Lecture 2:

Drawing a Triangle
(+ the basics of sampling and anti-aliasing)

Interactive Computer Graphics
Stanford CS248, Winter 2020
Tunes

Alabama Shakes
“Sound & Color”
(Sound & Color)

“Sometimes it’s helpful to think about sound when you’re trying to understand the artifacts in a 2D image.”
- Brittany Howard
Today: drawing a triangle to a frame buffer

"Triangle Rasterization"

Input:
projected position of triangle vertices: $P_0, P_1, P_2$

Output:
set of pixels “covered” by the triangle
Why triangles?
Triangles are a basic block for creating more complex shapes and surfaces.
Detailed surface modeled by tiny triangles (one pixel)
Why triangles?
- Most basic polygon
  - Can break up other polygons into triangles
  - Allows for optimizing one implementation

- Triangles have unique properties
  - Guaranteed to be planar
  - Well-defined interior
  - Well-defined method for interpolating values at vertices over triangle (a topic of a future lecture)
What does it mean for a pixel to be covered by a triangle?

Question: which triangles “cover” this pixel?
One option: compute fraction of pixel area covered by triangle, then color pixel according to this fraction.

Intuition: if triangle covers 10% of pixel, then pixel should be 10% red?
Analytical coverage schemes get tricky when considering occlusion of one triangle by another.

Interpenetration of triangles: even trickier.

Pixel covered by triangle 1, other half covered by triangle 2.

Two regions of triangle 1 contribute to pixel. One of these regions is not even convex.
Today we will draw triangles using a simple method: point sampling

(let’s consider sampling in 1D first)
Consider a 1D signal: $f(x)$
Sampling: taking measurements of a signal

Below: five measurements ("samples") of $f(x)$
Audio file: stores samples of a 1D signal

Audio is often sampled at 44.1 KHz
Sampling a function

- Evaluating a function at a point is sampling the function’s value

- We can discretize a function by periodic sampling

  ```
  for(int x = 0; x < xmax; x++)
    output[x] = f(x);
  ```

- Sampling is a core idea in graphics. In this class we’ll sample time (1D), area (2D), angle (2D), volume (3D), etc …
Reconstruction: given a set of samples, how might we attempt to reconstruct the original signal $f(x)$?
Reconstruction: given a set of samples, how might we attempt to reconstruct the original signal $f(x)$?
Piecewise constant approximation

\[ f_{recon}(x) = \text{value of sample closest to } x \]

\[ f_{recon}(x) \text{ approximates } f(x) \]
Piecewise linear approximation

\[ f_{\text{recon}}(x) = \text{linear interpolation between values of two closest samples to } x \]
How can we represent the signal more accurately?

Sample signal more densely (increase sampling rate)
More accurate reconstructions result from denser sampling

---

- Dotted line = reconstruction via nearest neighbor
- Dashed line = reconstruction via linear interpolation
Drawing a triangle by 2D sampling
Define binary function: \textit{inside}(\text{tri}, x, y)

\[\text{inside}(t, x, y) = \begin{cases} 
1 & (x, y) \text{ in triangle } t \\ 
0 & \text{otherwise} 
\end{cases}\]
Sampling the binary function: $\text{inside}(\text{tri}, x, y)$

Example:
Here I chose the sample position to be at the pixel center.

$\bullet = \text{triangle covers sample, should color in pixel}$

$\nabla = \text{triangle does not cover sample, do not color in pixel}$
Sample coverage at pixel centers
Sample coverage at pixel centers
Rasterization = sampling a 2D indicator function

\[
\text{for (int } x = 0; x < \text{ xmax; } x++)
\]
\[
\quad \text{for (int } y = 0; y < \text{ ymax; } y++)
\]
\[
\quad \text{image}[x][y] = f(x + 0.5, y + 0.5);
\]

- Rasterize triangle \text{tri} by sampling the function
  \[
  f(x,y) = \text{inside(tri,x,y)}
  \]
Evaluating inside(tri,x,y)
Triangle = intersection of three half planes
Point slope form of a line

(You might have seen this in high school)

\[ y - y_0 = m(x - x_0) \]

\[ m = \frac{y_1 - y_0}{x_1 - x_0} \]
Each line defines two half-planes

- Implicit line equation
  - \( L(x, y) = Ax + By + C \)
  - On the line: \( L(x, y) = 0 \)
  - “Negative side” of line: \( L(x, y) < 0 \)
  - “Positive” side of line: \( L(x, y) > 0 \)
Line Tangent Vector

\[ T = P_1 - P_0 = (x_1 - x_0, y_1 - y_0) \]
Line equation derivation

General Perpendicular Vector in 2D

Perp($x$, $y$) = ($y$, $-x$)
Line equation derivation

\[ N = \text{Perp}(T') = (y_1 - y_0, -(x_1 - x_0)) \]
Line equation derivation

Now consider a point \( P = (x, y) \). Which side of the line is it on?

