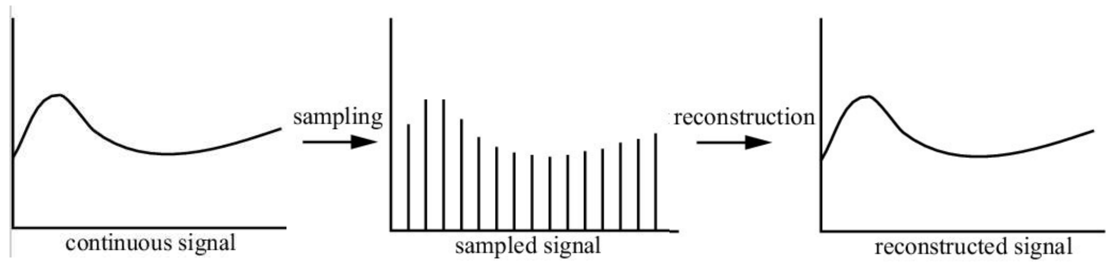
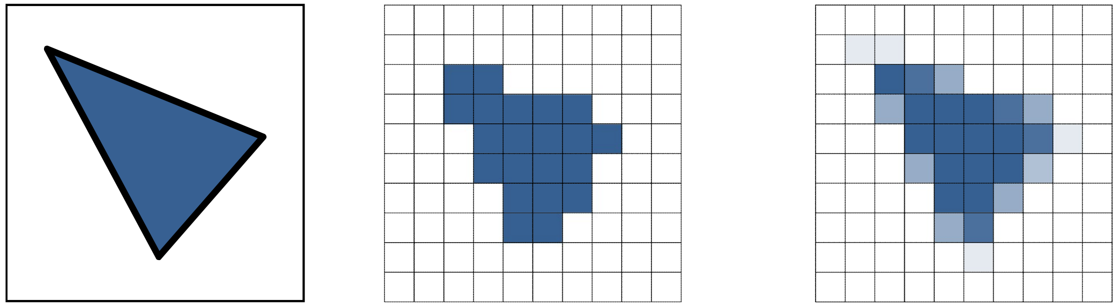
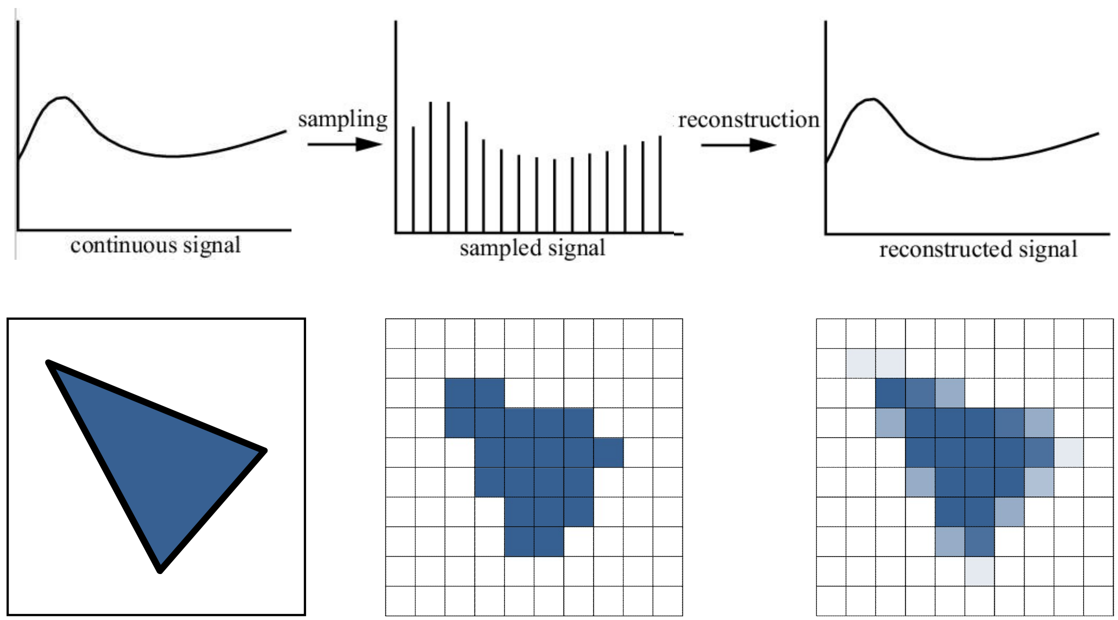



