

# Assignment 1: Transform Coding

CS 4600 Computer Graphics

Fall 2018

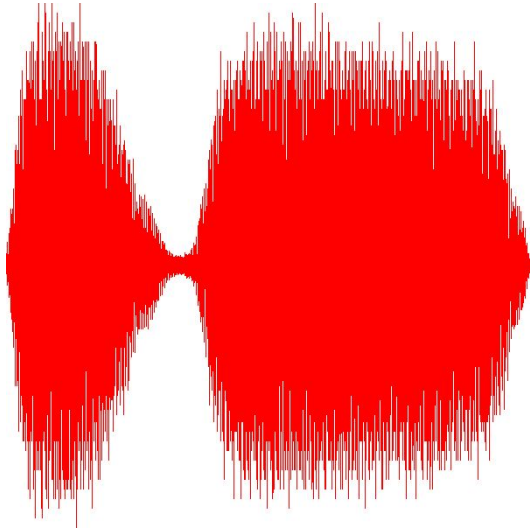
Ladislav Kavan

In this assignment we will experiment with audio and image compression. Even though this class is primarily about computer graphics, it is very useful to understand audio compression first before diving into image compression. This is an individual assignment, i.e., you have to work independently. All information needed to complete this homework is covered in the lectures and discussed at our Canvas Discussion Boards. You shouldn't have to use any textbooks or online resources, but if you choose to do so, you must reference these resources in your final submission. It is strictly prohibited to reuse code or fragments of code from textbooks, online resources or other students in this course. This is considered as academic misconduct (<https://www.cs.utah.edu/academic-misconduct/>). Do not share your homework solution with anyone -- this is also treated as academic misconduct in this course, even if nobody ends up copying your code.

The framework code is written in C++ with the following dependencies:

- OpenGL 1.0
- GL Utilities (GLU)
- C++ STL
- OpenGL Extension Wrangler Library ([GLEW](#))
- [GLFW3](#)

The recommended IDE is Visual Studio 2017 Community Edition, which is available free of charge for educational purposes. The framework code provides precompiled dependencies for Visual Studio 2017. If you choose to use a different platform or IDE version it is your responsibility to build the dependencies and get the project to work.



Audio Coding



Image Coding

The assignment consists of two parts: audio coding and image coding, which are under two separate folders: “AudioCoding” and “ImageCoding”. Both parts should be implemented in the corresponding *main.cpp* file using the specified subroutines. No other source code / dependencies / libraries are needed or allowed for this assignment. The provided source code, after being successfully compiled, linked, and executed, should display the images shown above.

## Introduction

The Discrete Cosine Transform is a variant of the Fourier Transform where a signal vector is expressed in terms of an orthonormal basis made up of sinusoids. The DCT is widely used in image compression. In this project you will implement the DCT and use it to compress both 1-dimensional and 2-dimensional signals, i.e. audio clips and images.

### 1 Audio Coding (50 points)

The whole secret to transform coding is to take your original signal, which you can think of as a vector in  $R^N$ , and express it in terms of an orthonormal basis where most of the coefficients are close to zero. The 1D Discrete Cosine Transform is used here to compute these intermediate coefficients. The formula for 1D DCT is as follows:

$$F(u) = c(u) \sum_{i=0}^{N-1} \cos\left(\frac{(2i+1)u\pi}{2N}\right) f(i), \quad (1)$$

Where  $i = 0, 1, \dots, N-1$ ,  $u = 0, 1, \dots, N-1$ ,  $f(i)$  is the original signal, the constant  $c(u)$  is  $\frac{\sqrt{2}}{4}$  if  $u = 0$  and  $\frac{1}{2}$  otherwise.  $F(u)$  is the resulting DCT. To perform decompression, we can use the 1D inverse Discrete Cosine Transform:

$$f(i) = \sum_{u=0}^{N-1} c(u) \cos\left(\frac{(2i+1)u\pi}{2N}\right) F(u) \quad (2)$$

Where  $i = 0, 1, \dots, N-1$ ,  $u = 0, 1, \dots, N-1$ ,  $F(u)$  are DCT coefficients, the constant  $c(u)$  is  $\frac{\sqrt{2}}{4}$  if  $u = 0$  and  $\frac{1}{2}$  otherwise, and  $f(i)$  is the original signal.

