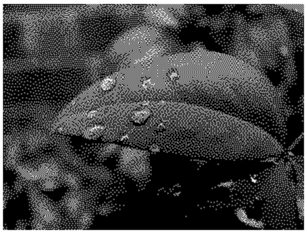
Multimedia Software Systems
CS4551

Digital Image Data Representations
24bit to 8bit Color Quantization

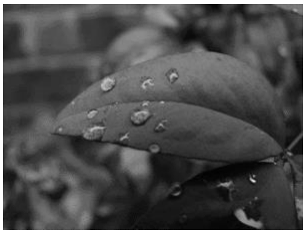
Image

- Consists of *pixels* (picture elements).
- 2D array of pixel values.
- Pixel depth of pixel
  - 1 bit (bi-level image or monochrome image)
  - 8 bits (gray image, color image)
  - 24 bits (color)
  - 32 bits
  - …
- Image resolution
  - the number of pixels in the image. High resolution yields better visual quality.
  - Computed by *pixels in width* x *pixels in height*. (eg. 640x480)

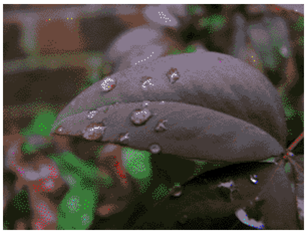
Images – Color Depth



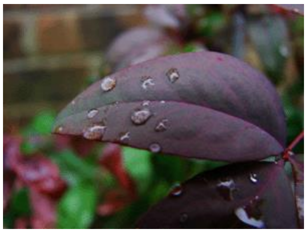
1 bit (2 colors)



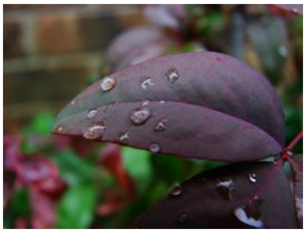
8 bits (256 colors)



4 bits (16 colors)



8 bits (256 colors)



24 bits (2^{24} colors, truecolor)

Images from wikipedia

Binary, Decimal, and Hexadecimal

Binary	**Decimal**	**Hexadecimal**
00000000	0	0x00
01000110	70	0x46
10011111	159	0x9F
11111111	255	0xFF

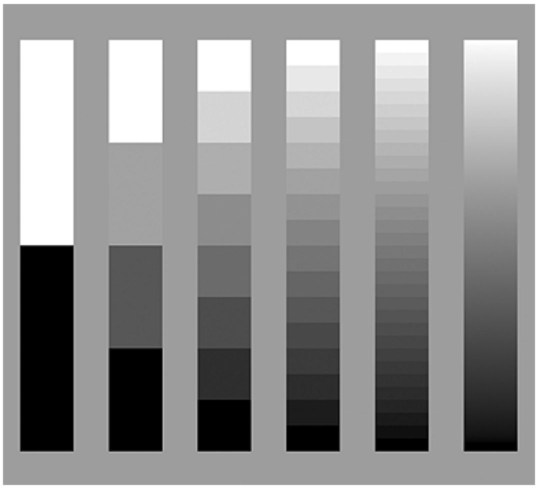
Images – 1-bit Images

- 1-Bit Images
  - Each pixel is stored as a single bit.
  - File size of 640x480 bi-level image : **37.5 KB (1KB = 1024bytes)**
  - Good for only simple graphics and text images (eg. Facsimile images)

Images – 8-bit *Gray-scale* Images

- 8-Bit **Gray** Images
  - Each pixel is stored as a byte (8 bits) and has a gray-value between 0 and 255. The smaller value is darker. Pure black is 0 and pure white is 255.
  - File size of a 640x480 8-bit image: **300KB (1KB = 1024bytes)**
  - Dithering : gray-scale to binary (skip for now and will come back)

