

# JPEG2000

Patrick J. Van Fleet

Center for Applied Mathematics  
University of St. Thomas  
St. Paul, MN USA

PREP - Wavelet Workshop, 2007



UNIVERSITY of ST. THOMAS

# OUTLINE

## TODAY'S SCHEDULE

### BASIC JPEG

Naive Algorithm

An Example



# TODAY'S SCHEDULE

9:00-10:15 **Lecture Nine:** The Lifting Method

10:15-10:30 **Coffee Break** (OSS 235)

10:30-11:45 ⇒ **Computer Session Five:** Computer Session:  
JPEG2000

11:45-12:00 Evaluations/Wrap Up

12:00-1:00 **Lunch** (Cafeteria)



- ▶ Given an  $N \times N$  image  $A$  ( $A$  square and  $N$  is divisible by 8 for simplicity):
  - ▶ **Subtract** 127 from each element in  $A$ .
  - ▶ **Partition**  $A$  into  $8 \times 8$  blocks.
  - ▶ **Transform** each block using the *Discrete Cosine Transform (DCT)*.
  - ▶ **Quantize** the elements in each block.
  - ▶ **Encode** using Huffman encoding.
- ▶ Note that the quantization step makes JPEG an example of lossy compression.



- ▶ Given an  $N \times N$  image  $A$  ( $A$  square and  $N$  is divisible by 8 for simplicity):
  - ▶ **Subtract** 127 from each element in  $A$ .
  - ▶ **Partition**  $A$  into  $8 \times 8$  blocks.
  - ▶ **Transform** each block using the *Discrete Cosine Transform (DCT)*.
  - ▶ **Quantize** the elements in each block.
  - ▶ **Encode** using Huffman encoding.
- ▶ Note that the quantization step makes JPEG an example of lossy compression.



- ▶ Given an  $N \times N$  image  $A$  ( $A$  square and  $N$  is divisible by 8 for simplicity):
  - ▶ **Subtract** 127 from each element in  $A$ .
  - ▶ **Partition**  $A$  into  $8 \times 8$  blocks.
  - ▶ **Transform** each block using the *Discrete Cosine Transform (DCT)*.
  - ▶ **Quantize** the elements in each block.
  - ▶ **Encode** using Huffman encoding.
- ▶ Note that the quantization step makes JPEG an example of lossy compression.



- ▶ Given an  $N \times N$  image  $A$  ( $A$  square and  $N$  is divisible by 8 for simplicity):
  - ▶ **Subtract** 127 from each element in  $A$ .
  - ▶ **Partition**  $A$  into  $8 \times 8$  blocks.
  - ▶ **Transform** each block using the *Discrete Cosine Transform (DCT)*.
  - ▶ **Quantize** the elements in each block.
  - ▶ **Encode** using Huffman encoding.
- ▶ Note that the quantization step makes JPEG an example of lossy compression.



- ▶ Given an  $N \times N$  image  $A$  ( $A$  square and  $N$  is divisible by 8 for simplicity):
  - ▶ **Subtract** 127 from each element in  $A$ .
  - ▶ **Partition**  $A$  into  $8 \times 8$  blocks.
  - ▶ **Transform** each block using the *Discrete Cosine Transform (DCT)*.
  - ▶ **Quantize** the elements in each block.
  - ▶ **Encode** using Huffman encoding.
- ▶ Note that the quantization step makes JPEG an example of lossy compression.



- ▶ Given an  $N \times N$  image  $A$  ( $A$  square and  $N$  is divisible by 8 for simplicity):
  - ▶ **Subtract** 127 from each element in  $A$ .
  - ▶ **Partition**  $A$  into  $8 \times 8$  blocks.
  - ▶ **Transform** each block using the *Discrete Cosine Transform (DCT)*.
  - ▶ **Quantize** the elements in each block.
  - ▶ **Encode** using Huffman encoding.
- ▶ Note that the quantization step makes JPEG an example of lossy compression.



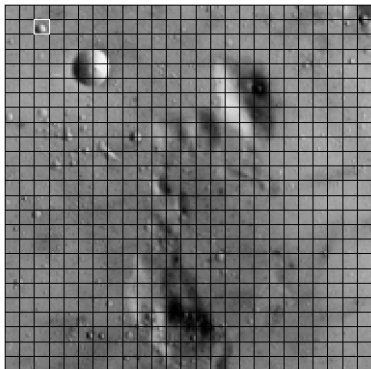
- ▶ Given an  $N \times N$  image  $A$  ( $A$  square and  $N$  is divisible by 8 for simplicity):
  - ▶ **Subtract** 127 from each element in  $A$ .
  - ▶ **Partition**  $A$  into  $8 \times 8$  blocks.
  - ▶ **Transform** each block using the *Discrete Cosine Transform (DCT)*.
  - ▶ **Quantize** the elements in each block.
  - ▶ **Encode** using Huffman encoding.
- ▶ Note that the quantization step makes JPEG an example of lossy compression.



