Lecture 3:
The Camera Image Processing Pipeline
(part 2: tone mapping and autofocus)

Visual Computing Systems
Stanford CS348K, Fall 2020
Previous class and today...

The pixels you see on screen are quite different than the values recorded by the sensor in a modern digital camera.

Computation is now a fundamental aspect of producing high-quality pictures.
## Summary: simplified image processing pipeline

- Correct pixel defects
- Align and merge (to create high signal to noise ratio RAW image)
- Correct for sensor bias (using measurements of optically black pixels)
- Vignetting compensation
- White balance

<table>
<thead>
<tr>
<th>Component</th>
<th>Bits per Pixel</th>
<th>Pixel Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vignetting compensation</td>
<td>10-12 bits per pixel</td>
<td>1 intensity value per pixel, Pixel values linear in energy</td>
</tr>
<tr>
<td>White balance</td>
<td>1 intensity value per pixel</td>
<td>Pixel values linear in energy</td>
</tr>
</tbody>
</table>

- Demosaic
- Denoise

<table>
<thead>
<tr>
<th>Component</th>
<th>Bits per Pixel</th>
<th>Pixel Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demosaic</td>
<td>3x10 bits per pixel</td>
<td>RGB intensity per pixel, Pixel values linear in energy</td>
</tr>
<tr>
<td>Denoise</td>
<td>3x10 bits per pixel</td>
<td>RGB intensity per pixel, Pixel values linear in energy</td>
</tr>
</tbody>
</table>

- Gamma Correction (non-linear mapping)

<table>
<thead>
<tr>
<th>Component</th>
<th>Bits per Pixel</th>
<th>Pixel Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gamma Correction</td>
<td>3x8-bits per pixel</td>
<td>Pixel values perceptually linear</td>
</tr>
</tbody>
</table>

- Local tone mapping
- Final adjustments sharpen, fix chromatic aberrations, hue adjust, etc.

Today
Auto Exposure and Tone Mapping
Global tone mapping

- Measured image values: 10-12 bits / pixel, but common image formats (8-bits/ pixel)
- How to convert 12 bit number to 8 bit number?

![Diagram showing tone mapping](image)

- Allow many pixels to “blow out” (detail in dark regions)
- Allow many pixels to clamp to black (detail in bright regions)

Stanford CS348K, Spring 2020
Global tone mapping

- Measured image values: 10-12 bits / pixel, but common image formats (8-bits/ pixel)
- How to convert 12 bit number to 8 bit number?

\[
\text{out}(x,y) = f(\text{in}(x,y))
\]
Lightness (perceived brightness) aka luma

\[
\text{Lightness (L*)} \quad \overset{?}{\longleftarrow} \quad \text{Luminance (Y)} = \int \lambda \quad \text{Radiance (energy spectrum from scene)}
\]

\[
\text{Spectral sensitivity of eye (eye’s response curve)}
\]

Dark adapted eye: \[ L^* \propto Y^{0.4} \]

Bright adapted eye: \[ L^* \propto Y^{0.5} \]

In a dark room, you turn on a light with luminance: \[ Y_1 \]

You turn on a second light that is identical to the first. Total output is now: \[ Y_2 = 2Y_1 \]

Total output appears \( 2^{0.4} = 1.319 \) times brighter to dark-adapted human

Note: Lightness (L*) is often referred to as luma (Y’)

Stanford CS348K, Spring 2020
Consider an image with pixel values encoding luminance (linear in energy hitting sensor)

Luminance (Y)

Perceived brightness: L*

Consider 12-bit sensor pixel:
Can represent 4096 unique luminance values in output image

Values are ~ linear in luminance since they represent the sensor’s response
Problem: quantization error

Many common image formats store 8 bits per channel (256 unique values)
Insufficient precision to represent brightness in darker regions of image

Luminance ($Y$)

Perceived brightness: $L^*$

$L^* = Y^{0.45}$

Bright regions of image: perceived difference between pixels that differ by one step in luminance is small!
(human may not even be able to perceive difference between pixels that differ by one step in luminance!)