Gray Scale Tones



Images – 24bit Color Image

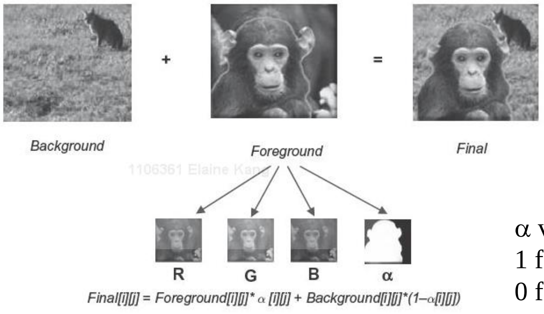
- 24-bit Color Images
  - Each pixel is represented by 3 bytes (24 bits) usually representing RGB.
  - A value of each pixel ranges 0 to 2^{24} -1.
  - RGB Color Chart (https://www.rapidtables.com/web/color/RGB_Color.html)
  - File size of a 640x480 24-bit image: **900KB (1KB = 1024bytes)**

Images – 32bit Color Image

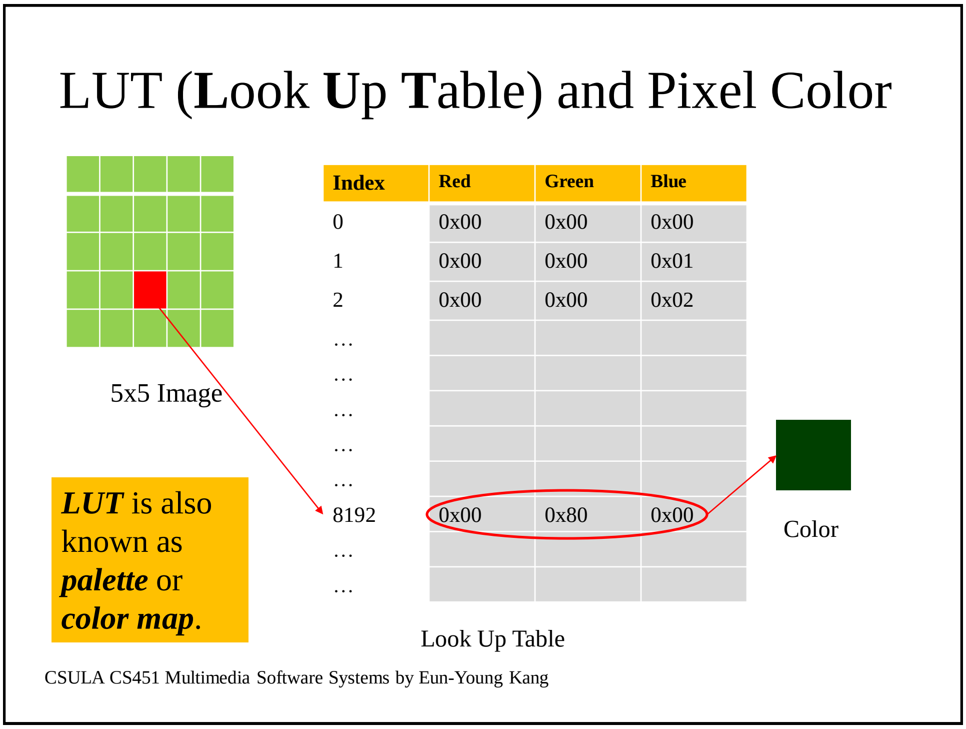
- Each pixel is represented by 4 bytes (32 bits)
- Usually 3 bytes representing RGB. The last 1 byte for **alpha** values.
- File size of a 640x480 32-bit image: 640 x 480 x 4bytes

Alpha Channel

- An additional channel that suggests a measure of the transparency for that pixel value.
- It is used in image **compositing** applications, such as blue screen matting.


