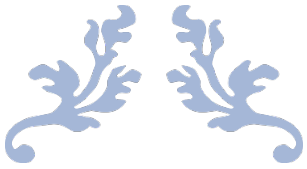


# CSE 534 SPRING2019 MINI-PROJECT

State University of New York at Buffalo

YICHEN WANG

#50161143

# 1. GitHub Link

**https link:**

https://github.com/yichenwa/MultimediaSystemsProject.git

**ssh link:**

git@github.com:yichenwa/MultimediaSystemsProject.git

# 2. Methods Description

## Problem 1- Otsu Method

**Step 1:** get Histogram and probability

**Step 2:** we need to find the threshold t makes within class variance minmum, there are 256 possible t.

within class variance = wight of class1 * sigma_sqr1 + wight of class2 * sigma_sqr2

within class variance = total variance - between class variance, so we can find the max between class variance

Ostu find between class variance = wight1*wight2*(mu1-mu2)^2

**Step3:** pixel <= t set to be white, pixel > t set to be black

## Problem 2-Quadtree Compression

**Step 1:** Split the image into four same size of sub-images. Original image ==>A,B,C,D

**Step 2:** For each sub-image calculate the mean and error.

**Step 3:** Find the sub-image with largest error, (assume it is A), split A as step 1 get $A_1$,$A_2$,$A_3$,$A_4$. Find the sub-image with largest error ($A_1$,$A_2$,$A_3$,$A_4$,B,C,D)

**Step 4:** Repeat until sub-image contains 1 pixel.

If mean square error is small, PSNR will be high and the ratio of Signal to Noise is high, if mean square error is large, PSNR will be low and the ratio of Signal to Noise is low. There is an inverse relationship between mean square error and PSNR.

If mean square error is large, compression rate will be high because more error.

## Problem 3

**Step 1:** In a bucket of pixels:

1-1 Compute the max and min value of R,G,B three channels. maxR/G/B-minR/G/B

1-2 Get the channel with max range, sort all pixels by this channel

1-3 Find the median pixel in the sorted list. Cut the list into two part.

**Step 2:** Repeat Step1 until we get 2^(bit-depth) lists.

**Step 3:** For each list, compute the mean color value and set all pixels in this list with this color value.

## Problem 4

I implement the Octree method and make a little change to reduce its runtime.

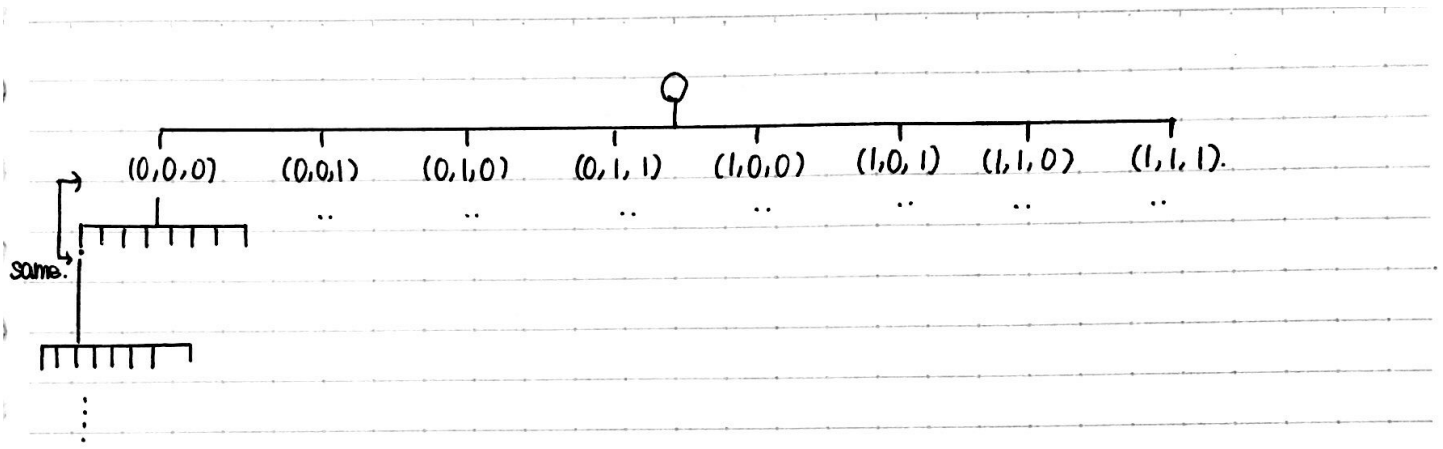
**Step 1:** Create an empty Octree tree. As the most bit-depth we want is 8, 2^8=256, so I create a 4 level Octree tree. On the 4th level, it has 512 leaf. Each leaf represents a range of RGB

**Step 2:** Go through all pixels in the image, add its index to the correct leaf.

**Step 3:** Sort all 512 leaf from the largest length to smallest length. Cut the leaf list into two part: The first 2^bit-depth leaf and the last.

**Step 4:** For each leaf in first part, compute the mean of their color value and set them color value equals to mean.

**Step 5:** For those leaf in last part, find the closest color in first part, and set their color equals to the closest color.



