CmpE 492 Project

New Approaches in Quantization and Dithering

Bilgehan Uygar ÖZTEKIN

Supervised by

Lale AKARUN

Bogazici University

Computer Engineering Department

Fall 1997

Table of Contents

Abstract ____ *3*

Introduction ____ *4*

Quantization and Dithering ____ *7*

Implementation Details ____ *11*

Building the Histogram ____ **13**

Quantization Methods ____ **15**

First K Modes 17

K-Means 17

Recursive Splitting 19

Particle Simulation 20

Valley Descending and Simulated Annealing..... 23

Palette Color Assignment Methods ____ **25**

Nearest Neighbor 25

Error Diffusion..... 26

Probabilistic Assignment 26

Conclusions ____ *34*

Future Works ____ *52*

Glossary ____ *53*

References ____ *54*

Appendix ____ *55*

Abstract

Quantization is the process of reducing the color depth of an image, it usually requires a selection of a representative palette. Dithering is the process of using the palette as good as possible so as to represent the image with minimum loss. Most of the quantization techniques are deterministic and does not consider the averaging property of the eye. In this project we will try to:

- implement existing quantization methods and compare their performance on relatively large images having thousands of different colors.
- develop a new method, a probabilistic assignment of palette colors as an alternative to post dithering.
- develop a new method which will emphasise on selecting distant representative colors for the palette during quantization so as to obtain better results after dithering.

We will also try to optimize the code to increase scalability: Large and more colored images should also be processed in reasonable amounts of time.

Keywords

Vector quantization, dithering, error diffusion, probabilistic color assignment, clustering, histogram.

Introduction

Due to practical considerations and limitations of display devices, reducing the color depth (number of colors) of images become more and more common place with the growth of graphical user interfaces and graphics based programs. This need arises mainly due to the limited hardware capabilities, or speed requirements of certain applications. With increasing usage of graphical user interfaces, optimisations of these methods become more and more important.

Today's understanding of human vision suggests that most of the colors that can be seen by the eye can be constructed by superposing different amounts of three "primary colors" red, green, and blue, ie. we can fool the eye that it is seeing naturally occurring colors by simulating them using red, green and blue lights. A generally accepted theory suggests that the human eye has three different kinds of color receptors, the cones. (There is also a different type of receptors, "rods" sensible to intensity). Each color receptor is sensible to a range of frequencies, and the three different kinds of receptors are dominantly sensible to the frequencies corresponding to around red, green, and blue wavelengths respectively. For example a color of wavelength around red light stimulates mostly the red cones, then in a small amount the green cones and maybe slightly the blue cones. Human eye somehow uses the different amount of stimuli occurring at the three cone types and maybe other information to see "colors". Most of the "naturally occurring colors" can be simulated by adjusting the amounts of red, green, blue lights and superposing them on each other. Today, most of the raster scan displays and television systems use this technique to produce colors. Although some naturally occurring colors outside the "color gamut" (see ref. [1], ref. [5]) of the display system cannot be simulated, the method works well for human beings though it may not work for different perception systems.

More information about human vision, and color systems can be found in "Computer Graphics" by Donald Hearn and M. Pauline Baker (ref [1]).

Nowadays most of the display adaptors found in both personal computers and workstations are RGB (red, green, blue) system based, and there are two different approaches to color modes:

1) Paletted approach: up to n colors can coexist (usually n is 16, or 256), where each color can be selected from a palette of a larger color space.(usually 4, 6, or 8 bits per RGB components). There is a mapping table between color numbers and the colors represented; only the color number is stored in the frame buffer.

2) True color approach: the RGB components of each pixel is written to the frame buffer. In this approach the number of different colors that can be used is only limited by the image size and the size of color space. All of the red, green, and blue components have associated bits in the corresponding buffer for each pixel. Here are some of the commonly used pixel packing formats in true color modes:

   - 15 bits: 5 bits red, 5 bits green, 5 bits blue, 1 bit alpha. (*)
   - 16 bits: 5 bits red, 6 bits green, 5 bits blue.
   - 24 bits: 8 bits red, 8 bits green, 8 bits blue.
   - 32 bits: 8 bits red, 8 bits green, 8 bits blue, 8 bits alpha.

(*) this mode is in practice mentioned as 15 bits mode in order to differentiate it between 16 bits mode although its fields sum up to 16 bits.

For Intel™ x86 and higher series, VESA® standards [2] does not restrict the size of the fields but most of the hardware manufacturers only implement a subset of these modes for different resolutions.

Although true color modes result in better color depths and ease of programming due to direct color manipulation, they are relatively slower and have

higher storage requirements than the paletted color modes. Thus most recent graphical user interfaces use 256 color paletted mode in order to speed up the graphics operations. Moreover limited frame buffer size limits the color depth in higher resolutions considerably. Those and similar practical considerations result in the need for optimised color depth reduction algorithms.

