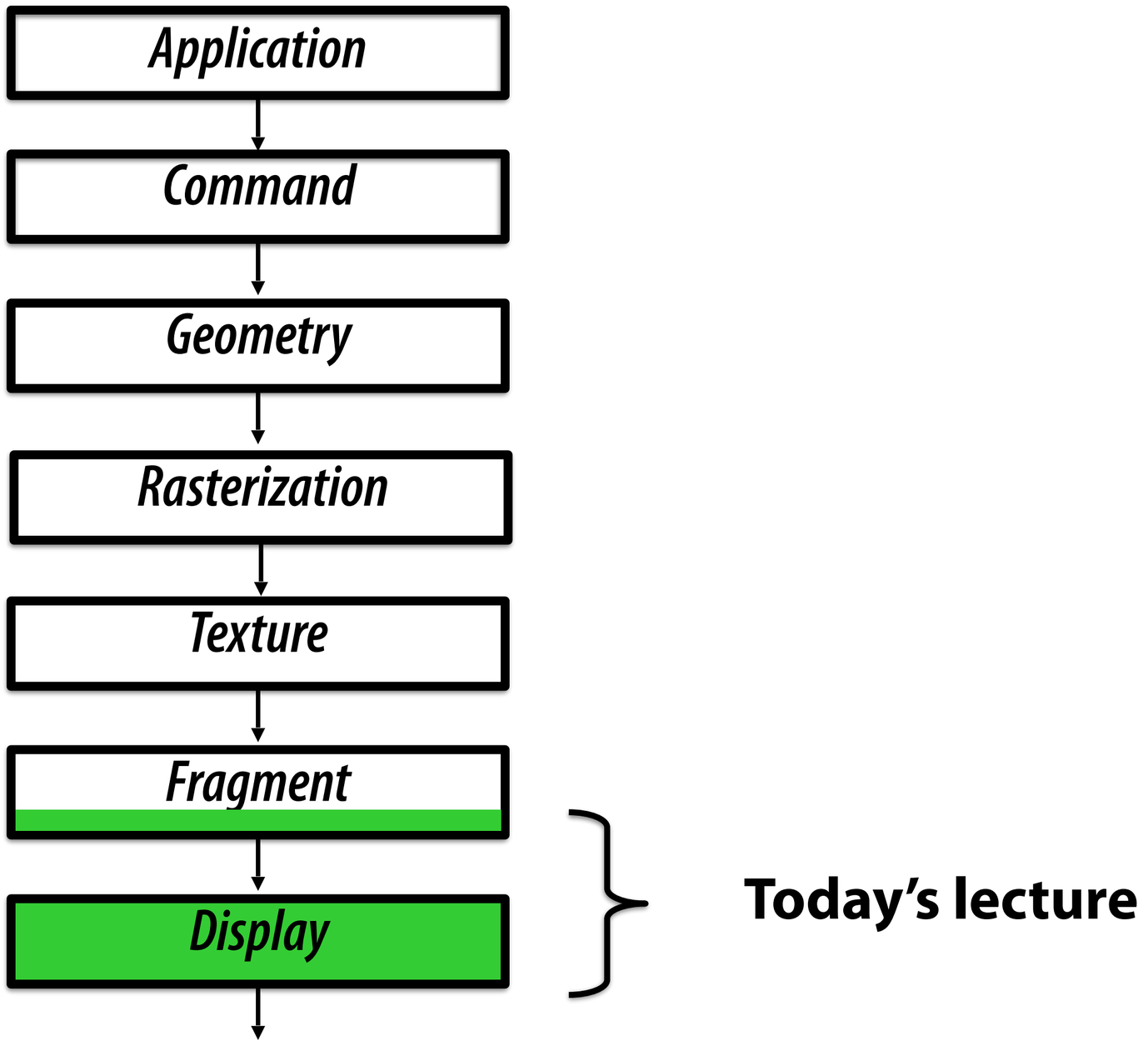
Lecture 10:

Composition/Display

EEC 277, Graphics Architecture
John Owens, UC Davis, Winter 2017


Application
Command
Geometry
Rasterization
Texture
Fragment
Display
Today's lecture

Outline

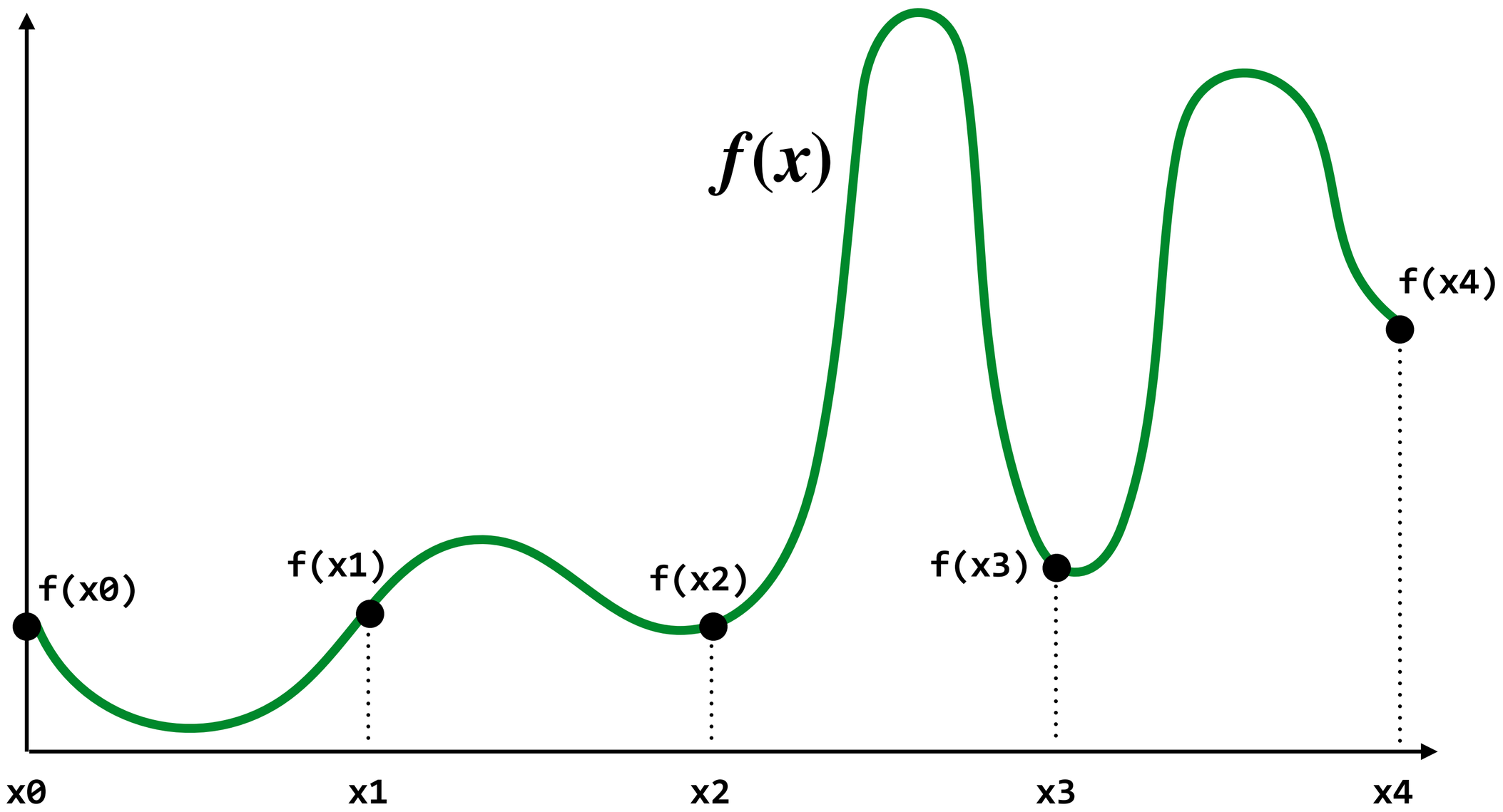
- Antialiasing
- Compositing
- Depth Buffer
- Monitors

What is a pixel?

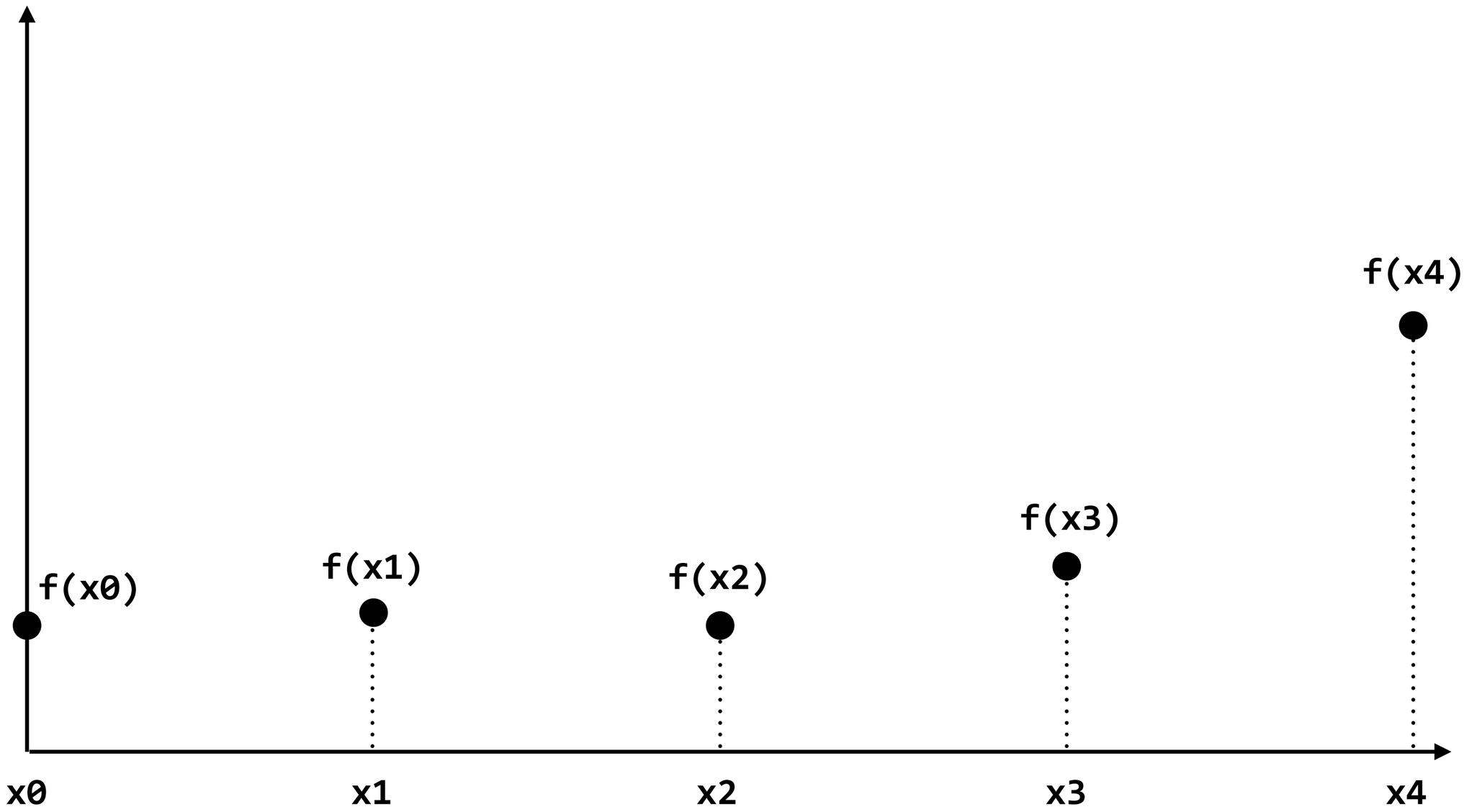
- "A Pixel Is Not A Little Square, A Pixel Is Not A Little Square, A Pixel Is Not A Little Square! (And a Voxel is Not a Little Cube)", Alvy Ray Smith, Microsoft Tech Memo 6
- A pixel is a point sample
- An image is a grid of point samples
- … can be reconstructed into a continuous function by
  - The Sampling Theorem [Shannon]
  - Your eye
- A pixel represents the contributions of an area in screen space—area integral
- Point sampling = approximation of that area with single point

Sampling: taking measurements a signal

Below: five measurements ("samples") of 1D signal $f(x)$



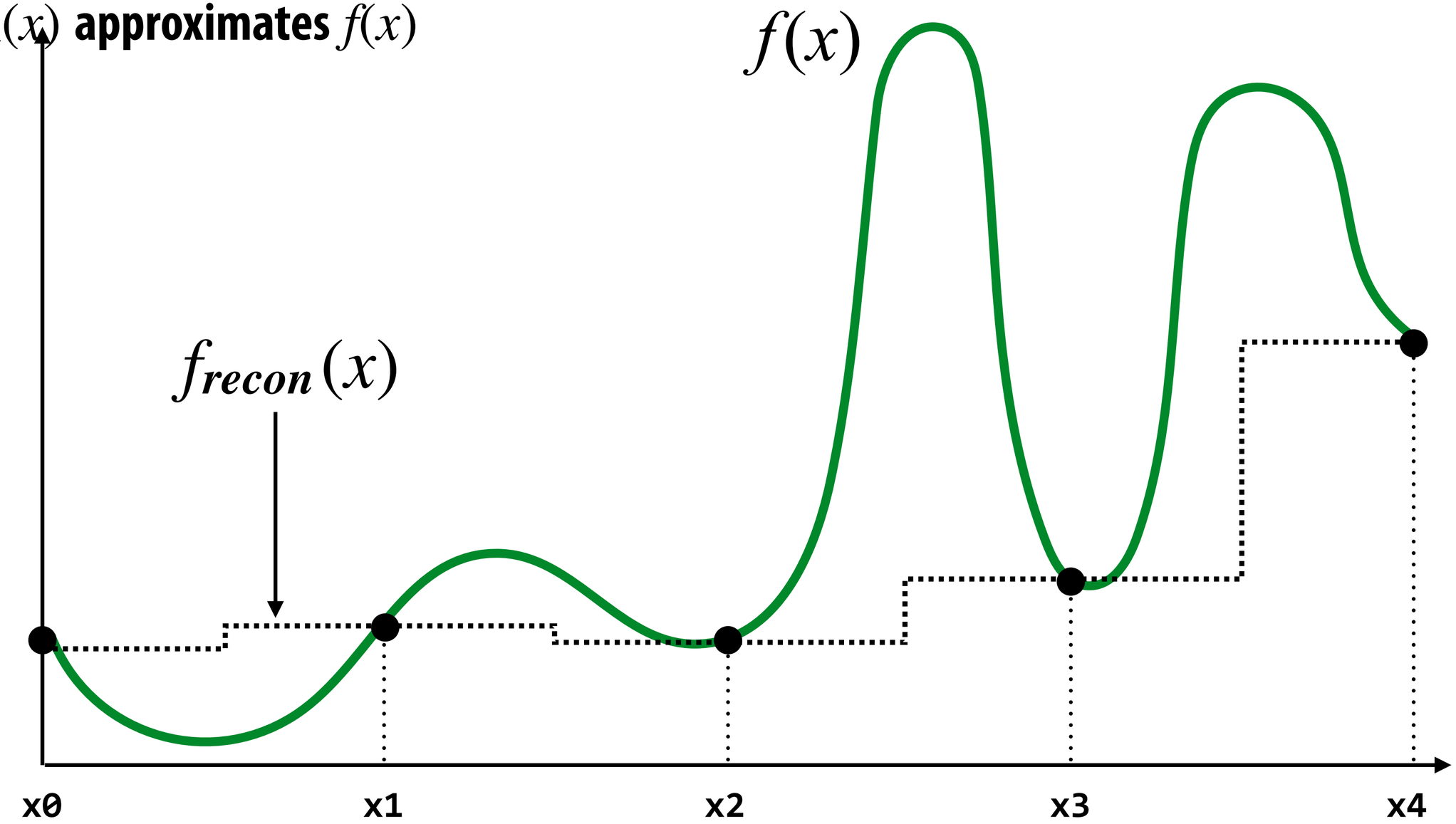
Reconstruction: given a set of samples, how might we attempt to reconstruct the original signal $f(x)$?



Piecewise constant approximation

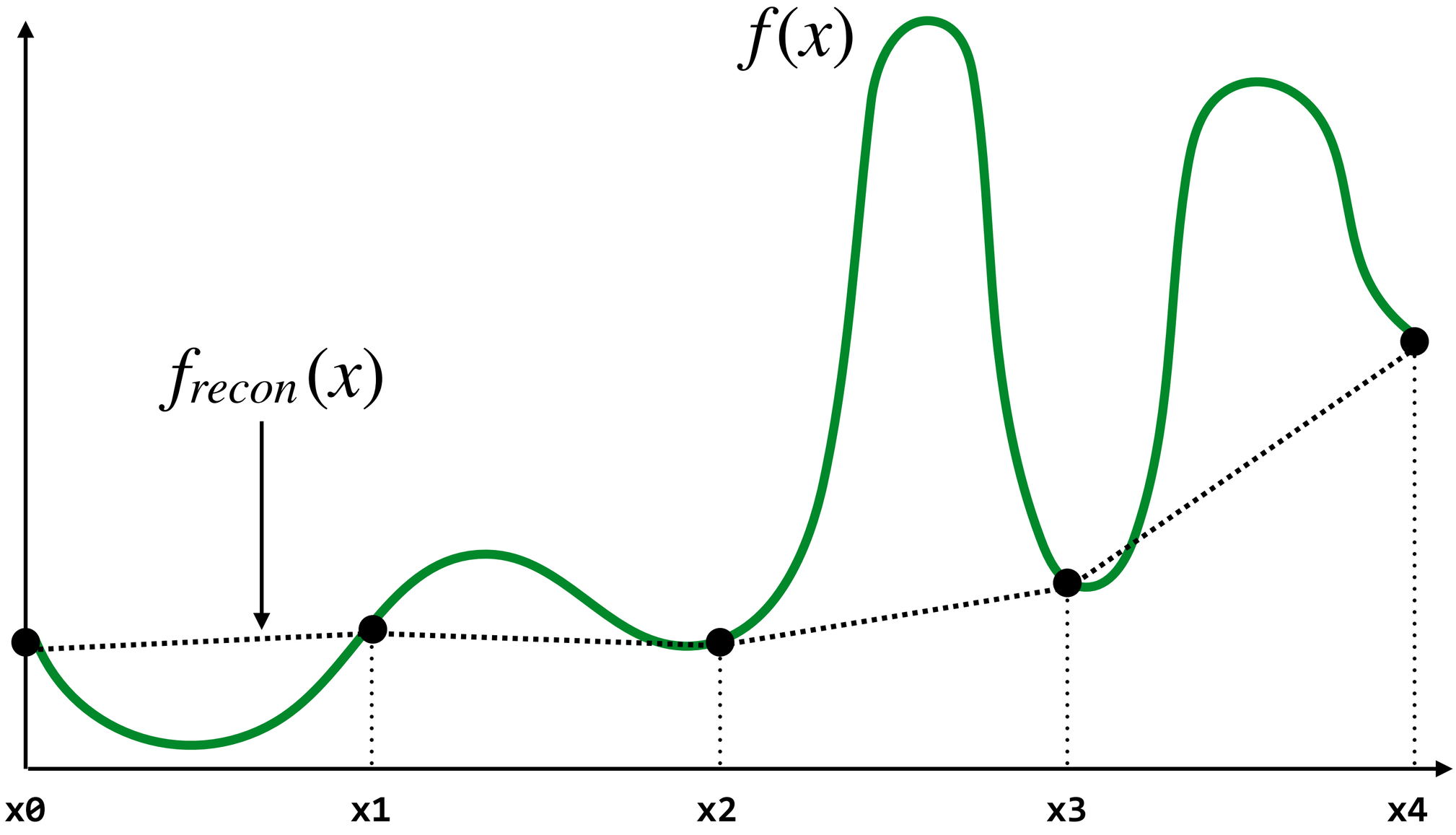
$f_{recon}(x)$ = **value of sample closest to** x

$f_{recon}(x)$ **approximates** $f(x)$

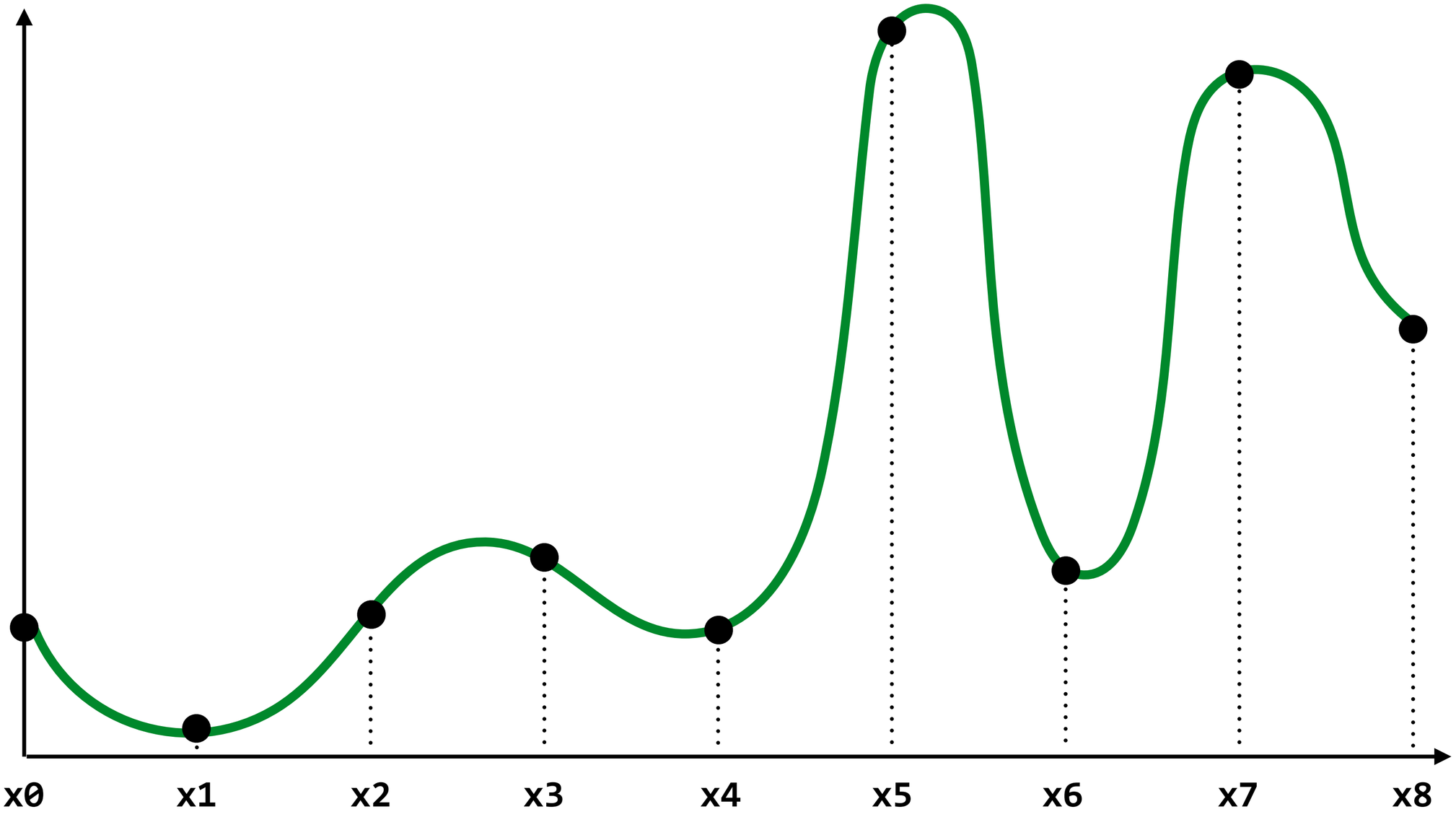


Piecewise linear approximation

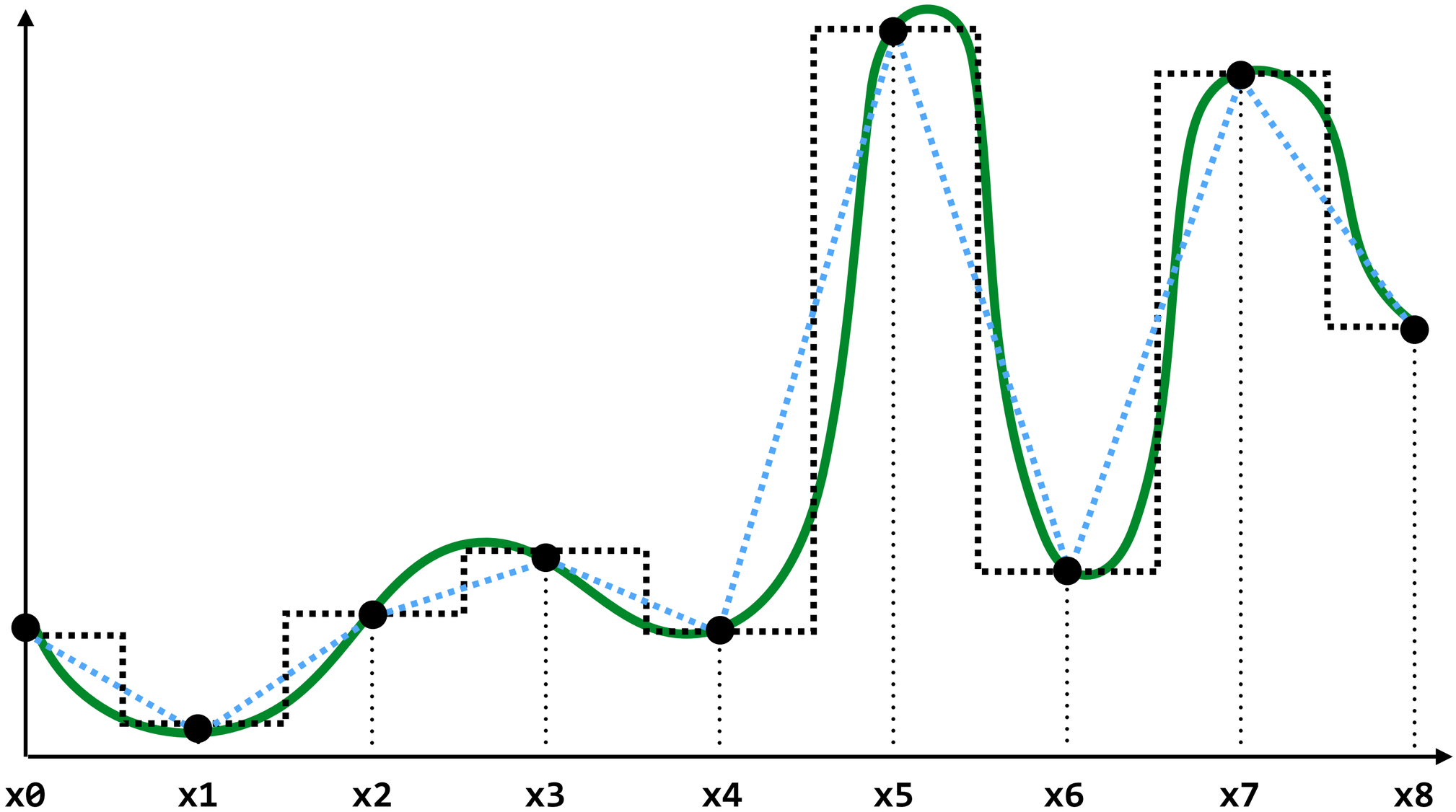
$f_{recon}(x)$ = linear interpolation between values of two closest samples to x



Sampling signal more densely (increasing sampling rate) enables more accurate reconstruction



Reconstruction from denser sampling



= reconstruction via nearest

= reconstruction via linear interpolation

Reconstruction as convolution (box filter)

Sampled signal:

