Lecture 14: Image Processing

Rui Wang
Topics

• Histogram
• Image Convolution
  • 1D and 2D convolution
  • Implementation using CUDA
  • Separable convolution
  • Edge-preserving image denoising
  • Median filtering
  • Bilateral filtering
• FFT (Fast Fourier Transform) - Later
Image Histogram

• Compute the number of pixels with a particular brightness/intensity value.
  - Usually only 256 values for an 8-bit image.
Image Histogram

- Histogram is very useful:
- Histogram Equalization

- Matching Images
  - A histogram is a small signature of the image.
Image Histogram

- Implementation: straightforward to use atomic add to compute histogram.
  - CUDA SDK has an example that simulates atomic add for a warp (32 threads) for CC1.0
Image Convolution (Filtering)

- Start with 1D “Moving Averages”:

\[ I'(i) = \sum_{k=-r}^{r} \frac{1}{2r+1} I(i+k) \]
1D Convolution

• Start with 1D “Moving Averages”:

\[ I'(i) = \sum_{k=-r}^{r} \frac{1}{2r+1} I(i+k) \]

\[ I'(i) = \sum_{k=-r}^{r} c_k I(i+k) \]

\[ \left( \sum_{k=-r}^{r} c_k = 1 \right) \]
1D Convolution

- Convolution Filters (or Kernels)
  - Average Filter (box filter):
    \[ c_k = \frac{1}{2r + 1}, \quad -r \leq k \leq r \]
1D Convolution

- Convolution Filters (or Kernels)
  - Average Filter (box filter):
    \[ c_k = \frac{1}{2r+1}, \quad -r \leq k \leq r \]
  - Tent Filter:
    - Convolve a box with a box
1D Convolution

• Convolution Filters (or Kernels)
  • Average Filter (box filter):
    
    $$c_k = \frac{1}{2r+1}, \quad -r \leq k \leq r$$

  • Tent Filter:
    − Convolve a box with a box
  • Gaussian Filter:
    
    $$c_k = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{\frac{-k^2}{2\sigma^2}}$$
2D Convolution

- Straightforward to extend 1D convolution to 2D

\[ I'(i, j) = \sum_{m,n} c_{m,n} I(i+m, j+n), \quad \sum_{m,n} c_{m,n} = 1. \]
2D Convolution

● Boundary Problems:
  ● Zero boundary extension
  ● Constant boundary extension
  ● Wrap around extension
  ● Keep track of partial sums and re-normalize.
2D Convolution

- 2D convolution kernels:
  - Box: 
    \[ c_{m,n} = \frac{1}{(2r+1)^2} \]
  - Tent
  - Gaussian: 
    \[ c(m,n) = \frac{1}{2\pi \sigma^2} e^{-\frac{m^2+n^2}{2\sigma^2}} \]
  - ...
2D Convolution

- Implementation using CUDA:
  - Each block is in charge of a tile of pixels (say, 16x16)
  - Each thread is in charge of a pixel
  - Simplest is to access source image as textures
  - Use SMEM to amortize GMEM access cost
  - The “Apron” region
    - SMEM usage
    - Load balancing problem
2D Convolution

- Load Balancing for the “apron” region:
  - Some threads will be idling during loading: this is all right for memory coalescing
  - Some threads will be idling during the computation: takes some overhead but all right if the idling threads are a multiple of warp size
Separable 2D Convolution

• Convolution kernel is the out-product of two 1D kernels:

\[ c(m, n) = c(m) \times c(n) \]

• Many common 2D filters are separable:
  • Box
  • Tent
  • Gaussian
Separable 2D Convolution

• Separable convolution:
  • 1D convolution in row direction, then another 1D convolution in column direction.
Separable 2D Convolution

- Separable convolution:
  - 1D convolution in row direction, then another 1D convolution in column direction.
- Benefits of separable convolution:
  - Reduced complexity: $O(r^2) \to O(r)$
  - Reduced SMEM usage
Nonlinear Filtering

- Linear Filtering
  - Output is a weighted sum of input; weights remain fixed \(\rightarrow\) spatially invariant.
  - Problem with linear filtering:
    - Over-blurring
    - Does not preserve edges
    - Unaware of local changes
- Nonlinear Filtering
Median Filter

- What is Median Filter?
  - Look at 3x3 neighborhood pixels, taken the median (not the mean!)
  - The median minimizes the sum of absolute differences:

\[
\sum_{i} |a_i - x|
\]
Median Filter

- What is Median Filter?
  - Look at 3x3 neighborhood pixels, taken the median (not the mean!)
  - The median minimizes the sum of absolute differences:
    \[ \sum_i |a_i - x| \]
  - In contrast, the mean (average) minimizes the sum of squared differences.
    \[ \sum_i (a_i - x)^2 \]
Median Filter

- What is Median Filter?
  - Look at 3x3 neighborhood pixels, taken the median (not the mean!)
- Median filter is nonlinear
  - Which pixel is the median depends on the local neighborhood
  - \( \text{Md} (R_1, R_2, R_3) \neq \text{Md} (\text{Md}(R_1), \text{Md}(R_2), \text{Md}(R_3)) \)
- Useful for image/signal denoising
  - Blur but preserves sharp edges/discontinuities/changes
Median Filter

- Median filter is one type of \textit{edge-preserving} filter.
Image Denoising

Noisy Image

Gaussian 5.0 pixel

Median 5.0 pixel
Median Filter

• Implementation on the GPU
  • Obtain neighborhood pixels, sort, and take median
  • Cannot do the separable trick anymore
  • Optimization for 3x3 median filter

http://graphics.cs.williams.edu/papers/MedianShaderX6/
Bilateral Filter

- The goal is to modify a standard blur kernel (such as the Gaussian) to be *edge aware*
- Start with a Gaussian
  - Weights only depend on the spatial distance
    - Pixels that are far away get small weights
  - 'Edge aware' means we modify weights based on 'distance' (or similarity) in intensity.
    - Pixels that have dissimilar intensity values get small weights.
Bilateral Filter

- Convolution Kernel:

\[ c(m, n) = G_\sigma(m, n) \times G_h(|I(m, n) - I_i|) \]

- Spatial Weight
- Intensity Weight
Bilateral Filter

- Convolution Kernel:

\[ c(m, n) = G_\sigma(m, n) \times G_h(|I(m, n) - I_i|) \]

- Spatial Weight
- Intensity Weight
Bilateral Filter

- Convolution Kernel:

\[ c(m, n) = G_\sigma(m, n) \times G_h(|I(m, n) - I_i|) \]

Spatial Weight \hspace{1cm} Intensity Weight

Convolve with a spatially varying kernel!
**Bilateral Filter**

- Convolution Kernel:

\[ c(m, n) = G_\sigma(m, n) \times G_h(\|I(m, n) - I_i\|) \]

- Spatial Weight
- Intensity Weight

Result: Sharp edges preserved.
Bilateral Filter

- Example: [Tomasi and Ramduchi 98]
Bilateral Filter

- Implementation on the GPU
  - Direct implementation (inefficient for large kernels)
  - Separable convolution (approximation but not accurate)
  - Discretizing possible intensity values
    - Convert to multiple layers, and interpolate
  - Convert to 3D