Sampling
Sampling

• The ray tracer **samples** geometry, only gathering information from a finite number of ray intersections (points)

• In contrast, a scanline renderer can push all the triangles onto the image plane
  – however, the simple approach of testing which pixel centers are inside which triangles is also a form of sampling
  – one needs to calculate the overlap between projected triangles and finite size (square) pixels in order to avoid sampling
Theoretically Perfect Sensor

- The value seen by a theoretically perfect sensor corresponds to the fraction of the sensor “covered” by the object as well as the exposure time.
Real World Sensors

- Even the eye and camera do not collect all of the information
Example: Jaggies

- Send a ray from the center of each pixel and shade the entire pixel based on the color returned by the ray
- “Jaggies” occur because lines of pixels do not line up with lines of objects
  - occurs in scanline rendering as well...
Sampling causes **aliasing** artifacts

An alias or imposter appears in the place of the actual/true feature

E.g., in the previous slide, a jagged line appears in the place of the straight line

**Anti-aliasing** techniques prevent/alleviate aliasing artifacts caused by inadequate sampling
Example: Texturing Artifacts

• Texture coordinates nonuniformly mapped onto a surface cause the sampling rate to vary across the texture.

Incorrect Aliasing

Correct
Example: Sparkling Highlights

• Aliasing in **normal vectors** can cause noise in specular highlights
Example: Wagon Wheel Effect

- A spinning wheel appears to spin backwards when the motion is insufficiently sampled in time.
- This is temporal aliasing.
Sampling Rate

• Sampling artifacts can be reduced by increasing the number of pixels in an image (and consequently the number of samples)

• Increasing the resolution of an image is costly
  – Requires additional time to render because the number of rays increases
  – Displaying the high resolution image and/or using the image as a texture requires both additional storage and computation

• Optimizing the Sampling Rate
  – Use the **lowest possible** sampling rate that does not result in noticeable aliasing artifacts
  – What is this ideal sampling rate?

11/60
Sampling Rate: 4 Samples per Period

\[ f(x) = \cos(2\pi x) \]
Discrete Samples

\[ f(x) = \cos(2\pi x) \]
Continuous Reconstruction

\[ f(x) = \cos(2\pi x) \]
Sampling Rate: 2 Samples per Period

$$f(x) = \cos(2\pi x)$$
Discrete Samples

\[ f(x) = \cos(2\pi x) \]
Continuous Reconstruction

\[ f(x) = \cos(2\pi x) \]
Sampling Rate: 1 Sample per Period

\[ f(x) = \cos(2\pi x) \]
Discrete Samples

\[ f(x) = \cos(2\pi x) \]
Continuous Reconstruction

- no longer matches the original function
- appears to be a different function
- Aliased!
Sampling Rate: 2/3 Sample per Period

\[ f(x) = \cos(6\pi x + \frac{\pi}{3}) \]
Discrete Samples

\[ f(x) = \cos(6\pi x + \frac{\pi}{3}) \]
Continuous Reconstruction

\[ f(x) = \cos(6\pi x + \frac{\pi}{3}) \]

- no longer matches the original function
- appears to be a different function
- Aliased!
Aliasing

For this sampling rate, these two cosine waves of different frequencies/amplitudes become aliased.
Can’t tell the difference between them.
Nyquist Frequency
Aliasing

- Sampling at too low a rate results in aliasing
- Two different signals become indistinguishable
  - an alias or imposter

- Nyquist-Shannon Sampling Theorem
  - If a function $f(t)$ contains no frequencies higher than $W$ hertz, it can be completely determined by samples at a series of points spaced $1/(2W)$ seconds apart
  - That is, 2 samples per period are required to prevent aliasing
Anti-Aliasing

• Given a fixed sampling frequency (e.g. #pixels in a feature), the **Nyquist frequency** is defined as half the sampling frequency
  – Need 2 samples per period to prevent aliasing

• If the real world function being sampled has no frequencies above the Nyquist frequency, no aliasing occurs

