Advanced Rendering Techniques

Dr. David Breen
University Crossings 149
Tuesday 6PM → 8:50PM

Presentation 5
5/5/09
Questions from Last Time?

Acceleration techniques
- Bounding Regions
- Acceleration Spatial Data Structures
  - Regular Grids
  - Adaptive Grids
  - Bounding Volume Hierarchies
- Dot product problem
Slide Credits

- David Luebke - University of Virginia
- Kevin Suffern - University of Technology, Sydney, Australia
- Linge Bai - Drexel
Anti-aliasing

- Aliasing: signal processing term with very specific meaning
- Aliasing: computer graphics term for any unwanted visual artifact based on sampling
  - Jaggies, drop-outs, etc.
- Anti-aliasing: computer graphics term for fixing these unwanted artifacts
- We’ll tackle these in order
Signal Processing

- Raster display: regular sampling of a continuous(?) function
- Think about sampling a 1-D function:
Signal Processing

- Sampling a 1-D function:
Signal Processing

Sampling a 1-D function:
Signal Processing

- Sampling a 1-D function:
  - What do you notice?
Signal Processing

- Sampling a 1-D function: what do you notice?
  - Jagged, not smooth
Signal Processing

- Sampling a 1-D function: what do you notice?
  - Jagged, not smooth
  - Loses information!
Sampling a 1-D function: what do you notice?
- Jagged, not smooth
- Loses information!

What can we do about these?
- Use higher-order reconstruction
- Use more samples
- *How many more samples?*
The Sampling Theorem

- Obviously, the more samples we take the better those samples approximate the original function.
- The sampling theorem:

  A continuous **bandlimited** function can be completely represented by a set of equally spaced samples, if the samples occur at more than twice the frequency of the highest frequency component of the function.
The Sampling Theorem

- In other words, to adequately capture a function with maximum frequency $F$, we need to sample it at frequency $N = 2F$.
- $N$ is called the *Nyquist limit*.
- What is the maximum frequency?
The Sampling Theorem

- An example: sinusoids
The Sampling Theorem

- An example: sinusoids
Fourier Theory

- Fourier theory lets us decompose any signal into the sum of (a possibly infinite number of) sine waves
- Next comes the math...
- Switch to hand-written notes
Fourier Theory

• Any signal $I(x)$ can be represented as a spectrum of sine waves using the Fourier integral:

$$F(u) = \int_{-\infty}^{\infty} I(x)e^{-i2\pi ux} \, dx$$

The transform is reversible:

$$I(x) = \int_{-\infty}^{\infty} F(u)e^{i2\pi ux} \, du$$

• $F(u)$ and $I(x)$ are a Fourier transform pair, or Fourier transform dual. For convenience, let $\supset$ denote “has a Fourier transform given by”. So the above reduces to $I(x) \supset F(u)$. 
Fourier Theory

• In the limit, this integral is just an infinite number of discreet terms summed together. Euler's formula tells us that:
  \[
  \frac{1}{e^{i\theta}} = \cos\theta - i\sin\theta
  \]

• Therefore our signal \( F(u) \) becomes the sum of a spectrum of sine waves, and \( u \) is known as the **frequency domain**.

• Notice: \( I(x) \) is (generally) real, but \( F(u) \) is usually complex. We therefore decompose \( F(u) \) into **amplitude** and **phase**.
Fourier Theory

- Think of $F(u)$ as sum of real and imaginary parts:
  $$F(u) = \text{Real}(u) + \text{Imag}(u)$$
- Then **amplitude** spectrum is
  $$|F(u)| = \sqrt{((\text{Real}(u))^2 + (\text{Imag}(u))^2}$$
- And **phase** spectrum is
  $$\Phi(u) = \tan^{-1}\left(\frac{\text{Imag}(u)}{\text{Real}(u)}\right)$$
- Why?
  - A point in $|F(u)|$ specifies the amplitude of a sine wave at frequency $u$ Hz, and of phase specified by the corresponding point in $\Phi(u)$.
  - Adding together all the sinusoids as specified by their amplitude & phase spectra generates the function $l(x)$!
Fourier Theory

• Some examples

If this is $I(x)$:

What is $|F(u)|$?

What is $\Phi(u)$?

perfect sine wave

Amplitude of sine wave

0
Fourier Theory

If this is $|F(u)|$:

What does $I(x)$ look like?

If this is $I(x)$:

What does $|F(u)|$ look like?

superposition of 2 sine waves

$\Delta X$

$1/\Delta X$
\[ I(x) : \quad |F(u)| : \]

perfect Gaussian

\[ |F(u)| : \quad I(x) : \]

perfect box

sinc function = \( \frac{\sin(x)}{x} \)

perfect Gaussian, narrow vs. wide
(variance = 1/variance)
Convolution

- Let the convolution operator be denoted by $*$:

$$I(x) * g(x) \equiv \int_{-\infty}^{\infty} I(\alpha)g(x - \alpha)\,d\alpha$$

- Intuitively, the convolution of $I(x) * g(x)$ is created by “sliding” one function past the other and integrating the areas of the function at a given point.