Background
+
Foreground
=
Final
1106361 Elaine Kang
R
G
B
α
Final[i][j] = Foreground[i][j]* α [i][j] + Background[i][j]*(1–α[i][j])

α value:
1 for opaque
0 for transparent



LUT (**L**ook **U**p **T**able) and Pixel Color

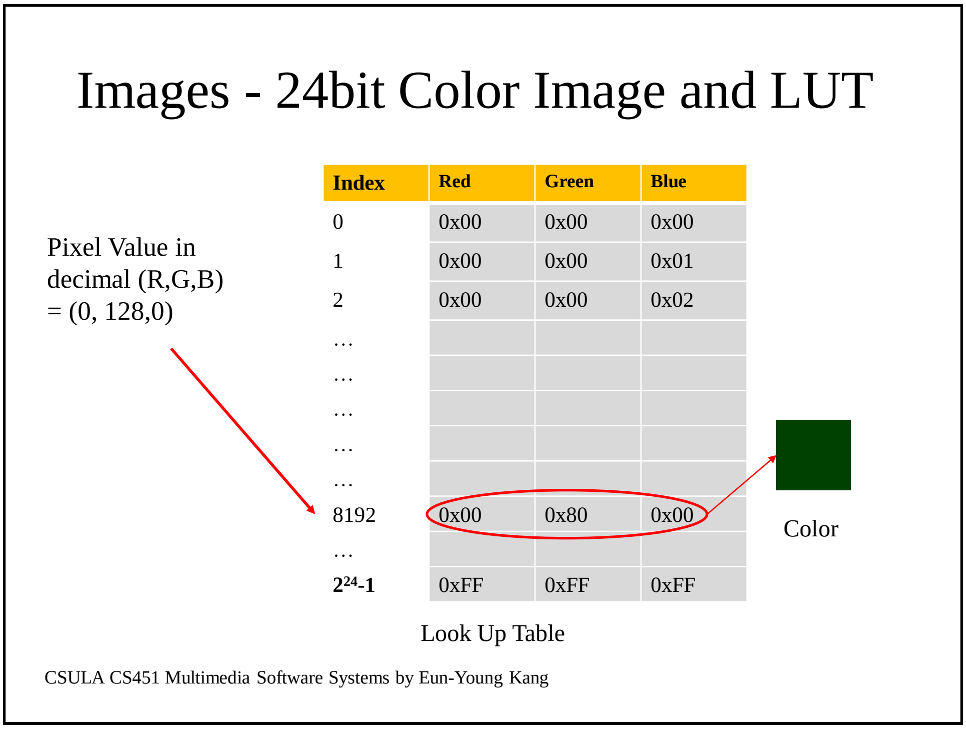
5x5 Image

LUT is also known as ***palette*** or ***color map***.

Index	**Red**	**Green**	**Blue**
0	0x00	0x00	0x00
1	0x00	0x00	0x01
2	0x00	0x00	0x02
…			
…			
…			
…			
…			
8192	0x00	0x80	0x00
…			
…			

Look Up Table

Color



Images - 24bit Color Image and LUT

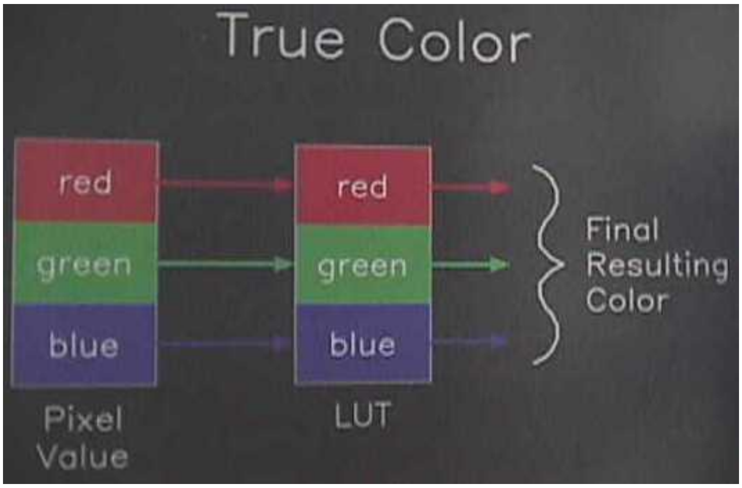
Pixel Value in decimal (R,G,B) = (0, 128,0)

Index	**Red**	**Green**	**Blue**
0	0x00	0x00	0x00
1	0x00	0x00	0x01
2	0x00	0x00	0x02
…			
…			
…			
…			
…			
8192	0x00	0x80	0x00
…			
2^{24}-1	0xFF	0xFF	0xFF

Look Up Table

Color

Images - 24bit Color Image and LUT



Images - 8-bit *Color* Images

- 8-bit Color Images
  - Many systems use only 8-bits (256 colors) per pixel when space is a concern.
  - File size of a 640x480 8-bit image: 300KB
  - Each pixel is represented by 1 byte (8 bits) representing an index to a 3-byte-value in the color Look-Up Table (**LUT**).
  - If a pixel stores the value 25, the meaning is to go to row 25 in the LUT and use the 3-byte-value representing RGB for actual display color.