Dark regions of image: perceived difference between pixels that differ by one step in luminance is large!
(quantization error: gradients in luminance will not appear smooth.)

Rule of thumb: human eye cannot differentiate <1% differences in luminance
Store lightness in 8-bit value, not luminance

Idea: distribute representable pixel values evenly with respect to perceived brightness, not evenly in luminance (make more efficient use of available bits)

Solution: pixel stores $Y^{0.45}$
Must compute $(\text{pixel\_value})^{2.2}$ prior to display on LCD

Warning: must take caution with subsequent pixel processing operations once pixels are encoded in a space that is not linear in luminance.

e.g., When adding images should you add pixel values that are encoded as lightness or as luminance?
Local tone mapping

- Different regions of the image undergo different tone mapping curves (preserve detail in both dark and bright regions)
Local tone adjustment

Improve picture’s aesthetics by locally adjusting contrast, boosting dark regions, decreasing bright regions (no physical basis)

Combined image (unique weights per pixel)

Image credit: Mertens 2007

Stanford CS348K, Spring 2020
High exposure weight
Low exposure image
High exposure weight
Combined result

Local tone mapping was performed on lightness (luma).
Now I added back in chrominance channels.
Challenge of merging images

Four exposures (weights not shown)

Merged result (based on weight masks)
Notice heavy “banding” since absolute intensity of different exposures is different

Merged result (after blurring weight mask)
Notice “halos” near edges
Review:
Frequency interpretation of images
Representing sound as a superposition of frequencies

\[ f_1(x) = \sin(\pi x) \]

\[ f_2(x) = \sin(2\pi x) \]

\[ f_4(x) = \sin(4\pi x) \]

\[ f(x) = f_1(x) + 0.75 f_2(x) + 0.5 f_4(x) \]
Audio spectrum analyzer: representing sound as a sum of its constituent frequencies

Intensity of low-frequencies (bass)

Intensity of high frequencies

Image credit: ONYX Apps
Fourier transform

- Convert representation of signal from spatial/temporal domain to frequency domain by projecting signal into its component frequencies

\[
f(\xi) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i x \xi} dx
\]

\[
eq \int_{-\infty}^{\infty} f(x) (\cos(2\pi \xi x) - i \sin(2\pi \xi x)) dx
\]

- 2D form:

\[
f(u, v) = \int \int f(x, y) e^{-2\pi i (ux + vy)} dx dy
\]
Visualizing the frequency content of images

Spatial domain result

Spectrum
Low frequencies only (smooth gradients)

Spatial domain result

Spectrum (after low-pass filter)
All frequencies above cutoff have 0 magnitude
Mid-range frequencies

Spatial domain result

Spectrum (after band-pass filter)
Mid-range frequencies

Spatial domain result

Spectrum (after band-pass filter)
High frequencies (edges)

Spatial domain result (strongest edges)

Spectrum (after high-pass filter) All frequencies below threshold have 0 magnitude
An image as a sum of its frequency components

\[
\begin{align*}
\text{Image} &= \text{Component 1} + \text{Component 2} + \text{Component 3} + \text{Component 4}
\end{align*}
\]
Another (linear) sharpening filter

\[
\text{blurred} = g \ast I \\
\text{fine} = I - \text{blurred} \quad \rightarrow \quad \text{Extract high frequencies}
\]

\[
\text{sharpened} = I + 0.5 \times \text{fine} \quad \rightarrow \quad \text{Boost high frequencies}
\]
But what if we wish to localize image edits both in space and in frequency?

