Advanced Computer Graphics (Fall 2010)
CS 283, Lecture 3: Sampling and Reconstruction
Ravi Ramamoorthi
http://inst.eecs.berkeley.edu/~cs283/fa10

Basics
- Start working on raytracer assignment (if necessary)
- First 3 lectures cover basic topics
  - Rendering: Raytracing (required for homeworks 2, 3)
  - Sampling and Reconstruction, Fourier Analysis
  - 3D objects and meshes
- Then we start main part of course
  - Meshes and assignment 1
- This lecture review for some of you
  - But needed to bring everyone up to speed
  - Much more detailed material available if interested (we have only limited time; cover it quickly)

Outline
- Basic ideas of sampling, reconstruction, aliasing
- Signal processing and Fourier analysis
- Implementation of digital filters
- Section 14.10 of FvDFH (you really should read)

Sampling and Reconstruction
- An image is a 2D array of samples
- Discrete samples from real-world continuous signal

(Spatial) Aliasing
(Spatial) Aliasing

- Jaggies probably biggest aliasing problem

Sampling and Aliasing

- Artifacts due to undersampling or poor reconstruction
- Formally, high frequencies masquerading as low
- E.g. high frequency line as low freq jaggies

Image Processing pipeline

Outline

- Basic ideas of sampling, reconstruction, aliasing
- Signal processing and Fourier analysis
- Implementation of digital filters
- Section 14.10 of textbook

Motivation

- Formal analysis of sampling and reconstruction
- Important theory (signal-processing) for graphics
- Also relevant in rendering, modeling, animation

Ideas

- Signal (function of time generally, here of space)
- Continuous: defined at all points; discrete: on a grid
- High frequency: rapid variation; Low Freq: slow variation
- Images are converting continuous to discrete. Do this sampling as best as possible.
- Signal processing theory tells us how best to do this
- Based on concept of frequency domain Fourier analysis
Sampling Theory
Analysis in the frequency (not spatial) domain
- Sum of sine waves, with possibly different offsets (phase)
- Each wave different frequency, amplitude

Fourier Transform
- Tool for converting from spatial to frequency domain
- Or vice versa
- One of most important mathematical ideas
- Computational algorithm: Fast Fourier Transform
- One of 10 great algorithms scientific computing
- Makes Fourier processing possible (images etc.)
- Not discussed here, but look up if interested

Fourier Transform
- Simple case, function sum of sines, cosines
  \[ f(x) = \sum_{n} F(u)e^{2\pi i nu} \]
  \[ F(u) = \int_{0}^{2\pi} f(x)e^{-2\pi i xu}dx \]
- Continuous infinite case
  Forward Transform: \[ F(u) = \int_{-\infty}^{\infty} f(x)e^{-2\pi i xu}dx \]
  Inverse Transform: \[ f(x) = \int_{-\infty}^{\infty} F(u)e^{2\pi i xu}du \]

Fourier Transform: Examples 1
- Single sine curve (+constant DC term)

Fourier Transform Examples 2
- Common examples
  \[ f(x) \]
  \[ F(u) \]
  \[ \delta(x - x_0) \]
  \[ e^{-2\pi i xu_0} \]
  \[ 1 \]
  \[ e^{ax^2} \]
  \[ \sqrt{\frac{a}{2\pi}} e^{-\frac{(x - \mu)^2}{2\sigma^2}} \]
Fourier Transform Properties

Forward Transform: 
\[ F(u) = \int_{-\infty}^{\infty} f(x)e^{-2\pi iux}dx \]

Inverse Transform: 
\[ f(x) = \int_{-\infty}^{\infty} F(u)e^{2\pi iux}du \]

- Common properties
  - Linearity: \( F(af(x) + bg(x)) = aF(f(x)) + bF(g(x)) \)
  - Derivatives: [Integrate by parts] \( F(f'(x)) = \int_{-\infty}^{\infty} f'(x)e^{-2\pi iux}dx \)
    \[ = 2\pi iuF(u) \]
  - 2D Fourier Transform
    Forward Transform: \( F(u,v) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x,y)e^{-2\pi iux}e^{-2\pi ivy}dxdy \)
    Inverse Transform: \( f(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} F(u,v)e^{2\pi iux}e^{2\pi ivy}dudv \)

