CMPT 365 Assignment 2 Documentation

# Overview

The purpose of this assignment is to learn the function of the essential components in a JPEG encoder/decoder. We are asked to implement the encoding/decoding process and show the results on a GUI.

## List of Features

JPEG encoding consists of mainly four parts when you exclude the lossless compression algorithms: Color conversion, downsampling, DCT transformation on pixels, and quantization. In addition to that, the program also needs to receive an image from the user, know which quantization method to use, and output both the original and the encoded version of the image.

## Graphic User Interface

For the GUI I’ve used the Qt module to implement a simple window that allows the program to receive an image provided by the user, then in turn transform it into three different ones – Y channel, JPEG encoding with normal Q tables, JPEG encoding where all channels use a particular Q table that has low values – based on user selection. The image (before processed) changes every time the user inputs a different image, and the processed one changes only when one of the three buttons is pressed.

## Programming Language, Libraries, and Platform

I’ve built this program using C++ with both the Visual Studio 2010 IDE and the Qt Creator IDE (I felt that it was easier to code the GUI on that one). Along with that, I’ve also used the OpenCV media libraries for functions such as image manipulation and matrix functions. I’ve also tried Cimg and DevIL libraries but they weren’t easy to setup and use.

(Note: OpenCV will be needed if you wish to compile the source code from scratch)

# DCT/IDCT

These two functions are included in the Majorsteps () function; IDCT after DCT. I’ve used the matrix representations of the DCT/IDCT coefficients on the website and simply created 2 functions – 2DDCT and 2DIDCT – that retrieve 8x8 blocks from a channel matrix and have the functions multiply the matrices with the predefined DCT/IDCT coefficients. In the major process DCT/IDCT will be applied to all 8x8 blocks. It will ignore excess pixels if the image does not divide into multiple 8x8 blocks evenly because practically processing those pixels will not bring a huge change to actual image. Note that the matrices are now converted into float representations for the upcoming quantization process.

# Quantization

Similar to DCT/IDCT, quantization and dequantization functions are also included in the Majorstep () function; in the order they are listed. These functions detect which DCT-processed channel of YUV they are processing and simply do matrix-element division on every single one of them. Two sets of tables are used: one set of JPEG encoding standard tables, and one set of tables with very low compression rate. After quantization, dequantization is called immediately followed by IDCT before the image recodes into its original format. As expected, having low compression rates produces a high quality image. In addition, the standard JPEG quantization table will destroy the image quality when it is applied to a small image, since the change in YUV values among pixels is extremely frequent.

Some examples of different results are provided in the zip file.

# Extra Work

Before the Majorsteps () process, I’ve split the YUV representation to separate channels for easier manipulation. In addition, they are *short* typed matrices since they need to be capable of holding negative values that reside in UV channels. A final function will be called after Majorsteps to convert the YUV channels back into one single 3-channeled matrix format. It is read for output after that has been done.

# Difficulties

During this project a few difficulties have been encountered. First of all, choosing the right tools bothered me a lot since I’ve never programmed anything media-integrated. Therefore, I have no experience regarding whether image library surpasses another, or which one is most suitable to beginners in the field. The same idea also applies to GUI implementations since this is my first time doing such. Next, I’ve encountered problems regarding image pixel sizes, particularly the question of whether the image can be divided into even 2x2 and 8x8 blocks. For downsampling, 2x2 blocks are required for taking their average and applying them to the UV channels. This was simply solved by making the blocks smaller where applicable (when 2 pixels is available but not 2x2, simply take the average of those two; when only one pixel is available (edge), skip it). However, I wasn’t able to find a way to successfully partition 8x8 blocks for DCT/IDCT and quantization. There are ways explaining that padding can be used to solve this problem but I did not think it will look good in the result. In the end I simply ignored those excessive pixels since I think encoding them will not bring a major change to the resulting image.