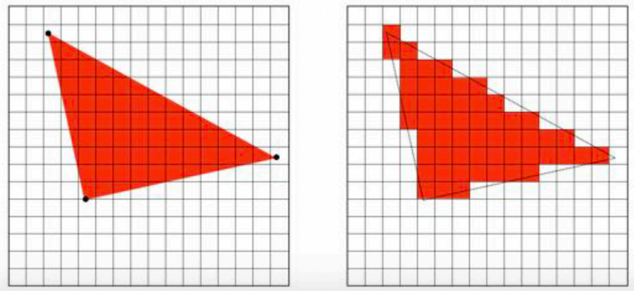
The Rasterization Pipeline

How to draw a triangle?

Two main ways to get stuff on a screen

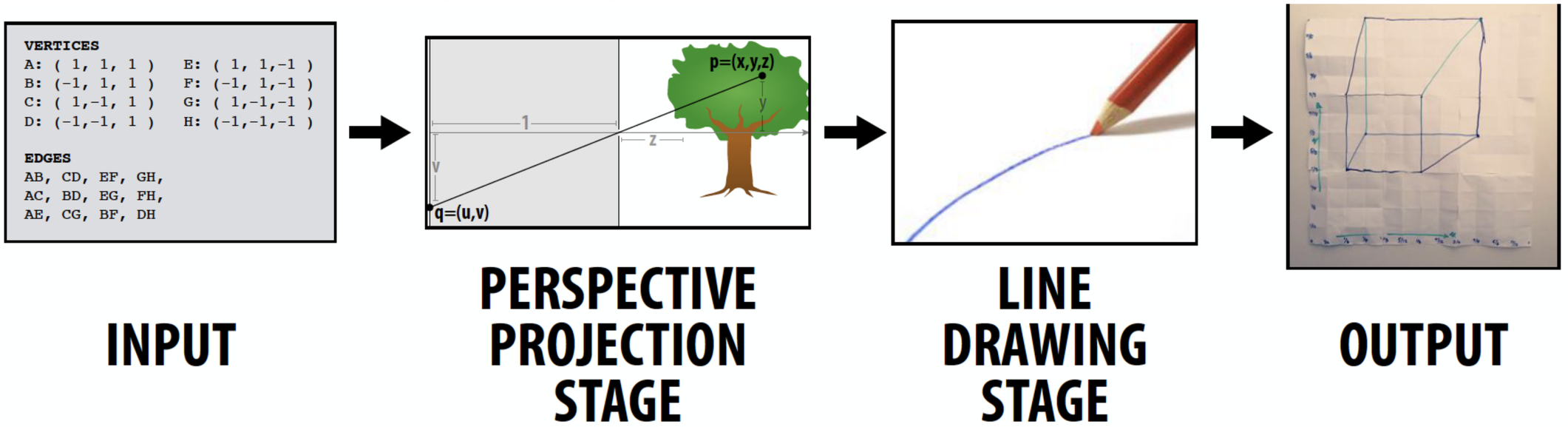
- **Rasterization** (today)
  - For each primitive (i.e., triangle), which pixels “light up”?
  - VERY fast! (Using modern GPUs)
  - Difficult to achieve photorealism (*not impossible*...)
  - Good for: fonts, vector graphics (e.g., cartoons), quick 3D previews, deep learning
- **Ray Tracing** (later)
  - For each *pixel*, which primitives are being viewed?
  - Easier to obtain photorealism
  - Slower



3D image generation pipeline

Talk about image generation in terms of a pipeline

- **Input** - object/s we want to draw
- **Stages** - sequence of transformations inputs -> outputs
- **Output** - the final image
  - e.g., our pipeline from our first lecture:

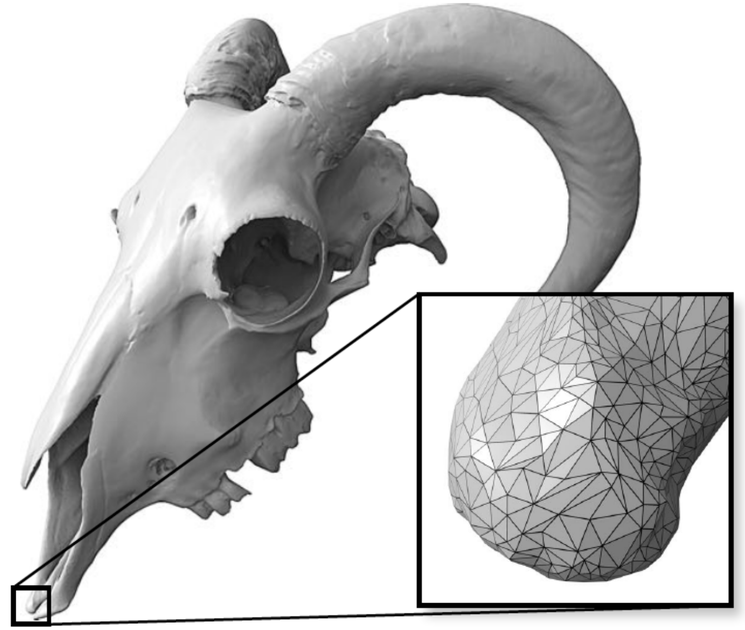


Rasterization Pipeline

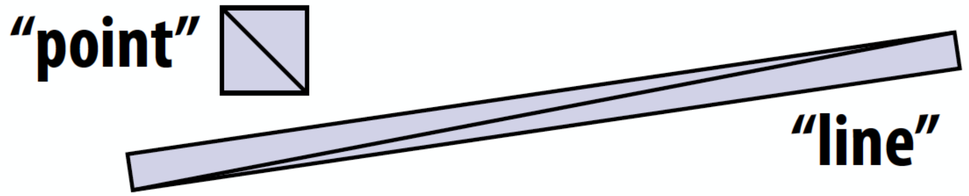
- Modern real time image generation based on rasterization
- **Input** is a 3D primitive — essentially all triangles
    - Possibly have additional attributes (e.g, color, texture)
- **Output** bitmap image (possibly w/ depth, alpha)

Triangles, triangles, triangles

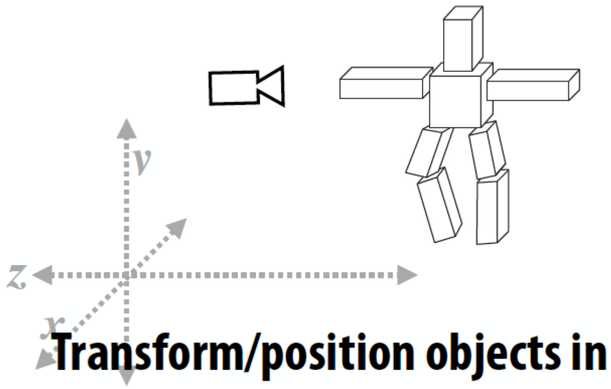
- Rasterization converts everything to a triangle
  - Even points and lines
- Can approximate any shape
- Always co-planar, well-defined normal
- Easy to interpolate values inside the triangle
  - Barycentric coordinates



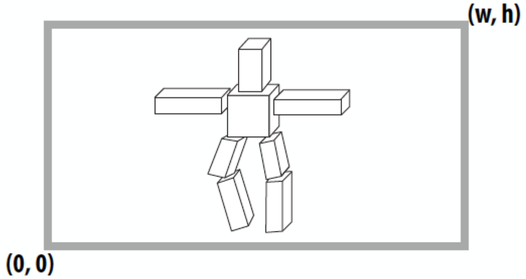
- Key reason: once everything is reduced to triangles, can focus on making an extremely well-optimized pipeline for drawing them



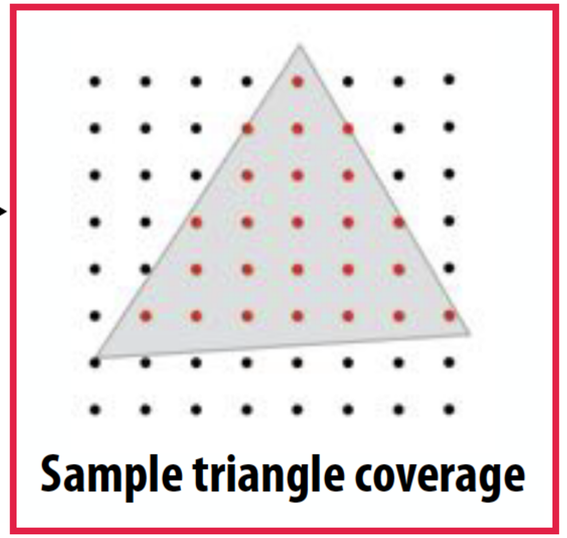
The Rasterization Pipeline



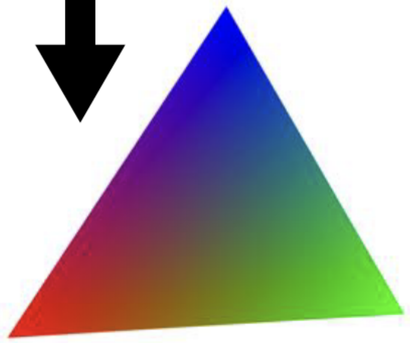
Transform/position objects in the world



Project objects onto the screen



Sample triangle coverage

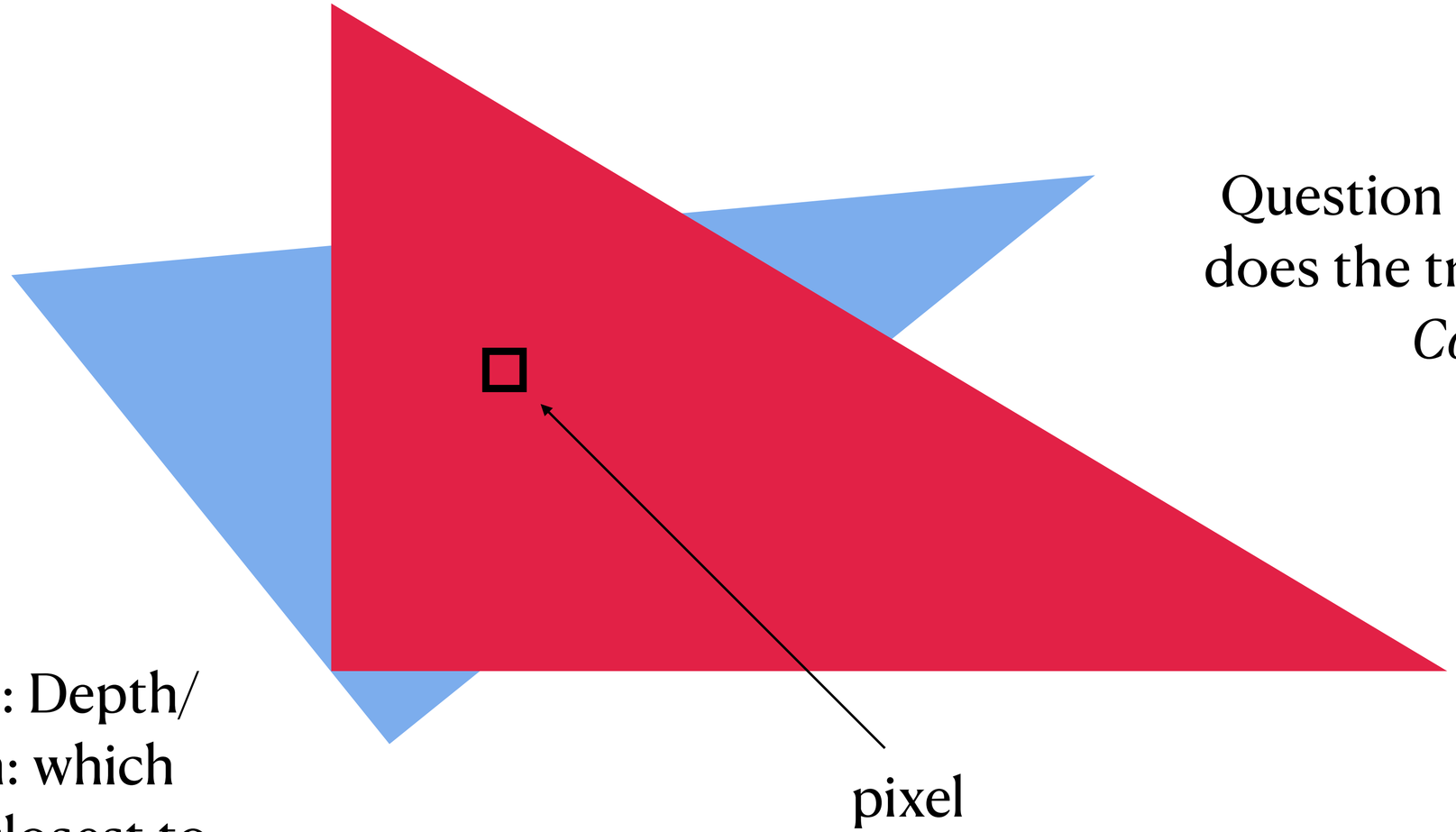


Interpolate triangle attributes at covered samples

Sample texture maps / evaluate shaders

Combine samples into final image (depth, alpha, . . .)

Draw triangles on an image

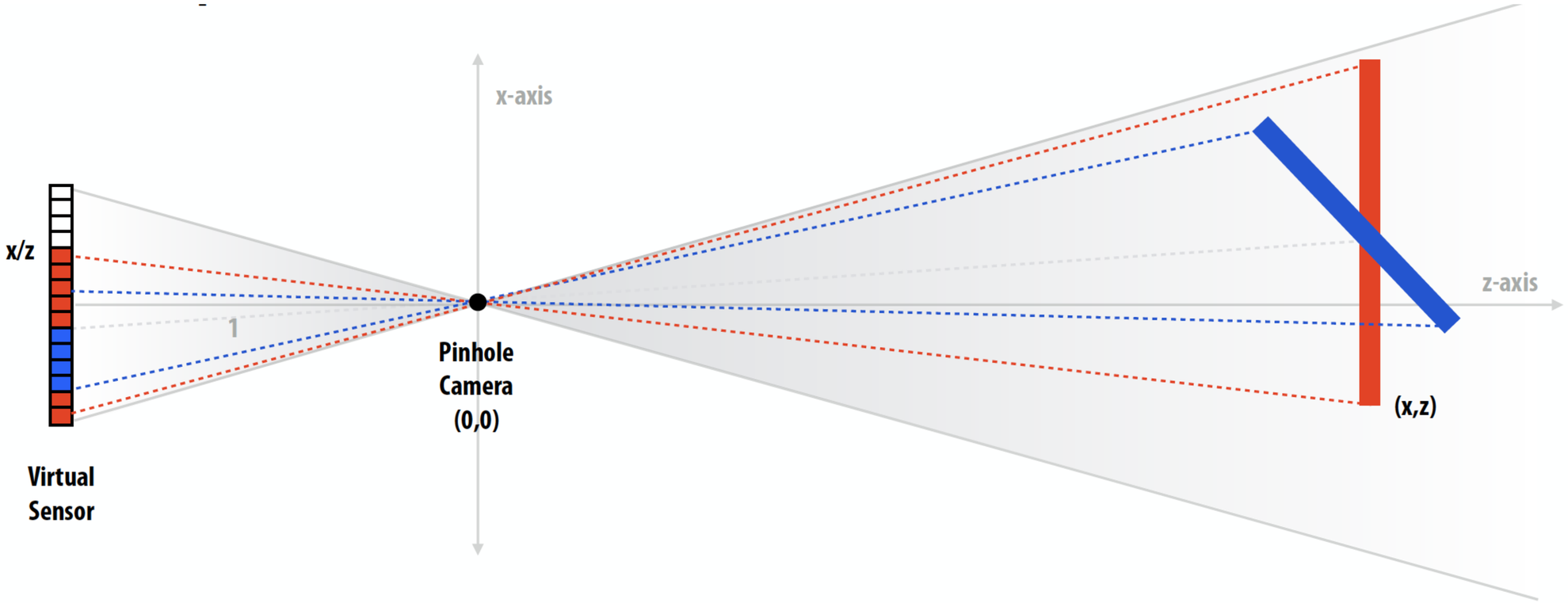


Question 1: Which pixels does the triangle overlap?
Coverage

Question 2: Depth/ Occlusion: which triangle is closest to the camera

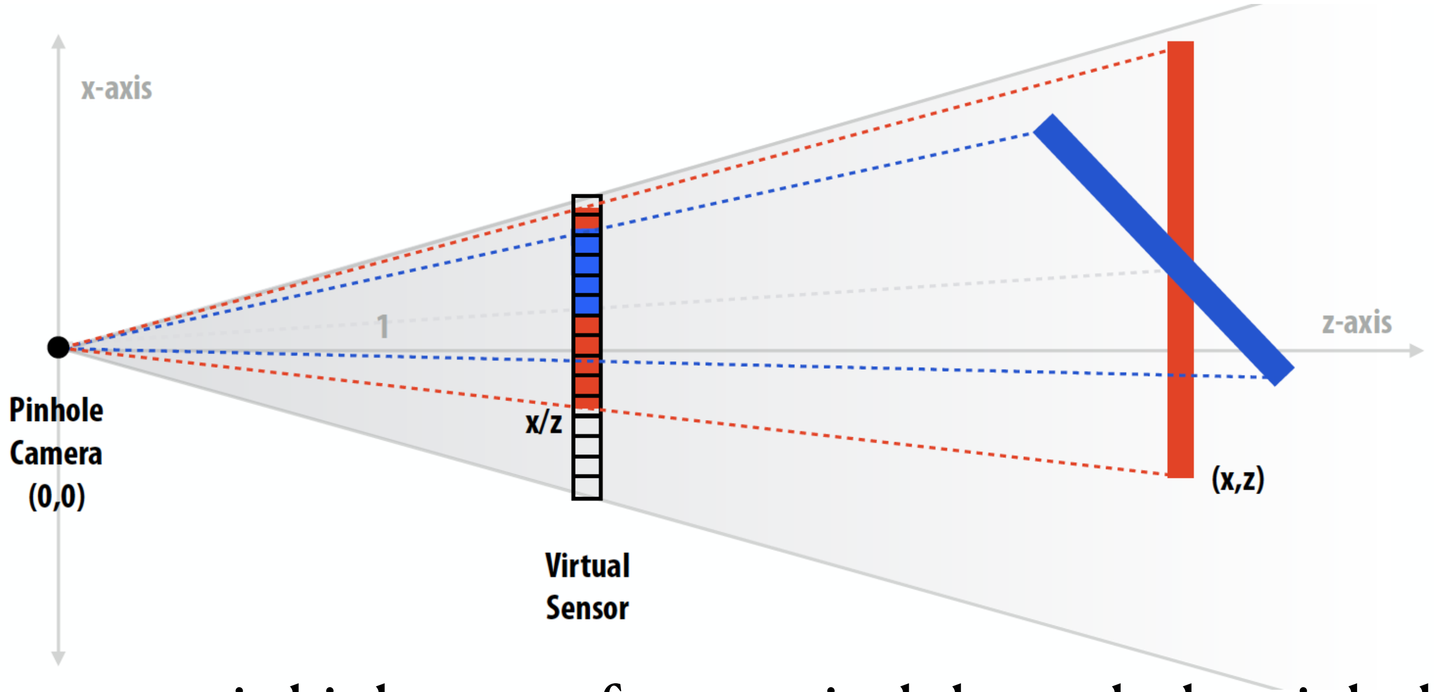
The visibility problem

Recall the pinhole camera



The visibility problem

Recall the pinhole camera... which we can simplify with a "virtual sensor"



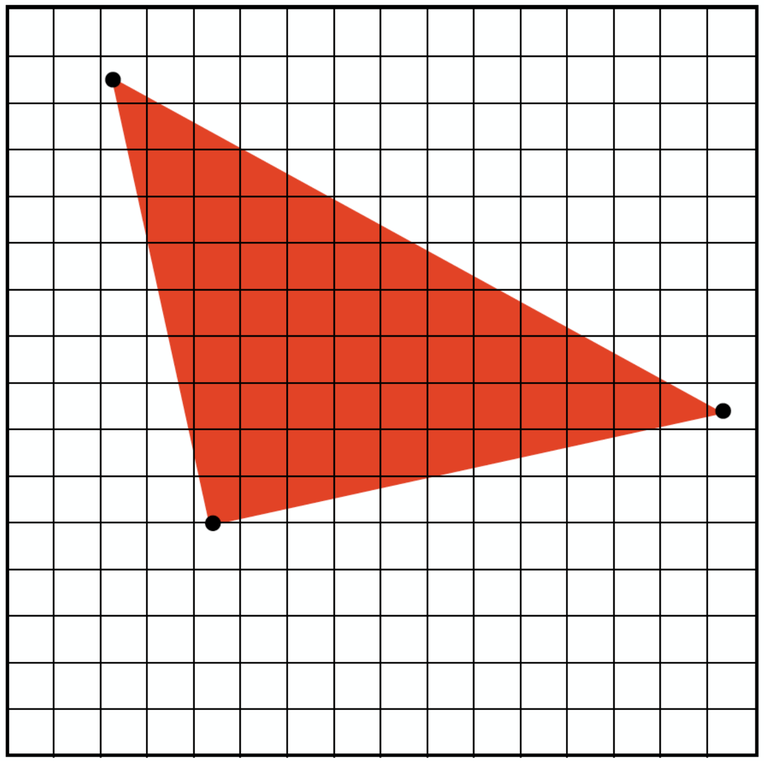
Visibility in terms of rays:

- **Coverage -** What scene geometry is hit by a ray from a pixel through the pinhole?
- **Occlusion -** Which object is the first hit along that ray?

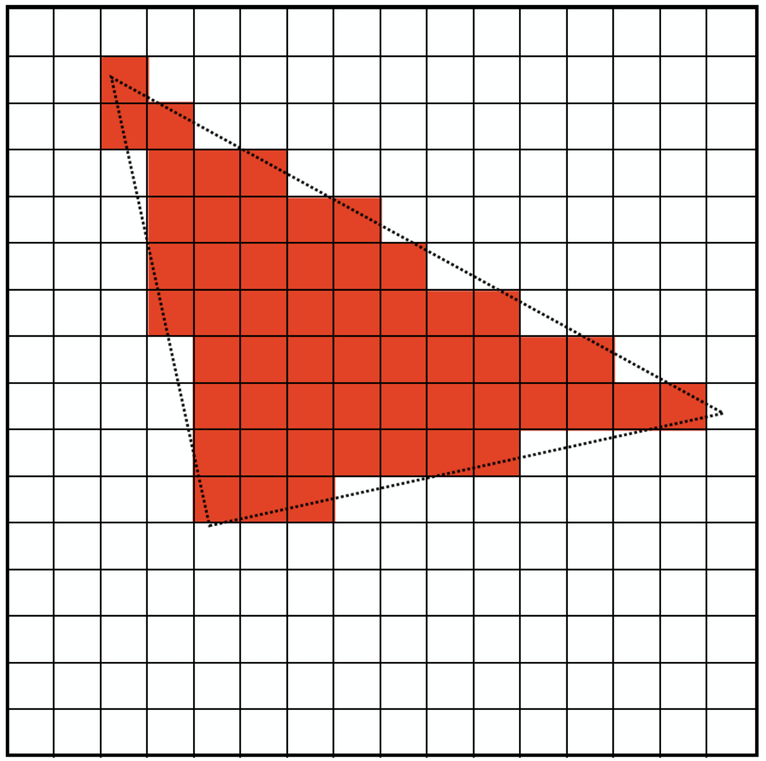
Computing Triangle Coverage

Which pixels does the triangle overlap?

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

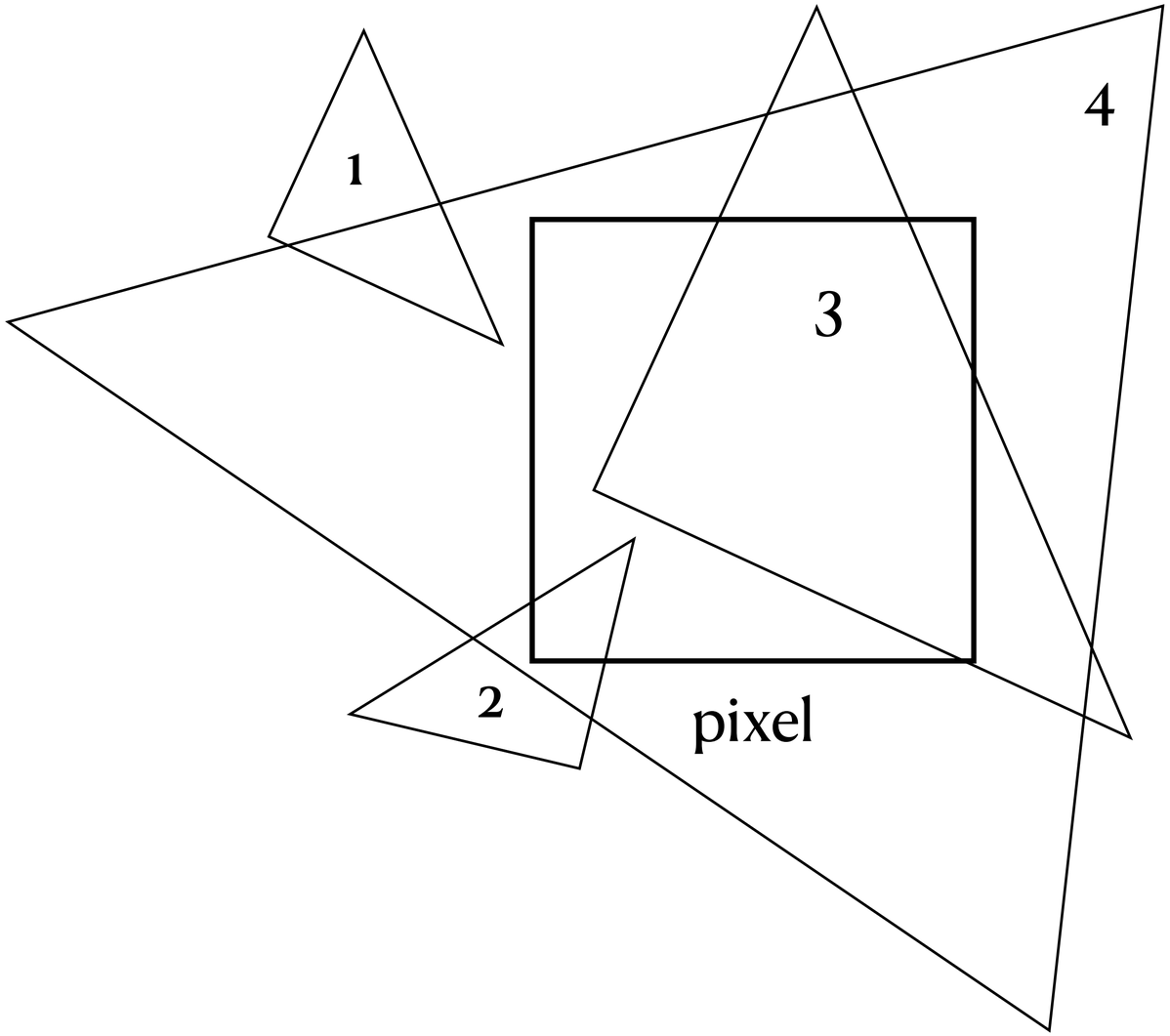


Output:
set of pixels "covered" by the triangle

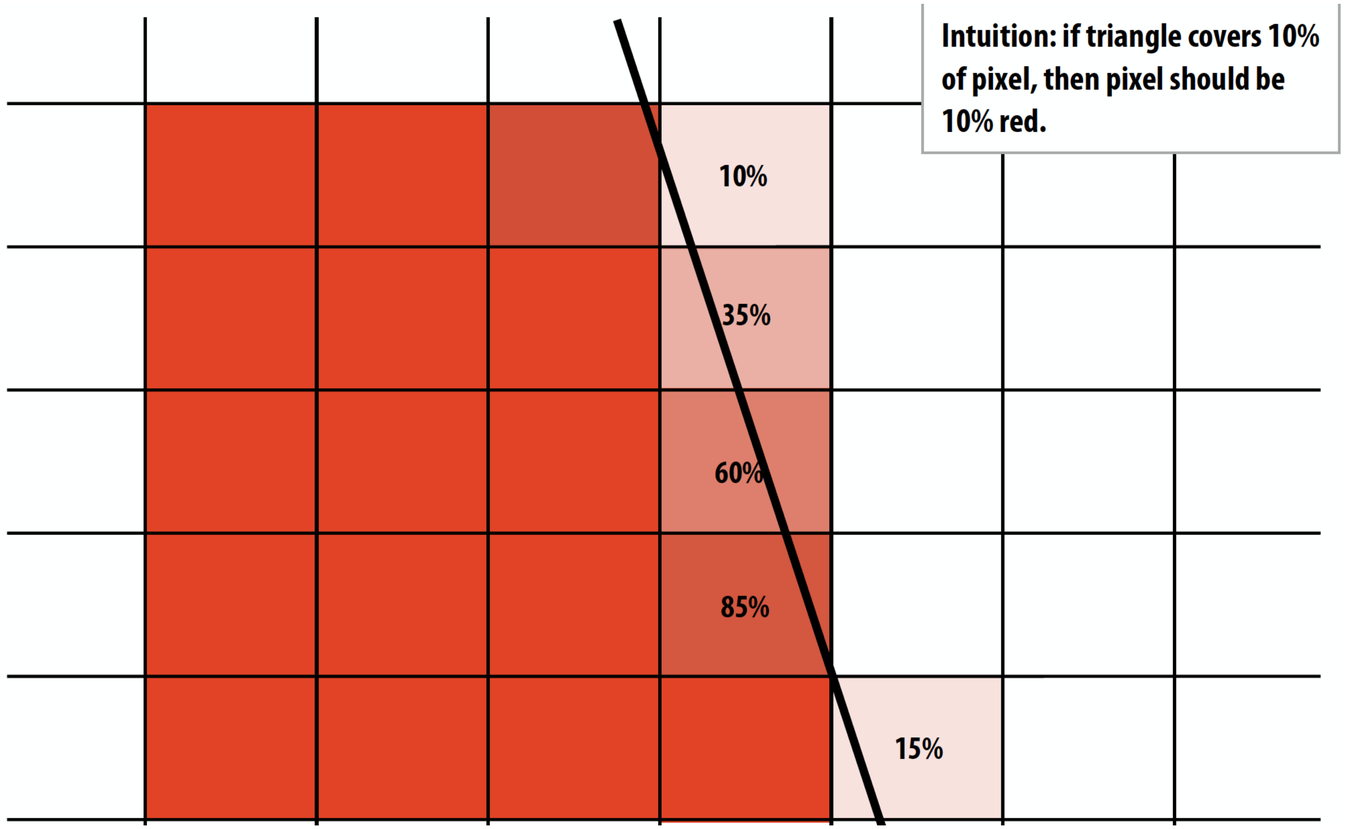


What does it mean for a pixel to be covered by a triangle

Q: what triangles cover this pixel?