\[
V = P - P_0 = (x - x_0, y - y_0)
\]
Line equation derivation

\[ L(x, y) = V \cdot N = -(y - y_0)(x_1 - x_0) + (x - x_0)(y_1 - y_0) \]
\[ = (y_1 - y_0)x - (x_1 - x_0)y + y_0(x_1 - x_0) - x_0(y_1 - y_0) \]
\[ = Ax + By + C \]
Line equation tests

\[ L(x, y) = V \cdot N > 0 \]
Line equation tests

\[ L(x, y) = V \cdot N = 0 \]
Line equation tests

\[ L(x, y) = V \cdot N < 0 \]
Point-in-triangle test

\[ P_i = (X_i, Y_i) \]

\[ dX_i = X_{i+1} - X_i \]
\[ dY_i = Y_{i+1} - Y_i \]

\[ L_i(x, y) = (x - X_i) dY_i - (y - Y_i) dX_i \]
\[ = A_i x + B_i y + C_i \]

\[ L_i(x, y) = 0 \text{ : point on edge} \]
\[ > 0 \text{ : outside edge} \]
\[ < 0 \text{ : inside edge} \]
Point-in-triangle test

\[ P_i = (X_i, Y_i) \]

\[ dX_i = X_{i+1} - X_i \]

\[ dY_i = Y_{i+1} - Y_i \]

\[ L_i(x, y) = (x - X_i) dY_i - (y - Y_i) dX_i \]

\[ = A_i x + B_i y + C_i \]

\[ L_i(x, y) = 0 : \text{point on edge} \]

\[ > 0 : \text{outside edge} \]

\[ < 0 : \text{inside edge} \]

\[ L_1(x, y) < 0 \]
Point-in-triangle test

\[ P_i = (X_i, Y_i) \]

\[ dX_i = X_{i+1} - X_i \]
\[ dY_i = Y_{i+1} - Y_i \]

\[ L_i(x, y) = (x - X_i) dY_i - (y - Y_i) dX_i = A_i x + B_i y + C_i \]

\[ L_i(x, y) = 0 : \text{point on edge} \]
\[ > 0 : \text{outside edge} \]
\[ < 0 : \text{inside edge} \]

\[ L_2(x, y) < 0 \]
Point-in-triangle test

Sample point \( s = (sx, sy) \) is inside the triangle if it is inside all three edges.

\[
inside(sx, sy) = \\
L_0(sx, sy) < 0 \&\& \\
L_1(sx, sy) < 0 \&\& \\
L_2(sx, sy) < 0;
\]

Note: actual implementation of \( inside(sx, sy) \) involves \( \leq \) checks based on the triangle coverage edge rules (see next slides)

Sample points inside triangle are highlighted red.
Edge cases (literally)

Is this sample point covered by triangle 1? or triangle 2? or both?
OpenGL/Direct3D edge rules

- When edge falls directly on a screen sample point, the sample is classified as within triangle if the edge is a “top edge” or “left edge”
  - Top edge: horizontal edge that is above all other edges
  - Left edge: an edge that is not exactly horizontal and is on the left side of the triangle. (triangle can have one or two left edges)

Source: Direct3D Programming Guide, Microsoft
Finding covered samples: incremental triangle traversal

\( P_i = (X_i, Y_i) \)

\( dX_i = X_{i+1} - X_i \)
\( dY_i = Y_{i+1} - Y_i \)

\( L_i(x, y) = (x - X_i) dY_i - (y - Y_i) dX_i = A_i x + B_i y + C_i \)

\( L_i(x, y) = 0 \) : point on edge
\( > 0 \) : outside edge
\( < 0 \) : inside edge

Efficient incremental update:

\( dL_i(x+1,y) = L_i(x,y) + dY_i = L_i(x,y) + A_i \)
\( dL_i(x,y+1) = L_i(x,y) + dX_i = L_i(x,y) + B_i \)

Incremental update saves computation:
Only one addition per edge, per sample test

Many traversal orders are possible: backtrack, zig-zag, Hilbert/Morton curves
Modern approach: tiled triangle traversal

Traverse triangle in blocks
Test all samples in block against triangle in parallel

