Lecture 5:

Image Processing

Algorithm Grab Bag

Visual Computing Systems
CMU 15-769, Fall 2016
Median filter

- Noise reduction filter
  - Unlike gaussian blur, one bright pixel doesn’t drag up the average

- 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
5x5 median filter

- O(N^2) work-per-pixel solution for 8-bit pixel data (radix sort 8 bit-integer data)
  - Bin all pixels in support region, then scan histogram bins 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++) {
    // construct histogram of support region
    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)]]++;

    // scan the 256 bins to find median
    // median value of 5x5=25 elements is bin containing 13th value
    int count = 0;
    for (int ii=0; ii<256; ii++) {
      if (count + histogram[ii] >= 13)
        output[j*WIDTH + i] = uint8(ii);
      count += histogram[ii];
    }
  }
}
```

See Weiss [SIGGRAPH 2006] for O(lg N) work-per-pixel median filter
(incrementally updates histogram)
Denoising using non-local means

- Main assumption: images have repeating texture
- Main idea: replace pixel with average value of nearby pixels that have a similar surrounding region

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

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

- \(N_p\) and \(N_q\) are vectors of pixel values in square window around pixels \(p\) and \(q\) (highlighted regions in figure)
- Difference between \(N_p\) and \(P_q\) = “similarity” of surrounding regions (here: L2 distance)
- \(C_p\) is a normalization constant to ensure weights sum to one for pixel \(p\).
- \(S\) is the search region (given by dotted red line in figure)
Denoising using non-local means

- Large weight for input pixels that have similar neighborhood as $p$
  - Intuition: “filtered result is 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 image pair above:
- Image at left shows the pixel to denoise.
- Image at right shows weights of pixels in 21x21-pixel kernel support window.

Buades et al. CVPR 2005
Recall from last week: bilateral filter

\[
BF[I](p) = \frac{1}{W_p} \sum_{q \in S} f(|I_p - I_q|)G_\sigma(||p-q||)I(q)
\]

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

Recall key property of a separable filter: can we decomposed into product of two (cheap-to-compute) 1D filters
- 2D gaussian is a separable filter

Bilateral filter is non-separable: execution has high cost \((S^2)\) when used with large support region \(S\) (large \(\sigma\))
Implementation via the bilateral grid

Consider image as surface embedded in 3D space
$X,Y$ corresponds to position in image place
$Z$ corresponds to image pixel value at $(x,y)$

Bilateral grid stores image values in homogeneous form:
for all pixels $(x,y)$ in image $I(x,y)$: $BG(x,y, I(x,y)) += (I(x,y), 1)$
So... $BG(x,y, I(x,y)) = (wl(x,y), w)$
Implementation via the bilateral grid

Consider effect of applying 3D Gaussian filter to contents of 3D bilateral grid. 
(Note: only 2D grid shown)

Bilateral grid stores image values in homogeneous form:
for all pixels \((x,y)\) in image \(I(x,y)\): \(BG(x,y, I(x,y)) += (I(x,y), 1)\)
So... \(BG(x,y, I(x,y)) = (wl(x,y), w)\)
Implementation via the bilateral grid

Consider effect of applying 3D Gaussian filter to contents of 3D bilateral grid. (Note: only 2D grid shown)

Notice how spatially adjacent (in x,y) values on the other side of the edge do not effect result of filtering via gaussian centered at position 5.
Implementation via the bilateral grid

“Slicing” step to recover final image (homogeneous divide)

\[ BF[I](x,y) = \frac{BG(x,y,I(x,y).value)}{BG(x,y,I(x,y).w)} \]
Bilateral grid notes

- High performance comes from using lower resolution grid (spatial dimensions) than dimensionality of source image
  - Applicable when filter support region is large (okay to pre-filter signal prior to convolving with wide Gaussian)
  - For small support region (e.g., for denoising) direct evaluation of bilateral filter is likely more efficient

- Bilateral filter using RGB distances represented as a 5D grid
  - Higher memory footprint and higher cost
Estimating Motion Using Optical flow
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 the per-pixel motion needed to correspond the two images.

- Major assumption 1: “brightness constancy”
  - The appearance of a scene surface point that is visible in both images A and B is the same in both images.

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$$

Taylor 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...

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

The observed change in pixel $(x, y)$

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

Is due to object motion at point by $(\Delta x, \Delta y)$
$I_y(x, y, t)$

$\rightarrow$

$I_x(x, y, t)$
Gradient-constraint equation for a pixel is 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 \((x,y)\) over consecutive frames
Known: spatial image gradients in image A

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

\[
I_x(x_0, y_0, t)\Delta x + I_y(x_0, y_0, t)\Delta y + I_t(x_0, y_0, t)\Delta t = 0
\]

\[
I_x(x_1, y_1, t)\Delta x + I_y(x_1, y_1, t)\Delta y + I_t(x_1, y_1, t)\Delta t = 0
\]

\[
I_x(x_2, y_2, t)\Delta x + I_y(x_2, y_2, t)\Delta y + I_t(x_2, y_2, t)\Delta t = 0
\]

\[
\cdots
\]

Now we have a overconstrained system, compute least squares solution
Weighted least-squares solution

Gradient-constraint equation is insufficient to solve for motion

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

\[
I_x(x_0, y_0, t)\Delta x + I_y(x_0, y_0, t)\Delta y + I_t(x_0, y_0, t)\Delta t = 0
\]

\[
I_x(x_1, y_1, t)\Delta x + I_y(x_1, y_1, t)\Delta y + I_t(x_1, y_1, t)\Delta t = 0
\]

\[
I_x(x_2, y_2, t)\Delta x + I_y(x_2, y_2, t)\Delta y + I_t(x_2, y_2, t)\Delta t = 0
\]

\[
\vdots
\]

Compute weighted least squares solution by minimizing:

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

\((x_i, y_i)\) 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.
Solving for motion

E (Δx, Δy) 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 Δx, Δy:

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

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

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 (Δx, Δy) at (x, y)
Optical flow, implemented in practice

Gradient-constraint equation makes a linear motion assumption

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

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

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

- Question: why is it important for optical flow implementation to be very efficient?
  - Hint: consider linear-motion assumption