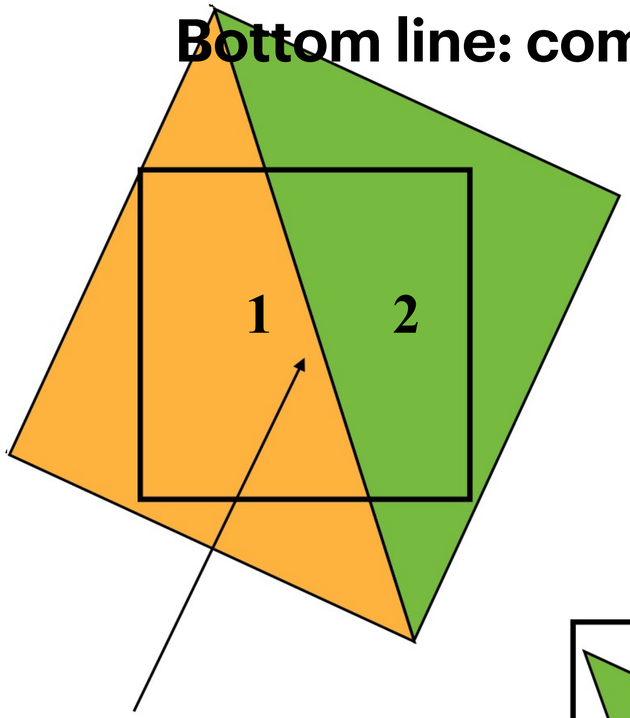


Color according to pixel area covered by triangle

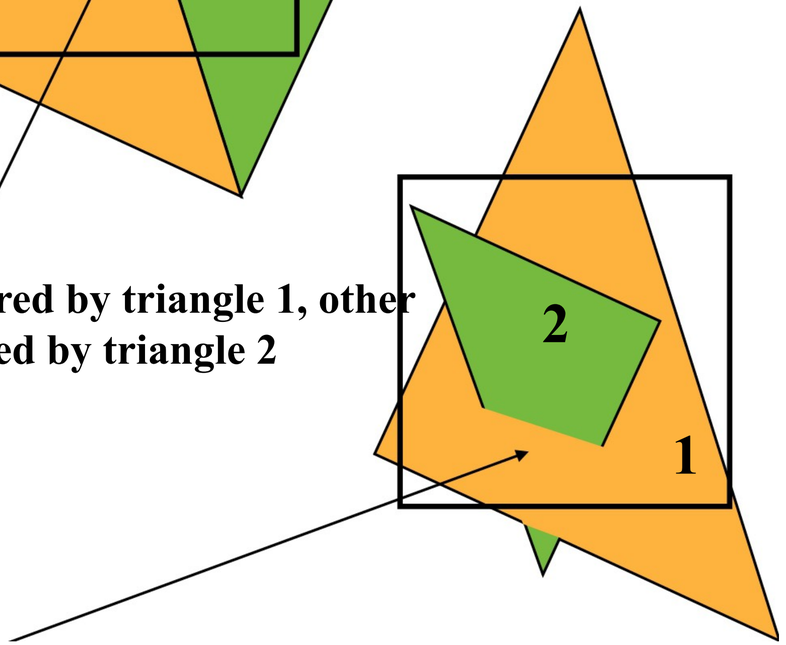


Coverage can be tricky

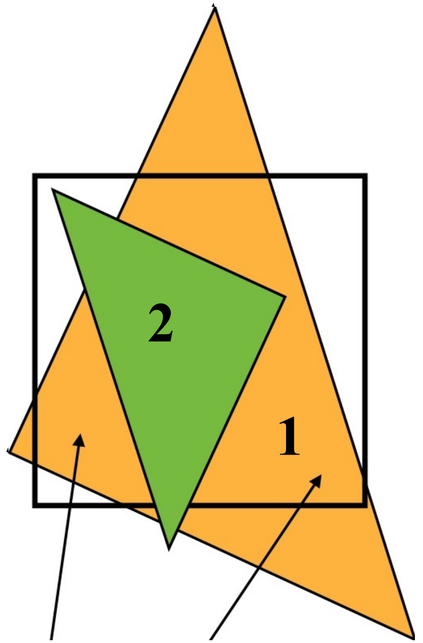
Bottom line: computing this exactly can get messy



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



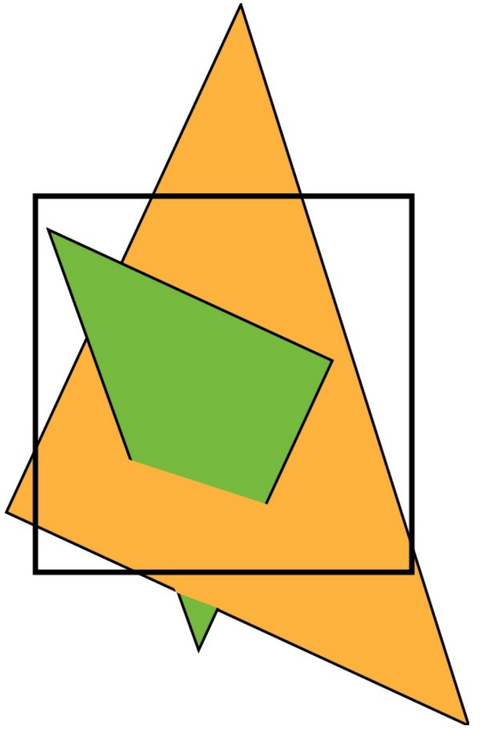
Interpenetration of triangles: even trickier

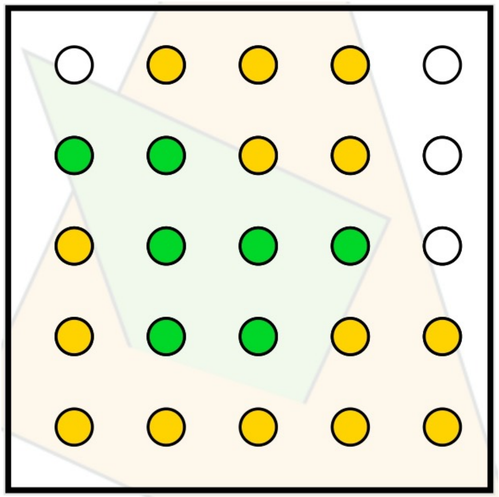


Two regions of triangle 1 contribute to pixel. One of these regions is not even convex.

Coverage via sampling

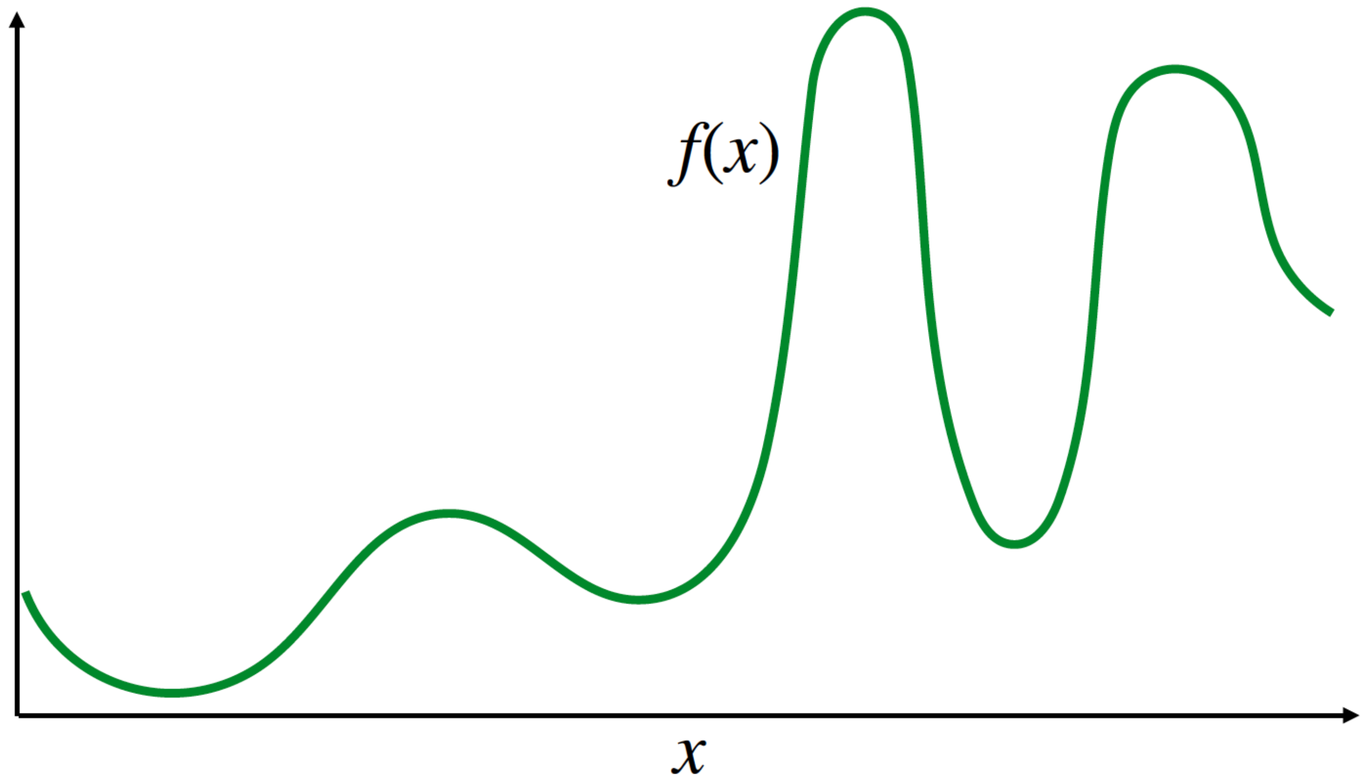
- Real scenes are complicated
  - Occlusion, transparency
- Computing exact fraction of coverage is not practical
- Instead it is a sampling problem
  - Don’t compute exact/analytical answer
  - Instead: test a collection of sample points
  - With enough points & sample locations, estimate is good enough





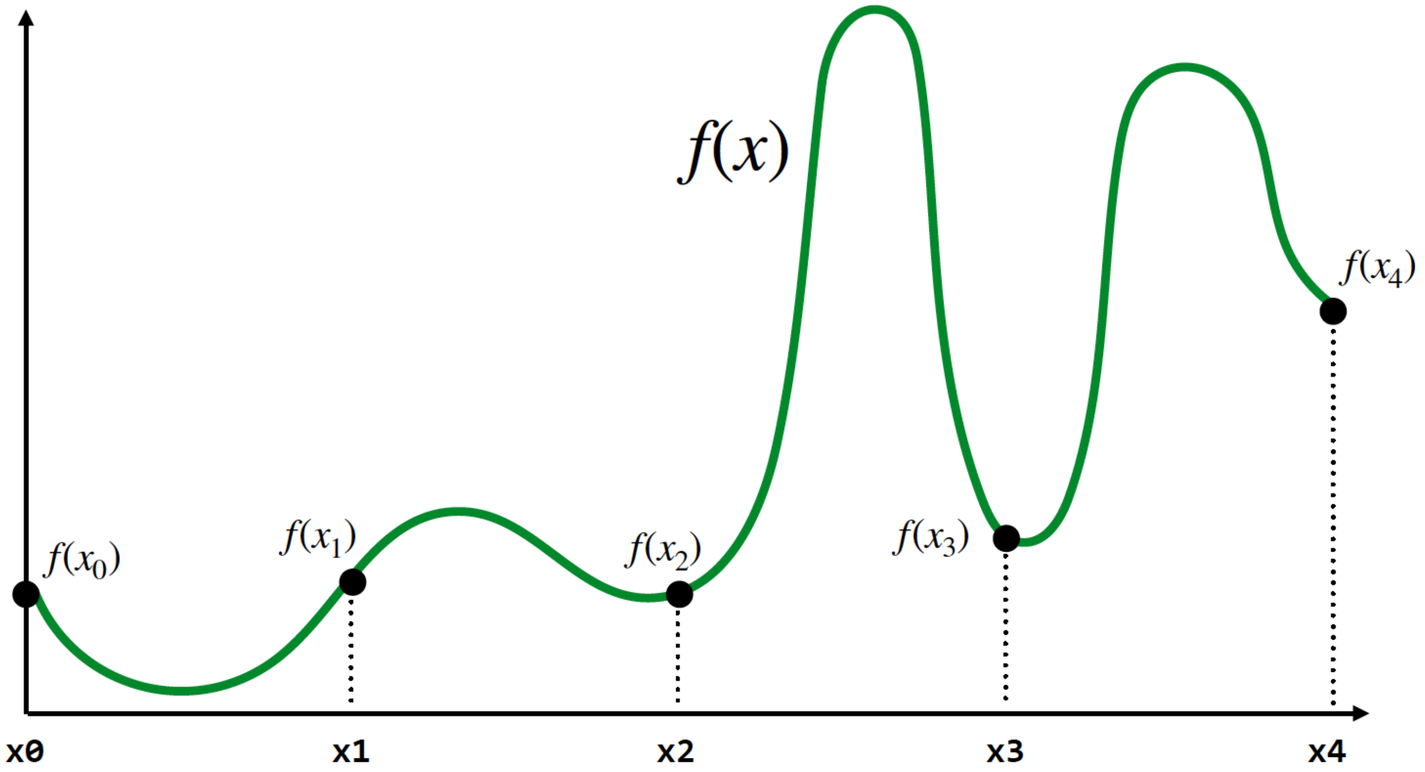
Sampling

Sampling 101: Sampling a 1D Signal



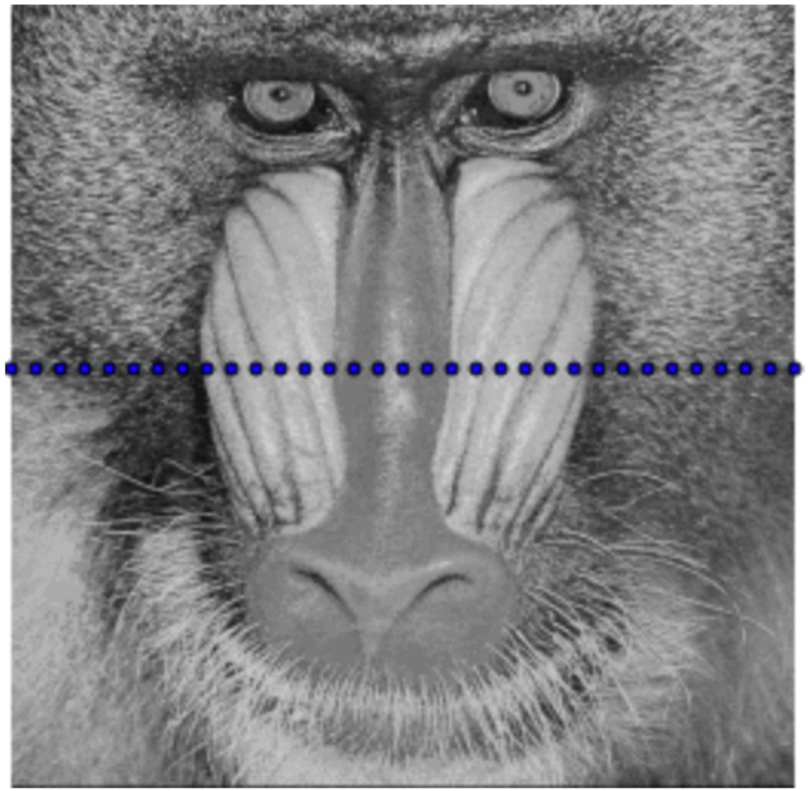
Sampling a 1D Signal

Take measurements at various intervals



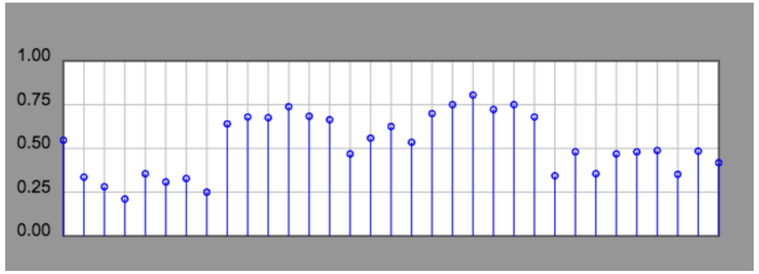
Measure value of f at x_o

Example: Image



Pixels are samples of a continuous function

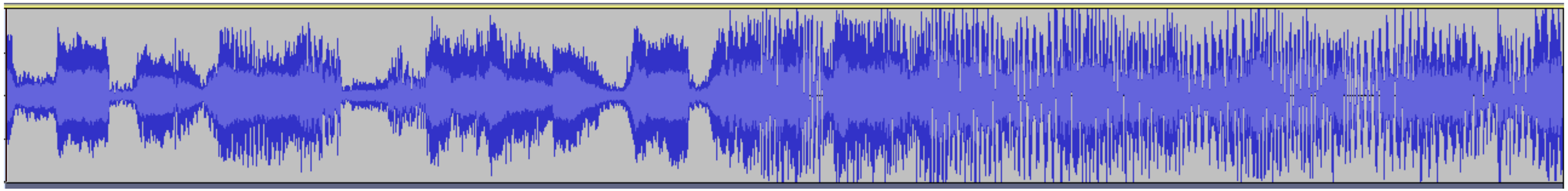
We can slice through the function and consider what happens in 1D



Example: stores samples of audio

- The amplitude of a signal over time

Amplitude

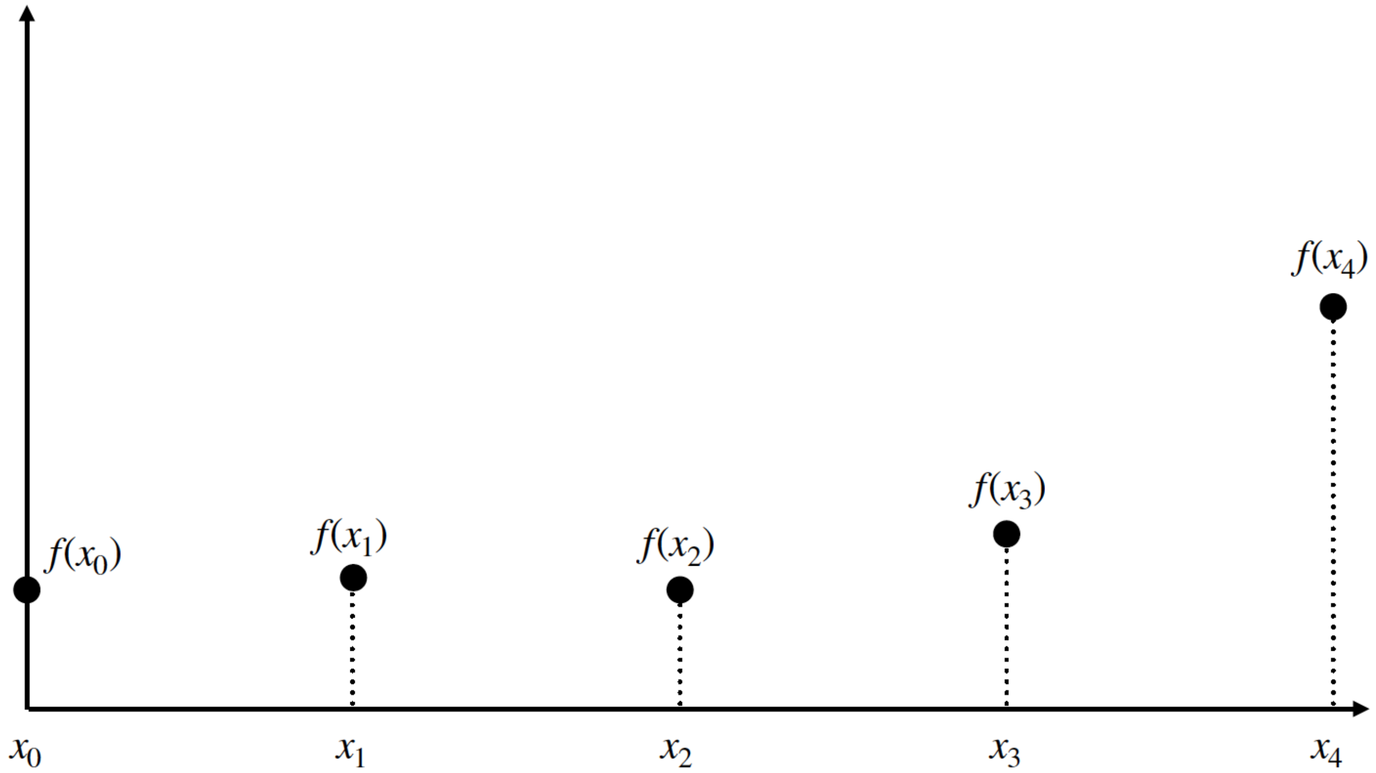


Time

CDs & MP3s are usually recorded at 44.1kHz - which means that every second, 44,100 samples were taken.

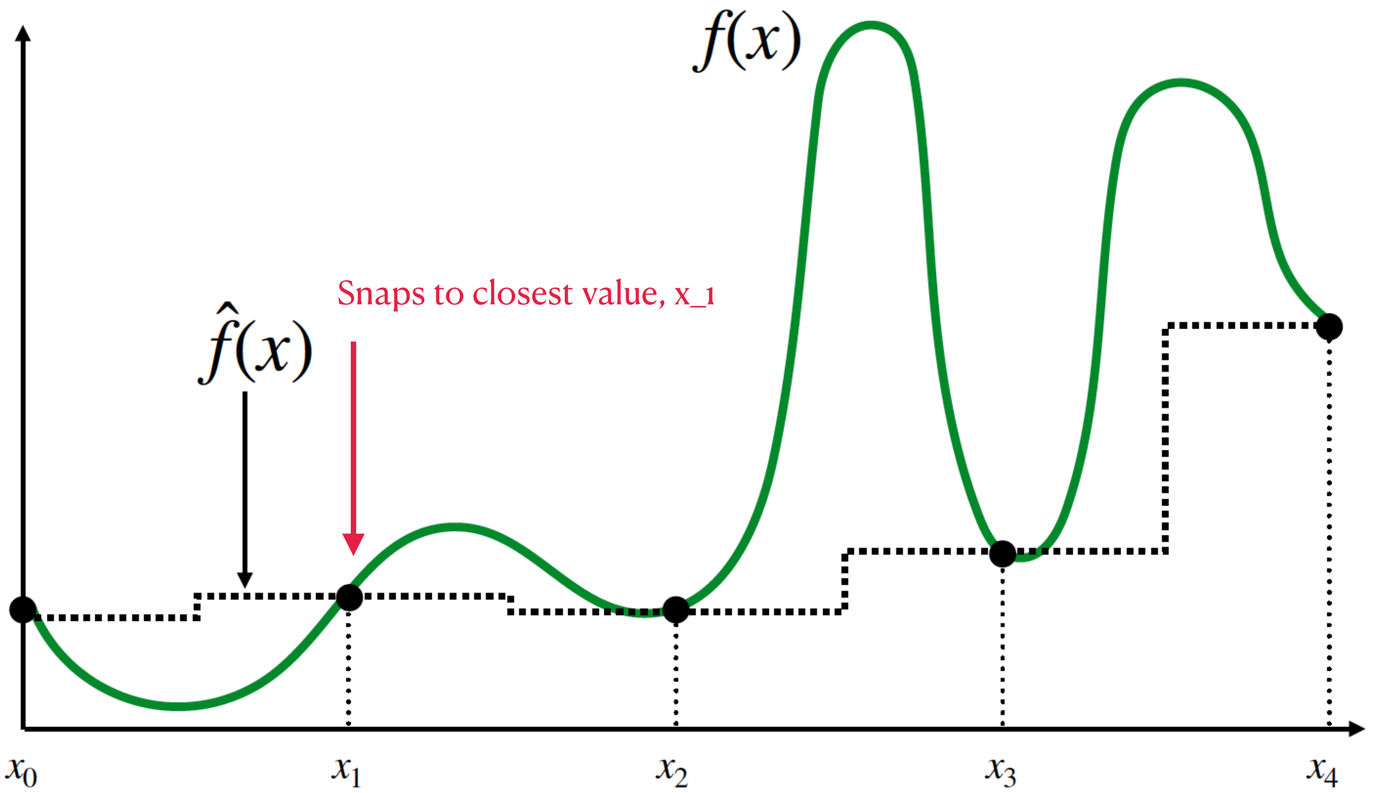
Given Samples, How to Reconstruct?

... want to restore the original signal f at every point x!



A Piecewise Constant Approximation

Assume the value of f(x) to be what the closest sampled value is



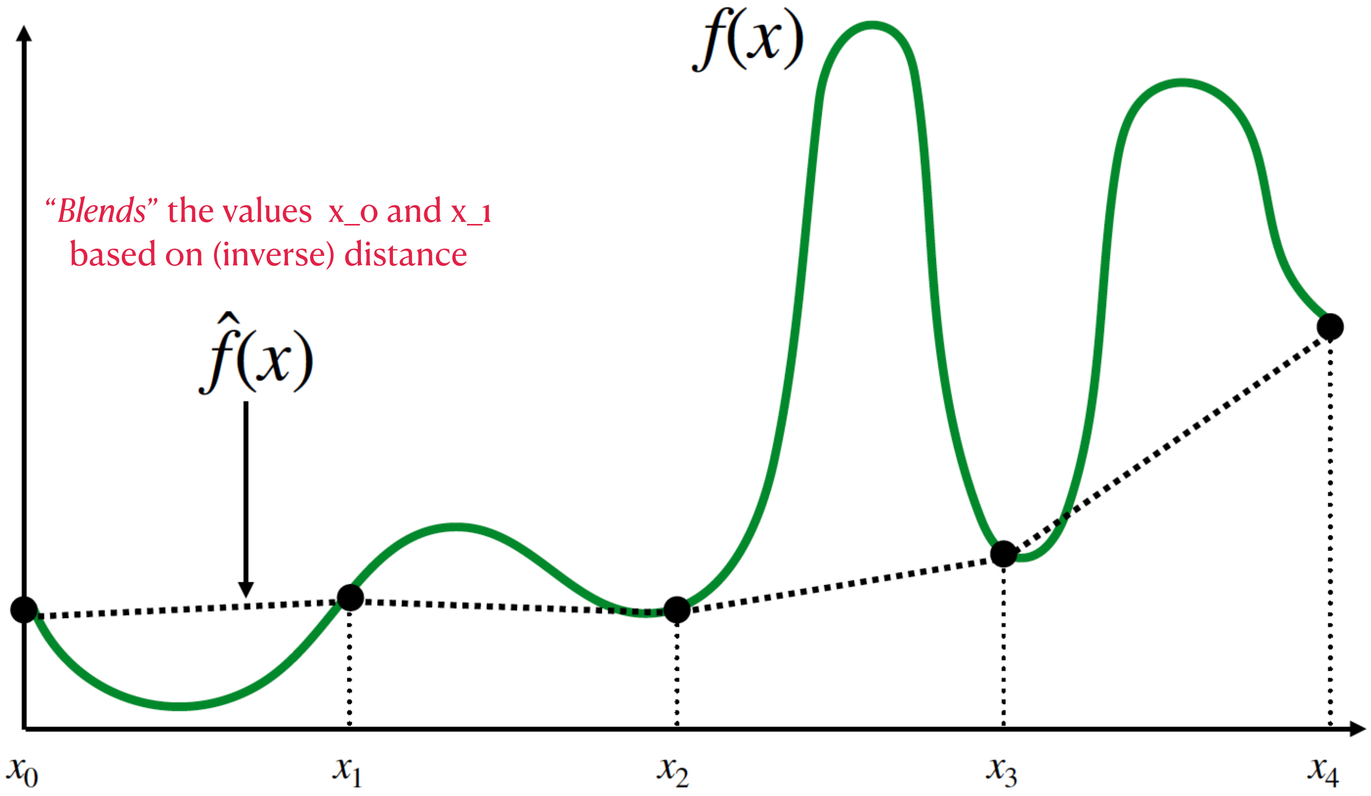
Build a new function

$\hat{f}(x)$

Which approximates the original function

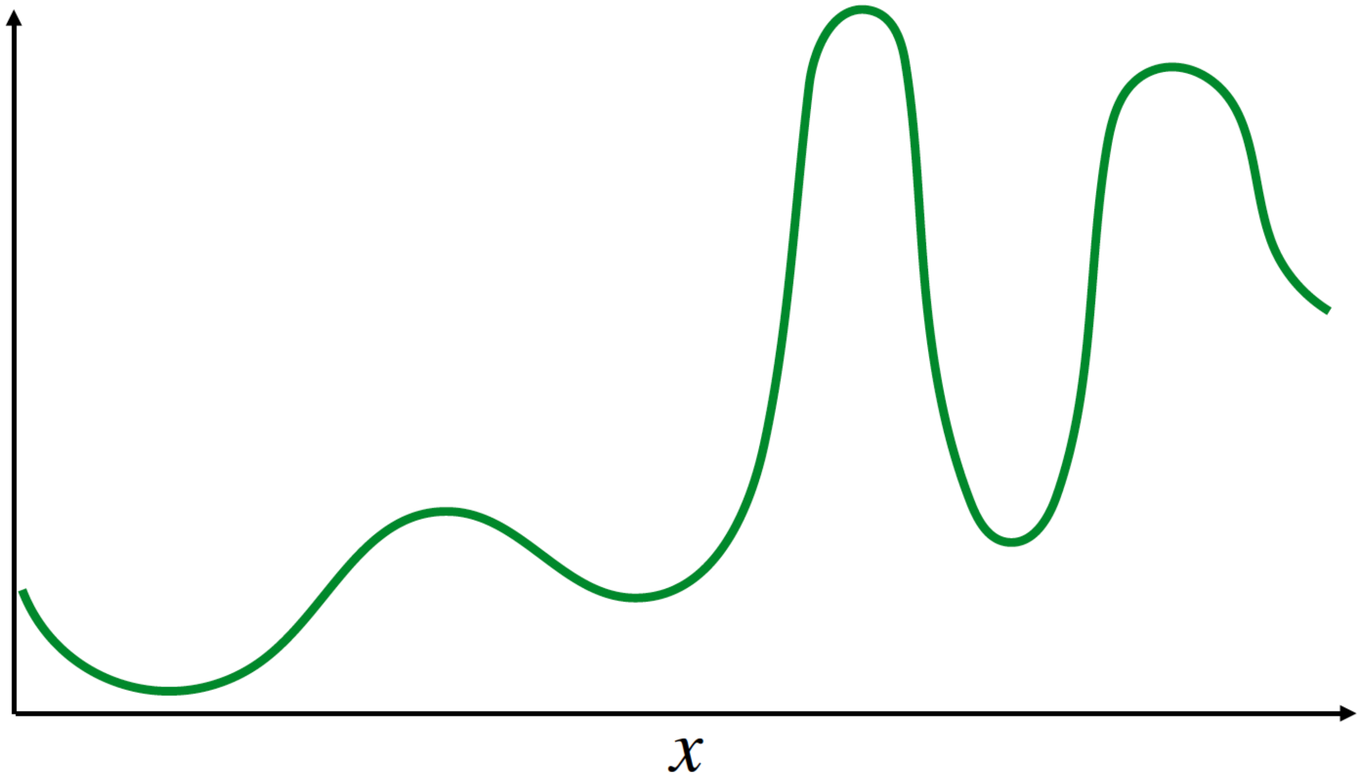
Piecewise linear approximation

Simple linear interpolation between adjacent samples



- Far from accurately representing the underlying signal…

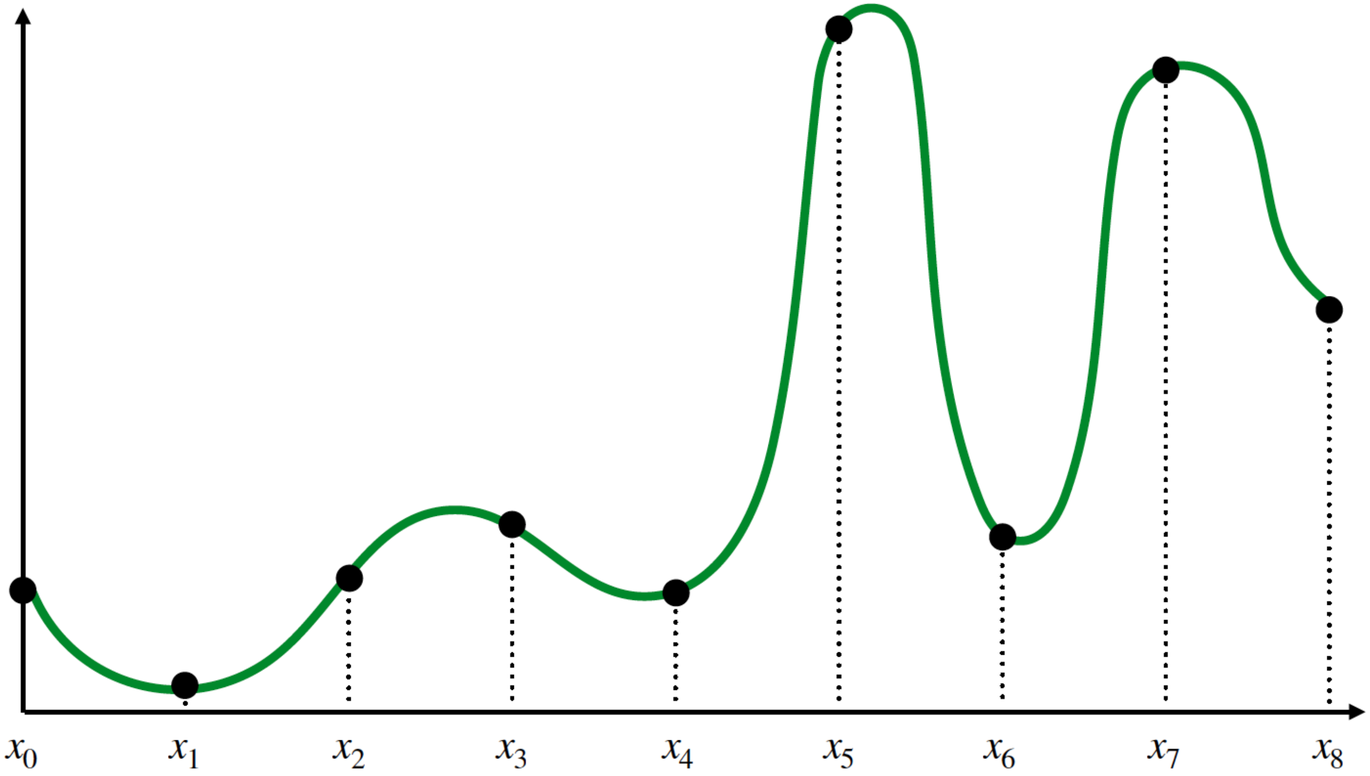
How can we represent the signal more accurately?



