Assignment 1: Transform Coding

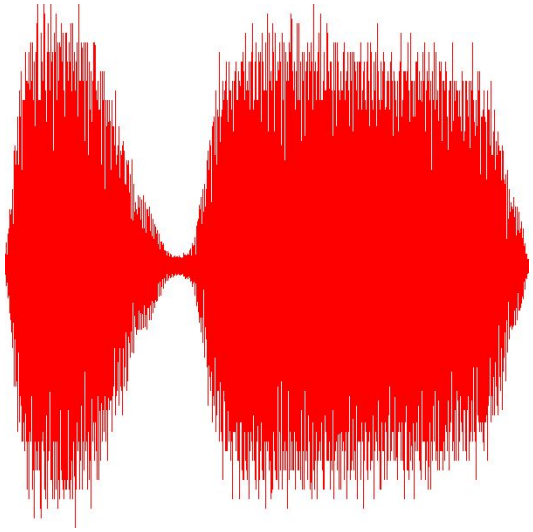
CS 4600 Computer Graphics
Fall 2019

In this assignment we will experiment with audio and image compression. Even though this course 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:

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

Where $i = 0, 1, ..., N-1$, $u = 0, 1, ..., N-1$, $f(i)$ is the original signal, the constant $c(u)$ is a normalizing constant to ensure the transformation is orthonormal (as opposed to just

orthogonal). Your task is to figure out the value of $c(u)$ and explain how you arrived to your answer in your writeup (see submission instructions below). $F(u)$ are the resulting DCT coefficients. To perform decompression, we use the 1D inverse Discrete Cosine Transform:

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

Where $i = 0, 1, ..., N-1,\ u = 0, 1, ..., N-1$, $F(u)$ are DCT coefficients, the constant $c(u)$ is the same as before. If you have determined $c(u)$ correctly, then eq. (2) returns the exact f(i) we entered into eq. (1) to begin with. Please explain in your writeup what would happen if $c(u)$ was not correct.

Your first programming 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 elements of C with index i >= *m*. Without this function (or with m = 8), the final result after decompression would be exactly like the original input signal (up to floating point rounding errors).

Finally, you need to code up function *inverseDCT(C, B, N)* that takes a vector $C \in R^N$ from the modified (zeroed-out) 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 example 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 Sheets, or Matplotlib). Please include the graphs in your writeup.

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 deliverable is to compress and decompress the sound “train.wav” by setting m = 4, which corresponds to discarding the last 4 coefficients per block

(each block has 8 coefficients (N = 8 in the formulas above), so you will keep coefficients number 0, 1, 2, 3 and discard 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 8×8 blocks. In this assignment, the provided input image has dimensions divisible by 8, to make things easier. The input images are grayscale, so you don’t have to worry about color formats; there is only one intensity value per pixel.

Similar to the 1D case, we can compute coefficients of the Discrete Cosine Transform but this time in 2D. In the 2D case, the coefficients are 8×8 matrices computed for each 8×8 block. The formula for general 2D DCT is as follows (in our case, N = 8):

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

Where $x = 0, 1, ..., N-1,\ y = 0, 1, ..., N-1,\ u = 0, 1, ..., N-1,\ v = 0, 1, ..., N-1$, $f(x,y)$ are the original values for the block of the input image, and the constant $c(u)$ is the same as before. The inverse transformation is:

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

Where $x = 0, 1, ..., N-1,\ y = 0, 1, ..., N-1,\ u = 0, 1, ..., N-1,\ v = 0, 1, ..., N-1$, $F(u,v)$ are coefficients computed using 2D DCT, and the constant $c(u)$ is the same as before.

The code for splitting the input image into 8×8 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 an 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 $0, 1, ..., N-1$ according to the C/C++ conventions). The C matrix (for "**C**oefficients") 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 directly 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 $m = 15$, there is no compression and B should contain the same numbers as A (up to floating point rounding errors). When you try to increase the compression level (which corresponds to decreasing m, i.e., 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 $m = 5$.

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

Note: this is an optional part of the assignment and different expectations apply to it. In this optional part, external resources are almost necessary to use (but you must still reference them, of course). The bar for receiving the bonus points is very high, getting 5 (out of the 20) is already an excellent outcome; full 20 bonus points corresponds roughly to a final-project-level effort. Below are some specific ideas on how to extend this assignment to the next level.

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., 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.

Another interesting extension is to implement a pair of compress-decompress functions that perform data compression and decompression, i.e., actually reducing the data size (as opposed

to just zeroing out DCT coefficients as discussed above). This will involve quantization and de-quantization, you may refer to the JPEG standard for inspiration.

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 or other files generated by Visual Studio.
- PDF document (writeup) 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 your writeup.
- Your decompressed results "out.wav" (decompressed "train.wav") and "out.ppm" (decompressed "cameraman.ppm").

Please pack all of your files (including the optional extra credit files) to a single ZIP file named Lastname_Firstname_HW1.zip

If you are solving the optional Task 3, make sure to still follow the instructions above! To submit your solution of Task 3, please submit a separate ZIP file named Lastname_Firstname_HW1_extra.zip

Please submit the ZIP file(s) via Canvas and in the comments specify how many late days you're using and have remaining (e.g. "Used 0 late days, 5 remaining.") Failure to comply with submission instructions may result in loss of points (up to zero in cases of severe negligence).

Late policy

Everyone has 5 late days to cover circumstances such as minor illness, competitions, family matters etc. You can distribute your late days arbitrarily among assignments (e.g. you can use 0 late days on HW assignment 1 and all 5 late days on HW assignment 2, leaving no late days left for subsequent assignments). After using up all of your late days, a 0.1*X*number_of_late_days will be deducted from your points, where X is the pre-penalty number of points. Each late day is defined as 24 hours after the submission deadline on Canvas, there is no distinction between weekdays and weekends or holidays. Requests for exceptions to this policy (e.g., due to major medical conditions) must be made in writing, with a date and a signature of the student or their legal representative, and include documentation pertaining to the case (e.g., a note from the University Center for Disability & Access).