Lossy Image Compression

- DCT and Scalar Quantization

Marcus Windmark, Dec 25 -14
Data Compression, NTNU

Outline of the Project

- Lossy image compression
- 8 bit grayscale images

Choice of Algorithms

- Adaptive Huffman Coding
- Discrete Cosine Transform
- Scalar Quantization

General Encoding Algorithm

1. Convert image into N * N tiles

2. Perform forward DCT on all tiles

3. Scalar Quantize all tiles

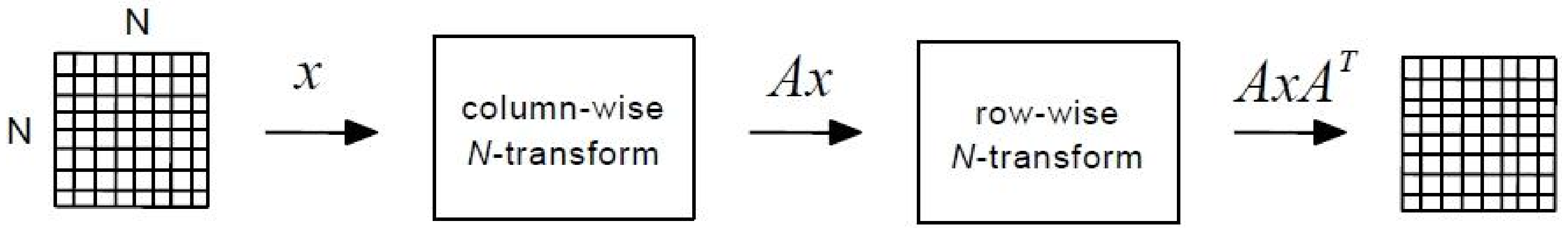
4. Encode using Adaptive Huffman

Implementation - DCT(1)

$$y(k) = \sqrt{\frac{2}{N}}\,\alpha(k)\sum_{n=0}^{N-1} x(n)\cos\frac{(2n+1)k\pi}{2N}; \quad k = 0,1,\ldots N-1$$

$$x(n) = \sqrt{\frac{2}{N}}\sum_{k=0}^{N-1} \alpha(k)y(k)\cos\frac{(2n+1)k\pi}{2N}; \quad n = 0,1,\ldots N-1$$

$$\alpha(0) = \frac{1}{\sqrt{2}} \quad and \quad \alpha(k) = 1;\ k \neq 0$$



Implementation - DCT(2)

```java
private double[] forwardDCT1D(double[] valueArray) {
    double[] outArray = new double[tileSize];

    for (int k = 0; k < tileSize; k++) {
        double sum = 0.0;

        for (int n = 0; n < tileSize; n++) {
            double cos = cosTable[k][n];
            double product = valueArray[n] * cos;
            sum += product;
        }

        double alpha;
        if (k == 0) {
            alpha = oneDivSqrtTwo;
        } else {
            alpha = 1;
        }

        outArray[k] = sum * alpha * sqrtTwoDivTileSize;
    }
    return outArray;
}
```

Implementation
- Scalar Quantization(1)

- Sample JPEG Quantization Table (show)

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}.$$

Implementation
- Scalar Quantization(2)