Two different approaches in color reduction must be considered:

1) The reduction will be made once and then the resultant image will be used instead of the original image. In this case of course what we want is the best image that we can have in a reasonable amount of time. Some examples can be: storing a scanned image in a database by reducing the color depth but “not loosing too much” of the original image, preparing images or movies to be distributed, printing a final version of a document containing graphics.

2) The reduction must be made on the fly so that the original image can be displayed with the limitations of the current hardware and software. For example an image of a large color depth can be stored; each machine can display it according to its limitations or time requirements of the application. Previewing a large amount of images in a limited amount of time, or preparing thumbnails also requires fast reduction algorithms.

The first case requires finer algorithms although they may require more computation power, whereas in the second case what we want is acceptable results in a minimum amount of time and computation.

Quantization and Dithering

In the context of this document, quantization is the process of reducing the color depth of an image. It can be applied to either reducing a true color image to a true color image of lesser color depth which is quite straightforward and done by mapping the red green and blue parts of each pixel pack to the corresponding target pixel pack format. A more difficult problem occurs when a true color image (or an image of a greater palette size) has to be mapped to a limited paletted format in which the selection of the colors in the palette greatly influences the results obtained. The problem can be formalised as finding the k representative colors that will be used to reduce the number of colors, n of the original image so that the original image will be preserved as much as possible.

Dithering is the process of using the palette as good as possible to represent the original image with minimum error. All dithering methods are based on the fact that the human eye tends to average the colors and intensities of neighbouring points if the points are not too distant. Most printing and displaying devices use this technique to increase the number of colors or color levels by using extra resolution. For example most laser printers do not have grey levels; they only have white and black as building blocks, but since the resolution of the devices is big enough (typically 300 to 1200 dots per inch) by cleverly placing the black and white, illusion of high number of grey levels is achieved.

Error diffusion is one of the finest dithering methods which is extensively used in most graphical packages. It is based on the principle of distributing the error made in the quantization of the current pixel to the neighbouring, not yet processed pixels hoping that during the quantization of the modified neighbouring pixels, the error introduced in the first quantization will be balanced or reduced by the averaging property of the eye. We can say that in the error diffusion method, at the quantization stage of each pixel, we try to balance the quantization errors done

in previous pixels. Error diffusion does not have the anomalies such as repeating patterns, artificial edges that occur in using fixed, look-up based dithering methods such as halftoning and ordered dither where each pixel of the original image is mapped to a box or rectangle of bigger resolution of predetermined pattern (see ref. [1] and ref. [5] for more information about dithering and commonly used dithering methods).

In this project we used the error diffusion method developed by Floyd and Steinberg in which the difference between the exact color vector and the approximated color vector is added to the values of the four neighbors of the pixel as follows: 7/16 of the error to the pixel to the right, 3/16 to the pixel below and the left, 5/16 to the pixel immediately below, and 1/16 to the pixel below and to the right. The image is constructed from upper left position to lower right position line by line and the error encountered on each pixel is diffused to the neighboring not yet processed pixels as described above. We also developed a probabilistic assignment of colors in the palette as an alternative to the error diffusion method.

In quantization what we want is minimum quantization error. The error obviously depends on how much we decrease the color depth and which colors are selected to build the palette from which we will try to represent all the colors that occur in the original image. Most of current color quantization methods use RGB color coordinate system which can be though to be equivalent to Cartesian coordinate system where x, y, and z axes are replaced by “red”, “green”, and “blue” axes. Generally the error introduced by using color b instead of color a is considered to be the square of the “Euclidian” distance between the two points corresponding to the colors a and b in the RGB coordinate system which is:

$$(a.red - b.red)^2 + (a.green - b.green)^2 + (a.blue - b.blue)^2 \qquad (1)$$

Total error encountered in the whole quantization process becomes:

$$\sum_{i=1}^{n}((a_i.red - b_i.red)^2 + (a_i.green - b_i.green)^2 + (a_i.blue - b_i.blue)^2) \quad (2)$$

where n is the total number of pixels in the image, a_i is the color vector of the ith pixel of the original image and b_i is the corresponding color vector in the generated image. Since the error function defined as above is differentiable, methods using the gradient vector such as gradient decent and artificial neural networks can be applied. Some of the methods can be found in "Artificial Intelligence" [3] by Rich and Knight. We used this error function to accept or reject an operator in valley descending and simulated annealing, and also to measure the quality of the solution obtained for each method quantitatively.

Even if optimal methods are suggested and implemented to minimise the quantization error for the above error definition, the results obtained from the quantization are not satisfactory if no further processing is applied. The main reason for that is smooth transitions in the original image are lost due to quantization (clusters of same colors occur), and contours occur at the boundaries where a slight change in the original image result in sensible changes in the resultant image.