Consider the moon surface image stored in *A*. The dimensions are  $200 \times 200$ .



We partition  $A$  into  $8 \times 8$  blocks. There are  $25 \times 25$  blocks. We have highlighted one block (call it  $A_{23}$ ) for use as a running example:



The elements in  $A_{23}$  are:

$$A_{23} = \begin{bmatrix} 140 & 142 & 127 & 126 & 124 & 129 & 135 & 129 \\ 142 & 142 & 132 & 125 & 125 & 124 & 130 & 133 \\ 143 & 135 & 122 & 134 & 140 & 115 & 124 & 139 \\ 121 & 93 & 104 & 159 & 209 & 166 & 117 & 130 \\ 96 & 58 & 83 & 150 & 220 & 224 & 142 & 123 \\ 81 & 37 & 68 & 124 & 203 & 231 & 161 & 113 \\ 88 & 29 & 56 & 101 & 166 & 201 & 141 & 118 \\ 126 & 72 & 57 & 92 & 131 & 135 & 135 & 133 \end{bmatrix}$$



Shifting by 127 gives

$$\tilde{A}_{23} = \begin{bmatrix} 13 & 15 & 0 & -1 & -3 & 2 & 8 & 2 \\ 15 & 15 & 5 & -2 & -2 & -3 & 3 & 6 \\ 16 & 8 & -5 & 7 & 13 & -12 & -3 & 12 \\ -6 & -34 & -23 & 32 & 82 & 39 & -10 & 3 \\ -31 & -69 & -44 & 23 & 93 & 97 & 15 & -4 \\ -46 & -90 & -59 & -3 & 76 & 104 & 34 & -14 \\ -39 & -98 & -71 & -26 & 39 & 74 & 14 & -9 \\ -1 & -55 & -70 & -35 & 4 & 8 & 8 & 6 \end{bmatrix}$$



The DCT is the  $8 \times 8$  matrix

$$U = \frac{1}{2} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \cos(\frac{\pi}{16}) & \cos(\frac{3\pi}{16}) & \cos(\frac{5\pi}{16}) & \cos(\frac{7\pi}{16}) & \cos(\frac{9\pi}{16}) & \cos(\frac{11\pi}{16}) & \cos(\frac{13\pi}{16}) & \cos(\frac{15\pi}{16}) \\ \cos(\frac{\pi}{8}) & \cos(\frac{3\pi}{8}) & \cos(\frac{5\pi}{8}) & \cos(\frac{7\pi}{8}) & \cos(\frac{9\pi}{8}) & \cos(\frac{11\pi}{8}) & \cos(\frac{13\pi}{8}) & \cos(\frac{15\pi}{8}) \\ \cos(\frac{3\pi}{16}) & \cos(\frac{9\pi}{16}) & \cos(\frac{15\pi}{16}) & \cos(\frac{21\pi}{16}) & \cos(\frac{27\pi}{16}) & \cos(\frac{33\pi}{16}) & \cos(\frac{39\pi}{16}) & \cos(\frac{45\pi}{16}) \\ \cos(\frac{\pi}{4}) & \cos(\frac{3\pi}{4}) & \cos(\frac{5\pi}{4}) & \cos(\frac{7\pi}{4}) & \cos(\frac{9\pi}{4}) & \cos(\frac{11\pi}{4}) & \cos(\frac{13\pi}{4}) & \cos(\frac{15\pi}{4}) \\ \cos(\frac{5\pi}{16}) & \cos(\frac{15\pi}{16}) & \cos(\frac{25\pi}{16}) & \cos(\frac{35\pi}{16}) & \cos(\frac{45\pi}{16}) & \cos(\frac{55\pi}{16}) & \cos(\frac{65\pi}{16}) & \cos(\frac{75\pi}{16}) \\ \cos(\frac{3\pi}{8}) & \cos(\frac{9\pi}{8}) & \cos(\frac{15\pi}{8}) & \cos(\frac{21\pi}{8}) & \cos(\frac{27\pi}{8}) & \cos(\frac{33\pi}{8}) & \cos(\frac{39\pi}{8}) & \cos(\frac{45\pi}{8}) \\ \cos(\frac{7\pi}{16}) & \cos(\frac{21\pi}{16}) & \cos(\frac{35\pi}{16}) & \cos(\frac{49\pi}{16}) & \cos(\frac{63\pi}{16}) & \cos(\frac{77\pi}{16}) & \cos(\frac{91\pi}{16}) & \cos(\frac{105\pi}{16}) \end{bmatrix}$$