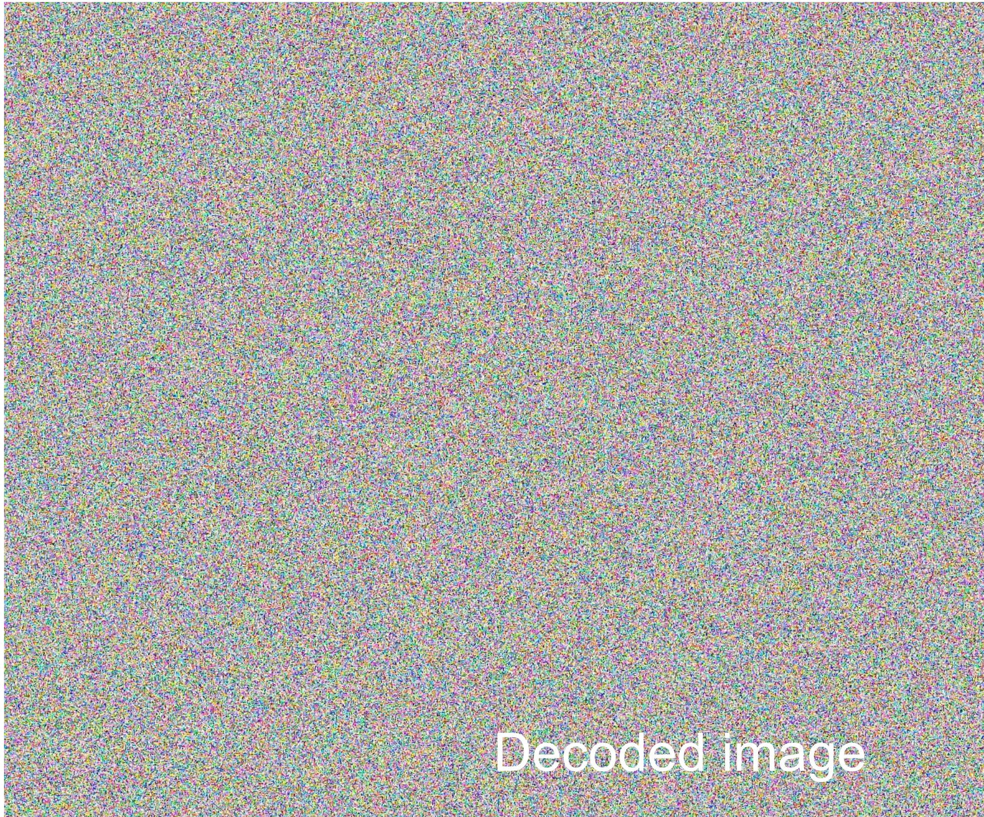
- Sample JPEG Quantization Table (show)
- Zig-zag scanning
- Encoding removes trailing zeroes

Obstacles and Solutions

- Negative values with Huffman coding
  - Solution: Extend bit size from 0-256 to 0-513 and convert

- Too large value interval by Inverse DCT
  - Solution: Normalize to 0 and 255 respectively

Result - Worst Case



Result - Best Case

Result - Measurements

Image	Raw Size	Lossy Size	Compression Ratio	Space Saving	SNR	Encoding time	Decoding time
1	1920000	383654	5.00	80.02%	26.84	836	825
2	1920000	266754	7.20	86.11%	17.73	848	780
3	1920000	317213	6.05	83.48%	27.88	785	777
4	1920000	310056	6.19	83.85%	26.27	807	773
5	1920000	867319	2.21	54.83%	18.00	973	859

Result - Compared to Lossless

Image	Raw Size	Lossy Size	CR	Space Saving	Lossless Size	CR	Space Saving
1	1920000	383654	5.00	80.02%	1802687	1.07	6.11%
2	1920000	266754	7.20	86.11%	1272218	1.51	33.74%
3	1920000	317213	6.05	83.48%	1566242	1.23	18.42%
4	1920000	310056	6.19	83.85%	1488520	1.29	22.47%
5	1920000	867319	2.21	54.83%	1921055	1.00	-0.05%

Result - Compared to JPEG

Image	Raw Size	Lossy Size	CR	Space Saving	JPEG Size	CR	Space Saving
1	1920000	383654	5.00	80.02%	265891	7.22	86.15%
2	1920000	266754	7.20	86.11%	328111	5.85	82.91%
3	1920000	317213	6.05	83.48%	318491	6.03	83.41%
4	1920000	310056	6.19	83.85%	241304	7.96	87.43%
5	1920000	867319	2.21	54.83%	1715439	1.12	10.65%

Questions?