• But **any real world frequencies above the Nyquist frequency will appear as aliases to the sampler**

• **Before sampling, pre-filter out all frequencies in the real world signal higher than the Nyquist frequency** in order to prevent aliasing

• We can do this by transforming the function **into frequency space**...
Fourier Transform
Fourier Transform

- Transform from the spatial domain to the frequency domain
  - and vice versa

\[
F(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi j k x} \, dx
\]

\[
f(x) = \int_{-\infty}^{\infty} F(k) e^{2\pi j k x} \, dk
\]

\(f(x)\) : Spatial Domain

\(F(k)\) : Frequency Domain

\[e^{jx} = \cos(x) + j \sin(x)\]

\[
\cos(x) = \frac{e^{jx} + e^{-jx}}{2} \quad \sin(x) = \frac{e^{jx} - e^{-jx}}{2}
\]
Constant Function

\[ f(x) = 1 \]
Low Frequency Cosine

\[ f(x) = \cos(\pi x) \]

\( \mathcal{F}(k) \)
High Frequency Cosine

\[ f(x) = \cos(2\pi x) \]
Narrow Gaussian

\[ f(x) = \frac{2}{\sqrt{\pi}} e^{-4x^2} \]
Wider Gaussian

\[ f(x) = \frac{1}{2\sqrt{\pi}} e^{-25x^2} \]

Wider

\[ \mathcal{F}(k) \]

Narrower
Anti-Aliasing
Sum of 2 Different Frequency and Amplitude Cosine Functions
Discrete Samples
Reconstructed Alias
Pre-filtering in Fourier Space

1. Compute the Fourier transform
Pre-filtering in Fourier Space

2. Apply a low-pass filter to remove frequencies higher than the Nyquist frequency
Pre-filtering in Fourier Space

2. Apply a low-pass filter to remove frequencies higher than the Nyquist frequency
Pre-filtering in Fourier Space

3. Compute the inverse Fourier transform of the low-pass filtered signal
Pre-filtering in Fourier Space

4. The lower frequency component of the original function can be sampled and reconstructed without aliasing.
4. The lower frequency component of the original function can then be sampled and reconstructed without aliasing.
Pre-filtering in Fourier Space

4. The lower frequency component of the original function can then be sampled and reconstructed without aliasing.
Pre-filter Before Sampling!

- Sampling causes high frequency components to masquerade as low frequency components
- There is no way to untangle the high frequencies masquerading as low frequencies from the actual low frequencies
- **Must** pre-filter out the high frequencies before sampling in order to avoid aliasing
Blurring vs. Antialiasing

Blurring Jaggies \textit{After} Sampling

Pre-filtering
Images

- Images have **discrete** values and are not continuous functions
  - Use a **discrete** Fourier transform
  - Fast Fourier Transform (FFT) algorithm computes the discrete Fourier transform and its inverse in $O(n \log n)$ time, where $n$ is the number of samples
  - 2D **discrete Fourier transform** is computed by applying 1D transforms (i.e. FFTs) along each dimension

1. Compute the discrete Fourier transform (via FFT)
   - Transform the discrete image values into another array of discrete values

2. Perform operations

3. Compute the inverse discrete Fourier transform (via FFT)
Constant Function

Spatial Domain

Frequency Domain
\[ \sin \left( \frac{2\pi}{32}x \right) \]
\[ \sin\left(\frac{2\pi}{16}\right)x \]
\( \sin\left(\frac{2\pi}{16}\right)y \)
$$\sin\left(\frac{2\pi}{32}\right)x \times \sin\left(\frac{2\pi}{16}\right)y$$
Real World Image

Spatial Domain

Frequency Domain
Lowest Frequencies

Spatial Domain

Frequency Domain
Intermediate Frequencies

Spatial Domain

Frequency Domain
Larger Intermediate Frequencies

Spatial Domain

Frequency Domain
Highest Frequencies (edges)