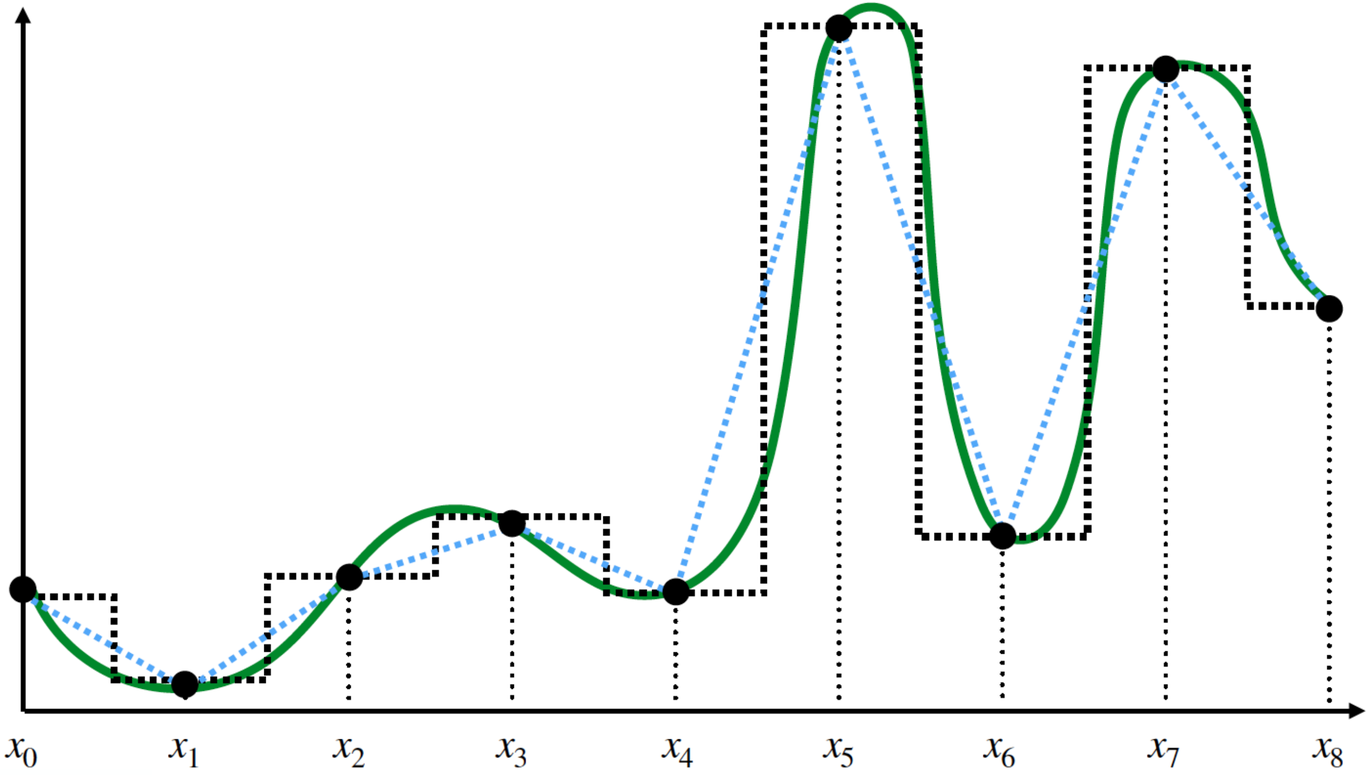
Better sampling of the signal

More accurate representation

- Sample signal more densely (increase sampling rate)



Approximation becomes more accurate



= reconstruction via nearest

= reconstruction via linear interpolation

2D sampling and reconstruction

- Measure the image at sample points
- Apply interpolation/reconstruction filter to approximate image

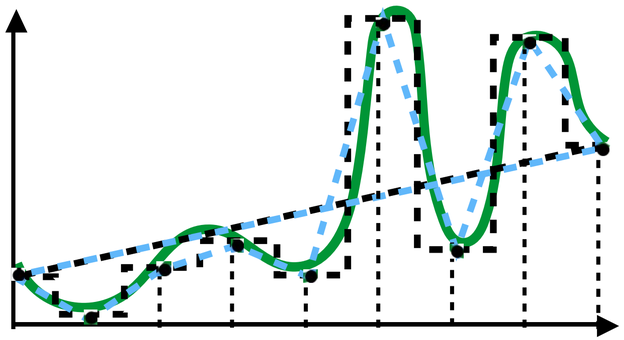
Original

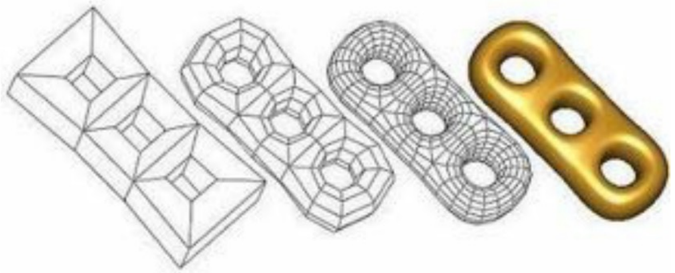
Piecewise constant
“nearest neighbor”

Bilinear Interpolation

Sampling Recap/Summary

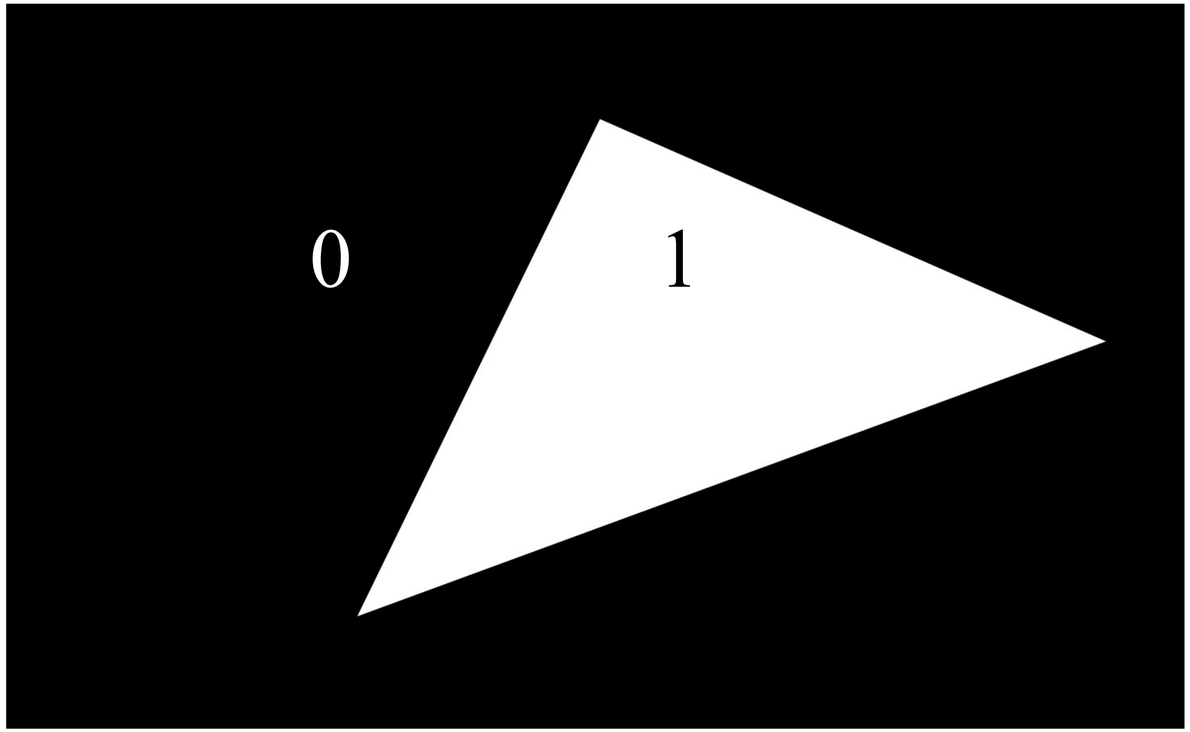
- Sampling = measurement of a signal
  - Encode signal as discrete set of samples
  - In principle, represent values at specific points
    (though hard to measure in reality!)
- Reconstruction = recovering a signal from a discrete set of samples
  - Construct a function that interpolates or approximates function values
  - E.g., piecewise constant/“nearest neighbor”, or piecewise linear
  - Many more possibilities! For all kinds of signals (audio, images, geometry…)





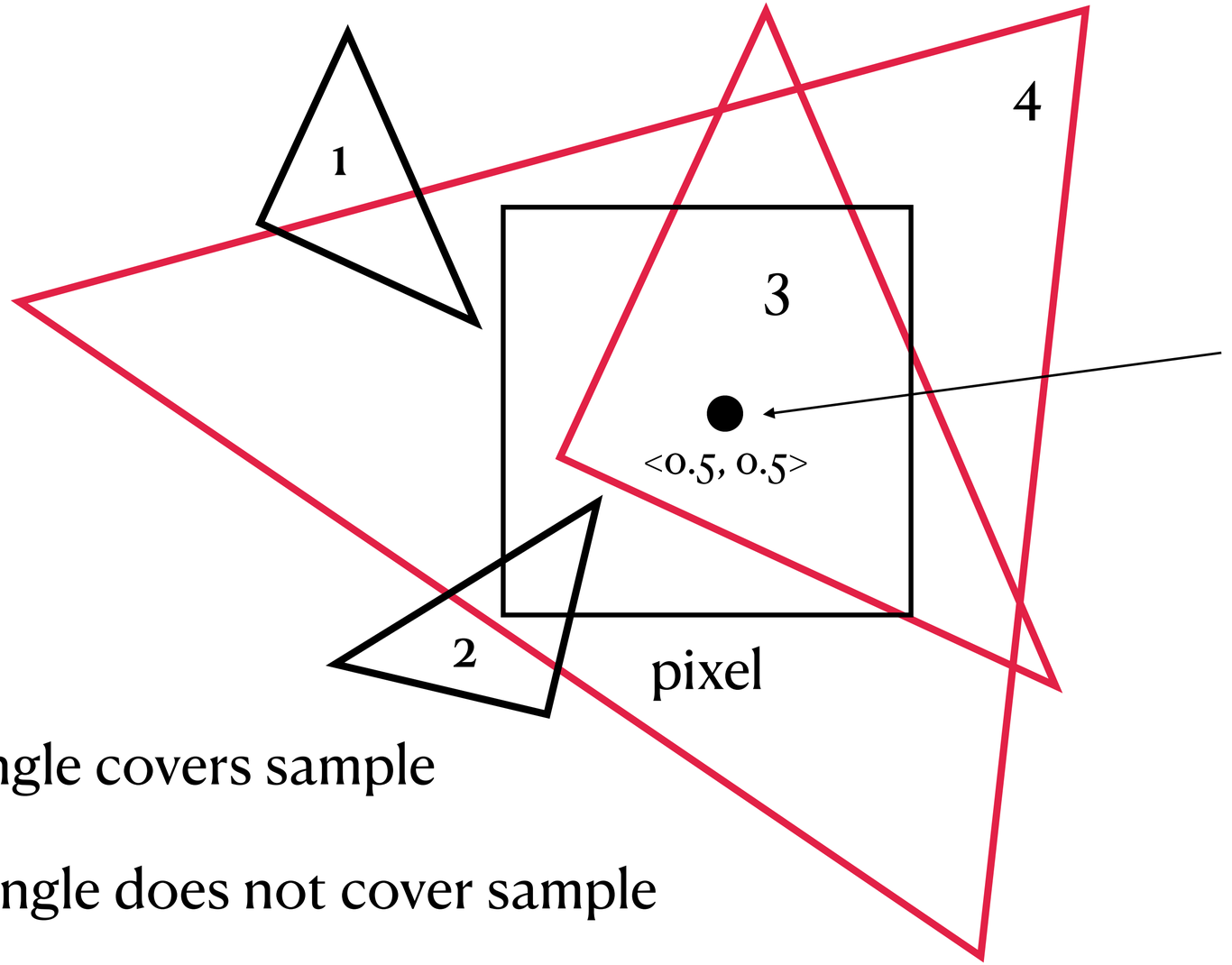
Rasterization: what function we are sampling?

$$\text{coverage}(x, y) := \begin{cases} 1, & \text{triangle contains point } (x, y) \\ 0, & \text{otherwise} \end{cases}$$



Rasterization: sample the coverage function onto the pixel grid

For every triangle, evaluate the coverage function (for that triangle) at the pixel center



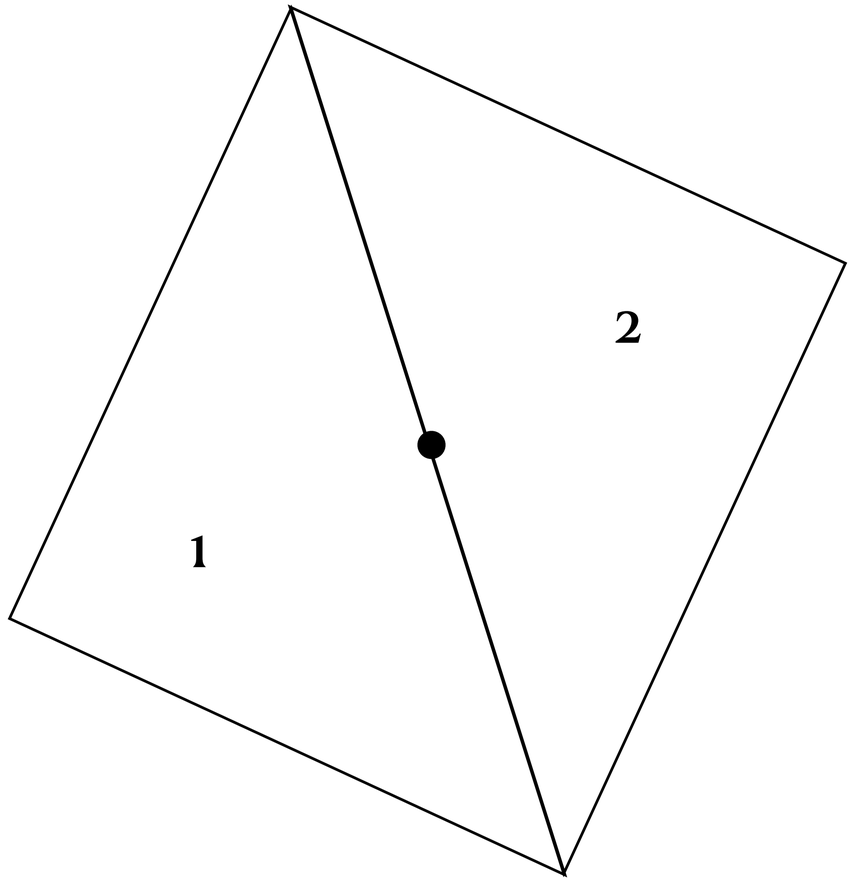
Sample the coverage function at the pixel center

Triangle covers sample

Triangle does not cover sample

Edge cases

Is this point covered by Triangle 1? Triangle 2? Both?

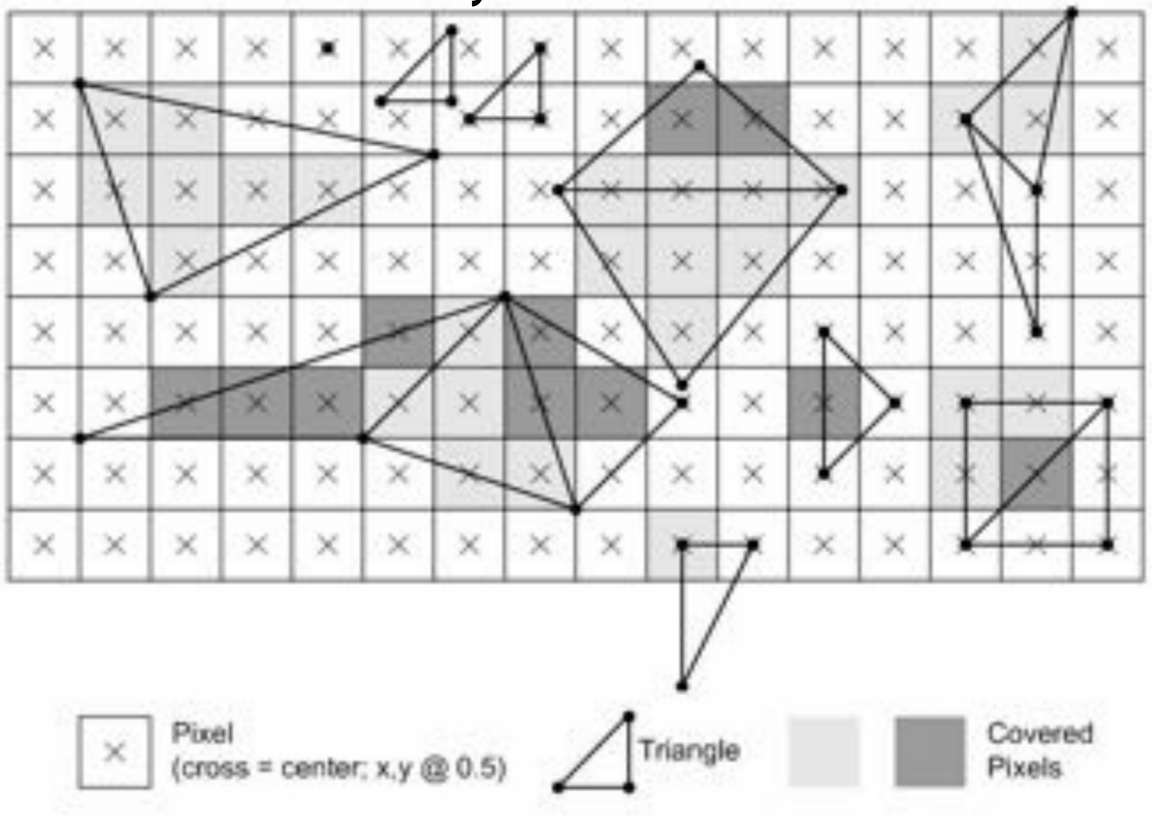


Key is to define a consistent convention

Breaking ties

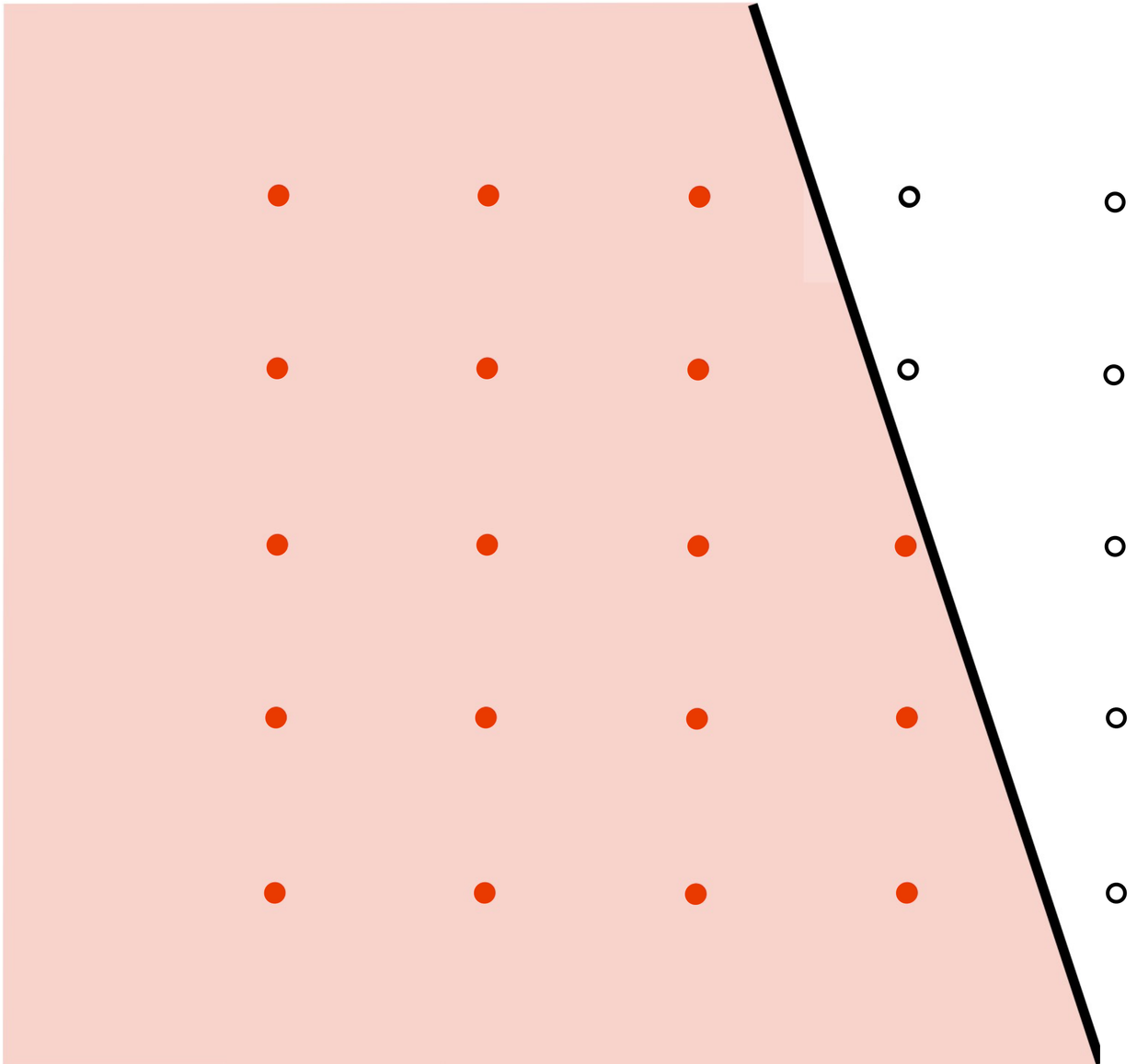
How it's defined in OpenGL

- If we sample a point on an edge we take the triangle for which the edge is a "top edge" or a "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



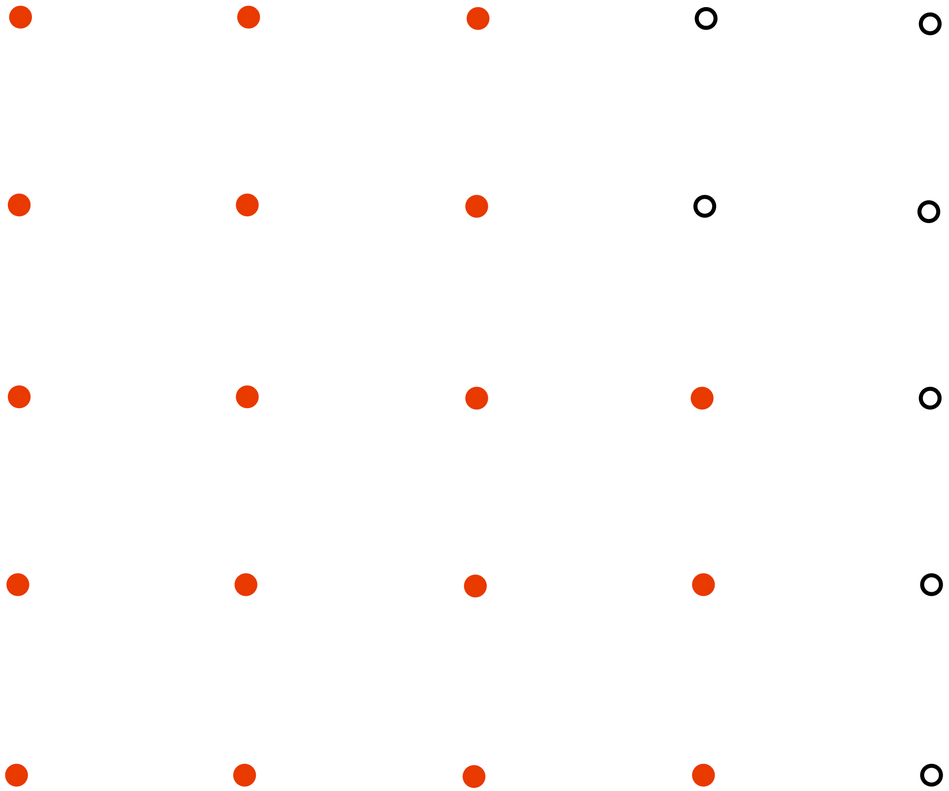
Direct3D Programming Guide, Microsoft

Results of sampling triangle function



Sampled a signal, now want to display it on a screen

What we actually have



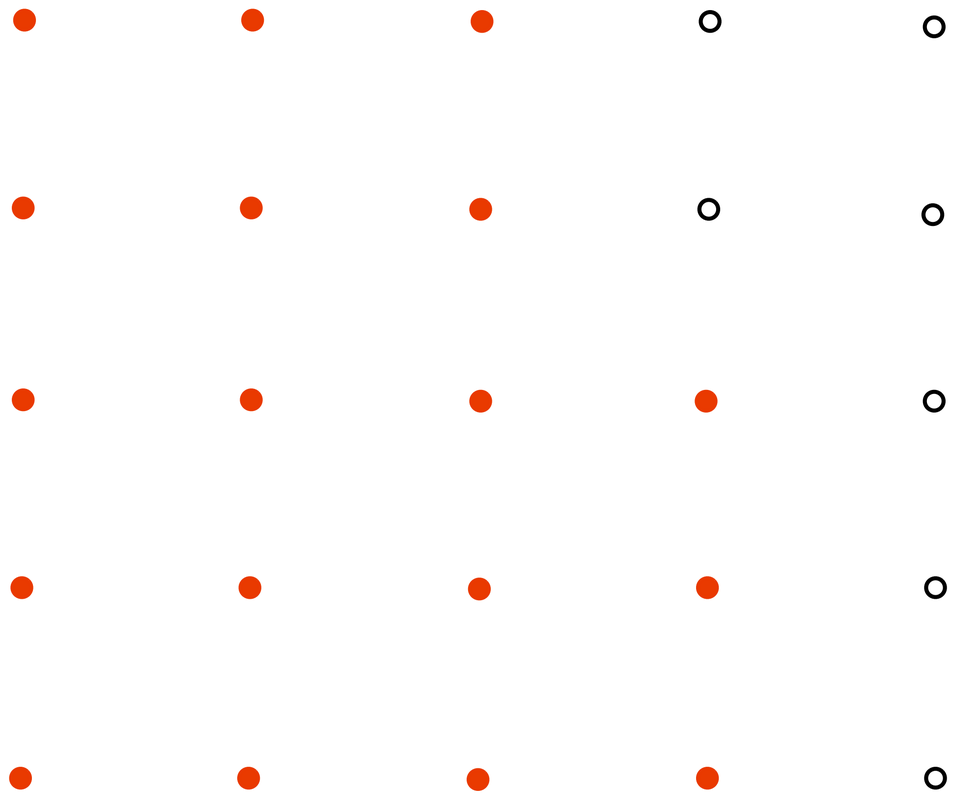
Display is “reconstructing” the image via pixels on a screen

- Each image sample sent to the display is converted into a little square of light with the appropriate color

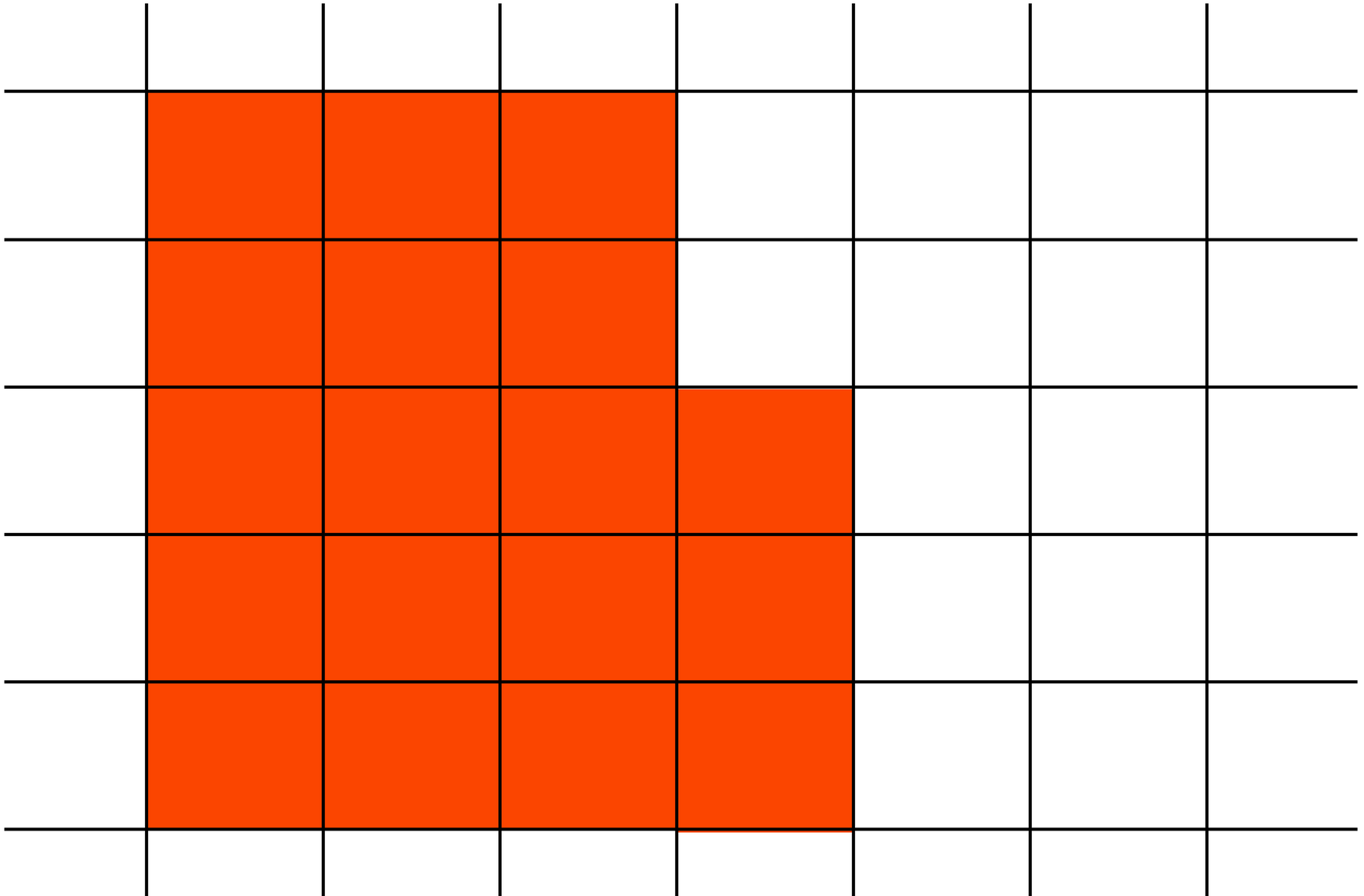
  Pixel = picture element

Not exactly a solid color,
but good enough.