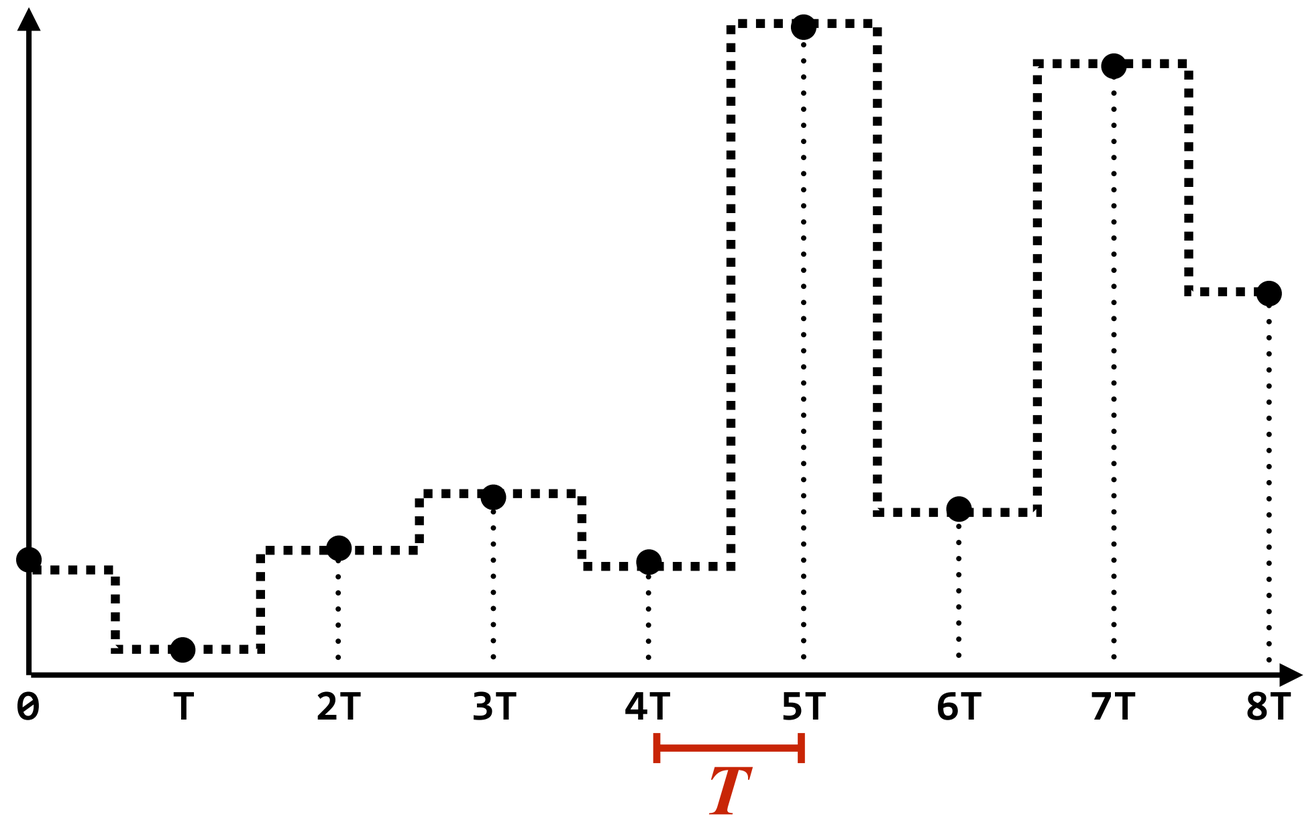
(with period T)

$$g(x) = \text{Ш}_T(x) f(x) = T \sum_{i=-\infty}^{\infty} f(iT)\delta(x - iT)$$

Reconstruction filter:

(unit area box of width T)

$$h(x) = \begin{cases} 1/T & |x| \leq T/2 \\ 0 & otherwise \end{cases}$$



Reconstructed signal:

(chooses nearest sample)

$$f_{recon}(x) = (h * g)(x) = T \int_{-\infty}^{\infty} h(y) \sum_{i=-\infty}^{\infty} f(iT)\delta(x - y - iT)dy = \int_{-T/2}^{T/2} \sum_{i=-\infty}^{\infty} f(iT)\delta(x - y - iT)$$

non-zero only for iT closest to x

Reconstruction as convolution (triangle filter)

Sampled signal:

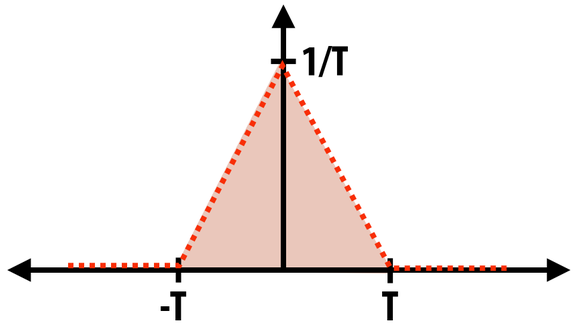
(with period T)

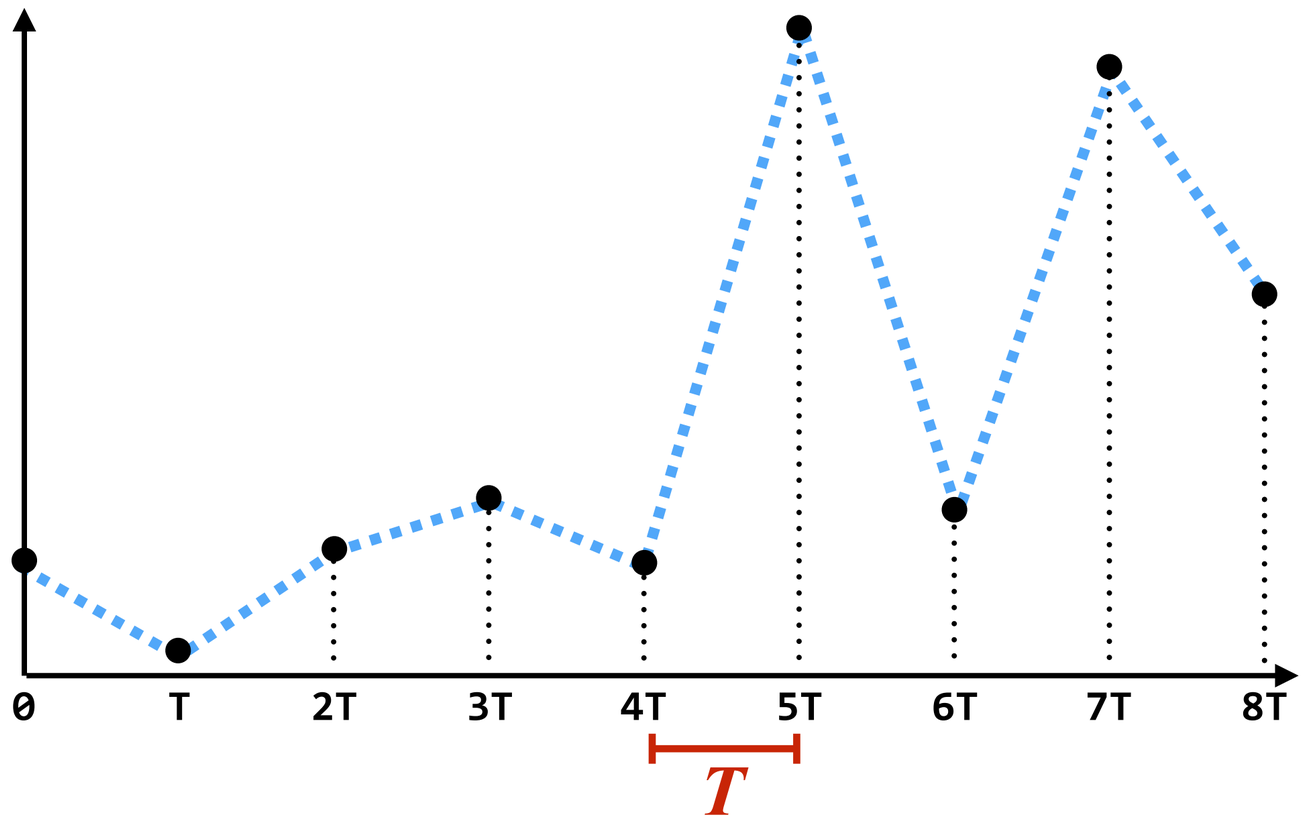
$$g(x) = \text{Ш}_T(x) f(x) = T \sum_{i=-\infty}^{\infty} f(iT)\delta(x - iT)$$

Reconstruction filter:

(unit area triangle of width T)

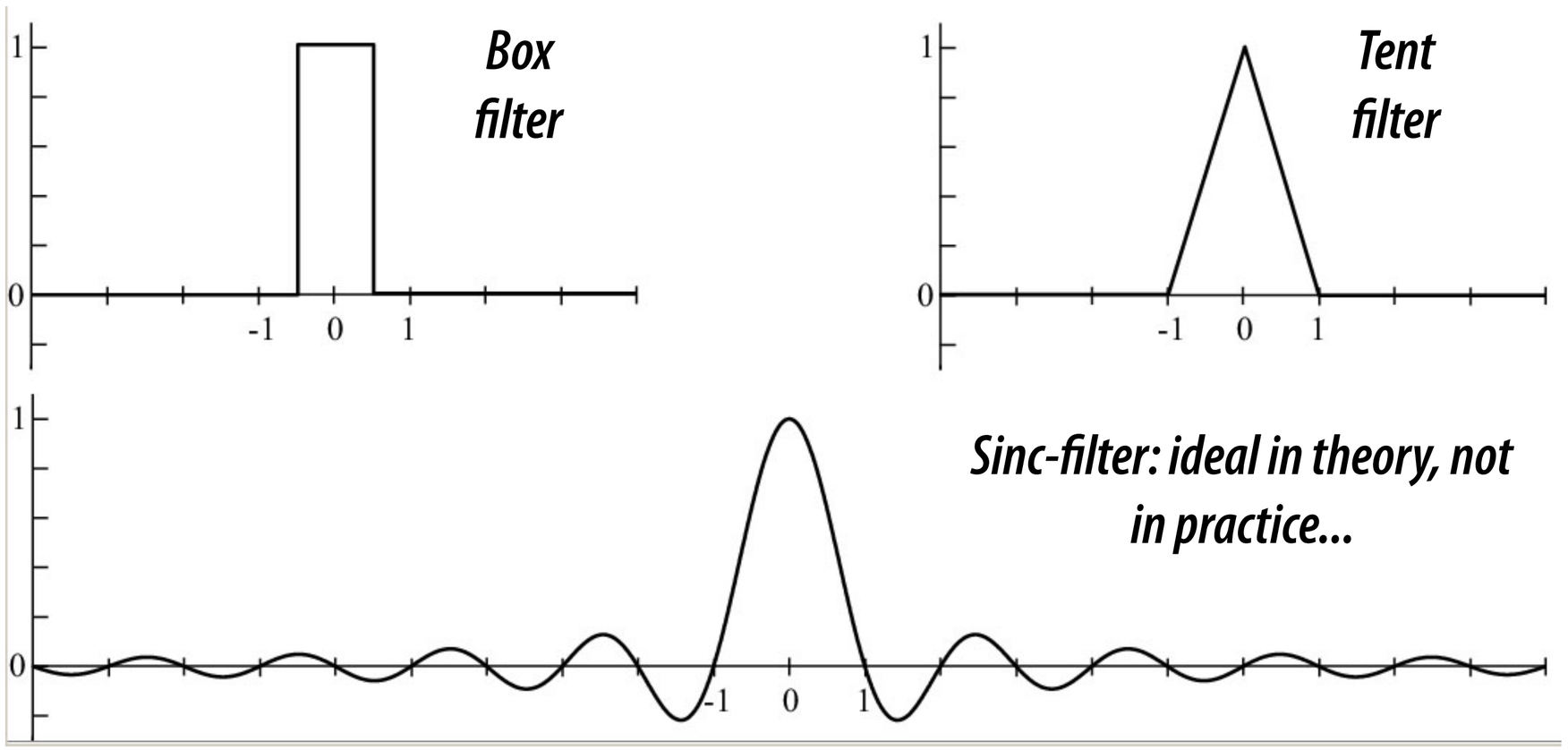
$$h(x) = \begin{cases} (1 - \frac{|x|}{T})/T & |x| \leq T \\ 0 & otherwise \end{cases}$$





Sampling theory

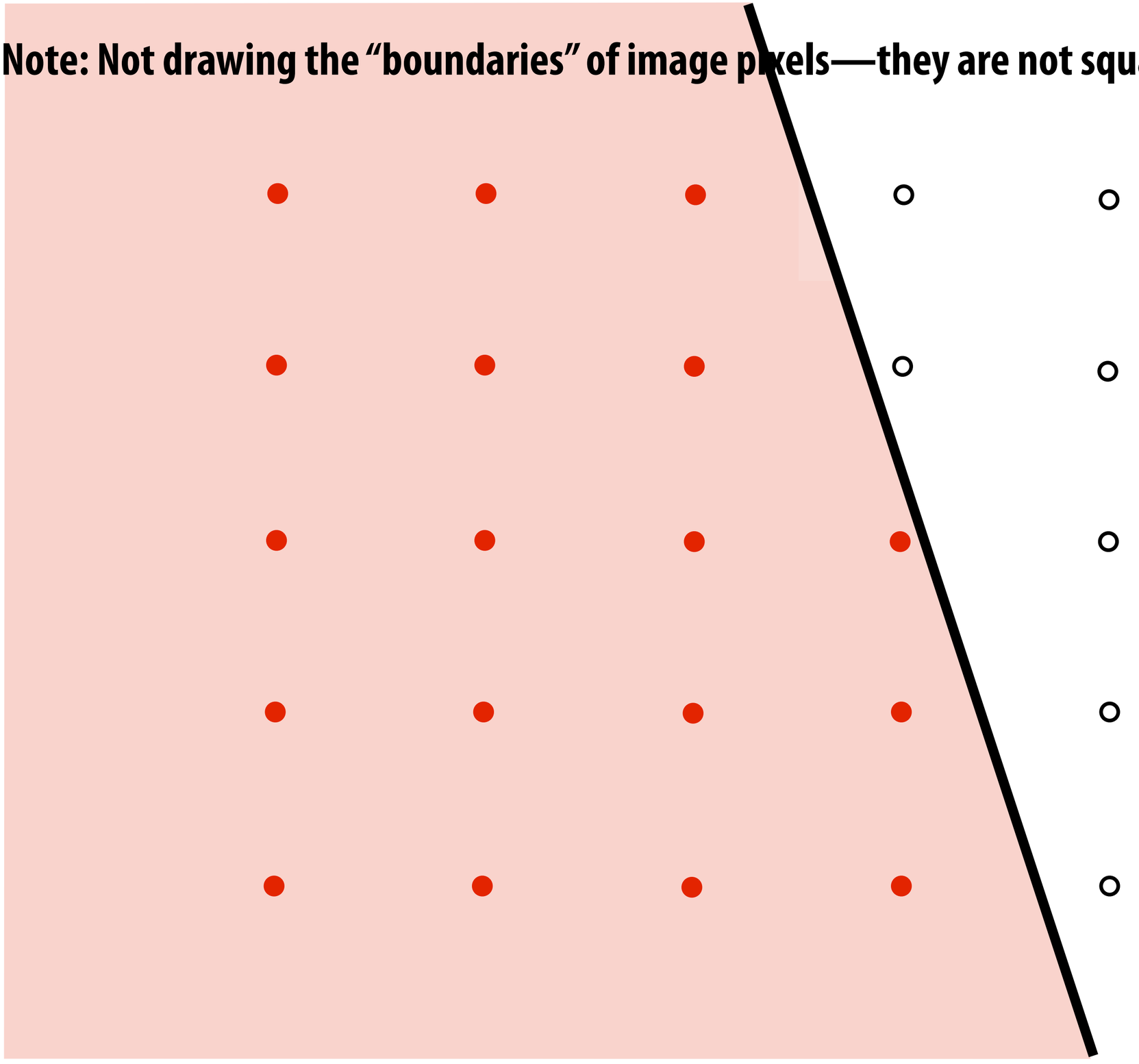
- The filter weights, w_i
  - The figure below assume a pixel starts at -0.5 and ends at +0.5



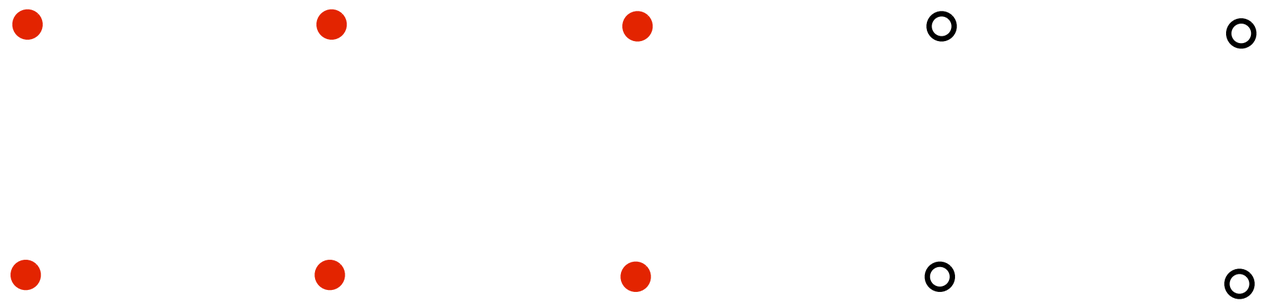
[courtesy Tomas Akenine-Möller]

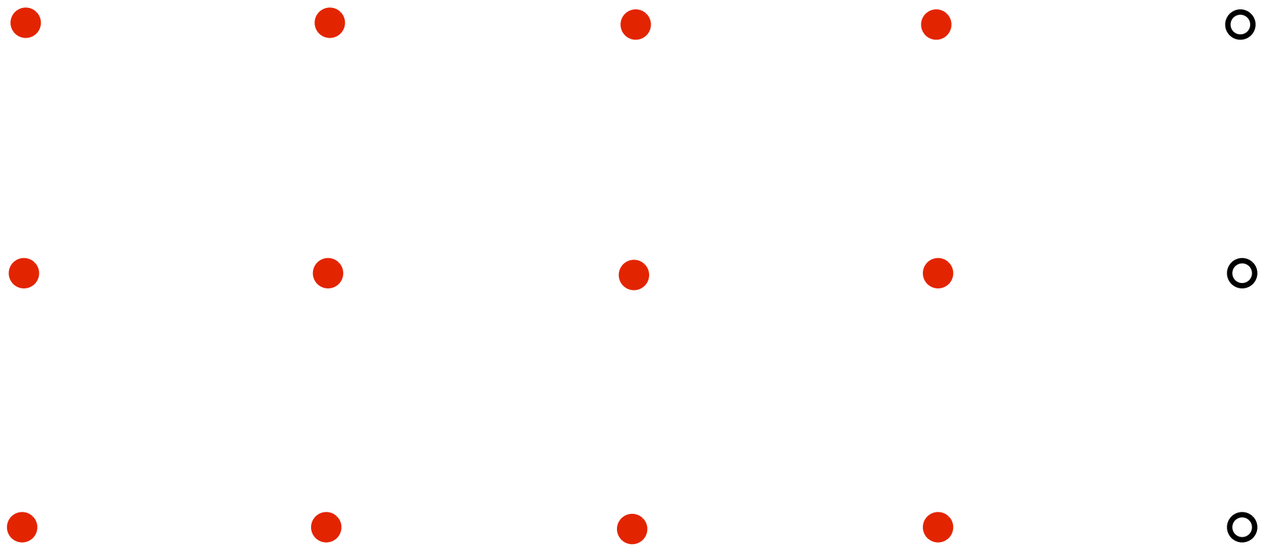
Results of sampling triangle coverage

Note: Not drawing the "boundaries" of image pixels—they are not squares.



I have a sampled signal, now I want to display it on a screen

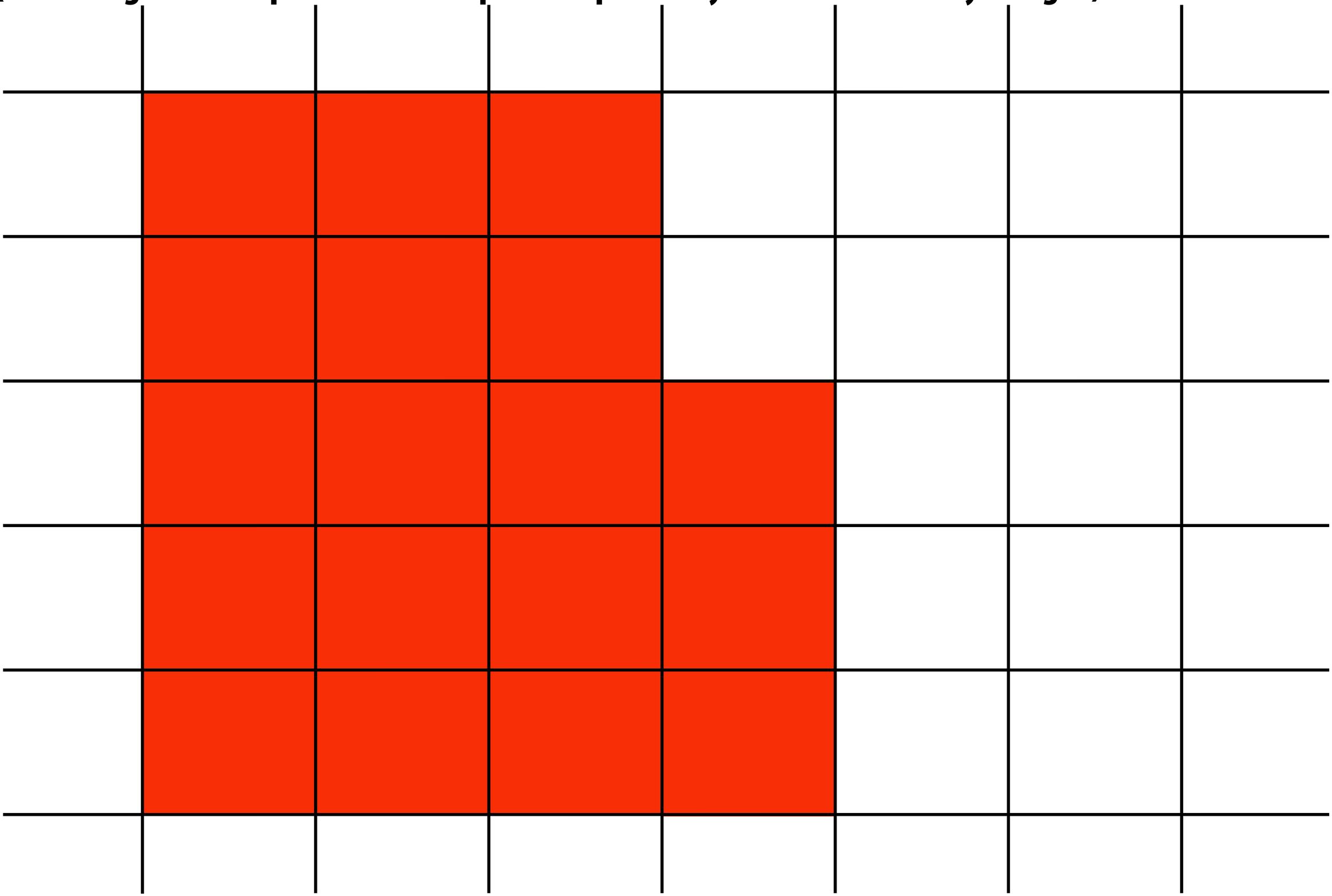




So if we send the display this…

We might see this when we look at the screen

(assuming a screen pixel emits a square of perfectly uniform intensity of light)



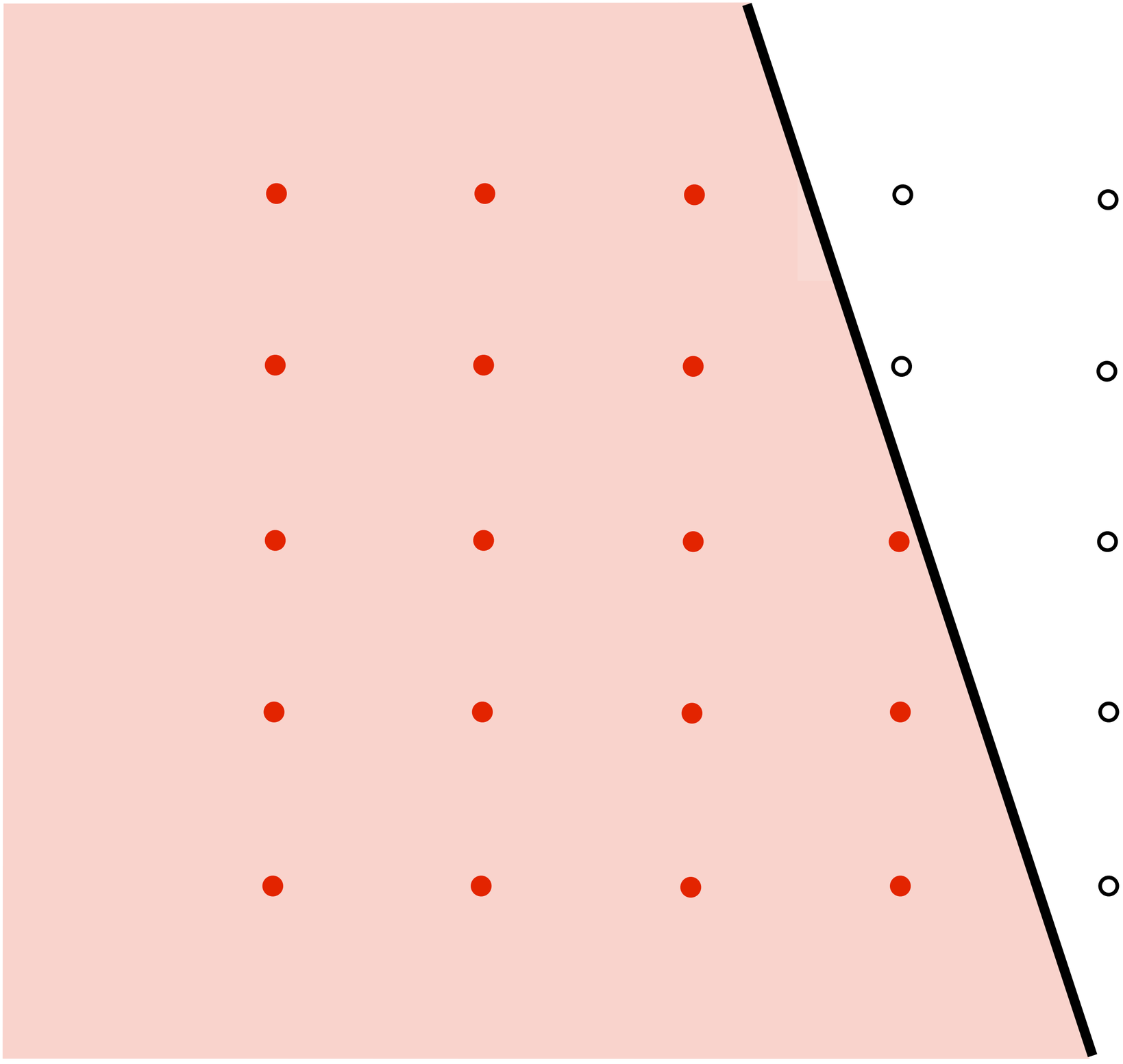
Recall: the real coverage signal was this