Advantages:
- Simplicity of parallel execution overcomes cost of extra point-in-triangle tests (most triangles cover many samples)
- Can skip sample testing work: entire block not in triangle ("early out"), entire block entirely within triangle ("early in")
- Additional advantages related to accelerating occlusion computations (not discussed today)

All modern graphics processors (GPUs) have special-purpose hardware for efficiently performing point-in-triangle tests
Recall: pixels on a screen

Each image sample sent to the display is converted into a little square of light of the appropriate color:
(a pixel = picture element)

* Thinking of each LCD pixel as emitting a square of uniform intensity light of a single color is a bit of an approximation to how real displays work, but it will do for now.
So, if we send the display this sampled signal
The display physically emits this signal

Given our simplified “square pixel” display assumption, we’ve effectively performed a piecewise constant reconstruction.
Compare: the continuous triangle function
What’s wrong with this picture?

Jaggies!
Jaggies (staircase pattern)

Is this the best we can do?
Reminder: how can we represent a sampled signal more accurately?

Sample signal more densely (increase sampling rate)
Point sampling: one sample per pixel
Supersampling: step 1

Take \( N \times N \) samples in each pixel

(but… how do we use these samples to drive a display, since there are four times more samples than display pixels!)
Supersampling: step 2

Average the NxN samples “inside” each pixel

Averaging down
Supersampling: step 2

Average the NxN samples “inside” each pixel
Supersampling: step 2

Average the NxN samples “inside” each pixel
Supersampling: result

This is the corresponding signal emitted by the display

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>25%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td>100%</td>
<td>100%</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>75%</td>
</tr>
</tbody>
</table>
Point sampling

One sample per pixel
4x4 supersampling + downsampling

Pixel value is average of 4x4 samples per pixel
Let’s understand what just happened in a more principled way
More examples of sampling artifacts in computer graphics
Jaggies (staircase pattern)
Moiré patterns in imaging

Full resolution image

1/2 resolution image: skip pixel odd rows and columns
Wagon wheel illusion (false motion)

Camera’s frame rate (temporal sampling rate) is too low for rapidly spinning wheel.

Created by Jesse Mason, https://www.youtube.com/watch?v=Q0wzkND_ooU
Sampling artifacts in computer graphics

- Artifacts due to sampling - “Aliasing”
  - Jaggies – sampling in space
  - Wagon wheel effect – sampling in time
  - Moire – undersampling images (and texture maps)
  - [Many more] …

- We notice this in fast-changing signals, when we sample the signal too sparsely
Sines and cosines

$\cos 2\pi x$

$\sin 2\pi x$
Frequencies

\[ \cos 2\pi f x \]

\[ f = \frac{1}{T} \]

\[ f = 1 \]

\[ \cos 2\pi x \]

\[ f = 2 \]

\[ \cos 4\pi x \]
Representing sound wave as a superposition of frequencies

\[ f_1(x) = \sin(\pi x) \]
\[ f_2(x) = \sin(2\pi x) \]
\[ f_4(x) = \sin(4\pi x) \]
\[ f(x) = 1.0 f_1(x) + 0.75 f_2(x) + 0.5 f_4(x) \]
Audio spectrum analyzer: representing sound as a sum of its constituent frequencies

Intensity of low-frequencies (bass)

Intensity of high frequencies
How to compute frequency-domain representation of a signal?
Fourier transform

Represent a function as a weighted sum of sines and cosines

\[ f(x) = \frac{A}{2} + \frac{2A \cos(t\omega)}{\pi} - \frac{2A \cos(3t\omega)}{3\pi} + \frac{2A \cos(5t\omega)}{5\pi} - \frac{2A \cos(7t\omega)}{7\pi} + \cdots \]
Fourier transform

- Convert representation of signal from primal domain (spatial/temporal) to frequency domain by projecting signal into its component frequencies

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

\[ = \int_{-\infty}^{\infty} f(x) (\cos(2\pi \omega x) - i\sin(2\pi \omega x)) \, dx \]

Recall:

\[ e^{ix} = \cos x + i \sin x \]

- 2D form:

\[ F(u, v) = \int \int f(x, y) e^{-2\pi i (ux + vy)} \, dx \, dy \]
Fourier transform decomposes a signal into its constituent frequencies

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

\[ f(x) = \int_{-\infty}^{\infty} F(\omega) e^{2\pi i \omega x} \, d\omega \]
Visualizing the frequency content of images

Visualization below is the 2D frequency domain equivalent of the 1D audio spectrum I showed you earlier.