Sampling Theorem, Bandlimiting

- A signal can be reconstructed from its samples, if the original signal has no frequencies above half the sampling frequency – Shannon
- The minimum sampling rate for a bandlimited function is called the Nyquist rate

Antialiasing

- Sample at higher rate
  - Not always possible
  - Real world: lines have infinitely high frequencies, can’t sample at high enough resolution
- Prefilter to bandlimit signal
  - Low-pass filtering (blurring)
  - Trade blurriness for aliasing

Ideal bandlimiting filter

- Formal derivation is homework exercise

Outline

- Basic ideas of sampling, reconstruction, aliasing
- Signal processing and Fourier analysis
  - Convolution
- Implementation of digital filters
- Section 14.10 of FvDFH
Convolution 1
- Spatial domain: output pixel is weighted sum of pixels in neighborhood of input image
  - Pattern of weights is the “filter”

Convolution 2
- Example 1:

Convolution 3
- Example 1:

Convolution 4
- Example 1:

Convolution in Frequency Domain
- Convolution (f is signal; g is filter [or vice versa])
  \[ h(y) = \int_{-\infty}^{\infty} f(x)g(y-x)dx = \int_{-\infty}^{\infty} g(x)f(y-x)dx \]

- Fourier analysis (frequency domain multiplication)
  \[ H(u) = F(u)G(u) \]

Convolution 5
- Example 1:
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

\[
I'(x, y) = \sum_{a=-\text{width}}^{\text{width}} \sum_{b=-\text{height}}^{\text{height}} f(x-a, y-b) \cdot I(x-a, y-b)
\]

Example:

\[
\begin{bmatrix}
2 & 0 & -7 \\
5 & 4 & -9 \\
1 & -6 & -2
\end{bmatrix}
\]
**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)

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

**Spatial**

**Frequency**

\[
s = \text{floor}(n/2)/2
\]

**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 = \( \frac{1}{2} \)

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

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

\[
\begin{bmatrix}
1 & 7 & 1 \\
7 & 54 & 7 \\
1 & 7 & 1
\end{bmatrix}
\]

**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
Details

- Step 1: Convolution with gradient (Sobel) filter
  - Edges occur where image gradients are large
  - Separately for horizontal and vertical directions
  \[
  f_{xx}(x,y) = \begin{pmatrix}
  -1 & 0 & 1 \\
  -2 & 0 & 2 \\
  -1 & 0 & 1 
  \end{pmatrix} \quad f_{yy}(x,y) = \begin{pmatrix}
  1 & 2 & 1 \\
  0 & 0 & 0 \\
  -1 & -2 & -1 
  \end{pmatrix}
  \]
- Step 2: Magnitude of gradient
  - Norm of horizontal and vertical gradients
  \[
  G = \sqrt{f_{xx}(x,y)^2 + f_{yy}(x,y)^2}
  \]
- Step 3: Thresholding

Outline

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

Antialiased Shift

Shift image based on (fractional) \( s_x \) and \( s_y \)
- Check for integers, treat separately
- Otherwise convolve/resample with kernel/filter \( h \):
\[
\begin{align*}
  u &= x - s_x, \quad v = y - s_y \\
  l(x,y) &= \sum_{u',v'} h(u'-u,v'-v) l(u',v')
\end{align*}
\]

Antialiased Scale Magnification

Magnify image (scale \( s \) or \( \gamma > 1 \))
- Interpolate between orig. samples to evaluate frac vals
- Do so by convolving/resampling with kernel/filter:
- Treat the two image dimensions independently (diff scales)
\[
\begin{align*}
  \gamma &= \frac{1}{\text{width}} \\
  l(x) &= \sum_{u'=\gamma \times \text{width}} h(u'-u) l(u')
\end{align*}
\]

Antialiased Scale Minification

Minify (reduce size of) image
- Similar in some ways to mipmapping for texture maps
- We use fat pixels of size \( 1/\gamma \), with new size \( \gamma \) orig size
- Each fat pixel must integrate over corresponding region in original image using the filter kernel.
\[
\begin{align*}
  \gamma &= \frac{1}{\text{width}} \\
  l(x) &= \sum_{u'=\gamma \times \text{width}} h(\gamma u'-x) l(u')
\end{align*}
\]

Antialiased Scale Minification

checkerboard.bmp 300x300: point sample
checkerboard.bmp 300x300: Mitchell