The image obtained can be improved drastically by applying dithering methods which use the averaging property of the eye to reduce the error perceived even if, in some cases, the error measured in the above error definition is enlarged. Applying a quantization method and then applying a variant of error diffusion is the most common color depth reduction method used nowadays.

Using a stochastic method in choosing which representative color to use for a given pixel may also help in avoiding the artificial edges and contouring effects. We developed a probabilistic palette assignment method to explore this possibility which shall be described later on.

Dithering methods use the averaging property of the eye, and the range of the colors that can be represented using dithering is improved if the colors are distant from each other. This can be visualised easily with an example. Let us suppose that we have two colors: white and black, and we want to generate a number of grey levels. We can use neighbouring pixels of different proportions of white and black pixels to generate grey levels, the more there are black pixels, the darker the grey level etc. Obviously we cannot obtain colors darker than the given black or lighter than the given white using this method. Traditional quantization methods do not take into account that dithering methods work better if the representative colors chosen are at the edges of the corresponding “color clusters” so that the ranges of colors obtainable using dithering is increased. In fact if we have very fine resolutions by using just three colors: red, green and blue we can construct all of the colors that can be obtained in the triangle shaped region drawn between these three colors in the CIE color coordinate system. This method is used in CRTs and television sets.

Implementation Details

The image format that we selected is Truevision's 24 bits uncompressed Targa format which is a truecolor format where each red, green, and blue components are assigned 8 bits allowing the storage of any of the pixel pack formats proposed by Vesa® without loss if the alpha bits are not used for special purposes. This is a widely recognized format and it is fairly easy to convert any type of image to this format using graphics packages such as Paint Shop Pro, Photo Shop, Corell Draw etc. Although paletted images are grown in size due to conversion, and the information about possible number of different colors is lost, we can reconstruct the histogram in a couple of seconds thanks to the optimized histogram building algorithm that we optimized.

The Targa format has a header of 18 bytes which includes the x and y sizes of the image and some parameters differentiating this format from other Targa formats. Each x and y size are assigned 2 bytes allowing up to 65536 by 65536 images.

We developed two template classes: Set, and Vector in order to obtain a better, flexible, and reliable code. Set is a doubly linked list of a generic element type. It has a couple of different constructors, many methods allowing easy traversing in both directions, methods to detect errors, seek an element, overloaded equalization operator, copy constructor, destructor etc. Any function can accept as both value and reference parameter set and return a set as a function return value (these operations require a copy constructor). Moreover any set can be made equal to another set which requires many operations that must be done transparent to the user of the class: first the destructor is called deallocating the memory assigned for all nodes in the first set, then memory is allocated for a new node corresponding to each node in the second set and elements are copied by calling the equalization operator of the element which can also be sets or any other

classes. For example you can have sets of integers, sets of doubles, sets of vectors, sets of sets of integers, sets of sets of sets of vectors etc. Thanks to object oriented features you can use sets very efficiently without fear of bugs once you are sure that the template class is bug free. The Set class requires that its elements have a default constructor and a "=" (equalization) operator to exist. The seek method requires that "==" (equality checking) operator to be overloaded. We also overloaded the "<<" operator of output streams for the set class so that we can print sets just like other types.

This code fragment shows how practical and powerful to use the set class:

```
#include <iostream.h>
#include "Set.h"

typedef Set<int,int> IntSet;    // define set of ints using ints as indices.
IntSet set1(3,1,2,3);                   // set1={1,2,3}
IntSet set2=set1;                       // set2={1,2,3} (copy of set1)
typedef Set<IntSet,int> SetofIntSet  // define set of integer sets
SetofIntSet SET(1,set2);                // it has one element: a copy of set2

void main(void)
{
        int temp;
        set1.GetFirst();set1.GetNext();         // go to second element
        set1.SetAt(5);                          // set1={1,5,3}
        SET.Add(set1);                          // SET={{1,2,3},{1,5,3}}
        cout << SET << '\n';
        for(set1.GetFirst();!set1.Failure();set1.GetNext)
                cout << set1.GetAt() <<',';
}
```

And here is the output of the code:

```
{ {  1,  2,  3 }, {  1,  5,  3 } }
1,5,3,
```

The vector class is similar in nature, it allows pass by value, pass by reference, has default constructor, a destructor and many operators overloaded allowing vector addition, subtraction, dot product, product by a double, Euclidian distance, norm and many methods such as normalize, getdim, get, set, fill etc. Moreover the dimension of the vector can be changed after it is declared. (In fact the default constructor constructs a vector of zero dimension, then you can freely change the dimension by assigning another vector using equalization operator, or by calling another constructor which allow direct initialization with parameters). The output stream's "<<" operation is also overloaded to print vectors. For example if "s" is a set of vectors having elements two vectors: (1,2) and (5,3) the statement "cout << s;" results in the output { (1,2), (5,3) }.

