Lecture 16:

Image Processing

Algorithm Grab Bag

Visual Computing Systems
CMU 15-869, Fall 2013
Today

- Grab bag of image processing techniques relevant to computational photography

- High level description of algorithms to help you build intuition (just scratching the surface of concepts and results from field of image processing)

- At the end of class:
  - We’ll discuss how we might design an efficient image processor for these types of workloads
Review: 2D convolution with 5x5 filter

```c
int WIDTH = 1024;
int HEIGHT = 1024;
uint8 input[(WIDTH+2) * (HEIGHT+2)];
uint8 output[WIDTH * HEIGHT];
uint8 weights[] = {1, 1, 1, 1, 1,
  2, 2, 2, 2, 2,
  3, 3, 3, 3, 3,
  2, 2, 2, 2, 2,
  1, 1, 1, 1, 1};

for (int j=0; j<HEIGHT; j++) {
  for (int i=0; i<WIDTH; i++) {
    int tmp = 0.f;
    for (int jj=0; jj<5; jj++)
      for (int ii=0; ii<5; ii++)
        tmp += (int)input[(j+jj)*(WIDTH+2) + (i+ii)] * weights[jj*5 + ii];
    output[j*WIDTH + i] = uint8(tmp / 25);
  }
}
```

Recall:
Total work = $25 \times WIDTH \times HEIGHT$
For $N \times N$ filter: $N^2 \times WIDTH \times HEIGHT$
2D convolution with 5x5 filter

```
int WIDTH = 1024;
int HEIGHT = 1024;
uint8 input[(WIDTH+2) * (HEIGHT+2)];
uint8 output[WIDTH * HEIGHT];
uint8 weights[] = {1, 1, 1, 1, 1,
  2, 2, 2, 2, 2,
  3, 3, 3, 3, 3,
  2, 2, 2, 2, 2,
  1, 1, 1, 1, 1};

for (int j=0; j<HEIGHT; j++) {
  int tmp = 0;
  for (int jj=0; jj<5; jj++)
    for (int ii=0; ii<5; ii++)
      tmp += (int)input[(j+jj)*(WIDTH+2) + ii] * weights[jj*5 + ii];
  output[j*WIDTH] = uint8(tmp);
}

for (int i=1; i<WIDTH; i++) {
  int tmp1=0, tmp2=0;
  for (int jj=0; jj<5; jj++) {
    tmp1 = (int)input[(j+jj)*(WIDTH+2) + i+4] * weights[jj*5 + 4];
    tmp2 = (int)input[(j+jj)*(WIDTH+2) + i] * weights[jj*5 + 0];
  }
  output[j*WIDTH + i] = output[j*WIDTH + i - 1] + uint8(tmp1-tmp2);
}
```

Incremental computation:

~ Total work = 2*N x WIDTH x HEIGHT

Filter is separable, so same work complexity as two-pass approach, but using only one pass over the data.

At what cost?
5x5 median filter

- Noise reduction filter
  - Unlike gaussian, one bright pixel doesn’t drag up the average for entire region
- Not linear, not separable
  - Filter weights are 1 or 0 (depending on image content)
- Naive algorithm for width N square kernel support region:
  - Sort $N^2$ elements in support region, pick median: $O(N^2 \log(N^2))$ work per pixel

```c
int WIDTH = 1024;
int HEIGHT = 1024;
uint8 input[(WIDTH+2) * (HEIGHT+2)];
uint8 output[WIDTH * HEIGHT];

for (int j=0; j<HEIGHT; j++) {
    for (int i=0; i<WIDTH; i++) {
        output[j*WIDTH + i] = // median of pixels in surrounding 5x5 pixel window
    }
}
```
5x5 median filter

- O(N^2) work-per-pixel solution: radix sort algorithm for 8 bit-integer data
  - Bin elements in support region. Scan histogram to find median

```c
int WIDTH = 1024;
int HEIGHT = 1024;
uint8 input[(WIDTH+2) * (HEIGHT+2)];
uint8 output[WIDTH * HEIGHT];
int histogram[256];

for (int j=0; j<HEIGHT; j++) {
    for (int i=0; i<WIDTH; i++) {
        for (int ii=0; ii<256; ii++)
            histogram[ii] = 0;
        for (int jj=0; jj<5; jj++)
            for (int ii=0; ii<5; ii++)
                histogram[input[(j+jj)*(WIDTH+2) + (i+ii)]]++;
        int count = 0;
        for (int ii=0; ii<256; i++) {
            if (count + histogram[i] >= 13) // median of 25 elements is bin containing 13th value
                output[j*WIDTH + i] = uint8(i);
            count += histogram[i];
        }
    }
}
```

Can you design a O(N) work-per-pixel median filter?