# Sampling and Reconstruction



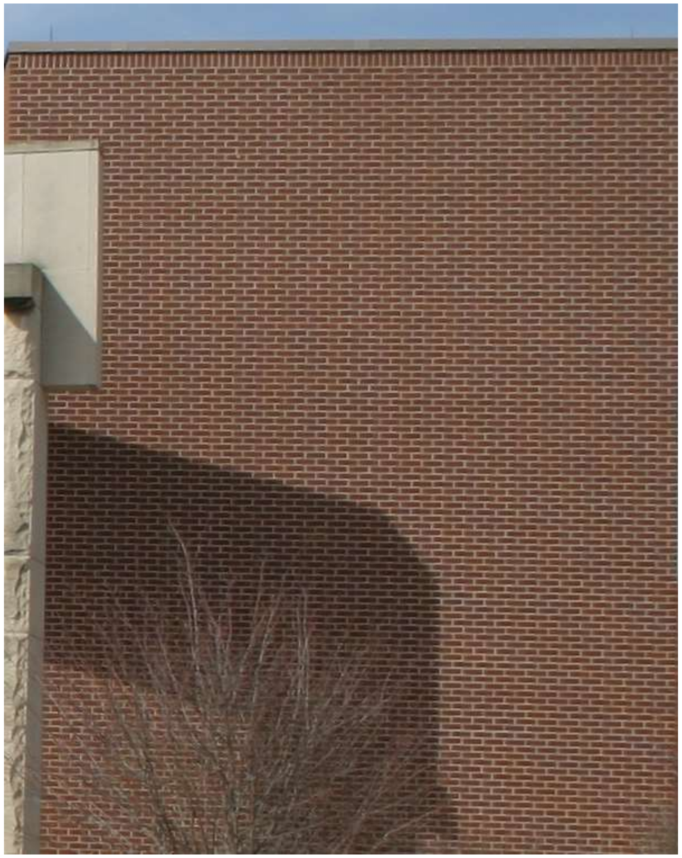
# Sampling and Reconstruction

- Sampling: from continuous signal to discrete
- Reconstruction recovers the original signal
- Errors are called aliasing



