Non-linearity of intensity

- Non-linear mapping from "amount of light" to perceived brightness
- Want uniform mapping of intensities -> perception
  - Level 1, 2, 3, ..., 255 -> 1, 1.01, 1.02, ..., 99, 100

- Worse: displays are non-linear too
  - Voltage -> amount of light is non-linear
  - Different displays are different

- Want to linearize the system
  - Intensity levels map nicely to perceived levels

Gamma correction

- Idea: put a non-linear function between intensity and output
  - Done as the last step (usually) – after all computations

- Could create arbitrary functions for mapping
  - Too cumbersome

- Exponential is a good approximate model
  - Exponential non-linearity of perception
  - Exponential power laws in CRTs

Modeling a display device

- 5/2 power law (five-halves)
  - Models physics of a CRT
  - Real CRTs are close, LCDs designed to be similar

- \[ L = M (i+\epsilon)^{\gamma} \]
  - \( i \) = input intensity value
  - \( L \) = amount of light
  - \( \epsilon \) = since zero isn’t really black
  - \( M \) = maximum intensity
  - \( \gamma \) = specific property of display

Linearizing the display

- Define a function \( g \) that corrects for non-linearity

- \[ L = M (g(i))^{\gamma} \] (ignoring \( \epsilon \))
  - \( G \) = \( 1/\gamma \)

- Where do we get \( \gamma \) from?
  - Pick it so things look right

- Note: 1st order approximation (very simple)
  - Only 1 parameter to specify (\( \gamma \)), many factors

Gamma correction

- Want value 0 = minimum intensity
- Want value max (1 or 255) = maximum intensity
  --- those 2 are easy to get

- Pick one more point
  - Midpoint should be 50%
  - Easy – show 50% black white + 50% gray
  - Adjust gamma until it looks the same

- All this happens "behind the scenes"
- Everything gets harder when we deal with color
What to store in the frame buffer?

- Frame Buffer = rectangular chunk of memory
- Intensity measurements
  - Deal with color later, basically store multiple monochrome
- Continuous range of intensities
  - 8-9 bits of precision ideally
    - More since can’t get exactly right (10-12 bits)
    - More since want more dynamic range (12-14 bits)
    - More since want linear space to make math easy (16-32 bits)
- Discrete set of choices – QUANTIZATION
  - Inks, palettes, color tables, ...
  - Less storage cost + Color table animation

Faking more “colors” than you have

- Eye tends to average stuff together
  - Trade spatial resolution for intensity resolution

Quantization

- What happens when we want smaller numbers of values?
  - Black and white for printing
  - Limited color palette
- Old problem
  - Printing
  - Artists (pen and ink drawing)

Thresholding

- Threshold – pick value / above or below
- Each pixel picks nearest value
  - 49% looks the same as 1%
  - 49% looks very different than 51%
- Better: trade spatial resolution for value resolution
  - Brain blurs stuff together anyway
  - Art example: hatching to show “gray”

Dithering

- Add some random noise
- 50% + noise -> half black, half white
- Values at extreme less likely to get changed
- Eye doesn’t mind noise as much as it does blocky edges

Patterns

- Make display resolution greater than image resolution
- Each pixel gives a block w/appropriate number of pixels on
  - For example: 3x3 blocks give 10 levels
Ordered Halftoning

- Do patterns, but apply for each pixel separately (no scaling images)
- Divide image into nxn blocks (repeated pattern)
  - Each pixel decides if it would be turned on if its value was used to pick the pattern
- Easy implementation: Threshold Matrix or Mask
  - Used in traditional printing (a halftone screen)
  - Each pixel has a different threshold
  - Example: 4 values

<table>
<thead>
<tr>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

More on Halftone screens

- Other factors can go into designs
- Cluster things together (since you know that ink tends to clump)
- Or make artistic effects
- Can be used with dithering (adding randomness)

Error Minimization

- For each pixel, compute error (how different from result)
  - Try to pick result to minimize error
- Global minimization: each pixel should equal average of destination image
  - Too hard to solve efficiently since it’s a combinatorial problem
- Local minimization: each pixel should be as close as possible (thresholding) – but spread error around evenly

Error Diffusion

- Many ways to do this
- Old standby: Floyd-Steinberg
- Start at upper left
- Pick value for pixel
- Push error into neighbors (that haven’t been visited yet)
- Problem: directional artifacts (fix by alternating directions)
- Good news: generalizes (colors, multiple levels, …)