- Think of this equation for various fixed values of $\alpha$. 
The Big Result

- Fourier theory tells us that convolution in one domain = multiplication in the other!
  \[ I(x) * g(x) \supseteq F(u)G(u) \quad (I \supseteq F, g \supseteq G) \]
- Now we can start to explain the sampling theorem and Nyquist limit.
Here’s a function $I(x)$

A sampling function $S(x)$ is an infinite series of pulses, at $\Delta x$ intervals

Sampling $I(x)$ corresponds to multiplying by $S(x)$...

... and its Fourier transform $F(u)$

Its FT $S(u)$ is another, at $1/\Delta x$ intervals

Which is the same as convolving in the frequency domain with $S(u)$. 
What happens if the sample rate of $S(x)$ is too low?

Which can be recovered by reversing the Fourier transform.

$$I(x)$$

Throw away the “extra copies” by multiplying with a box function...

$$\frac{1}{2\Delta}$$

And get the original Fourier domain version $F(u)$ of the function $I(x)$!

$$F(u)$$

$$= \text{F.T.}$$
The adjacent spectra overlap! Multiplying by a box filter will not recover the correct information.

High Frequencies (from adjacent spectrum) masquerading as lower frequencies!
Aliasing

• So...we can now see how aliasing occurs when a function containing high-frequency components is sampled at too low a rate.

• What can we do about this?
Prefiltering

• We can **prefilter** the image, removing all high-frequency components **before** sampling:

To remove high-frequency components, multiply by a box filter:

\[ X = \ldots \]

... to give a smooth, rounded-off **blurred** box!

The result is a chopped-off frequency spectrum, which can be F.T.’d back to the space domain...
Prefiltering (cont’d)

• Prefiltering -- also called low-pass filtering -- blurs the resulting image, losing some of the original information. But it guarantees that what information is recovered:
  -- is the maximum amount possible
  -- is not misrepresented
• This is why, say, anti-aliased text in Photoshop looks blurry, yet better overall.
• Q: How might we implement prefILTERING in computer graphics?
• Problem: Using (say) Z-buffer or ray tracing, we have no continuous function $f(x)$!
Recap: Fourier Theory

- Can decompose any signal \( l(x) \) into a spectrum of sine waves \( F(u) \)
- \( F(u) \) is complex; we represent it with:
  
  \[
  |F(u)| = \left[\text{Real}(u)^2 + \text{Imag}(u)^2\right]^{1/2}
  \]
  
  \[
  \phi(u) = \tan^{-1}\left[\frac{\text{Imag}(u)}{\text{Real}(u)}\right]
  \]
- These are called the *amplitude* and *phase* spectra
Recap: Aliasing

- Sampling a function
  = multiplying $I(x)$ by sampling fn $S_x$
  = convolving $F(u)$ by sampling fn $S_{1/x}$
  = copying $F(u)$ at regular intervals
- If the copies of $F(u)$ overlap, high frequencies “fold over”, appearing as lower frequencies
- This is known as *aliasing*
Recap: Prefiltering

- Eliminate high frequencies before sampling
  - Convert I(x) to F(u)
  - Apply a low-pass filter (e.g., multiply F(u) by a box function)
  - Then sample. Result: no aliasing!

- Problem: most rendering algorithms generate sampled function directly
  - e.g., Z-buffer, ray tracing
The simplest way to reduce aliasing artifacts is *supersampling*

- Increase the resolution of the samples
- Average the results down

Sometimes called *postfiltering*

- Create virtual image at higher resolution than the final image
- Apply a low-pass filter
- Resample filtered image
Supersampling: Limitations

- Q: *What practical consideration hampers supersampling?*
  - A: *Storage* goes up quadratically

- Q: *What theoretical problem does supersampling suffer?*
  - A: *Doesn’t eliminate aliasing!*  
    Supersampling simply shifts the Nyquist limit higher
Supersampling: Worst Case

Q: Give a simple scene containing infinite frequencies

A: A checkered ground plane receding into the background
Supersampling

- Create virtual image at higher resolution than the final image
- This is easy
Supersampling

- Apply a low-pass filter
  - Convert to frequency domain
  - Multiply by a box function
- Expensive! *Can we simplify this?*
  - Recall: multiplication in frequency equals convolution in space, so…
  - Just convolve initial sampled image with the FT of a box function
- Isn’t this a lot of work?
Supersampling

Q: *What filter should we use in space to effect multiplying by a box function in frequency?*

A: The *sinc* function \((\sin(x) / x)\)

Q: *Why is this hard?*

- Sinc function has infinite support
- Sinc function has negative lobes
Supersampling

- Other common filters:
  - Truncated sinc
  - Box
  - Triangle
  - Gaussian

- Each has their own frequency spectrum and artifacts
Supersampling: Digital Convolution

- In practice, we combine steps 2 & 3:
  - Create virtual image at higher resolution than the final image
  - Apply a low-pass filter
  - Resample filtered image

- Idea: only create filtered image at new sample points
  - I.e., only convolve filter with image at new points
Q: What does convolving a filter with an image entail at each sample point?