(Adjust certain frequency content of image, in a particular region of the image)
Downsample

- Step 1: Remove high frequencies
- Step 2: Sparsely sample pixels (in this example: every other pixel)
Downsample

- **Step 1: Remove high frequencies**
- **Step 2: Sparsely sample pixels (in this example: every other pixel)**

```c
float input[(WIDTH+2) * (HEIGHT+2)];
float output[WIDTH/2 * HEIGHT/2];

float weights[] = {1/64, 3/64, 3/64, 1/64,    // 4x4 blur (approx Gaussian)
  3/64, 9/64, 9/64, 3/64,
  3/64, 9/64, 9/64, 3/64,
  1/64, 3/64, 3/64, 1/64};

for (int j=0; j<HEIGHT/2; j++) {
    for (int i=0; i<WIDTH/2; i++) {
        float tmp = 0.f;
        for (int jj=0; jj<4; jj++)
            for (int ii=0; ii<4; ii++)
                tmp += input[(2*j+jj)*(WIDTH+2) + (2*i+ii)] * weights[jj*3 + ii];
        output[j*WIDTH/2 + i] = tmp;
    }
}
```
Upsample
Via bilinear interpolation of samples from low resolution image
Upsample
Via bilinear interpolation of samples from low resolution image

float input[WIDTH * HEIGHT];
float output[2*WIDTH * 2*HEIGHT];

for (int j=0; j<2*HEIGHT; j++) {
    for (int i=0; i<2*WIDTH; i++) {
        int row = j/2;
        int col = i/2;
        float w1 = (i%2) ? .75f : .25f;
        float w2 = (j%2) ? .75f : .25f;

        output[j*2*WIDTH + i] = w1 * w2 * input[row*WIDTH + col] +
                                (1.0-w1) * w2 * input[row*WIDTH + col+1] +
                                w1 * (1-w2) * input[(row+1)*WIDTH + col] +
                                (1.0-w1)*(1.0-w2) * input[(row+1)*WIDTH + col+1];
    }
}
Gaussian pyramid

\[ G_0 = \text{image} \]

Each image in pyramid contains increasingly low-pass filtered signal

\[ \text{down}() = \text{downsample operation} \]
Gaussian pyramid
Gaussian pyramid

\[ G_1 \]
Gaussian pyramid

$G_2$
Gaussian pyramid

$G_3$
Gaussian pyramid

$G_4$
Gaussian pyramid
Laplacian pyramid

Each (increasingly numbered) level in Laplacian pyramid represents a band of (increasingly lower) frequency information in the image.

$L_0 = G_0 - \text{up}(G_1)$

$G_1 = \text{down}(G_0)$

[Burt and Adelson 83]
Laplacian pyramid

\[ L_0 = G_0 - \text{up}(G_1) \]

\[ L_1 = G_1 - \text{up}(G_2) \]
Laplacian pyramid

\[ L_0 = G_0 - \text{up}(G_1) \]
\[ L_1 = G_1 - \text{up}(G_2) \]
\[ L_2 = G_2 - \text{up}(G_3) \]
\[ L_3 = G_3 - \text{up}(G_4) \]
\[ L_4 = G_4 \]

Question: how do you reconstruct original image from its Laplacian pyramid?
Laplacian pyramid

\[ L_0 = G_0 - \text{up}(G_1) \]
Laplacian pyramid

\[ L_1 = G_1 - \text{up}(G_2) \]
Laplacian pyramid

\[ L_2 = G_2 - \text{up}(G_3) \]
Laplacian pyramid

$L_3 = G_3 - \text{up}(G_4)$
Laplacian pyramid

$L_4 = G_4 - up(G_5)$
Laplacian pyramid

\[ L_5 = G_5 \]
Summary

- Gaussian and Laplacian pyramids are image representations where each pixel maintains information about frequency content in a region of the image.

