OPTIMAL PALETTE CREATE WITH THE SPECIFIED NUMBER OF COLORS USING OCTREE QUANTIZATION

Faisel G. Mohammed
faisel@scbaghdad.edu.iq
Department of Computer Science, College of Science , University of Baghdad. Baghdad-Iraq

Abstract

In the current research work, improved and fast method to get a good approximation of the most important 256 colors of a RGB-Picture was introduced. In this paper, we propose a spatially-adaptive optimum octree quantization method for robust color printing. The goal of spatially-adaptive octree quantization is to use quantization well-suited for smooth areas, and to use octree quantization method better suited for edges. It may be useful for digital image compression, graphics- and game programmers. Also, this includes reduction of the overall number of colors, quantization or patterning the reduced number of colors, generating bitmap images, and even handling Boolean (on/off) transparency. The results show the suggested adaptive octree data-structure is useful for color-quantization.

Keywords: Color reduction, octree data-structure quantization, Color image processing.

خلق صفيحة لونية مثلى بعدد معين من الالوان باستخدام التكميم الثماني

فيصل غازي محمد
faisel@scbaghdad.edu.iq
قسم علوم الحاسبات ، كلية العلوم ، جامعة بغداد. بغداد–العراق

الخلاصة

في العمل البحثي الحالي تم تطوير طريقة متقنة سريعة للحصول على تقريب جيد للالوان المهمة ذات ٢٥٦ مستوى لوني في الصور الملونة. في هذا البحث اقترحنا طريقة تكمية ثمانية مثلى مطورة حيزياً لعرض فائق للالوان. الهدف من هذه الطريقة هو استخدام التكمية اللونية بشكل ملائم للمناطق المتجانسة ، واستخدام هذه الطريقة بشكل ملائم للحواف. يمكن ان يستفاد من هذه الطريقة في ضغط الصور والرسوميات وبرمجة الالعاب. كذلك هذه الطريقة تتضمن تقليص المستويات اللونية (التكمية) او تنقيش عدد الالوان المقلصة ، توليد صور نقطية وكذلك معالجة المناطق الشفافة. النتائج بينت ان الطريقة المعدلة الثمانية المهيكلة للبيانات مفيدة لتكمية الالوان.

الكلمات المفتاحية: معالجة الصور الملونة، تكميم هيكل البيانات الثماني، تقليص الالوان

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. In this paper the assumption that each color is represented by its 24-bit RGB value.

Color quantization methods can be broadly classified into two categories: image-independent methods that determine a universal (fixed) palette without regard to any specific image, and image-dependent methods that determine a custom (adaptive) palette based on the color distribution of the images. Despite being very fast, image-independent methods usually give poor results since they do not take into account the image contents. Therefore, most of the studies in the literature consider only image-dependent methods, which strive to achieve a better balance between computational efficiency and visual quality of the quantization output [1].

Different quantization methods are investigated in previous related works: Static Color Table, Median cut, Popularity and Octree and combined with error diffusion techniques: Dithering and Floyd Steinberg.

To assess quantization techniques, the considered quality criteria's are Human Perception, Run Time and Memory Requirement

The objective of color quantization is displaying a full color image (24 Bits per pixel) with a restricted set of color numbers (256, 64, 16) without a significant (almost preceptually not noticeable by the spectator) lack of color impression approximation as closely as possible when quantized.

Generally speaking quantization can be viewed as a stepwise process:[2]

1. In the first step statistics on the used colors in the image that is to be quantized are generated (histogram analysis)
2. **a)** Based on the analysis the color lookup-table has to be filled with values **b)** The true color values are mapped to the values of the color table. The color values have to be mapped to the nearest color entries in the color table.
3. The original image is quantized. Each pixel is transformed to the appropriate index of the color table.
4. Optionally an error diffusion technique can be applied.

In the recent research work, the octree algorithm was based in colored image quantization in adaptively form. This algorithm showed in the next section.

2. Octree Algorithm

The octree color quantization algorithm, invented by Gervautz and Purgathofer in 1988, encodes image color data as an octree up to nine levels deep. Octree is a tree-structure which contains data. The data is stored in a hierarchical way so that we can search an element very fast. Every tree has at least a root-node which is the anchor of all subnodes. Every Node has pointers to 8 subnodes [4], see (Figure1).