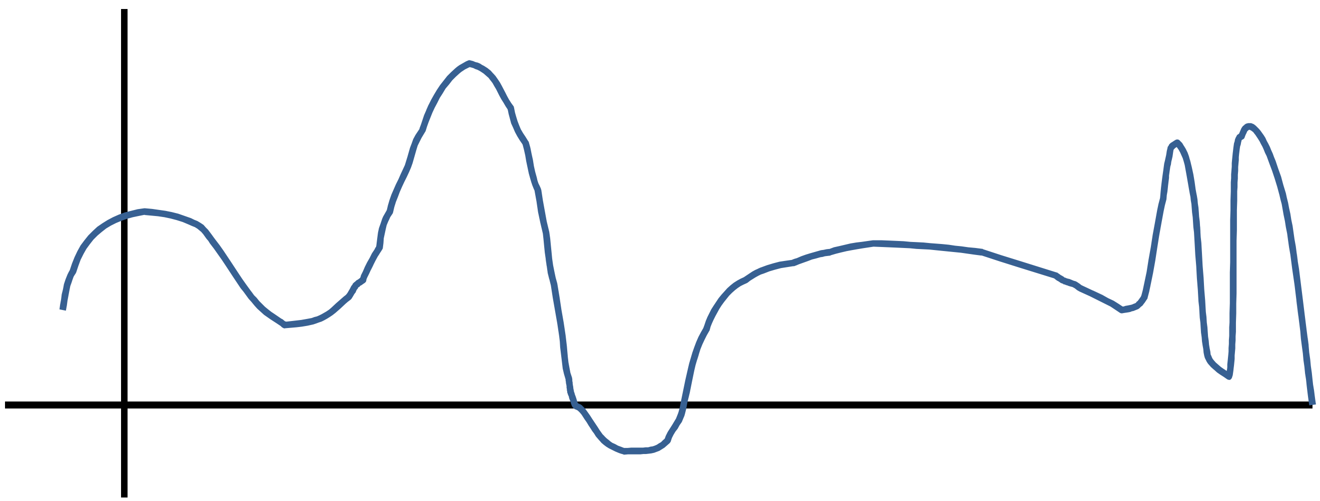
# Aliasing vs. Anti-Aliasing

- "alias" (word)
  - A name that has been assumed temporarily
  - Synonym, pseudonym
- Aliasing (signal processing)
  - *High frequencies that can not be represented alias (masquerade) as lower frequencies.*
- Aliasing (computer graphics)
  - Visual artefact
- Anti–aliasing (computer graphics)
  - Avoiding of unwanted artefacts

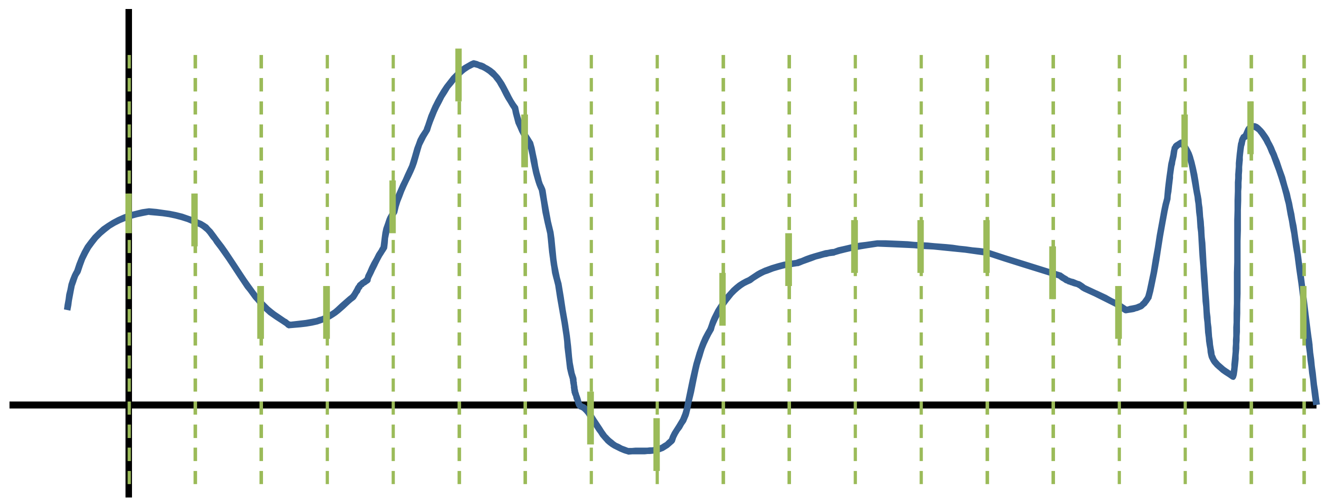


# Signal Processing

- Raster display: regular sampling of a continuous function
- Think about sampling a 1-D function