In decimal form, we have  $U =$

0.353553	0.353553	0.353553	0.353553	0.353553	0.353553	0.353553	0.353553
0.490393	0.415735	0.277785	0.097545	-0.097545	-0.277785	-0.415735	-0.490393
0.461940	0.191342	-0.191342	-0.461940	-0.461940	-0.191342	0.191342	0.461940
0.415735	-0.097545	-0.490393	-0.277785	0.277785	0.490393	0.097545	-0.415735
0.353553	-0.353553	-0.353553	0.353553	0.353553	-0.353553	-0.353553	0.353553
0.277785	-0.490393	0.097545	0.415735	-0.415735	-0.097545	0.490393	-0.277785
0.191342	-0.461940	0.461940	-0.191342	-0.191342	0.461940	-0.461940	0.191342
0.097545	-0.277785	0.415735	-0.490393	0.490393	-0.415735	0.277785	-0.097545



- ▶ We won't talk much about  $U$  today, but we will note:
  - ▶  $U$  is *orthogonal* so  $U^{-1} = U^T$ .
  - ▶  $U$  maps the vector  $(1, 1, 1, 1, 1, 1, 1, 1)^T$  to  $(2\sqrt{2}, 0, 0, 0, 0, 0, 0, 0)^T$
- ▶ To apply the DCT to  $A_{23}$ , we compute  $UAU^T$ .



- ▶ We won't talk much about  $U$  today, but we will note:
  - ▶  $U$  is *orthogonal* so  $U^{-1} = U^T$ .
  - ▶  $U$  maps the vector  $(1, 1, 1, 1, 1, 1, 1, 1)^T$  to  $(2\sqrt{2}, 0, 0, 0, 0, 0, 0, 0)^T$
- ▶ To apply the DCT to  $A_{23}$ , we compute  $UAU^T$ .



- ▶ We won't talk much about  $U$  today, but we will note:
  - ▶  $U$  is *orthogonal* so  $U^{-1} = U^T$ .
  - ▶  $U$  maps the vector  $(1, 1, 1, 1, 1, 1, 1, 1)^T$  to  $(2\sqrt{2}, 0, 0, 0, 0, 0, 0, 0)^T$
- ▶ To apply the DCT to  $A_{23}$ , we compute  $UAU^T$ .



- ▶ We won't talk much about  $U$  today, but we will note:
  - ▶  $U$  is *orthogonal* so  $U^{-1} = U^T$ .
  - ▶  $U$  maps the vector  $(1, 1, 1, 1, 1, 1, 1, 1)^T$  to  $(2\sqrt{2}, 0, 0, 0, 0, 0, 0, 0)^T$
- ▶ To apply the DCT to  $A_{23}$ , we compute  $UAU^T$ .



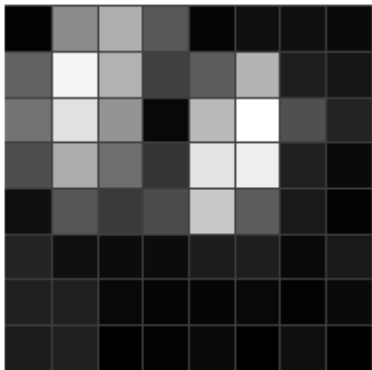
We have

$$T_{23} = U\tilde{A}_{23}U^T$$

$$= \begin{bmatrix} 2.875 & -136.615 & -80.091 & 131.114 & 52.125 & -7.580 & 20.592 & 2.797 \\ 55.579 & 121.818 & 48.461 & -92.981 & -18.519 & -9.120 & -25.666 & -1.5478 \\ -50.705 & 41.645 & 95.027 & -35.707 & -27.598 & 15.933 & -12.217 & -0.976 \\ 13.668 & -64.300 & -58.929 & 50.519 & -24.407 & 5.640 & 19.388 & -5.375 \\ 13.125 & 10.890 & -7.207 & -0.237 & 20.375 & -7.510 & -6.621 & -1.543 \\ -8.119 & -4.627 & -12.092 & 2.622 & 8.589 & -1.813 & 4.358 & 1.207 \\ 1.384 & 8.555 & 16.283 & -7.630 & -3.587 & 3.495 & -12.277 & 2.213 \\ -4.652 & -2.066 & 7.586 & -1.387 & -3.124 & 2.021 & 3.110 & -4.523 \end{bmatrix}$$



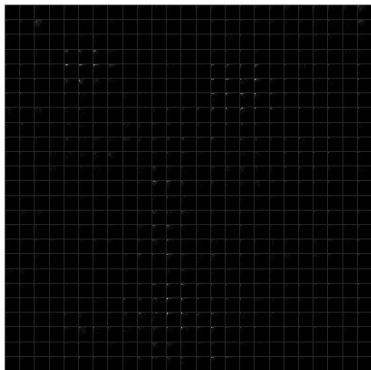
Perhaps a plot is more informative:



The DCT tends to store information about all 64 values into a few values and “shove” them to the upper left of the output. The remaining values are either 0 or approximately 0.