Problem: aliasing

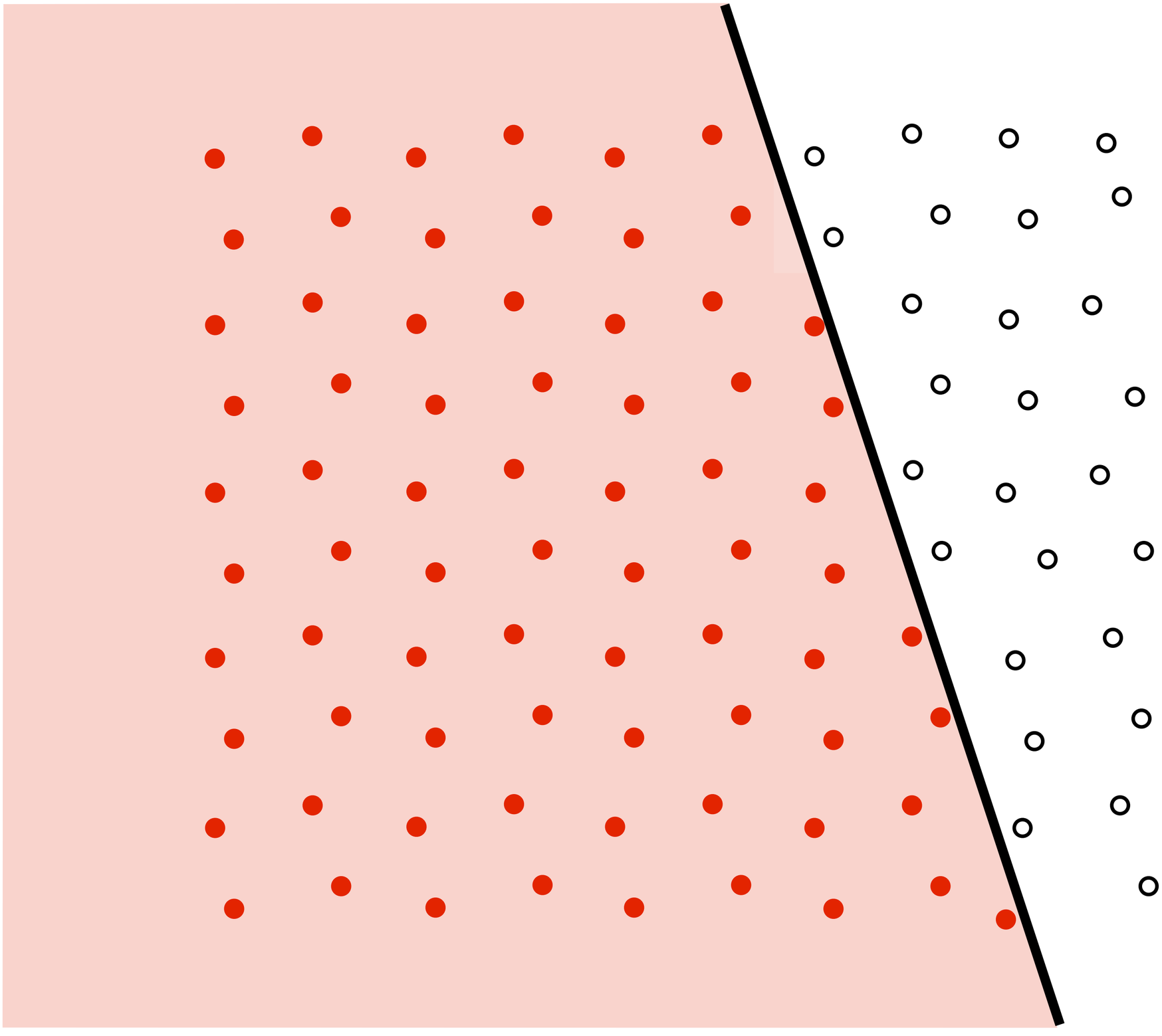
- Undersampling high frequency signal results in aliasing
  - "Jaggies" in a single image
  - "Roping" or "shimmering" in an animation
  - Wagon wheel effect (temporal aliasing)

- High frequencies exist in coverage signal because of triangle edges (sharp edges == high frequencies)

Initial coverage sampling rate (1 sample per pixel)

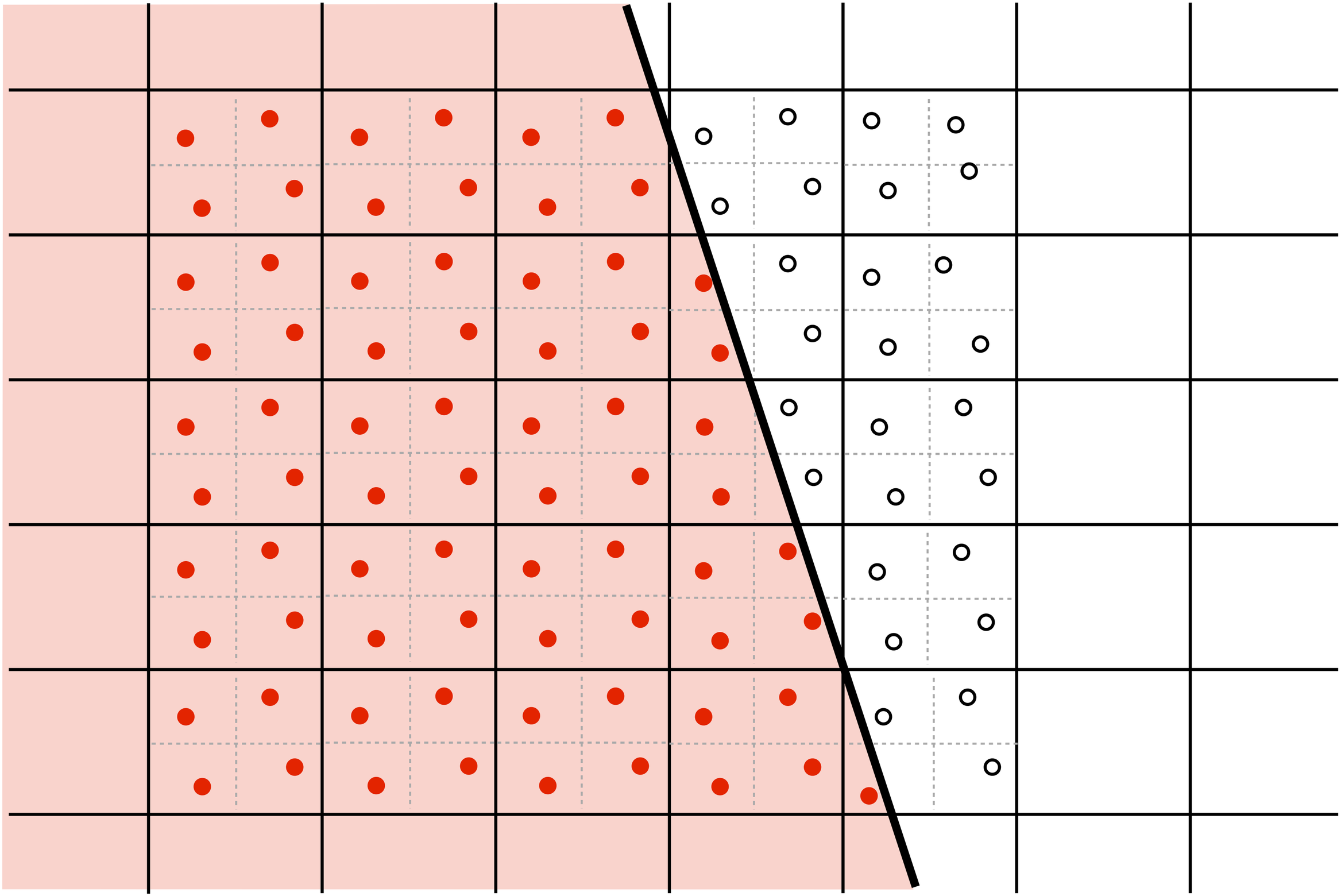


Increase density of sampling coverage signal



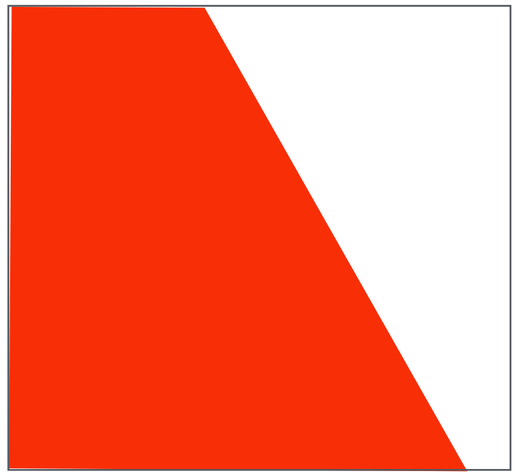
Supersampling

Example: stratified sampling using four samples per pixel

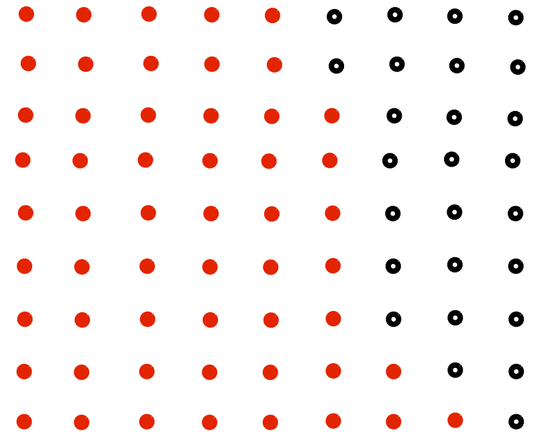


Resampling

Converting from one discrete sampled representation to another

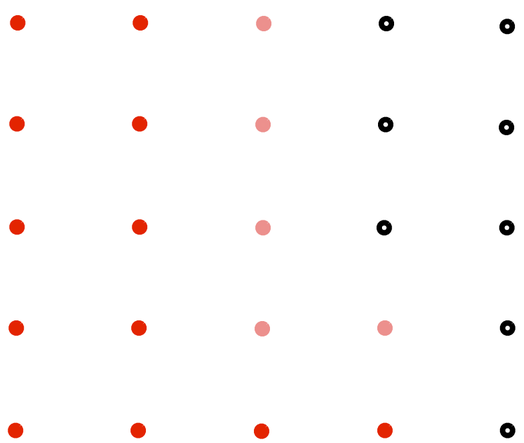


Original signal
(high frequency edge)



Dense sampling of
reconstructed signal

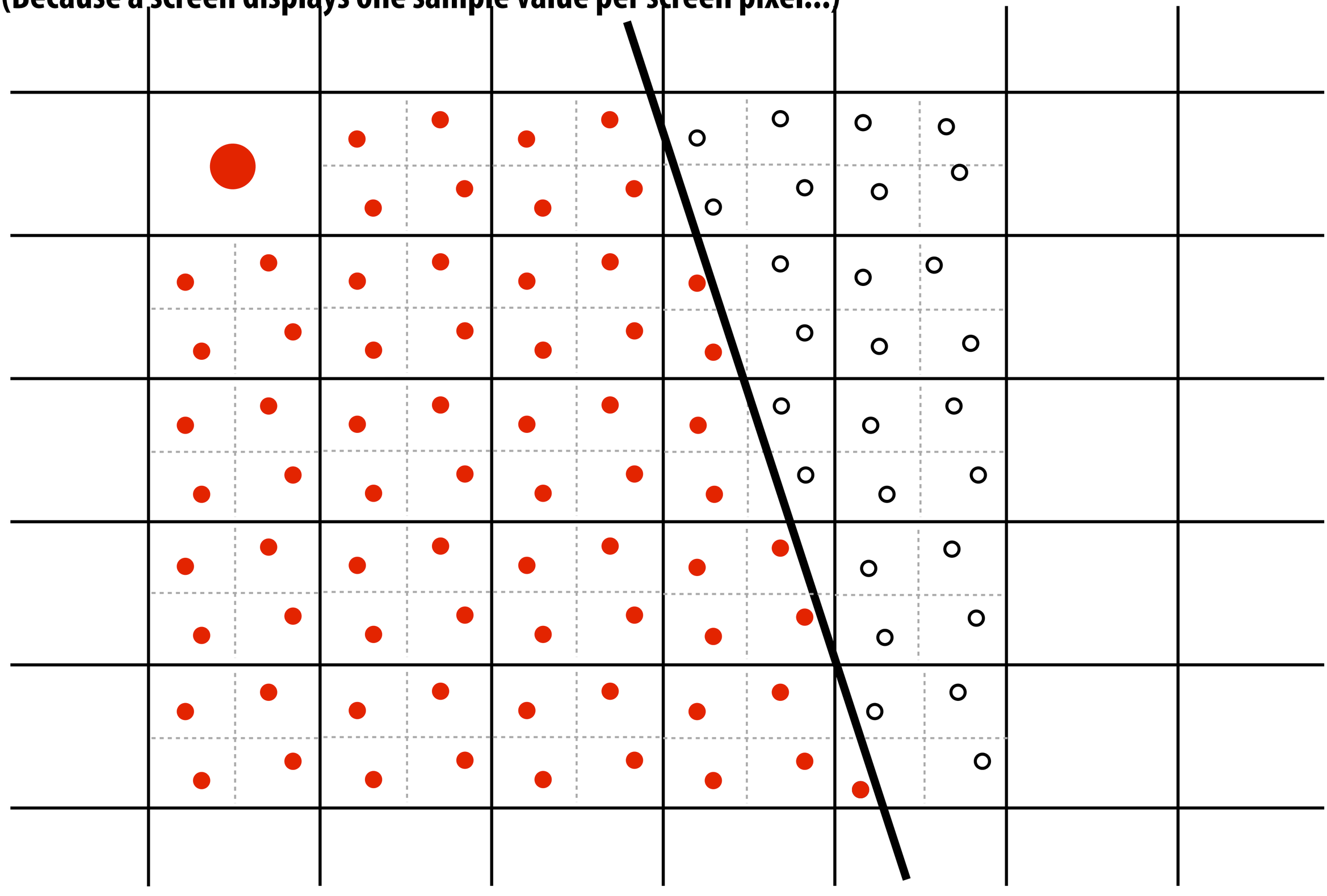
Reconstructed signal
(lacks high frequencies)



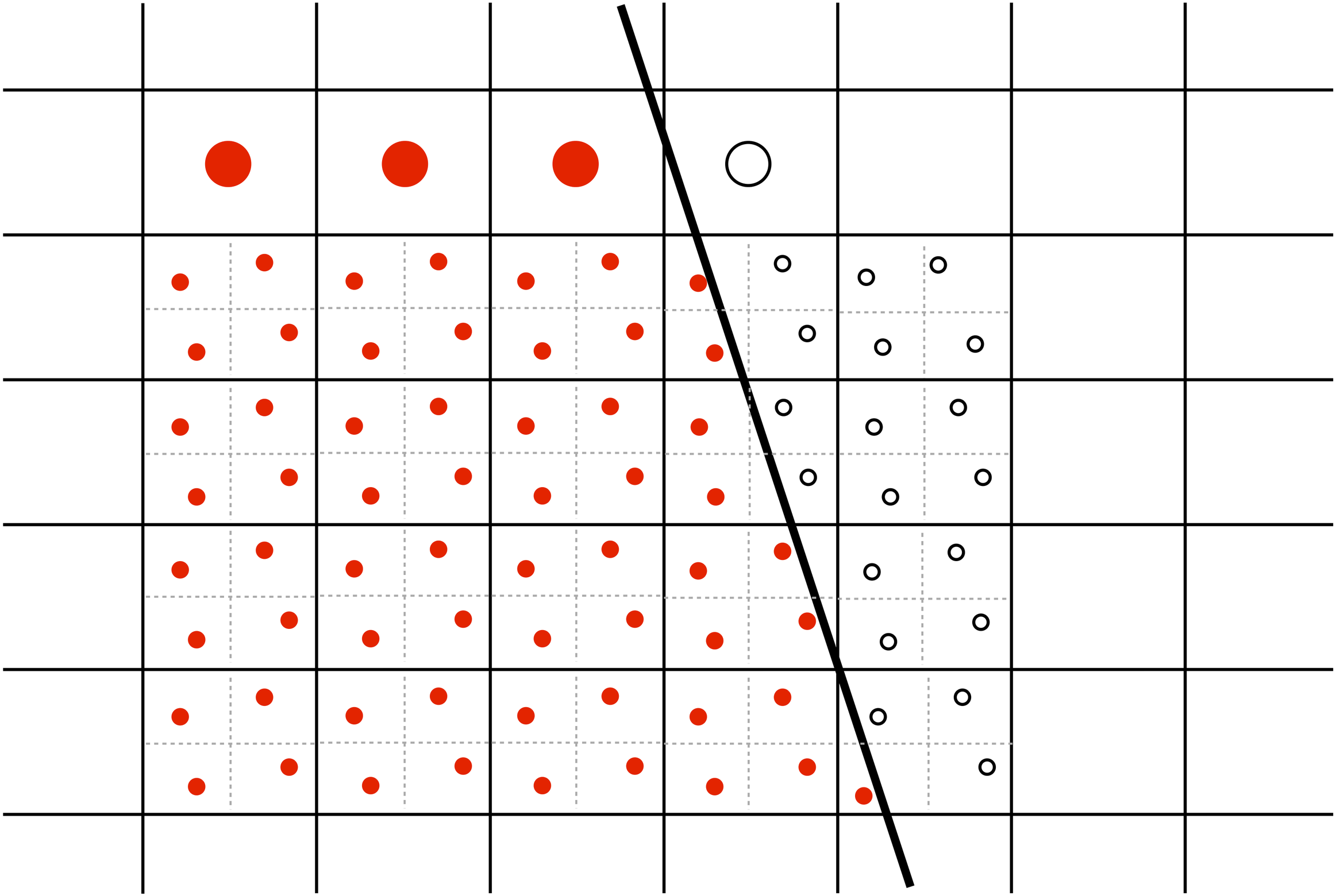
Coarsely sampled signal

Resample to display's pixel resolution

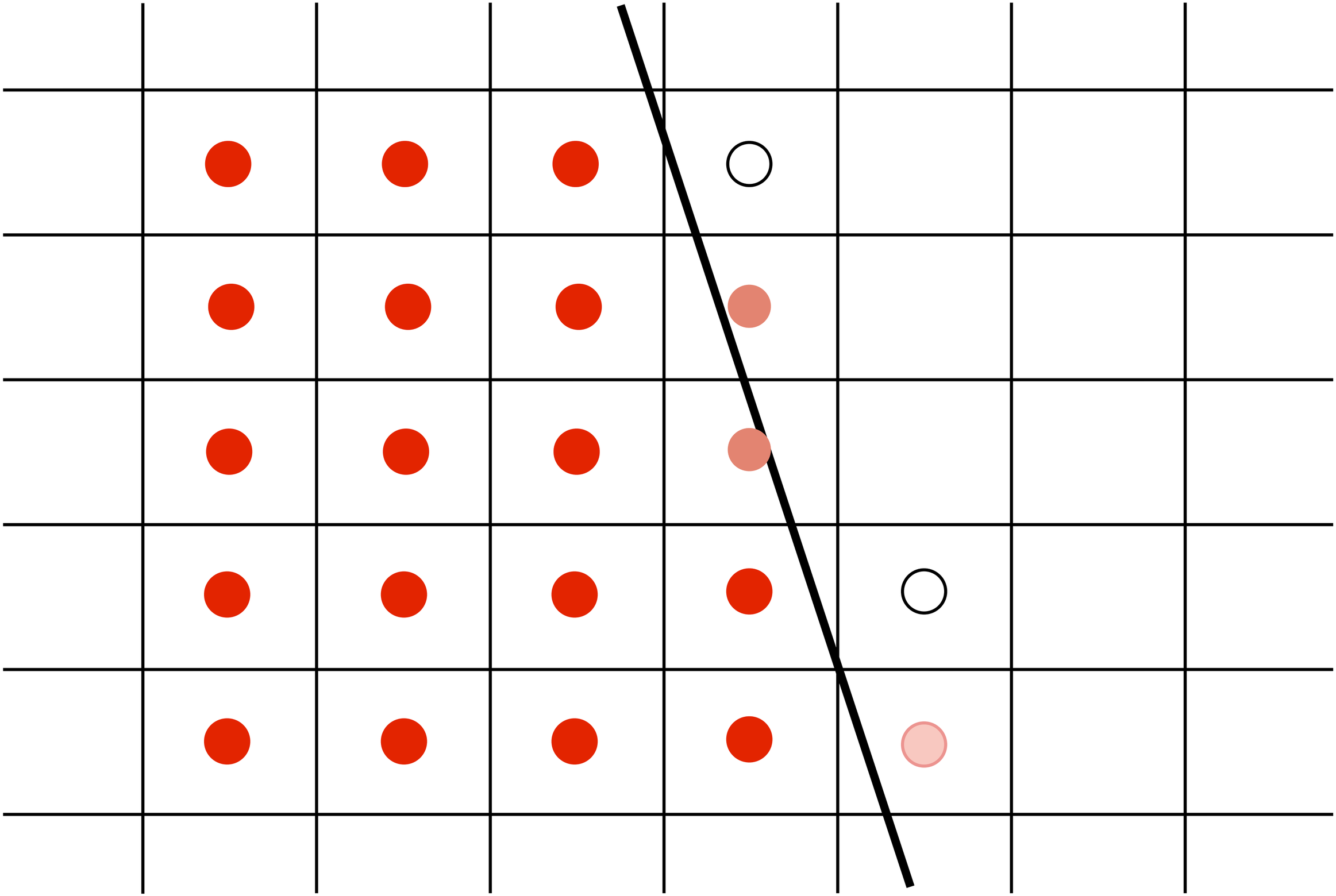
(Because a screen displays one sample value per screen pixel...)



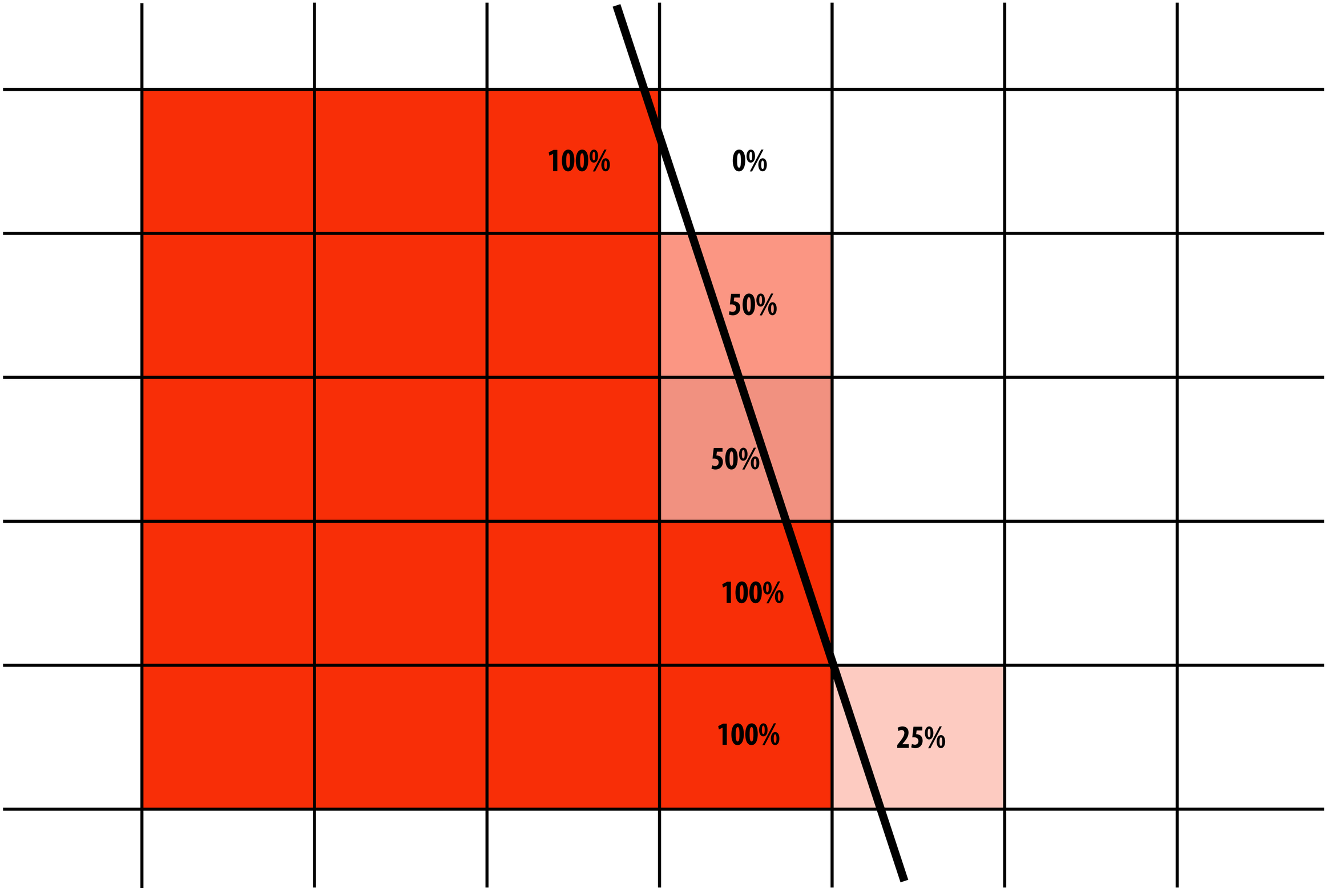
Resample to display's pixel rate (box filter)



Resample to display's pixel rate (box filter)



Displayed result (note anti-aliased edges)



Recall: the real coverage signal was this

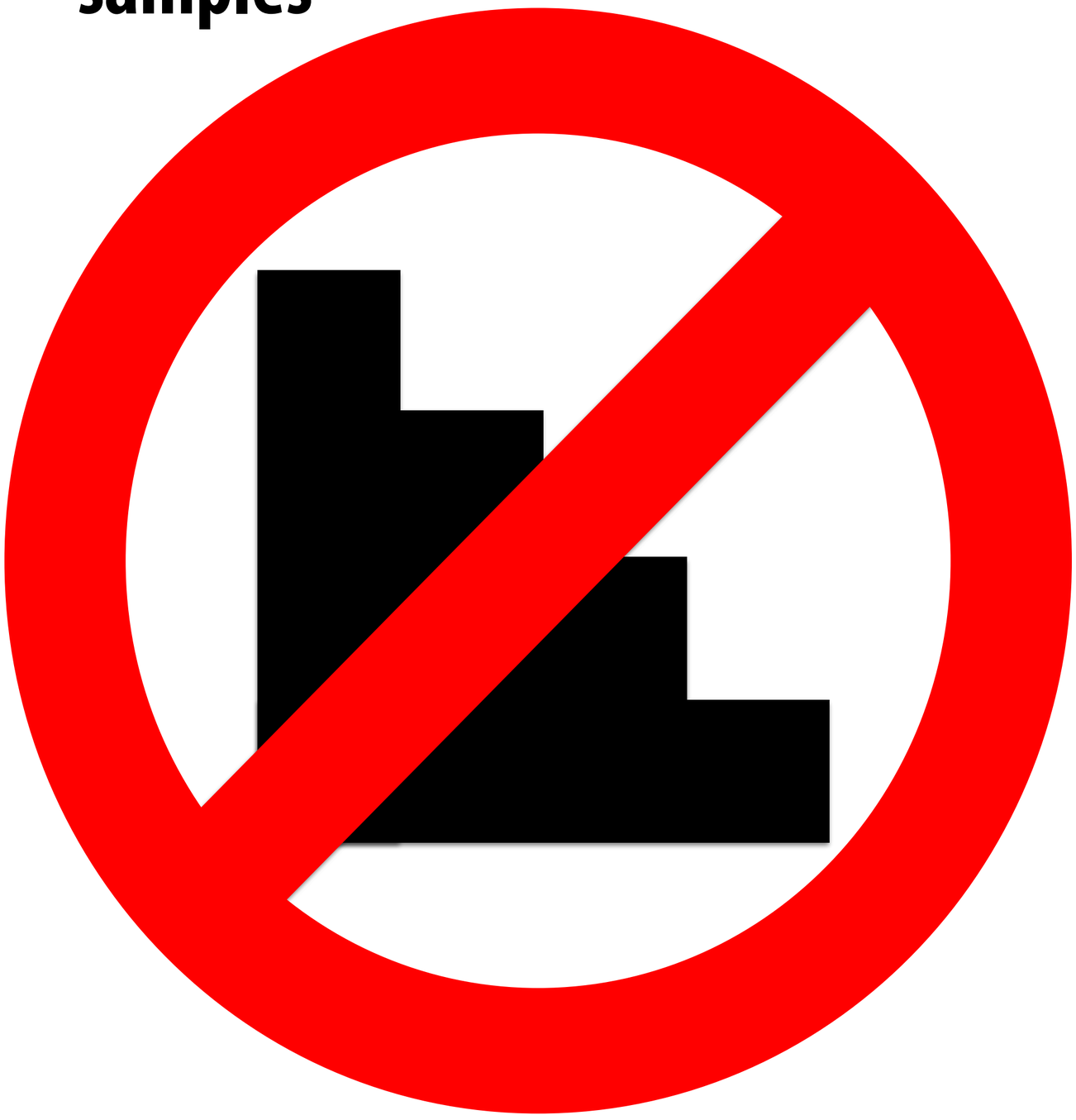
Sampling coverage

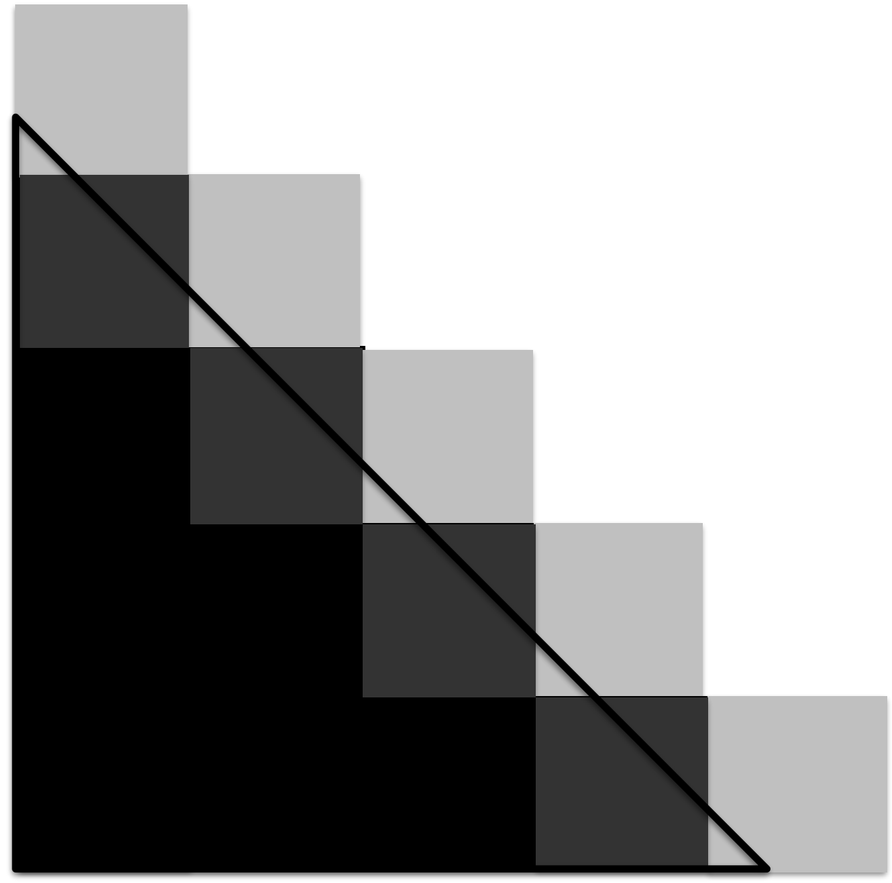
- We want the light emitted from a display to be an accurate to match the ground truth signal: `coverage(x,y)`)

- Resampling a densely sampled signal (supersampled) integrates coverage values over the entire pixel region. The integrated result is sent to the display (and emitted by the pixel) so that the light emitted by the pixel is similar to what would be emitted in that screen region by an "infinite resolution display"

What is “Antialiasing”?

