Lecture 4:

Finishing up the Camera Pipeline + Frankencamera Discussion

Visual Computing Systems
Stanford CS348K, Spring 2023
Picking up from last time . . .
(The HDR part of HDR+)
Saturated pixels
Saturated pixels

Credit: P. Debevec
Global tone mapping

- Measured image values (by camera’s sensor): 10-12 bits / pixel, but common image formats are 8-bits/pixel
- How to convert 12 bit number to 8 bit number?

![Image of church interior with stained glass window]

**Figure 7:** The sun is directly behind the rightmost stained glass window, making it especially bright. The blue borders seen in some of the image margins are induced by the image registration process.

**Figure 6:** Sixteen photographs of a church taken at 1-stop increments from 30 sec to 0.75 sec. The response curves for the imaging system used in the church photographs in Fig. 8. From the SIGGRAPH'97 Conference Proceedings, August 1997

![Response curves for red, green, and blue channels, plotted with the underlying pixel value]

**Response curves** for red, green, and blue plotted on the same axes. Note that while the red and green curves are very consistent, the blue curve rises significantly above the others for low exposure values. This indicates that dark regions in the images exhibit a slight blue cast. Since this artifact is recovered by the

- How to convert 12 bit number to 8 bit number?

- Measured image values (by camera’s sensor): 10-12 bits / pixel, but common image formats are 8-bits/pixel

- How to convert 12 bit number to 8 bit number?

![Graph showing how to convert 12 bit to 8 bit]

- How to convert 12 bit number to 8 bit number?

- Measured image values (by camera’s sensor): 10-12 bits / pixel, but common image formats are 8-bits/pixel

- How to convert 12 bit number to 8 bit number?
High dynamic range image (HDR)
Detail in dark and light images
Local tone adjustment

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

Combined image (unique weights per pixel)
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
Image blending

Consider a simple case where we want to blend two patterns:

Problem: not “smooth”
"Feather" the alpha mask

For a "smoother" look...

\[
I_{\text{blend}} = \alpha I_{\text{left}} + (1 - \alpha) I_{\text{right}}
\]
Effect of feather window size

“Ghosting” visible if the feather window (transition) is too large

Slide credit: Efros
Effect of feather window size

Seams visible if the feather window (transition) is too small
What do we want

- To avoid seams, transition window should be $\geq$ size of largest prominent feature
- To avoid ghosting, transition window should be smaller than $\sim 2X$ smallest prominent feature
- In other words, the largest and smallest features need to be within a factor of two for feathering to generate good results

Intuition:
- Coarse structure of images (large features) should transition slowly between images
- Fine structure should blend quickly
Gaussian pyramid

$G_0 = \text{image}$

Each image in pyramid contains increasingly low-pass filtered signal

down() = image downsample operation
Downsample

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

- Step 1: Remove high frequency detail (blur)
- 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;
    }
}
```
Gaussian pyramid

$G_0$
Gaussian pyramid

\( G_1 \)
Gaussian pyramid

$G_2$
Gaussian pyramid

\[ G_3 \]
Gaussian pyramid

$G_4$
Gaussian pyramid

$G_5$
Upsample

Via bilinear interpolation of samples from low resolution image
Upsample

Via bilinear interpolation of samples from low resolution image

