Advanced Computer Graphics
CSE 190 [Spring 2015], Lecture 4
Ravi Ramamoorthi
http://www.cs.ucsd.edu/~ravir

To Do
- Assignment 1, Due Apr 24.
  - Please START EARLY
  - This lecture completes all the material you need

Outline
- Implementation of digital filters
  - Discrete convolution in spatial domain
  - Basic image-processing operations
  - Antialiased shift and resize

Discrete Convolution
- Previously: Convolution as mult in freq domain
  - But need to convert digital image to and from to use that
  - Useful in some cases, but not for small filters
- Previously seen: Sinc as ideal low-pass filter
  - But has infinite spatial extent, exhibits spatial ringing
  - In general, use frequency ideas, but consider
    implementation issues as well
- Instead, use simple discrete convolution filters e.g.
  - Pixel gets sum of nearby pixels weighted by filter/mask

Implementing Discrete Convolution
- Fill in each pixel new image convolving with old
  - Not really possible to implement it in place

\[ I_{\text{new}}(a,b) = \sum_{x-a}^{x+a} \sum_{y-b}^{y+b} f(x\!-\!a, y\!-\!b) I_{\text{old}}(x,y) \]
- More efficient for smaller kernels/filters f
- Normalization
  - If you don’t want overall brightness change, entries of filter
    must sum to 1. You may need to normalize by dividing
- Integer arithmetic
  - Simpler and more efficient
  - In general, normalization outside, round to nearest int
Basic Image Processing (Assn 3.4)

- **Blur**
- **Sharpen**
- **Edge Detection**

All implemented using convolution with different filters

### Blurring

- Used for softening appearance
- Convolve with gaussian filter
  - Same as mult. by gaussian in freq. domain, so reduces high-frequency content
  - Greater the spatial width, smaller the Fourier width, more blurring occurs and vice versa

- How to find blurring filter?
Blurring Filter

- In general, for symmetry \( f(u,v) = f(u) f(v) \)
- You might want to have some fun with asymmetric filters
- We will use a Gaussian blur
- Blur width sigma depends on kernel size \( n \) (3, 5, 7, 11, 13, 19)

\[
f(u) = \frac{1}{\sqrt{2\pi} \sigma} \exp\left(-\frac{u^2}{2\sigma^2}\right)
\]
\[
\sigma = \text{floor}(n/2)/2
\]

Discrete Filtering, Normalization

- Gaussian is infinite
  - In practice, finite filter of size \( n \) (much less energy beyond 2 sigma or 3 sigma)
  - Must renormalize so entries add up to 1
- Simple practical approach
  - Take smallest values as 1 to scale others, round to integers
  - Normalize. E.g. for \( n = 3 \), sigma = \( \frac{1}{4} \)

\[
f(u,v) = \frac{1}{2\pi \sigma} \exp\left(-\frac{u^2+v^2}{2\sigma^2}\right)
\]

\[
\begin{bmatrix}
0.012 & 0.09 & 0.012 \\
0.09 & 0.64 & 0.09 \\
0.012 & 0.09 & 0.012
\end{bmatrix}
\]

Basic Image Processing

- Blur
- Sharpen
- Edge Detection

All implemented using convolution with different filters

Sharpening Filter

- Unlike blur, want to accentuate high frequencies
- Take differences with nearby pixels (rather than avg)

\[
f(x,y) = \frac{1}{7} \begin{bmatrix}
-1 & -2 & -1 \\
-2 & 19 & -2 \\
-1 & -2 & -1
\end{bmatrix}
\]
Basic Image Processing

- Blur
- Sharpen
- Edge Detection

All implemented using convolution with different filters

Edge Detection

- Complicated topic: subject of many PhD theses
- Here, we present one approach (Sobel edge detector)
- Step 1: Convolution with gradient (Sobel) filter
  - Edges occur where image gradients are large
    - Separately for horizontal and vertical directions
- Step 2: Magnitude of gradient
  - Norm of horizontal and vertical gradients
- Step 3: Thresholding
  - Threshold to detect edges
Edge Detection

Details

- Step 1: Convolution with gradient (Sobel) filter
  - Edges occur where image gradients are large
  - Separately for horizontal and vertical directions

  \[ f_{\text{horiz}}(x,y) = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad f_{\text{vert}}(x,y) = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \]

- Step 2: Magnitude of gradient
  - Norm of horizontal and vertical gradients

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

- Step 3: Thresholding

Outline

- Implementation of digital filters
  - Discrete convolution in spatial domain
  - Basic image-processing operations
  - Antialiased shift and resize (Assn 3.5, brief)