Sending to a display this



We see something like this on the screen



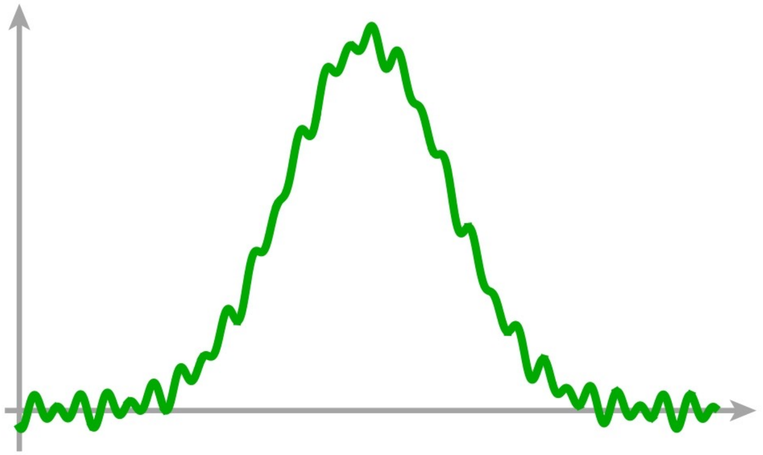
… But the real coverage signal looks like this

Aliasing

When reconstructed/sampled signal presents a false sense of what the actual signal looks like

Sampling and Reconstruction

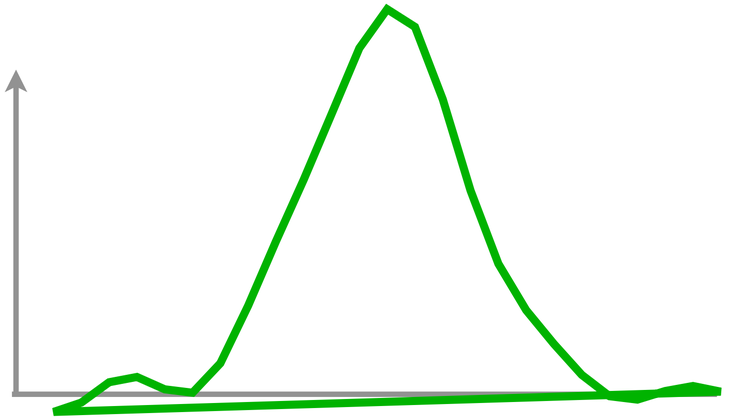
Continuous Signal
(Original)



- 0.0457447
- 0.0209434
0.0328632
0.0468068
0.0141212
0.00506562
0.0829571
0.235369
0.405682
0.569204
0.742511
0.916946
1.02
0.973226
0.781528
0.539974
0.3464
0.223501
0.134011
0.0228676
0.00789505

Discrete
information

Continuous Signal
(Approximation)



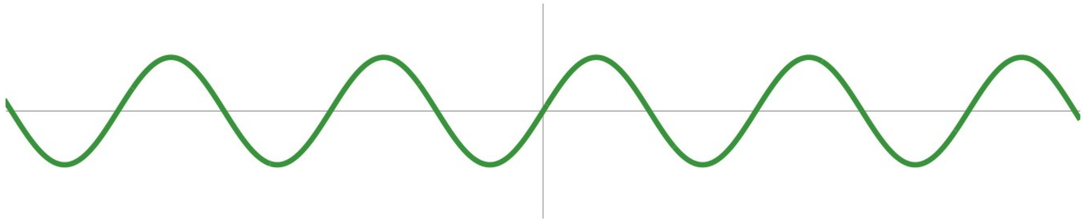
Goal is to reconstruct the signal as accurately as possible

Sample

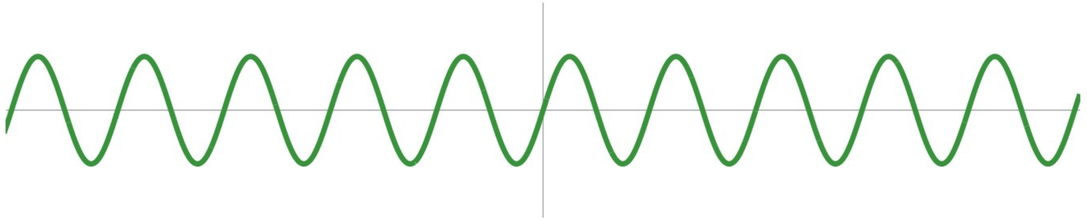
Reconstruct

Expressing a signal as a superposition of frequencies

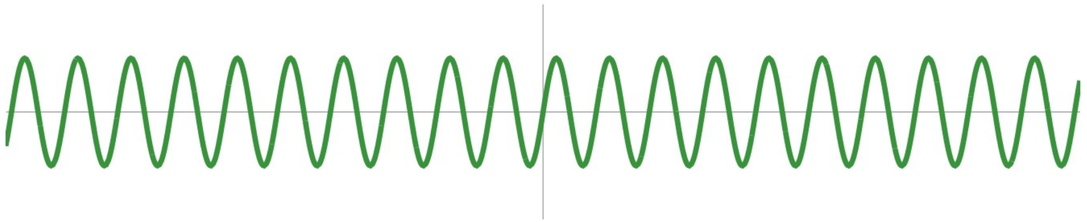
$f_1(x) = \sin(\pi x)$



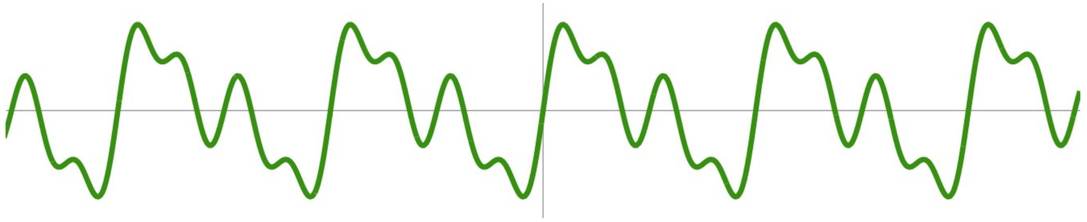
$f_2(x) = \sin(2\pi x)$



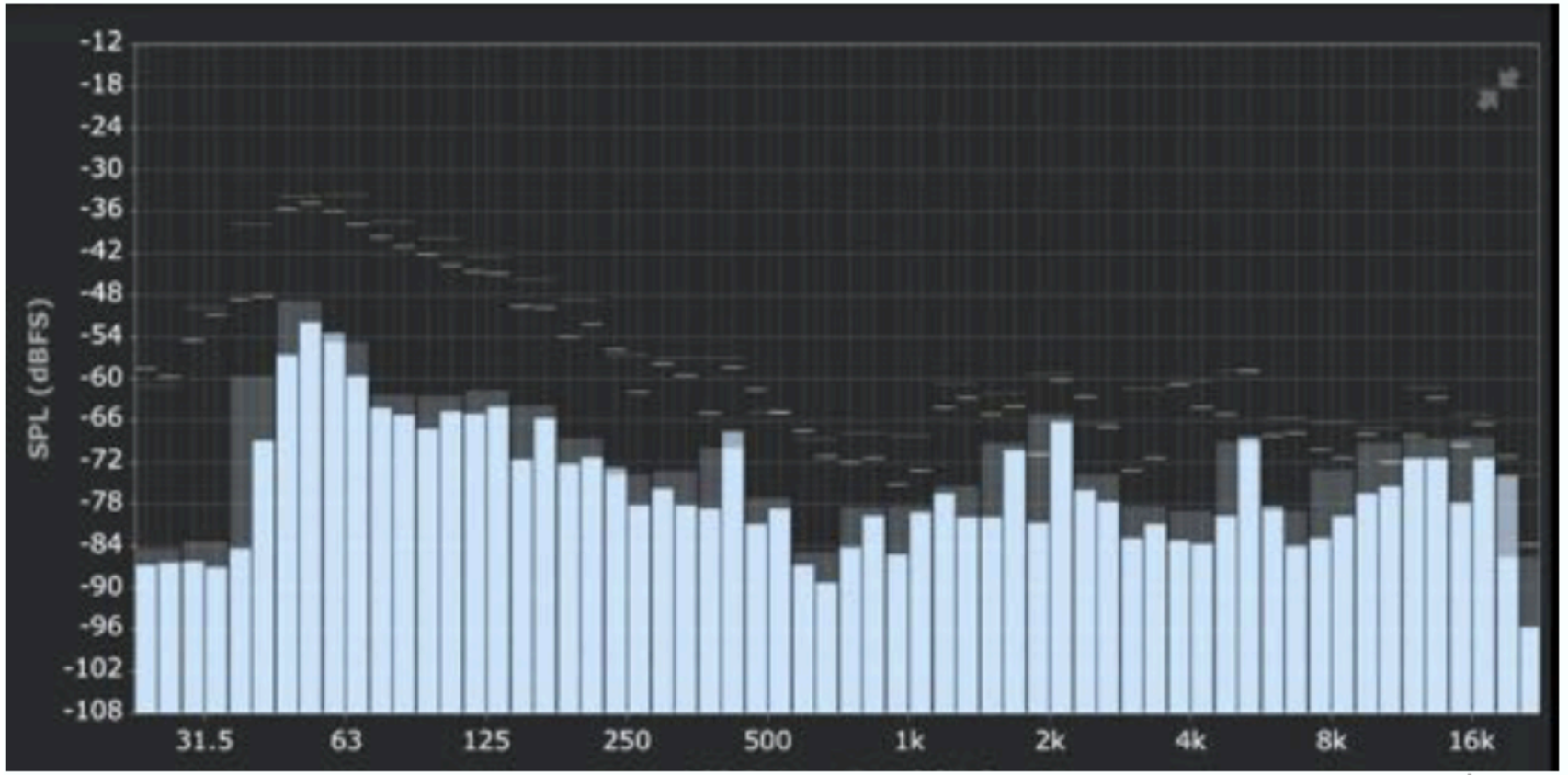
$f_4(x) = \sin(4\pi x)$



$f(x) = f_1(x) + 0.75 \cdot f_2(x) + 0.5 \cdot f_4(x)$



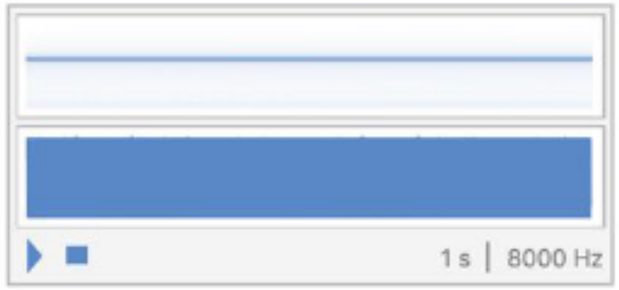
E.g., audio spectrum analyzer shows the amplitude of each frequency



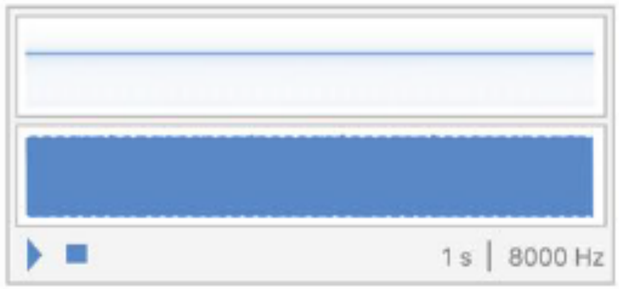
Aliasing in Audio

Get a constant tone by playing a sinusoid of frequency ω

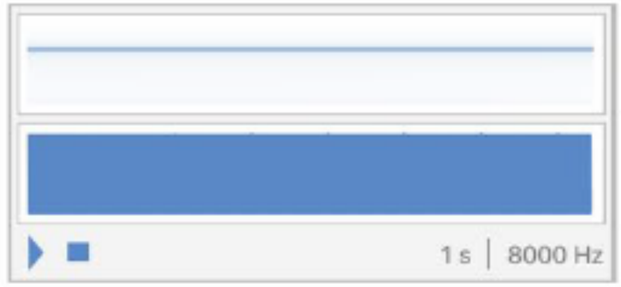
```
Play[Sin[4000 t], {t, 0, 1}]
```




```
Play[Sin[5000 t], {t, 0, 1}]
```

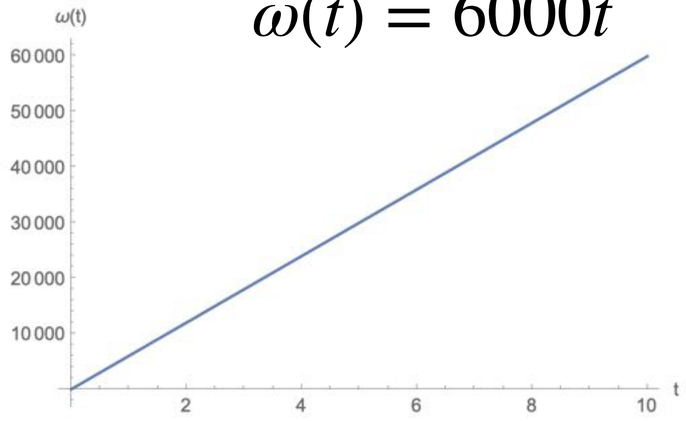



```
Play[Sin[6000 t], {t, 0, 1}]
```

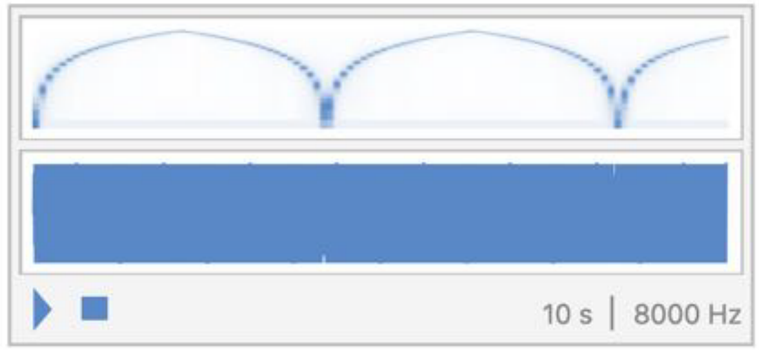


- What happens if we increase ω over time?

$$\omega(t) = 6000t$$

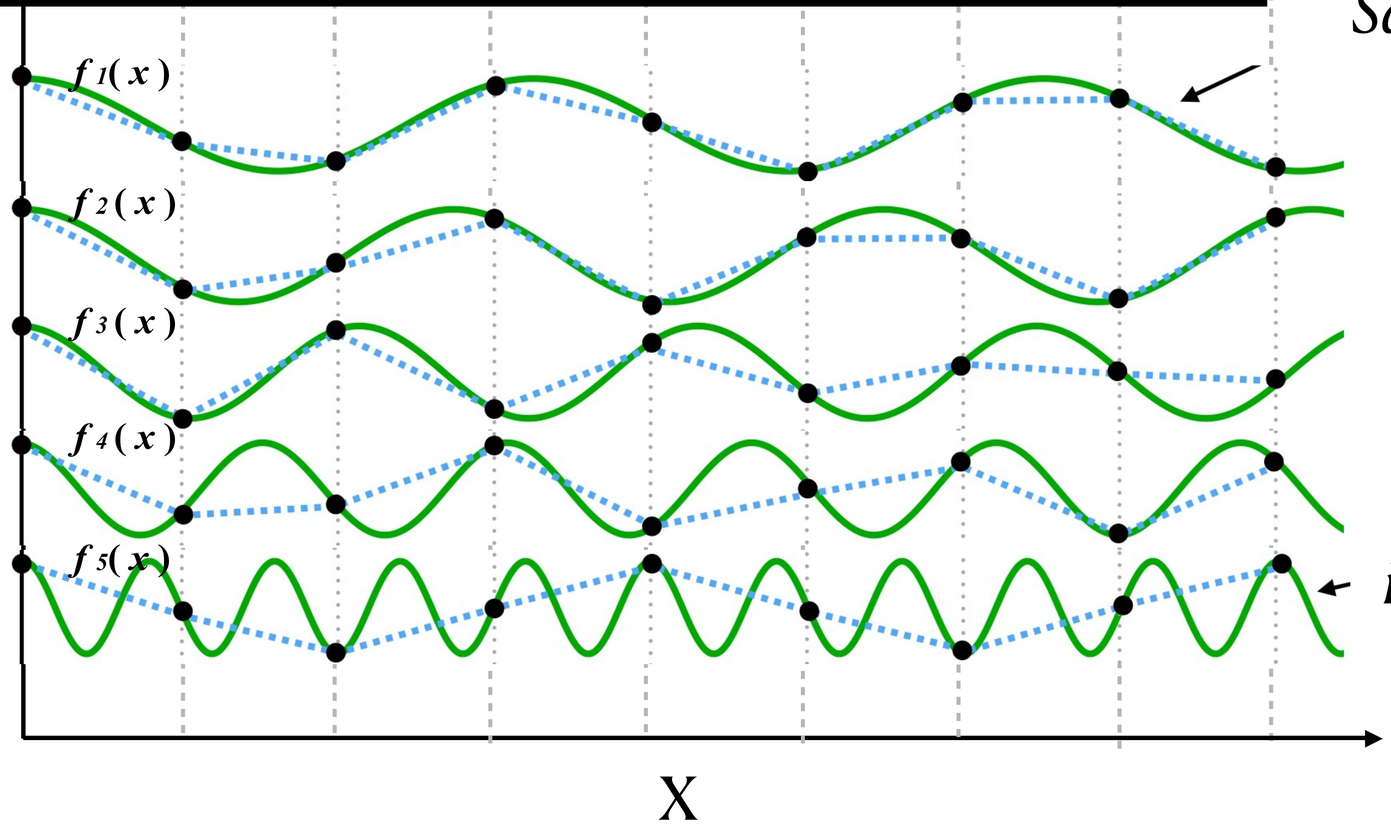



```
Play[Sin[ω t], {t, 0, 10}]
```



Undersampling high-frequency signals results in *aliasing*

"Aliasing": high frequencies in the original signal masquerade as low frequencies after reconstruction (due to undersampling)



Low frequency signal
Sampled sufficiently for reconstruction

High frequency signal
insufficiently sampled for reconstruction

Images in the spectral domain

‘*Frequencies*’ present in the image

Image (spatial domain)

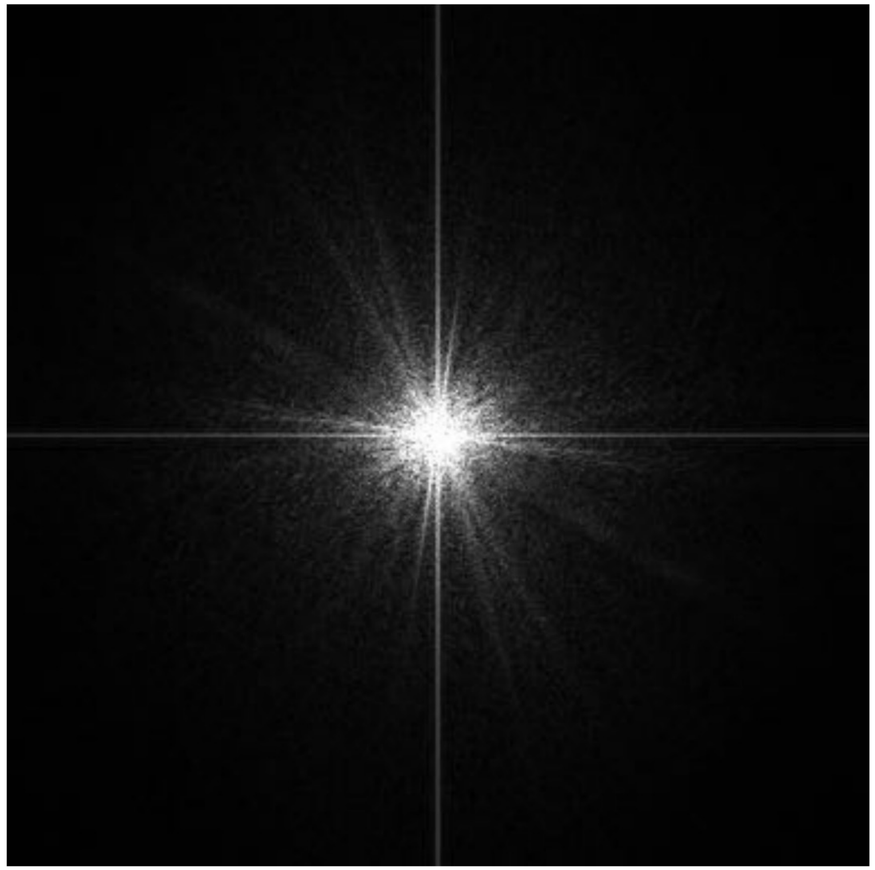
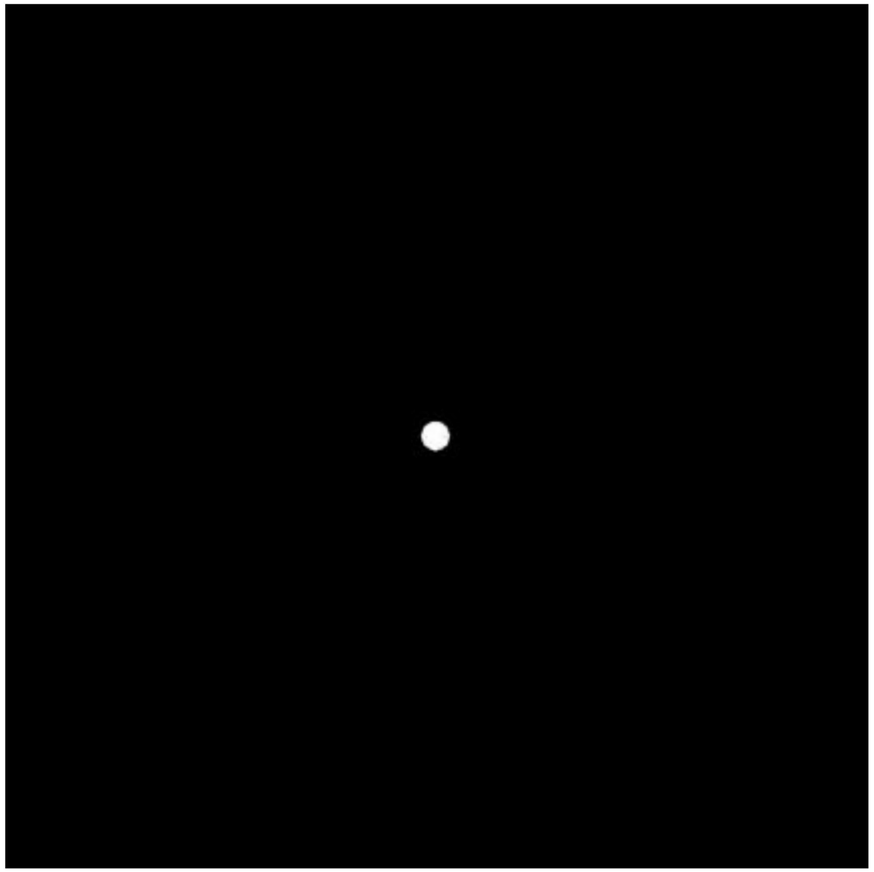


Image (spectral domain)

Low frequencies (smooth regions)

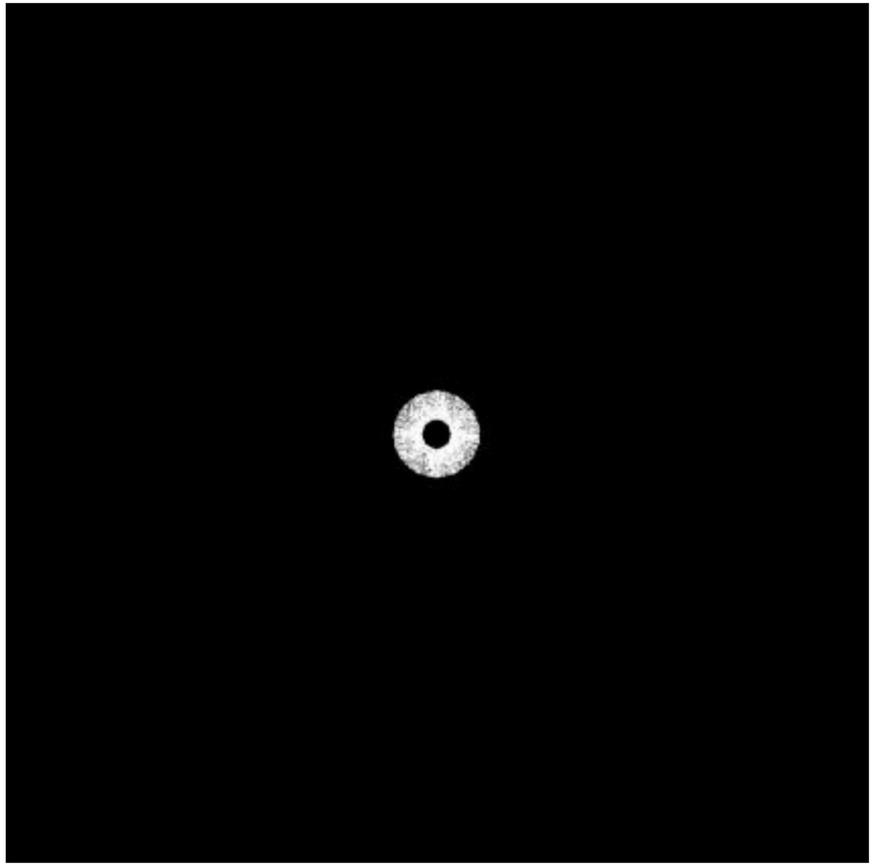
Spatial Domain Result



Spectral domain after lowpass filtering

Mid-range frequencies

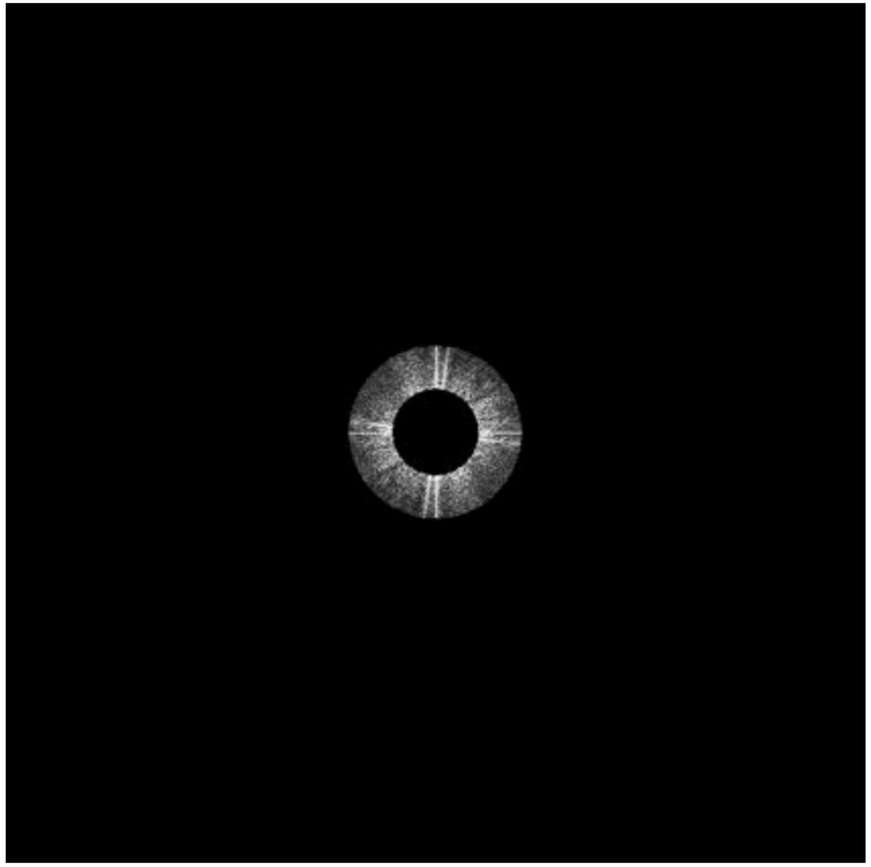
Spatial Domain Result



Spectral domain after bandpass filtering

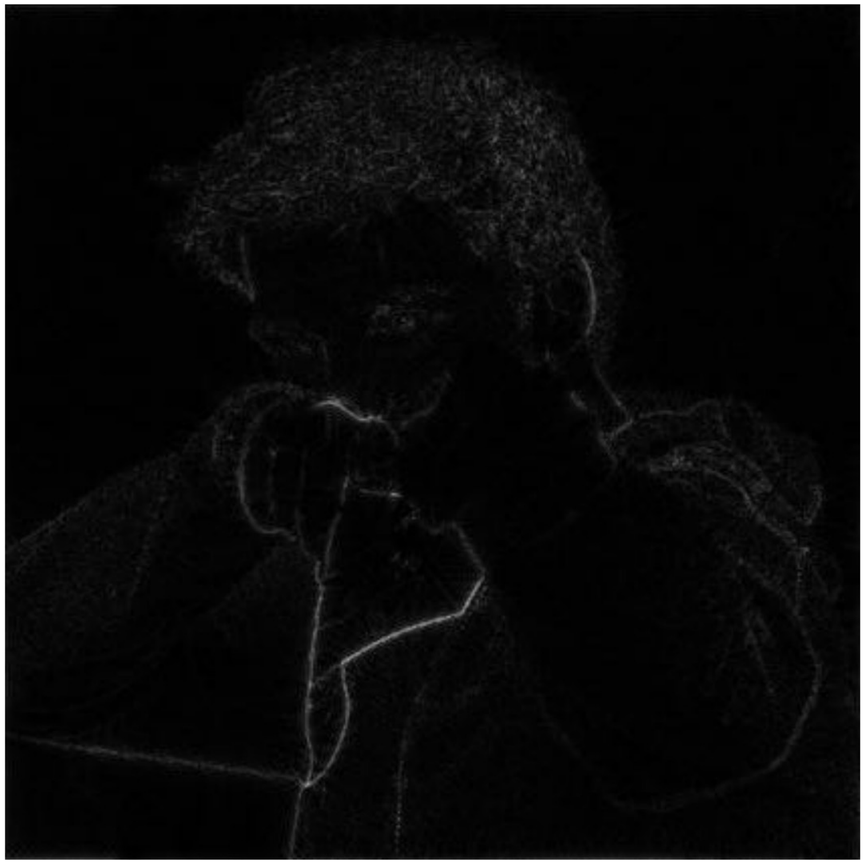
Mid-range frequencies

Spatial Domain Result

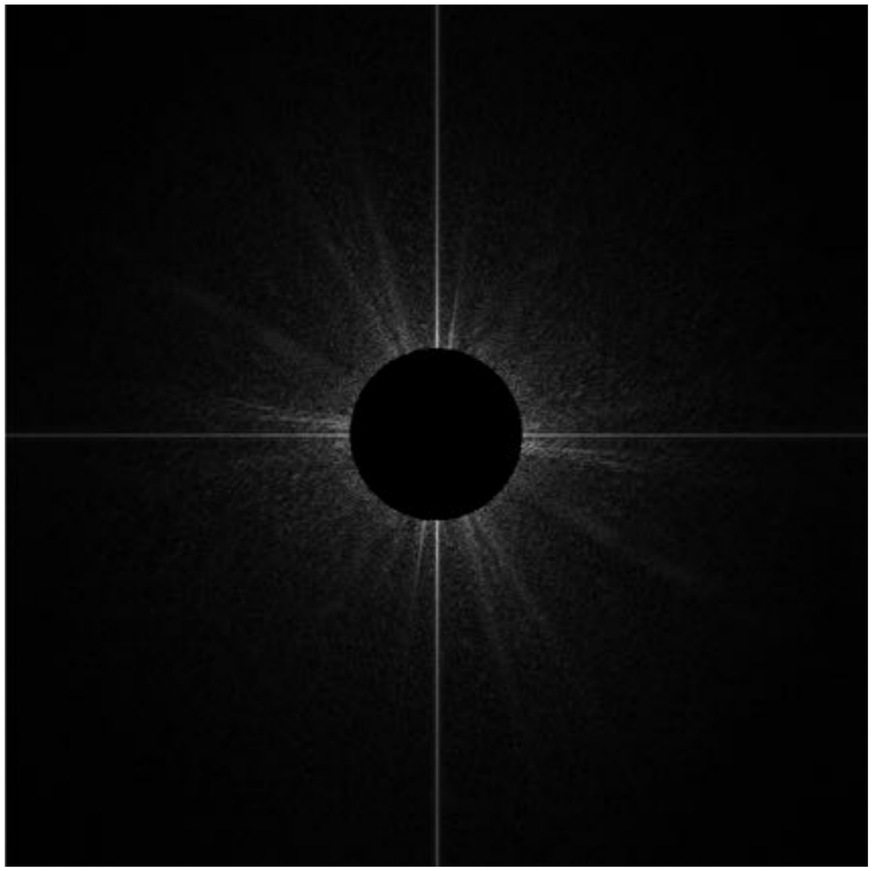


Spectral domain after bandpass filtering

High frequencies (edges)