```c
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];
    }
}
```
Laplacian pyramid

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

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

down() = image downsample operation
up() = image upsample operation

[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

\[ L_4 = G_4 - \text{up}(G_5) \]
Laplacian pyramid

$L_5 = G_5$
Gaussian/Laplacian pyramid 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 local tone mapping

- Build Gaussian pyramid of weight map
- Merge laplacian pyramids (image features) not image pixels according to weight pyramid
- “Flatten” merged laplacian pyramid to get final image
Merging Laplacian pyramids

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?
### 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
  - (10-12 bits per pixel)
  - 1 intensity value per pixel
  - Pixel values linear in energy
- Demosaic
  - 3x10 bits per pixel
  - RGB intensity per pixel
  - Pixel values linear in energy
- Denoise
- Gamma Correction (non-linear mapping)
  - 3x8-bits per pixel
  - Pixel values *perceptually* linear
- Local tone mapping
- Final adjustments sharpen, fix chromatic aberrations, hue adjust, etc.
Frankencamera
(Discussion)
Choosing the “right” representation for the job

* Good representations are productive to use:
  - They embody the natural way of thinking about a problem

* Good representations enable the system to provide the application developer useful services:
  - Validating/providing certain guarantees (correctness, resource bounds, conversation of quantities, type checking)
  - Performance optimizations (parallelization, vectorization, use of specialized hardware)
  - Implementations of common, difficult-to-implement functionality (texture mapping and rasterization in 3D graphics, auto-differentiation in ML frameworks)
Frankencamera: some 2010 context

- Cameras were becoming increasingly cheap and ubiquitous
- Cameras featured increasing processing capability
- Significant graphics research focus on developing techniques for combining multiple photos to overcome deficiencies of traditional camera systems
Multi-shot photography example: high dynamic range (HDR) images

Source photographs: each photograph has different exposure

Tone mapped HDR image

Credit: Debevec and Malik
More multi-shot photography examples

“Lucky” imaging
Take several photos in rapid succession: likely to find one without camera shake

Flash-no-flash photography [Eisemann and Durand]
(use flash image for sharp, colored image, infer room lighting from no-flash image)
More multi-shot photography examples

Panorama capture

individual images

extended dynamic range panorama
Frankencamera: some 2010 context

- Cameras were becoming increasingly cheap and ubiquitous
- Cameras featured increasing processing capability
- Significant graphics research focus on developing techniques for combining multiple photos to overcome deficiencies of traditional camera systems

Problem: the ability to implement multi-shot techniques on cameras was limited by camera system programming abstractions

- Programmable interface to camera was very basic
- Echoed physical button interface to a point-and-shoot camera:
  - takePhotograph(parameters, output_jpg_buffer)
- Result: on most camera implementations, latency between two photos was high, mitigating utility of multi-shot techniques (large scene movement or camera shake between shots)
Frankencamera (F-cam) goals

1. Create open, handheld computational camera platform for researchers

2. Define system architecture for computational photography applications
   - Motivated by impact of OpenGL on graphics application and graphics hardware development (portable apps despite highly optimized GPU implementations)
   - Motivated by proliferation of smart-phone apps

**F2 Reference Implementation**

Note: Apple was not involved in Frankencamera’s industrial design. ;-)
F-cam scope

- F-cam provides a set of abstractions that allow for manipulating configurable camera components
  - Timeline-based specification of actions
  - Feed-forward system: no feedback loops

- F-cam architecture performs image processing, but...
  - This functionality as presented by the architecture is not programmable
  - Hence, F-cam does not provide an image processing language (it’s like fixed-function OpenGL)
  - Other than work performed by the image processing stage, F-cam applications perform their own image processing (e.g., on smartphone/camera’s CPU or GPU resources)
Android Camera2 API

- Take a look at the documentation of the Android Camera2 API, and you’ll see influence of F-Cam.
Modern smartphone cameras perform advanced image analysis functions

Image analysis examples from prior lectures:
auto white balance, auto exposure, image denoising
Auto Focus
Pinhole camera (no lens)

- Sensor plane: \((X,Y)\)
- Pixel P1
- Pixel P2
- Pinhole
- Scene object 1
- Scene object 2
What does a lens do?

A lens refracts light.

Camera with lens: every pixel accumulates all rays of light that pass through lens aperture and refract toward that pixel.

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

