CS 4600 HW2: Transform Coding

Cameron Jackson
u0927597

Part 1: Audio

The Audio signal compression was much easier to understand intuitively. First I just implemented the DCTvector function by iterating an int i from 0 to 7 and calculating the values for a given DCT vector based on the formula given to us. I did not worry about the s_k coefficient out front because the normalize function handles that.

```
void DCTvector(int N, int k, float* q)
{
    // TODO: part of Homework Task 1
    // generate vector q, which is defined in equation (1)
    for (int i = 0; i < N; i++)
    {
        q[i] = cos((M_PI / 16) * (k) * (2 * i + 1));
    }

}
```

The DCT method just walks through every vector in the DCT matrix and calculates its dot product with y, the value of which is placed in x. I realize now that I could have just called the dot product function instead of doing it manually, but the result is the same.

```
void DCT(float* x, const float* y, const float* q, int size)
{
    // TODO: part of Homework Task 1
    // takes a vector y and produce as output a vector x,  where

    for (int k = 0; k < size; k++)
    {
        float x_k = 0;
        for (int i = 0; i < size; i++)
        {
            x_k += q[8 * k + i] * y[i];
        }
        x[k] = x_k;
    }
}
```

For the quantization, I just zeroed out every value with an index greater than the compression level.

```
// TODO: set last m element as zero
for (int j = 0; j < 8; j++)
{
    if (j > takeData - 1)
        x[j] = 0;
}
```

The InverseDCT function is fairly similar, except for every pixel in y, I summed the products of the corresponding values in every dct vector and the that vector's coefficient in x.

```
void InverseDCT(float* y, const float* x, const float* q, int size)
{
    // TODO: part of Homework Task 1
    // takes a vector x and produce as output a vector y where y = sum x_k * q_k

    for (int i = 0; i < size; i++)
    {
        float y_i = 0;
        for (int k = 0; k < size; k++)
        {
            y_i += q[8 * k + i] * x[k];
        }
        y[i] = y_i;
    }
}
```

Part 2: Image

This video was very useful in helping me intuitively understand what the image DCT is actually doing:
https://www.youtube.com/watch?v=Q2aEzeMDHMA

For the CompressBlock function, the order of operation is as follows (note: an screencap of the function would be too large to be useful in this PDF, so it would be easier to just look at the code):

1. Shift all values in the block by -0.5, so they are zero-based (ZeroBased() function)
2. Compute DCT coefficients by finding the double contraction of the each DCT matrix and the data matrix. (note: I copied the DCTvector function from the previous part. I also made a new function called DCTMatrix which obtains a certain DCT matrix by finding the outer product of the corresponding DCT vectors.)
3. Zero out all values whose index is greater than our compression level (Quantize() function)

4. Zero out all the values in the output, so the sum doesn't accumulate between blocks. (ClearOut())
5. Undoes the DCT by going through every pixel and summing value of that pixel in a given DCT matrix times that matrix's calculated coefficient.
6. Shift all values by +0.5 to undo the zero base. (UndoZeroBase())

CompressImage() walks through every block and:

1. Copies the corresponding data to an array. (TakeInBlock())
2. Compresses the block. (CompressBlock())
3. And copies the data from that block to the correct places in the output vector. (PutOutBlock())

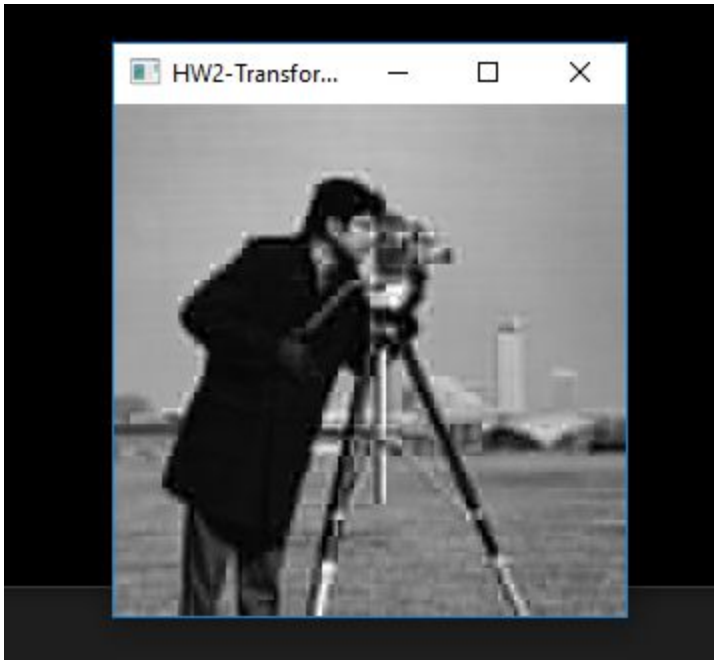
```
void CompressImage(const std::vector<float> I, std::vector<float>& O, int m)
{
    // TODO: Homework Task 2 (see the PDF description)
    float In[64];
    float Out[64];

    for (int blockx = 0; blockx < g_image_width / 8; blockx++)
    {
        for (int blocky = 0; blocky < g_image_height / 8; blocky++)
        {
            TakeInBlock(I, In, blocky, blockx);
            CompressBlock(In, Out, m);
            PutOutBlock(O, Out, blocky, blockx);
        }
    }

}
```

Hear are some of my resulting outputs:
n=1

n=2



n=3

n=4

n=8

n=16

And here is a fun output that I got back when I fundamentally misunderstood how the inverse DCT works: