**CS559 – Lecture 9**  
**JPEG, Raster Algorithms**

These are course notes (not used as slides)  
Written by Mike Gleicher, Sept. 2005  
With some slides adapted from the notes of Stephen Chenney

---

**Lossy Coding 2**

- Suppose we can only send a fraction of the image  
  - Which part?
- Send half an image:  
  - Send the top half (not too good)  
  - Halve the image in size (send the low frequency half)
- Idea: re-order (transform) the image so the important stuff is first

---

**Perceptual Image Coding**

- Idea: lose stuff in images that is least important perceptually  
  - Stuff least likely to notice  
  - Stuff most likely to convey image
- Who knows about this stuff: The experts!  
  - Joint Picture Experts Group  
  - Idea of perceptual image coding

---

**JPEG**

- Key Ideas  
  - Frequency Domain (small details are less important)  
  - Block Transforms (works on 8x8 blocks)  
    - Discrete Cosine Transform (DCT)  
  - Control Quantization of frequency components  
    - More quality = use more bits  
    - Generally, use less bits for HF

---

**JPEG**

- Multi-stage process intended to get very high compression with controllable quality degradation
- Start with YIQ color  
  - Why? Recall, it’s the color standard for TV

---

**Discrete Cosine Transform**

- A transformation to convert from the spatial to frequency domain – done on 8x8 blocks
- Why? Humans have varying sensitivity to different frequencies, so it is safe to throw some of them away
- Basis functions:
Quantization

• Reduce the number of bits used to store each coefficient by dividing by a given value
  – If you have an 8 bit number (0-255) and divide it by 8, you get a number between 0-31 (5 bits = 8 bits – 3 bits)
  – Different coefficients are divided by different amounts
  – Perceptual issues come in here
• Achieves the greatest compression, but also quality loss
• "Quality" knob controls how much quantization is done

Entropy Coding

• Standard lossless compression on quantized coefficients
  – Delta encode the DC components
  – Run length encode the AC components
    • Lots of zeros, so store number of zeros then next value
  – Huffman code the encodings

Lossless JPEG With Prediction

• Predict what the value of the pixel will be based on neighbors
• Record error from prediction
  – Mostly error will be near zero
• Huffman encode the error stream
• Variation works really well for fax messages

Video Compression

• Much bigger problem (many images per second)
• Could code each image separately
  – Motion JPEG
  – DV (need to make each image a fixed size for tape)
• Need to take advantage that different images are similar
  – Encode the Changes?

MPEG

• Motion Picture Experts Group
  – Standards organization
• MPEG-1 simple format for videos (fixed size)
• MPEG-2 general, scalable format for video
• MPEG-4 computer format (complicated, flexible)
• MPEG-7 future format
• What about MPEG-3? – it doesn’t exist (?)
  – MPEG-1 Layer 3 = audio format

MPEG Concepts

• Keyframe
  – Need something to start from
  – "Reset" when differences get too far
• Difference encoding
  – Differences are smaller/easier to encode than images
• Motion
  – Some differences are groups of pixels moving around
  – Block motion
  – Object motion (models)
**MPEG**

- **Frame 1** (keyframe) → **Frame 1 (comp) + motion** → **Find motion vectors** → **Frame 2** (keyframe) → **Encode vectors** → **Encode Difference (lossy)** → **Frame 2**

**Other Practical Tricks…**

- Don’t really know what is in image
  - Makes it hard to make changes
- Getting rid of noise
  - Low pass filters
  - Edge-preserving filtering
- "Sharpening"
  - Can we actually do it? (no – adding aliasing)
  - High-Pass attenuation
  - Unsharp mask (subtract out low frequencies)
- Feathering
  - Sharp transitions are noticeable
  - Blend/Blur around edges of changes

**Geometric Graphics**

- Mathematical descriptions of sets of points
  - Rather than sampled representations
- Ultimately, need sampled representations for display
- Rasterization
  - Usually done by low-level
    - OS / Graphics Library / Hardware
    - Hardware implementations counter-intuitive
      - Modern hardware doesn’t work anything like what you’d expect