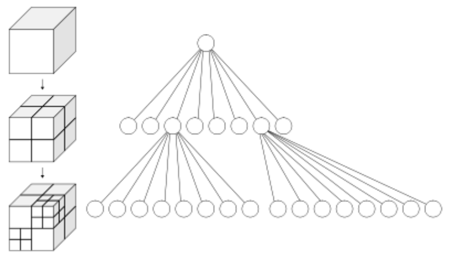


Fig. 1: Left: Recursive subdivision of a cube into octants. Right: The corresponding octree.

The principle of the octree algorithm is to sequentially read in the image. Every color is then stored in an octree of depth 8 (every leaf at depth 8 represents a distinct color). A limit of **G** (in this case **G** = 256) leaves is placed on the tree. Insertion of a color in the tree can result in two outcomes:-

1. If there are less than **G** leaves the color is filtered down the tree until either it reaches some leaf node that has an associated representative color or it reaches the leaf node representing its unique color.
2. If there are greater than **G** leaves in the tree some set of leaves in the tree must be merged (their representative colors averaged) together and a new representative color stored in their parent.

Gervautz & Purgathofer [5] offer two possible criteria to be used in the selection of leaves to be merged.

1. Reducible nodes that have the largest depth in the tree should be chosen first. They represent colors that lie closest together
2. If there is more than one group of leaves at the maximum depth the algorithm could:
    - Merge the leaves that represent the fewest number of pixels. This will help keep the error small
    - Reduce the leaves that represent the most pixels. In this case large areas will be uniformly filled in a slightly wrong color while maintaining detailed shadings.

Once the entire image has been processed in this manner the color map consists of the representative colors of the leaf nodes in the tree. The index of the color map is then stored at that leaf, and the process of quantizing the image is simply filtering each color down the tree until a leaf is hit. Because a limit is placed on the number of leaves in the tree this algorithm has a modest memory complexity, **O(G)**, compared to the median cut and popularity algorithms. The time complexity is more unclear. Gervautz & Purgathofer [5] site the search phase as being **O(N)** where **N** is the number of pixels in the image. This is clearly best case behavior. The average case needs to address the complexity of the merging algorithm. The advantage of the octree quantization is that it is simple to generate both a good partitioning of the color space and a fast inverse color table to find the color index for each pixel in the image [6].

3. Proposed method description

In this section we describe the proposed spatially-adaptive octree method. All processing is done independently for each color plane. Consider a particular color plane of the image, and a threshold image the same size as the image. As is standard, different octree masks are used for each color plane to create clusters of dots at different angles. Instead, each color plane of the image is partitioned into non-overlapping windows; in this paper we use 12 × 12 pixel windows. Using non-overlapping windows preserves the ability to implement the octree as a parallel process. For each window, a decision is made as to whether there is intense spatial activity (such as an edge or texture). If there is not substantial spatial activity within the window, then standard octree is applied: the image pixel is compared to its corresponding octree to determine if a dot should be printed there. However, if there is substantial spatial activity within the window, then octree is used. The algorithm is highly memory efficient because the tree's size can be limited. The bottom level of the octree consists of leaf nodes that accrue color data not represented in the tree; these nodes initially contain single bits. If much more than the desired number of palette colors are entered into the octree, its size can be continually reduced by seeking out a bottom-level node and averaging its bit data up into a leaf node, pruning part of the tree. Once sampling is complete, exploring all routes in the tree down to the leaf nodes, taking note of the bits along the way, will yield approximately the required number of colors. The list of code of our proposed method showed in details in appendix A.