There are many assertions in the definitions of both of the classes which allows early detection of bugs such as trying to go to the next element when we are already at the end or trying to get an element when the set is empty etc. Note that these assertions are included to the code only in debug versions, in the final version they are ignored and thus do not reduce performance.

For more information about these class templates see the appendix where all of the codes with detailed comments are given or examine the files Set.h and Vector.h

Building the Histogram

Most of the quantization methods use the histogram of the image. Some of them just use the pixels in the original image but by using the histogram instead of the image itself drastically improves performance. In some algorithms we just use the different colors and in some others we also use the number of occurrences of each color. It is fairly easy to construct the histogram of a paletted image where standard palette sizes are 16 and 256. For a palette image of 256 colors, it is priori known that at most 256 different colors can coexist in the image. We can keep an

array of size 256 where each element is set to 0 initially, and for each pixel that we process we can increment the color counter in the corresponding array element. This method requires only one pass of the image thus has the time complexity O(s) where s is the number of pixels in the image which is (x size) * (y size).

Unfortunately we cannot use this method for truecolor images where the number of possible different colors is only limited of the smallest of these two:

- the number of pixels in the image
- the number of different colors that can be constructed using the current truecolor mode.

For example for a 24 bits truecolor 640 by 480 image, number of different colors that can be obtained is 256x256x256 which is 16,777,216 and number of pixels in the image is 640x480 which is 307,200.

The obvious solution is to initialize a set of histogram nodes containing the color and a counter (which counts the occurrence of the color) to an empty set and for each pixel in the image, check if its color is already in the histogram; if it is we increment the corresponding counter; otherwise we add a new node to the histogram set containing the color of the pixel with associated counter set to one. The performance of the algorithm depends heavily on the time complexity of finding the color in the histogram if it exists, or determining that it does not exists if this is a new color.

A hashing algorithm is developed in order to speed up the process. Each color is mapped to 766 different sets according to the sum of its red, green, and blue components. And this sets are than linearly searched for occurrence of current color. In the best case the algorithm works in time complexity O(s), if all the sets contains at most 1 element, and O(s*s), if all colors are hashed to a single set where s is the total number of pixels in the image.

Application of the hashing method greatly improved the performance of building the histogram : as an example the program can construct the histogram of an 579 by 824 (477,096 pixels) image having 91,944 different colors in less than 10 seconds on 150Mhz Cyrix 6x86 which is comparable to existing commercial packages. Note that the method works very fine for images having a few thousands of different colors due to the relatively small number of hashing sets.

An alternative approach which may give better results in the case of large numbers of different colors can be the usage of balanced trees which will result in time complexity of O(s*log(s)). Note that using a large number of hashing sets and a good hashing function we can effectively approach complexities of O(s).

Quantization Methods

The program developed have six different quantization methods and three different palette assignment methods implemented. Thus for each sample of quantization methods three different images are constructed and saved to files. The parameters are passed from the command line. Here is the help screen which is displayed if the number of parameters passed is not sufficient:

```
Usage: Quantize source.tga colors samples [methods]

Source is the picture in uncompressed 24 bit targa format.
Colors is the number of colors in the destination image.
Samples is the number of samples taken for each method.
Methods is optional and must be a list of method numbers.

These methods are implemented:
  1:First K Modes
  2:K-Means
  3:Recursive Splitting
  4:Particle Simulation
  5:Valley Descending
```

6:Simulated Annealing

Each sample of the methods will result in three images:

1:Nearest Neighbor assigned (no dithering)

2:Floyd-Steinberg, error diffused

3:Probabilistic assignment

The command:

quantize picture.tga 32 20 2 3 5

applies the methods 2, 3, and 5 on the image “picture.tga”, where 20 samples are taken for each method and the number of colors in the resultant images is set to be 32. The program will time each sample, store the error defined in equation (2) for each sample and obtain mean time and its standard deviation, mean error and its standard deviation for each method. Some of the methods starts with random initial values, whereas some of the methods uses random numbers in the algorithm allowing each run to result in different images. The standard deviation in both time and quality is important. Note that If only one sample is taken, standard deviation calculations are bypassed in order to avoid division by zero.

The quantization methods that we called recursive splitting and particle simulation are developed throughout this project. The probabilistic palette assignment method is also developed by the author. Although similar methods could exist we are unaware of such methods and developed the three methods independently. Before explaining each quantization and color assignment methods in detail let us give some common assumptions:

- N is the number of colors in the original image, K is the desired number of colors in resultant images.

- The histogram is obtained and placed in the set H having entries for each color occurring in the image along a counter of occurrence for each color.

- P will be the palette, the set of colors that will be in the resultant image. Note that P will also be a set of histogram nodes thus will also have a count field which is used in some algorithms.