Out of focus camera: rays of light from one point in scene do not converge to the same point on the sensor.

Diagram showing:
- Scene object 1
- Scene object 2
- Sensor plane (X,Y)
- Lens
- Circle of confusion
- Previous sensor plane location
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)
Bokeh
Sharp foreground, defocused background

Common technique to emphasize subject in a photo
Cell phone camera lens(es) (small aperture)
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)

(a) Input image with detected face

(b) Person segmentation mask

(c) Mask + disparity from DP

(d) Our output synthetic shallow depth-of-field image

Generated image (note blurred background. Blur increases with depth)
What part of image should be in focus?

Consider possible 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
Split pixel sensor

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

Now two pixels under each microlens (not one)
Estimating depth

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

iPhone Xr depth estimate with lights ON in room

iPhone Xr depth estimate with lights OFF in room (No help from RGB)

Image credit: https://blog.halide.cam/iphone-xr-a-deep-dive-into-depth-47d36ae69a81
Magic eraser
(Feature in recent Google Pixel phones)
Summary
Summary

- Computation now a fundamental part of producing a pleasing photograph
- Used to compensate for physical constraints (demosaic, denoise, lens corrections, portrait mode)
- Used to analyze image to estimate system parameters (autofocus, autoexposure, white balance, depth estimation)
- 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 transforms
  - We didn’t talk about Fourier domain techniques in class (but Hasinoff 16 reading has many examples)

- Long pipelines of these operations

Next class: efficiently mapping these workloads to modern processors
Abstractions for authoring image processing pipelines
Choosing the “right” representation for the job (again)

- This was the theme of our Frankencamera discussion

- Good representations are productive to use:
  - They embody the natural way of thinking about a problem

- Good representations enable the system to provide the application developer useful services:
  - Validating/providing certain guarantees (correctness, resource bounds, conversation of quantities, type checking)
  - Performance optimizations (parallelization, vectorization, use of specialized hardware)
  - Implementations of common, difficult-to-implement functionality (texture mapping and rasterization in 3D graphics, auto-differentiation in ML frameworks)
Consider a single task: sharpen an image

Example: sharpen an image

\[
F = \begin{bmatrix}
0 & -1 & 0 \\
-1 & 5 & -1 \\
0 & -1 & 0
\end{bmatrix}
\]

Input → Output
Four different representations of sharpen

1. Image input;
   Image output = sharpen(input);

2. F = 
   \[
   \begin{bmatrix}
   0 & -1 & 0 \\
   -1 & 5 & -1 \\
   0 & -1 & 0
   \end{bmatrix}
   \]

3. Image input;
   Image output = convolve(input, F);

4. float input[(WIDTH+2) * (HEIGHT+2)];
   float output[WIDTH * HEIGHT];

   float weights[] = {0., -1., 0., -1., 5, -1., 0., -1., 0.};

   for (int j=0; j<HEIGHT; j++) {
     for (int i=0; i<WIDTH; i++) {
       float tmp = 0.f;
       for (int jj=0; jj<3; jj++)
         for (int ii=0; ii<3; ii++)
           tmp += input[(j+jj)*(WIDTH+2) + (i+ii)] * weights[jj*3 + ii];
       output[j*WIDTH + i] = tmp;
     }
   }

Image input;
Image output = sharpen(input);

Image input;
Image output = convolve(input, F);

output[i][j] = F[0][0] * input[i-1][j-1] +
F[0][1] * input[i-1][j] +
F[0][2] * input[i-1][j+1] +
F[1][0] * input[i][j-1] +
F[1][1] * input[i][j] +
F[1][2] * input[i][j+1] +
...
Image processing tasks from previous lectures

**Sobel Edge Detection**

\[
G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \times I
\]

\[
G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \times I
\]

\[
G = \sqrt{G_x^2 + G_y^2}
\]

**3x3 Gaussian blur**

\[
F = \begin{bmatrix} .075 & .124 & .075 \\ .124 & .204 & .124 \\ .075 & .124 & .075 \end{bmatrix}
\]

**2x2 downsampling (via averaging)**

\[
\text{output}[x][y] = \frac{\text{input}[2x][2y] + \text{input}[2x+1][2y] + \text{input}[2x][2y+1] + \text{input}[2x+1][2y+1]}{4.0f};
\]

**Gamma Correction**

\[
\text{output}[x][y] = \text{pow}(\text{input}[x][y], 0.5f);
\]

**LUT-based correction**

\[
\text{output}[x][y] = \text{lookup_table}[\text{input}[x][y]];
\]

**Local Pixel Clamp**

```
float f(image input) {
    float min_value = min( min(input[x-1][y], input[x+1][y]),
                          min(input[x][y-1], input[x][y+1]));
    float max_value = max( max(input[x-1][y], input[x+1][y]),
                          max(input[x][y-1], input[x][y+1]));
    output[x][y] = clamp(min_value, max_value, input[x][y]);
    output[x][y] = f(input);
}
```

**Histogram**

```
bin[input[x][y]]++;
```
New goals (setting up for next class)

- Be expressive: facilitate intuitive expression of a broad class of image processing applications
  - e.g., all the components of a modern camera RAW pipeline

- Be high performance: want to generate code that efficiently utilizes the multi-core and SIMD processing resources of modern CPUs and GPUs, and is memory bandwidth efficient