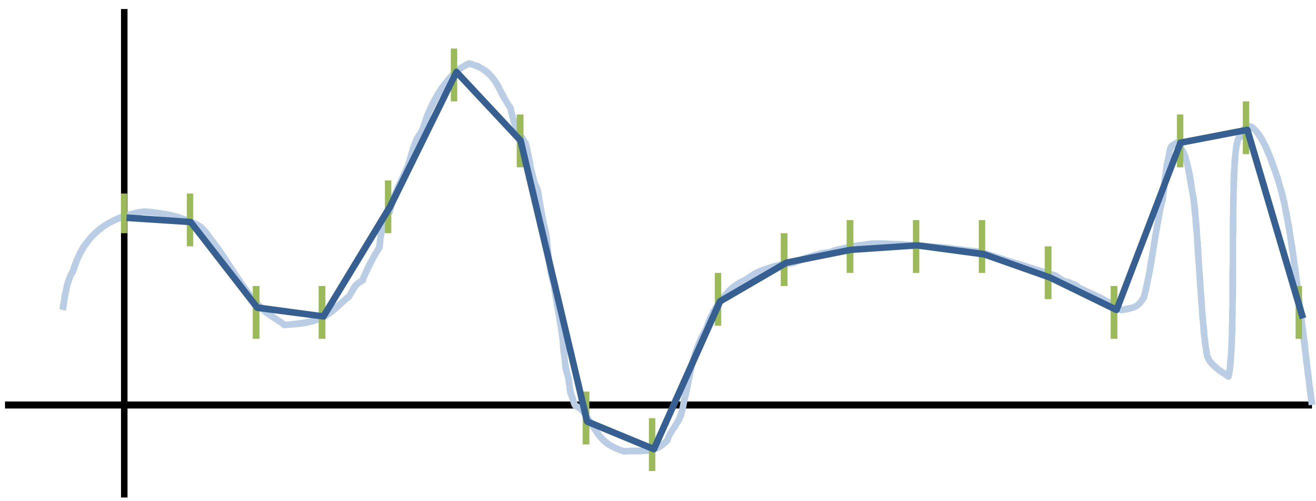


# Signal Processing – Sampling a 1-D function



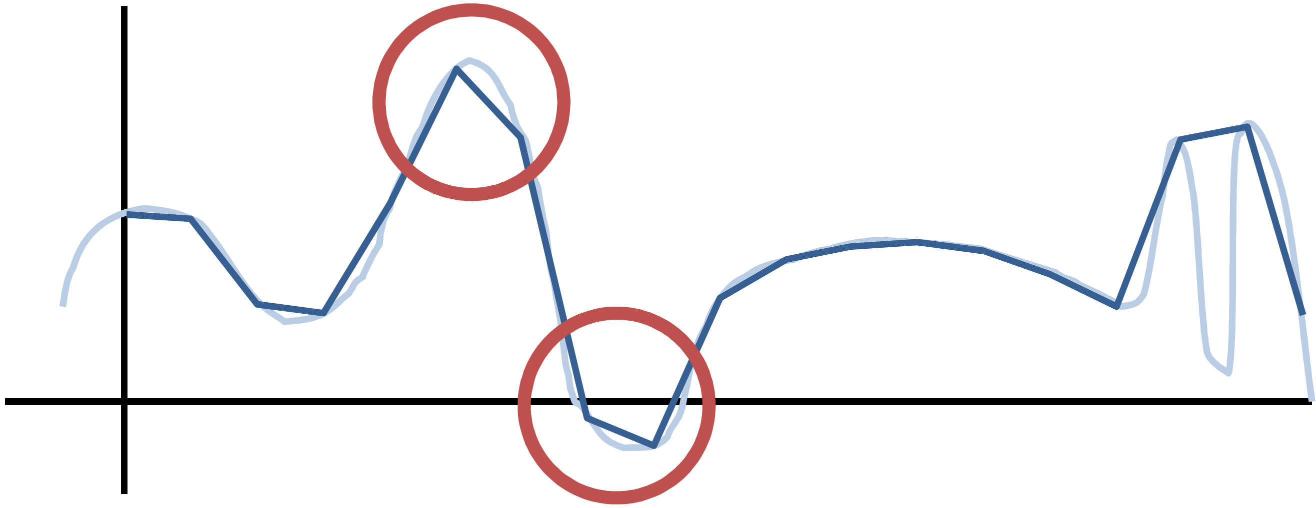
# Signal Processing – Sampling a 1-D function

- What do you notice?