Your first task is to complete the function  $DCT(A, C, N)$  that takes a vector  $A \in R^N$  from the input signal and computes its DCT coefficients  $C \in R^N$  using Equation 1.

Your next job is to do the actual compression by zeroing out some less important coefficients computed by the function  $DCT$ . You need to write the code for function  $compress(C, N, m)$ , which zeros out last  $m$  elements of  $C$ . Without this function, the final result after decompression should be exactly like the original audio signal.

Finally, you need to code up function  $inverseDCT(C, B, N)$  that takes a vector  $C \in R^N$  from the modified (compressed) coefficients and computes inverse Discrete Cosine Transform (Equation 2) yielding vector  $B \in R^N$ , which is the decompressed version of the original signal (not exactly the same because we are considering lossy compression).

After you have written your  $DCT$  and  $inverseDCT$  functions you will use them to compress some audio signals. There are three sounds provided in “data” directory, gong.wav, handel.wav, and train.wav. Which input file will be picked is controlled by the “WAV\_FILE” define. Pressing ‘1’ will show the original wav signal and pressing ‘2’ will show the decompressed wav signal. You can also find your compressed audio under “data” named out.wav -- try playing it back and listening to it! Once you compute the Discrete Cosine Transform of your input signals you can output the resulting coefficients and/or plot them in a spreadsheet (such as Excel, Google Docs, or Matplotlib).

Your task is compressing the audio signal by computing its DCT, keeping only the first few leading coefficients and discarding the rest. The decompression then works by adding zeros instead of the discarded (i.e., non-transmitted) coefficients and computing the Inverse DCT. You can experiment with different audio files and different levels of compression, i.e. different numbers of discarded coefficients. Your task is to compress and decompress the sound “train.wav” by discarding the last 5 coefficients per block (each block has 8 coefficients, so you will keep coefficients number 0, 1, 2 and discard 3, 4, 5, 6, 7). The decompressed sound will be slightly distorted. Save the decompressed file as “out.wav” and submit it along with your solution.

## 2 Image Coding (50 points)

Just as the DCT can be used to compress 1D signals, it can also be used to compress 2D signals like images, in fact it's the basis for the JPEG compression standard. JPEG begins by splitting each image into  $8 \times 8$  blocks. In this assignment, the provided input image has dimensions divisible by 8. The input images are grayscale, so you also don't have to worry about color formats; there is only one intensity value per pixel.

Similar to the 1D case, we can compute intermediate coefficients using Discrete Cosine Transform formula but for 2D in case of image compression. In this case, the coefficients are  $8 \times 8$  matrices computed for each  $8 \times 8$  block. The formula for 2D DCT is as follows:

$$F(u, v) = c(u) c(v) \sum_{x,y=0}^{N-1} \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right) f(x, y) \quad (3)$$

Where  $x = 0, 1, \dots, N-1$ ,  $y = 0, 1, \dots, N-1$ ,  $u = 0, 1, \dots, N-1$ ,  $v = 0, 1, \dots, N-1$ ,  $f(x, y)$  are the original values for the block of the input image, and the constant  $c(u)$  is  $\frac{\sqrt{2}}{4}$  if  $u = 0$  and  $\frac{1}{2}$  otherwise. To do the decompression, we can use the 2D inverse Discrete Cosine Transform:

$$f(x, y) = \sum_{u,v=0}^{N-1} c(u) c(v) \cos\left(\frac{(2x+1)u\pi}{2N}\right) \cos\left(\frac{(2y+1)v\pi}{2N}\right) F(u, v) \quad (4)$$

Where  $x = 0, 1, \dots, N-1$ ,  $y = 0, 1, \dots, N-1$ ,  $u = 0, 1, \dots, N-1$ ,  $v = 0, 1, \dots, N-1$ ,  $F(u, v)$  are coefficients computed using 2D DCT, and the constant  $c(u)$  is  $\frac{\sqrt{2}}{4}$  if  $u = 0$  and  $\frac{1}{2}$  otherwise.

