Color quantization using modified median cut

Dan S. Bloomberg
Leptonica

Abstract

We describe some observations on the practical implementation of the median cut color quantization algorithm, suitably modified for accurate color rendering. The RGB color space is successively divided in such a way that colors with visual significance, even if relatively small in population, are given representatives in the colormap. Appropriately modified, median cut quantization is nearly as good as our best octree quantization. As with octree quantization, error-diffusion dithering is useful for reducing posterization in large regions with a slow variation in color.

1 Introduction

The problem of color quantization is to represent full color RGB images, where each pixel is typically described by three 8-bit color samples, in an approximate fashion by a relatively small number of colors. We will assume that each color is represented by its 24-bit RGB value. Historically, the number of colors used has been determined by the depth of the display frame buffer, often 8 bits. The full color space, consisting of about 16 million pixels (2^{24}), is divided into a small number of regions, and for each region, a single representative color is used for each pixel that falls into the region.

There are in general two steps in the color quantization of a 24-bit RGB image. The first is to partition the color space into a small number of colors, and the second is to assign one of these colors to each pixel in the image. The second step requires a traversal through every pixel in the input image, using an *inverse color table* to map from the RGB value to the color table index. The results are significantly improved with *error diffusion dithering* (EDD). For decent results, the partitioning is adaptive, and depends on the pixel density within the color space.

There are many algorithms for vector color quantization that can be found in the literature, some of which are quite good, and others quite poor. In this report, we consider variants of Heck-

bert's *median cut* method, originally described by Paul Heckbert [1]. Two subsequent variations include those of A. Kruger [2] and the open source JFIF jpeg library [3].

The original *median cut* method partitions the color space into three-dimensional rectangular regions, that we call *vboxes*, with roughly equal numbers of pixels within each region. It is implemented by repeatedly dividing selected vboxes of the space in planes perpendicular to one of the color axes. The region to be divided is chosen as the one with the most pixels, and the division is made along the largest axis. The division itself is chosen to leave half the pixels in each part. The method does well for pixels in a high density region of color space, where repeated divisions result in small cell volumes and color errors. However, the volume in a low density part of the color space can be very large, resulting in large color errors. There are numerous other problems as well, which will be discussed below.

1.1 JFIF jpeg implementation

The open source jpeg library implementation is quite good, and makes a number of different choices. The space is quantized to 6 bits in green and red, and 5 bits in blue. Call the minimum volume the *quantum volume*. For this implementation, a quantum volume occupies 4 * 4 * 8 = 128 colors in the 8 bit/component color space (of which there are $2^(24)$ colors. The full color space occupies $2^(17)$ of these quantum volumes. Each vbox that is to be divided in two parts is first shrunk to a minimum (quantized) rectangular volume that includes all the pixels. If any quantized dimension of the vbox is 1, it will not be eligible for further subdivision along the other two axes. Otherwise, it can be split in half, by volume, along the largest axis, without regard for the number of pixels placed in each half. This is already a significant deviation from making a "median cut."

How are the rectangles chosen for subdivision? Suppose we wish to quantize into N colors. For the first N/2 divisions, the vboxes with the maximum population are subdivided. Then for the remaining divisions, the largest vboxes are subdivided. Making some of the cuts based on vbox volume has two important effects. First, such cuts reduce the largest color errors that may occur. Second, by spreading the representative colors in the color table more evenly through the color space, they allow better results with dithering.

Once the color space has been divided up and the colors chosen, an inverse colormap is generated. This is somewhat complex because the N vboxes had been shrunk before each division to smaller rectangles to eliminate empty space. Consequently, they do not fill the entire color space, but they do cover the part of the color space that is occupied by the pixels in the RGB image. Dithering will put pixels into those interstices, so it is necessary to be able to assign a close representative (quantized) color for those locations in color space. This is the most complicated part of

the JFIF jpeg implementation of color quantization.

2 Our modified median cut implementation

The primary issues that must be resolved in the implementation of the modified mean cut quantizer (*MMCQ*) are the following:

1. How to choose the vboxes for subdivision?
2. Which axis of the vbox is subdivided?
3. Where along that axis is the vbox subdivided?

Items (1) and (3) are particularly important for getting a balanced result that subdivides high density regions very closely and yet sufficiently subdivides low-density regions sufficiently so that the maximum error is not too large for small population clusters of pixels. *Excellent results will be obtained if (1) the most populated vboxes that can be further subdivided (i.e., have more than one quantum volume in color space) all have approximately the same population, and (2) all remaining large vboxes have no significant population.*

Through trial and much error, we have arrived at the following answers to these three questions:

1. *We first subdivide a fraction f of the vboxes based on population, and then subdivie the remaining fraction $1 - f$ of the vboxes based on the product of the vbox volume and its population.* The results are relatively insensitive to the value of f; anything between 0.3 and 0.9 yields good results. We use the product of vbox volume and population, rather than the vbox volume alone, because in the second stage, it is most important to split the vboxes that are both large and have a substantial population. No benefit comes from splitting large empty vboxes; even with dithering, few pixels are likely to fall far from regions of significant population.
2. *We subdivide a vbox along the largest axis.*
3. *The largest axis is divided by locating the bin with the median pixel (by population), selecting the longer side, and dividing in the center of that side.* We could have simply put the bin with the median pixel in the shorter side, but in the early stages of subdivision, this tends to put low density clusters (that are not considered in the subdivision) in the same vbox as part of a high density cluster that will outvote it in median vbox color, even with future median-based subdivisions. The algorithm used here is particularly important in early subdivisions, and

is useful for giving visible but low population color clusters their own vbox. This has little effect on the subdivision of high density clusters, which ultimately will have roughly equal population in their vboxes.