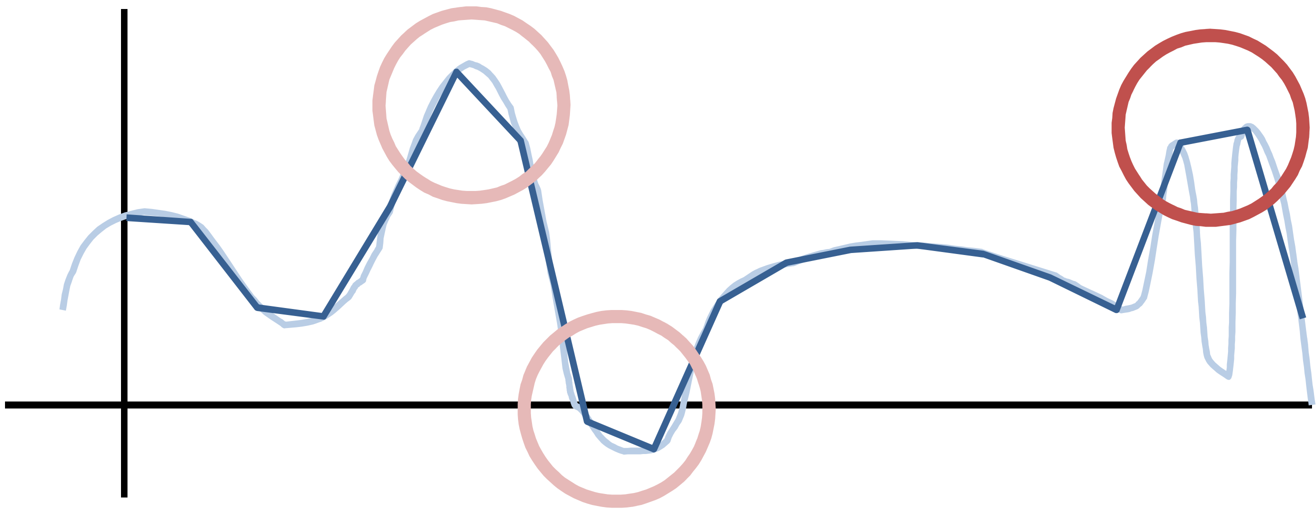
# Signal Processing – Sampling a 1-D function

- What do you notice?
  - Jagged, not smooth



# Signal Processing – Sampling a 1-D function

- What do you notice?
  - Jagged, not smooth
  - Loss of information

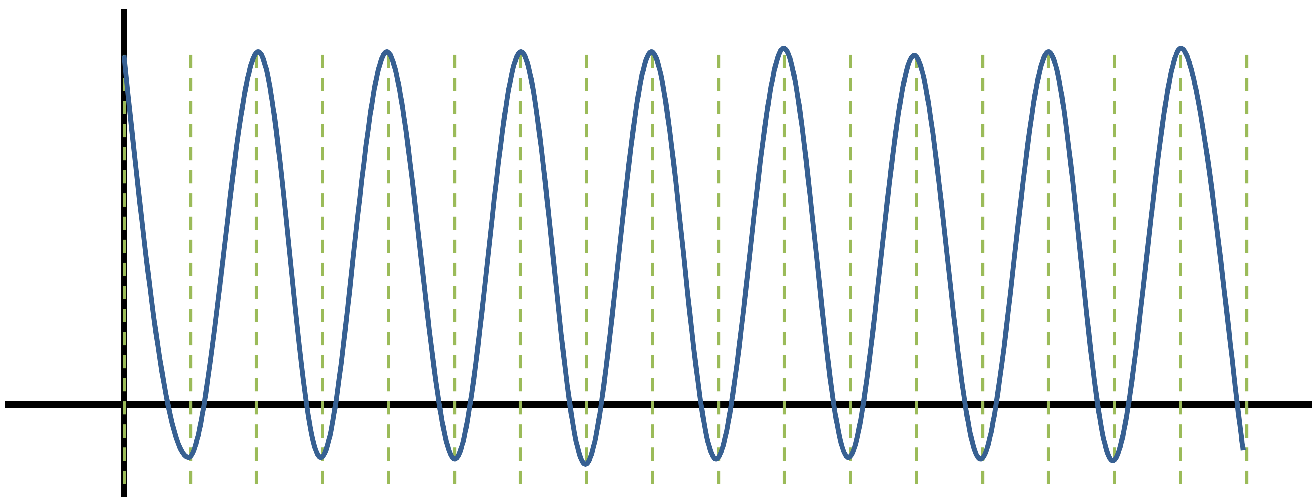


# Signal Processing

- What do you notice?
  - Jagged, not smooth
  - Loss of information
- What can we do about these?
  - Use higher-order reconstruction
  - Use more samples → better approximation
  - *How many more samples?*

# The Sampling Theorem

- Given a certain sampling
- What is the fastest changing function that can be expressed this way?
- Frequency?