Images - 8bit Color Image & LUT

Pixel Value is a value in [0,255].

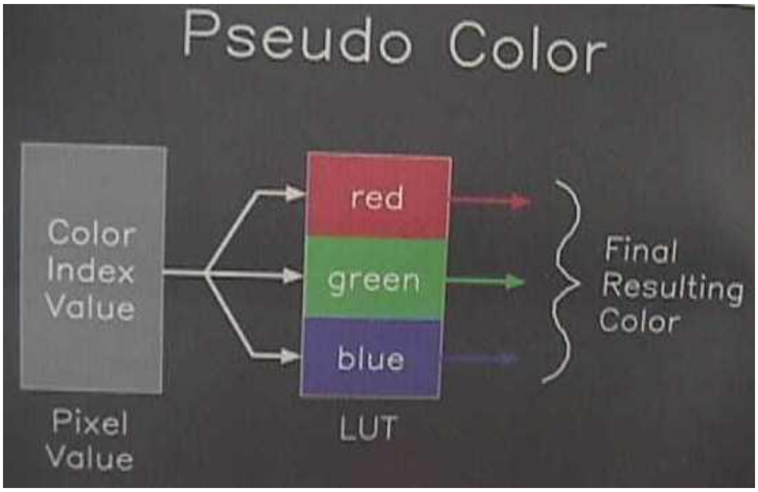
Index	Red	Green	Blue
0	0x00	0x00	0x00
1	0x00	0x00	0x01
2	0x00	0x00	0x02
…			
…			
…			
…			
…			
200	0x00	0x80	0x00
…			
255			

