The blue and green colors are actually the same
How is it that a 4MP image can be compressed to a few hundred KB without a noticeable change?
Lossy Image Compression (JPEG)

Block-based Discrete Cosine Transform (DCT)
Using DCT in JPEG

• The first coefficient $B(0,0)$ is the DC component, the average intensity

• The top-left coeffs represent low frequencies, the bottom right – high frequencies
Image compression using DCT

- Quantize
  - More coarsely for high frequencies (which also tend to have smaller values)
  - Many quantized high frequency values will be zero

- Encode
  - Can decode with inverse dct

Filter responses

\[
G = \begin{bmatrix}
-415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\
-46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\
-48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\
12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\
-7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\
-1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\
-0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68
\end{bmatrix}
\]

\[G \overset{u}{\rightarrow} v
\]

Quantization table

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

Quantized values

\[
B = \begin{bmatrix}
-26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\
0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\
-3 & 1 & 5 & -1 & 0 & 0 & 0 & 0 \\
-3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]
JPEG Compression Summary

1. Convert image to YCrCb
2. Subsample color by factor of 2
   - People have bad resolution for color
3. Split into blocks (8x8, typically), subtract 128
4. For each block
   a. Compute DCT coefficients
   b. Coarsely quantize
      • Many high frequency components will become zero
   c. Encode (with run length encoding and then Huffman coding for leftovers)
Why do we get different, distance-dependent interpretations of hybrid images?
Clues from Human Perception

- Early processing in humans filters for various orientations and scales of frequency
- Perceptual cues in the mid-high frequencies dominate perception
- When we see an image from far away, we are effectively subsampling it

Early Visual Processing: Multi-scale edge and blob filters
Campbell-Robson contrast sensitivity curve
Hybrid Image in FFT

Hybrid Image

Low-passed Image + High-passed Image
Review: Image filtering

\[ h[m,n] = \sum_{k,l} f[k,l] g[m+k, n+l] \]
Image filtering

\[ f[\cdot,\cdot] \quad h[\cdot,\cdot] \]

\[ g[\cdot,\cdot] \frac{1}{9} \]

\[ h[m,n] = \sum_{k,l} f[k,l] \ g[m+k, n+l] \]

Credit: S. Seitz
Image filtering

\[ h[m, n] = \sum_{k,l} f[k, l] g[m + k, n + l] \]
Filtering in spatial domain

\[
\begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix}
\]

\[
\text{intensity image} \ast \begin{bmatrix}
1 & 0 & -1 \\
2 & 0 & -2 \\
1 & 0 & -1 \\
\end{bmatrix} = \text{resulting image}
\]
Filtering in frequency domain
Review of Filtering

• Filtering in frequency domain
  – Can be faster than filtering in spatial domain (for large filters)
  – Can help understand effect of filter
  – Algorithm:
    1. Convert image and filter to fft (fft2 in matlab)
    2. Pointwise-multiply ffts
    3. Convert result to spatial domain with ifft2
Review of Filtering

• Linear filters for basic processing
  – Edge filter (high-pass)
  – Gaussian filter (low-pass)

\([-1 \ 1]\)

FFT of Gradient Filter

FFT of Gaussian

Gaussian
Things to Remember

• Sometimes it makes sense to think of images and filtering in the frequency domain
  – Fourier analysis

• Can be faster to filter using FFT for large images (N logN vs. N^2 for auto-correlation)

• Images are mostly smooth
  – Basis for compression

• Remember to low-pass before sampling
Previous Lectures

• We’ve now touched on the first three chapters of Szeliski.
  – 1. Introduction
  – 2. Image Formation
  – 3. Image Processing

• Now we’re moving on to
  – 4. Feature Detection and Matching
  – Multiple views and motion (7, 8, 11)
Edge / Boundary Detection

Computer Vision

James Hays

Szeliski 4.2
Edge detection

- **Goal:** Identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels

- **Ideal:** artist’s line drawing (but artist is also using object-level knowledge)

Source: D. Lowe
Why do we care about edges?

- Extract information, recognize objects
- Recover geometry and viewpoint

Vanishing point
Vanishing line
Vertical vanishing point (at infinity)
Origin of Edges

- Edges are caused by a variety of factors

Source: Steve Seitz
Closeup of edges

Source: D. Hoiem
Closeup of edges
Closeup of edges
Closeup of edges

Source: D. Hoiem
Characterizing edges

An edge is a place of rapid change in the image intensity function.
Intensity profile
With a little Gaussian noise
Effects of noise

- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal

Where is the edge?

Source: S. Seitz
Effects of noise

• Difference filters respond strongly to noise
  – Image noise results in pixels that look very different from their neighbors
  – Generally, the larger the noise the stronger the response

• What can we do about it?

Source: D. Forsyth
Solution: smooth first

To find edges, look for peaks in \( \frac{d}{dx} (f * g) \)

Source: S. Seitz
Derivative theorem of convolution

- Differentiation is convolution, and convolution is associative:
  \[ \frac{d}{dx} (f * g) = f * \frac{d}{dx} g \]

- This saves us one operation:

![Graphs showing differentiation and convolution](image-url)

Source: S. Seitz
Derivative of Gaussian filter

\[ * [1 \ -1] = \]
Smoothing derivative removes noise, but blurs edge. Also finds edges at different “scales”.

Source: D. Forsyth
Designing an edge detector

• Criteria for a good edge detector:
  – **Good detection**: the optimal detector should find all real edges, ignoring noise or other artifacts
  – **Good localization**
    • the edges detected must be as close as possible to the true edges
    • the detector must return one point only for each true edge point

• Cues of edge detection
  – Differences in color, intensity, or texture across the boundary
  – Continuity and closure
  – High-level knowledge

Source: L. Fei-Fei
Canny edge detector

• This is probably the most widely used edge detector in computer vision

• Theoretical model: step-edges corrupted by additive Gaussian noise

• Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of signal-to-noise ratio and localization


27,000 citations!
Example

original image (Lena)
Derivative of Gaussian filter

$x$-direction

$y$-direction
Compute Gradients (DoG)

X-Derivative of Gaussian  Y-Derivative of Gaussian  Gradient Magnitude
Get Orientation at Each Pixel

- Threshold at minimum level
- Get orientation

\[ \text{theta} = \text{atan2}(\text{gy}, \text{gx}) \]
Non-maximum suppression for each orientation

At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.

Source: D. Forsyth
Sidebar: Interpolation options

- `imx2 = imresize(im, 2, interpolation_type)`

- `'nearest'`
  - Copy value from nearest known
  - Very fast but creates blocky edges

- `'bilinear'`
  - Weighted average from four nearest known pixels
  - Fast and reasonable results

- `'bicubic'` (default)
  - Non-linear smoothing over larger area (4x4)
  - Slower, visually appealing, may create negative pixel values

Before Non-max Suppression
After non-max suppression
Hysteresis thresholding

- Threshold at low/high levels to get weak/strong edge pixels
- Do connected components, starting from strong edge pixels
Hysteresis thresholding

• Check that maximum value of gradient value is sufficiently large
  – drop-outs? use **hysteresis**
    • use a high threshold to start edge curves and a low threshold to continue them.

Source: S. Seitz
Final Canny Edges
Canny edge detector

1. Filter image with x, y derivatives of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   - Thin multi-pixel wide “ridges” down to single pixel width
4. Thresholding and linking (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

• MATLAB: edge(image, ‘canny’)

Source: D. Lowe, L. Fei-Fei
The choice of $\sigma$ depends on desired behavior

- large $\sigma$ detects large scale edges
- small $\sigma$ detects fine features

Source: S. Seitz