Assignment 2: Transform Coding

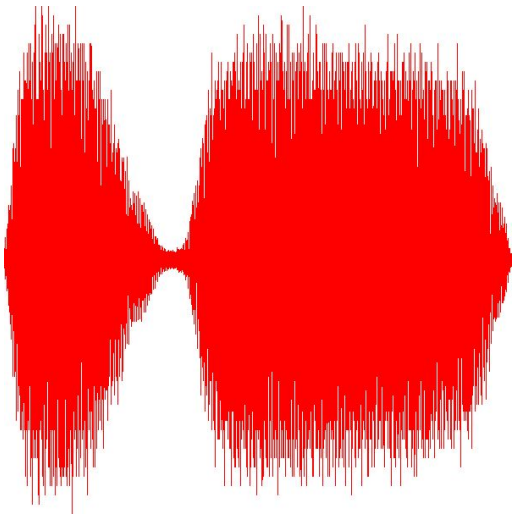
CS 4600 Computer Graphics
Fall 2016
Ladislav Kavan

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 2013 Community Edition, which is available free of charge for educational purposes. The framework code provides precompiled dependencies for Visual Studio 2013. 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: “Audio-Coding” and “Image-Coding”. 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^8, and express it in terms of an orthonormal basis where most of the coefficients are close to zero. The Discrete Cosine Transform begins with a collection of 8 orthonormal vectors $\{q_1, q_2, ..., q_8\}$ that span R^8. These vectors can be constructed from the following equation which gives us the i-th element of vector q_k as follows:

$$q_k{}^i = s_k cos\left(\left(\frac{\pi}{16}\right)(k-1)(2i-1)\right) \quad (1)$$

where

$$q_k = \left(q_k{}^1, q_k{}^2, ..., q_k{}^8\right)^T$$

The scalar s_k is just the normalization factor which is required to make sure that each of the vectors q_k has unit norm. In these equations the parameter $k \in \{1...8\}$ acts as a frequency index, higher values of k correspond to sinusoids with more cycles.

Your first task is to complete the following function *DCTvector(*$8, k, q$*)* that computes the elements of vectors $q_k \in R^8$. Note that this function takes k as an argument and produces vector $q \in R^8$. Make sure to normalize your output vectors to unit length (which corresponds to finding the right value of s_k).

You should test your function by generating an 8 by 8 matrix Q whose columns correspond to the 8 DCT vectors in R^8. You should print out these vectors to see what they look like. You should also verify that the resulting matrix Q is orthogonal by checking that $Q^T Q = I_8$ (i.e. the 8×8 identity matrix).

Your next job is to write two functions, the first of which computes the Discrete Cosine Transform and the second of which computes the Inverse Discrete Cosine Transform.

Function *DCT(*$x, y, q, 8$*)* takes a vector $y \in R^8$ and produces as output a vector $x \in R^8$ where the k-th element of the vector is computed as $x_k = q_k \cdot y$, where the dot symbol denotes the dot product.

Function *InverseDCT(*$y, x, q, 8$*)* takes a vector $x \in R^8$ and produces as output a vector $y \in R^8$ where $y = \sum_{k=1}^{8} x_k q_k$.

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 Matlab).

Your last 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 x 8 blocks. In this assignment, the provided input image has dimensions divisible by 8 so you don’t have to worry about the corner cases. The input images are grayscale, so you also don’t have to worry about color formats -- there is only one intensity value per pixel. We can form an orthonormal basis for the set of 8 by 8 images by taking outer products of the 1D DCT vectors. That is if $\{q_1, q_2, ..., q_b\} \in R^8$ denote the 8 DCT vectors that span R^8 then we can construct 64 matrices of size 8 by 8 by taking all possible outer products $q_i q_j^T$. You may want to use the function *outerProduct* which we have prepared for your convenience. These 64 matrices happen to form an orthonormal basis for the set of all 8 by 8 matrices. You should verify this for yourself.

Your task is to write a function *CompressBlock(*A*,* B*,* m*)* which takes an 8 by 8 image block, A, transforms it into an intermediate matrix of 8 by 8 DCT coefficients C, and zeroes out all of the DCT coefficients with i + j > m (where both indices i and j range from 0, 1, ..., 7 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 to disk the non-zero coefficients. For the purpose of this task, however, we won’t be saving a compressed file and instead take a look at how distorted the image will be after decompression. The decompression should also happen in the function *CompressBlock* and it is done by computing the Inverse DCT on the C matrix. The result will be an 8 by 8 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 will be close to A, but not exactly the same.

Your next job is to use the function *CompressBlock* to write function *CompressImage(*I*,* O*,* m*)*, which compresses an input image, I, into an output image, O, by compressing every 8 by 8 block using the *CompressBlock* function with parameter m. 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 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 for R^N 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 references 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 “HW2-extra.pdf”.

Please pack all of your files (including the optional extra credit files) to a single ZIP file named Lastname_Firstname_HW2.zip

Please submit this ZIP file via Canvas.