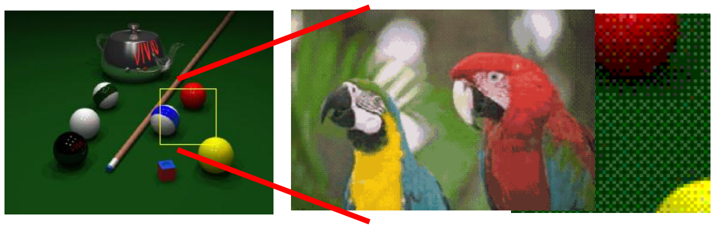
4. Results and Experiments

Experiments compared standard halftone with the proposed optimum octree color quantization method, as described in previous section. The effectiveness of a quantization method was quantified by the commonly used Mean Squared Error (MSE) measure:

$$MSE = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}\left\| I(h,w) - Q(h,w)\right\|^2$$

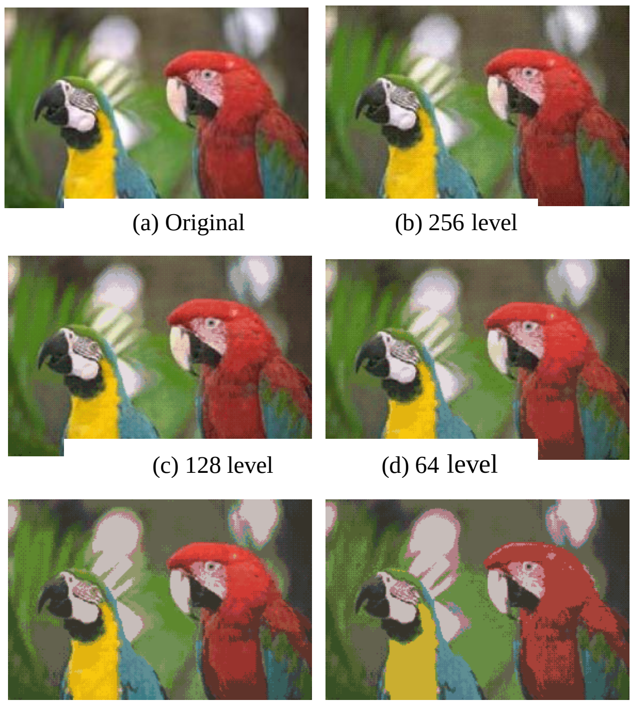
where ***I*** and ***Q*** denote the original and the quantized images with height ***H*** and width ***W***, respectively.

(Figure 2) shows sample quantization results for the "Billiard" image. It can be seen that halftone method generates severe dots especially on the flat, red and yellow balls. In contrast, adaptive octree method obtains visually pleasing results with less prominent false dots. Also, example results are shown in (Figure 3). Figures should be viewed from some distance to get a feel for what the octree would look like printed at a reasonable dpi. Edges are clearly sharper in the spatially adaptive octree method (the bottom halftone in each figure), for example the red arm of the horse-rider in (Figure 3). Smooth areas of the spatially-adaptive method look like the ordered dither halftone, for example in the background of Figure 4, because if a window was judged to be smooth, ordered dither was used.(Table 1) show MSE results for "Parrots" image.



(a) Original (b) Octree (c) Halftone

Fig. 2: Sample quantization results on "Billiard" image.



(a) Original (b) 256 level

(c) 128 level (d) 64 level

(e) 32 level (f) 16 level

Fig. 3: Sample quantization results on "Parrots" image.

Table1: MSE measures of octree and halftone on "Billiard" and "Parrots" images using different octree color depth to obtain different number of final quantized colors.

Number of final quantized color levels	MSE "Billiard" synthesized image	MSE "Parrots" real image
256	0.0009	0.0044
128	0.0000	0.0030
64	0.0000	0.0044
32	0.0000	0.0023
16	0.0000	0.0023
8	0.0002	0.0157

In the spatially-adaptive octree the normal rosette pattern of the octree can be seen, but in none of the test images could we see an artifact pattern from the square windows. Also, the rosette pattern from the dither tends to be slightly less visible because it is broken up by the octree windows, for example the yellow background in (Figure 3-f).

The spatially-adaptive octree uses two repeating patterns: the regular pattern of the octree mask, and the regular square windows used for the spatial adaptation. Often when different repeating patterns are used moiré problems arise.

In fact, we saw less moiré with the spatially-adaptive octree, we conjecture this is because this method simply does a better job at capturing edges.