Here is a picture of the DCT of the entire image  $\tilde{A}$ :



The quantization step involves dividing element-wise the individual  $8 \times 8$  blocks by the matrix

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

and then rounding the results to the nearest integer. Note that the rounding step is irreversible.



For our  $8 \times 8$  block  $T_{23}$  we have

$$\begin{bmatrix} 0 & -12 & -8 & 8 & 2 & 0 & 0 & 0 \\ 5 & 10 & 3 & -5 & -1 & 0 & 0 & 0 \\ -4 & 3 & 6 & -1 & -1 & 0 & 0 & 0 \\ 1 & -4 & -3 & 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$



- ▶ We then add 128 to each element of each block, and then apply Huffman encoding to the result.
- ▶ The compressed image can be represented using 57231 bits or 1.43078 bits per pixel.
- ▶ That is quite a substantial savings over the original 320000 bits needed to represent the image.
- ▶ There are some drawbacks:
  - ▶ the compression is lossy
  - ▶ the uncompressed image is "blocky"
  - ▶ the transform doesn't handle edge effects



- ▶ We then add 128 to each element of each block, and then apply Huffman encoding to the result.
- ▶ The compressed image can be represented using 57231 bits or 1.43078 bits per pixel.
- ▶ That is quite a substantial savings over the original 320000 bits needed to represent the image.
- ▶ There are some drawbacks:
  - ▶ the compression is lossy
  - ▶ the uncompressed image is "blocky"
  - ▶ the transform doesn't handle edge effects



- ▶ We then add 128 to each element of each block, and then apply Huffman encoding to the result.
- ▶ The compressed image can be represented using 57231 bits or 1.43078 bits per pixel.
- ▶ That is quite a substantial savings over the original 320000 bits needed to represent the image.
- ▶ There are some drawbacks:
  - ▶ the compression is lossy
  - ▶ the uncompressed image is "blocky"
  - ▶ the transform doesn't handle edge effects



- ▶ We then add 128 to each element of each block, and then apply Huffman encoding to the result.
- ▶ The compressed image can be represented using 57231 bits or 1.43078 bits per pixel.
- ▶ That is quite a substantial savings over the original 320000 bits needed to represent the image.
- ▶ There are some drawbacks:
  - ▶ the compression is lossy
  - ▶ the uncompressed image is “blocky”
  - ▶ the transform doesn’t handle edge effects



- ▶ We then add 128 to each element of each block, and then apply Huffman encoding to the result.
- ▶ The compressed image can be represented using 57231 bits or 1.43078 bits per pixel.
- ▶ That is quite a substantial savings over the original 320000 bits needed to represent the image.
- ▶ There are some drawbacks:
  - ▶ the compression is lossy
  - ▶ the uncompressed image is “blocky”
  - ▶ the transform doesn’t handle edge effects



- ▶ We then add 128 to each element of each block, and then apply Huffman encoding to the result.
- ▶ The compressed image can be represented using 57231 bits or 1.43078 bits per pixel.
- ▶ That is quite a substantial savings over the original 320000 bits needed to represent the image.
- ▶ There are some drawbacks:
  - ▶ the compression is lossy
  - ▶ the uncompressed image is “blocky”
  - ▶ the transform doesn't handle edge effects



- ▶ We then add 128 to each element of each block, and then apply Huffman encoding to the result.
- ▶ The compressed image can be represented using 57231 bits or 1.43078 bits per pixel.
- ▶ That is quite a substantial savings over the original 320000 bits needed to represent the image.
- ▶ There are some drawbacks:
  - ▶ the compression is lossy
  - ▶ the uncompressed image is “blocky”
  - ▶ the transform doesn’t handle edge effects



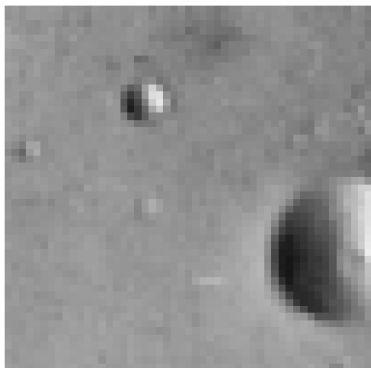
Compressed:



Original:



Here is a blow-up of the original:



Here is a blow-up of the uncompressed image:

