Lecture 4:

Frankencamera +
Finishing up the Camera Pipeline

Visual Computing Systems
Stanford CS348K, Spring 2022
Local tone adjustment

- **Pixel values**
- **Weights**

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

**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
Gaussian pyramid

$G_0 =$ image

Each image in pyramid contains increasingly low-pass filtered signal

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

$G_1$
Gaussian pyramid

$G_2$
Gaussian pyramid

$G_3$
Gaussian pyramid

$G_4$
Gaussian pyramid

$G_5$
Laplacian pyramid

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

$L_0 = G_0 - up(G_1)$

[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

\[ L_3 = G_3 - \up(G_4) \]
Laplacian pyramid

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

\[ L_S = G_S \]
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</th>
<th>High Exposure</th>
<th>Weight (for Low Exposure)</th>
<th>Merged (after flatten)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laplacian Pyramid</td>
<td>Laplacian Pyramid</td>
<td>Gaussian Pyramid</td>
<td></td>
</tr>
<tr>
<td>clipped</td>
<td>clipped</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```
L0
L1
L2
L3
G4
```

```
L0
L1
L2
L3
G4
```

```
G0
G1
G2
G3
G4
```

edge magnitude reduced, but detail remains on both sides
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, conservation 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: becoming increasingly cheap and ubiquitous
- Significant processing capability available on cameras
- Many techniques for combining multiple photos to overcome deficiencies of traditional camera systems
Multi-shot photography example: high dynamic range (HDR) images
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 are cheap and ubiquitous
- Significant processing capability available on cameras
- Many emerging techniques for combining multiple photos to overcome deficiencies in 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
- Influenced by physical button interface to a point-and-shoot camera:
  - `take_photograph(parameters, output_jpg_buffer)`
- Result: on most implementations, latency between two photos was high, mitigating utility of multi-shot techniques (large scene movement, camera shake, between shots)
Frankencamera 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. ;-)

Nokia N900 Smartphone Implementation

[Adams et al. 2010]
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.
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.
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)

Input image /w detected face

(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?

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)
Additional sensing devices and 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
  - \( \text{output\_pixel} = f(\text{input\_pixel}) \)

- "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
Abstractions for authoring image processing pipelines
Reminder: choosing the “right” representation for the job

- 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, conservation 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)
Goals

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

- 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
What does this code do?

```c
void mystery (const Image &in, Image &output) {
    _m128i one_third = _mm_set1_epi16(21846);
    #pragma omp parallel for
    for (int yTile = 0; yTile < in.height(); yTile += 32) {
        _m128i a, b, c, sum, avg;
        _m128i tmp[(256/8)*(32+2)];
        for (int xTile = 0; xTile < in.width(); xTile += 256) {
            _m128i *tmpPtr = tmp;
            for (int y = -1; y < 32+1; y++) {
                const uint16_t *inPtr = &(in(xTile, yTile+y));
                for (int x = 0; x < 256; x += 8) {
                    a = _mm_loadu_si128((._m128i*)(inPtr-1));
                    b = _mm_loadu_si128((._m128i*)(inPtr+1));
                    c = _mm_load_si128((._m128i*)(inPtr));
                    sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
                    avg = _mm_mulhi_epi16(sum, one_third);
                    _mm_store_si128(tmpPtr++, avg);
                    inPtr += 8;
                }
            }
            tmpPtr = tmp;
            for (int y = 0; y < 32; y++) {
                _m128i *outPtr = (._m128i *)&(output(xTile, yTile+y));
                for (int x = 0; x < 256; x += 8) {
                    a = _mm_load_si128(tmpPtr+(2*256)/8);
                    b = _mm_load_si128(tmpPtr+256/8);
                    c = _mm_load_si128(tmpPtr++);
                    sum = _mm_add_epi16(_mm_add_epi16(a, b), c);
                    avg = _mm_mulhi_epi16(sum, one_third);
                    _mm_store_si128(outPtr++, avg);
                }
            }
        }
    }
}
```

I'll tell you next class.
Consider a single task: sharpen an image

Example: sharpen an image

\[ F = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \]
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] = $F[0][0] \times \text{input}[i-1][j-1] + F[0][1] \times \text{input}[i-1][j] + F[0][2] \times \text{input}[i-1][j+1] + F[1][0] \times \text{input}[i][j-1] + F[1][1] \times \text{input}[i][j] + \ldots$

Image input;
Image output = sharpen(input);

Image input;
Image output = convolve(input, F);
Image processing tasks from previous lectures

**Sobel Edge Detection**

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

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

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

**Local Pixel Clamp**

```plaintext
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);
}
```

**3x3 Gaussian blur**

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

**2x2 downsample (via averaging)**

\[ \text{output}[x][y] = (\text{input}[2x][2y] + \text{input}[2x+1][2y] + \text{input}[2x][2y+1] + \text{input}[2x+1][2y+1]) / 4.f; \]

**Gamma Correction**

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

**LUT-based correction**

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

**Histogram**

\[ \text{bin}[[\text{input}[x][y]]]++; \]

**Stanford CS348K, Spring 2022**