5. Conclusions

There are some final things that can be used to gain more speed. Most of the time is spent finding the node which has to be reduced. It doesn't hurt to maintain a list of all nodes that are on the same level (level means how deep the node is stored in the tree). A simple linked list could be very useful for this. The best node to reduce is always stored at the highest level in the tree. It also doesn't hurt to reduce random nodes and don't analyze the contents of the node as long as we reduce in the highest level. The palette won't be as good, but it'll be a lot faster (and the difference is hardly noticeable).

Also after created the palette the tree is still useful. It could use them to fast map a RGB-triple to a palette index. To do this we have to store the palette indices into the nodes during the palette calculation.

When we want to find a color we simply traverse the tree down until we find a node which reference-count is greater than zero. The palette indices can be directly read out of the node. To our knowledge this is the fastest way to map RGB colors to adaptive palettes. Normally we don't need to use all 8 bits of the color-components. Using only 6 bits will give more performance and because standard VGA-adapters can only use 6 bits per primary color no one would be able to spot the difference at all. It also saves a lot of memory

If memory-usage is an issue we might reduce the nodes after we inserted a complete scan line. We wouldn't reduce to 256 colors in this case but to 4000 or so. This will help us to keep the tree small Some Comments and Reactions. A genius idea how to speed up the quantization process is to keep the tree small. We reduced the nodes which have a reference count of 1 (thus represent only one pixel of the source-image). This of cause keeps the tree small and saves lots needless tree-traversal.

The suggested algorithm very elegant and useful. It's a good example of the high art of computer programming and maybe it makes we interested in other algorithms.

On the one hand the algorithm leads to the best results but on the other hand, it is small memory and time consuming. The algorithm is well suited for images that have to be displayed with the best quality and the quantization process is not considered. Artifacts may appear as local discontinuities in a color shade. A combination between the octree algorithm and the dithering technique is not appreciated because the color table entries are not equidistant. In comparison with the other algorithms this approach needs less implementation effort.

5. References

1. Celebi, M. E. **2009** "Effective initialization of k-means for color quantization", IEEE International Conference on Image Processing, USA, ICIP**.**
2. hpabax H., **2009** "Color Quantization Overview", technical report, Germany, http://algolist.manual.ru/graphics/quant/qoverview.php
3. Chang,Y.; Lee,D.; James, Y.H.J. and Liang, D., **2008** "A Robust Color Image Quantization Algorithm Based on Knowledge Reuse of K-Means Clustering Ensemble", Journal of Multimedia, Vol. **3**, No. 2, June**.**
4. Yurong, L.; Zhengdong, D. and Hongguang,F.**2010.**" Adaptive Extraction of Principal Colors Using an Improved Self-Growing Network", Journal of Computers, Vol. **5**, No. 2, Feb.

Appendix A

List code of our suggested optimal octree quantization method were showed in the next sections:

(A-1)Main Procedure "Create Optimal octree palette"

Procedure benefit: Create an optimal palette with the specified number of colors using proposed adaptive octree quantization.

```
1. Start Create Optimal Procedure
   Inputs:
     • Number of levels of color to create (nLevels)
     • Max. number of colors (nColors)
     • Channel weights [0,1] (RedW=GreenW=BlueW=1)
2. Allocates initial storage:
     • Set: number of color depth = Number of levels of
       color to create (nDepth=nColor)
     • Set: Current Node; cNodes = 1: Current Color; cClr =
       0
3. For All image pixele in x,y dimension Do
   Adds the current pixel to the color octree
     • Call Add_Colour (1, 1, 0, 255, 0, 255, 0, 255, Bits(x,
       y).R, Bits(x, y).G, Bits(x, y).B)
   Combine the levels to get down to desired palette size
     • Do While (cClr > nColors)
     •     If (pvCombineNodes = 0) Then Exit Do
     • Loop
   end.
4. Call Initialize_Palette
5. Go through octree and extract colors
     • For y := 1 To UBound(aNodes) Do
     • If (aNodes(y).bIsLeaf) Then Do
           With aNodes(y)
             o   m_tPal(lEnt).R = .vR / .cClrs
```

```
            o m_tPal(lEnt).G = .vG / .cClrs
            o m_tPal(lEnt).B = .vB / .cClrs
            o lEnt = lEnt + 1
            end With
      • end If
      end.
      m_Entries = lEnt
6. Call "Create logical palette"
```
End Procedure