- … better approximation for area integral through more samples



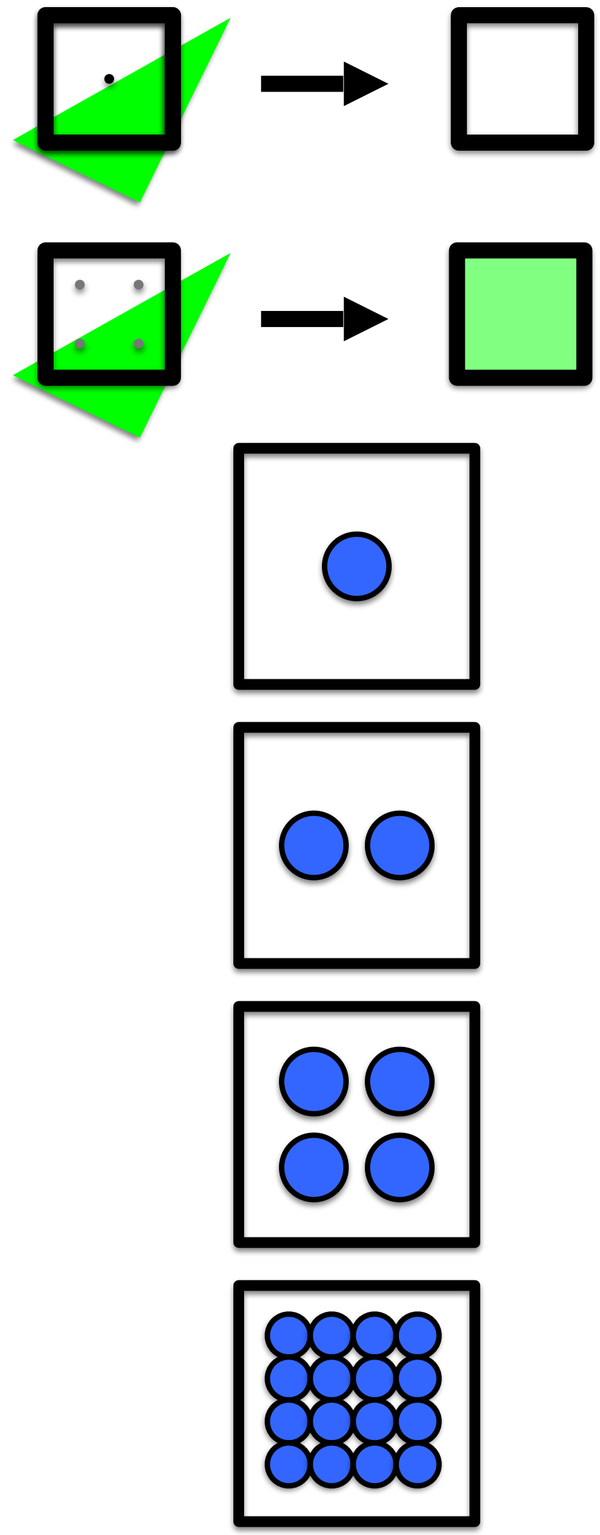


Antialiasing Considerations

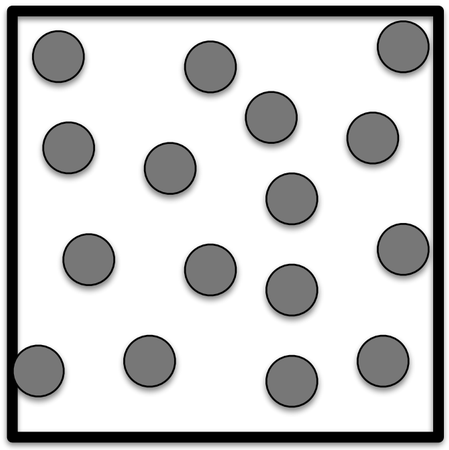
- Additional hardware cost
- Impact on programmer
- Ability to control sample locations
- Behavior with horizontal/vertical lines

Supersampling

- Very simple—render at higher resolution, downsample to screen resolution
- $n \times m$ supersampling: sample grid per pixel
- Color buffer must be $n \times m \times l$
- Graphics card term: "FSAA" (full-screen antialiasing)



Computing pixel color from samples

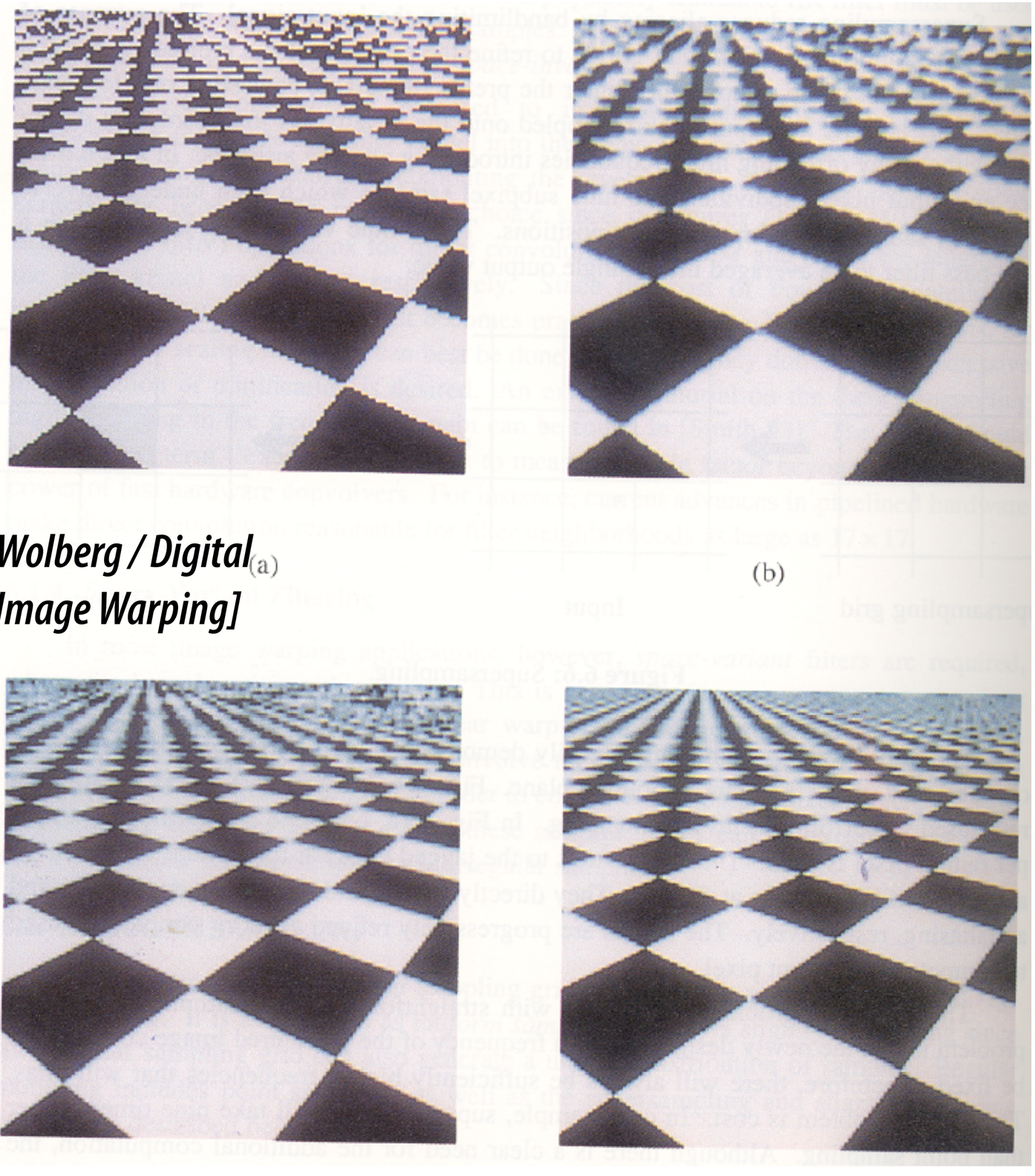


$$\mathbf{p}(x, y) = \sum_{i=1}^{n} w_i \mathbf{c}(i, x, y)$$

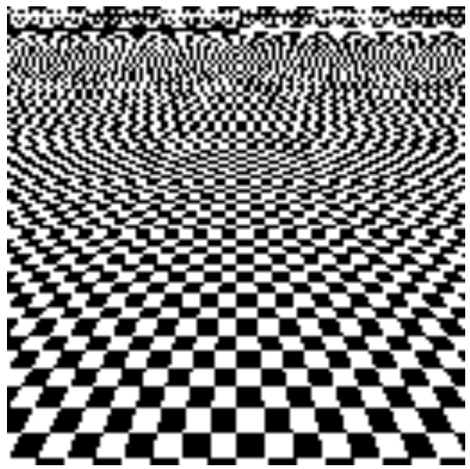
- w_i are the weights in [0,1]
  - Depends on the filter you use!
- $c(i,x,y)$ is the color of sample i inside pixel
- $p(x,y)$ is the color of the pixel
- Desirable: Each sample contributes equal amount to image

[courtesy Tomas Akenine-Möller]

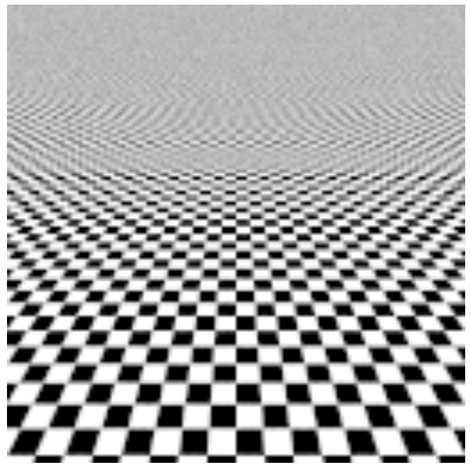
Supersampling Examples



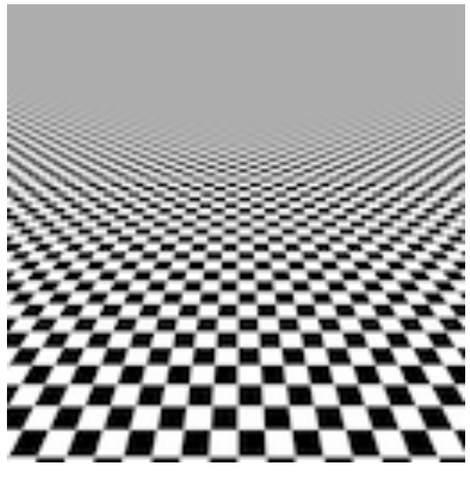
[Wolberg / Digital Image Warping]



No filter



Non-ideal filter



Sinc

http://en.wikipedia.org/wiki/Spatial_anti-aliasing

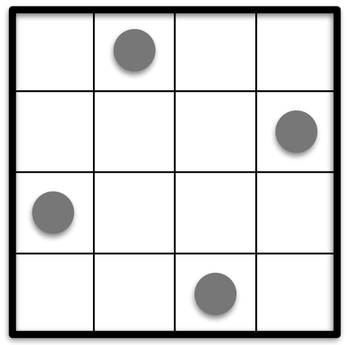
http://www.anandtech.com/show/592/5

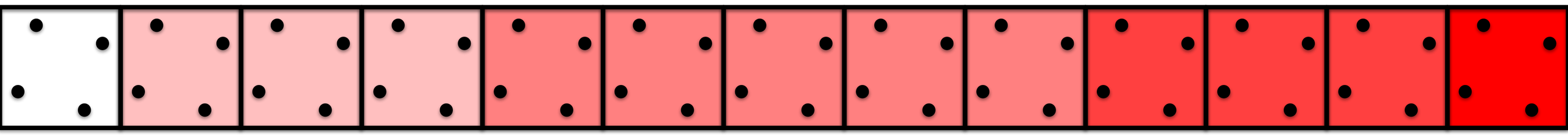
Multisample Antialiasing ("Multisampling")

- Much more common than FSAA
- Specify the location of multiple sample points per pixel
  - Patterns may differ spatially, but not temporally
- Rasterize fragments that include
  - A bitmask of occluded samples
  - Appropriate color, depth, and texture coords
- Evaluate texture once per fragment (not per sample)
  - This is the key difference from FSAA
- Store color and depth for each sample in framebuffer
- Resolve samples to final pixel value either
  - Each time the pixel is modified, or
  - Once, before the buffer is displayed

Standard supersampling schemes

- Rotated Grid Supersampling (RGSS)
  - Good for near horizontal and vertical edges
  - Cost: 4 samples/pixel



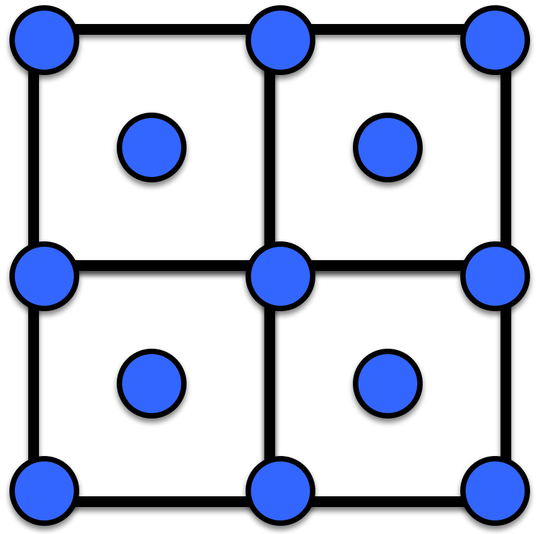


- A study by a researcher called Naiman:
  - Near-horizontal and near-vertical edges are the most annoying to humans
    - Then comes near-45-degree edges…
  - RGSS: is very good for those cases!

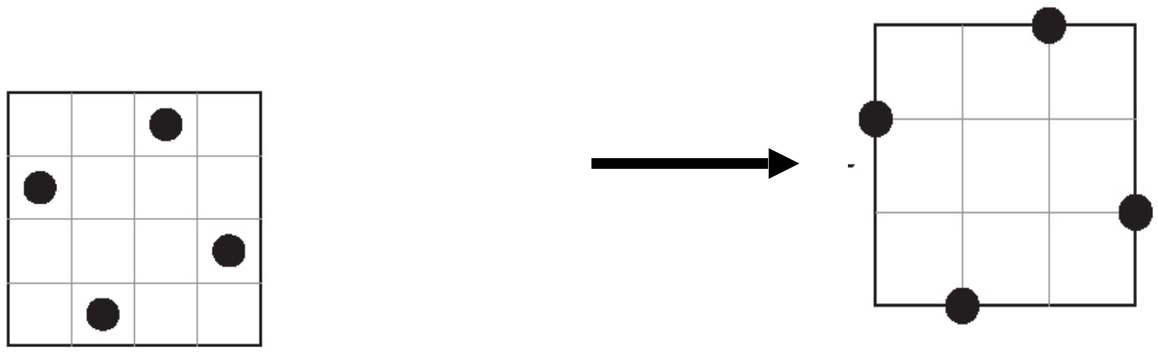
[courtesy Tomas Akenine-Möller]

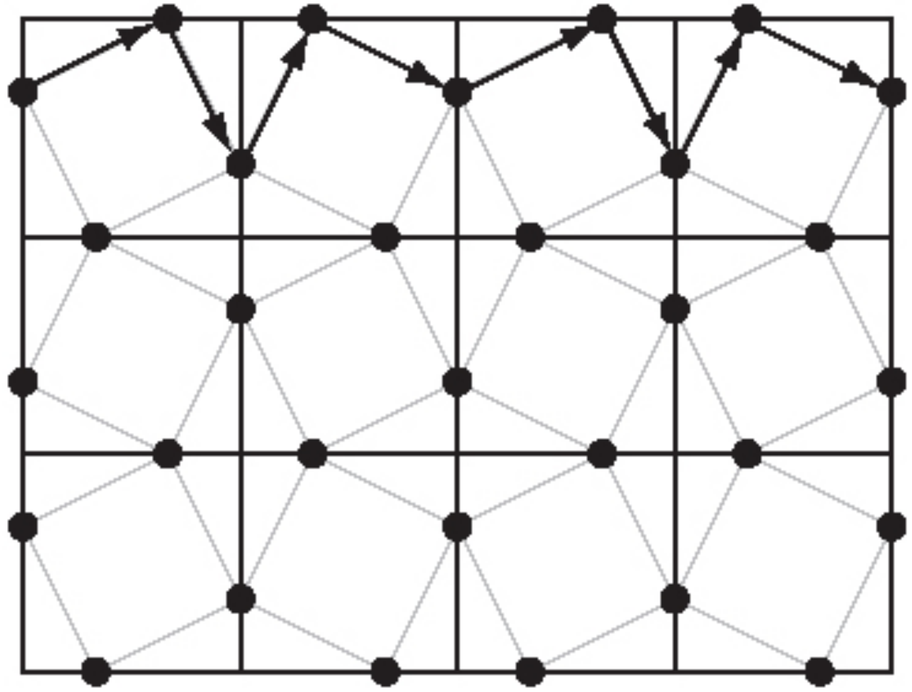
NVIDIA Quincunx

- 5 samples contribute to each pixel
- Only 2 samples per pixel
- Weight of middle sample: 1/2
- Weight of corner samples: 1/8



FLIPQUAD supersampling





Quality: quite near RGSS (costing 4 samples)

[courtesy Tomas Akenine-Möller]

HW2 Results (2007)

- Michael Rush (ATI Mobility 9600, 1x 2x 4x 6x)
- Mike Goldman (ATI x1600, 4x 6x)

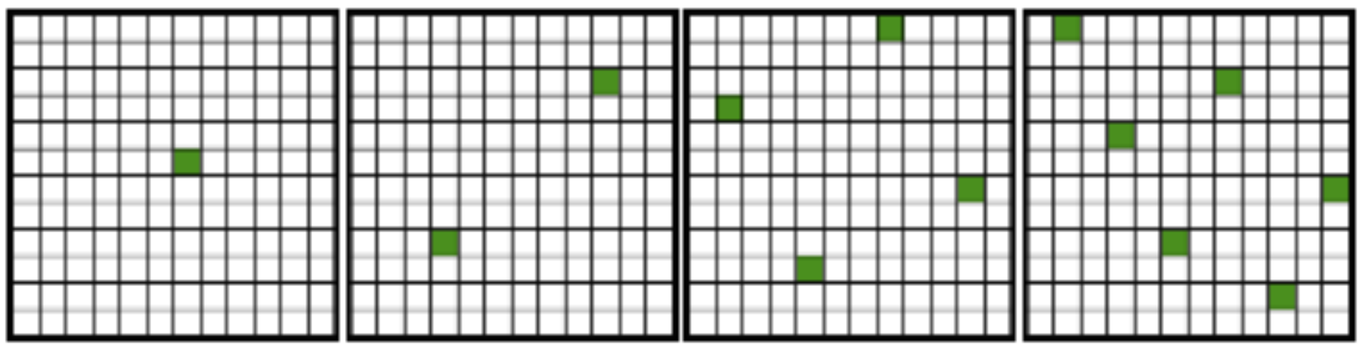
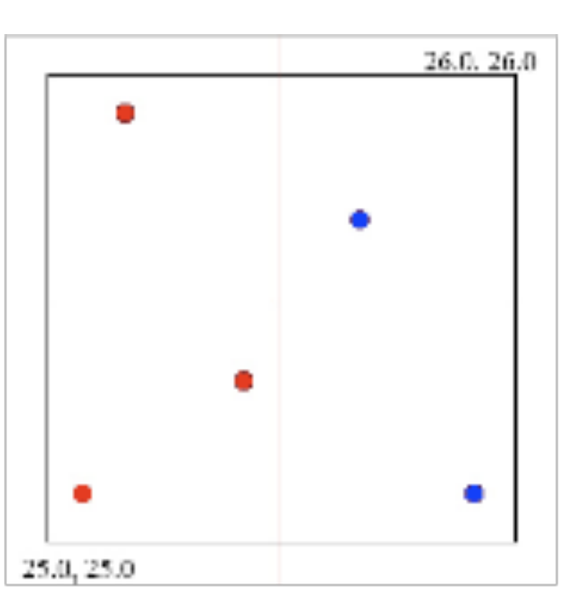
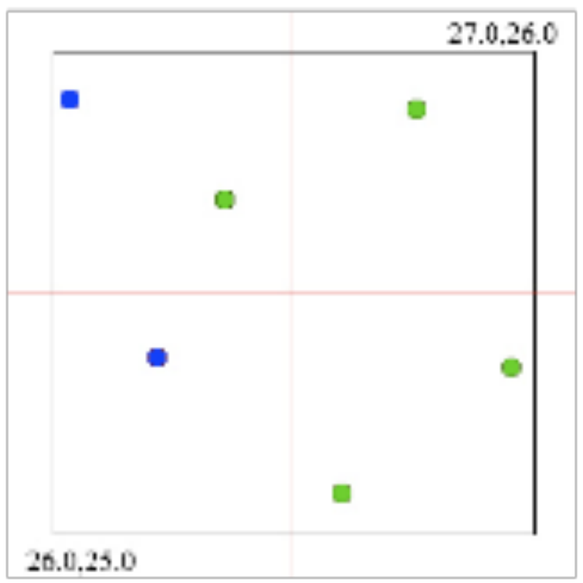
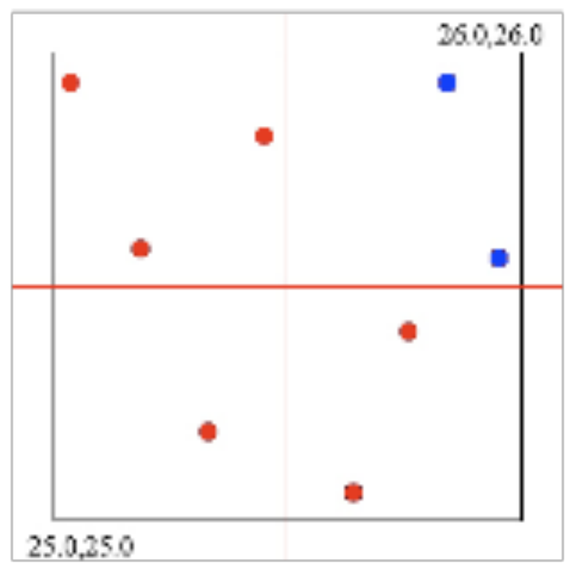


Figure 2. Varying Degrees of Anti-Aliasing (no AA, 2x, 4x, 6x)

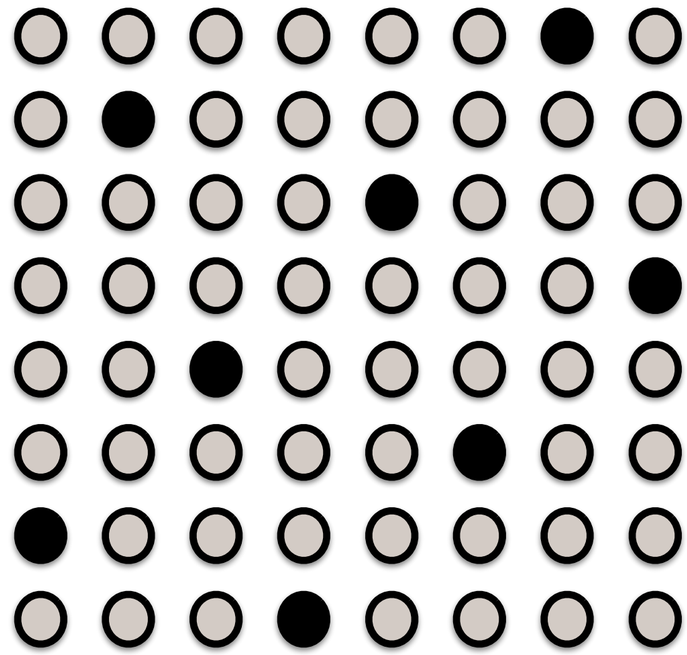






Multisample Sample Pattern

- Trade-off
  - Pseudo random
    - Better, more efficient filtering
  - Regular
    - Easier, more efficient rasterization
- Compromise pattern is regular subset
  - Manageable sample count
  - Empirically best
  - Pixar owned patent on this



