CMPT-365 Assignment 2 Report

By: Kier Lindsay and Rafid Ashab Pranta

Short Description:

Our program takes an image provided by the user and initially crops it to have width and height that are multiple of 8x8 matrix. This allows pre-calculating the DCT matrix for the image's 8x8 blocks.

First, we convert the RGB image provided to a YUV image. The channels are then separated in 3 matrices. A chroma subsampling between 4:4:4 (lossless), 4:2:2, 4:2:0 can be applied to the U and V channels. This significantly reduces the chromatic information of the image without having a significant impact on the image quality.

After conversion to YUV and chroma sampling. We take every channel of the picture and divide them 8x8 blocks. We apply the DCT function to calculate the DCT coefficients using Matrix Multiplication and the pre-computed conversion matrix.

This DCT coefficient matrix is then quantize if the user enables quantization from the UI, using different quantization tables for Y and UV. We Provide the quantization tables from the slides tuned for best appearance and scale them. These quantization tables determine the loss of quality in the image and can significantly reduce the size of the image in the JPEG Compression Algorithm by getting rid of high-frequency DCT coefficients of the image.

To show the impact of these algorithms in the image. We apply the inverse quantization matrix on the quantized matrix. This might returns us a DCT coefficient matrix that is different from our original DCT matrix due rounding off error when we initially apply Quantization on the matrix.

Then we apply inverse DCT function on the DCT coefficient matrix to get our original picture back (lossy/lossless), which is showed in the graphical user interface we created.

Implementation:

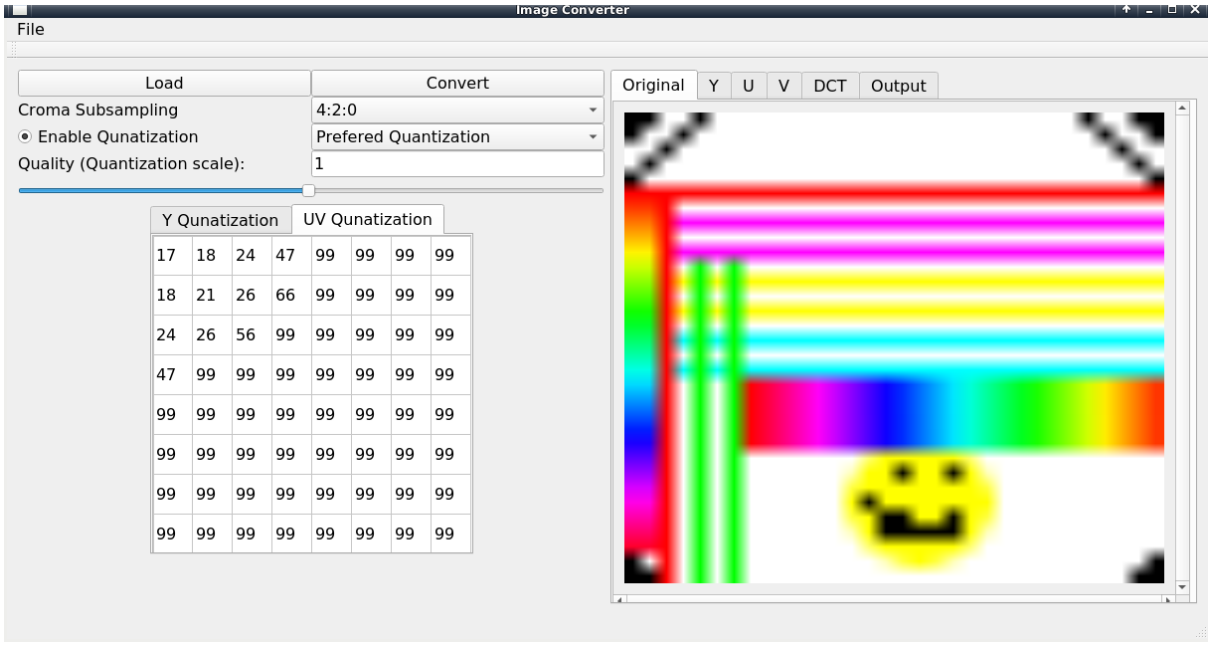
ALL required functions are implemented in "math.cpp" and the header file is called "math.h"

