CSCI 3280 Introduction to Multimedia Systems

Spring 2022, Assignment 4 – Discrete Cosine Transformation

Deadline: Apr. 24, 2022 (11:59pm)

Submission via Blackboard.cuhk.edu.hk

Late submission penalty: 10% deduction per day (max: 30% deduction)

PLAGIARISM penalty: whole course failed

Introduction

As the key to the JPEG baseline compression process, the Discrete Cosine Transform (DCT) is a mathematical transformation that transforms a signal from spatial representation into frequency representation. In an image, most of the energy will be concentrated in the lower frequencies. Therefore, once an image is transformed into its frequency components, we can treat them selectively, *e.g.,* retaining lower frequency coefficients with high accuracy while squeezing the size of high frequency coefficients, so as to reduce data amount needed to describe the image without sacrificing too much image quality. In practice, a specially configured quantization table will be used to realize such selective operation. And then we can use the Inverse Discrete Cosine Transform (IDCT) to reconstruct the image when we want to see the image. In this assignment, you are required to implement a program that performs DCT on a 2D image, quantize its frequency coefficients, and IDCT on the quantized coefficients to reconstruct the image.

Implementation Guideline

1. Discrete Cosine Transformation

Given an image in the spatial domain, the pixel value at coordinates (x, y) is denoted as $f(x, y)$, which is in the range of 0~255. The goal is to transform the image $f(x, y)$ into the frequency domain $F(u, v)$, where $F(u, v)$ has the same resolution as the image $f(x, y)$. We can do the DCT as the following steps:

i. The first thing we should do is to shift the pixel value by -128 to centralize the pixel value into the range of -128~127.

ii. Then, we should cut up the image into blocks of 8×8 pixels, *e.g.,* an image of resolution 256×256 will be divided into 32×32 blocks. The DCT will be individually applied on the blocks one by one. Doing so can greatly reduce the computation complexity.

iii. To transform the image block f into the frequency domain F, the most straightforward formula is:

$$F(u,v) = \frac{1}{4} c_u c_v \sum_{x=0}^{x=7} \sum_{y=0}^{y=7} cos\frac{(2x+1)u\pi}{16} cos\frac{(2y+1)v\pi}{16} f(x,y),$$

$$where\ c_\varepsilon = \begin{cases} 1/\sqrt{2}, & \varepsilon = 0 \\ 1, & \text{otherwise} \end{cases}$$

where x, y, u, v are in the range of [0,7] since the block size is 8x8.

Efficient Discrete Cosine Transformation. However, the formula above uses four nested loops and has complexity $O(n^4)$. To implement a more efficient version, we can equivalently build the 2D DCT by running a 1D DCT over every row and then every column. Specifically, for each block f with 8×8 pixels, the 2D DCT is performed through the following two 1D DCT steps:

a) Apply 1D DCT to a row v_i of f : $F^r(u, v_i) = \frac{1}{2} c_u \sum_{x=0}^{7} cos\left(\frac{(2x+1)u\pi}{16}\right) f(x,y), \forall u \in [0,7]$, and then repeat it over every row of f.

b) Apply 1D DCT to column u_i of F^r: $F(u_i, v) = \frac{1}{2} c_v \sum_{y=0}^{7} cos\left(\frac{(2y+1)v\pi}{16}\right) F^r(u,y), \forall v \in [0,7]$, and repeat it over every column of F^r.

This efficient version of 2D DCT can yield a $O(n^3)$ algorithm.

iv. After the above steps, the 8 × 8 frequency coefficient array $F(u,v)$ for a block is obtained. By computing the frequency coefficients for every block, we can get the final DCT result, i.e., a N × M coefficient array.