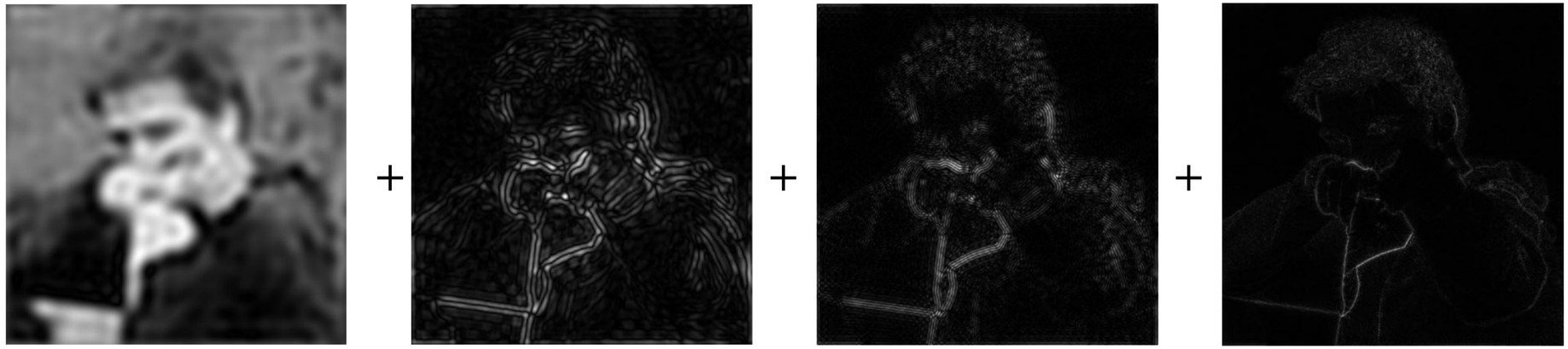


Spatial Domain Result

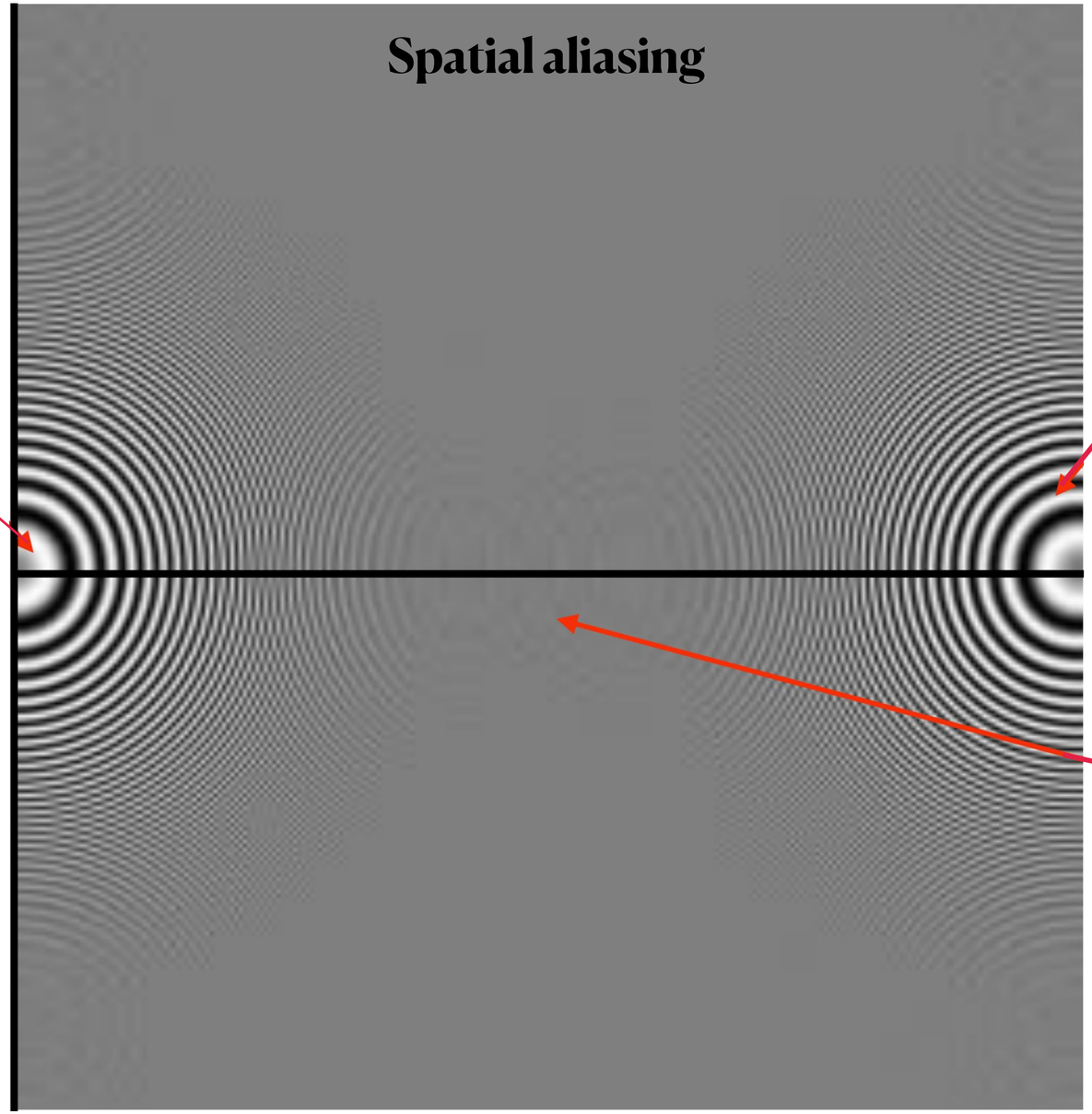


High pass filtering the frequencies

Decomposing images into a sum of frequencies



Create super high-frequency function: $\sin(x^2 + y^2)$

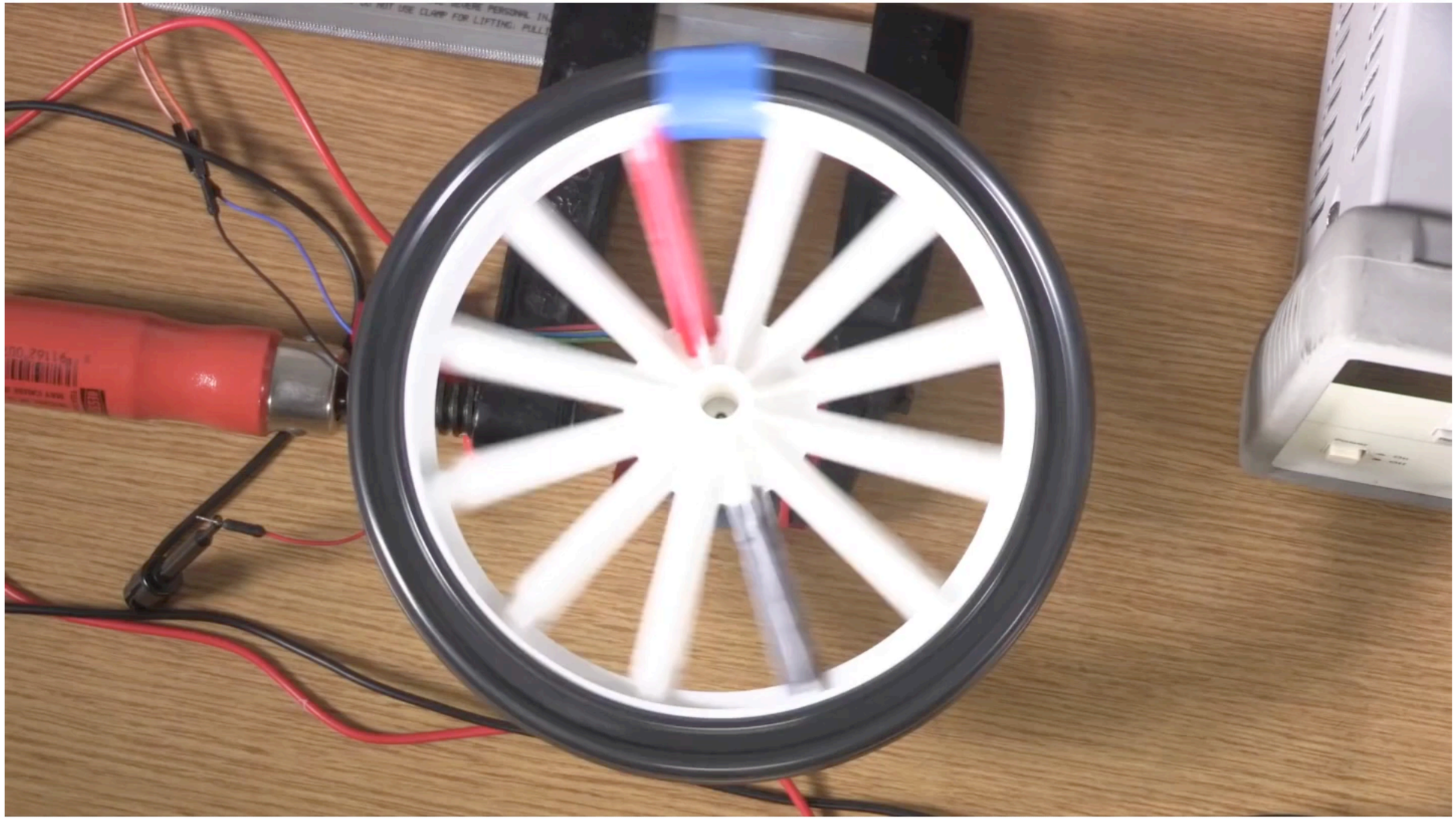


Actual signal (low frequency oscillation)

Under-sampling high frequency oscillation *appears* low frequency even though it is **not**

Limit of what can be represented given pixel grid size

Temporal Aliasing



Nyquist-Shannon theorem

- In the case of a band-limited signal with no frequencies above some threshold ω_{SIGNAL}
  - 1D example: low-pass filtered audio signal
  - 2D example: low-pass filtered image signal

$$\omega_{SAMPLE} \geq 2\omega_{SIGNAL}$$

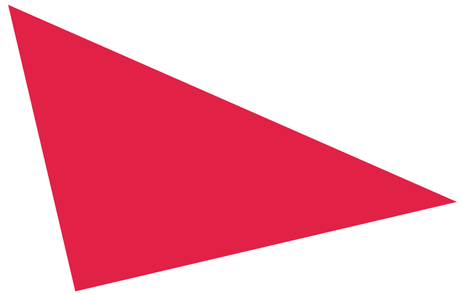
- *If a system uniformly samples an analog signal at a rate that exceeds the signal's highest frequency by at least a factor of two, the original analog signal can be perfectly recovered from the discrete values produced by sampling.*

- *Interestingly, that frequency used for CDs and MP3s (44.1 kHz) is exactly twice the maximum frequency our ears can perceive!*

Challenges of sampling in graphics

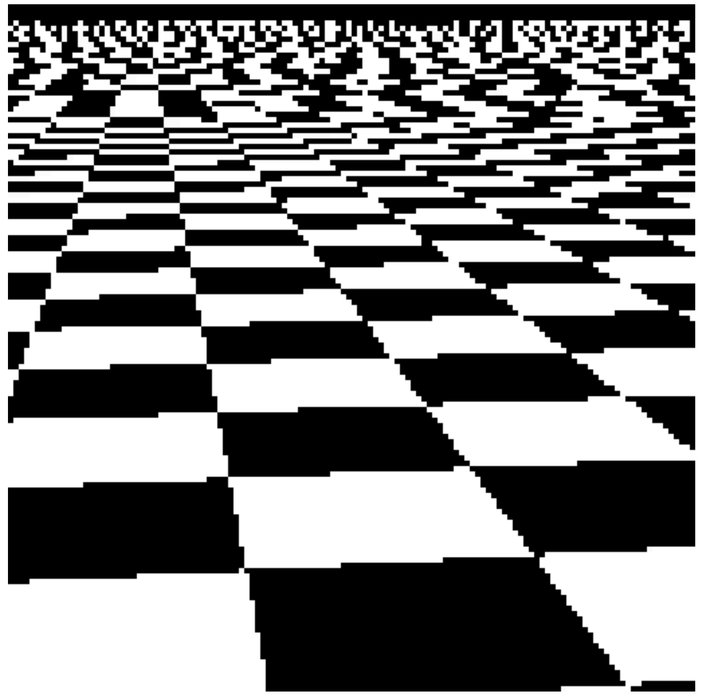
- Signals usually are not band-limited in computer graphics!

Q: why?



Aliasing artifacts in images

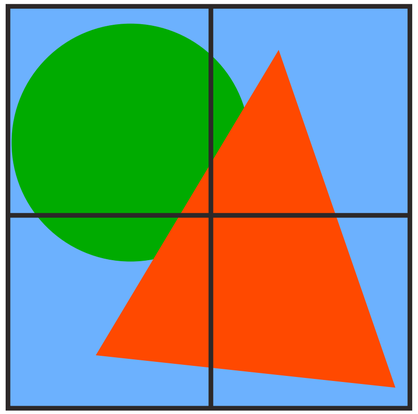
- Imperfect sampling + imperfect reconstruction leads to image artifacts
  - “Jaggies” in a static image
  - “Roping” or “shimmering” of images when animated
  - Moire patterns in high-frequency areas of images



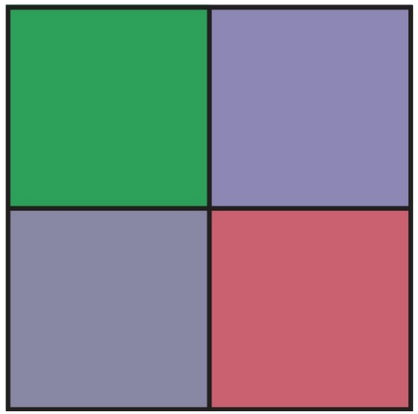
How to reduce aliasing?

In general: blurring/low pass filtering

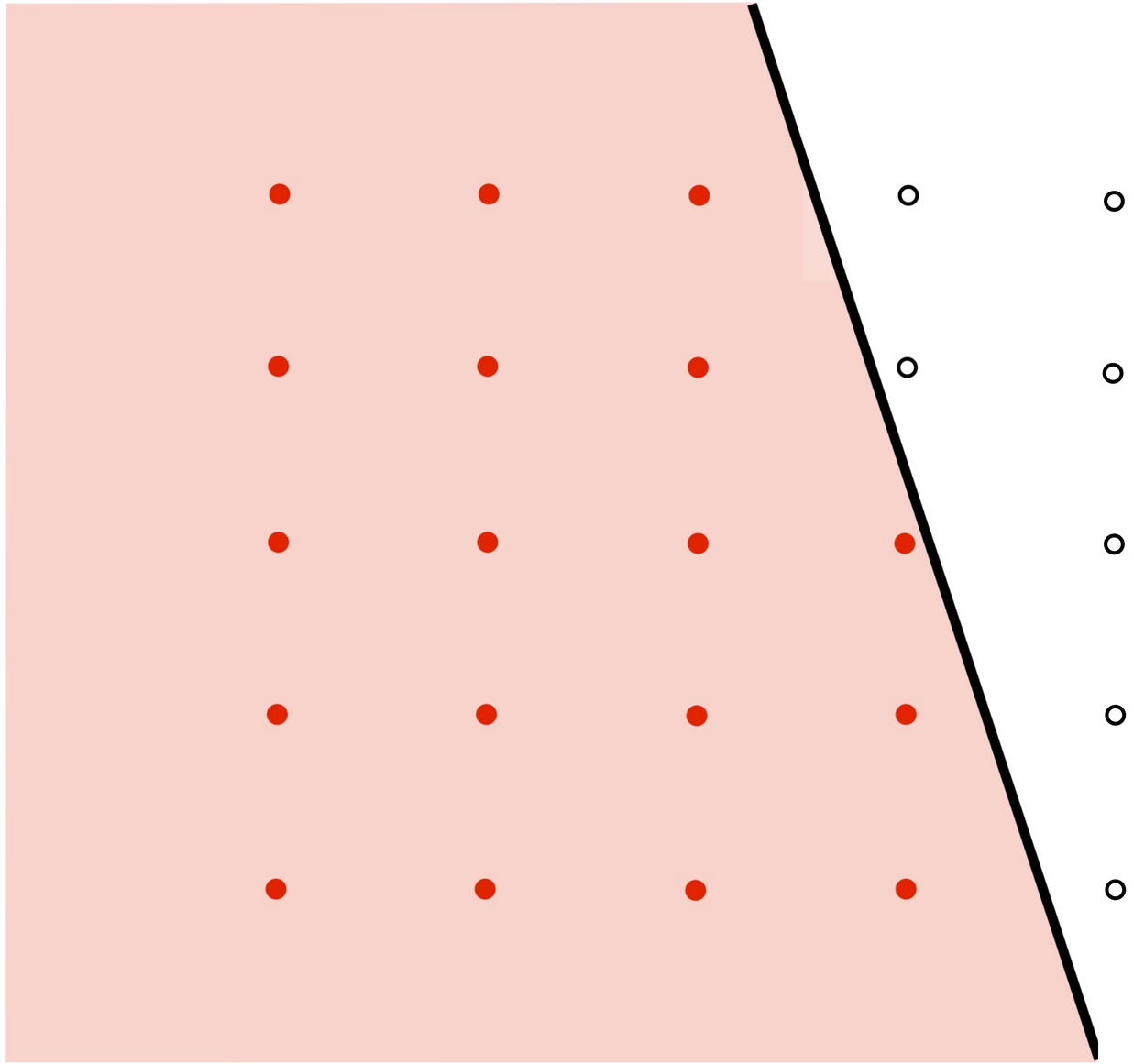
- No matter what, aliasing is a fact of life. Any sampled representation eventually fails to capture frequencies that are too high.
- But we can still do our best to try to match sampling & reconstruction so that the signal we reproduce looks as much as possible like the signal we acquire



- For instance, if we think of a pixel as a “little square” of light, then we want the total light emitted to be the same as the total light in that pixel
  - i.e., we want to integrate the signal over the pixel “box filter”



- Let’s approximately integrate the signal coverage(x,y) by sampling…

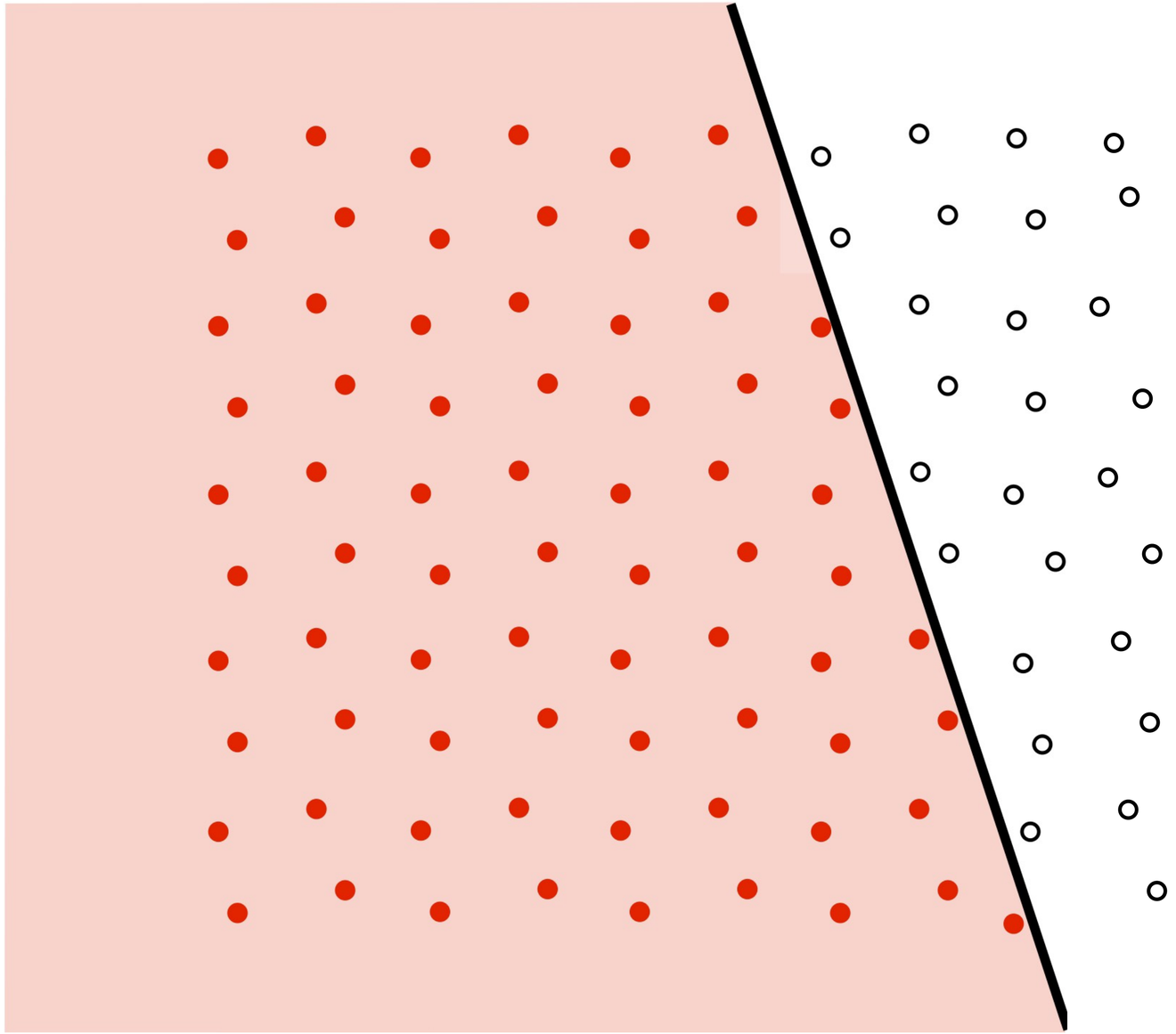


Initial coverage sampling rate

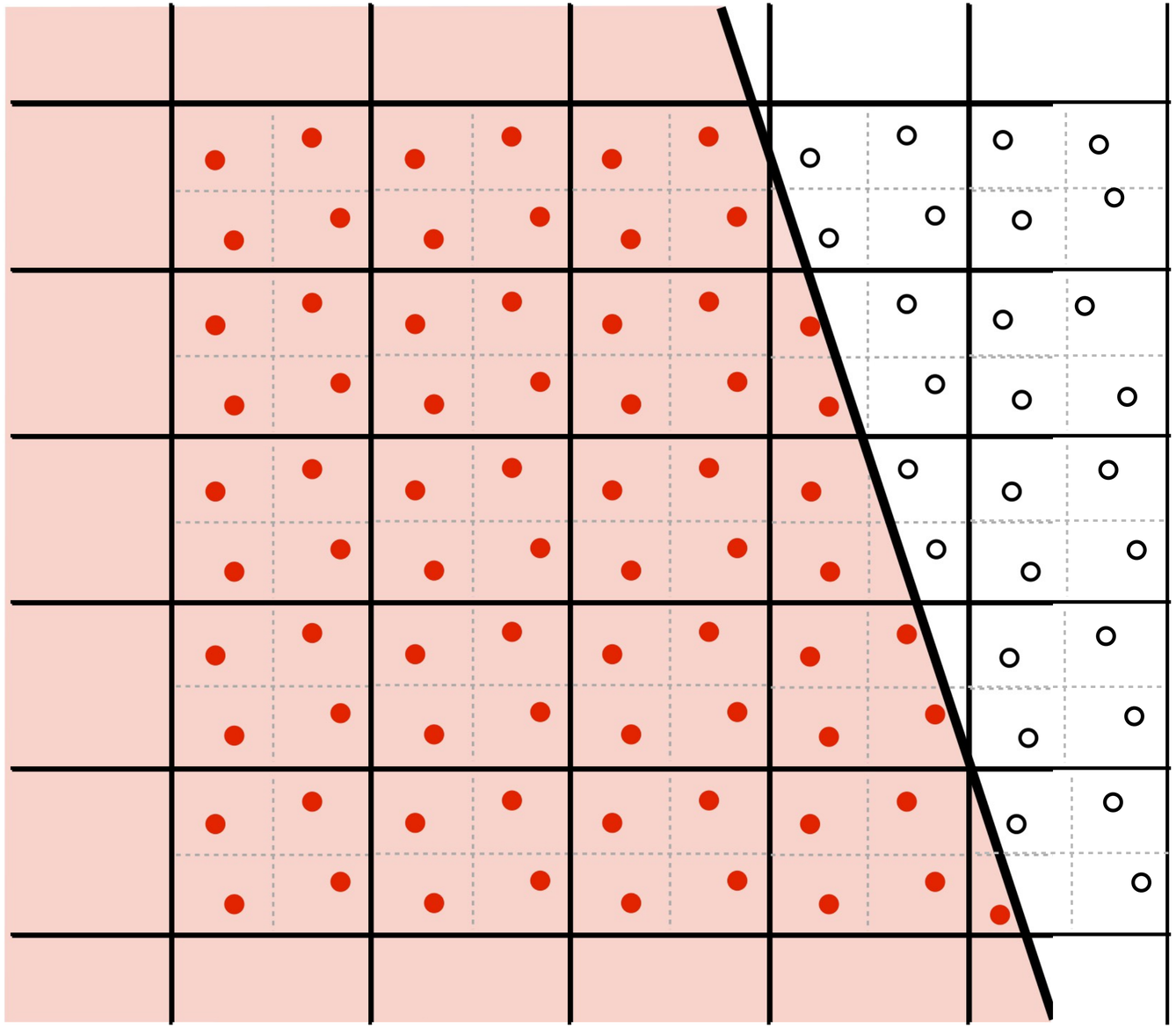
Take 1 sample at the pixel center

How to reduce reconstruction error?

Increase frequency of sampling coverage signal

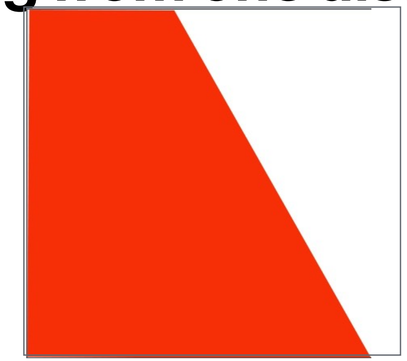


Supersampling

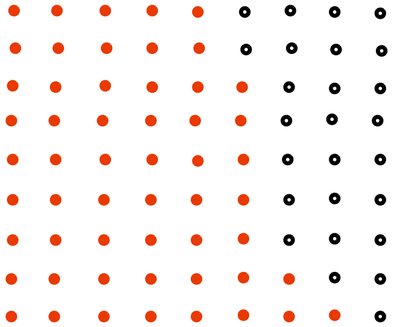


Resampling

Converting from one discrete sampled representation to another

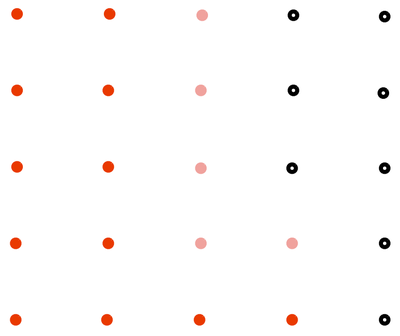


Original signal
(high frequency edge)



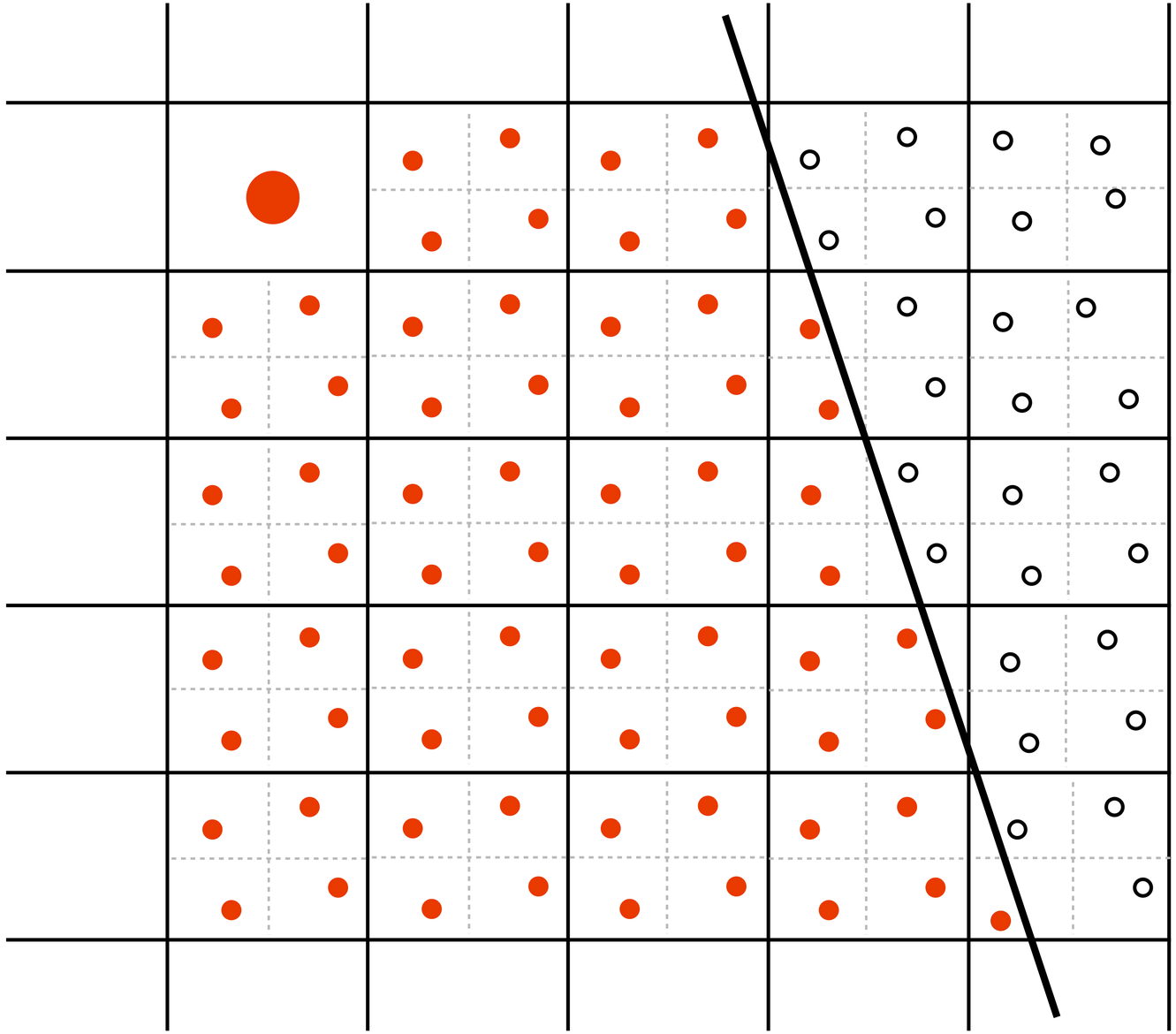
Dense sampling of
reconstructed signal

Reconstructed signal
(lacks high frequencies)



Coarsely sampled signal

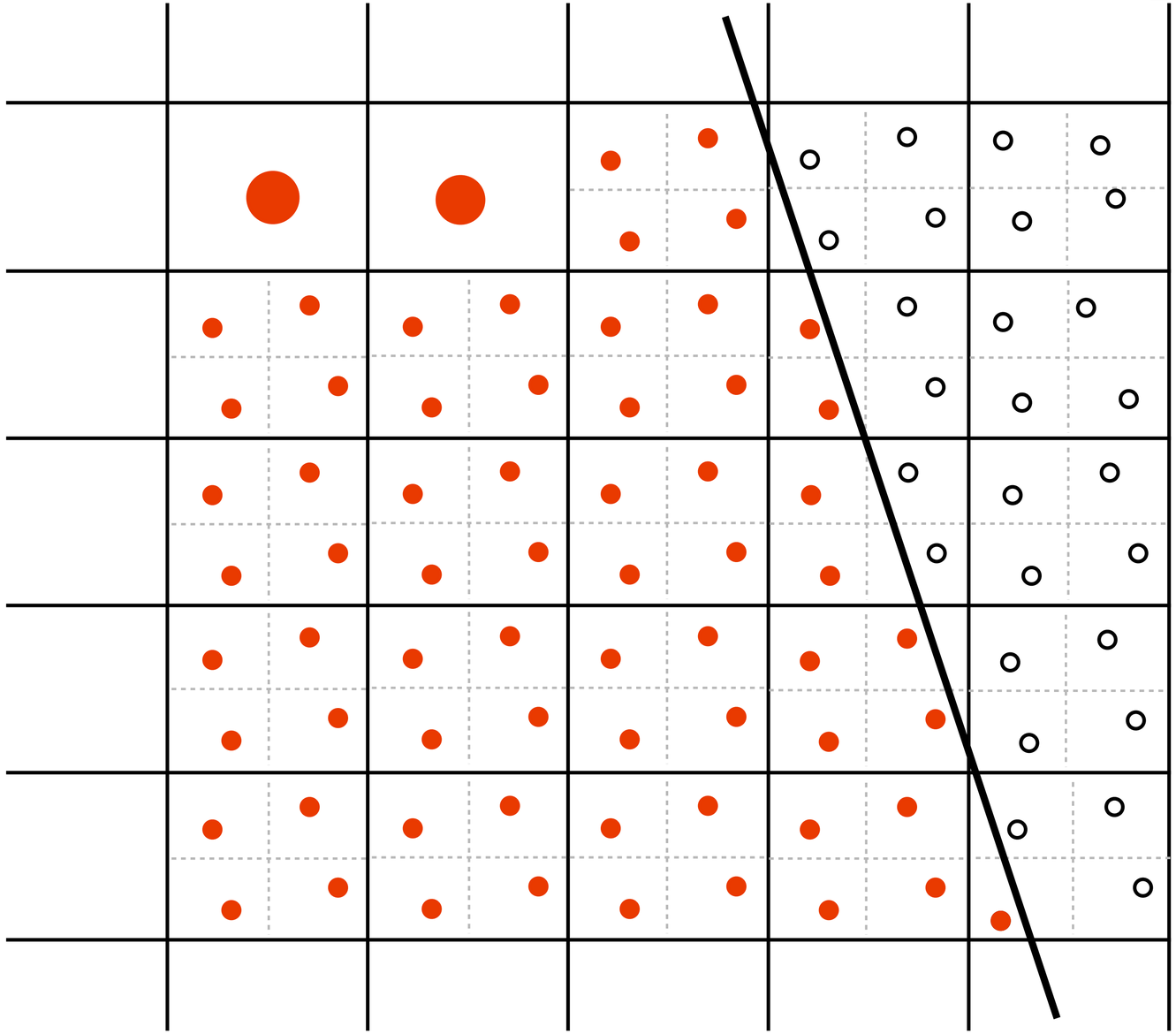
Resample to display's pixel resolution



Because a screen displays one sample value per screen pixel...