Spatial domain result

Spectrum
Constant signal (in primal domain)
$\sin\left(\frac{2\pi}{32}\right)x$ — frequency $1/32$; 32 pixels per cycle

Max signal freq $= 1/32$

(0,0)

Spatial domain

Frequency domain
$$\sin\left(\frac{2\pi}{16}\right)x$$ — frequency 1/16; 16 pixels per cycle

Max signal freq = 1/16

Spatial domain

Frequency domain
$$\sin\left(\frac{2\pi}{16}\right)y$$

Spatial domain

Frequency domain
\[
\sin\left(\frac{2\pi}{32}\right)x \times \sin\left(\frac{2\pi}{16}\right)y
\]
\[ \exp(-r^2/16^2) \]
\[ \exp(-r^2 / 32^2) \]

Spatial domain

Frequency domain
\[ \exp(-x^2/32^2) \times \exp(-y^2/16^2) \]
Rotate 45 \[ \exp\left(-\frac{x^2}{32^2}\right) \times \exp\left(-\frac{y^2}{16^2}\right) \]
Image filtering
(in the frequency domain)
Manipulating the frequency content of images
Low frequencies only (smooth gradients)

Spatial domain

Frequency domain
(after low-pass filter)
All frequencies above cutoff have 0 magnitude
Mid-range frequencies

Spatial domain

Frequency domain
(after band-pass filter)
Mid-range frequencies

Spatial domain

Frequency domain (after band-pass filter)
High frequencies (edges)

Spatial domain
(strongest edges)

Frequency domain
(after high-pass filter)
All frequencies below threshold have 0 magnitude
An image as a sum of its frequency components

\[ \text{Image} = \text{Texture} + \text{Outline} + \text{Silhouette} \]
Back to our problem of artifacts in images

Jaggies!
Higher frequencies need denser sampling sampling locations

Periodic sampling locations

Low-frequency signal: sampled adequately for reasonable reconstruction

High-frequency signal is insufficiently sampled: reconstruction incorrectly appears to be from a low frequency signal
Undersampling creates frequency “aliases”

High-frequency signal is insufficiently sampled: samples erroneously appear to be from a low-frequency signal.

Two frequencies that are indistinguishable at a given sampling rate are called “aliases”
Anti-aliasing idea: filter out high frequencies before sampling
Video: point vs antialiased sampling

Shutter Speed = 1/800s

Point in time

Shutter Speed = 1/30s

Motion blurred
Video: point sampling in time

30 fps video. 1/800 second exposure is sharp in time, causes time aliasing.

Credit: Aris & cams youtube, https://youtu.be/NoWwxTktoFs
Video: motion-blurred sampling

Shutter Speed = 1/30s

30 fps video. 1/30 second exposure is motion-blurred in time, reduces aliasing.
Rasterization: point sampling in 2D space

Note jaggies in rasterized triangle
(pixel values are either red or white: sample is in or out of triangle)
Rasterization: anti-aliased sampling

Pre-filter
(remove frequencies above Nyquist limit)

Sample

Note anti-aliased edges of rasterized triangle:
where pixel values take intermediate values
Point sampling

One sample per pixel
Anti-aliasing
Point sampling vs anti-aliasing

Jaggies

Pre-filtered
Anti-aliasing vs blurring an aliased result

Blurred Jaggies
(Sample then filter)

Pre-Filtered
(Filter then sample)
How much pre-filtering do we need to avoid aliasing?
Nyquist-Shannon theorem

- Consider a band-limited signal: has no frequencies above $\omega_0$
  - 1D: consider low-pass filtered audio signal
  - 2D: recall the blurred image example from a few slides ago

The signal can be perfectly reconstructed if sampled with period $T = 1 / 2 \omega_0$

- And reconstruction is performed using a “sinc filter”
  - Ideal filter with no frequencies above cutoff (infinite extent!)

\[
sinc(x) = \frac{\sin(\pi x)}{\pi x}
\]
Signal vs Nyquist frequency: example

\[ \sin\left(\frac{2\pi}{32}\right)x \] — frequency \( \frac{1}{32} \); 32 pixels per cycle

Spatial domain

Frequency domain

No Aliasing!

Max signal freq = \( \frac{1}{32} \)

Nyquist freq. = \( 2 \times \frac{1}{32} \) = \( \frac{1}{16} \)
Signal vs Nyquist frequency: example

\[ \sin\left(\frac{2\pi}{16}\right)x \] — frequency 1/16; 16 pixels per cycle

Max signal freq = 1/16

Nyquist freq.
= 2 * 1/16
= 1/8

Aliasing! (due to undersampling)
Reminder: Nyquist theorem

Theorem: We get no aliasing from frequencies in the signal that are less than the Nyquist frequency (which is defined as half the sampling frequency)

Consequence: sampling at twice the highest frequency in the signal will eliminate aliasing
Challenges of sampling-based approaches in graphics

- Our signals are not always band-limited in computer graphics. Why?