A: Multiplying and summing values

Example (also F&vD p 642):
Supersampling

- Typical supersampling algorithm:
  - Compute multiple samples per pixel
  - Combine sample values for pixel’s value using simple average

Q: What filter does this equate to?
A: Box filter -- one of the worst!

Q: What’s wrong with box filters?
- Passes infinitely high frequencies
- Attenuates desired frequencies
Supersampling
In Practice

- Sinc function: ideal but impractical
- One approximation: $\text{sinc}^2$
- Another: Guassian falloff
- Q: *How wide (what res) should filter be?*
- A: As wide as possible!
  In practice: 3x3, 5x5, at most 7x7
Supersampling: Summary

- Supersampling improves aliasing artifacts by shifting the Nyquist limit.
- It works by calculating a high-res image and filtering down to final res.
- “Filtering down” means simultaneous convolution and resampling.
- This equates to a weighted average.
- Wider filter ➔ better results ➔ more work.
Supersampling: Cons & Pros

- Supersampling cons
  - Doesn’t eliminate aliasing, just shifts the Nyquist limit higher
    - Can’t fix some scenes (e.g., checkerboard)
  - Badly inflates storage requirements

- Supersampling pros
  - Relatively easy
  - Often works all right in practice
  - *Can be added to a standard renderer*
Catmull’s Algorithm

- Find fragment areas
- Multiply by fragment colors
- Sum for final pixel color
Catmull’s Algorithm

- First real attempt to filter in continuous domain
- Very expensive
  - Clipping polygons to fragments
  - Sorting polygon fragments by depth
- Equates to box filter
The A-Buffer

- Idea: approximate continuous filtering by subpixel sampling
- Summing areas now becomes simple
The A-Buffer

Advantages:

- Incorporating into scanline renderer reduces storage costs dramatically
- Processing per pixel depends only on number of visible fragments
- Can be implemented efficiently using bitwise logical ops on subpixel masks
The A-Buffer

Disadvantages

- Still basically a supersampling algorithm
- Not a hardware-friendly algorithm
  - Lists of potentially visible polygons can grow without limit
  - Work per-pixel non-deterministic
Stochastic Sampling

- Sampling theory tells us that with a regular sampling grid, frequencies higher than the Nyquist limit will alias.
- Q: What about *irregular* sampling?
- A: High frequencies appear as noise, not aliases.
- This turns out to bother our visual system less!
Stochastic Sampling

- An intuitive argument:
  - In stochastic sampling, every region of the image has a finite probability of being sampled
  - Thus small features that fall between uniform sample points tend to be detected by non-uniform samples
Stochastic Sampling

- Idea: randomizing distribution of samples scatters aliases into noise
- Problem: what type of random distribution to adopt?
- Reason: type of randomness used affects spectral characteristics of noise into which high frequencies are converted
Stochastic Sampling

Problem: given a pixel, how to distribute points (samples) within it?
Stochastic Sampling

- **Poisson** distribution:
  - Completely random
  - Add points at random until area is full.
  - Uniform distribution: some neighboring samples close together, some distant
Stochastic Sampling

- Poisson disc distribution:
  - Poisson distribution, with minimum-distance constraint between samples
  - Add points at random, removing again if they are too close to any previous points
  - Very even-looking distribution
Stochastic Sampling

- *Jittered* distribution
  - Start with regular grid of samples
  - Perturb each sample slightly in a random direction
  - More “clumpy” or granular in appearance
Stochastic Sampling

- Spectral characteristics of these distributions:
  - Poisson: completely uniform (white noise). High and low frequencies equally present
  - Poisson disc: Pulse at origin (DC component of image), surrounded by empty ring (no low frequencies), surrounded by white noise
  - Jitter: Approximates Poisson disc spectrum, but with a smaller empty disc.
Comparison

Regular

Random

Poisson disk

Jitter
Recap: Antialiasing Strategies

- Prefiltering: low-pass filter the signal before sampling

  - Pros:
    - Guaranteed to eliminate aliasing
    - Preserves *all* desired frequencies

  - Cons:
    - Expensive
    - Can introduce “ringing”
    - Doesn’t fit most rendering algorithms
Recap:
Antialiasing Strategies