First K Modes

This is a very simple and fast method. It just finds the most occurring K colors in the histogram and sets the palette to these K colors. Here is the algorithm:

```
Add to P the first K histogram nodes in H

for(each color in the histogram from K+1 to N) {

        find  c = color having the minimum count value in P,

        if H.current_color.count is larger than c.count

                replace c with H.current_color in P;

}
```

K-Means

This is a hard decision version of k-means clustering algorithm. We start with a precalculated or randomly selected palette of K entries. The principle is to assign each pixel in the original image to the nearest color in the palette in terms of a distance measure which is in general Euclidian distance in RGB coordinate system; thus we obtain clusters for each color in the palette. The new value of each color in the palette is calculated by averaging the colors of each pixel in the cluster corresponding to the palette color. We do not use the whole image but the

histogram in order to speed up the process by using the count values kept for each color in the histogram. Here is the final algorithm:

Initialize P to contain K histogram nodes.

For each K nodes in P allocate space for four counters: avr, avg, avb, and count

1: reset all counters to zero

For(each node in H){

find the nearest color in P in terms of Euclidian distance to the current color in H.

update the counters of corresponding color in P as follows:

avr=avr+H.current_color.red*H.current_color.count

avg=avg+H.current_color.green*H.current_color.count

avb=avb+H.current_color.blue*H.current_color.count

count=count+H.current_color.count

}

calculate the new coordinates of each color in P using the counters

if any of the palette colors is changed go to 1.

Note that by using just the histogram instead of the image itself, we did not changed the result (except some possible round-off errors) since we weight each color in the histogram with its occurrence counter.

Recursive Splitting

In this method we start with a single large set containing the colors in the histogram. At each iteration we select the largest set (the set having largest cardinality) and split it into two sets until we have K different sets. The splitting process consists of finding the component among red, green, and blue components having the largest spread (having largest max-min value), we find the average in the selected components and remove from current set all entries having selected component greater than the average and, put them to a new set. When we have K sets we calculate the average color in each set and add an entry to the palette having the average color calculated. Here is the algorithm:

let number of sets be 1

let P be empty initially

let the first set contain all colors in the histogram, and all other K-1 sets be empty.

While number of sets is less then K do {

find the largest set among current sets,

find the component having the largest spread

find the average according to the selected component

remove all entries having component greater than the average and put them to a new set.

}

Calculate the average color in each set and add a corresponding entry in P.

Particle Simulation

In quantization we want minimum quantization error and in dithering we want relatively distant colors so as to be able to approximate large amounts of intermediate colors by using combinations of the distant colors. Most quantization techniques does not consider post dithering thus only tries to minimize the quantization error. By using the averaging property of the eye, we can obtain better looking images if we try to minimize the quantization error while at the same time trying to select distant colors although the error measured in (2) may become larger.

We tried to develop an algorithm which we thought may achieve these goals.

The pixels in the histogram will be represented by points or "particles" located according to their color in RGB color space. We shall call these points stationary points which will not be moved during the iterations having a mass proportional to the number of occurrence of the color.

Let us assume that there will be n stationary points each having a corresponding weight w_i and we want to approximate the image using k different colors.

k new free points will be added to the system which will be free to move at each iteration. The initial location of the points may be chosen from the most frequent colors in the histogram or chosen randomly. Each free points will have a mass of *N/K/alpha* where N is the total number of colors in the original image and K is the number of colors to which we intend to reduce the image. Alpha is a parameter of the system, which is around 10 in current implementation.

Each free point will be attracted by the stationary points according to the weights of the stationary points and will be repelled by other free points. The

direction and magnitude of the “net force” on each free point will be calculated and the point will be moved according to the calculated values. Attraction and repulsion will be inversely proportional to a function of the distance *d* between the points, and directly proportional to the weights of the points. This functions is selected to be $f(d)=d^2$ analogous to Universal Gravitation although other functions can also be selected.

Weights associated to the free points is very important in obtaining good results since too much weight may result in divergence and too little weight may not satisfy the requirement of distant representative colors.

The process will stop when a predetermined number of iterations is reached. Let us now give the algorithm:

> “Cluster points” are the free points, and “ordinary points” are the static ones. Coordinates of ordinary points are of discrete type (0..255) where as the coordinates of cluster points are of floating point type.
>
> maxdelta=10, mindelta=1, deltadec=0.3
>
> cluster_point_multiplicator=0.01 // these values can be changed
>
> ordinary_point_multiplicator=0.01 // to obtain different results
>
> alpha=10
>
> cluster_point_mass=N/K/alpha
>
> ordinary_point_mass=1
>
> randomly initialize the location of cluster_points
>
> ordinary_points have locations corresponding to the color they represent.
>
> delta=maxdelta
>
> while(delta>mindelta) {

for each cluster point calculate the netforce1 due to ordinary points

where the contribution of each ordinary point is: (let p be the current ordinary point)

(cluster_mass*point_mass*p.count/square(Euclidian_distance)) * unit direction

for each cluster point calculate the netforce2 due to other cluster points where the contribution of each cluster point is:

(cluster_mass*cluster_mass/square(Euclidian_distance)) * unit direction * (-1)

for each cluster point let

netforce = (cluster_point_multiplicator*netforce2 + ordinary_point_multiplicator*netforce1)

Normalize the netforce and multiply it with delta. (so as to avoid big jumps in coordinates)

Move each cluster_point in the direction and amount suggested by the corresponding netforce.