See Weiss [SIGGRAPH 2006] for O(lg N) work-per-pixel median filter
Bilateral filter

\[ BF[I](p) = \sum_{q \in S} f(|I_p - I_q|)G_{\sigma}(|p - q|)I(q) \]

Output pixel \( p \) is the weighted sum of all pixels in the support region \( S \) of a truncated gaussian kernel (width \( \sigma \))

But weight is combination of \textit{spatial distance} and \textit{input image pixel intensity} difference. (like median filter, filter weights depend on image content)

- Non-linear filter
- An “edge preserving” filter: down weight contribution of pixels on the other side of strong edges. \( f(x) \) defines what “strong edge means”
- Spatial distance weight term \( f(x) \) could be a gaussian
  - Or very simple: \( f(x) = 0 \) if \( x \) > threshold, 1 otherwise
Bilateral filter

- Non-linear, edge preserving, smoothing filter

Pixels with significantly different intensity contribute little to filtered result

Figure credit: Durand and Dorsey, “Fast Bilateral Filtering for the Display of High-Dynamic-Range Images”, SIGGRAPH 2002
Bilateral filter: kernel depends on image content

See Paris et al. [ECCV 2006] for a fast approximation to the bilateral filter

Question: describe a type of edges the bilateral filter will not respect (it will blur across).
Denoising using non-local means

- **Main idea:** replace pixel with average value of nearby pixels that have a similar surrounding region.

- Prior: images have repeating texture

\[
NL[I](p) = \sum_{q \in S} w(p, q) I(q)
\]

\[
w(p, q) = \frac{1}{C_p} e^{-\frac{||N_p - N_q||^2}{2h^2}}
\]

*N_p and *P_q are vectors of pixel values in square window around pixels *p* and *q*. (Difference of these vectors = “similarity” of surrounding regions)

*C_p* is just a normalization constant to ensure weights sum to one for pixel *p*. 
Non-local means

- Large weight for pixels that have similar neighborhood
  - “Take the average of pixels “like” this one”
  - In example below-right: \( q_1 \) and \( q_2 \) have high weight, \( q_3 \) has low weight

In each pair below:
- Image at left shows pixel to denoise.
- Image at right shows weights of pixels in 21x21-pixel kernel support window.

Buades et al. CVPR 2005
Optical flow

- Goal: determine 2D screen-space velocity of visible objects in image
Optical flow

- Given image A (at time $t$) and image B (at time $t + \Delta t$) compute optical flow between the two images
- Major assumption 1: brightness constancy
  - The appearance of point in image A is same as same point in image B

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) \quad \text{The point observed at (x,y) at time t moves to (x+\Delta, y+\Delta) at t+\Delta t}$$

Tailor expansion

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + I_x(x, y, t)\Delta x + I_y(x, y, t)\Delta y + I_t(x, y, t)\Delta t + \text{higher order terms}$$

So...

$$I(x, y, t) \approx I(x, y, t) + I_x(x, y, t)\Delta x + I_y(x, y, t)\Delta y + I_t(x, y, t)\Delta t$$

$$[I_x(x, y, t)\Delta x + I_y(x, y, t)\Delta y + I_t(x, y, t)\Delta t] = 0 \quad \text{The observed change in pixel (x,y) Is due to object motion at point by (\Delta x, \Delta y)}$$
Problem: underconstrained

Gradient-constraint equation is insufficient to solve for motion

One equation, two unknowns: $(\Delta x, \Delta y)$

\[ I_x(x, y, t) \Delta x + I_y(x, y, t) \Delta y + I_t(x, y, t) \Delta t = 0 \]

Known: observed change in pixel $(x, y)$

Known: image gradients in image A

Major assumption 2: nearby pixels have similar motion (Lucas-Kanade)

\[ I_x(x_0, y_0, t) \Delta x + I_y(x_0, y_0, t) \Delta y + I_t(x_0, y_0, t) \Delta t = 0 \]
\[ I_x(x_1, y_1, t) \Delta x + I_y(x_1, y_1, t) \Delta y + I_t(x_1, y_1, t) \Delta t = 0 \]
\[ I_x(x_2, y_2, t) \Delta x + I_y(x_2, y_2, t) \Delta y + I_t(x_2, y_2, t) \Delta t = 0 \]
\[ \vdots \]