//Converts a 3-byte rgb pixel to yuv and returns it. The function uses the given matrix to transform RGB to YUV. We shifted the yuv channel values to be represented as a Grayscale Image.

```
cv::Vec3b rgb2yuv(cv::Vec3b pixel);
```

```
//converts a cv::Mat image from rgb to yuv by passing every pixel to the rg2yuv(cv::Vec3b) function
void rgb2yuv(cv::Mat &img);

// We have similarly implemented yuv2rgb function. We made sure that converting the RGB to YUV gives unsigned values.
cv::Vec3b yuv2rgb(cv::Vec3b pixel);
void yuv2rgb(cv::Mat &img);

// Given a,b,c values the function performs appropriate chroma sampling. In our case 4:2:0 takes 2x2 block of the image and colors all 4 pixels by the color of the top-left pixel in the 2x2 block
void chroma_subsample(cv::Mat &img, int a=4, int b=2, int c=0);

// RUN DCT Algorithm on a 8x8 matrix and return the 8x8 DCT Matrix / IDCT Matrix. This requires the a precomputed conversion matrix as seen in http://www.cs.sfu.ca/CourseCentral/365/li/material/lectureslides/5-DCT-implementation.pdf
cv::Matx<double,8,8> dct88(cv::Matx<double,8,8> f)
cv::Matx<double,8,8> idct88(cv::Matx<double,8,8> F);

//  The quant() function and the iquant() function take a 8x8 matrix and the Quantization Matrix/ Inverse Quantization Matrix and a scaling factor that is multiplied with the other Quantization Matrix to provide different quantization matrix that provides different compression quality
void quant(cv::Matx<double,8,8> block, int q[8][8], double scale );
void iquant(cv::Matx<double,8,8> block, int q[8][8], double scale );

// Extract and pass every 8*8 blocks on every channel (YUV) of the image to dct88 (cv::Mat<double,8,8>) function and passes them through the Quantization function using different quantization matrix for luminance and chromaticity provided from the UI. This function also combines the resulting Quantized DCT matrix of the whole image on all channels and returns it in a cv::Mat dct. It then decodes the quantized dct matrix using idct88 and stores the result in the original matric (img).
cv::Mat runDctOnImage(cv::Mat &img, double quality);
void runDctOnImage(cv::Mat &img,cv::Mat &dct,  double quality, bool should_quant);

//helper function that crops images to have columns and rows that are multiple of 8 when loading the image to the program
void resize8x8(cv::Mat &img);
```

Implementing the UI:



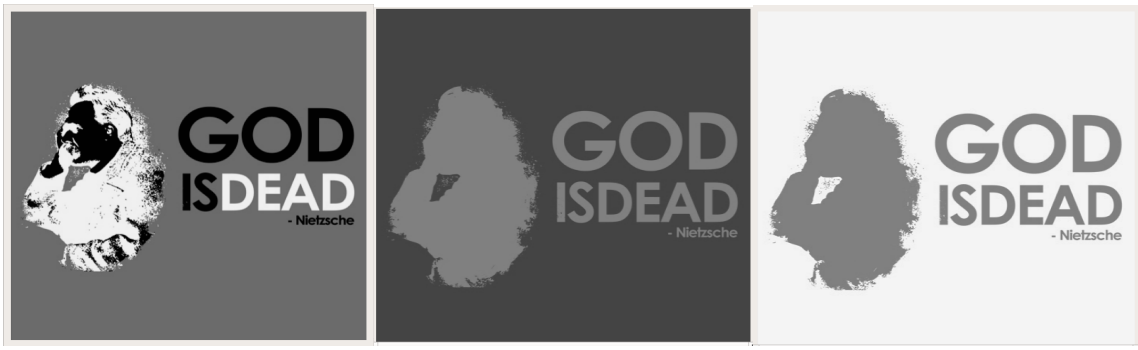
We used Qt to design a simple UI which allowed us to adjust the parameters for the encoding of the image and display images that represent various stages of the process. This helped both debugging and smoothly demonstrates the effect of different parameters. By using a tabbed window we can display all the different steps in the process. In order to see the details of images we did stretch small images so they fill the tab but if an image larger then the box is loaded we can inspect it by pixel-wise using a scroll box.

RESULTS:

Original RGB Image

RGB to YUV :

The Y, U and V channels are shown as grayscale images below using the MatGrayScale2QImg



Y channel **U channel** **V channel**

Chroma Sub-Sampling :

We created a custom 64 x 64 test image in GIMP to able to test effects of the chroma sampling of the image since it is hard to observe chroma sampling in larger images. The “smiley” and all the colors are distorted that as we decrease the “b” and “c” values of chroma sampling.

4:4:4 **4:2:2** **4:2:0**

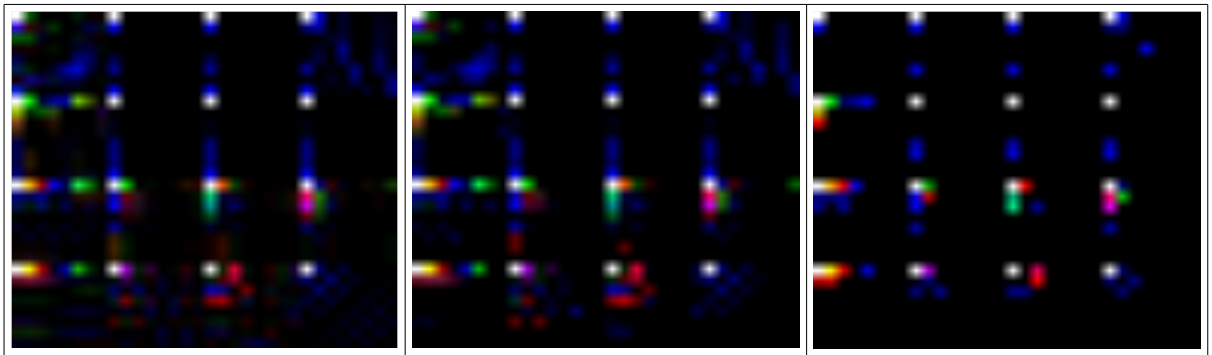
Quantization :

Original Picture

Q. Scale = 1.0 **Q.Scale = 6.2** **Q.Scale=10**

DCT View of Quantization:

Here we can see that The small values in the lower right are lost as we increase quantization parameters.



No Quantization **Preferred Quantization** **Extreme Quantization**