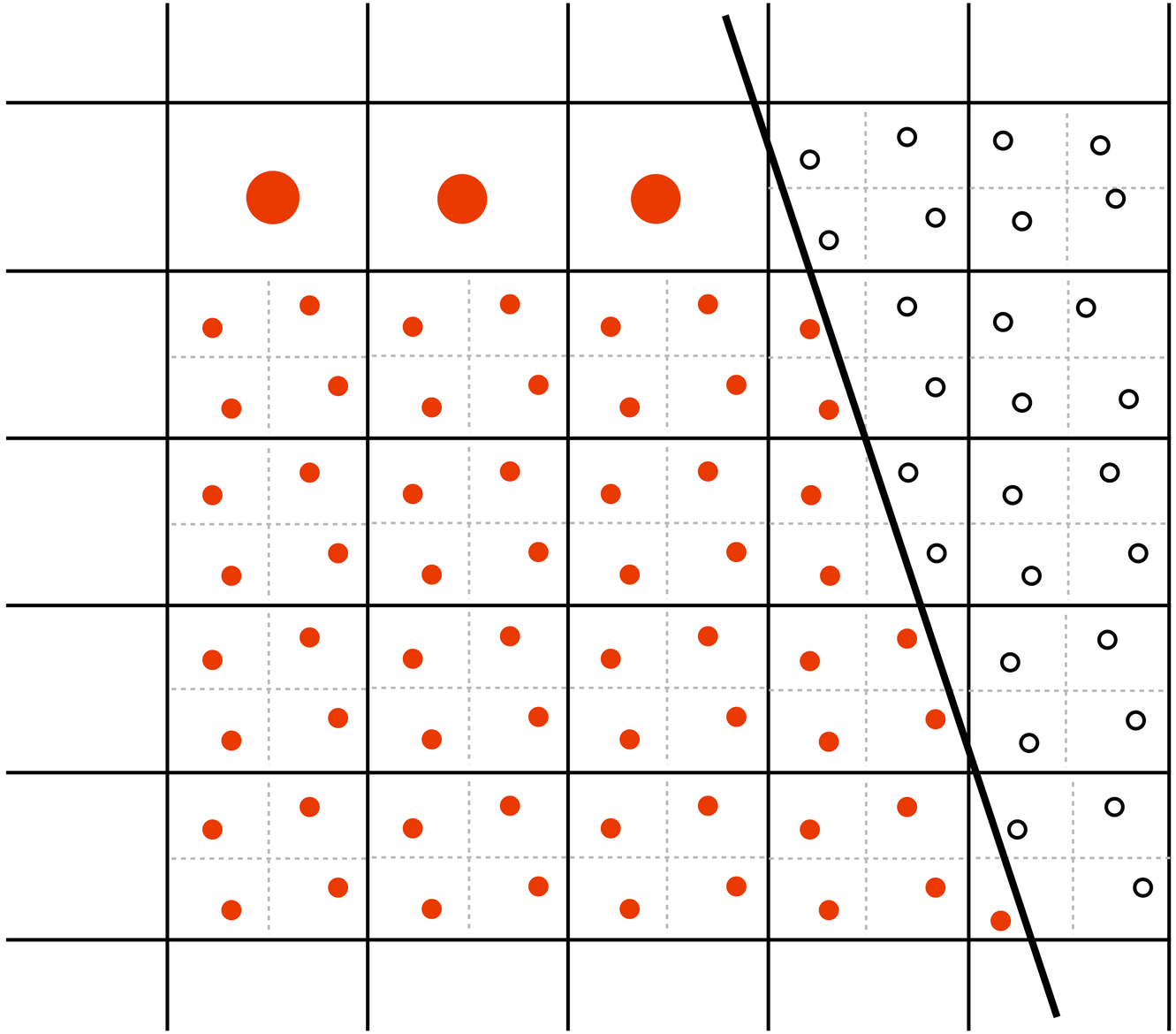
Resample to display's pixel rate

(Box filter)



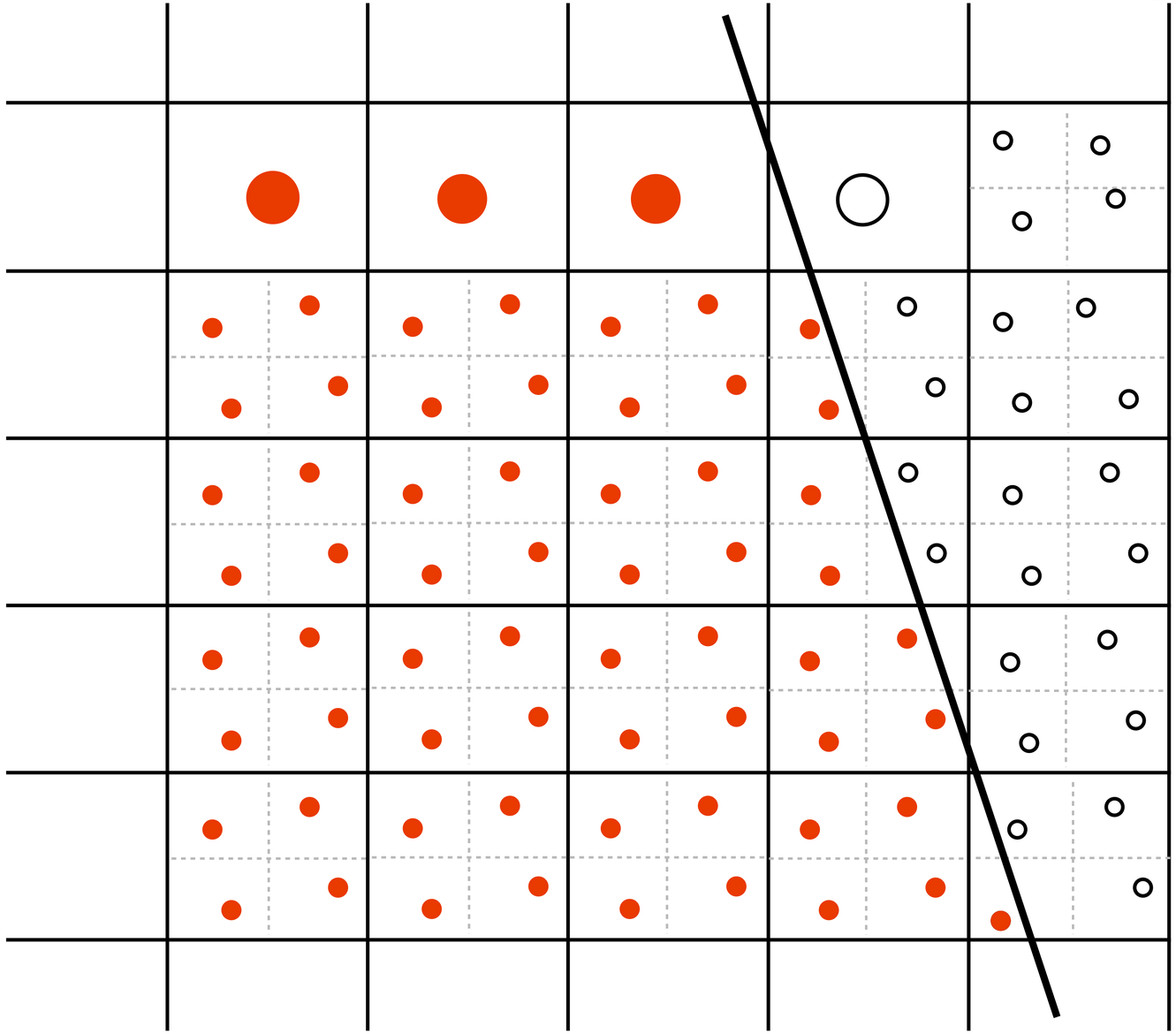
Resample to display's pixel rate

(Box filter)



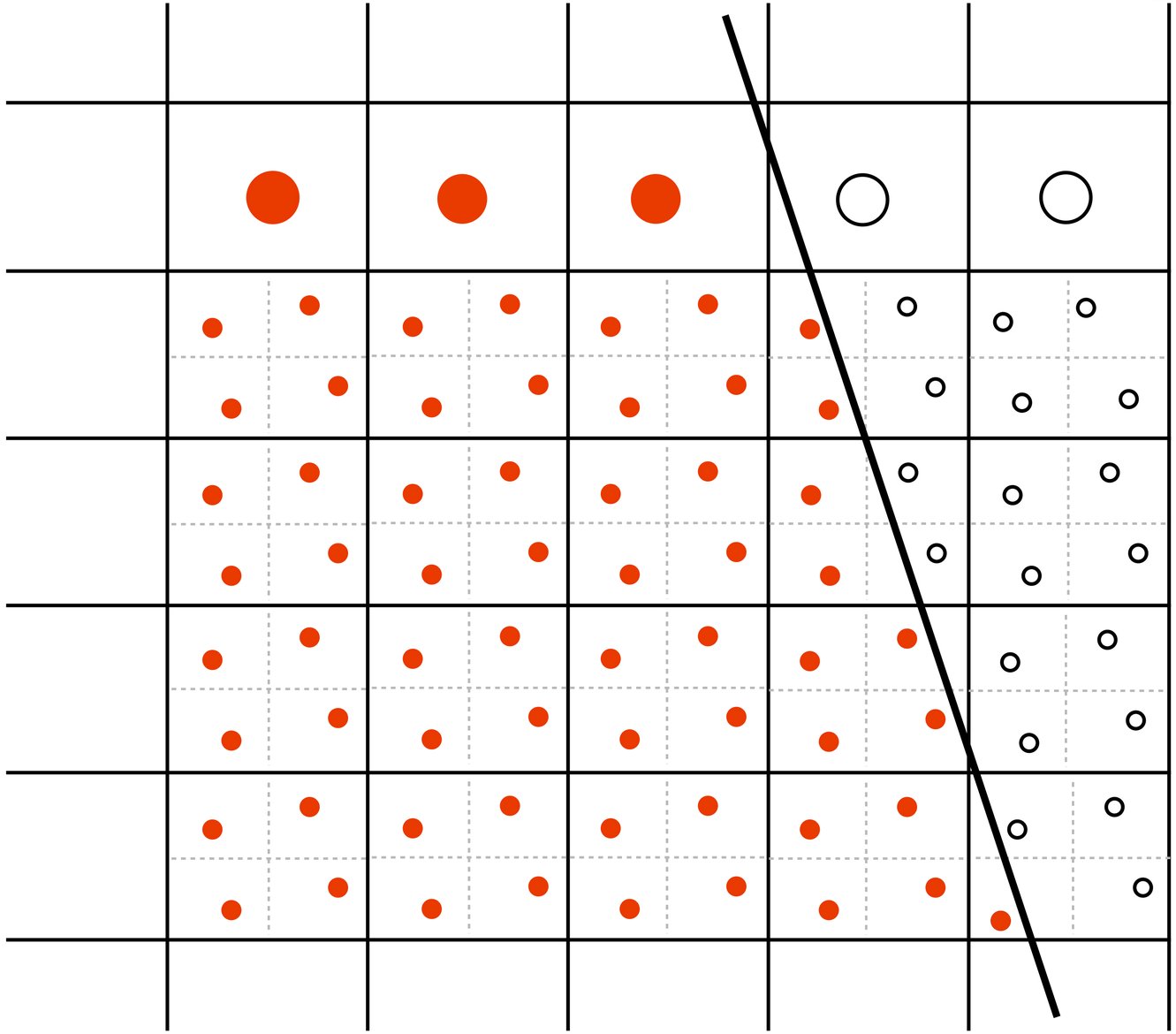
Resample to display's pixel rate

(Box filter)



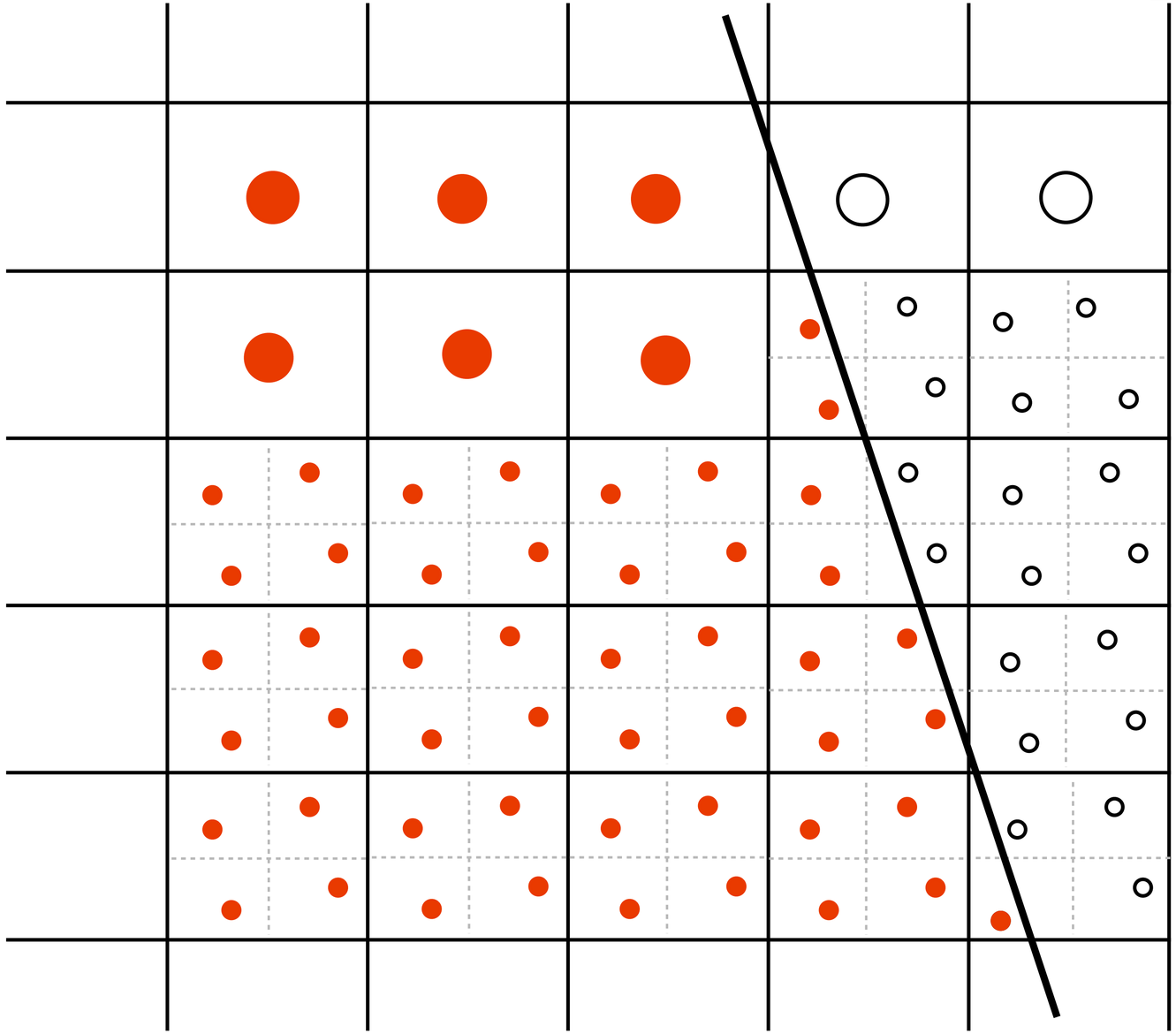
Resample to display's pixel rate

(Box filter)



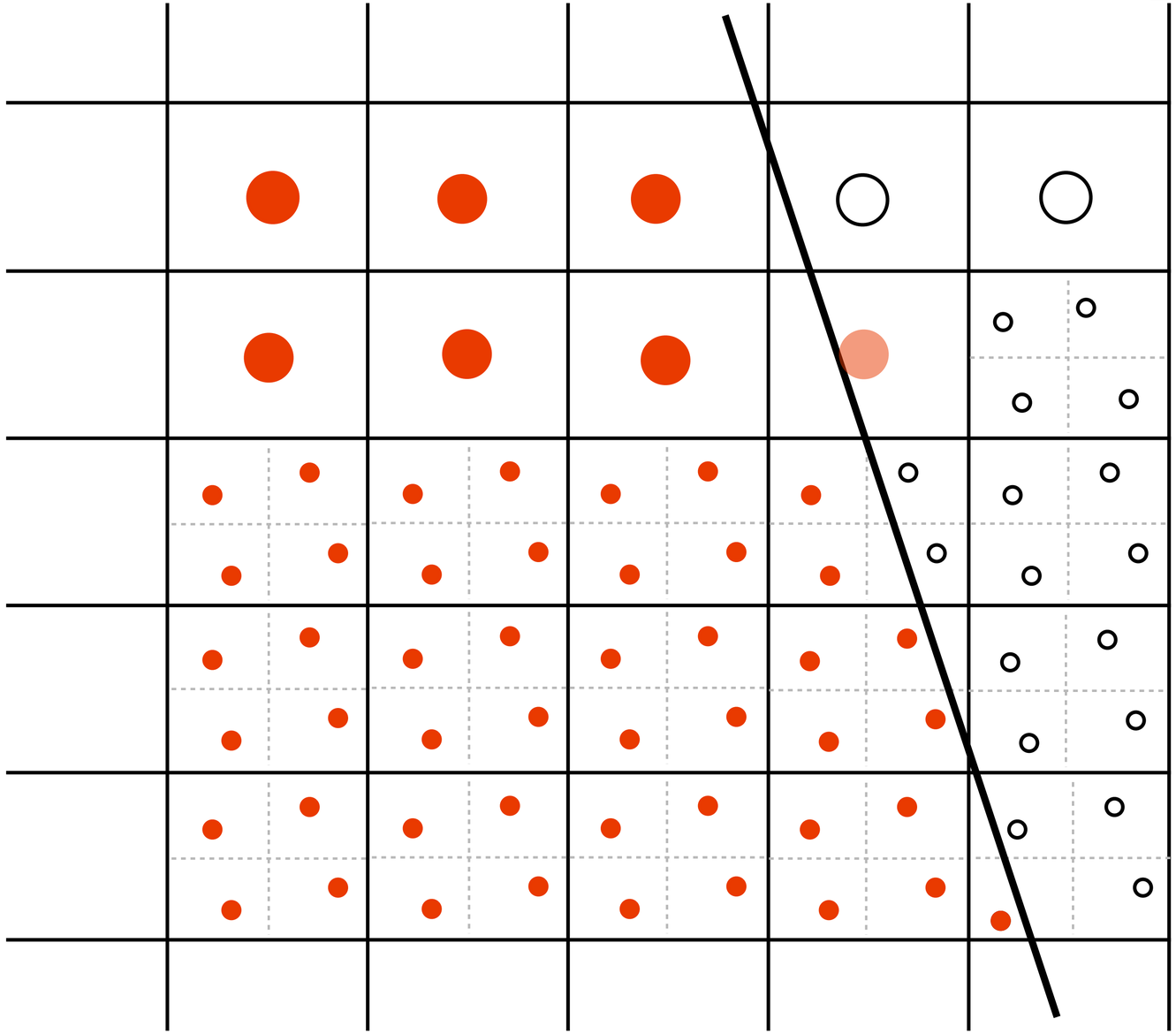
Resample to display's pixel rate

(Box filter)



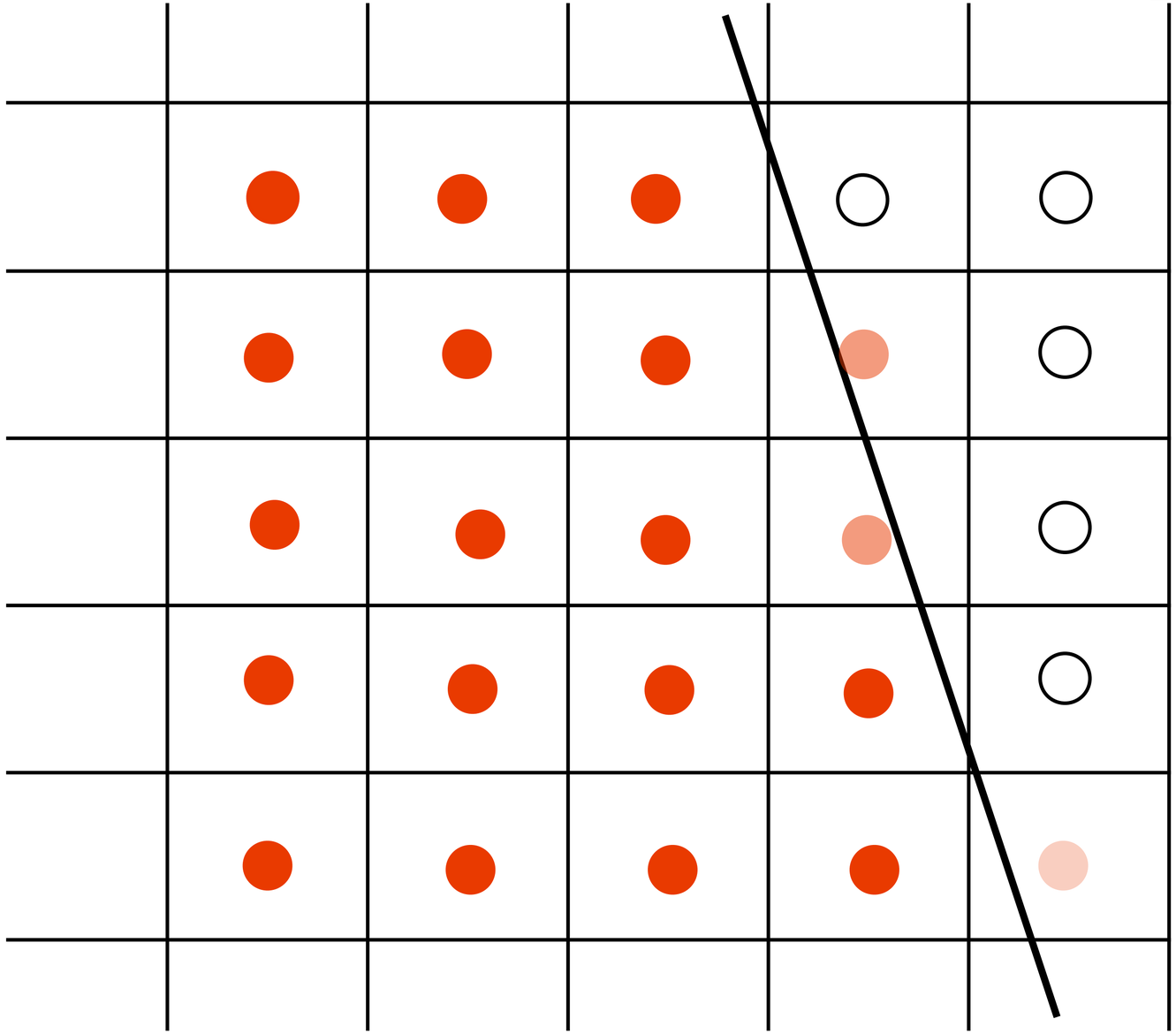
Resample to display's pixel rate

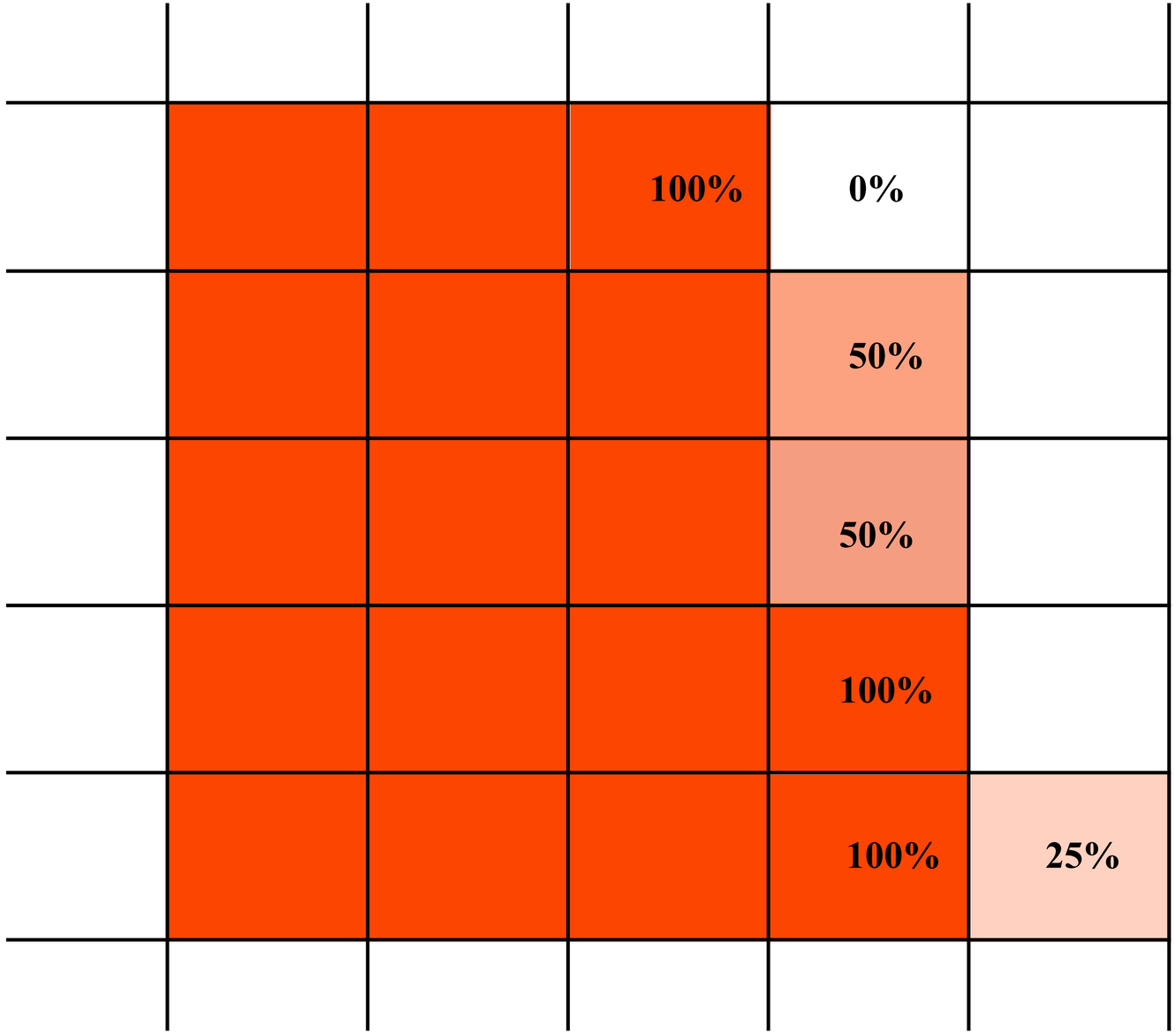
(Box filter)



Resample to display's pixel rate

(Box filter)