Now overconstrained system, compute least squares solution
Least-squares solution

\[ I_x(x_0, y_0, t) \Delta x + I_y(x_0, y_0, t) \Delta y + I_t(x_0, y_0, t) \Delta t = 0 \]
\[ I_x(x_1, y_1, t) \Delta x + I_y(x_1, y_1, t) \Delta y + I_t(x_1, y_1, t) \Delta t = 0 \]
\[ I_x(x_2, y_2, t) \Delta x + I_y(x_2, y_2, t) \Delta y + I_t(x_2, y_2, t) \Delta t = 0 \]
\[ \vdots \]

Now overconstrained system, compute least squares solution by minimizing:

\((x_i, y_i)\) are pixels in region around \((x, y)\).

Weighting function \(w()\) weights error contribution based on distance between \((x_i, y_i)\) and \((x, y)\). e.g., Gaussian fall-off.

\[ E(\Delta x, \Delta y) = \sum_{x_i, y_i} w(x_i, y_i, x, y) \left[ I_x(x_i, y_i, t) \Delta x + I_y(x_i, y_i, t) \Delta y + I_t(x_i, y_i, t) \Delta t \right]^2 \]
Solving for motion

$E(\Delta x, \Delta y)$ minimized when derivatives are zero:

$$\frac{dE(\Delta x, \Delta y)}{d(\Delta x)} = \sum_{x_i, y_i} w(x_i, y_i, x, y)\left[I_x^2 \Delta x + I_x I_y \Delta y + I_x I_t\right] = 0$$

$$\frac{dE(\Delta x, \Delta y)}{d(\Delta y)} = \sum_{x_i, y_i} w(x_i, y_i, x, y)\left[I_y^2 \Delta y + I_x I_y \Delta x + I_y I_t\right] = 0$$

Rewrite, now solve the following linear system for $\Delta x, \Delta y$:

$$\Delta x \sum_{x_i, y_i} w(x_i, y_i, x, y)I_x^2 + \Delta y \sum_{x_i, y_i} w(x_i, y_i, x, y)I_x I_y + \sum_{x_i, y_i} w(x_i, y_i, x, y)I_x I_t = 0$$

$$\Delta x \sum_{x_i, y_i} w(x_i, y_i, x, y)I_x I_y + \Delta y \sum_{x_i, y_i} w(x_i, y_i, x, y)I_y^2 + \sum_{x_i, y_i} w(x_i, y_i, x, y)I_y I_t = 0$$

Precompute partial derivatives $I_x, I_y, I_t$ from original images A and B

For each pixel $(x, y)$: evaluate $A0, B0, C0, A1, B1, C1$, then solve for $(\Delta x, \Delta y)$ at $(x, y)$
Optical flow, implemented in practice

Gradient-constraint equation makes a linear motion assumption

\[ I(x, y, t) \approx I(x, y, t) + I_x(x, y, t)\Delta x + I_y(x, y, t)\Delta y + I_t(x, y, t)\Delta t \]

\[ I_x(x, y, t)\Delta x + I_y(x, y, t)\Delta y + I_t(x, y, t)\Delta t = 0 \]

The observed change in pixel \((x,y)\) is due to object motion at point by \((\Delta x, \Delta y)\)

- Improvement: iterative techniques use this original flow field to compute higher order residuals (non-linear motion)

- Question: Why is it important for optical flow implementation to be very efficient?
  - Hint: consider linear-motion assumption, consider aliasing
Imagine the your final project is to architect a processor to handle image processing tasks for the widely anticipated kPhone. (like the iPhone, but better)

How would you characterize image processing workloads?
- Parallelism?
- Data-access patterns?
- Predictability? (of data access, of instruction stream)

What are good characteristics of a processor for image processing tasks?
- Programmable, or fixed-function?
  - If programmable, do we need: branch-prediction? out-of-order execution?
  - If fixed-function, in what ways can it be configured?
- What forms of parallelism? (SIMD, multi-core)
- Support for multi-threading, prefetching?
- Data caches or on-chip buffers/scratchpads?
Readings