If any coordinate of the cluster_points is less than 0.0 set these coordinates to 0.0

If any coordinate of the cluster_points is larger than 255.0 set these coordinates to 255.0

delta=delta-deltadec // delta must be decreased in some way.

}

Note that some optimizations can be done such as not to multiplying with cluster_mass in the calculation of both netforces, but we left it there to emphasize

the analogy to Gravitation. This algorithm is very new and most of the parameters can be optimized to obtain better results. The attraction and repulsion formulas can be changed; in fact we tried dependence on other powers of distance for one or both of the forces. The alpha constant is very crucial to obtain equilibrium: if the repulsion are given too much weight the points diverge, if no weight is given the nearby initial points tends to converge to the same or very near positions which effectively reduce the number of available colors. Once the algorithm is finalized optimizations can be started. We obtained satisfactory results in relatively small amounts of times (about 50 seconds) for small images (such as 256 by 256 images having about 200-250 different colors). The method can be improved to give better results for better time complexities.

Valley Descending and Simulated Annealing

The simulated annealing is a slightly modified version of the Valley Descending algorithm. Both of them uses the error function defined in (2). They turned out to be impractical for large images since the calculation of the error function is too costly if the number of colors in the original image is high. Most of the time is spent in the calculation of the error. Apparently the error function has time complexity O(N*K) and requires many floating point operations. Unlike the particle simulation method the number of iterations is not bounded, the process continues until a local minimum is found. Thus by increasing the number of colors in the original image we both increase the time required to calculate the error and increase the number of iterations needed to hit a local minimum.

We start with K entries in the palette either preselected or chosen randomly. We tried to select the operators so that minimum change will occur by applying an operator. The operators are defined as increasing or decreasing the red, green or blue component of a color in the palette by 1 which makes 6 possible operators for each color in the palette. Since we have K colors in the palette we have at most 6*K different operators to choose from at each iteration (note that if

some components of the colors are at the boundaries (0 or 255) the number of possible operators are decreased.) Since the error function is costly we mark all the tried and rejected operators and chose a new operator randomly among the remaining operators until an operator is accepted or all the operators are marked as tested in which case a local minimum is found. An operator is accepted if applying it, results in smaller error in the case of valley descending, and if exp((initial error-final error)/temperature) is larger than a selected random number in the case of simulated annealing. This function results in acceptance of all of the operators resulting in better states and some of the operators resulting in worse states. If temperature is high, acceptance probability of any operator is high, and when temperature is low, only small amounts of operators resulting in worse states are accepted. We start with a high temperature and at each iteration we decrease the temperature, thus "cool the system". At the beginning simulated annealing process works more or less randomly and at the end of the process, when it is cooled enough it becomes more or less like valley descending since only good moves are accepted most of the time. The simulated annealing has a chance to avoid local minimums by the randomness effect in the beginning of the process.

In the case of simulated annealing the stopping criteria is:

- temperature should be less than a predetermined threshold,
- a local minimum should be detected,
- the best_so_far state (that we update each time a new state is better than best_so_far) should not be better than current state.

If best_so_far is better than the local minimum state, we go back to best_so_far state and start a new iteration. Note that at the beginning of each iteration all of the possible operators are unmarked.

A rule of thumb that is generally applied in simulated annealing states that the initial temperature should be selected so that about half of the initial operators should be accepted, but in the case of this project selecting such a number is impossible due to wide variety of images that are possible requiring selection of temperatures for each type of image specifically. Maybe a function of N and K can be found which will work for most of the images but it is not easy to find and express this function.

Since valley descending has also a randomness in that with different initial seeds it can converge to different local minimums, applying simulated annealing thus increasing the time needed to converge considerably is not compensated by the slight improvements in the quality of the solution.

Note that we can improve the results obtained by allowing larger changes to be accepted by redefining the operators: allowing more than one components to be changed at a time or increasing or decreasing a component by a value larger than one. Note that these approaches may decrease the probability of being stuck in local minimums but in this case marking rejected operators becomes impractical due to large numbers of operators that are possible.

Palette Color Assignment Methods

Nearest Neighbor

In this method for each pixel in the original image, we find the nearest pixel in the palette in terms of Euclidian distance. Each pixel is mapped to a single color in the palette. This is the simplest palette color assignment method which results in loss of smooth transitions and adds artificial edges and contouring effects to the resultant image. Thus generally after the quantization process, a dithering method is applied such as the Floyd-Steinberg error diffusion method while building the resultant image.