## 3. Quantitative results with discussions/finding

Otsu-t1

Otsu-t2


Otsu-t3


Quadtree-t1

Quadtree-t2

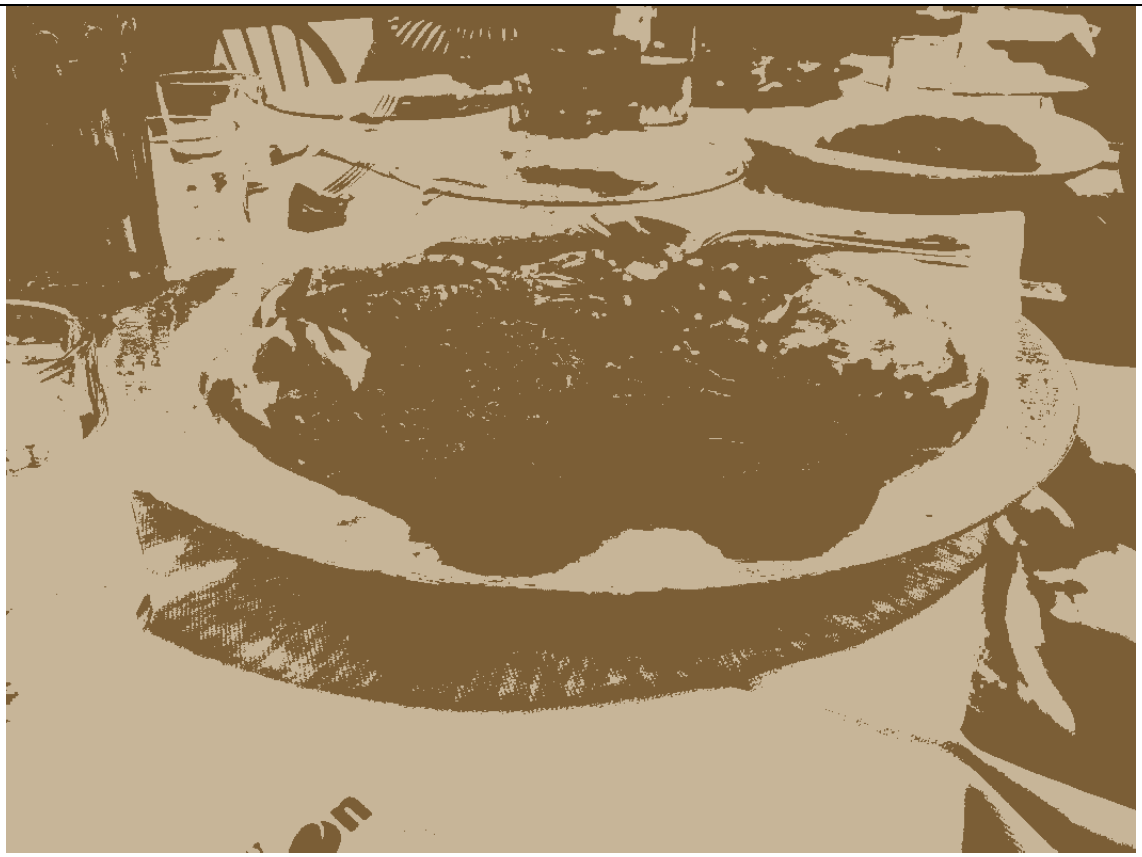

Quadtree-t3

median-cut for t1 1-bit depth


median-cut for t1 2-bit depth


median-cut for t1 3-bit depth


median-cut for t1 8-bit depth



median-cut for t2 1-bit depth

median-cut for t2 2-bit depth

median-cut for t2 4-bit depth

median-cut for t2 8-bit depth

median-cut for t3 1-bit depth


median-cut for t3 4-bit depth

median-cut for t3 2-bit depth


median-cut for t3 8-bit depth

My own quantization method (Based on Octree) t1 1bitdepth

My own quantization method (Based on Octree) t1 2bitdepth


My own quantization method (Based on Octree) t1 4bitdepth


My own quantization method (Based on Octree) t1 8bitdepth

My own quantization method (Based on Octree) t2 1bitdepth

My own quantization method (Based on Octree) t2 2bitdepth

My own quantization method (Based on Octree) t2 4bitdepth

My own quantization method (Based on Octree) t2 8bitdepth

My own quantization method (Based on Octree) t3 1bitdepth

My own quantization method (Based on Octree) t3 4bitdepth

My own quantization method (Based on Octree) t3 2bitdepth

My own quantization method (Based on Octree) t3 8bitdepth

The larger bit-depth we used, the color quantization results closer to the original one, but the runtime is longer.

The smaller size image needs small bit-depth, compare t2 and t3, when bit-depth equals to 4, the t3 output image already very close to the original one, but t2 output not very close to original one.

Compare the median-cut method and my method, we can see there is a big difference between them when bit-depth is small. For example, bit-depth equals to 1, median-cut method selects the "two mean value from two same size bucket" but my method select "two mean of two most common color range". The median-cut method is better for a low bit-depth, but their result of a high bit-depth is very similar.