- $G_i(x,y)$ — frequencies up to limit given by $i$

- $L_i(x,y)$ — frequencies added to $G_{i+1}$ to get $G_i$

- Notice: to boost the band of frequencies in image around pixel $(x,y)$, increase coefficient $L_i(x,y)$ in Laplacian pyramid.
Use of Laplacian pyramid in tone mapping

- Compute weights for all Laplacian pyramid levels
- Merge pyramids (image features) not image pixels
- Then “flatten” merged pyramid to get final image
Challenges of merging images

Four exposures (weights not shown)

Merged result (after blurring weight mask)
Notice “halos” near edges

Merged result (based on multi-resolution pyramid merge)

Why does merging Laplacian pyramids work better than merging image pixels?
Consider low and high exposures of an edge.

<table>
<thead>
<tr>
<th>Low Exposure Laplacian Pyramid</th>
<th>High Exposure Laplacian Pyramid</th>
<th>Weight (for Low Exposure) Gaussian Pyramid</th>
<th>Merged (after flatten)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="L0_graph.png" alt="L0 graph" /></td>
<td><img src="L0_graph.png" alt="L0 graph" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="L1_graph.png" alt="L1 graph" /></td>
<td><img src="L1_graph.png" alt="L1 graph" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="L2_graph.png" alt="L2 graph" /></td>
<td><img src="L2_graph.png" alt="L2 graph" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="L3_graph.png" alt="L3 graph" /></td>
<td><img src="L3_graph.png" alt="L3 graph" /></td>
<td></td>
<td></td>
</tr>
<tr>
<td><img src="G4_graph.png" alt="G4 graph" /></td>
<td><img src="G4_graph.png" alt="G4 graph" /></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **clipped**
- **edge magnitude reduced, but detail remains on both sides**
Consider low and high exposures of flat image region

<table>
<thead>
<tr>
<th>Low Exposure Laplacian Pyramid</th>
<th>High Exposure Laplacian Pyramid</th>
<th>Weight (for Low Exposure) Gaussian Pyramid</th>
<th>Merged (after flatten)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="L0" alt="L0" /></td>
<td><img src="L0" alt="L0" /></td>
<td><img src="G0" alt="G0" /></td>
<td><img src="Merged" alt="Merged" /></td>
</tr>
<tr>
<td><img src="L1" alt="L1" /></td>
<td><img src="L1" alt="L1" /></td>
<td><img src="G1" alt="G1" /></td>
<td><img src="Merged" alt="Merged" /></td>
</tr>
<tr>
<td><img src="L2" alt="L2" /></td>
<td><img src="L2" alt="L2" /></td>
<td><img src="G2" alt="G2" /></td>
<td><img src="Merged" alt="Merged" /></td>
</tr>
<tr>
<td><img src="L3" alt="L3" /></td>
<td><img src="L3" alt="L3" /></td>
<td><img src="G3" alt="G3" /></td>
<td><img src="Merged" alt="Merged" /></td>
</tr>
<tr>
<td><img src="L3" alt="L3" /></td>
<td><img src="L3" alt="L3" /></td>
<td><img src="G4" alt="G4" /></td>
<td><img src="Merged" alt="Merged" /></td>
</tr>
<tr>
<td><img src="L0" alt="L0" /></td>
<td><img src="L0" alt="L0" /></td>
<td><img src="G4" alt="G4" /></td>
<td><img src="Merged" alt="Merged" /></td>
</tr>
</tbody>
</table>

Smooth transition despite sharp weight change (using hard weight change as an example)
## Summary: simplified image processing pipeline

- Correct pixel defects
- Align and merge (to create high signal to noise ratio RAW image)
- Correct for sensor bias (using measurements of optically black pixels)
- Vignetting compensation
- White balance
- Demosaic
- Denoise
- Gamma Correction (non-linear mapping)
- Local tone mapping
- Final adjustments sharpen, fix chromatic aberrations, hue adjust, etc.