A-buffer and Relatives

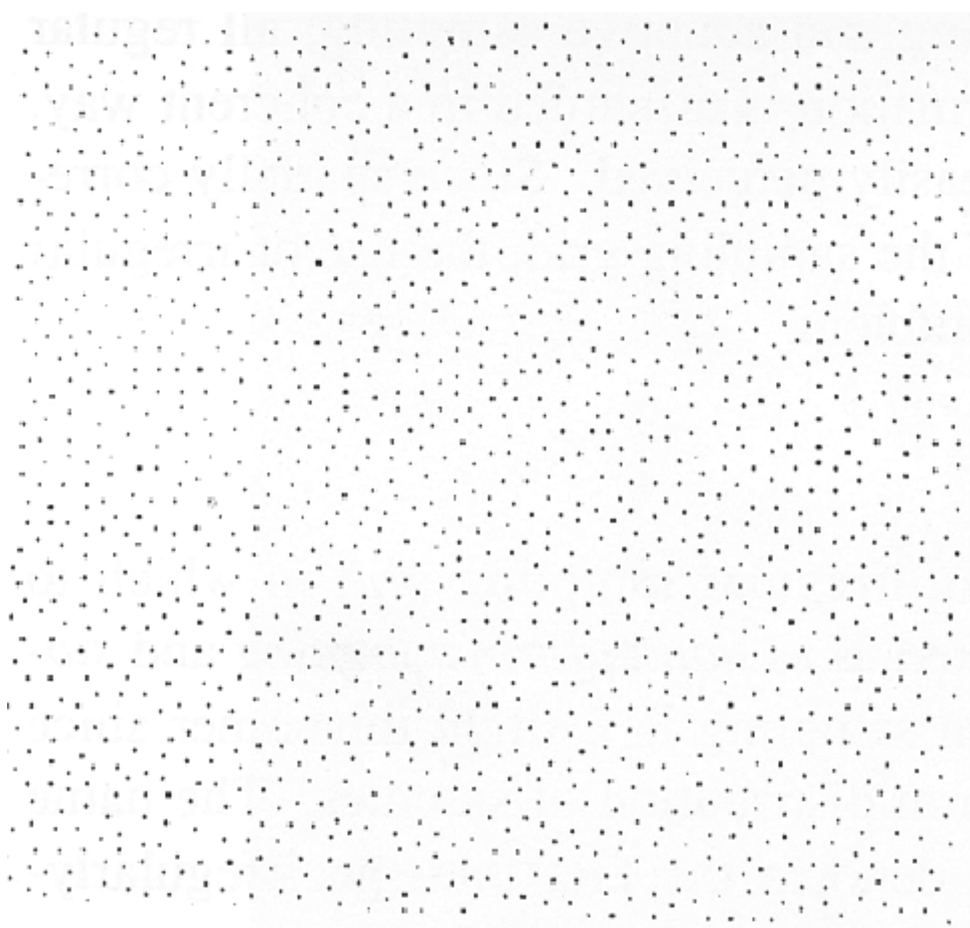
- "The A-Buffer, an Antialiased Hidden Surface Method", Loren Carpenter, Siggraph '84
- Focus: edge antialiasing, transparency
- "Multisampling" = more than one sample per pixel in single pass
  - Per-pixel: depth, variable-length list of "fragments"
  - Per-"fragment": coverage mask, color, max depth, min depth, transparency
- Used in RenderMan
- Not in current hardware (unbounded storage requirement)
  - Holy grail: order-independent transparency with bounded hardware

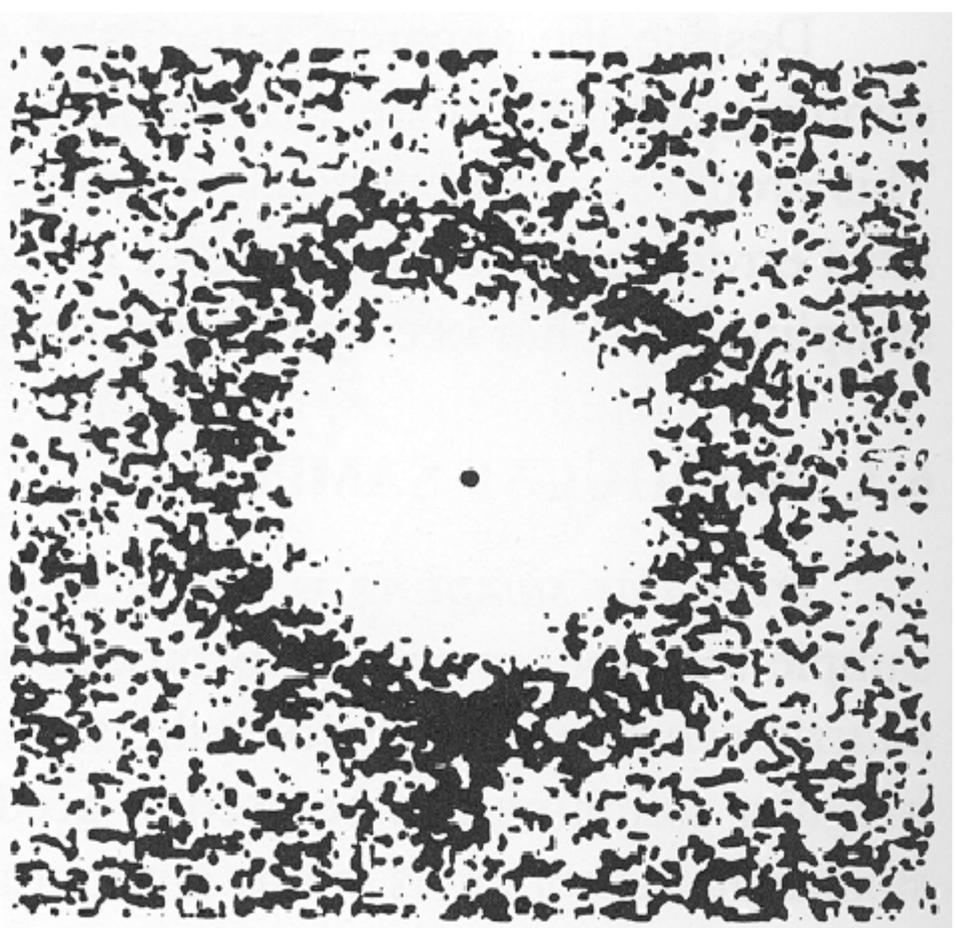
Sampling Theory

- Real-world is continuous, but pixels are discrete
  - Sampling is process of continuous → discrete
- Sampling grid imposes upper limit on frequencies that can be sampled
  - Upper limit: "Nyquist limit" (one cycle/2 pixels)
    - Supersampling raises the Nyquist limit
    - Reconstruction of continuous signal with Sampling Theorem cannot represent any frequencies above the Nyquist Limit
    - Any frequency components in original signal above the Nyquist limit will appear (alias) as lower frequency components
    - Aliasing is a highly objectionable artifact
    - Goal: Avoid aliasing. Avoid high-frequency signal appearing as low-frequency signal.

Stochastic Sampling: Poisson

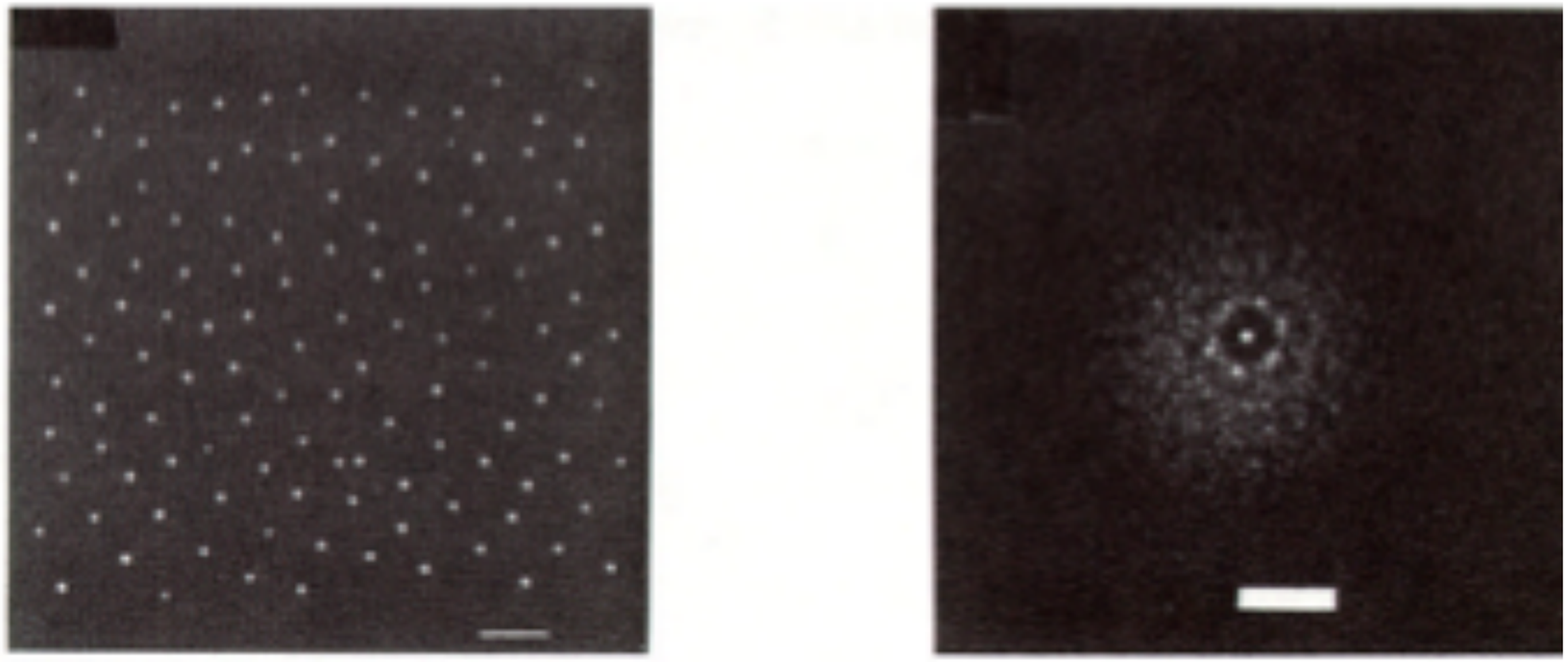
- "Stochastic Sampling in Computer Graphics", Rob Cook, TOG January '86
- Ideal sampling pattern: "Broad noisy spectrum with minimal low-frequency energy" [Wolberg]
- Stochastic: Irregular grid
- Poisson sampling (blue noise)—aliasing or noise?



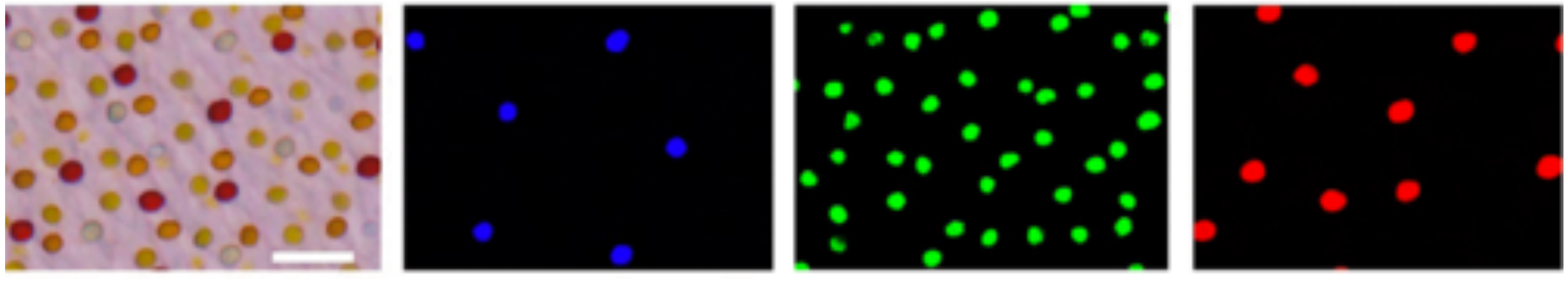


[Wolberg / Digital Image Warping]

Stochastic Sampling: Real Life



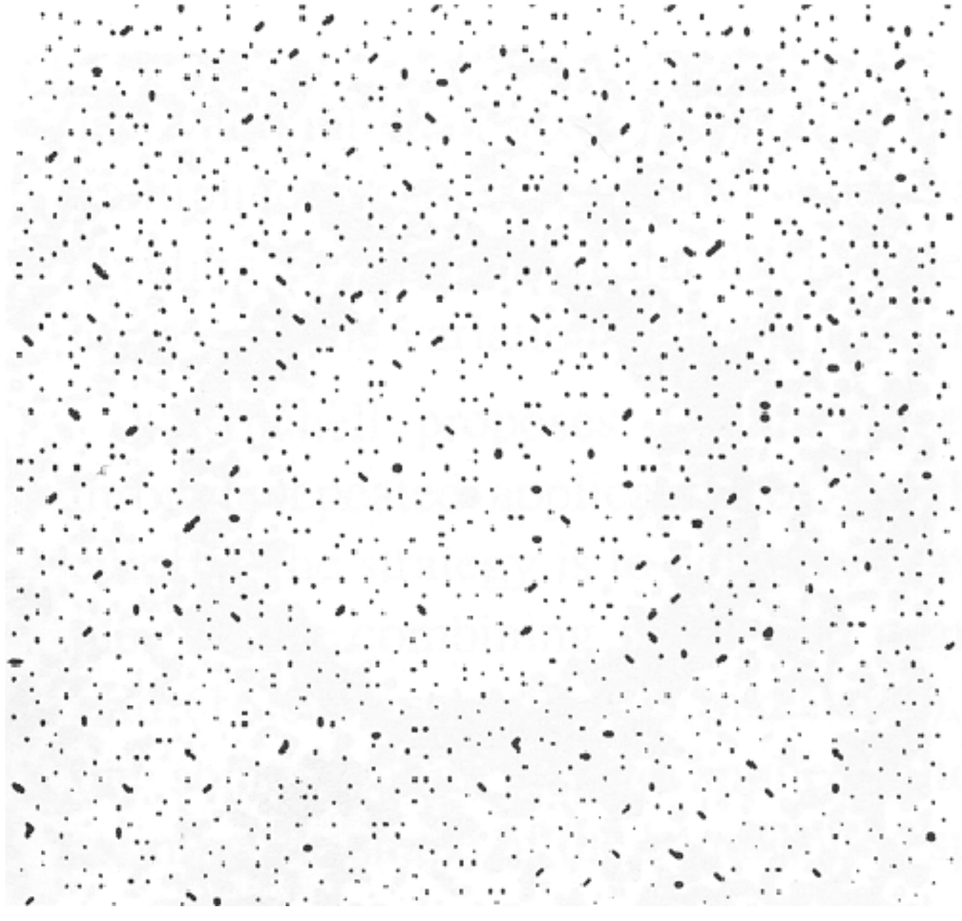
[Cook / Stochastic Sampling]

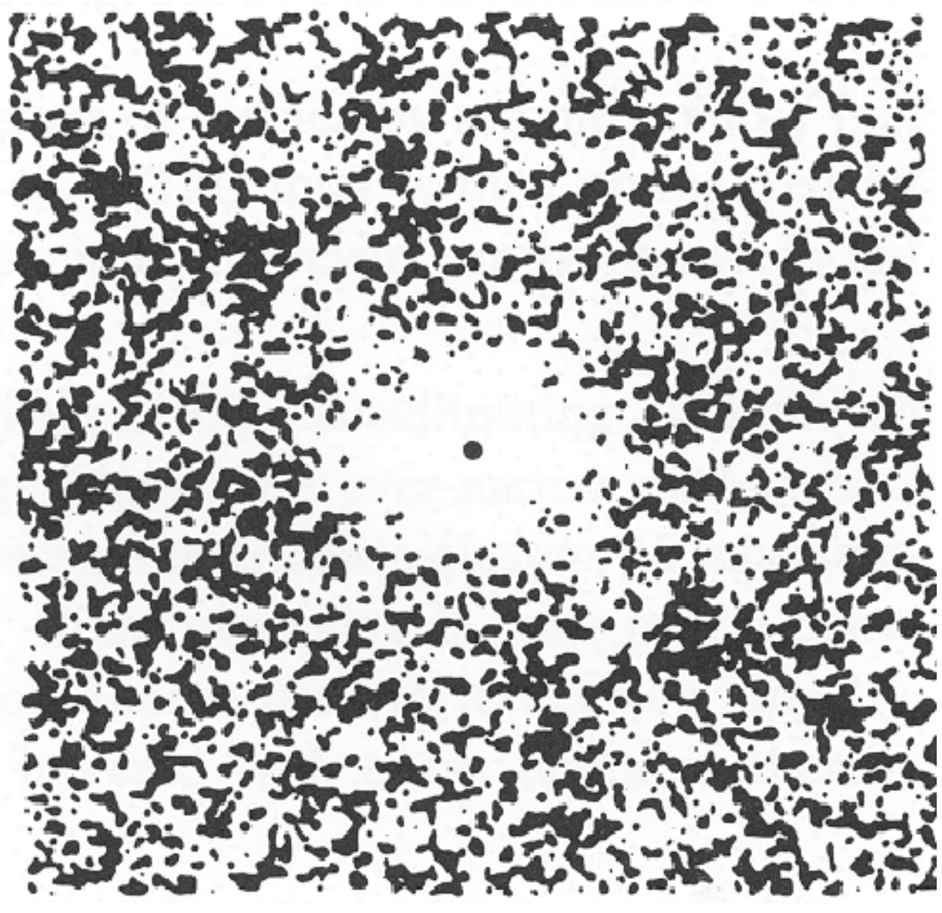


"Spatial arrangements of chicken cone photoreceptors". Avian photoreceptor patterns represent a disordered hyperuniform solution to a multiscale packing problem. Yang Jiao et al., Phys. Rev. E 89, 022721 – Published 24 February 2014.

Stochastic Sampling: Jittering

- Jittering (Stratified Sampling): random locations within subpixel areas
- ATI Smoothvision: 16 patterns over 4x4 tile (Keller/Heidrich, Interleaved Sampling, EGRW '01)





[Wolberg / Digital Image Warping]

Outline

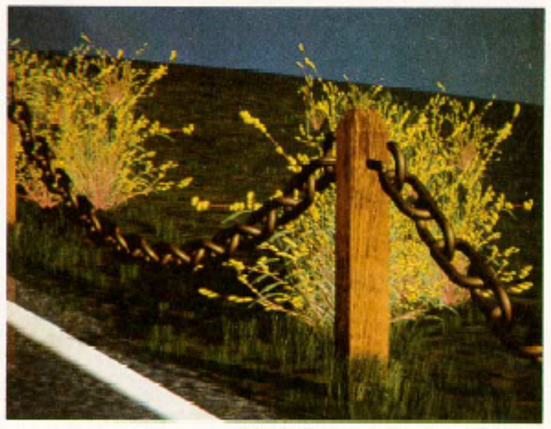
- Antialiasing
- Compositing
- Depth Buffer
- Monitors

Compositing

Foreground = *FrgdGrass* **over** *Rock* **over** *Fence* **over** *Shadow* **over** *BkgdGrass*;

Hillside = *Plant* **over** *GlossyRoad* **over** *Hill*;

Background = *Rainbow* **plus** *Darkbow* **over** *Mountains* **over** *Sky*;



Pt.Reyes = *Foreground* **over** *Hillside* **over** *Background*.

[Lucasfilm, "Road to Point Reyes"]

What's this "Alpha"?

- Transparency
- How transparent is the object?
  - 1.0 = opaque; 0.0 = transparent
- Pixel coverage
  - How much of the pixel is covered by this object?

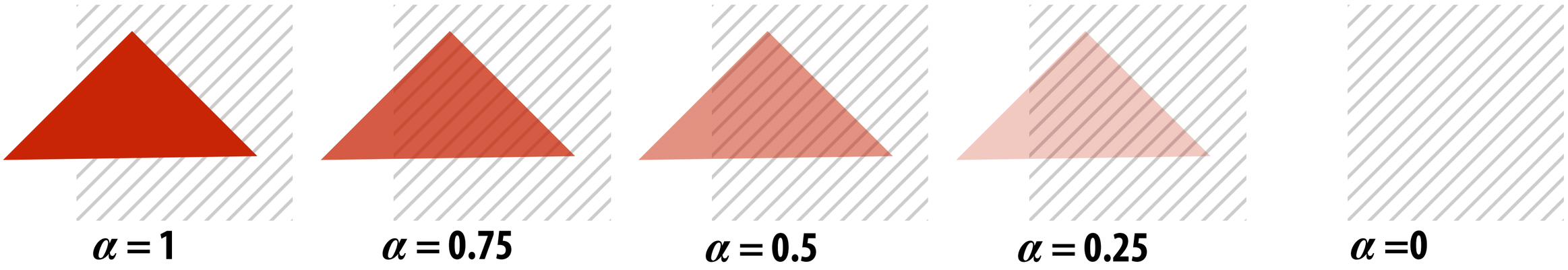
- Transparency extensible to subsamples, but not pixel coverage

- Jim Blinn: Top 10 problem

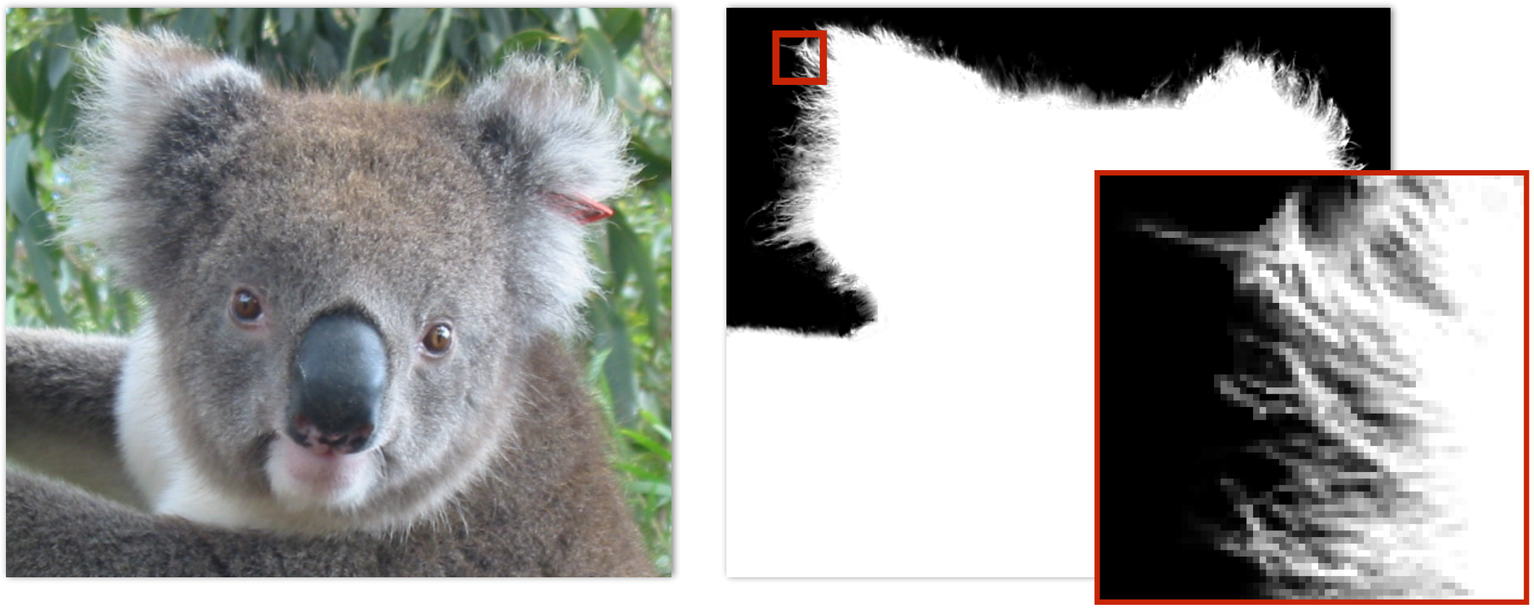
Representing opacity as alpha

- Alpha describes the opacity of an object
- Fully opaque surface: $\alpha = 1$
- 50% transparent surface: $\alpha = 0.5$
- Fully transparent surface: $\alpha = 0$

Red triangle with decreasing opacity



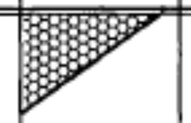
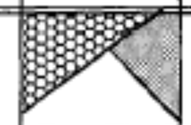
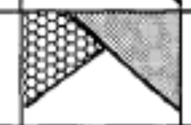
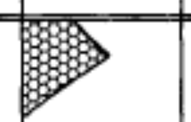
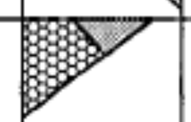
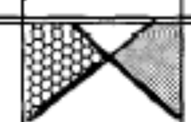
Alpha: additional channel of image (rgba)



α of foreground object

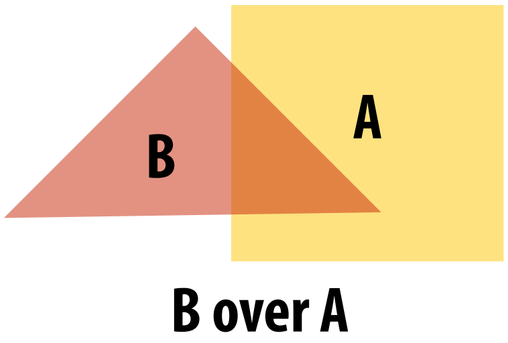
Porter-Duff Compositing

- "Compositing Digital Images", Tom Porter/Tom Duff, Siggraph '84
  - Algebra for image composition
- Most important: "over"
  - $c_0 = \alpha_s c_s + (1-\alpha_s) c_d$
  - s: source (incoming fragment)
  - d: destination (pixel color)

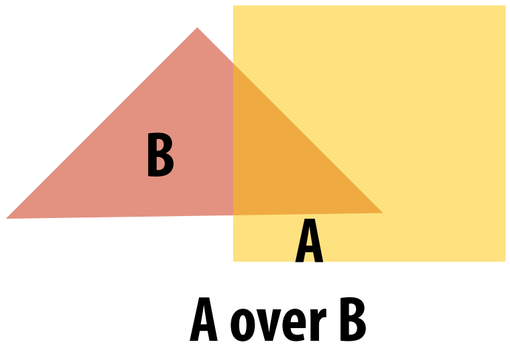
operation	quadruple	diagram	F_A	F_B
clear	(0,0,0,0)		0	0
A	(0,A,0,A)		1	0
B	(0,0,B,B)		0	1
A **over** B	(0,A,B,A)		1	$1-\alpha_A$
B **over** A	(0,A,B,B)		$1-\alpha_B$	1
A **in** B	(0,0,0,A)		α_B	0
B **in** A	(0,0,0,B)		0	α_A
A **out** B	(0,A,0,0)		$1-\alpha_B$	0
B **out** A	(0,0,B,0)		0	$1-\alpha_A$
A **atop** B	(0,0,B,A)		α_B	$1-\alpha_A$
B **atop** A	(0,A,0,B)		$1-\alpha_B$	α_A
A **xor** B	(0,A,B,0)		$1-\alpha_B$	$1-\alpha_A$

Over operator:

Composite image B with opacity α_B <u>over</u> image A with opacity α_A



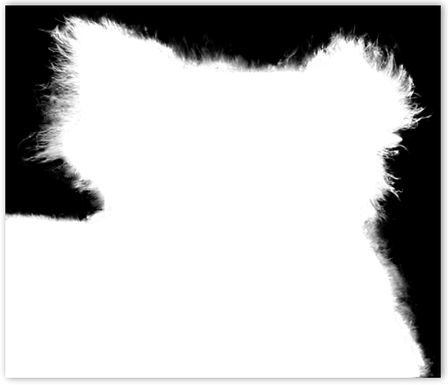
B over A



A over B

A over B != B over A

"Over" is not commutative



Koala over NYC

Color buffer update: semi-transparent surfaces

Color buffer values and tri_color are represented with premultiplied alpha

```
over(c1, c2) {
   return c1 + (1-c1.a) * c2;
}

update_color_buffer(tri_d, tri_color, x, y) {

   if (pass_depth_test(tri_d, zbuffer[x][y]) {
       // update color buffer
       // Note: no depth buffer update
       color[x][y] = over(tri_color, color[x][y]);
  }
}
```

What is the assumption made by this implementation?