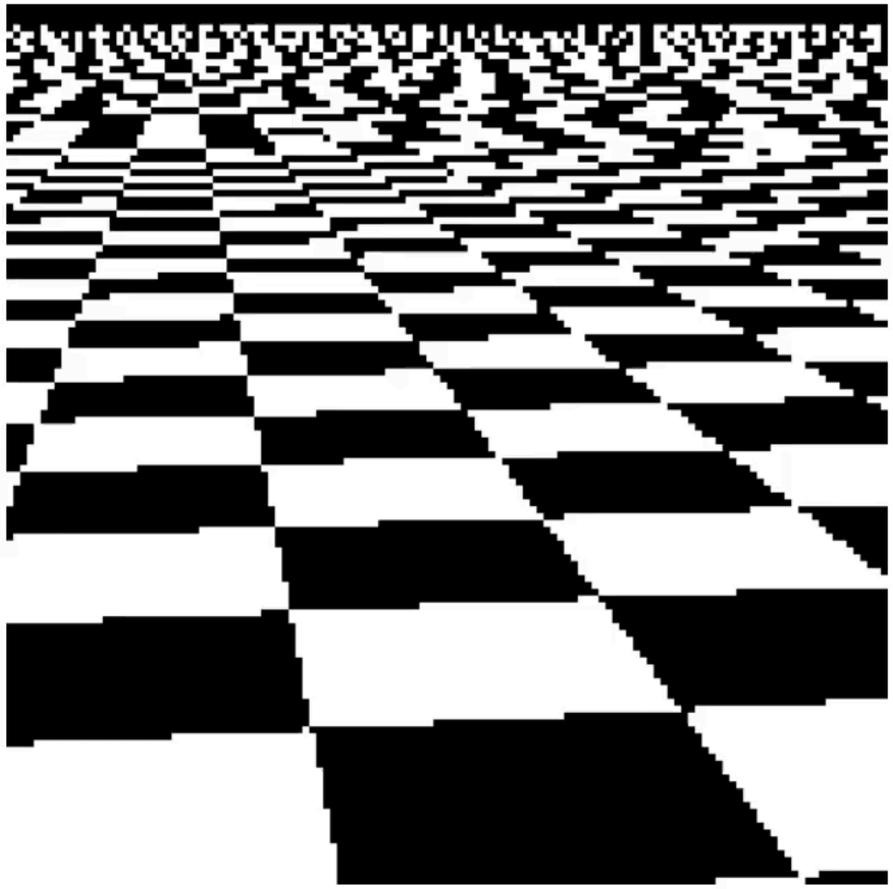


Displayed result

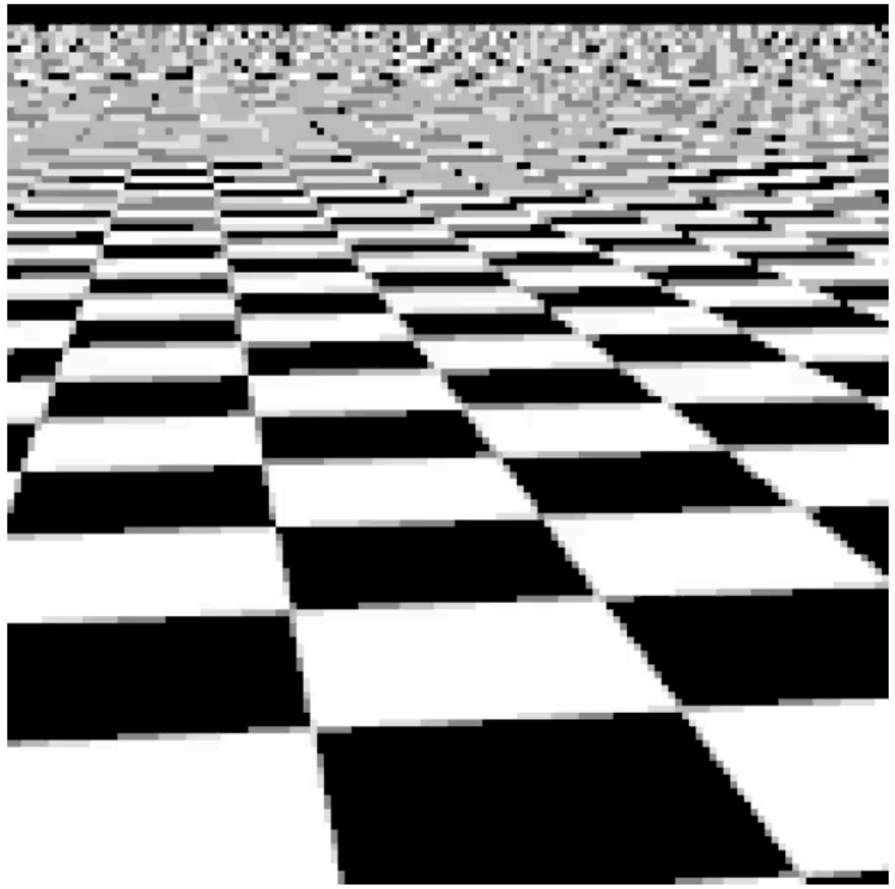
Note anti-aliased edges

Recall: the real coverage

Single sample vs. super sampling

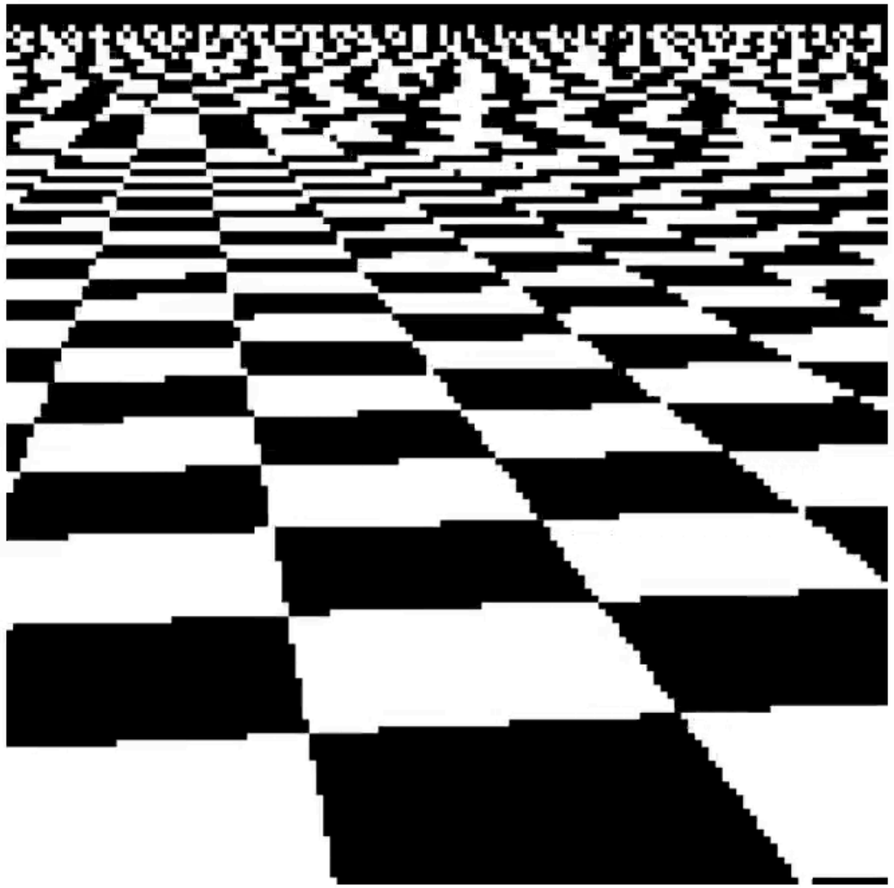


single sampling

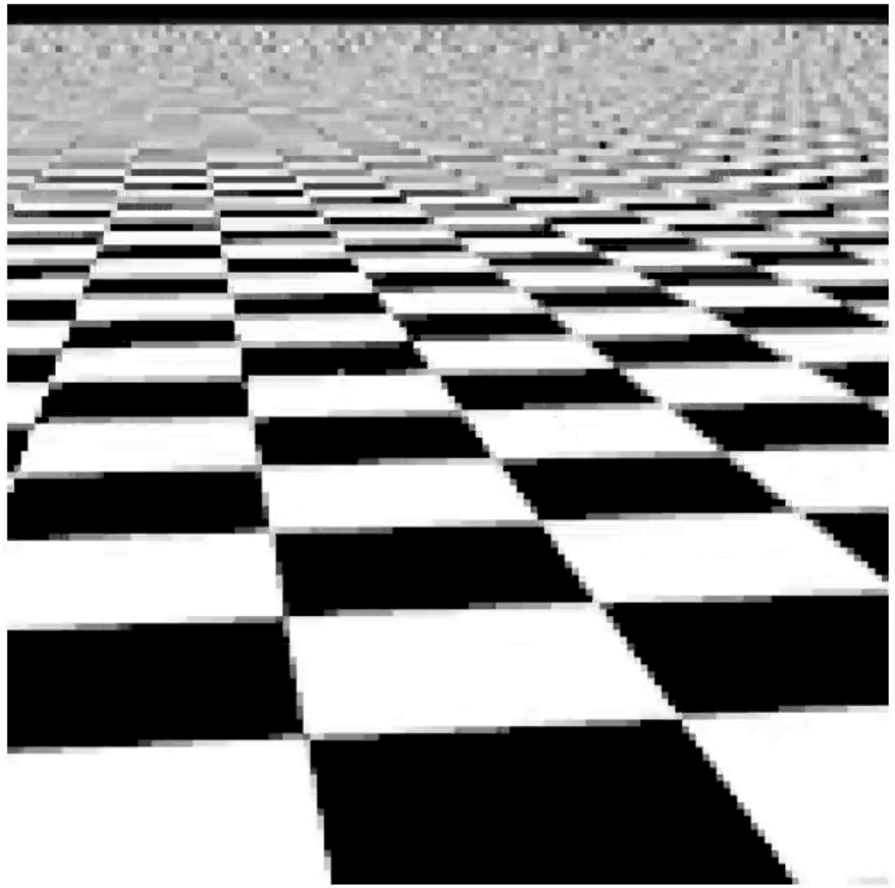


2x2 supersampling

Single sample vs. super sampling

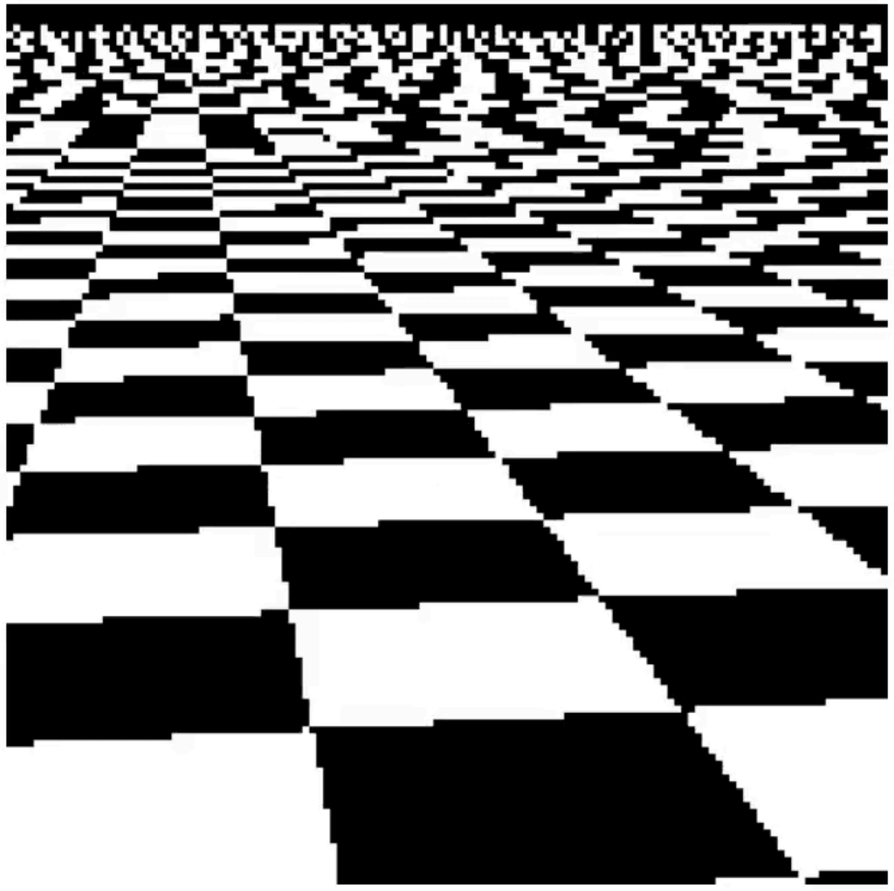


single sampling

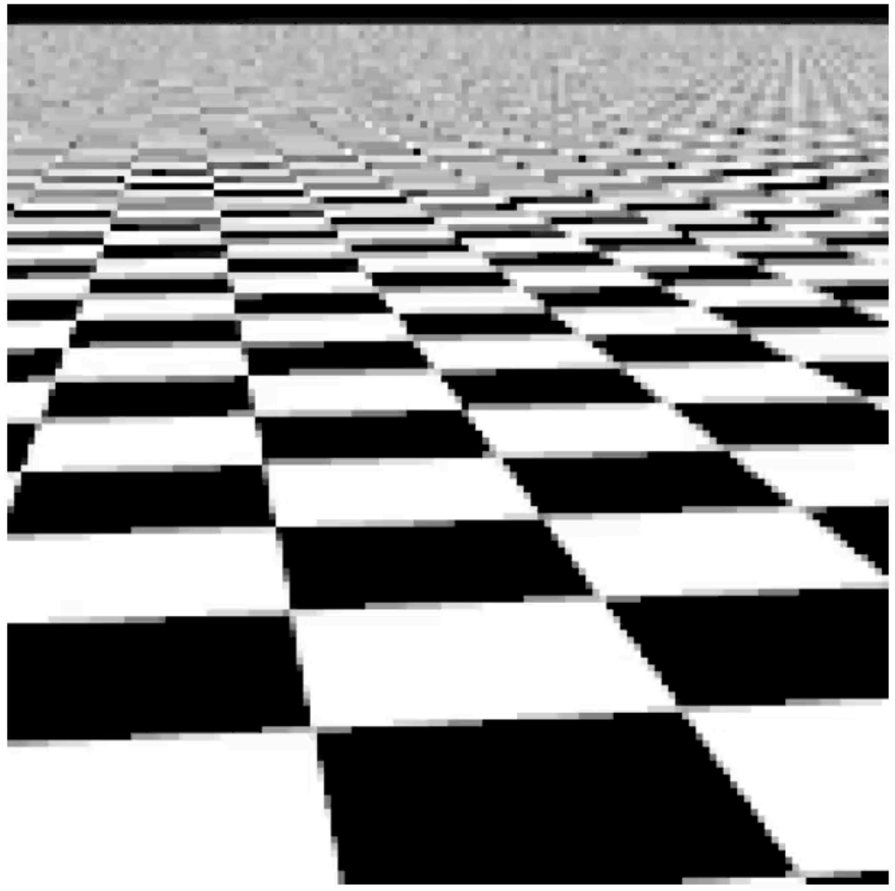


4x4 supersampling

Single sample vs. super sampling

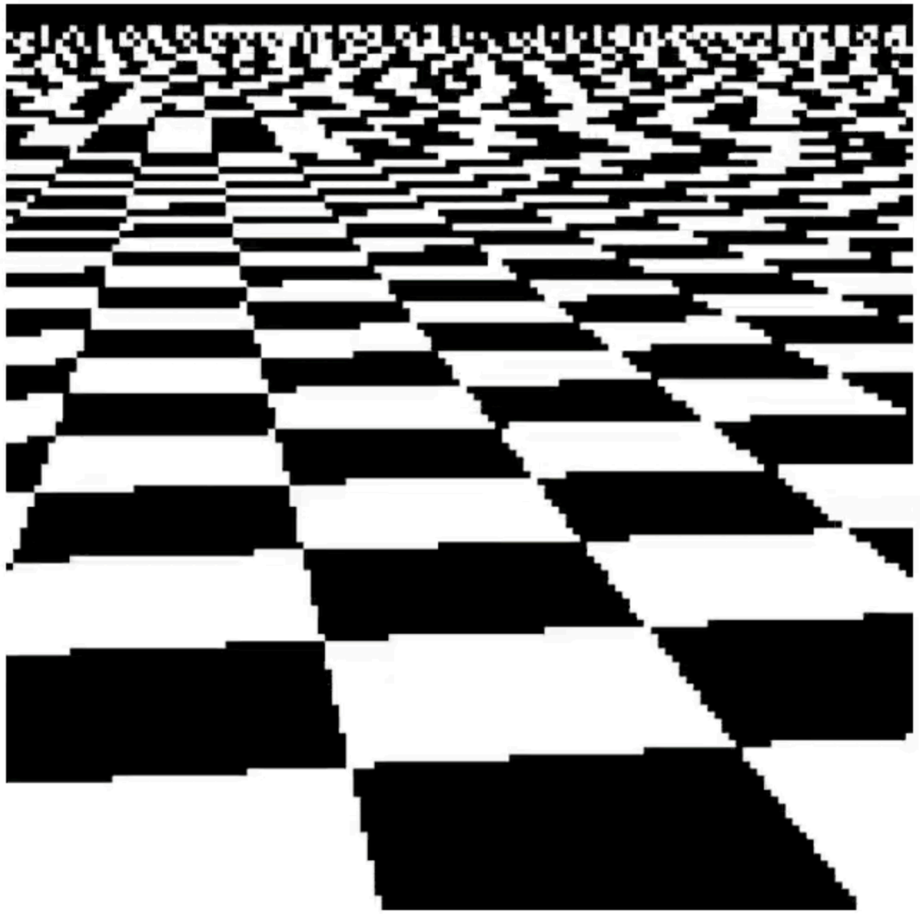


single sampling

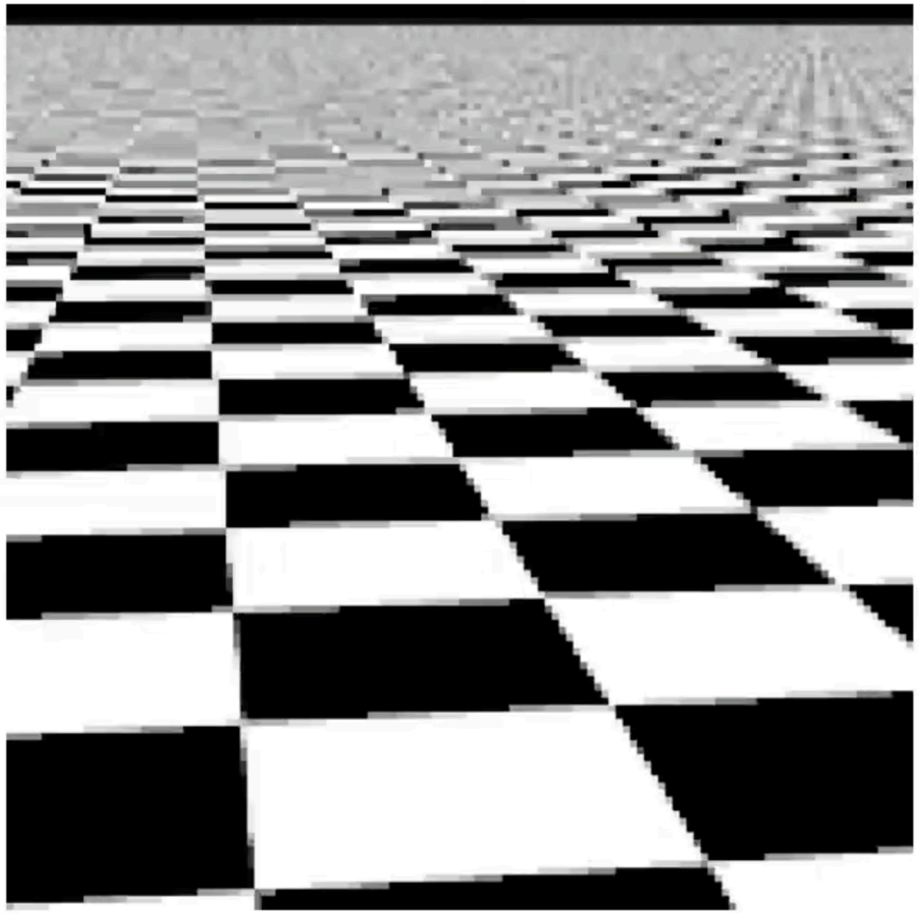


32x32 supersampling

Single sample vs. super sampling



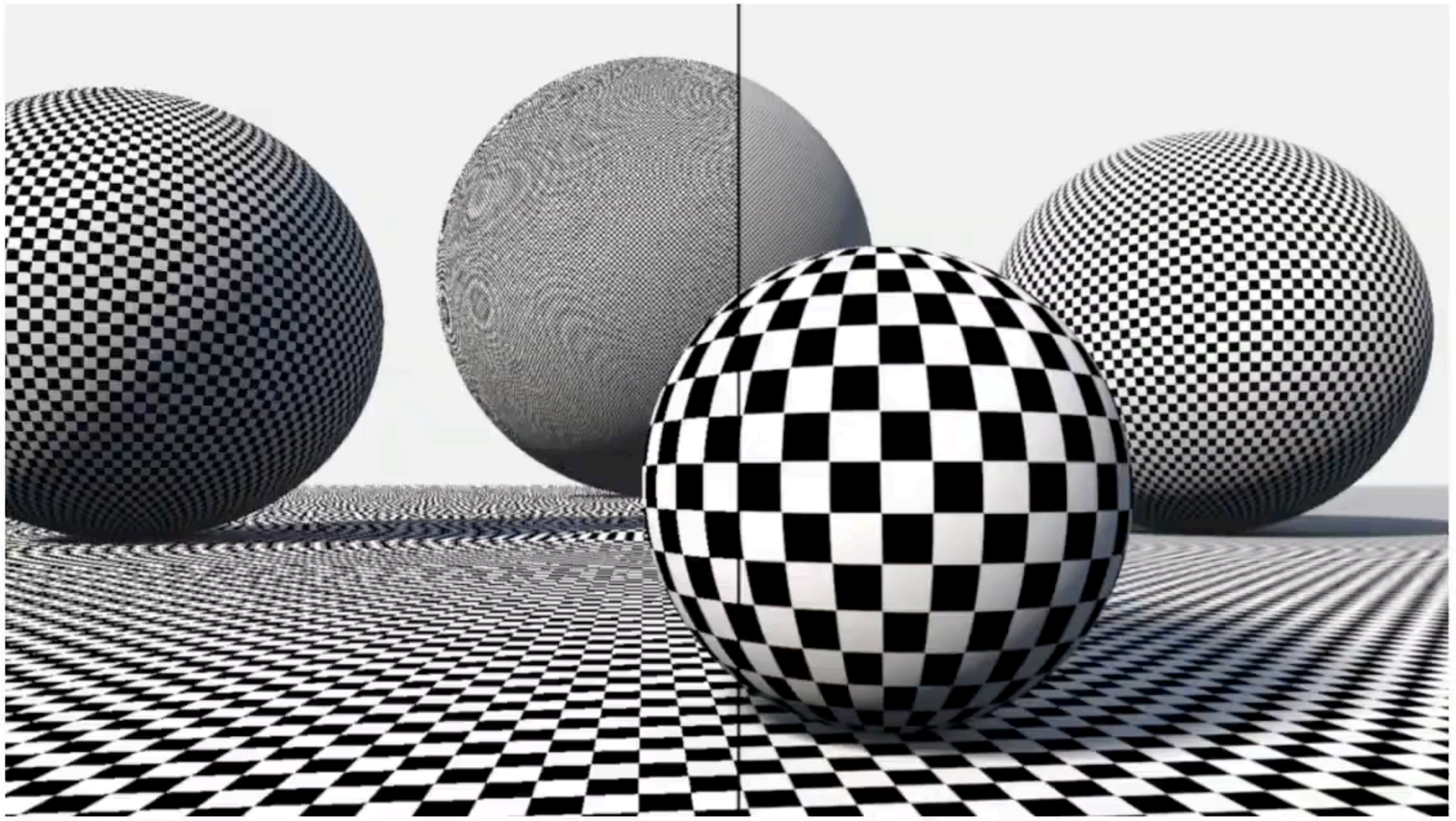
single sampling



32x32 supersampling

Checkerboard - Exact Solution

Special case: can compute the exact coverage of a checkerboard



Typical aliased result

Analytical Solution
Sampling+Filtering

See: Inigo Quilez, "Filtering the Checkerboard Pattern" & Apodaca et al, "Advanced Renderman" (p. 273)

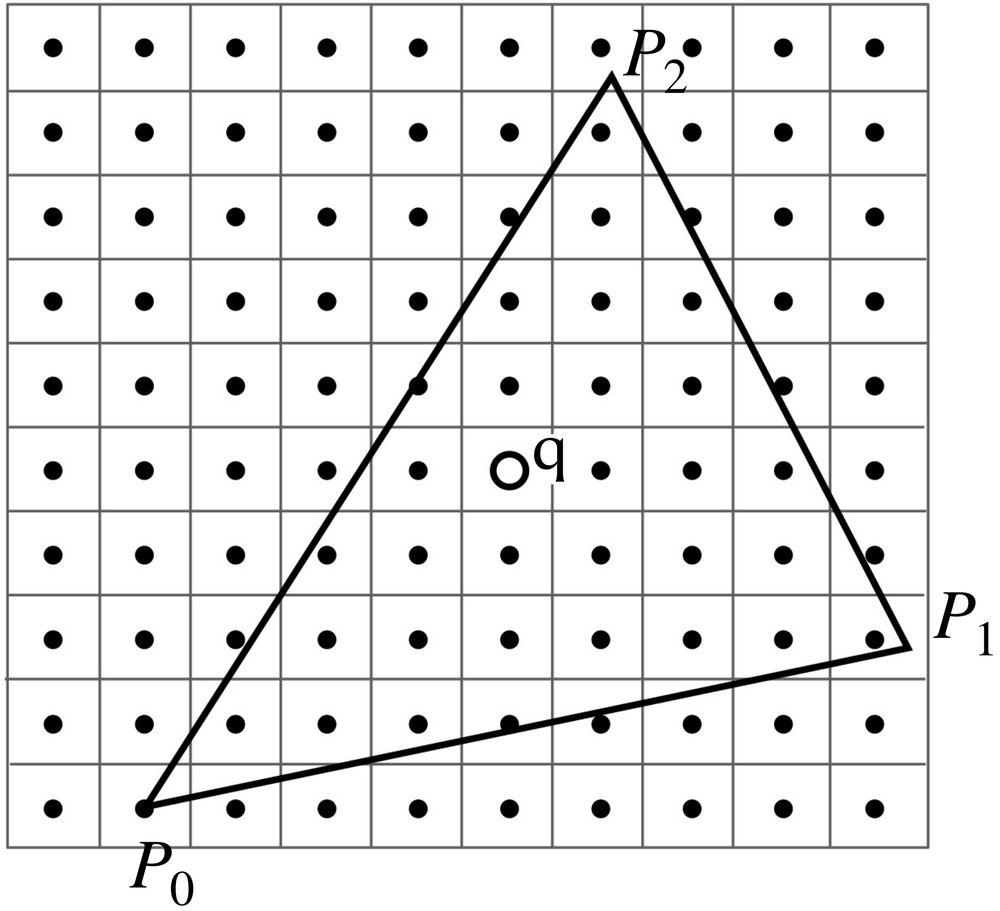
How to evaluate coverage(x,y) for a triangle?

Test whether a point is inside a triangle

“Point in triangle test”

- For a given point q, check whether it lies inside a triangle defined by P0, P1, P2.

  A: Check if it’s contained in three half planes associated with the edges.

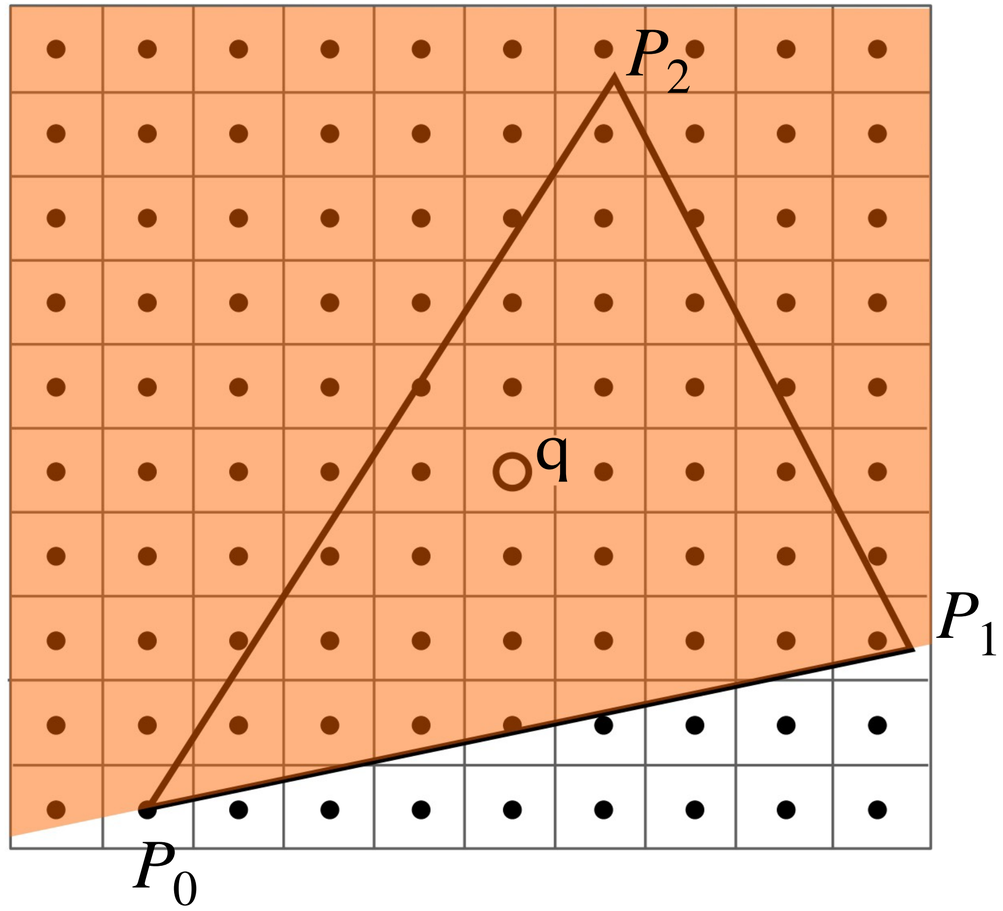


Test whether a point is inside a triangle

“Point in triangle test”

- For a given point q, check whether it lies inside a triangle defined by P0, P1, P2.

  A: Check if it’s contained in three half planes associated with the edges.

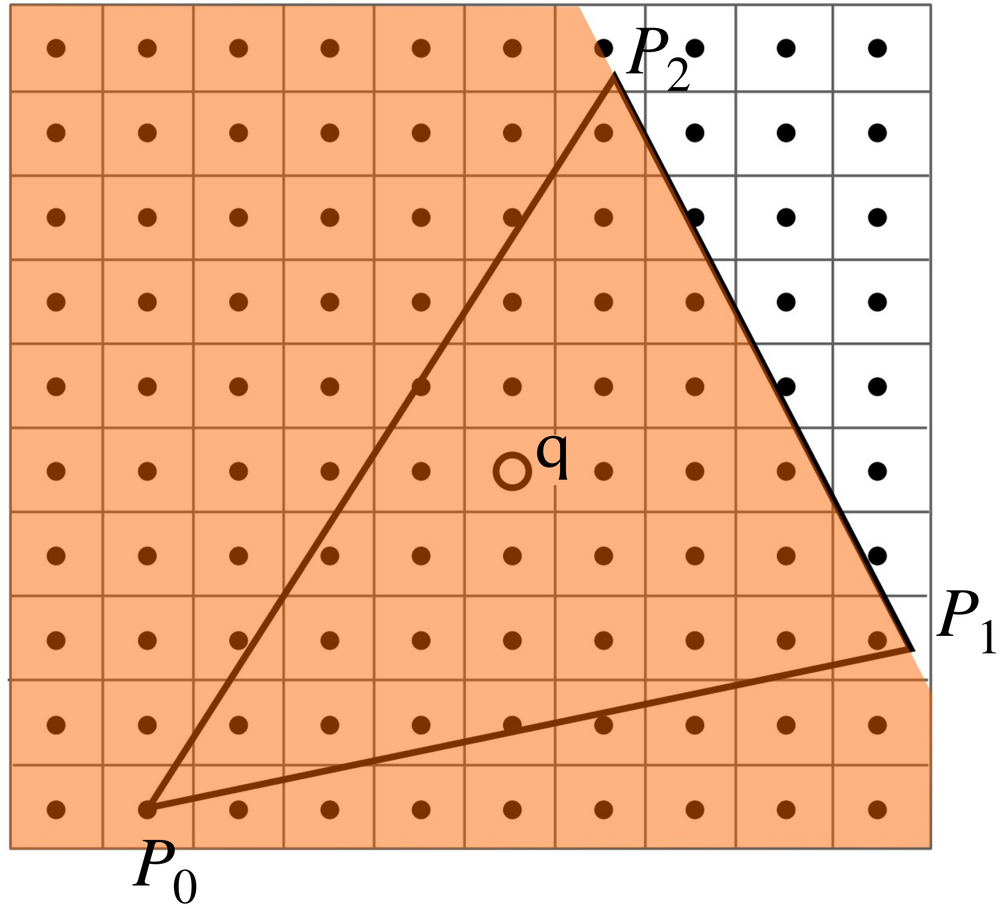


Test whether a point is inside a triangle

“Point in triangle test”

- For a given point q, check whether it lies inside a triangle defined by P0, P1, P2.

  A: Check if it’s contained in three half planes associated with the edges.



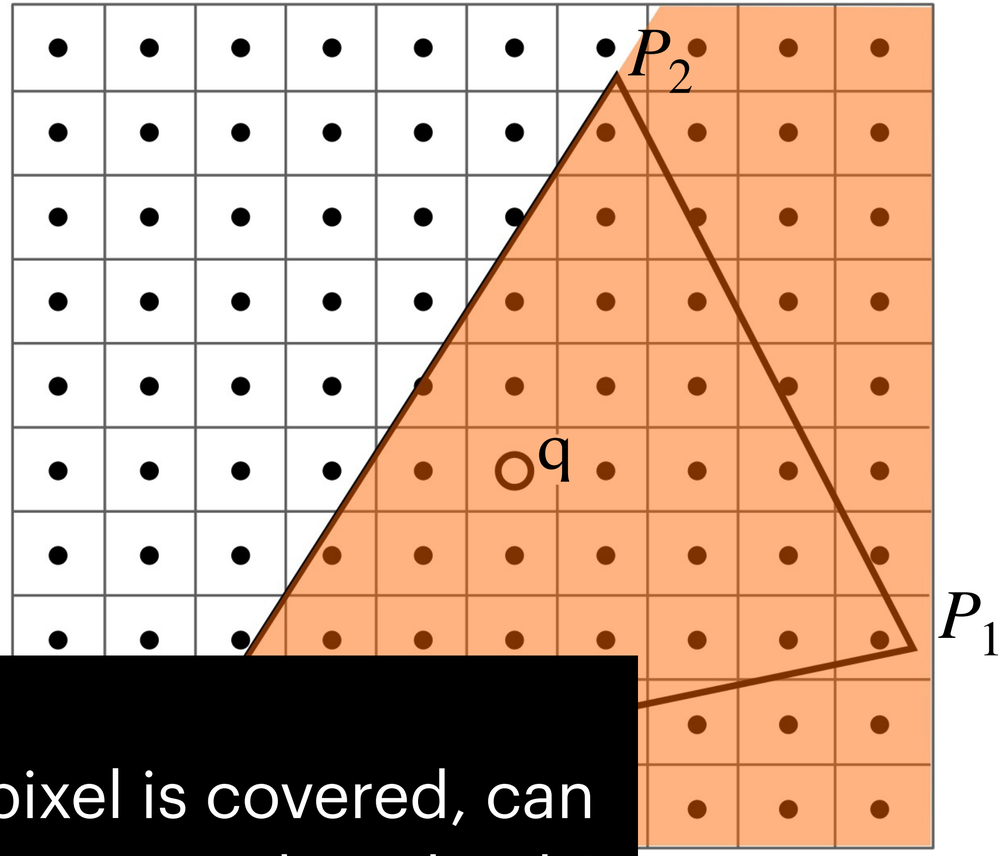
Test whether a point is inside a triangle

“Point in triangle test”

- For a given point q, check whether it lies inside a triangle defined by P0, P1, P2.

A: Check if it’s contained in three half planes associated with the edges.

You will implement this in your exercise!



