

Predominant Color Extraction from Images on Mobile Devices

by Brian Westphal, cliqcliq co-founder

Factors and Compromises

It's incredible to see how far mobile devices have come over the years. Just a few short years ago, writing a paper about performing image processing tasks on a mobile device would have been unthinkable. Of course, as powerful as mobile devices have become, there are still several factors limiting very complex computing tasks, including image processing. The factors to take into account when performing image processing tasks on mobile devices are: power consumption, memory use, time, and result quality. On a desktop machine with a recent CPU, one can perform most standard image processing tasks virtually instantaneously. On a mobile device, it's a different story. Even taking a single-pass through the pixels of an image can be time and battery consuming, for instance. However, many of the latest mobile devices, such as the iPhone, have special hardware features that help compensate.

Version 1.1 of cliqcliq Colors adds support for extracting the predominant colors of images, including the up to 2.1MP images taken by iPhone cameras. It is able to do this in a fraction of a second while maintaining excellent quality, using a few simple techniques and taking advantage of the built-in iPhone (and iPod Touch) hardware capabilities.

Because designers and developers need high levels of precision when choosing colors, the one thing we do not want to compromise too heavily on is result quality. An obvious choice for processing images quickly might be to compute simple local averages. This could be performed in near real-time, but would produce bad results because of not taking object boundaries into account. To get extremely accurate results, on the other hand, one might compute a histogram from an image and then process the histogram to find local maxima. However, this would take several processing steps and delay the user from getting results quickly. We chose a solution that can be computed in a fraction of a second, that produces high-quality results.

Solution

Our solution can be broken up into four main parts: reduction, search, scoring, and ranking. The first part of our solution, "reduction", is necessary partly because of usability and partly because of the screen limitations on the iPhone. To initially reduce the processing demands, we deal only with the part of an image that is visible on

screen. This helps the user get the results they're looking for while increasing image processing performance. Reduction takes advantage of the phone's built-in graphics processing hardware to perform scaling with smooth interpolation and reduces the number of pixels to be processed to a constant ($320 \times 286 = 91\text{K}$ in our case). This is still too many pixels to process quickly, so the next step is to reduce even more. We scale the visible portion of the image to only 10×10 pixels. If we started with a 2.1MP image, this would be a 21000x reduction. There are a couple of factors that make this acceptable and actually help improve the quality of our results. First, the usable pixel density of photos taken with the iPhone's camera is actually much lower than 2.1MP because of the way images are captured on the sensor and because of the poor sensor quality in general. One needs about 9 pixels from a captured image on the iPhone to represent one useful, precise pixel. Second, and more importantly, since we're only interested in extracting predominant colors, which must by definition take up some significant space in the image, we can reduce the noise of non-predominant colors by scaling down. Of course, if the user wants to focus on colors that are not predominant in terms of the entire image, one can zoom and pan around the image, thus finding colors that are locally predominant.

The iPhone can scale images incredibly quickly and with little power consumption by using the built-in graphics hardware, which is specifically designed for performing such operations. Once we have the image down to a reasonable size, we begin processing it, looking for the most predominant colors. With a 10×10 image, one could easily extract up to 100 unique colors without proper processing. However, we are only interested in significantly represented colors so as we walk through the pixels of the reduced image, we keep track of the top 12 most prevalent colors. The initial prevalence score is the number of similarly-colored pixels in the connected group. Pixels form a group when they touch other similarly-colored pixels. We compute similarity based on a single sample pixel rather than a group average so there could be some slight variance depending on which pixels are samples for starting groups. The color stored however, is the average color for the group.

Groups of similarly-colored pixels are found by "flood filling" (using a depth-first-search) for color similarity from a starting pixel. Later, if another group of pixels, which is disconnected from the first group, is found to be similar to the first group, the scores are added and colors re-averaged. Using this technique allows us to visit each pixel only once. We use two different similarity metrics depending on whether or not the pixels are in a group or are disconnected. More specifically, pixels that are slightly less-alike are

considered similar if they are connected. Groups of pixels that are disconnected must be more-alike to be considered similar (and have their prevalence scores added).

```
rrrrrrggggr  
rrrggggrrr  
rrgggggrrr  
rrrggbbbb  
grgggggbb  
grrrggbbb  
rrrbbbbbbb
```

Figure 1. Two, disconnected groups of similarly-colored pixels. The letters r, g, and b might represent reddish, greenish, and blueish pixels in this case, but this is used as an example only. The realistic variations in tone may be much more subtle.

As a matter of aesthetics and function, while we limit the resulting colors to the, at most 12, most prevalent colors, in the result set we order the colors in the order they appear in the image. We experimented with returning the colors in order of prevalence from least to most prevalent. However, this tended to make palettes appear disorganized. Organizing by the order colors appear in an image has the tendency to follow natural gradients, giving a more organized look to a palette.



Figure 2. Color Palette organized by the order colors appear in an image (bottom-to-top). Images are sometimes processed bottom-to-top because of their memory layout.