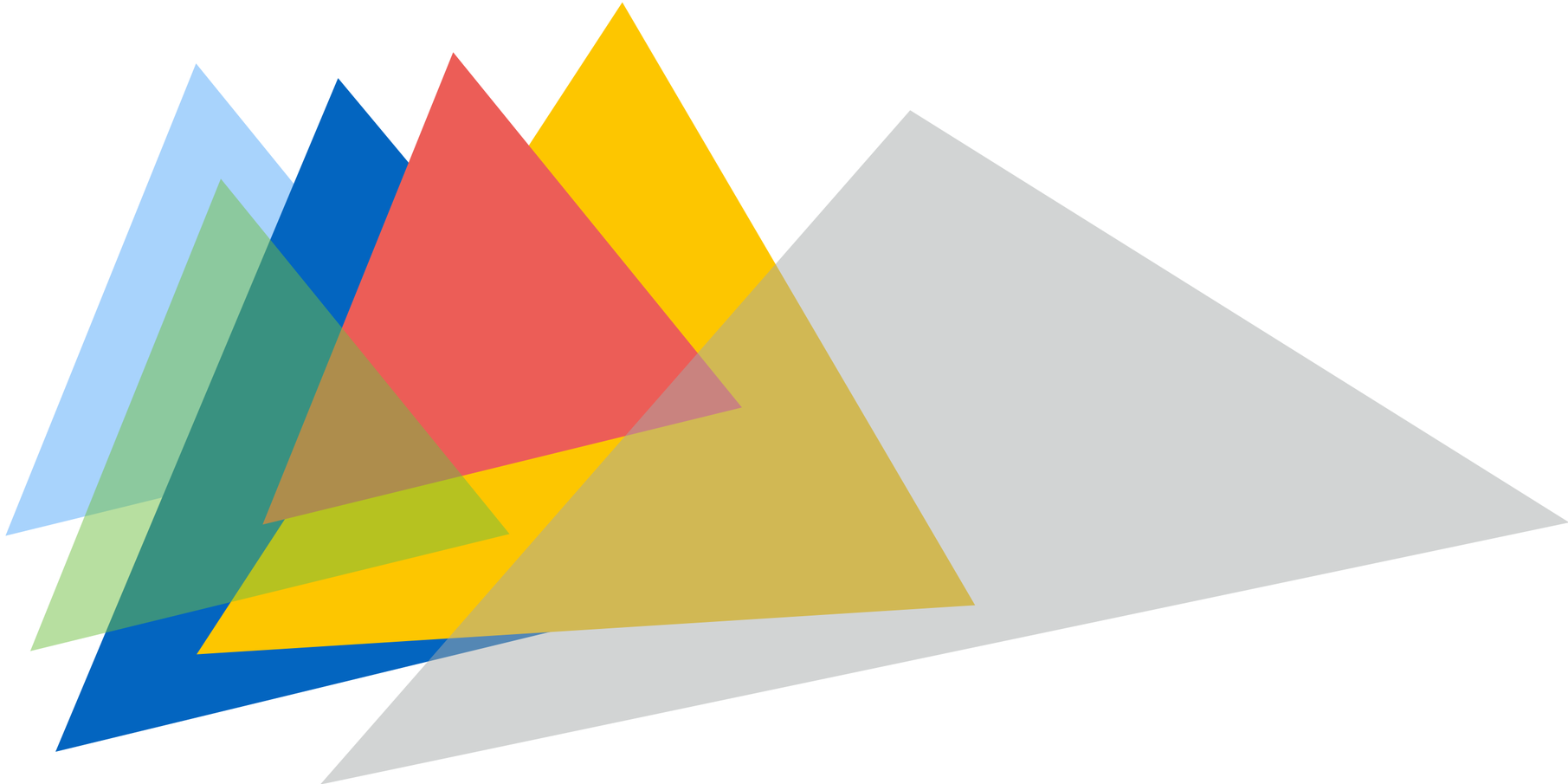
Triangles must be rendered in back to front order!

What if triangles are rendered in front to back order?

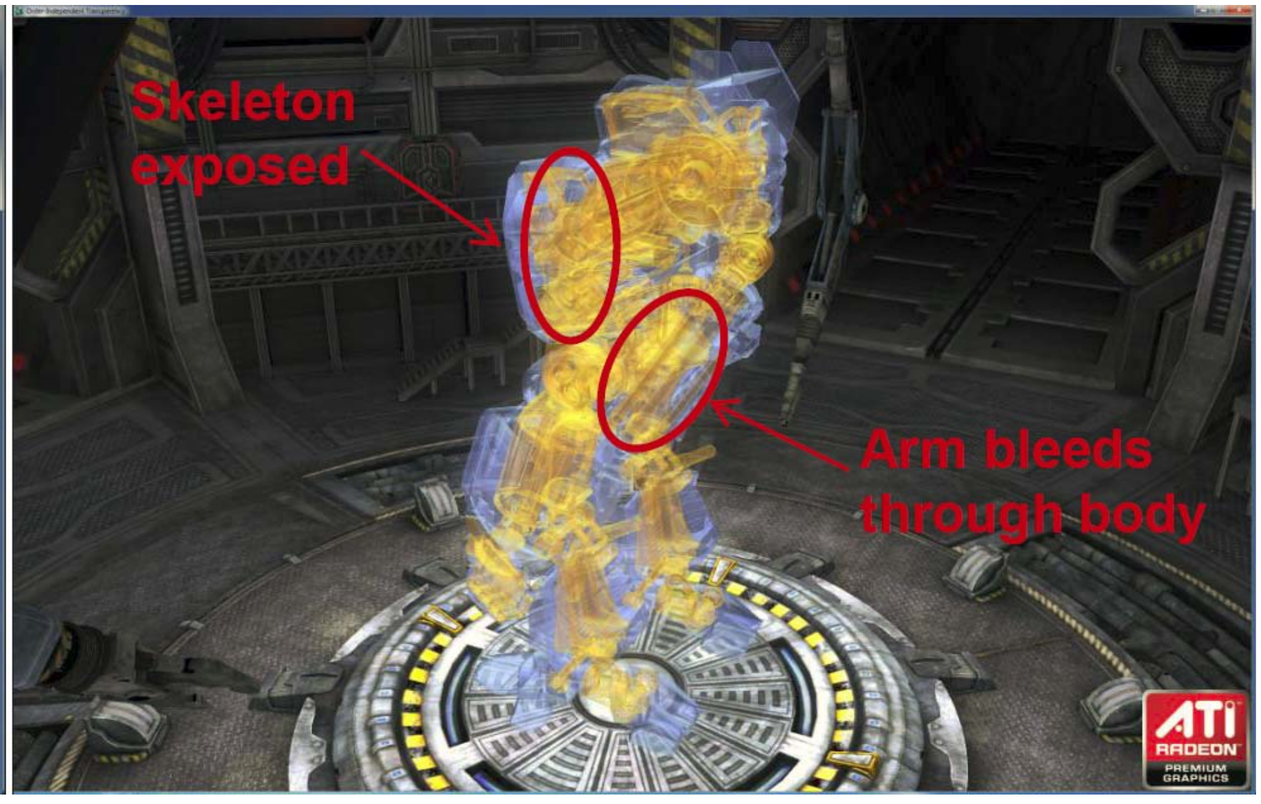
Modify code: `over(color[x][y], tri_color)`

Putting it all together

- Consider rendering a mixture of opaque and transparent triangles
- Step 1: render opaque surfaces using depth-buffered occlusion
  - If pass depth test, triangle overwrites value in color buffer at sample
- Step 2: disable depth buffer update, render semi-transparent surfaces in back-to-front order.
  - If pass depth test, triangle is composited OVER contents of color buffer at sample



Recall Decision: Ordering



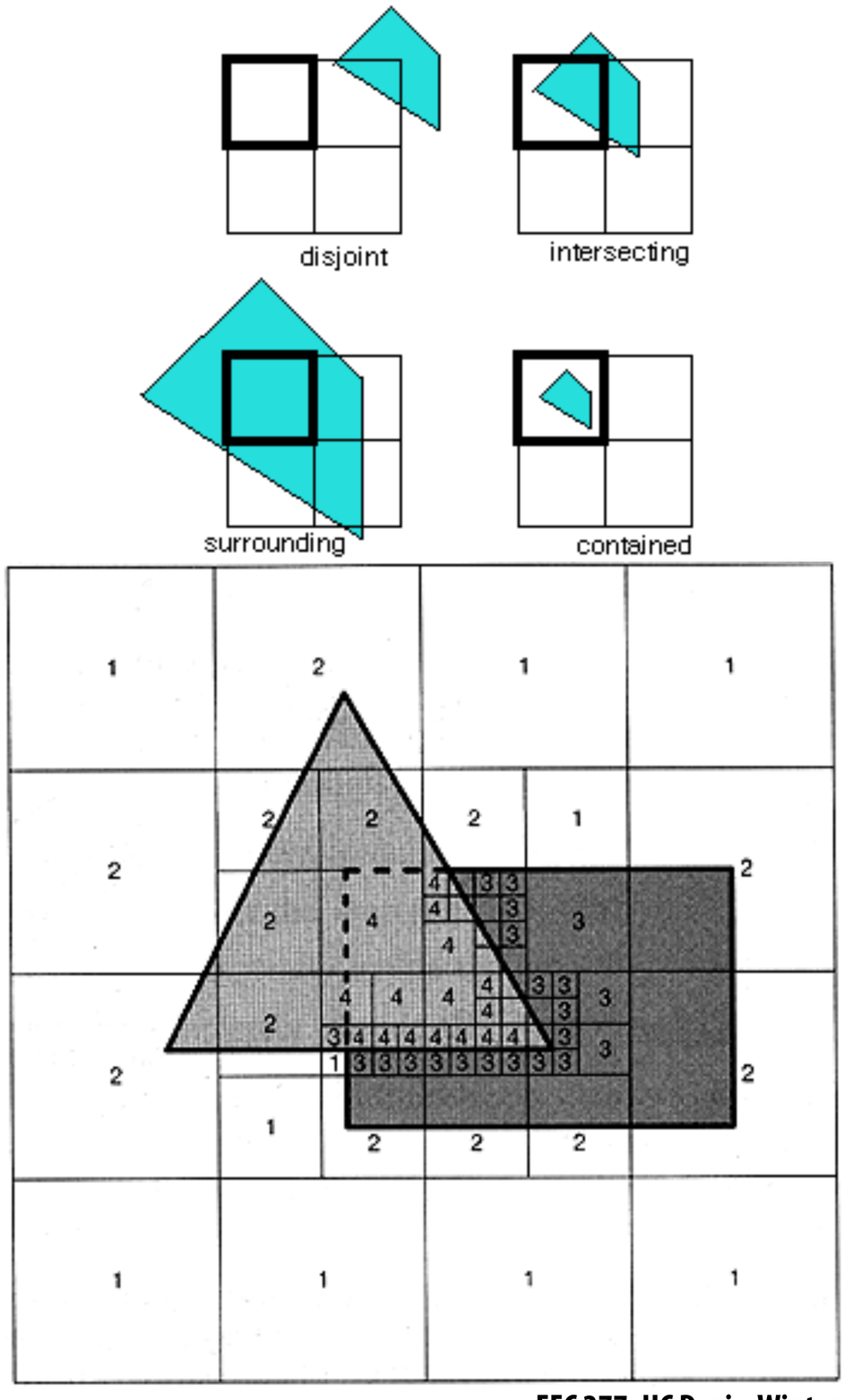
[Courtesy of AMD "Mecha" demo]

Outline

- Antialiasing
- Compositing
- Depth Buffer
- Monitors

Warnock's Algorithm

1. All polys disjoint → background color
2. 1 contained or intersecting → background + color
3. 1 surrounding, no inters/ cont → fill
4. 1 surrounding in front of others → fill
5. Otherwise recurse



[courtesy of Andy Johnson / UIC]

Occlusion via the depth buffer

```
bool pass_depth_test(d1, d2) {
   return d1 < d2;
}

depth_test(tri_d, tri_color, x, y) {

  if (pass_depth_test(tri_d, zbuffer[x][y]) {

    zbuffer[x][y] = tri_d;     // update zbuffer
    color[x][y] = tri_color;   // update color buffer
  }
}
```

Occlusion using a depth buffer

- Store one depth value per coverage sample (not per pixel!)
- Constant space per sample
  - Implication: constant space for depth buffer
- Constant time occlusion test per covered sample
  - Read-modify write of depth buffer if "pass" depth test
  - Just a read if "fail"
- Not specific to triangles: only requires that surface depth can be evaluated at a screen sample point

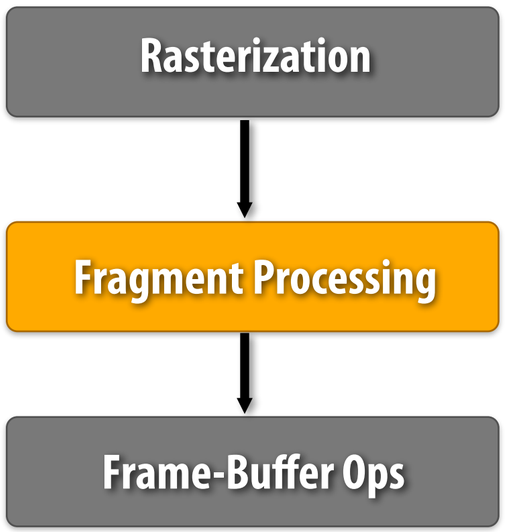
ATI HyperZ

- 3 technologies:
  - Hierarchical Z
  - Lossless Z compression
  - (Fast Z clear)

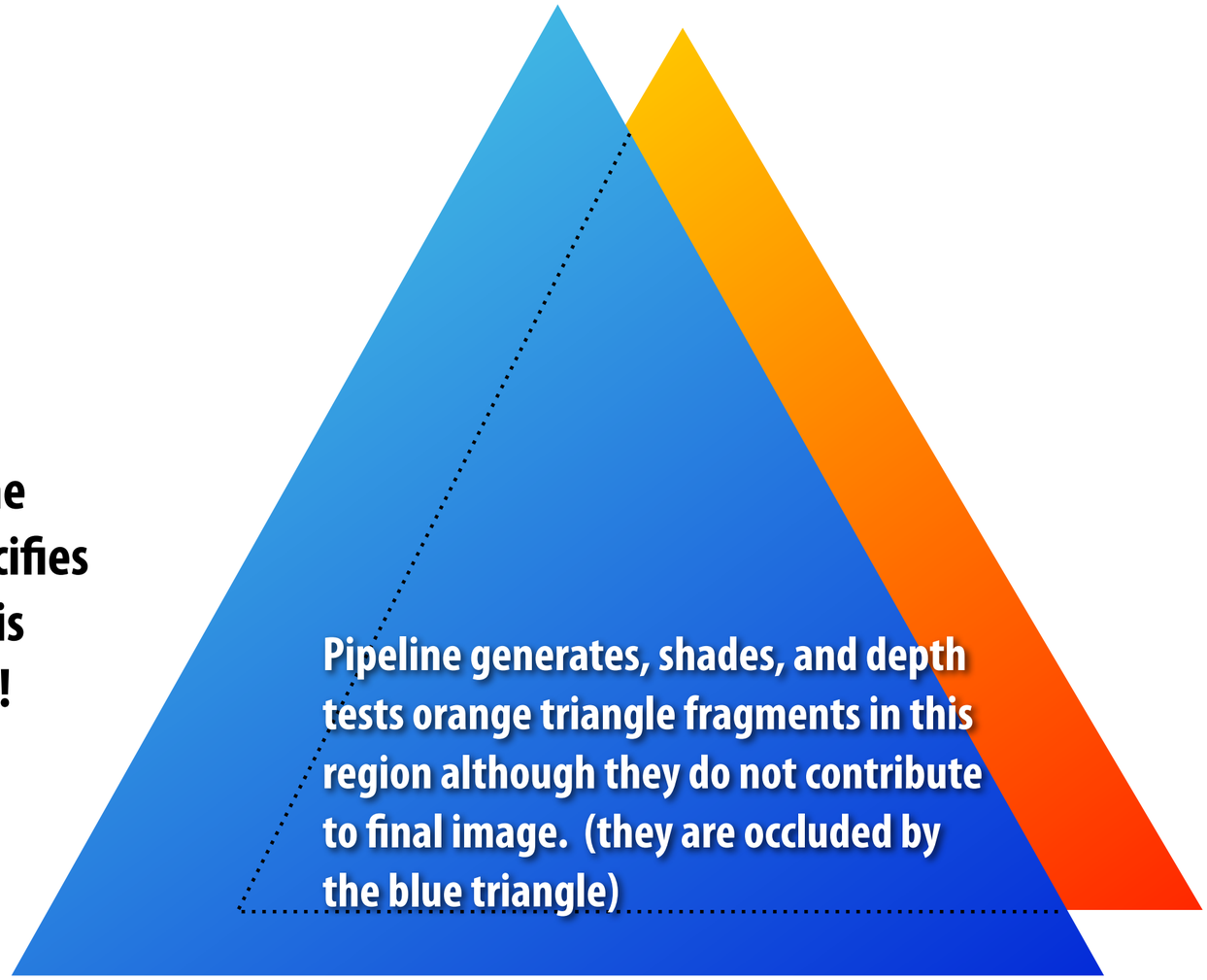
[www.yesky.com/20020930/1633308.shtml]

Early occlusion-culling ("early Z")

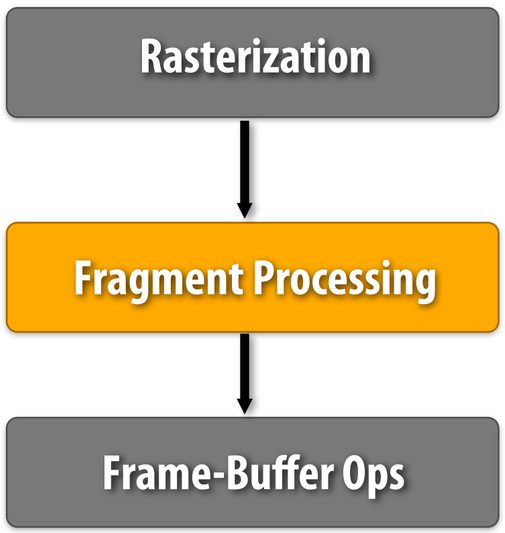
- Idea: discard fragments that will not contribute to image as quickly as possible in the pipeline



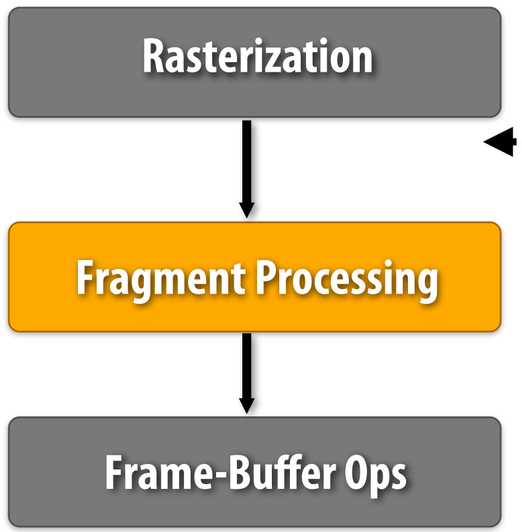
Graphics pipeline abstraction specifies that depth test is performed here!



Early occlusion-culling ("early Z")



Graphics pipeline specifies that depth test is performed here!



Optimization: reorder pipeline operations: perform depth test immediately following rasterization and before fragment shading

A GPU implementation detail: not reflected in the graphics pipeline abstraction

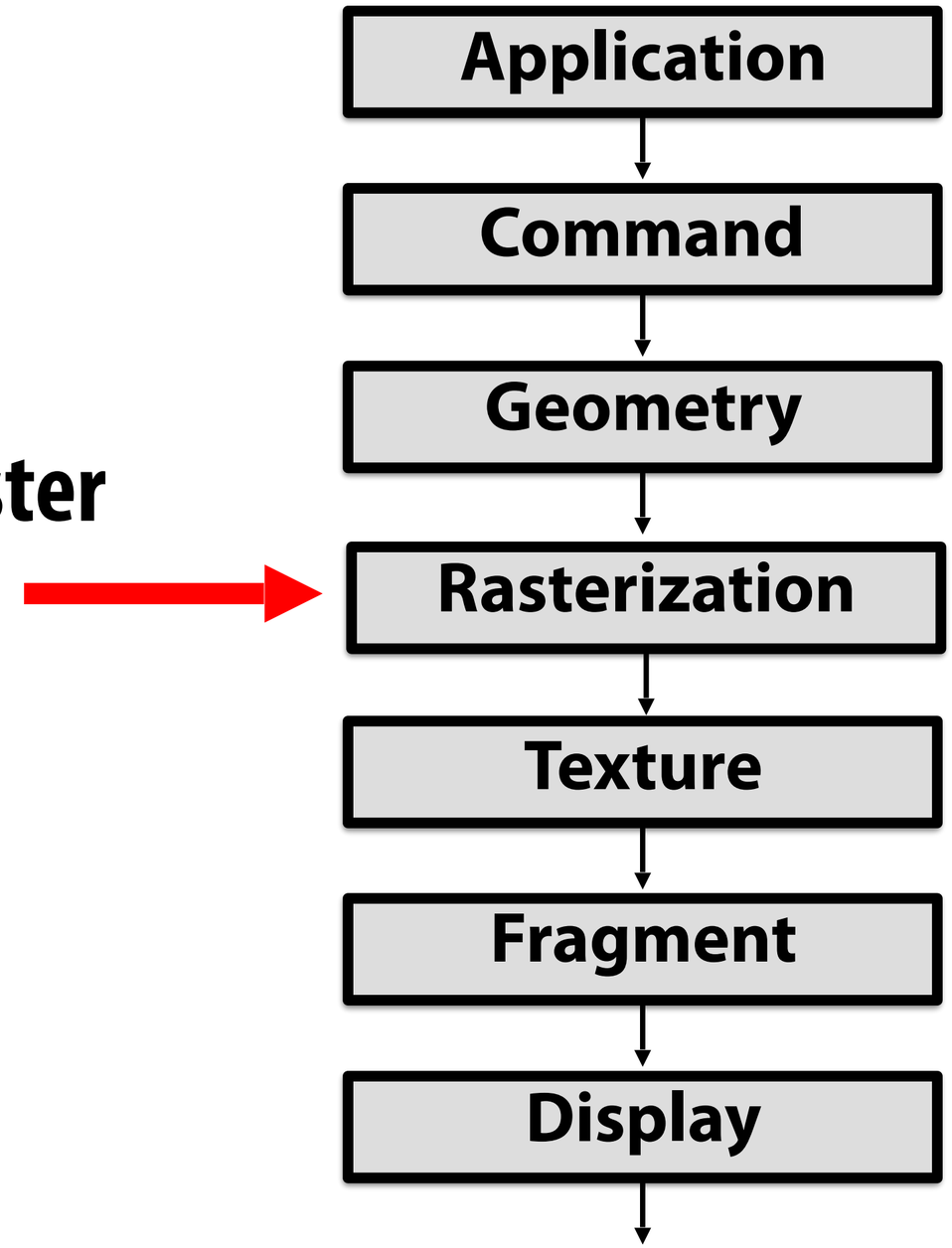
Key assumption: occlusion results do not depend on fragment shading

- Example operations that prevent use of this early Z optimization: enabling alpha test, fragment shader modifies fragment's Z value

Note: early Z only provides benefit if closer triangle is rendered by application first!
(application developers are encouraged to submit geometry in as close to front-to-back order as possible)

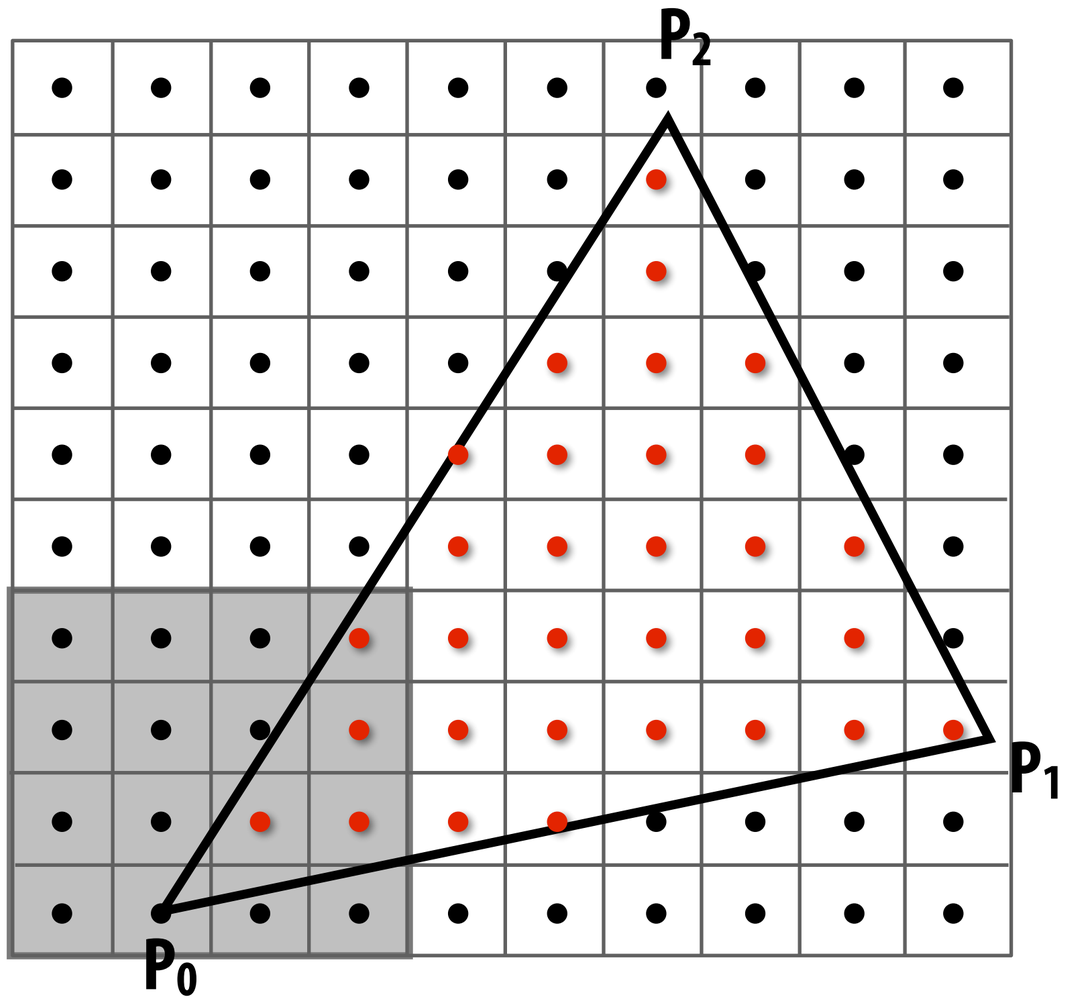
Hierarchical Z

- After triangle setup, before fragment ops
- Lower precision Z buffer with:
  - Max Z for tile: why?
  - Min Z for tile: why?
- Intimately associated with tiled raster
- All on chip (cache)

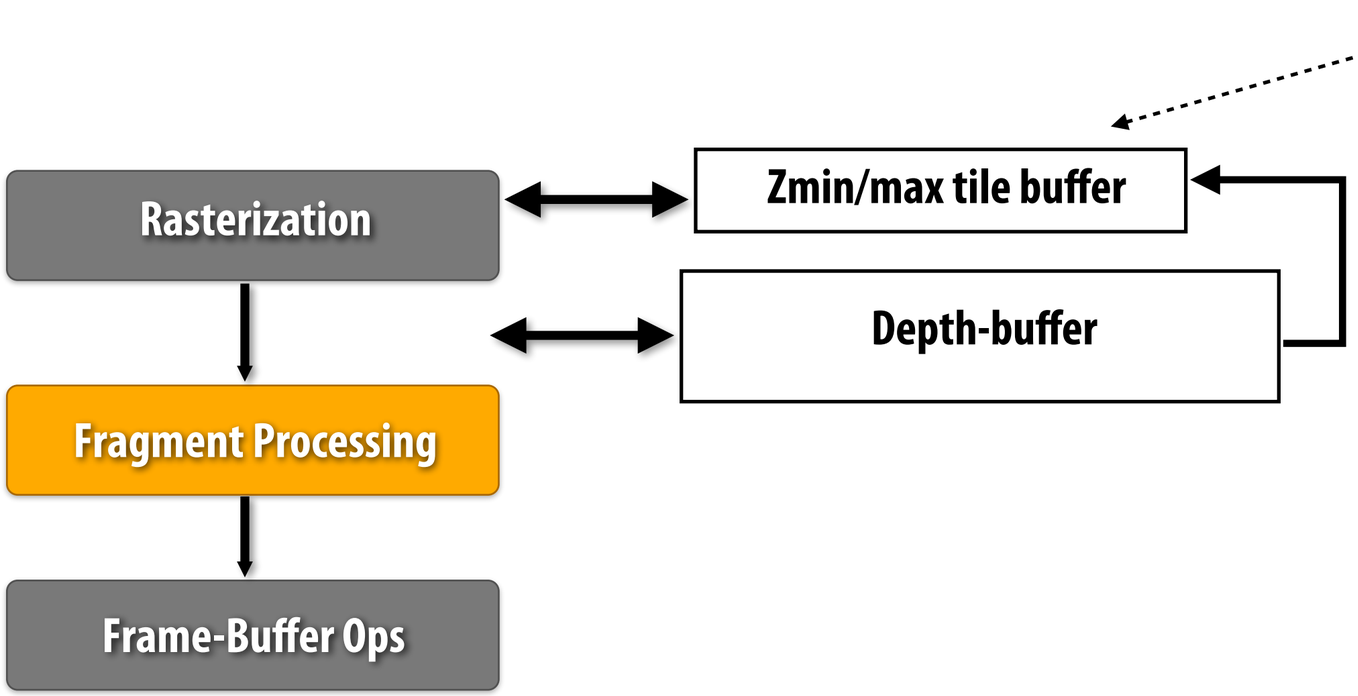


Hierarchical early occlusion culling: "hi-Z"

- Recall hierarchical traversal during rasterization
- *Z-Max culling:*
  - For each screen tile, compute farthest value in the depth buffer: z_max
  - During traversal, for each tile:
    - Compute closest point on triangle in tile: tri_min (using Z plane equation)
    - If tri_min > z_max, then triangle is completely occluded in this tile. (The depth test will fail for all samples in the tile.) Proceed to next tile without performing coverage tests for individual samples in tile.
- *Z-min optimization:*
  - Depth-buffer also stores z_min for each tile.
  - If tri_max < z_min, then all depth tests for fragments in tile will pass. (No need to perform depth test on individual fragments.)