Look Up Table

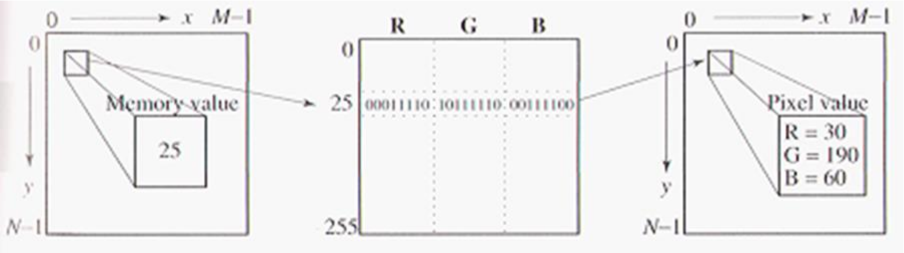
🟩

Color

Images - 8-bit Color Images and LUT



Images - 8-bit Color Images



Display of a 8-bit color image

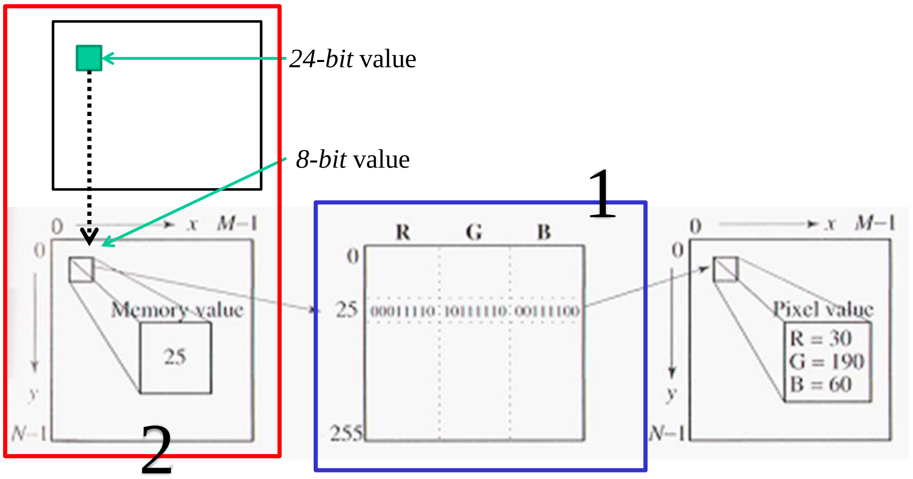
Color Quantization

- Suppose you have a 24bit color image and need to convert it to a 8bit color image. So we have to devise a Color LUT for this image.
- This process is called ***color quantization***.

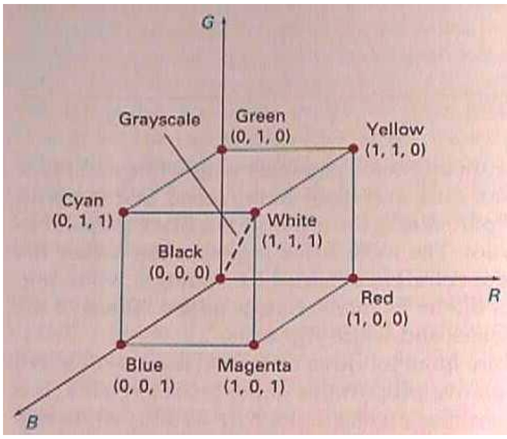
Color Quantization (2)

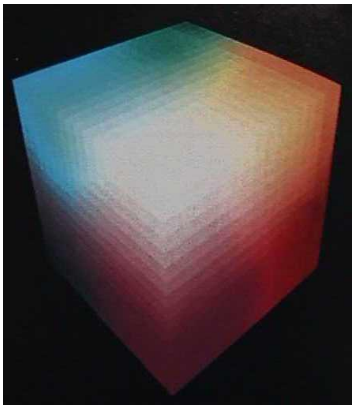
- 2 steps:
  - Step1 : Generate 8-bit LUT
  - Step2 : Convert each 24-bit pixel to 8 bit LUT index
- You may choose the method in two ways
  - Pre-designed or fixed version
  - Adaptively -“quantized” version for the best result. Most adaptive quantization methods are based on popularity.

Color Quantization: 24bit -> 8 bit



RGB Cube





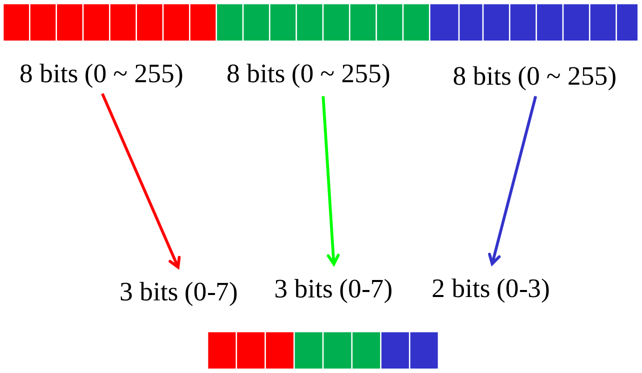
Color Quantization

- Let’s assume that we have 24bit RGB color space and want to quantized in K=256.

Uniform Color Quantization

- Uniform Color Quantization is one of the fixed color quantization.
- Step 1:
  - Divide RGB cube into equal slices in each dimension. That is each axis is divided into equal sized segments. (eg. red and green axis into **8** segments, blue axis into **4** segments resulting into 256 regions.)
  - Each one of these regions will produce a color for the LUT. The representative color for each region is the average (or center) of all colors in that region.
- Step 2: Find a region which the original 24bit color is mapped to. Convert 24-bit to