GIVEN: two poi

FIND: whether q

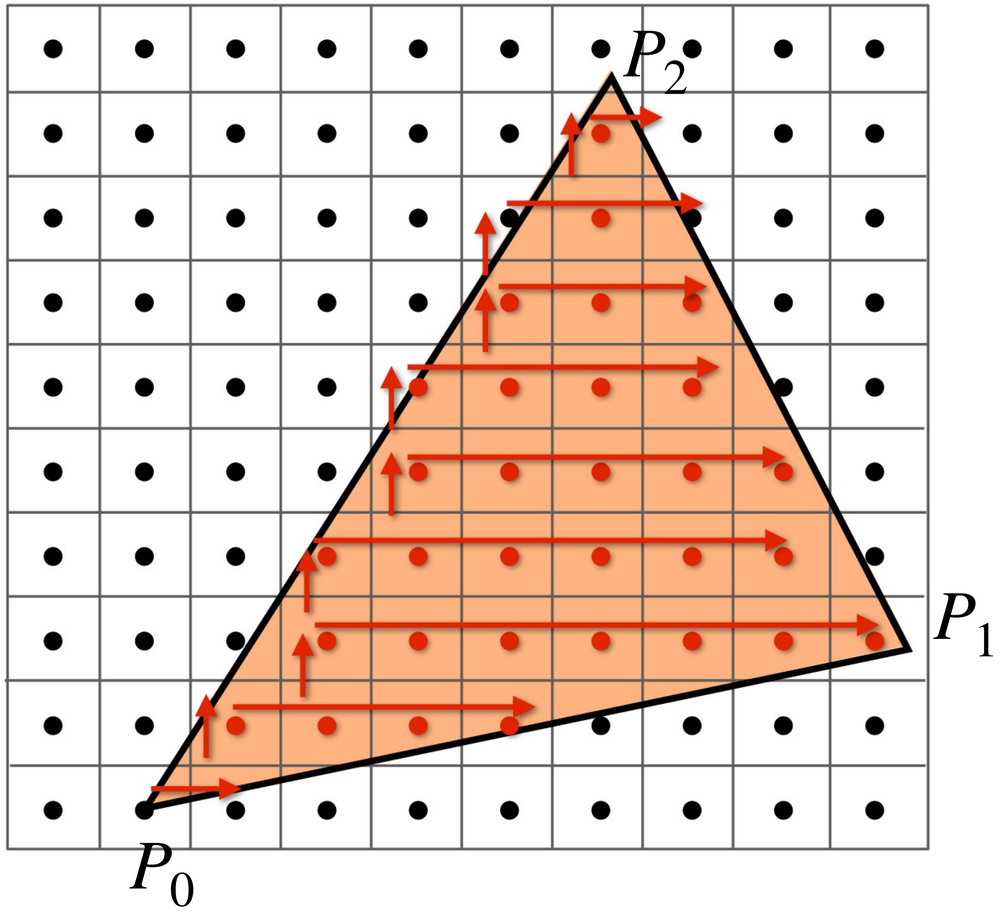
Q: We know how to check if a pixel is covered, can we be smart about which pixels we need to check?

Note the edge cases...

Traditional Approach: incremental traversal

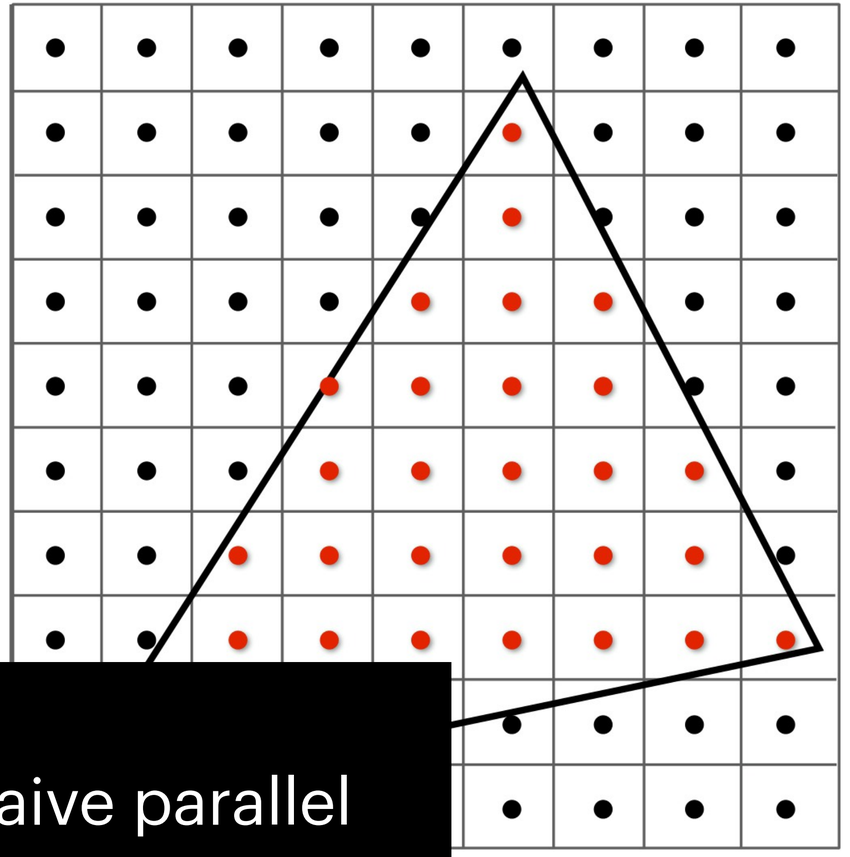
Half plane check looks very similar for different points, can save arithmetic by clever “incremental” schemes

Incremental approach also visits pixel in an order that improves memory coherence: backtrack, zig zag, Hilbert/ Morton curves...



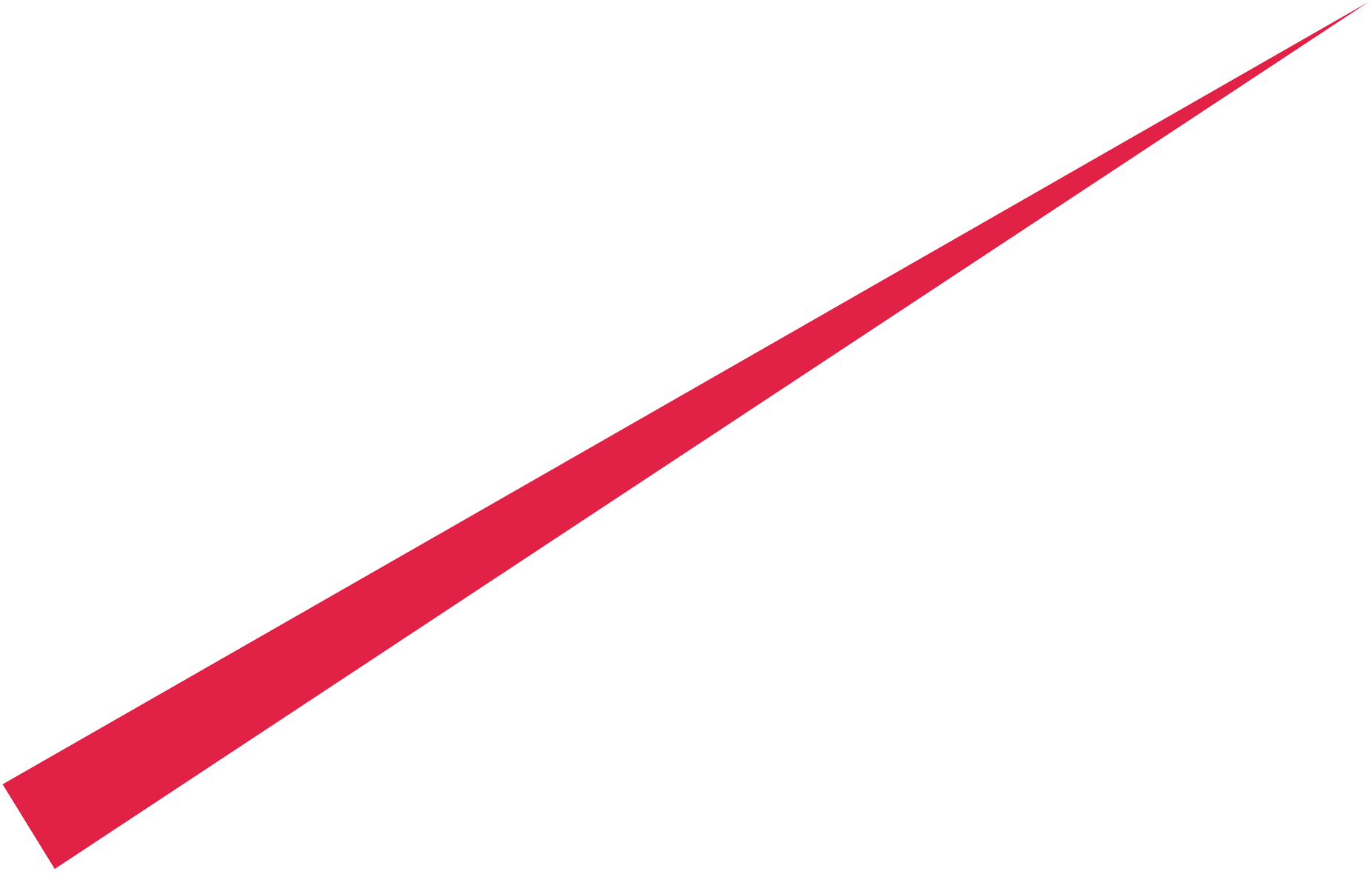
Modern approach: parallel coverage tests

- Incremental traversal is very serial; modern hardware is highly parallel
- Alternative: test all samples in triangle "bounding box" in parallel
- Wide parallel execution overcomes cost of extra tests (most triangles cover many samples, especially when super-sampling)
- All tests share s
- Modern GPUs
for efficiently p



Q: What's a case where the naive parallel approach is still very inefficient?

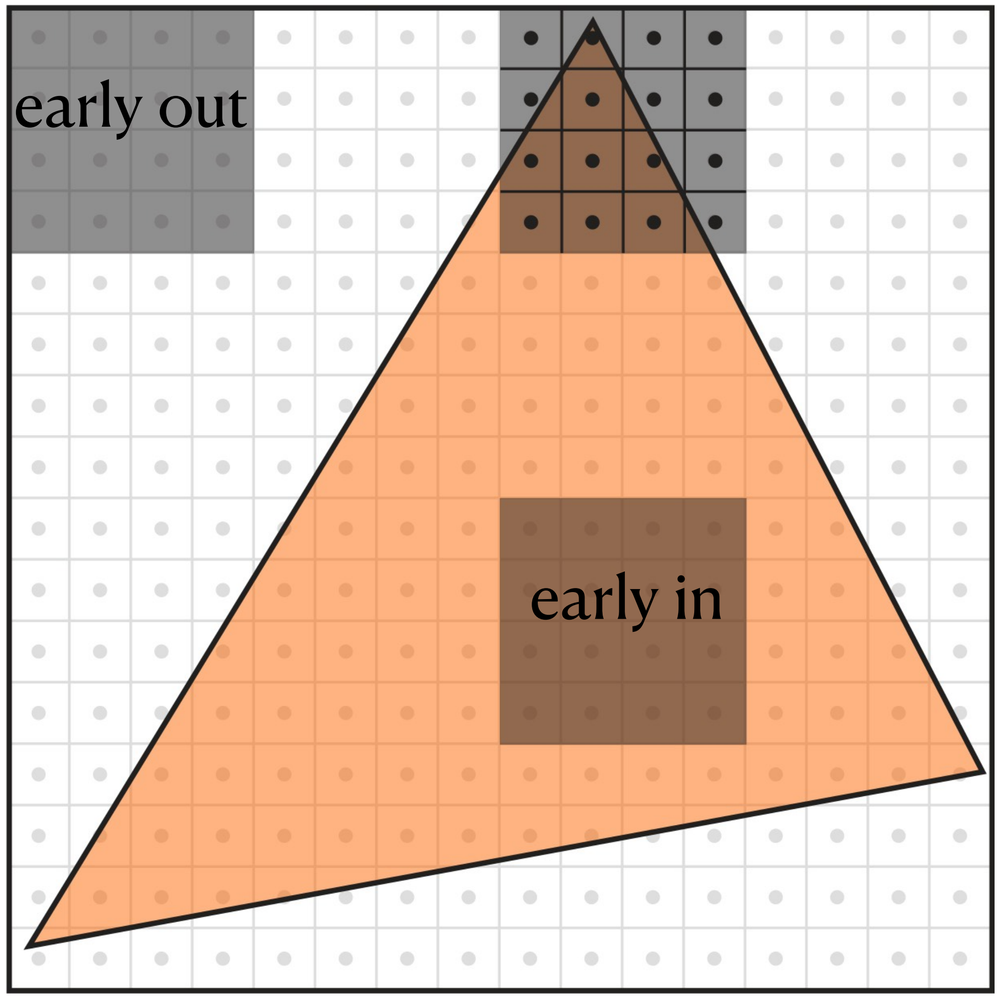
Naive approach can be wasteful



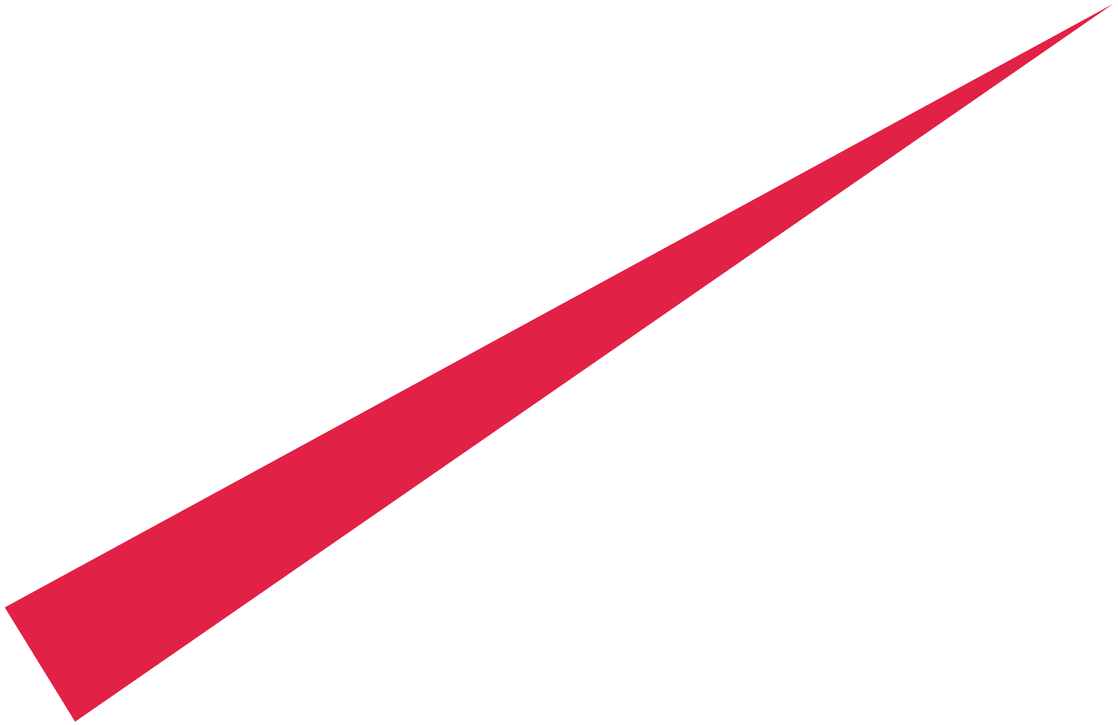
Hybrid approach: tiled triangle traversal

How real graphics hardware works!

- General idea: work “coarse to fine”
- First, check if large blocks intersect the triangle
- If not, skip this block entirely “early out”
- If the block is contained inside the triangle, know all samples are covered (“early in”)
- Otherwise, test individual sample points in the block, in parallel

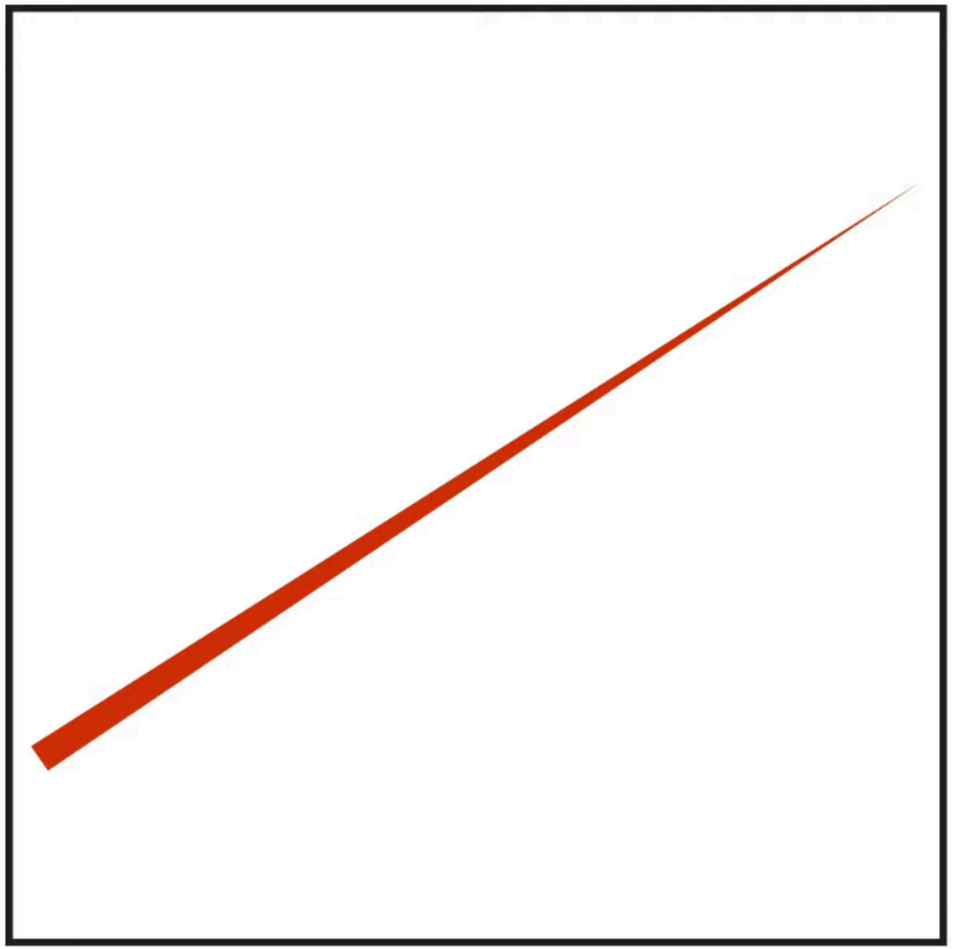


Can we do better?



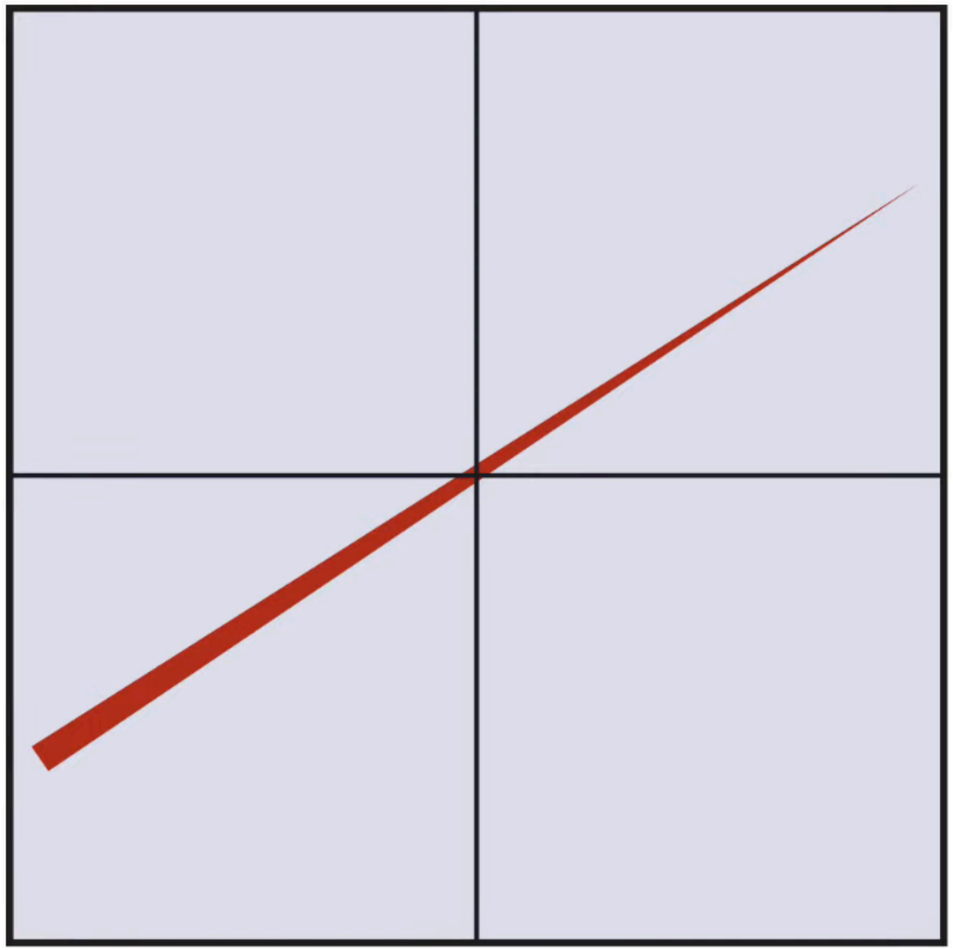
Hierarchal Strategies

Is the triangle inside this box?



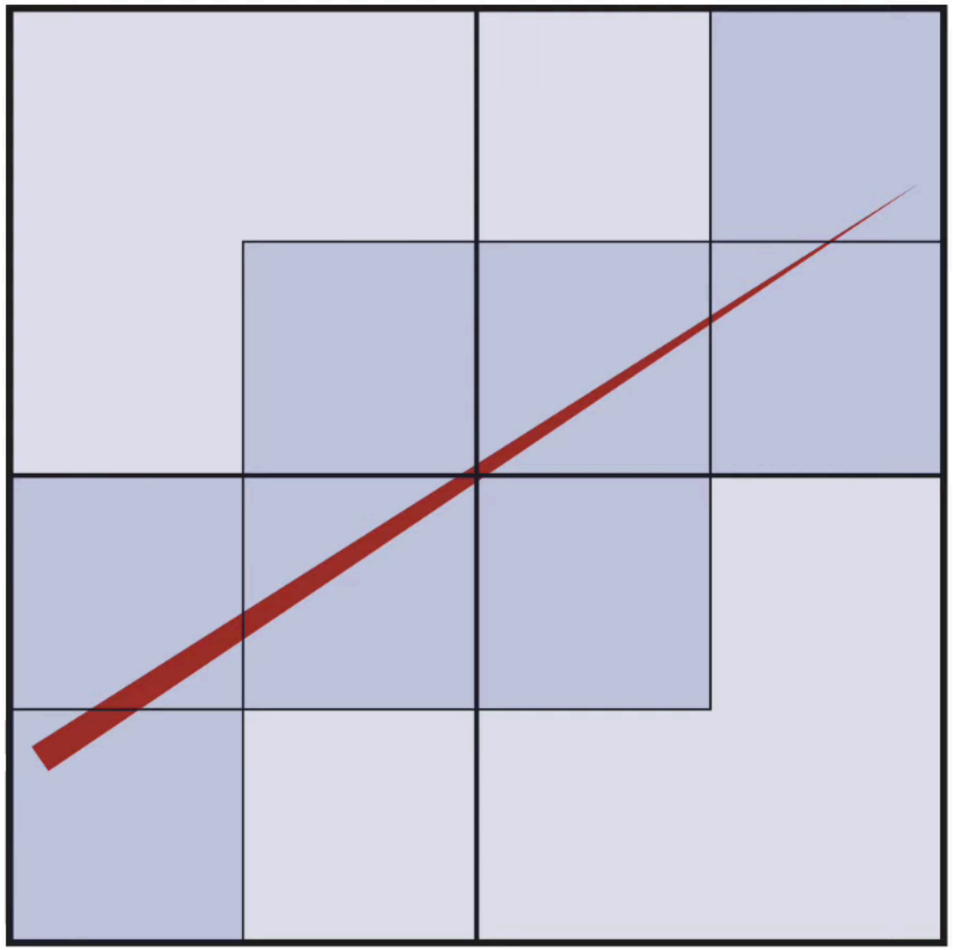
Hierarchal Strategies

Is the triangle
inside these boxes?

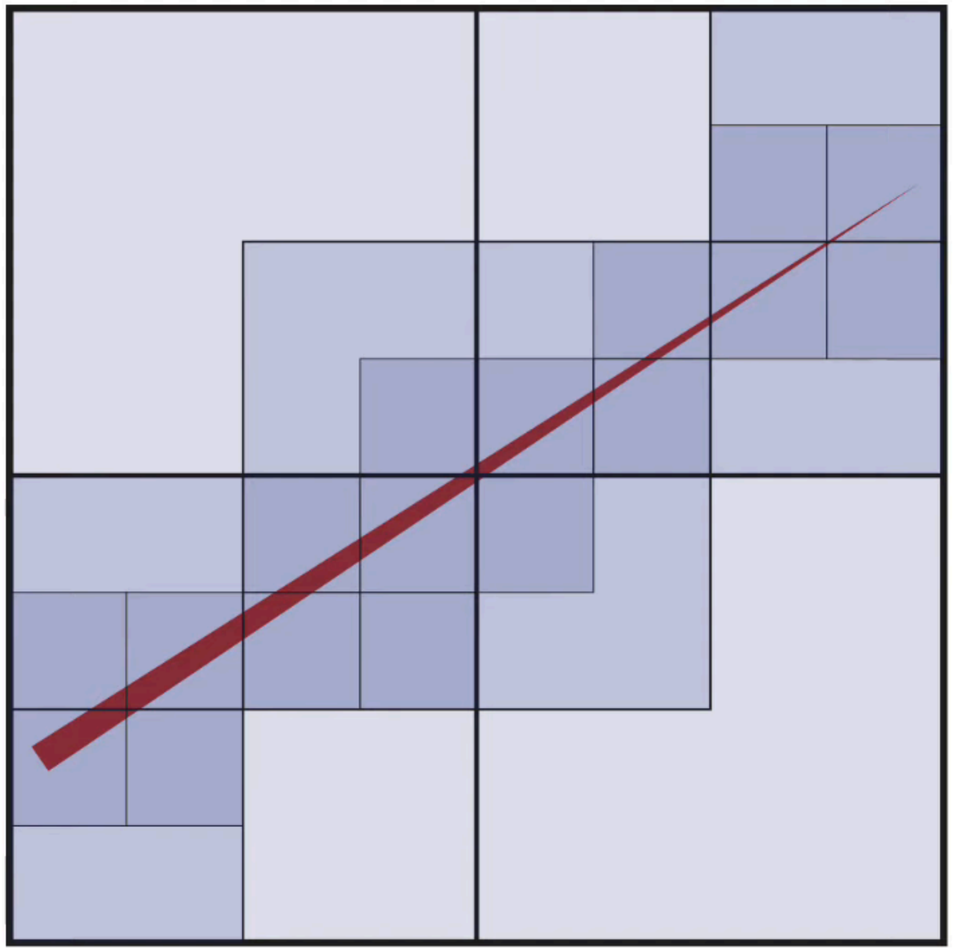


Hierarchal Strategies

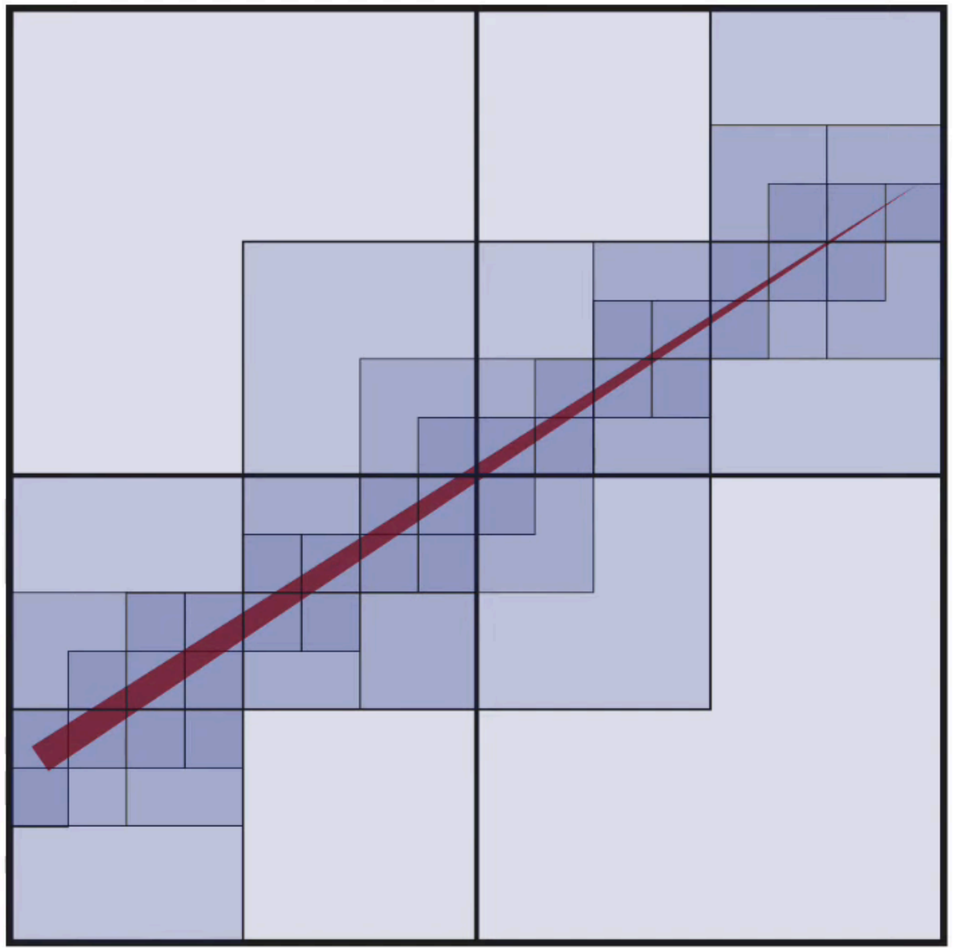
Recursively divide...



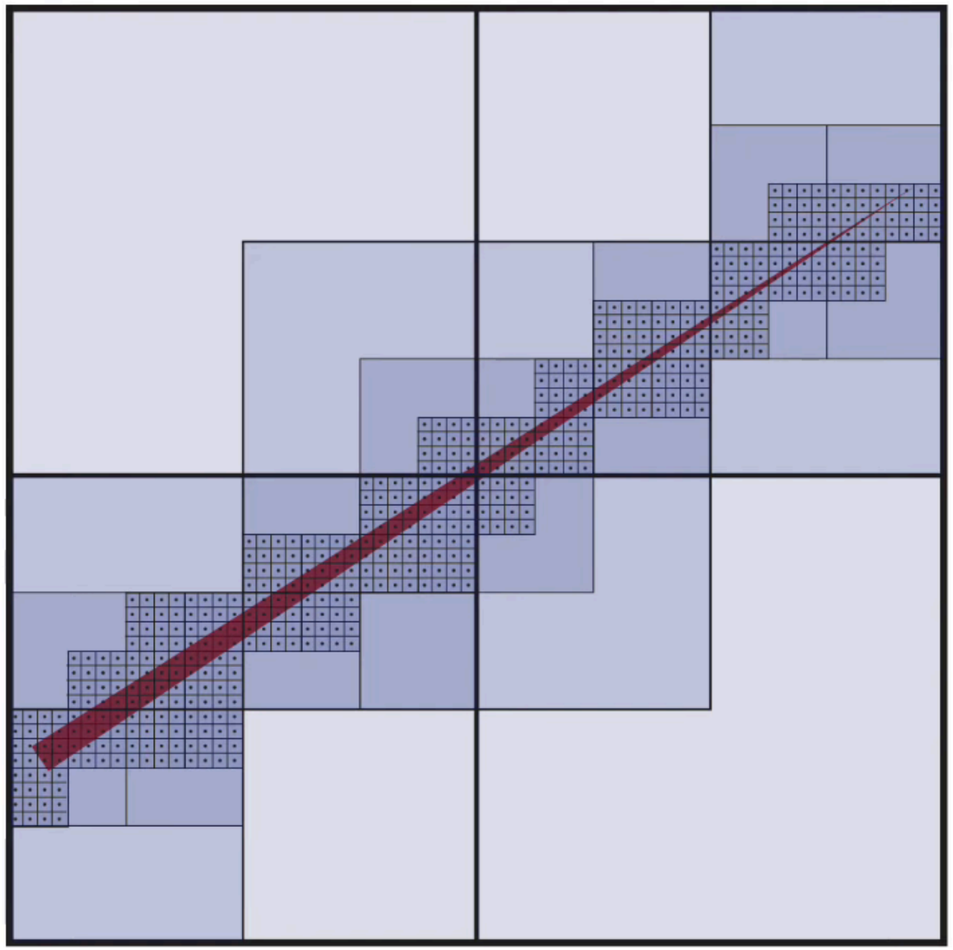
Hierarchal Strategies



Hierarchal Strategies



Hierarchal Strategies



Summary

- Many problems in graphics framed as sampling and reconstruction
  - **Sampling**: turn a continuous signal into discrete information
  - **Reconstruction**: turn discrete information into continuous signal
  - **Aliasing**: occurs when the reconstructed signal presents a false sense of what the original signal looks like
- Can frame rasterization as sampling problem
  - Sample coverage function into a pixel grid
  - Reconstruct by emitting a square of light for each pixel
  - Aliasing manifests as jagged edges, shimmering artifacts
  - Reduce aliasing via super sampling
- Triangle rasterization is basic building block for graphics pipeline
  - Amounts to 3 half-plane tests
  - Atomic operation - make it fast!
  - Several strategies: incremental, parallel, clockwise, hierarchical….