Spatial Domain

Frequency Domain
Convolution
Anti-Aliasing

- Let $f$ and $g$ be functions in the spatial domain, and $\mathcal{F}(f)$ and $\mathcal{F}(g)$ be the transformations of these functions into the frequency domain.
- $f$ is an image, $\mathcal{F}(f)$ is the frequency space version of the image, and $\mathcal{F}(g)$ is a low pass filter to be applied in the frequency domain via multiplication component by component, i.e. $\mathcal{F}(f) \cdot \mathcal{F}(g)$
  - this multiplication zeroes out high frequency components of $\mathcal{F}(f)$
- Then, inverse transform to the final result to obtain:

\[(f \ast g) = \mathcal{F}^{-1} (\mathcal{F}(f) \cdot \mathcal{F}(g))\]
Convolution

• Applying low pass filter $\mathcal{F}(g)$ to the transform $f$ in the frequency domain is equivalent to a spatial convolution of $f$ with the inverse transform of the filter (i.e. $g$):

$$ (f * g) = \mathcal{F}^{-1} (\mathcal{F}(f) \cdot \mathcal{F}(g)) $$

• Spatial convolution is defined by

$$ (f * g)(t) = \int_{-\infty}^{\infty} f(\tau) g(t - \tau) \, d\tau $$

$$ = \int_{-\infty}^{\infty} f(t - \tau) g(\tau) \, d\tau $$
Convolution

• Skip the FFT, and instead apply convolution in the spatial domain

• Unfortunately, (recall) narrow low pass filters in the frequency domain result in wide convolution kernels in the spatial domain (and vice versa)
  – So it takes a lot of work to apply a narrow low pass frequency domain filter in the spatial domain via convolution (one has to consider a large number of non-local pixels)

• For efficiency, we’d prefer a small local convolution kernel in the spatial domain
  – This gives a wide frequency domain filter, and so won’t perfectly filter out all high frequencies
  – But, meh, it’s ok-ish...
RECALL: Narrow Gaussian

\[ f(x) = \frac{2}{\sqrt{\pi}} e^{-4x^2} \]

Narrow

Wide
RECALL: Wider Gaussian

\[ f(x) = \frac{1}{2\sqrt{\pi}} e^{-25x^2} \]

Wider

Narrower

\[ \mathcal{F}(k) \]
Box Filter

• Contains uniform weights throughout a certain range \([x_0, x_1] \times [y_0, y_1]\) and zeroes elsewhere

• Apply a box filter via discrete convolution:
  – Overlay the filter on the 2D array of image values
  – Multiply each pair of overlayed values
  – Sum (integrate) the results to obtain a new value

  • usually defined at the center of the filter

\[
\begin{array}{cccc}
\frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\
\frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\
\frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\
\frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\
\frac{1}{16} & \frac{1}{16} & \frac{1}{16} & \frac{1}{16} \\
\end{array}
\]
Box Filter Approximates a Low-Pass Filter

Spatial Domain  

Frequency Domain
Wider Box Filter Removes More Frequencies

Spatial Domain

Frequency Domain
Supersampling
Supersampling

- Average multiple point samples distributed throughout a single pixel to obtain the value for that pixel
- Rendering a 100 by 100 image using uniform 4 by 4 supersampling is equivalent to rendering a 400 by 400 image and applying a 4 by 4 box filter on every 16th point (deleting the other 15 out of 16 points)
- Sample at a higher resolution, but still store the same (lower) resolution image
- NB: As the number of points per pixel approaches infinity, it approximates the area coverage integral (a “smart” random distribution is often used... )
Point Sampling

4 by 4

Supersampling

Exact

Area Coverage
This Matters!
This Matters!

Jaggies

Anti-aliased
Sketch Your Scene
Question 1 (short/long)

- Sketch the most important parts first
- Model the most important parts in CG first too, in case you run out of time

- Is it hard to draw?
- Will it be hard to model?

- Does it make sense or need explanation?
- Will your CG scene make sense?

- Etc.