Once the color space has been subdivided, the colormap of representative colors is generated, with one color for each vbox, using the average color in the vbox. The histogram, which was used in the subdivisions, is no longer needed, so we convert it to an inverse colormap, where each element in the array (corresponding to a quantum volume) now holds the index of the colormap for the vbox that contained it. This allows a rapid lookup, in the final quantization stage, of the color index from the RGB value.

Without dithering, the initial vbox can be taken to be the minimum quantized volume that includes all pixels in the image. With dithering in the stage where the quantized pixels are found, it is necessary to represent all RGB values by a colormap index, so the initial vbox includes all of RGB space. Consequently, we avoid the difficulty of selecting color indices for regions of RGB space that are not within any vbox.

This implementation differs in several other ways from the JFIF jpeg one, as is evident from the description above. The scheduling of vbox splits is a two-part sequence, with the first fraction based on the pixel population and the second on the product of pixel population and vbox size. The population-bsed splits can continue down to the quantum volume, which is a cubical region of RGB color space. The location of the split depends in a particular way on the distribution of pixels in the vbox histogram, not simply at the center of the vbox.

Initially, I implemented MMCQ with all vbox divisions based on population, and found that serious problems can arise from the dithering process on some images. Suppose the image has (1) large regions of nearly constant color and (2) some vboxes that are both large and have a sizeable population, but not enough pixels to get to the head of the queue. If one of these large, occupied vboxes is near in color to a nearly constant color region of the image, dithering can inject pixels from the large vbox into the nearly uniform region. The representative color for the large vbox can be very far away from the color of the nearly constant region, and as a result the color oscillations are highly visible. The use of a second sequence of subdivisions based on the product of vbox volume and population directly targets these problematic large vboxes that have significant population, and, from what I can tell, cures the problem. Another way to prevent large color oscillations is to put a cap on the amount of pixel color transferred in the dither process. We have done this as well, with a cap on each component transfer corresponding to about 12 levels. (The dither computation is carried out at 8x the pixel level; in this scale the cap is set at 100.) Think of the cap as a damping function on large dithering oscillations.

2.1 Comparision of MMCQ with the best octree quantizer

The best color quantizer in **Leptonlib** has been the two-pass octree quantizer (*OQ*), which is implemented in `colorquant1.c` and also described in a report. [4]. Comparing the results of MMCQ with the two-pass OQ on the *prog/test24.jpg* painting, we find for non-dithered results:

1. For rendering small color clusters, such as the various reds and pinks in the image, MMCQ is not as good as OQ.

2. For rendering majority color regions, MMCQ does a better job of avoiding posterization. This is because it is more thorough at dividing the color space up in the most heavily populated regions.

Notwithstanding these differences, the results are very similar, both in quality and in speed of computation. On photographs such as *prog/marge.jpg*, with large regions of slowly varying color, both methods work quite well and there is little reason to prefer one over the other.

There are many other experiments one can do with variations of the median cut algorithm. The present implementation provides a simple framework for carrying out those experiments. For example, it is trivial to replace the existing function that selects the cut plane by one that always cuts in the center.

At this time, I believe the following conclusions are safe to make:

- Different methods of dividing up color space (octrees, vboxes), when carefully implemented, give similar results.

- The results are sufficiently good on many images that dithering is not needed. However, in situations where posterization is evident, dithering will eliminate it, at a cost of poorer compression and increased "speckle." (The cap on dither color transfer will reduce most of the visible speckle.)

- The fact that there are multiple straightforward ways to solve the problem of color quantization explains the large number of papers on the subject.

As usual with **Leptonlib**, the code is documented so that you (and I) can understand it without a lot of "what-the-hell-is-going-on-here" angst. If you can't figure something out from the code, send me a query.

References

[1] P. Heckbert, “Color image quantization for frame buffer display,”, *Computer Graphics*, **16**(3), pp. 297-307 (1982).

[2] A. Kruger, “Median-cut color quantization,” *Dr. Dobb’s Journal*, pp. 46-54 and 91-92, Sept. 1994.

[3] ftp://ftp.uu.net/graphics/jpeg, see jquant2.c in version 6b of JPEG library, 1998.

[4] Leptonica report: “Color quantization using octrees”, http://www.leptonica.org/papers/colorquant.pdf.