Hierarchical Z + early Z-culling



Remember: these are GPU implementation details (common optimizations performed by most GPUs). They are invisible to the programmer and not reflected in the graphics pipeline abstraction

Depth-buffer compression

- **Motivation: reduce bandwidth required for depth-buffer accesses**
  - Worst-case (uncompressed) buffer allocated in DRAM
  - Conserving memory footprint is a non-goal

    (Need for real-time guarantees in graphics applications requires application to plan for worst case anyway)

- **Lossless compression**
  - Question: why not lossy?

- **Designed for fixed-point numbers (fixed-point math in rasterizer)**

Depth-buffer compression is tile based

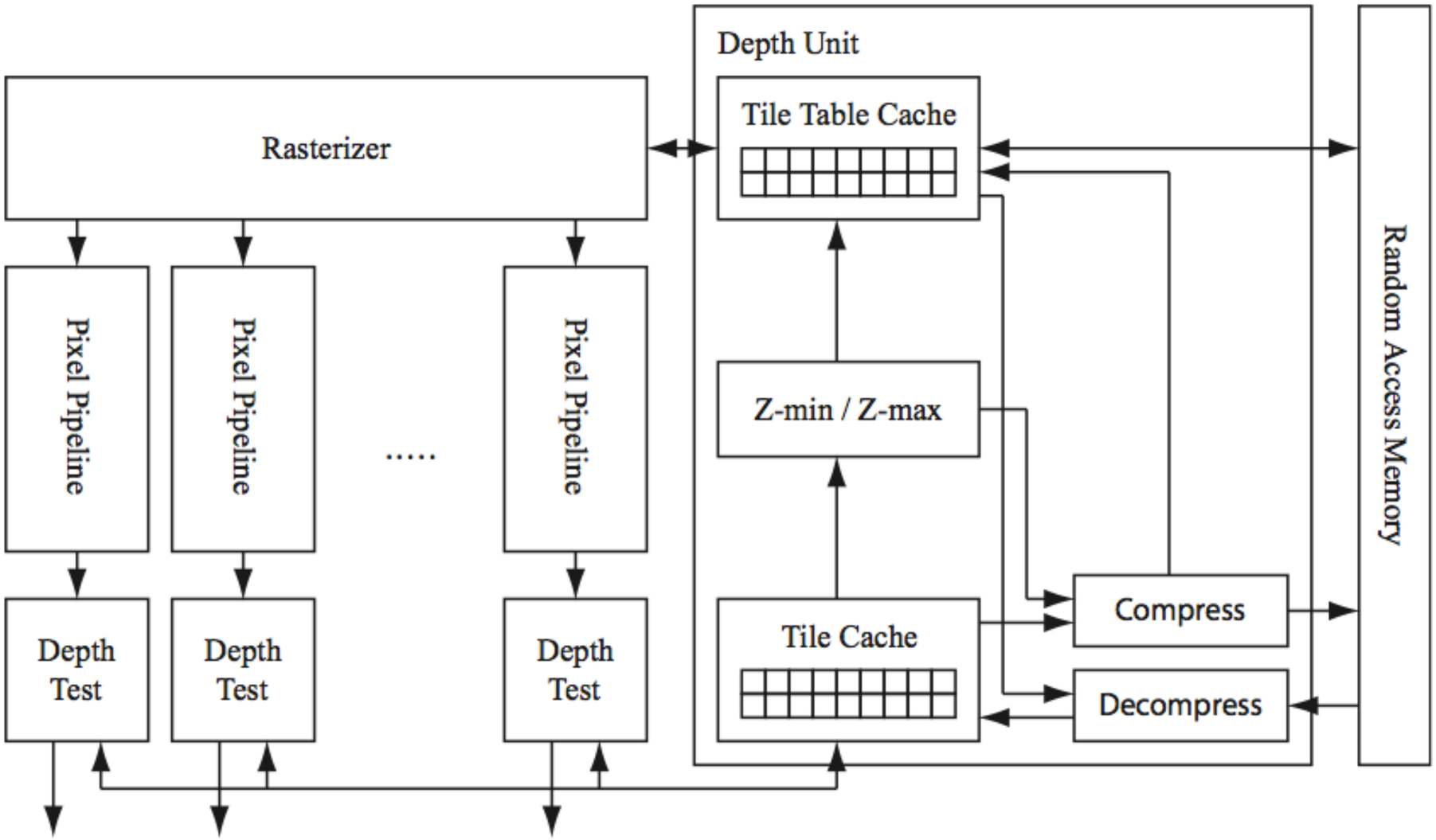


Figure credit: [Hasselgren et al. 2006]

On tile evict:

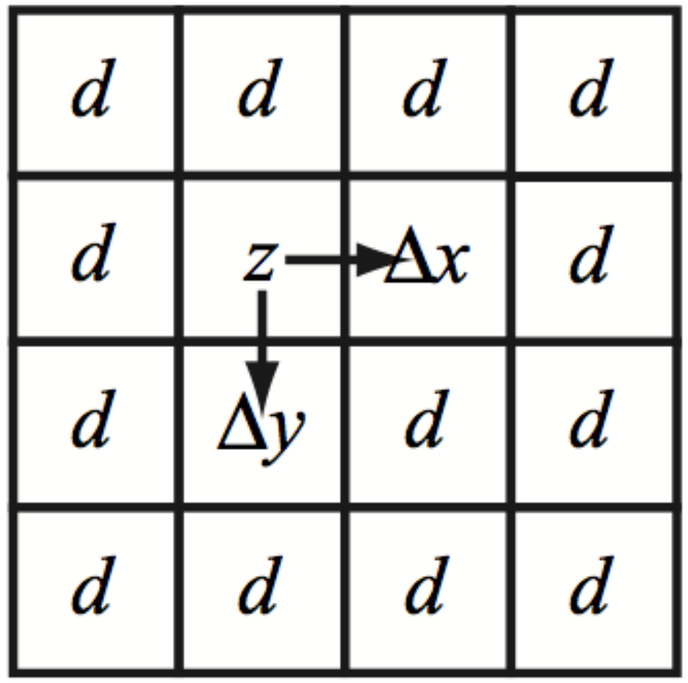
1. Compute zmin/zmax (needed for hierarchical culling and/or compression)
2. Attempt to compress
3. Update tile table
4. Store tile to memory

On tile load:

1. Check tile table for compression scheme
2. Load required bits from memory
3. Decompress into tile cache

Anchor encoding

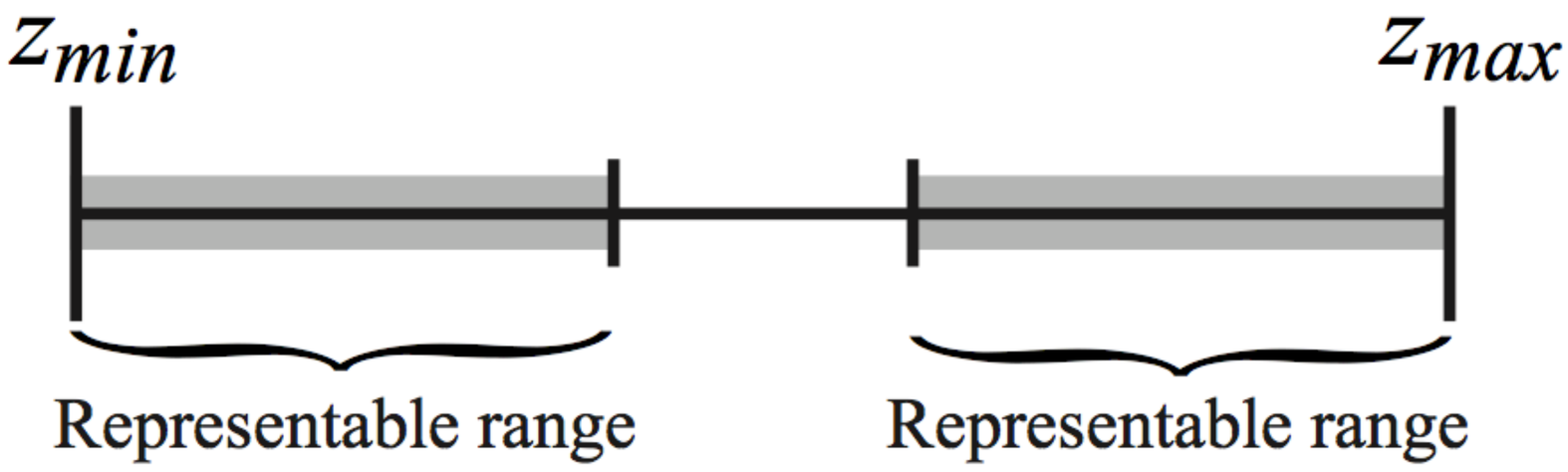
- Choose anchor value and compute DX, DY from adjacent pixels (fits a plane to the data)
- Use plane to predict depths at other pixels, store offset d from prediction at each pixel
- Scheme (for 24-bit depth buffer)
  - Anchor: 24 bits (full resolution)
  - DX, DY: 15 bits
  - Per-sample offsets: 5 bits



[Van Dyke and Margeson]

Depth-offset compression

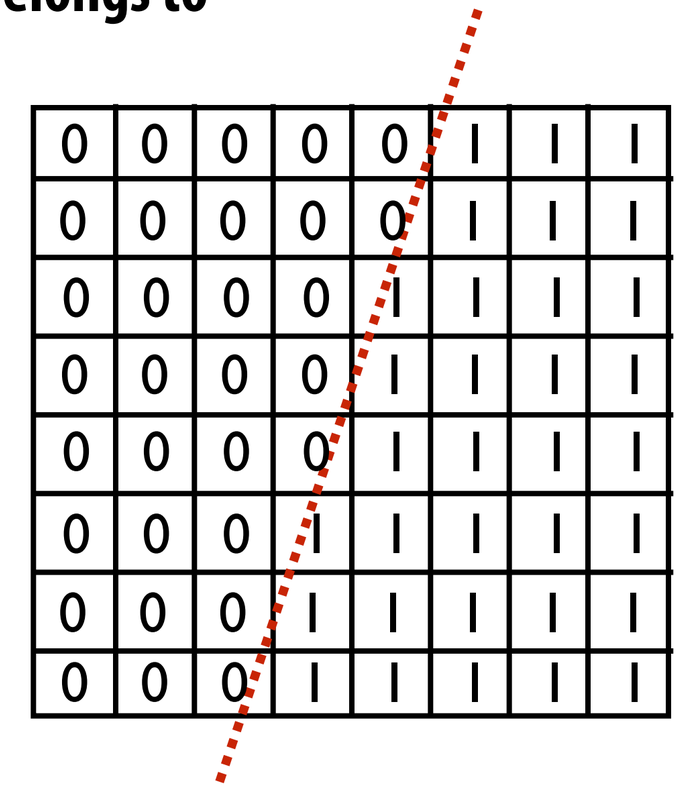
- Assume depth values have low dynamic range relative to tile's zmin and zmax (assume two surfaces)
- Store zmin/zmax (need to anyway for hierarchical Z)
- Store low-precision (8–12 bits) offset value for each sample
  - MSB encodes if offset is from zmin or zmax



[Morein and Natali]

Explicit plane encoding

- Do not attempt to infer prediction plane, just get the plane equation directly from the rasterizer
  - Store plane equation in tile (values must be stored with high precision: to match exact math performed by rasterizer)
  - Store bit per sample indicating coverage
- Simple extension to multiple triangles per tile:
  - Store up to N plane equations in tile
  - Store $\log_2(N)$ bit id per depth sample indicating which triangle it belongs to
- When new triangle contributes coverage to tile:
  - Add new plane equation if storage is available, else decompress
- To decompress:
  - For each sample, evaluate Z(x,y) for appropriate plane



Outline

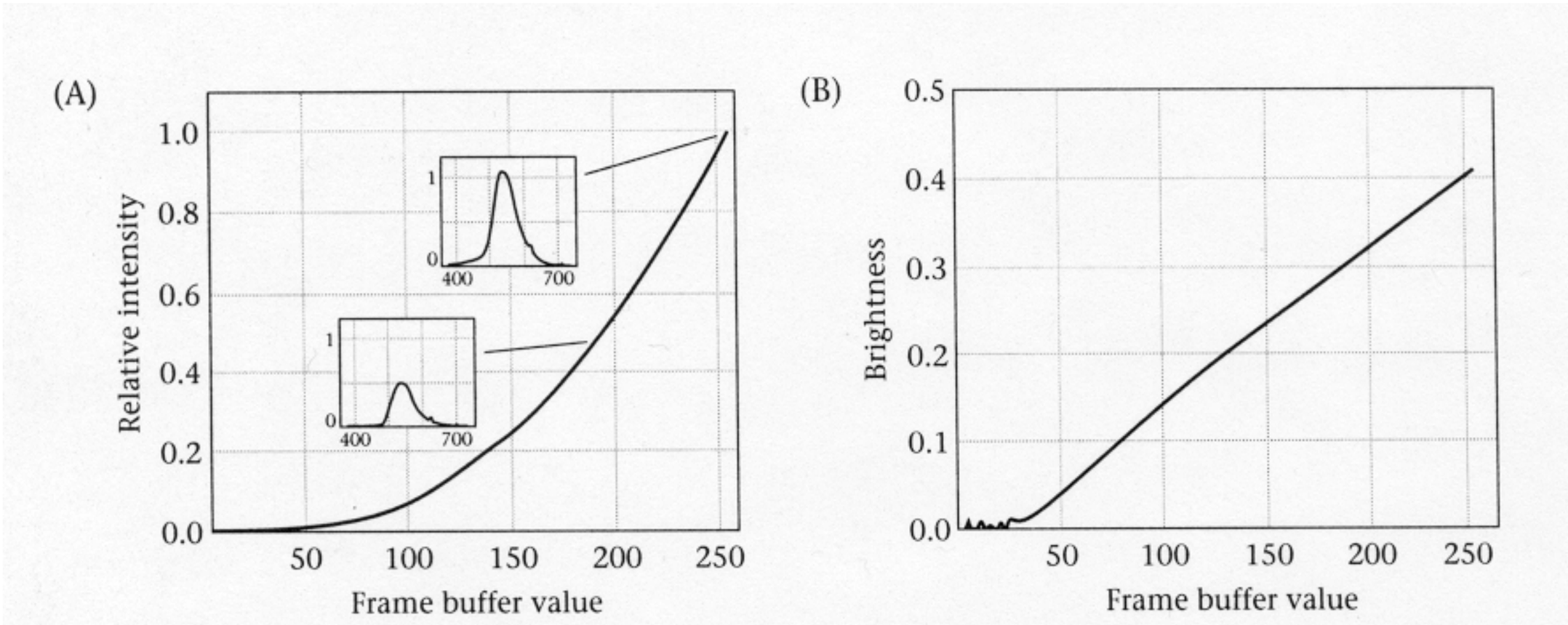
- Antialiasing
- Compositing
- Depth Buffer
- Monitors

Intensity

- CRT has nonlinear response to input signal
  - $I = a\,(V + e)^{gamma}$
  - Gamma: nonlinearity of monitor, usually ~2.2
  - I: intensity; V: voltage; e: brightness knob
- Eye has nonlinear response to intensity
  - Minimum visible (static) contrast ratio is 1%
  - Perceived brightness = k Intensity 0.4
- Combined response is near-linear
  - 0.4 (2.2) ~= 1.0

"In LCDs such as those on laptop computers, the relation between the signal voltage VS and the intensity I is very nonlinear and cannot be described with gamma value. However, such displays apply a correction onto the signal voltage in order to approximately get a standard $\gamma = 2.5$ behavior. In NTSC television recording, $\gamma = 2.2$." —Wikipedia

Graphs

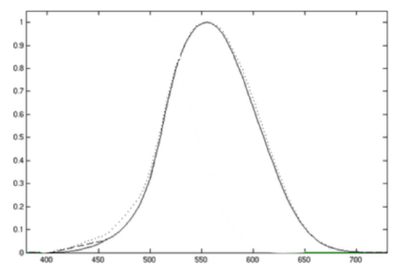
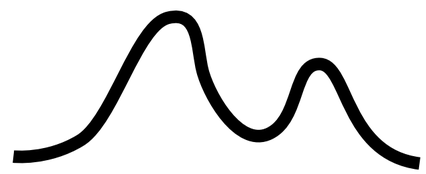


B.2 FRAME-BUFFER VALUE AND DISPLAY INTENSITY. (A) The curve measures the intensity of the emitted light relative to the maximum intensity. The data shown are for the green phosphor. The insets in the graph show the complete spectral power distribution of the light at two different frame-buffer levels. (B) The curve describing the relative intensities is replotted, using Stevens's power law, to show the linear relationship between the frame-buffer value and perceived brightness.

Gamma Correction

- Store image linear in brightness
  - We perceive brightness
  - Best use of available storage precision
  - 256 representable levels are enough
  - Requires conversion for each pixel operation
  - Historically unusual design choice
- Store image linear in intensity
  - Intensity is a physical value
  - Native arithmetic format
  - Requires conversion during display
  - Large brightness steps at low intensities
  - 256 representable levels are not enough!
  - Historically typical design choice
  - Also the right thing to do

Lightness (<u>perceived</u> brightness)

Lightness (L*) $\xleftarrow{?}$ **Luminance (Y)** $= \int_{\lambda}$  **Spectral sensitivity of eye** $*$ 

Radiance (energy spectrum from scene)

Dark adapted eye: $L^* \propto Y^{0.4}$

Bright adapted eye: $L^* \propto Y^{0.5}$

In a dark room, you turn on a light with luminance: Y_1

You turn on a second light that is identical to the first. Total output is now: $Y_2 = 2Y_1$

Total output appears $2^{0.4} = 1.319$ **times brighter to dark-adapted human**

Aside: gamma correction (non-linear correction for CRT display)

Old CRT display:

1. Image contains value X
2. CRT display converts digital signal to an electron beam voltage V(x) (linear relationship)
3. Electron beam voltage converted to light: (non-linear relationship)

$$Y \propto V(\gamma)$$

Where: $\gamma \approx 2.5$

(Note: this effect does not apply to modern LCD displays, whose luminance output is linearly proportional to input values!)

So if pixels store Y, what will the display's output look like?

Fix: pixels sent to display must store:

$$Y^{1/2.5} = Y^{0.4}$$

Observed display output **Desired display output**

Image credit: http://creativebits.org/mac_os_x/windows_vs_mac_monitor_gamma

Non-linear correction (for adaptation)

Goal: want viewer to perceive luminance <u>differences</u> as if they were present in the environment where a picture is taken (reproducing the absolute values of Y is often not practical)

Example: TV camera records Y

But now consider <u>ratio</u> of perceived brightness of these two pixels:

Viewer in outdoor scene (bright adapted): $\frac{Y_0^{0.5}}{Y_1^{0.5}}$

Not the same ratio due to differences in adaptation

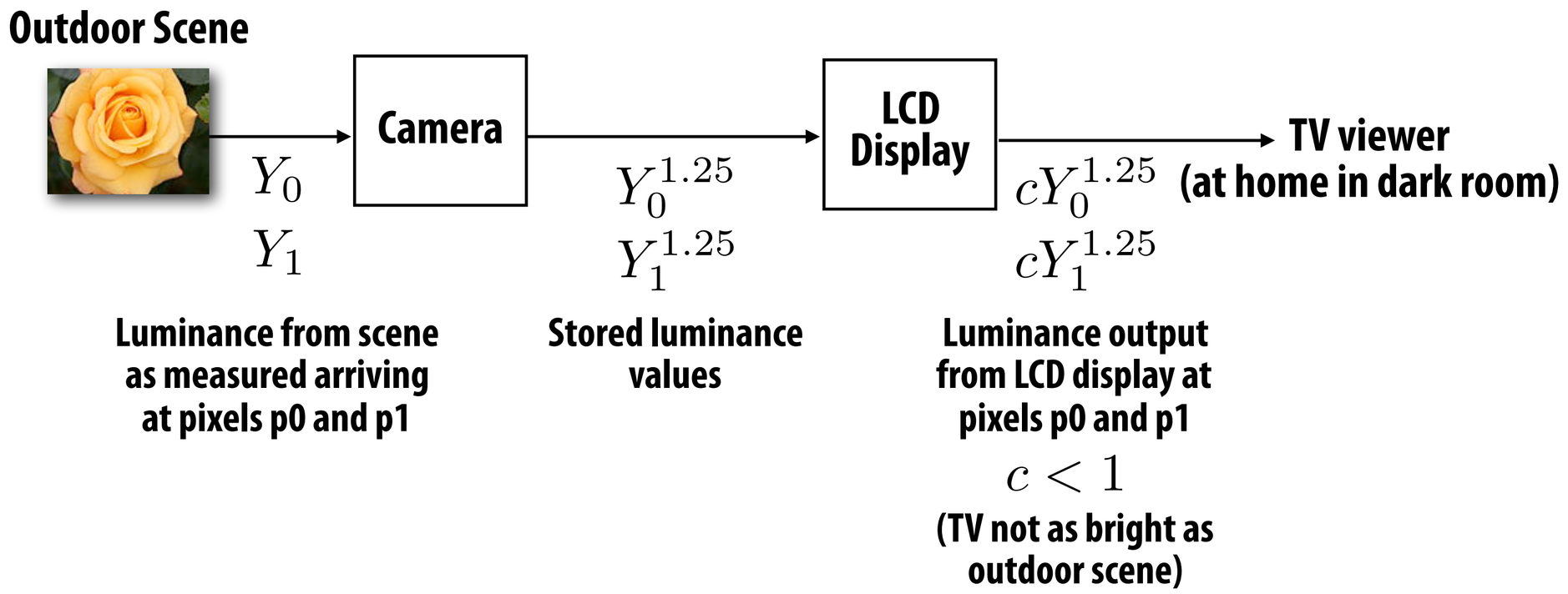
Viewer in living room at night (dark adapted): $\frac{cY_0^{0.4}}{cY_1^{0.4}} = \frac{Y_0^{0.4}}{Y_1^{0.4}}$

[Example credit: Marc Levoy]

Non-linear correction (for adaptation)

Goal: want viewer to perceive luminance <u>differences</u> as if they were present in the environment where a picture is taken (reproducing the absolute values of Y is often not practical)