# The Sampling Theorem

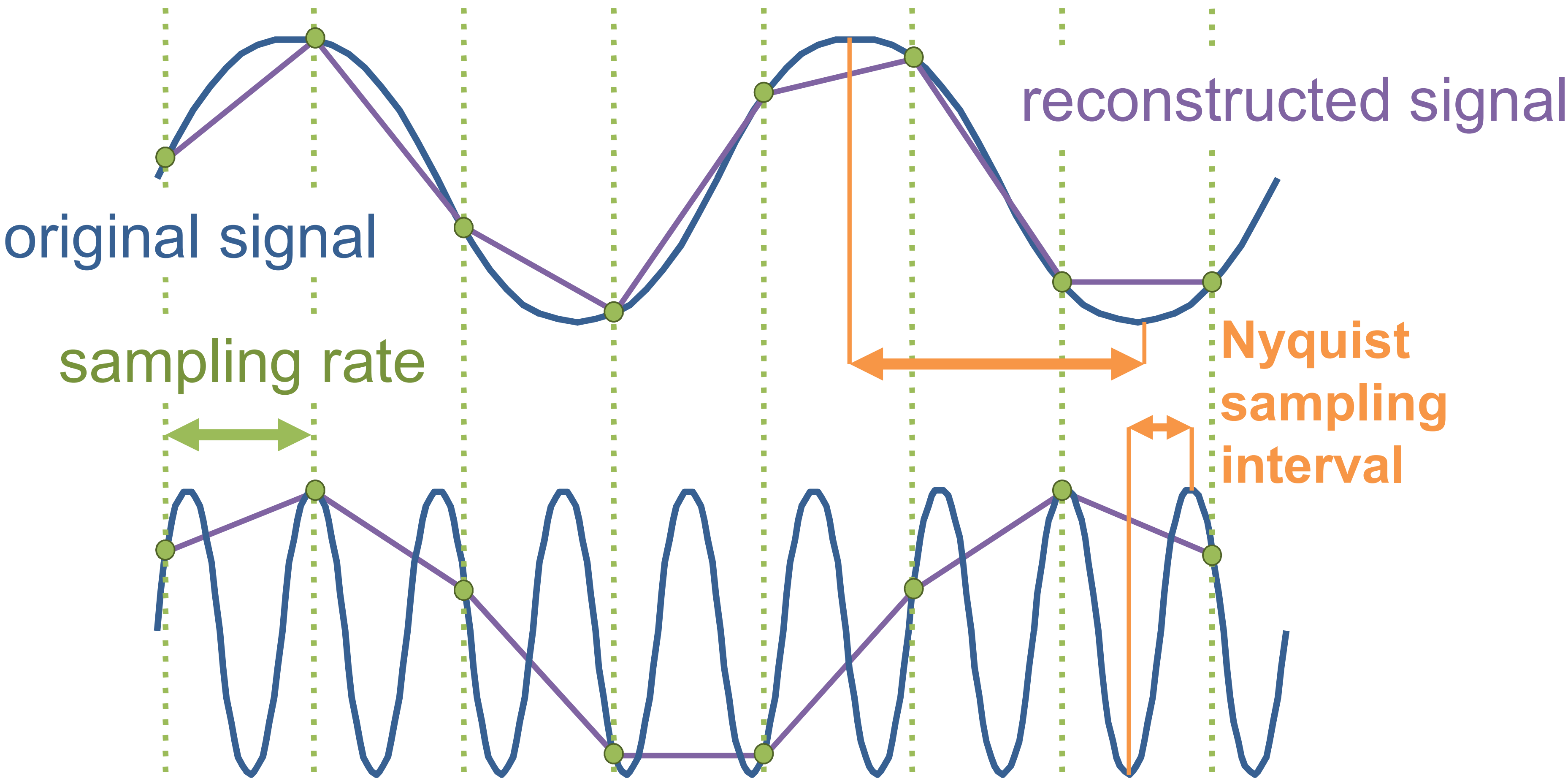
*A signal can only be reconstructed without loss of information if the sampling frequency is at least twice the highest frequency of the signal*