LUT by Uniform Color Quantization



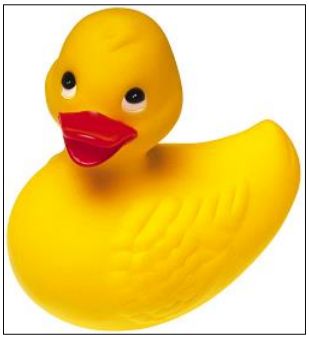
LUT by Uniform Color Quantization

Index	**R**	**G**	**B**
0 (00000000)	?	?	?
1 (00000001)			
…			
255 (11111111)			

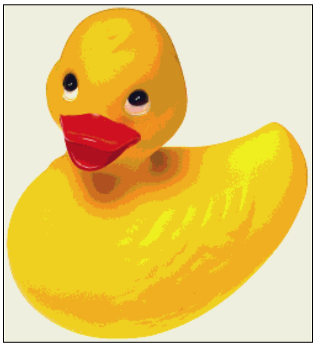
Pixel Conversion by Uniform Color Quantization

- For a 24bit color (255, 255, 100) pixel, when the uniform color quantization is used to convert to 8-bit color image,
  - What is the LUT ***index*** for the pixel***?***
  - What is the RGB ***value*** for that index***?***

Uniform Color Quantization - Example

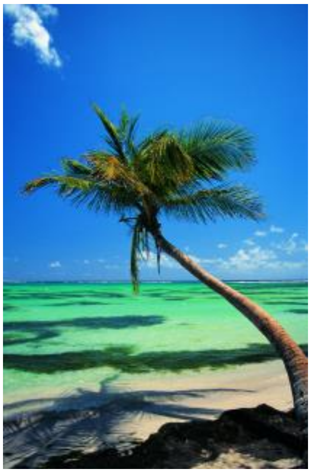


Original

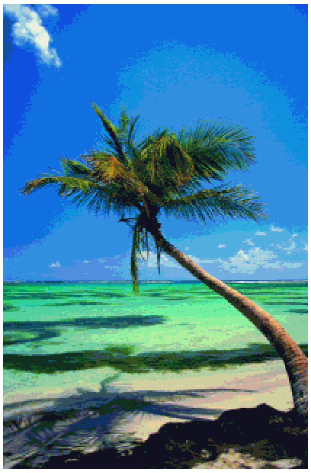


After Uniform Quantization

Uniform Color Quantization - Example



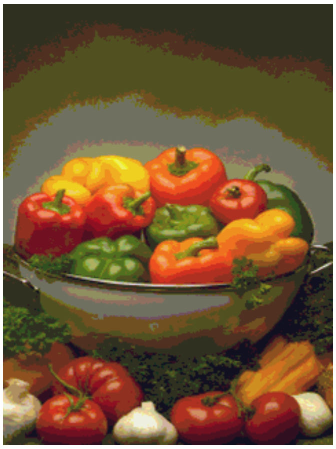
Original



After Uniform Quantization

Uniform Color Quantization - Example

Original



After Uniform Quantization

Uniform Color Quantization

- Pros
  - Quick and easy to implement.
- Cons
  - But it does not yield good results.
- Modification
  - Because the human eye cannot distinguish dark colors as well as bright ones, this algorithm can also be applied in a non-linear manner if the axis are broken on a logarithmic scale instead of linear. This will produce slightly better result.

Adaptive Color Quantization Methods

- The process occurs in stages
  - 1st stage: Sampling the original image for color statistics (Histogram)
  - 2nd stage: Create a LUT based on those statistics
  - 3rd stage: Mapping colors to their representative colors to get new image
- Different algorithms proposed and used depend on how the second step above is done
  - ***Popularity Algorithm***
  - ***Median Cut Algorithm***
  - Octree –quantization

Popularity Algorithms

- Popularity algorithms are a variation of uniform quantization.
- Instead of dividing the color space into 256 regions, the algorithm break into much smaller regions initially (eg. divide 64 segments for each axis resulting in 262,144 regions in total)
- The original colors are again mapped to the regions they fall in. The representative color for each region is the average of the colors mapped to it. The color map is selected by **taking the representative colors of the 256 most popular regions**. If a non-empty region is not selected for the color map, the index into the color map is the entry in the color map that is closest to its representative color.