2. **Coefficient Quantization**

The frequency coefficients generated by DCT are float numbers and many of them have very tiny values. To convert these coefficients into integers while keep less information loss, **you are required to quantize the frequency coefficients of every blocks by the provided quantization matrix**. Specifically, for every element in the coefficient array $F(u,v)$, the quantized element should be $F^Q(u,v) = \left[\!\left[\frac{F(u,v)}{Q(u,v)}\right]\!\right]$, where $[\![\cdot]\!]$ denotes rounding a float number to the nearest integer, and the 8×8 quantization matrix $\mathbf{Q}$ is defined as:

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

Finally, a $N \times M$ quantized coefficient array F^Q will be achieved.

3. **Inverse Discrete Cosine Transformation**

If we want to reconstruct the image from the quantized coefficient array F^Q, we should do the inverse process of the above algorithms. They can be divided into following steps:

i. Multiply the quantization table to reproduce the “de-quantized” coefficient array:

$$\hat{F}(u,v) = F^Q(u,v) * Q(u,v)$$

ii. *Inverse Discrete Cosine Transformation (IDCT).* After we obtain the “de-quantized” coefficient array $\hat{F}$, we can do the IDCT on it. The IDCT is the inverse function of DCT, which can be formulated as:

$$\hat{f}(x,y) = \frac{1}{4}\sum_{u=0}^{u=7}\sum_{v=0}^{v=7} c_u c_v \cos\frac{(2x+1)u\pi}{16}\cos\frac{(2y+1)v\pi}{16}\hat{F}(u,v),$$

$$where\ c_\varepsilon = \begin{cases} 1/\sqrt{2}, & \varepsilon = 0 \\ 1, & \text{otherwise} \end{cases}$$

We can easily see that this formulation is also a $O(n^4)$ algorithm. The basic idea for the efficient version of this algorithm is similar with the efficient version 2D DCT. You are encouraged to derive it by yourself.

iii. Then you can shift back the pixel values by adding 128 and combine all the blocks to obtain the final reconstructed image.

Basic Requirements: No quantization (60 points)

1. You are required to implement the **Discrete Cosine Transform (DCT)** as described in Section 1 (***no need*** to implement the efficient version) and the **Coefficient Quantization** step as described in Section 2. You should apply your implemented program to the provided testing image.
2. To simply the assignment, you can assume the resolution of the input image is fixed to be 256x256.
3. Program must be coded in ANSI C/C++ and uses standard libraries only.
4. The compiled program must run in Windows 10 command prompt as a console program and accepts source bitmap (.bmp format) with the following syntax and save generated images to the current directory.

   ```
   C:\> dct.exe <img_path> <apply_idct>
   ```

   dct.exe is the executable file of your program, *e.g.,* the command: **`dct.exe lighthouse.bmp 0`** is to transform the input image from spatial domain into frequency domain and output the quantized frequency coefficient array. **<img_path>** is the path where the input image is located.

   <apply_idct> specifies whether applying the IDCT to the quantized coefficient array to reconstruct the image, where 1 means TRUE while 0 means FALSE. Note that basic requirement does NOT require the implementation of the IDCT part.
5. You are required to submit source code only. We will use Visual Studio 2015 (or newer) C++ compiler and have your program compiled via Visual Studio command prompt (Tools Command Prompt) with the command line: **`C:\> cl dct.cpp bmp.cpp`** (**Please ensure your source code gets compiled well with it**).

Enhanced Features (40 points)

You DO NOT need to implement all the following features. You may choose some of them that you are interested in to implement. The grading of this part will be based on the overall situation of the whole class submissions.

- Implement the IDCT as described in Section 3.
- Implement the efficient version of DCT as described in Section 1.
- Implement the efficient version of IDCT.

- Other interesting applications based on DCT.

Note: If you have finished the efficient version of DCT/IDCT for the enhancement part, you do NOT need to implement/submit the non-efficient version.

Submission

We expect the following files zipped into a file named by your SID (e.g. s1155xxxxxx.zip) and have it uploaded to the Blackboard by due date: **Apr. 24, 2022 (11:59pm)**

- **README.txt** (Tell us the enhanced features you implemented and anything else that we should pay attention to.)
- **bmp.h & bmp.cpp** *(No need to change)*
- **dct.cpp** (write your code in this file only)