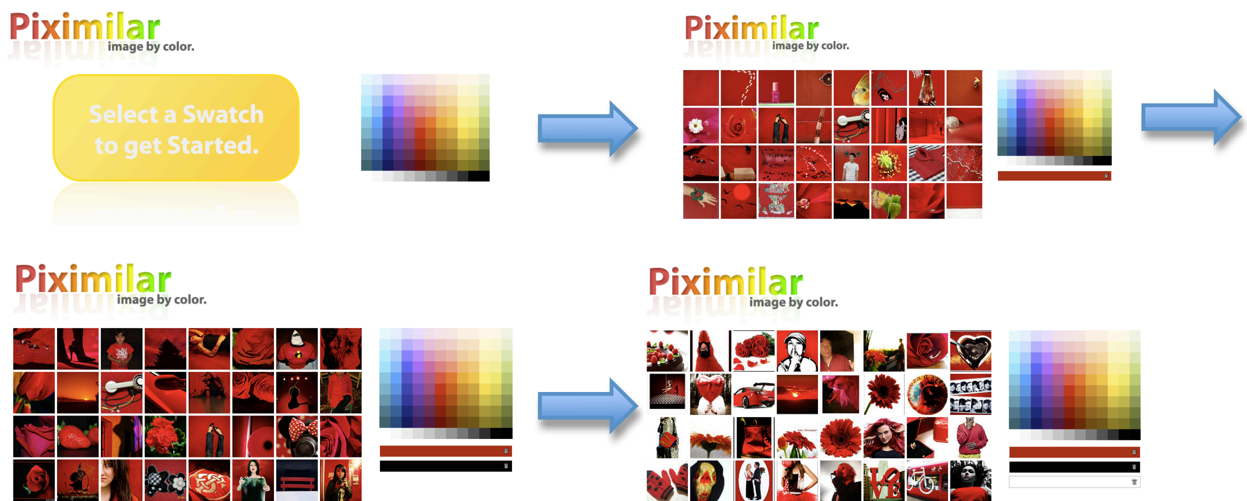
(65_126_218*199_34_5) is the result column of multiplying the two color columns together. Once the PHP retrieves the rows in the database, it generates HTML that will format each of the photos, points the image tags to the appropriate image directory, and

sends the HTML to a Javascript file that will update the browser asynchronously. These are the main steps to solve this problem. The next section contains the results of my implementation.

* Their API can be found at http://www.flickr.com/services/api/

Results

The following shows the web application timeline:



The results clearly demonstrate selecting images by color. As you add more colors you get more specific, honing in on images that are perfect for your current website.

Concluded Remarks

The results turned out great, but there is still room for improvements. Perhaps the biggest improvement would be to automatically link each image to its location on Flickr. The best way of solving this problem is to save the *owner* information from the XML feed to the database, allowing you to generate images that link to the following:

`http://www.flickr.com/photos/`**`owner`**`/`**`id`**

where both the id and the owner information are retrieved from the database. Another improvement would be to acquire more images. The Multicolr Search Lab has over 10

million images making their results much more impressive. The final improvement could target performance. The web application takes about two seconds to load the images after the user clicks on a color. If I were to organize the data in a clever way, perhaps by using a tree or graph, I might be able to squeeze some extra performance out of my application. Despite these areas that need improvement, I took a complicated problem and broke it down into steps that were each solvable to create an innovative web application that allows users to search through images by color rather than by text.

References

- Multicolr Search Lab: Flickr Set. Idée Labs. 2008. http://labs.ideeinc.com/multicolr/
- Google Image Search. Google. 2009. http://images.google.com/
- Flickr Interesting Photos. Yahoo. 2009. http://www.flickr.com/explore/interesting/
- PHP Manual. PHP. 2009. http://www.php.net/
- jQuery Documentation. jQuery. 2009. http://docs.jquery.com/Main_Page/