- Supersampling: sample at higher resolution, then filter down

- Pros:
  - Conceptually simple
  - Easy to retrofit existing renderers
  - Works well most of the time

- Cons:
  - High storage costs
  - Doesn’t eliminate aliasing, just shifts Nyquist limit upwards
Recap: Antialiasing Strategies

- A-Buffer: approximate prefiltering of continuous signal by sampling

- **Pros:**
  - Integrating with scan-line renderer keeps storage costs low
  - Can be efficiently implemented with clever bitwise operations

- **Cons:**
  - Still basically a supersampling approach
  - Doesn’t integrate with ray-tracing
Recap: Antialiasing Strategies

- Stochastic supersampling: convert high frequencies into noise
  - Pros:
    - Retains advantages of supersampling
    - Noise is visually better than aliasing
  - Cons:
    - Ideal sampling distribution (Poisson Disc) is expensive
Computer Graphics Rendering Techniques
Advanced Image Synthesis Techniques

Adaptive Super-Sampling
Antialiasing in ray tracing through adaptive super-sampling


In ray tracing, the samples are the rays we shoot through each pixel.

We can increase the sampling rate by shooting more rays.

Super-sampling refers to shooting more than one ray per pixel.

In adaptive super-sampling, we only shoot more rays in areas where the intensity varies greatly, for example across edges and other areas with high intensity gradients.
Consider ray tracing two pixels 1 & 2, where pixel 2 overlaps an object.
To perform adaptive super-sampling we compare the colours at the four corners of each pixel.

For simplicity, let the colours at the corners be $A, B, C, \text{etc}$.

If the colours differ by more than a pre-defined *tolerance*, we perform the super-sampling by shooting more rays through the pixel.

For pixel 1 we can test

$$|A - B| > \text{tolerance}$$
$$|A - D| > \text{tolerance}$$
$$|E - D| > \text{tolerance}$$
$$|E - B| > \text{tolerance}$$
The tolerance can be 1%, 5%, 10%, etc.

We must compare the red, green, and blue components of the colours separately.

For a given tolerance, a slightly stronger test would be to include the diagonals, $|A - E|$ and $|D - D|$.

If all corners of pixel 1 are within tolerance, the colour of pixel 1 is

$$C = \frac{A + B + D + E}{4}$$
Suppose we are ray tracing a yellow object on a black background, and the rays through the corners of pixel 1 do not intersect the object.

Then the colour of pixel 1 is black:
If any of the corners are out of tolerance, we shoot 5 extra rays:

one through the centre of the pixel
4 through the midpoints of the sides.

If pixel 2 is out of tolerance, the following figure shows the additional rays.
We now test the four sub pixels, to see if they are out of tolerance.

If they are all in tolerance, or we decide not to further subdivide pixel 2, its colour is

\[
C = \frac{1}{4} \left\{ \frac{B + G + H + I}{4} + \frac{G + C + I + J}{4} + \frac{H + I + E + K}{4} + \frac{I + J + K + F}{4} \right\}
\]
In this case the colour of pixel 2 is constructed as

Here the sub pixel $GCIJ$ is 75% black and 25% yellow.
If any of the sub pixels are out of tolerance, we can, if we wish, subdivide again.

If $GCIJ$ is out of tolerance, we shoot another five rays as shown in the following figure.
After this subdivision, the colour of pixel 2 becomes

\[
C = \frac{1}{4} \left\{ \frac{B + G + H + I}{4} + \frac{1}{4} \left[ \frac{G + L + M + N}{4} + \frac{L + C + N + O}{4} + \frac{M + N + I + P}{4} + \frac{N + O + P + J}{4} + \frac{H + I + E + K}{4} + \frac{I + J + K + F}{4} \right] \right\}
\]
Now the colour of pixel 2 is constructed as
Here:

- the sub pixel *LCNO* is 50% black, 50% yellow.

- the sub pixel *NOPJ* is 75% black, 25% yellow.

In this two colour situation, the colour of a pixel is a *weighted average* of the areas of the two colours over the area of the pixel.

The more the pixel is subdivided, the more accurate the weightings become.

The same principle applies when there are more than two colours in the pixel area.
Example

Ray trace a yellow sphere on a black background
Programming aspects

We need to shoot an additional row and column of rays: one on the top and one on the right.

These are required to give four corners for the top row and the right column of pixels.
\( = \) original rays
\( = \) new rays
The best way to program the super-sampling is to use recursion.

We must make sure we set a limit on the number of times we subdivide.

For efficiency, we should also check if the new rays at the centres and midpoints of the pixel sides have already been traced.

However, this complicates the code.

**Shortcuts:**

1. Do antialiasing as a post process, and don't antialias the first row and last column of pixels.

2. One level of subdivision (no recursion)
Wrap Up

- Discuss next programming assignment
  - Add adaptive supersampling
- Discuss status/problems/issues with this week’s programming assignment