Solution: TV camera stores $Y^{1.25}$

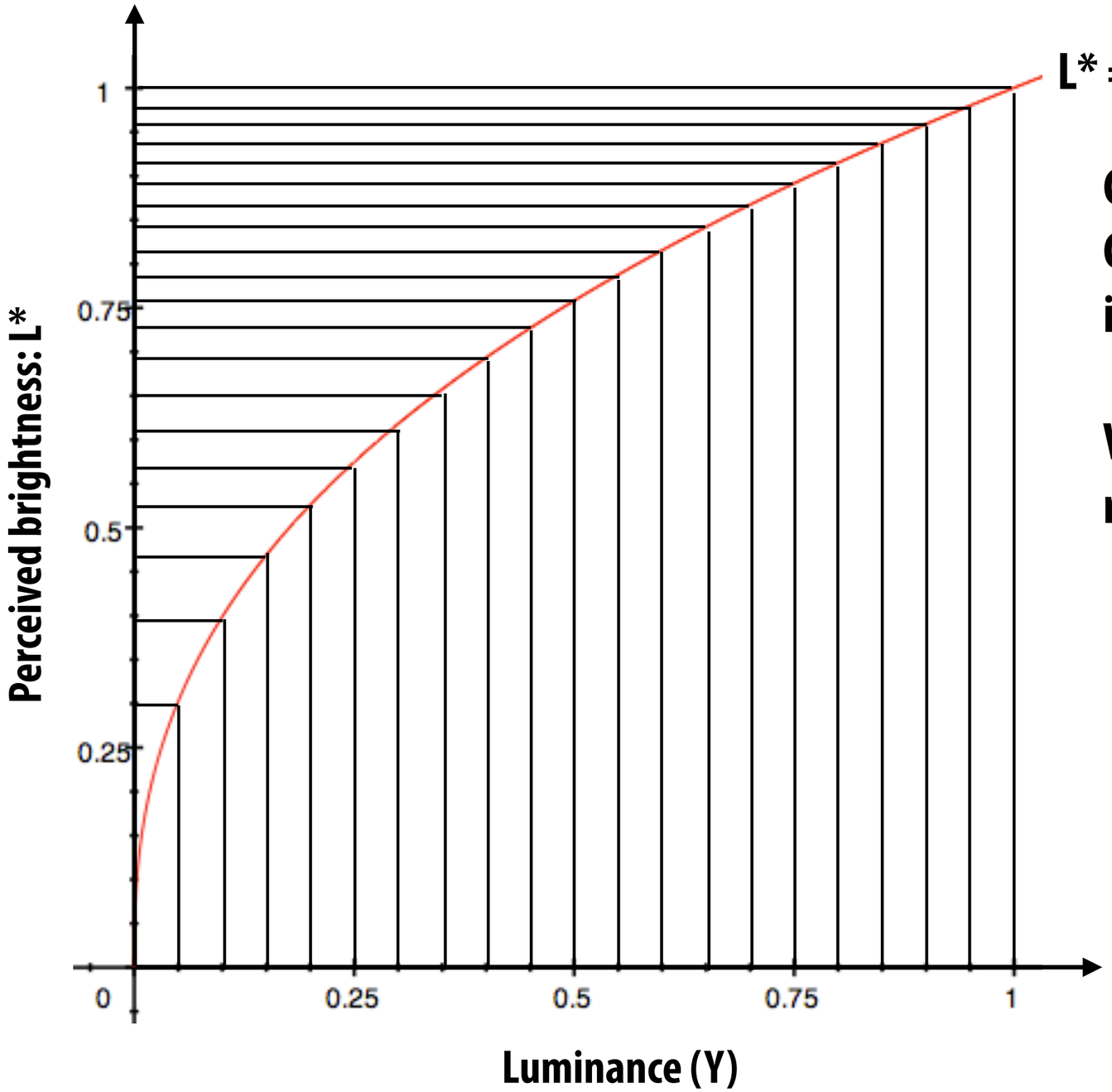


Viewer in outdoor scene (bright adapted): $\frac{Y_0^{0.5}}{Y_1^{0.5}}$ ← **Relative brightnesses of TV's pixels are perceived to be similar to those in real scene!**

Viewer in living room at night (dark adapted): $\frac{(cY_0^{1.25})^{0.4}}{(cY_1^{1.25})^{0.4}} = \frac{Y_0^{0.5}}{Y_1^{0.5}}$

[Example credit: Marc Levoy]

Another reason for non-linear correction: quantization error



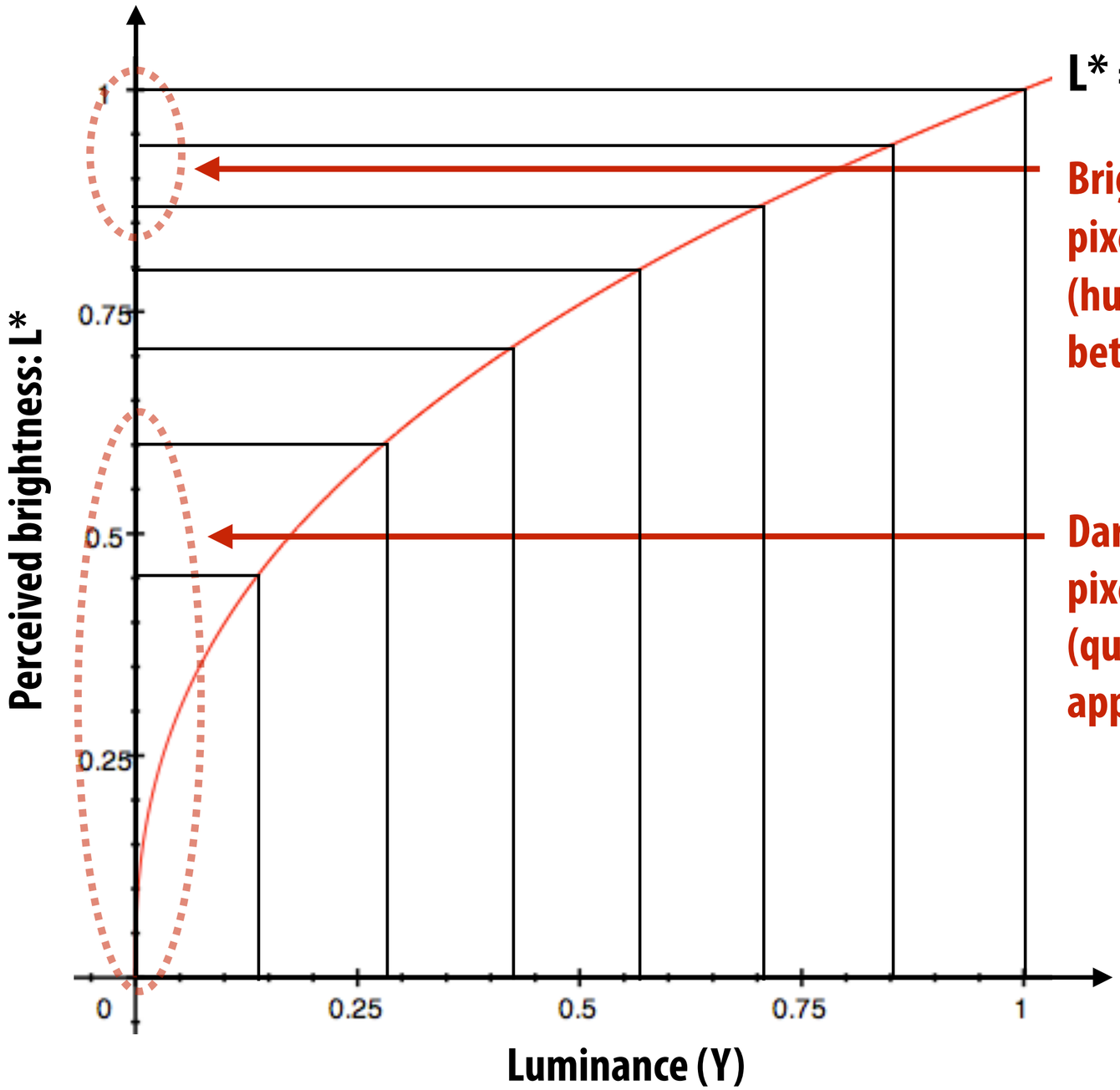
Consider 12-bit sensor pixel:
Can represent 4096 unique luminance values in output image

Values are ~ linear in luminance since they represent the sensor's response

Problem: quantization error

Many common image formats store 8 bits per channel (256 unique values)

Insufficient precision to represent brightness in darker regions of image



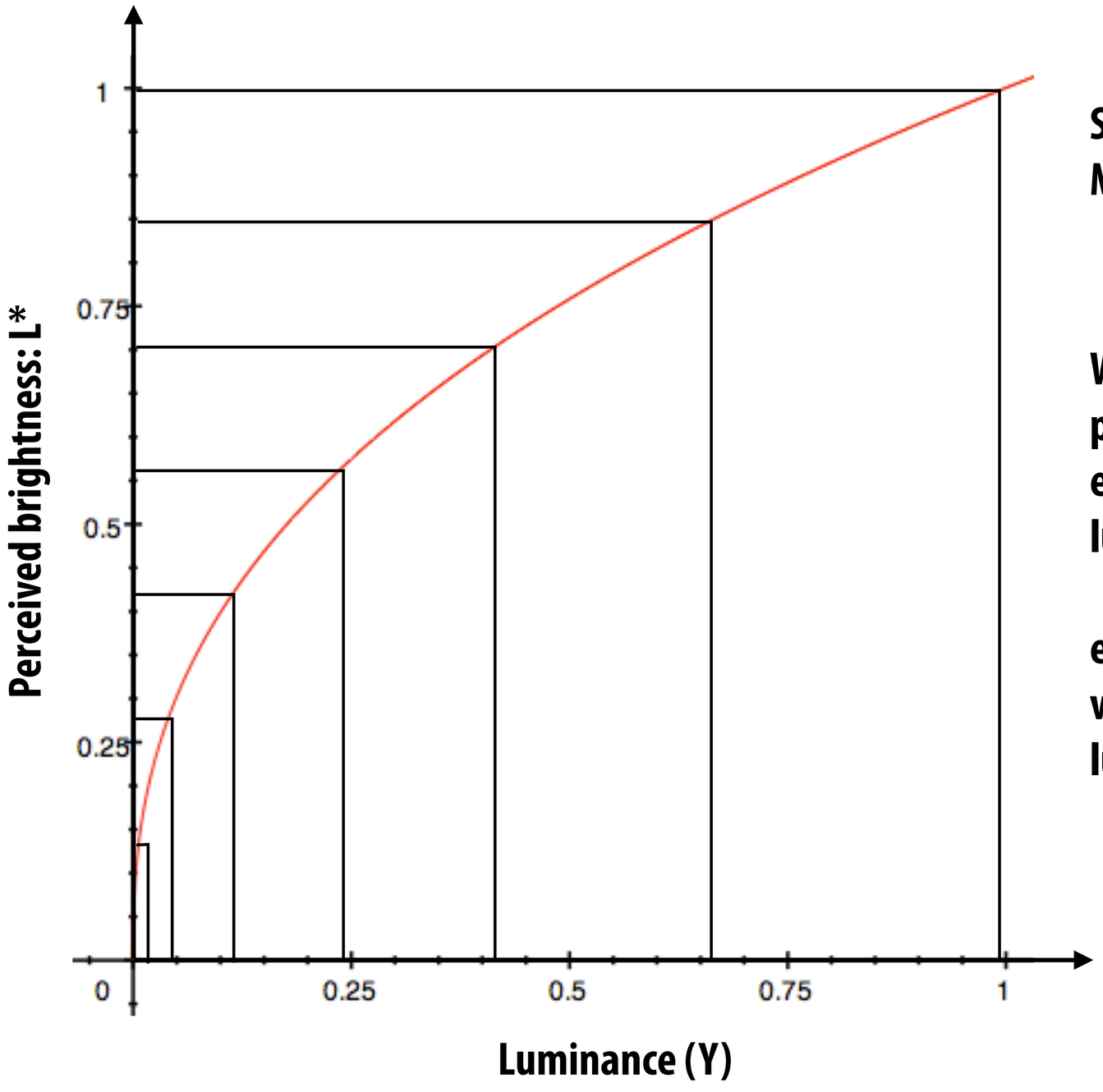
Bright regions of image: perceived difference between pixels that differ by one step in luminance is small! (human may not even be able to perceive difference between pixels that differ by one step in luminance!)

Dark regions of image: perceived difference between pixels that differ by one step in luminance is large! (quantization error: gradients in luminance will not appear smooth.)

Rule of thumb: human eye cannot differentiate <1% differences in luminance

Store lightness, not luminance

Idea: distribute representable pixel values evenly with respect to <u>perceived brightness</u>, not evenly in luminance (make better use of available bits)



Solution: pixel stores $Y^{0.45}$
Must compute $(pixel\_value)^{2.2}$ prior to display on LCD

Warning: must take caution with subsequent pixel processing operations once pixels are encoded in a space that is not linear in luminance.

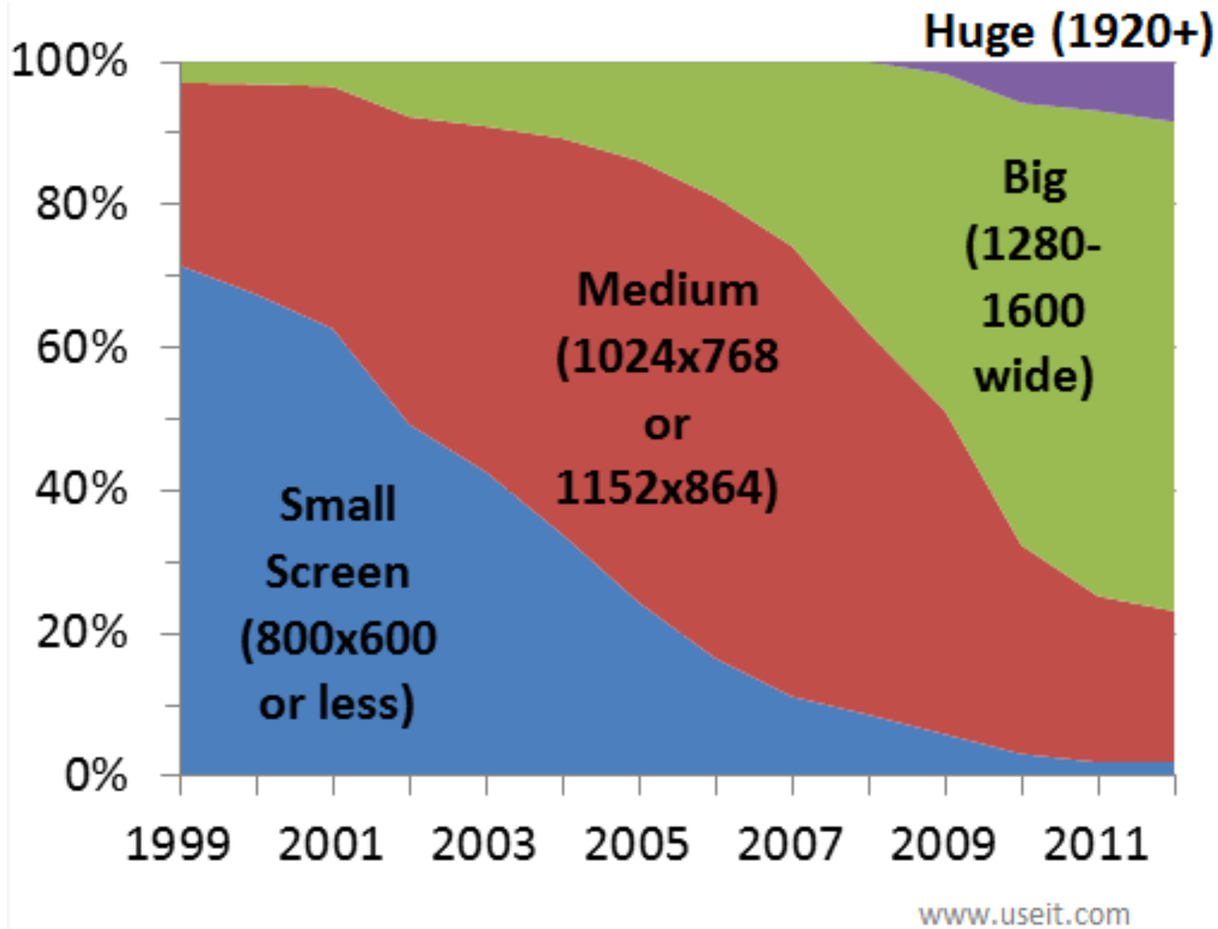
e.g., When adding images should you add pixel values that are encoded as lightness or as luminance?

Display Resolution History

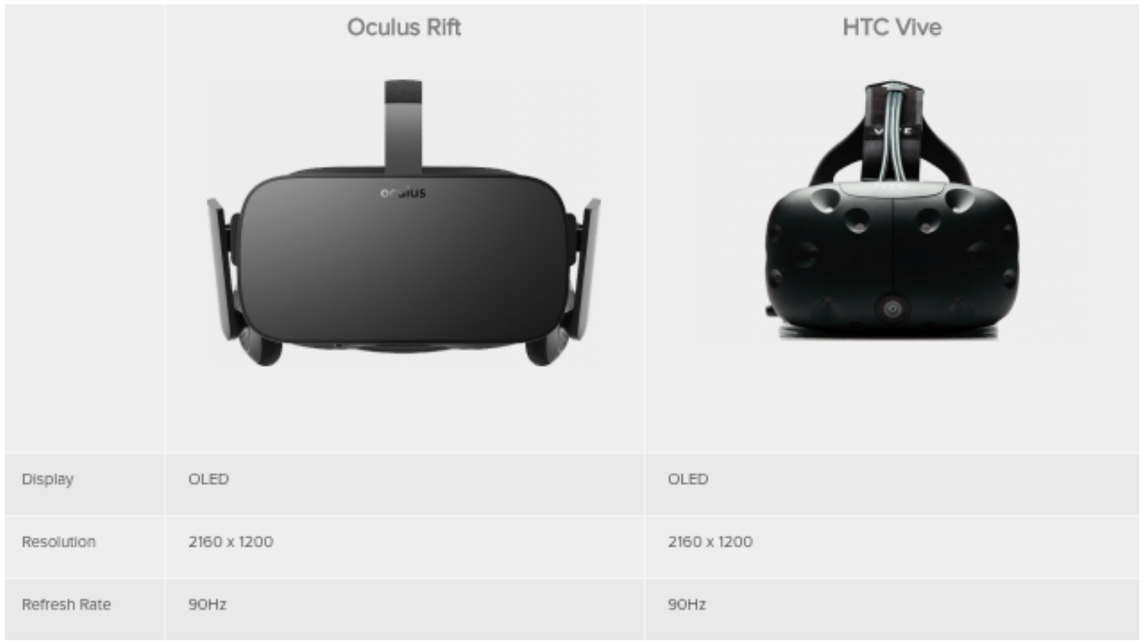
- Rate of increase is low (1.1 compound overall)

Date	Format and Technology	Bandwidth	Rate
1980	1024 x 768 x 60Hz, CRT	0.14 GB	
1988	1280 x 1024 x 72Hz, CRT	0.29 GB	1.1
1996	1920 x 1080 x 72Hz, HD CRT	0.60 GB	1.1
2001	3840 x 2400 x 56Hz, active LCD	1.55 GB	1.2

[Kurt Akeley estimate]



http://www.nngroup.com/articles/computer-screens-getting-bigger/



http://www.digitaltrends.com/virtual-reality/oculus-rift-vs-htc-vive/

High-dynamic range images

- Problem: ratio of brightest object to darkest object in real-world scenes can be quite large
  - Human eye can discern ratio of 100,000:1 (even more if accounting for adaptation)

- High-dynamic range (HDR) image: encodes large range of luminance (or lightness) values
  - Common format: 16-bits per channel EXR

- Modern camera senses can only sense much narrower range of luminances (e.g., 12-bit pixels)

- But most modern displays can only display a much narrower range of luminances
  - Luminance of white pixel / luminance of black pixel for a high-end LCD TV ~ 3000:1 *

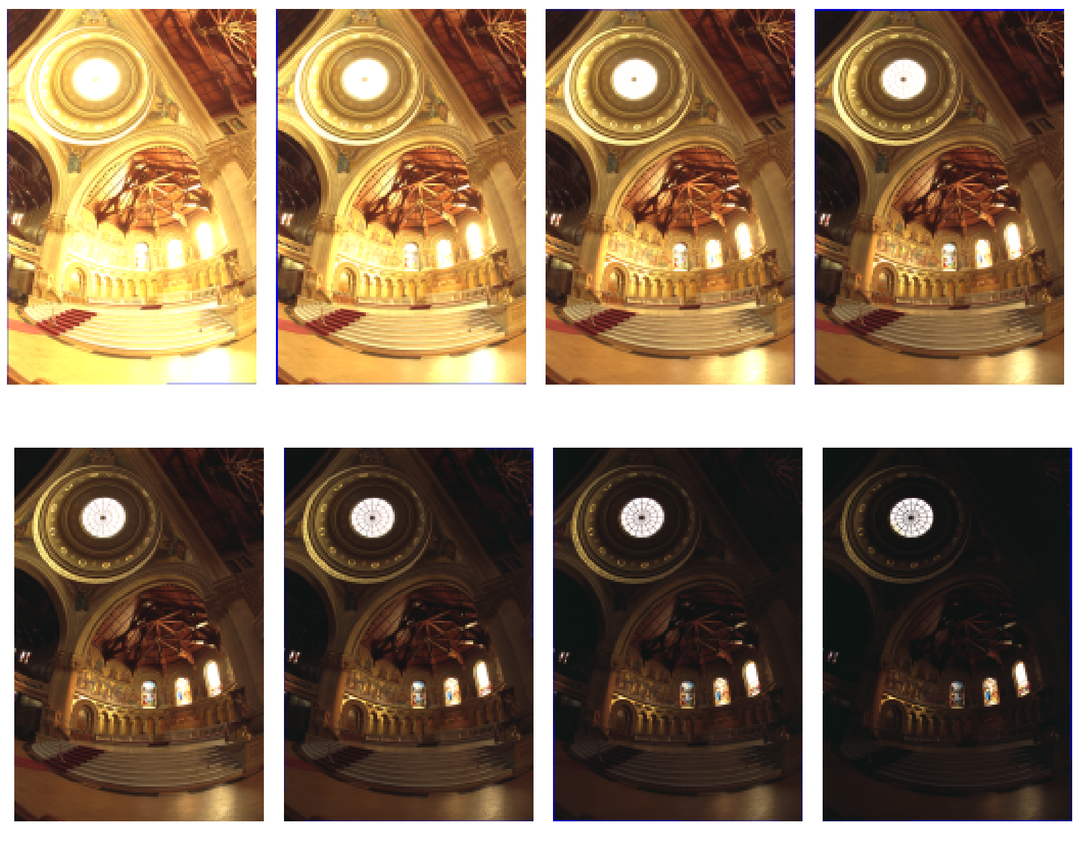
Overexposed (loss of detail in brightest areas since they are clamped to 1)

Underexposed (detail remains in brightest areas, but large regions of image clamped to 0)

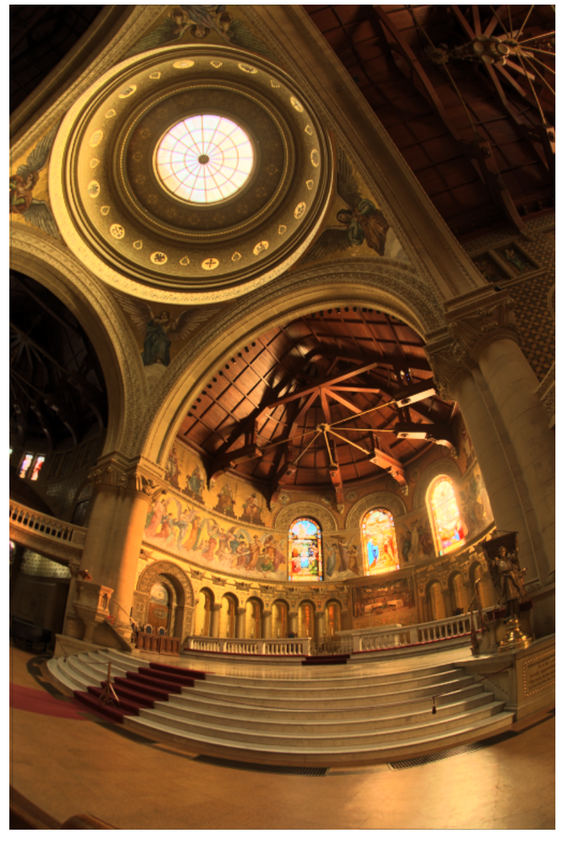
* Ignore most marketing specs, which are now claiming over 2,000,000:1

Tone mapping

- Tone mapping: non-linear mapping of wide range of luminances into a narrower range (for storage in low-bit depth image, or for presentation on a low-dynamic range display)
  - For examples see Debevec 1997, Reinhard 2002, Fattal 2002
- How to acquire HDR images with conventional camera?
  - Take photos at multiple exposures, combine into a single HDR image



Low-dynamic range images taken at multiple exposures (to capture detail in all regions)



Low-dynamic range image that is result of tone mapping HDR image

Special Purpose DRAM?

- ’80s: VRAM
- ’90s: FBRAM is DRAM with video output buffers (as in VRAM) and a cached ALU to perform fragment operations. Unsuccessful.
- *FBRAM: A New Form of Memory Optimized for 3D Graphics*, Deering, Schlapp, and Lavelle, SIGGRAPH ’94

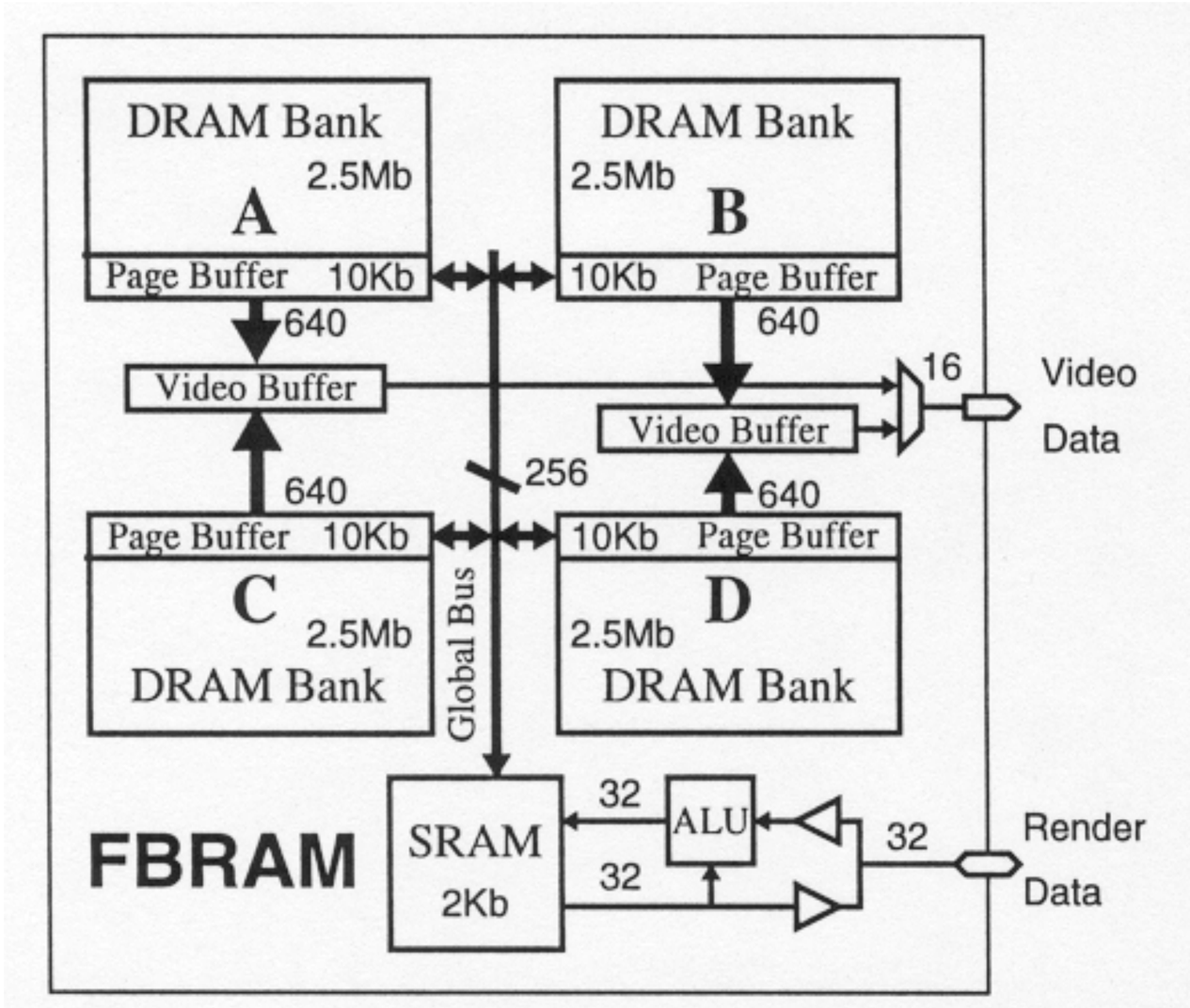


Figure 2. Internal block diagram of a single FBRAM.

Maximize Framebuffer Bandwidth

- Use the fastest, widest DRAMs possible
- Operate them at the highest possible clock rate
  - Separate "pixel" clock and "memory" clock
  - Bin memory (and GPU) parts
  - Provide elasticity (FIFO) and synchronization
- Make all wiring point-to-point
  - Optimize signal paths
  - Separate memory controller for each DRAM