(A-2) Procedure "Add_Colour"

Adds a color to the Octree palette. Will call itself if not in correct level. (Recursive procedure)

```
1. Inputs:
      • iBranch (Branch to look down)
      • nLevel (Current level (depth) in tree)
      • vMin(R, G, B) (The minimum branch value)
      • vMax(R, G, B) (The maximum branch value)
      • R, G, B (The Red, Green, and Blue color
        components)
2. For all Color Bands: Find mid values for colors and decide
which path to take.
            • (vMid = vMinR / 2 + vMaxR / 2) for Red channel
3. Also update max and min values for later call to self.
      • If (wR * R > vMid) Then iR = 1: vMinR = vMid Else iR
        = 0: vMaxR = vMid
      • If no child here then... If
        (aNodes(iBranch).iChildren(iR, iG, iB) = 0) Then 4
4. Get a new node index
   iIndex = Get _Free_ Node
   aNodes(iBranch).iChildren(iR, iG, iB) = iIndex
   aNodes(iBranch).cClrs = aNodes(iBranch).cClrs + 1
      • Else  Has a child hereiIndex =
        aNodes(iBranch).iChildren(iR, iG, iB)
      • End If
5. If it is a leaf
      • If (aNodes(iIndex).bIsLeaf) Then
      • With aNodes(iIndex)
      • If (.cClrs = 0) Then cClr = cClr + 1
      •   .cClrs = .cClrs + 1
      •   .vR = .vR + R
      •   .vG = .vG + G
      • .  vB = .vB + B
      • End With
      • Else 6
6. If 2 or more colors, add to reducible a Nodes list
      • With aNodes(iIndex)
      • If (.bAddedReduce = 0) Then
            o .iNext = aReduce(nLevel)
            o aReduce(nLevel) = iIndex
            o .bAddedReduce = -1
      • End If
      • End With
7. Search a level deeper
            • Call Add_ Colour (iIndex, nLevel + 1, vMinR,
              vMaxR, vMinG, vMaxG, vMinB, vMaxB, R, G, B)
      • End If 5
```
End Procedure

(A-3) Public Sub Initialize(Optional ByVal InitialEntries As Integer = 0)

```
  Re-dimension palette
   m_Entries = InitialEntries
   ReDim m_tPal(255) As RGBQUAD
```
End Sub

(A-4) Function "Get _Free_ Node"

Start Get _Free_ Node Function

Gets a new node index from the trash list or the end of the list. Clears child pointers.

```
      • Outputs: - Node index
      •   cNodes = cNodes + 1
      • If (TopGarbage = 0) Then
      •   If (cNodes > UBound(aNodes)) Then
      •     i = cNodes * 1.1
      •     ReDim Preserve aNodes(1 To i)
      •   end If
      •   pvGetFreeNode = cNodes
      • else
      •   pvGetFreeNode = TopGarbage
      •   TopGarbage = aNodes(TopGarbage).iNext
      •   For i := 0 To 7 Do
      •     If (i And 1) = 1 Then iR = 1 Else iR = 0
      •     If (i And 2) = 2 Then iG = 1 Else iG = 0
      •     If (i And 4) = 4 Then iB = 1 Else iB = 0
      •  aNodes(pvGetFreeNode).iChildren(iR, iG, iB) = 0
      •    end Do
      • end If
```
End Function

(A-5) Procedure "Create Logical Palette"

Build logical palette here

```
      • With logPal256
      •   .palNumEntries = m_Entries
      •    .palVersion = &H300
      • End With
      • m_hPal = CreatePalette(logPal256)
```
End Sub

(A-6) Private Type PALETTEENTRY

```
(Structure of octree node)
   peR     As Byte
   peG     As Byte
   peB     As Byte
   peFlags As Byte
```
End Type

(A-7) Private Type LOGPALETTE256

```
   palVersion       As Integer
   palNumEntries    As Integer
   palPalEntry(255) As PALETTEENTRY
```
End Type