Popularity Algorithms

- Pros
  - – Easy to implement. Give better results than the uniform quantization algorithm.
- Cons
  - – But it takes slightly longer to execute and requires significantly larger storage requirement depending on the size of regions.
  - – Also, depending on the image characteristics, it may not produce a good result.

Median Cut Algorithms

- The premise behind median cut algorithm is to have every entry in the LUT represent the same/similar number of pixels in the original image. The algorithm divides the color space based on the distribution of the original colors.

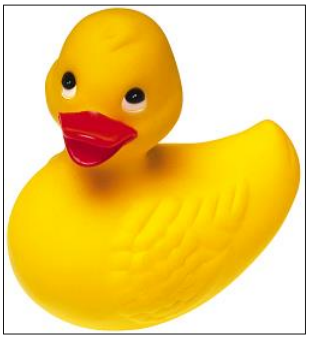
Median Cut Algorithms

- Process
  1. Find the smallest bounding box which contains all the colors in the image
  2. Sort the enclosed colors along the longest axis of the box
  3. Split the box into 2 regions at median of the sorted list
  4. Repeat the above process (1~3) until the original color space has been divided into 256 regions
- The representative colors are found by averaging color in each box, and the appropriate color map index assigned to each color in that box.

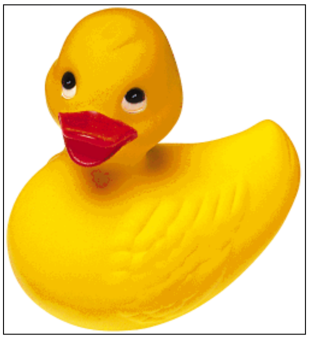
Median Cut Algorithms

- Comments: It produces good results because the algorithm use image information while having memory and time complexity no worse than popularity algorithm.

Median Cut Quantization - Example

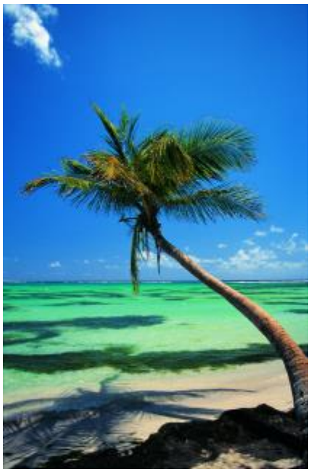


Original

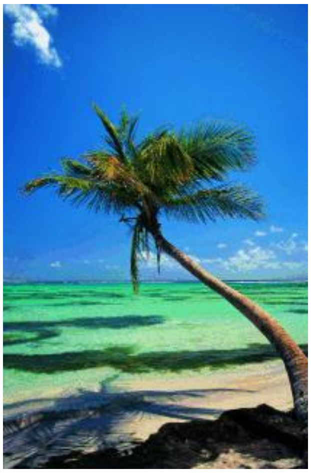


After Median Cut Quantization

Median Cut Quantization - Example

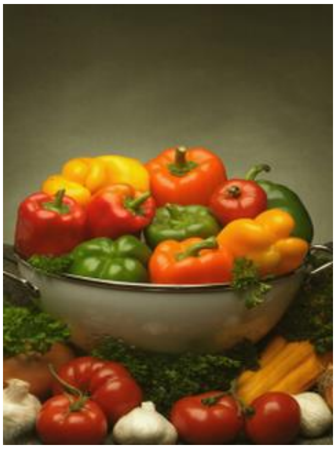


Original

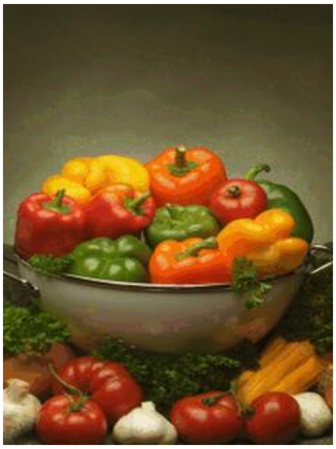


After Median Cut Quantization

Median Cut Quantization - Example



Original



After Median Cut Quantization