  Hint:

- Also, infinite extent of “ideal” reconstruction filter (sinc) is impractical for efficient implementations. Why?
Recall our anti-aliasing technique in the first half of lecture.
Filtering = convolution
Convolution

Signal: [1, 3, 5, 3, 7, 1, 3, 8, 6, 4]

Filter: [1, 2, 1]
Convolution

Signal: [1, 3, 5, 3, 7, 1, 3, 8, 6, 4]

Filter: [1, 2, 1]

Result: [12]

$1 \times 1 + 3 \times 2 + 5 \times 1 = 12$
Convolution

Signal: 1 3 5 3 7 1 3 8 6 4

Filter: 1 2 1

Result: 12 16

3x1 + 5x2 + 3x1 = 16
Convolution

Signal: 1 3 5 3 7 1 3 8 6 4

Filter: 1 2 1

Result: 5x1 + 3x2 + 7x1 = 18

Result: 12 16 18
Discrete 2D convolution

\[(f \ast g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x - i, y - j)\]

Consider \(f(i, j)\) that is nonzero only when: \(-1 \leq i, j \leq 1\)

Then:

\[(f \ast g)(x, y) = \sum_{i,j=-1}^{1} f(i, j)I(x - i, y - j)\]

And we can represent \(f(i,j)\) as a 3x3 matrix of values where:

\[f(i, j) = F_{i,j}\] (often called: “filter weights”, “filter kernel”)
Box filter (used in a 2D convolution)

Example: 3x3 box filter
2D convolution with box filter blurs the image

Original image

Blurred
(convolve with box filter)

Hmm… this reminds me of a low-pass filter…
Convolution theorem

Convolution in the spatial domain is equal to multiplication in the frequency domain, and vice versa.

Convolve

Spatial Domain

Fourier Transform

Frequency Domain

Inv. Fourier Transform
Convolution theorem

- Convolution in the spatial domain is equal to multiplication in the frequency domain, and vice versa

- Pre-filtering option 1:
  - Filter by convolution in the spatial domain

- Pre-filtering option 2:
  - Transform to frequency domain (Fourier transform)
  - Multiply by Fourier transform of convolution kernel
  - Transform back to spatial domain (inverse Fourier)
Box function = “low pass” filter

Spatial domain

Frequency domain
Wider filter kernel = lower frequencies

Spatial domain

Frequency domain
Wider filter kernel = lower frequencies

- As a filter is localized in the spatial domain, it spreads out in frequency domain

- Conversely, as a filter is localized in frequency domain, it spreads out in the spatial domain
How can we reduce aliasing error?

- Increase sampling rate (increase Nyquist frequency)
  - Higher resolution displays, sensors, framebuffers…
  - But: costly and may need very high resolution to sufficiently reduce aliasing

- Anti-aliasing
  - Simple idea: remove (or reduce) signal frequencies above the Nyquist frequency before sampling
  - How to filter out high frequencies before sampling?
Anti-aliasing by averaging values in pixel area

- Convince yourself the following are the same:

  - Option 1:
    - Convolve $f(x,y)$ by a 1-pixel box-blur
    - Then sample at every pixel

  - Option 2:
    - Compute the average value of $f(x,y)$ in the pixel
Anti-aliasing by computing average pixel value

In rasterizing one triangle, the average value inside a pixel area of $f(x,y) = \text{inside}(\text{tri},x,y)$ is equal to the area of the pixel covered by the triangle.
Putting it all together:
anti-aliasing via supersampling

Original signal
(with high frequency edge)

Dense sampling of signal
(supersampling)

Reconstructed signal
(averaging over pixel (via convolution) yields new signal with high frequencies removed)

Coarse sampling of reconstructed signal exhibits less aliasing
Today’s summary

- Drawing a triangle = sampling triangle/screen coverage
- Pitfall of sampling: aliasing
- Reduce aliasing by prefiltering signal
  - Supersample
  - Reconstruct via convolution (average coverage over pixel)
    - Higher frequencies removed
  - Sample reconstructed signal once per pixel

- There is much, much more to sampling theory and practice...
Acknowledgements

- Thanks to Ren Ng, Pat Hanrahan, Keenan Crane for slide materials