The code for splitting the input image into  $8 \times 8$  blocks is provided for you in function *processImage*. You need to complete similar functions as you did in the 1D case to process a block of input image which includes computing 2D DCT, zeroing out some coefficients and finally decompressing the block using 2D inverse DCT.

Specifically, your first task is to complete function *DCT(A, C, N)* which takes an N by N block from the original image and computes a intermediate matrix of N by N DCT coefficients C using Equation 3 (the input matrix A and the output matrix C are stored as 1D arrays in the provided source code).

Your next task is to do the actual compression by zeroing out some less important coefficients computed by function *DCT*. You need to write the code for function *compress(C, N, m)*, which zeroes out all of the DCT coefficients in C with  $i+j > m$  (where both indices  $i$  and  $j$  range from  $0, 1, \dots, N-1$  according to the C/C++ conventions). The C matrix (for “Coefficients”) is a temporary matrix. This zeroing is where the compression happens, because we only save (e.g.,

to disk) the nonzero coefficients. For the purpose of this assignment, however, we won't be saving a compressed file and instead we will study how distorted the image will be after decompression.

The decompression is done by computing the inverse DCT on the  $C$  matrix using function *inverseDCT*( $C, B, N$ ). You can use Equation 4 to compute the decompressed block of image. The result will be an  $N$  by  $N$  block of pixels which you return in  $B$ . If  $m = 0$ , there is no compression and  $B$  should contain the same numbers as  $A$ . When you try to increase the compression level (i.e.,  $m$ , which corresponds to zeroing out more coefficients), the  $B$  should be similar to  $A$ , but not exactly the same.

Three test images, "cameraman.ppm", "mandi.ppm", and "moon.ppm", have their width and height divisible by 8. The ".ppm" extension denotes the portable pixmap format image. The input image is passed using 1D array  $I$  (for Input), which contains intensity values in floats (from 0 to 1) for all pixels in the input image, scanning the image lines (scanlines) from top to bottom, left to right (which is the usual convention in computer graphics). The variables *g\_image\_height* and *g\_image\_width* are global variables which store the image size.

Which input file will be picked is controlled by the "IMAGE\_FILE" define. Pressing '1' will show the original image and pressing '2' will show your decompressed image. You can also find your decompressed image as "out.ppm" under "data/". Your task is to submit an "out.ppm" file (decompressed version of "cameraman.ppm") with compression level  $m = 3$ .

### 3 Extra Credit (up to 20 bonus points at instructor's discretion)

Wavelets are another popular approach to constructing an orthonormal basis which also has very good properties for signal compression. Try repeating the audio and image coding experiments using the Haar wavelet basis instead of the DCT basis. You can find more information about the Haar wavelet online, e.g. at Wikipedia. You can also experiment with larger blocks, e.g., 16 x 16 or 32 x 32 and see if you can achieve better compression results.

## 4 Submission

When you're finished with Tasks 1 and 2, you'll need to submit the following:

- Source code (you should only modify the two main.cpp files and name them as main-audio.cpp and main-image.cpp). These two CPP files are all we need. Please do not submit any other files, especially NOT .exe files and other files created by Visual Studio.

- PDF document describing what you did, screenshots / graphs are recommended. If you used any textbooks or online resources that may have inspired your way of thinking about the assignment, you must reference these resources in this document.
- Your decompressed results “out.wav” (decompressed “train.wav”) and “out.ppm” (decompressed “cameraman.ppm”).

If you are solving the optional Task 3, make sure to still submit your solution of Tasks 1 and 2 according to the instructions above without any improvements or extensions! Please submit your solution of Task 3 in separate files, such as “main-image-extra.cpp” and “HW1-extra.pdf”.

Please pack all of your files (including the optional extra credit files) to a single ZIP file named `LastName_Firstname_HW1.zip`

Please submit this ZIP file via Canvas.