Antialiased Shift

Shift image based on (fractional) \( s_x \) and \( s_y \)
- Check for integers, treat separately
- Otherwise convolve/resample with kernel/filter \( h \):

\[ u = x - s_x \quad v = y - s_y \]

\[ I(x,y) = \sum_{u',v'} h(u' - u, v' - v) I(u',v') \]

Antialiased Scale Magnification

Magnify image (scale \( s \) or \( \gamma > 1 \))
- Interpolate between org. samples to evaluate frac vals
- Do so by convolving/resampling with kernel/filter
- Treat the two image dimensions independently (diff scales)

\[ u = \frac{x}{\gamma} \]

\[ I(x) = \sum_{u'=\gamma, u'\not\in \text{width}} h(u' - u) I(u') \]

Antialiased Scale Minification

checkerboard.bmp 300x300: point sample checkerboard.bmp 3000x300: Mitchell
Antialiased Scale Minification

Minify (reduce size of) image
- Similar in some ways to mipmapping for texture maps
- We use fat pixels of size $\frac{1}{\gamma}$, with new size $\gamma \times \text{orig size}$ ($\gamma$ is scale factor $< 1$).
- Each fat pixel must integrate over corresponding region in original image using the filter kernel.

$$u = \frac{x}{\gamma}, \quad I(x) = \sum_{u' = u - \epsilon}^{u' = u + \epsilon} \gamma h(u' - u') I(u')$$

### Bonus and Details: Image Warping

- Define transformation
  - Describe the destination $(x, y)$ for every location $(u, v)$ in the source (or vice versa, if invertible)

![Image Warping Diagram]

A note on notation

- This segment uses $(u, v)$ for warped location in the source image (or old coordinates) and $(u', v')$ for integer coordinates, and $(x, y)$ for new coordinates.

- Most of the homework assignment uses $(x, y)$ for old integer coordinates and $(a, b)$ for new coordinates. The warped location is not written explicitly, but is implicit in the evaluation of the filter.

### Example Mappings

- Scale by factor:
  - $x = \text{factor} \times u$
  - $y = \text{factor} \times v$

- Rotate by $\theta$ degrees:
  - $x = u \cos \theta - v \sin \theta$
  - $y = u \sin \theta + v \cos \theta$

- Any function of $u$ and $v$:
  - $x = f_1(u, v)$
  - $y = f_2(u, v)$

- "Swirl"
- "Fish-eye"
- "Rain"
Forward Warping/Mapping
- Iterate over source, sending pixels to destination
  - Forward mapping:
    ```
    for (int u = 0; u < u_max; u++) {
        for (int v = 0; v < v_max; v++) {
            float u = f_x(u,v);
            float v = f_y(u,v);
            dat(x,y) = src(u,v);
        }
    }
    ```

Inverse Warping/Mapping
- Iterate destination, finding pixels from source
  - Reverse mapping:
    ```
    for (int x = 0; x < x_max; x++) {
        for (int y = 0; y < y_max; y++) {
            float u = f_x^{-1}(x,y);
            float v = f_y^{-1}(x,y);
            src(x,y) = dat(u,v);
        }
    }
    ```

Filtering or Resampling
- Compute weighted sum of pixel neighborhood
  - Weights are normalized values of kernel function
  - Equivalent to convolution at samples with kernel
  - Find good (normalized) filters \( h \) using earlier ideas
    ```
    s = 0;
    for (u' = u - width; u' <= u + width; u'++)
        for (v' = v - width; v' <= v + width; v'++)
            s += h(u'-u, v'-v) * src(u', v');
    ```

Inverse Warping/Mapping
- Iterate destination, finding pixels from source
  - Non-integer evaluation source image, resample
    - May oversample source
    - But no holes
    - Simpler, better than forward mapping
Filters for Assignment

Implement 3 filters (for anti-aliased shift, resize):
- Nearest neighbor or point sampling
- Hat filter (linear or triangle)
  \[ h(u) = 1 - |u| \]
- Mitchell cubic filter (form in assignments). This is a good finite filter that approximates ideal sinc without ringing or infinite width. Alternative is gaussian

Construct 2D filters by multiplying 1D filters
\[ h(u, v) = h(u)h(v) \]

Filtering Methods Comparison

- Trade-offs
  - Aliasing versus blurring
  - Computation speed

Point  Bilinear  Gaussian