Error Diffusion

In this project we used the Floyd-Steinberg error diffusion method for dithering as an example of classical methods. The method is described in detail in Quantization and Dithering section of this document.

Probabilistic Assignment

The probabilistic assignment method is developed during this project and is proposed as an alternative to other diffusion algorithms such as the error diffusion method used in the program. This algorithm also tends to preserve smooth transitions in the original image.

The method is based on assigning for each pixel, a color from the palette stochastically. Once we have a pixel to approximate we calculate a probability of being able to represent the color in the resultant image for each color in the palette. We used a function of the Euclidian distance between the color to be approximated and the colors in the palette in order to assign the probabilities. It is natural to assign low probabilities to distant colors and high probabilities to near ones. The selection of the function is very crucial in obtaining good results. If the function gives equal or comparable weights to both near and distant colors we end up with an image looking as if a large random noise were applied on it. If nearby pixels gets too much weight the image looks much like the image obtained from nearest neighbor assigned image. Thus the selection of the function is very important.

We selected a sigmoid like function which is: f(x)=1/(1+exp((x-3)*mul)) where mul is a parameter that we found out that values around 0.001 gives very satisfactory results for every image that we tried.

Let us give examples of three palette assignment methods:

The image is a 256 by 256 image having 230 different colors. I selected this gray level image which become a standard demonstration image in computer

graphics society since this is a relatively small image and all of the methods can be applied in reasonable amounts of time. The number of colors is reduced to 16 for all of the methods.

Quantization method: First K-modes

Original image

Nearest neighbor assignment

Error diffusion

Probabilistic color assignment

Quantization method: K-means

Original image

Nearest neighbor assignment

Error diffusion

Probabilistic color assignment

Quantization method: Recursive splitting

Original image

Nearest neighbor assignment

Error diffusion

Probabilistic color assignment

Quantization method: Particle simulation

Original image

Nearest neighbor assignment

Error diffusion

Probabilistic color assignment

Quantization method: Valley descending

Original image

Nearest neighbor assignment

Error diffusion

Probabilistic color assignment

Quantization method: Simulated annealing

Original image

Nearest neighbor assignment

Error diffusion

Probabilistic color assignment

Conclusions

For small images having relatively small number of colors all of the methods can find a solution in a reasonable amount of time. But for large, truecolor images the winner in both performance and quality is the recursive splitting method that we developed, whereas the k-means clustering method was the second best.

The recursive splitting method is the fastest method among the implemented methods if we exclude the first k-modes method which is only used to initialize the palette of some other algorithms such as valley descending, k-means etc. It also resulted in similar quality images as k-means method. Note that the recursive splitting is deterministic while k-means algorithm depends on initial conditions although the quality of the image were acceptable for all of the different initial conditions that are tried.

The valley descending and simulated annealing methods turned out to be inadequate for color quantization as they require large amounts of time and the variance in quality is high compared to other methods if we exclude particle simulation. Sometimes good results are obtained and sometimes they are stuck to a local minimum having a large error.

The following data is directly taken from the output of the program which is run for the image Lena.tga for 16 different colors and for 10 samples of each method. The image is 256 by 256 and has 230 different colors. The command used to run the program was: quantize lena.tga 16 10

```
S U M M A R Y

Number of samples taken for each method:10

First K Modes
  Mean time needed    :      1.03125 ms Standard deviation :      0.37702 ms
  Mean error          : 1.48403e+006    Standard deviation :            0

K-Means
  Mean time needed    :          141 ms Standard deviation :      37.2529 ms
  Mean error          :       764189    Standard deviation :      65551.5

Recursive Splitting
  Mean time needed    :        13.75 ms Standard deviation :      1.31762 ms
  Mean error          :       403873    Standard deviation :            0

Particle Simulation
  Mean time needed    :        48940 ms Standard deviation :      26655.2 ms
  Mean error          : 1.36301e+006    Standard deviation :       582263

Valley Descending
  Mean time needed    :        30446 ms Standard deviation :      12706.6 ms
  Mean error          :       903544    Standard deviation :       168933

Simulated Annealing
  Mean time needed    :        64273 ms Standard deviation :      20283.8 ms
  Mean error          :       707238    Standard deviation :      99778.6
```

Note that the standard deviation of the error in recursive splitting is zero since the method is deterministic.

We conclude from the data that simulated annealing resulted in better images in terms of quantization error than the valley descending method but its time requirement was more than twice. We can associate this fact to the better behavior of simulated annealing in local minimums especially if the temperature is high.

As can be seen particle simulation resulted in large errors compared to other methods but for the case of Lena it took less time than simulated annealing. Maybe better results can be obtained by adjusting the parameters of the system in the future.

Let us now compare the performance of recursive splitting algorithm and k-means clustering for large true color images (we don't even bother running other methods, since they are too slow). We will try three different images: Peppers.tga, Boots.tga and Hajime56.tga.