**Drawing Points**

- What is a point?
  - Position – without any extent
  - Can’t see it – since it has no extent, need to give it some
- Position requires co-ordinate system
  - Consider these in more depth later
- How does a point relate to a sampled world?
  - Points at samples?
  - Pick closest sample?
  - Give points finite extent and use little square model?
  - Use proper sampling

**Sampling a point**

- Point is a spike – need to LPF
  - Gives a circle w/roll-off
- Point sample this
  - Or…
    - Samples look in circular (kernel shaped) regions around their position
- But, we can actually record a unique “splat” for any individual point

**Anti-Aliasing**

- Anti-Aliasing is about avoiding aliasing
  - once you’ve aliased, you’ve lost
- Draw in a way that is more precise
  - E.g. points spread out over regions
- Not always better
  - Lose contrast, might not look even if gamma is wrong, might need to go to binary display, …
Line drawing

• Was really important, now, not so important
• Let us replace expensive vector displays with cheaper raster ones
• Modern hardware does it differently
  – Actually, doesn’t draw lines, draws small, filled polygons
• Historically significant algorithms

Line Drawing (2)

• Consider the integer version
  – (x1,y1) \rightarrow (x2,y2) are integers
  – Not anti-aliased (binary decision on pixels)
• Naïve strawman version:
  – \( Y = mx + b \)

For \( x = x_1 \) to \( x_2 \)

\[ y = mx + b \]

set(\( x, y \))

• Problems:
  – Too much math (floating point)
  – gaps

Brezenham’s algorithm
(and variants)

• Consider only 1 octant (get others by symmetry)
  – \( 0 \geq m > = 1 \)
• Loop over x pixels
  – Guarantees 1 per column
• For each pixel, either move up 1 or not
  – If you plotted \( x,y \) then choose either \( x+1,y \) or \( x+1,y+1 \)
  – Trick: how to decide which one easily
  – Same method works for circles (just need different test)
• Decision variable
  – Implicit equation for line (\( d=0 \) means on the line)

Midpoint method

\[ x_k \]

\[ y_k \]

\[ d_1 = y_k - y \]

\[ d_2 = y_{k+1} - y \]

If \( d_1 < d_2 \) pick \( y_k \)

\[ \Delta d = d_1 - d_2 \]

\[ \Delta d = (y_{k+1} - y_k) - (y_k + 1 - y) \]

If \( d_1 \neq d_2 \) pick \( y_{k+1} \)

\[ \Delta d = 2(mx + b) - 2y_k - 1 \]

Multiply both sides by \( \Delta x \) (since we know its positive)

\[ \Delta d \Delta x = 2y \Delta x y_k + 2 \Delta y - 2 \Delta x y_k - \Delta x \]

\[ \Delta d \Delta x = 2 \Delta y x_k + 2 \Delta y \Delta x + 2 \Delta x y_k + c \]

\[ c = 2 \Delta y + \Delta x(2b - 1) \]

(all the stuff that doesn’t depend on \( k \))

Derivation

Incremental Algorithm

• Suppose we know \( p_k \) – what is \( p_{k+1} \)?
• \[ p_{k+1} = p_k + 2 \Delta y - 2 \Delta x (y_{k+1} - y_k) \]
  – Since \( x_{k+1} = x_k + 1 \)
• And \( y_{k+1} - y_k \) is either 1 or 0, depending on \( p_k \)
Brezenham’s Algorithm

- \( P_k = 2 \Delta y + x \)
- \( Y = y_1 \)
- For \( X = x_1 \) to \( x_2 \)
  - Set \( X, Y \)
  - If \( P_k < 0 \)
    - \( Y += 1 \)
    - \( P_k += 2 \Delta y - 2 \Delta x \)
  - Else: \( P_k += 2 \Delta y \)

Why is this cool?

- No division!
- No floating point!
- No gaps!
- Extends to circles

But…
- Jaggies
- Lines get thinner as they approach 45 degrees
- Can’t do thick primitives