<table>
<thead>
<tr>
<th>Step</th>
<th>Resolution</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Correct pixel defects</td>
<td>(10-12 bits per pixel)</td>
<td></td>
</tr>
<tr>
<td>2. Align and merge</td>
<td>1 intensity value per pixel</td>
<td></td>
</tr>
<tr>
<td>3. Correct for sensor bias</td>
<td>Pixel values linear in energy</td>
<td></td>
</tr>
<tr>
<td>4. Vignetting compensation</td>
<td>3x10 bits per pixel</td>
<td></td>
</tr>
<tr>
<td>5. White balance</td>
<td>RGB intensity per pixel</td>
<td></td>
</tr>
<tr>
<td>6. Demosaic</td>
<td>Pixel values linear in energy</td>
<td></td>
</tr>
<tr>
<td>7. Denoise</td>
<td>3x8-bits per pixel</td>
<td></td>
</tr>
<tr>
<td>8. Gamma Correction (non-linear mapping)</td>
<td>Pixel values perceptually linear</td>
<td></td>
</tr>
<tr>
<td>9. Local tone mapping</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10. Final adjustments</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Today*
Auto Focus
What does a lens do?

Recall: pinhole camera you may have made in science class (every pixel measures ray of light passing through pinhole and arriving at pixel)
What does a lens do?

Camera with lens:

Every pixel accumulates all rays of light passing through lens aperture and refracted to location of pixel.

In-focus camera: all rays of light from one point in scene arrive at one point on sensor plane.
Out of focus camera

Out of focus camera: rays of light from one point in scene do not converge at point on sensor

Stanford CS348K, Spring 2020
Bokeh
Out of focus camera

Out of focus camera: rays of light from one point in scene do not converge at point on sensor

Rays of light from different scene points converge at single point on sensor
Sharp foreground / blurry background
AutoFocus demos

- Phase-detection auto focus
  - Common in SLRs

- Contrast-detection auto focus
  - Smartphone cameras
Single lens reflex (SLR) camera

Pentaprism

autoexposure (AE)
viewfinder
focusing screen
autofocus (AF)

Image credits: Nikon, Marc Levoy
Split pixel sensor

When both pixels have the same response, camera is in focus, why?

Now two pixels under each microlens (not one)

Image credit: Nikon
What part of image should be in focus?

Heuristics:
- Focus on closest scene region
- Put center of image in focus
- Detect faces and focus on closest/largest face

Image credit: DPReview:
https://www.dpreview.com/articles/9174241280/configuring-your-5d-mark-iii-af-for-fast-action
Portrait mode in modern smartphones

- Smart phone cameras have small apertures
  - Good: thin, lightweight lenses, often fast focus
  - Bad: cannot physically create aesthetically please photographs with nice bokeh, blurred background

- Answer: simulate behavior of large aperture lens (hallucinate image formed by large aperture lens)

![Input image /w detected face](image1)
![Segmentation](image2)
![Scene Depth Estimate](image3)
![Generated image](image4)

Image credit: [Wadha 2018]
Additional sensing modalities

Apple’s TrueDepth camera
(infrared dots projected by phone, captured by infrared camera)
Additional sensing modalities

Fuse information from all modalities to obtain best estimate of depth

Image credit: https://blog.halide.cam/iphone-xr-a-deep-dive-into-depth-47d36ae69a81
Summary
Summary

- Computation now a fundamental part of producing a pleasing photograph
- Used to compensate for physical constraints (demosaic, denoise, lens corrections)
- Used to analyze image to guess system parameters (focus, exposure), or scene contents (white balance, portrait mode)
- Used to make non-physically plausible images that have aesthetic merit
Image processing workload characteristics

- **“Pointwise” operations**
  - output\_pixel = f(input\_pixel)

- **“Stencil” computations (e.g., convolution, demosaic, etc.)**
  - Output pixel (x,y) depends on fixed-size local region of input around (x,y)

- **Lookup tables**
  - e.g., contrast s-curve

- **Multi-resolution operations (upsampling/downsampling)**

- **Fast-fourier transform**
  - We didn’t talk about Fourier domain techniques in class (but Hasinoff 16 reading has many examples)

- **Long pipelines of these operations**

Upcoming classes: efficiently mapping these workloads to modern processors