Peppers.tga, 86786 colors are reduced to 32 colors:

```
S U M M A R Y

Number of samples taken for each method:1

K-Means
  Mean time needed    :       414030 ms Standard deviation :            0 ms
  Mean error          :  3.8157e+006    Standard deviation :            0

Recursive Splitting
  Mean time needed    :        54430 ms Standard deviation :            0 ms
  Mean error          :  4.4386e+006    Standard deviation :            0
```

Here are the images obtained:

Original image

K-means, nearest neighbor assigned.

K-means, error diffusion.

K-means, probabilistic assignment.

Recursive splitting, nearest neighbor assignment.

Recursive splitting, error diffusion.

Recursive splitting, probabilistic assignment.

Boots.tga, 86,786 colors are reduced to 32 colors:

```
S U M M A R Y

Number of samples taken for each method:1

K-Means
  Mean time needed    :       354660 ms Standard deviation :            0 ms
  Mean error          : 4.10924e+006    Standard deviation :            0

Recursive Splitting
  Mean time needed    :        73610 ms Standard deviation :            0 ms
  Mean error          : 4.91863e+006    Standard deviation :            0
```

Original image.

K-means, error diffusion.

K-means, probabilistic.

Recursive splitting, error diffusion.

Recursive splitting, probabilistic.

The probabilistic color assignment method also resulted in images comparable to other dithering methods in quality, it suppresses contouring effects and resulted in images that look like the original image on top of which a small amount of random noise is added. It can be used as an alternative to the current dithering methods if such a noise effect is also desired.

Hajime56.tga, 91,944 colors are reduced to 32 colors:

Original image

Recursive splitting, nearest neighbor assignment.

Recursive splitting, error diffusion.

Recursive splitting, probabilistic color assignment.

Future Works

Most of the quantization methods tries to minimize the error defined in (2). They mostly rely on RGB color system. It may be a good idea to choose another color system and develop new quantization methods for this system. Quantization can be applied in a coordinate system where differences in each direction are perceived more or less in the same manner by the eye which is not the case in the RGB color system.

Particle simulation method can be improved and optimized further by trying different parameters and attraction/repulsion forces.

Number of operators in valley descending and simulated annealing can be increased in order to allow large steps in state space thus allowing the system to get rid of some local minimums more easily.

Instead of using the hashing function one may try to use balanced trees or similar data structures while building the histogram. Alternatively different hashing functions probably having larger hashing sets can be tried.

Median cut and recursive splitting algorithms can be compared in terms of time requirements and quality of solutions.

New functions can be used in probabilistic assignment method such as the normal distribution.

Glossary

alpha: Region of the pixel pack not used by the display adaptor to calculate the color, it can be used by the program to store additional information. It is used to set the pack size to 16 bits or 32 bits.

clustering: Process of dividing a set of values or vectors into local regions of smaller sizes. In the context of this document we talk about clustering the colors in the histogram into smaller regions where each region is approximated by a corresponding color in the palette.

color depth: Number of different colors that can be used. In true color modes, number of colors in the color space.

color gamut: Total color space, or total range of colors that can be displayed by the system. For an RGB monitor it is the triangular area inside the red, green, and blue phosphor's colors in the CIE chromaticity diagram.

color space: Total range of colors that a system can produce.

dithering: see Quantization and Dithering.

error diffusion: see Quantization and Dithering.

quantization: see Quantization and Dithering.

sigmoid function: The sigmoid function f(x) is 1/(1+exp(-x)) and is used extensively as a thresholding function in Artificial intelligence due to its differentiability. As x goes to minus infinity f(x) approaches zero, and as x goes to infinity it approaches one asymptotically. We can practically say that it is close to zero and one if x is below -3 or above 3.

References

[1] D. Hearn, M. P. Baker, “Computer Graphics” second edition, Prentice Hall 1994.

[2] Video Electronics Standards Association, “VESA® Bios Extention (VBE) Core Functions Standard” , version 2.0, revision 1.1, November 1994. (can be downloaded from VESA®: http://www.vesa.org)

[3] E. Rich, K. Knight, “Artificial Intelligence”, second edition, Mc Graw Hill 1991.

[4] M. T. Orchard, C. A. Bouman, “Color Quantization of Images”, IEEE Transactions on Signal Processing, vol 39, no 12, December 1991.

[5] Foley, vanDam, Feiner, Hughes, “Computer Graphics”, second edition, Addison Wesley, 1990.

Appendix

The program is compiled using Microsoft Visual C++ 4.0 for Win32 environment and requires either Win95, Win NT or Windows 3.1 with Win32 installed to run. It is ANSI C++ compliant and should also work on other compatible platforms.

The files required to compile the program are:

- Quantize.cpp, main program,
- Set.h, header containing the definition of the class template “Set”,
- Vector.h, header containing the definition of the class template “Vector”.

These three files are printed in the following pages in the given order.