# The Sampling Theorem

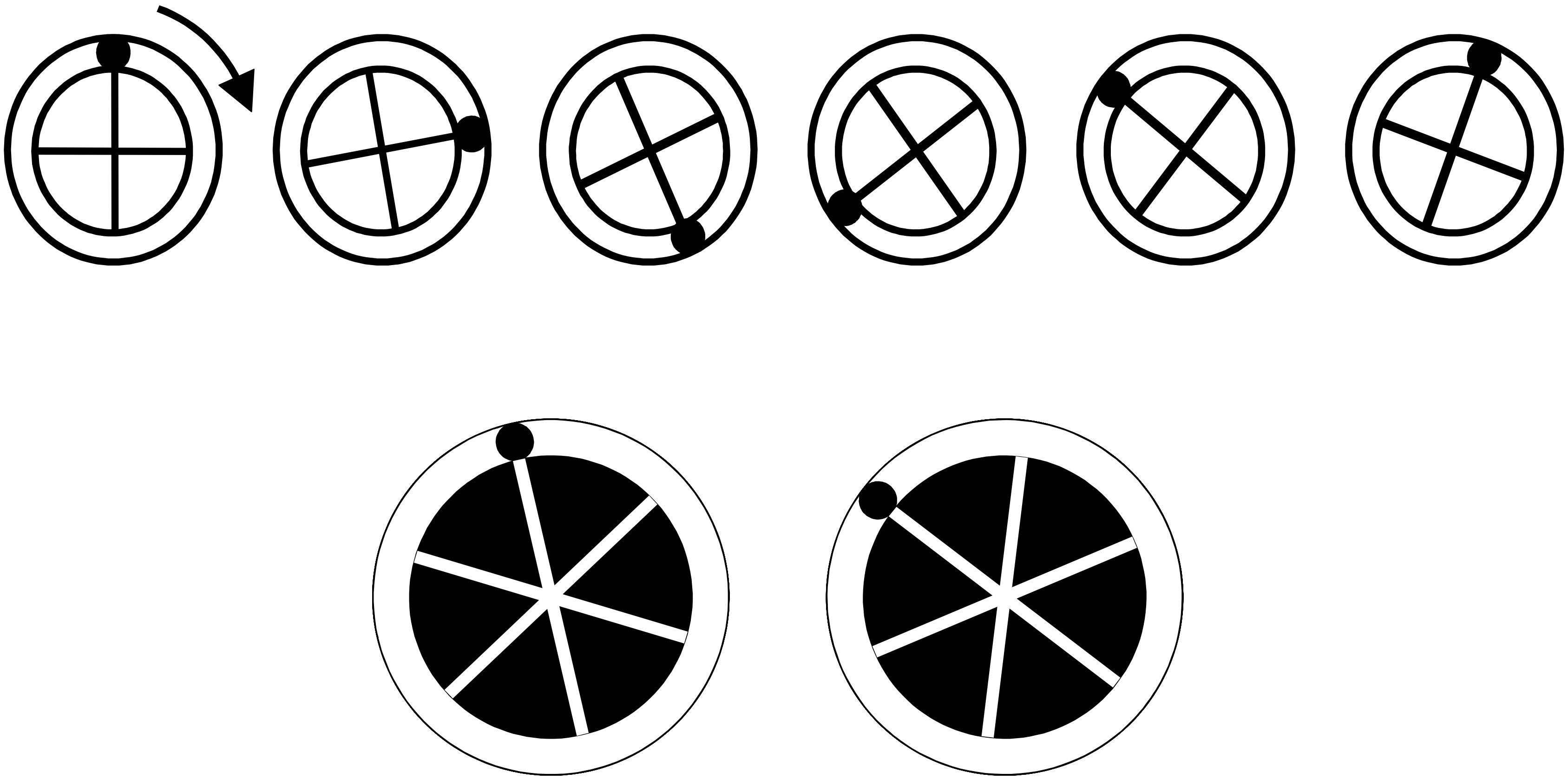
- Function with maximum frequency $F$
- Need to sample it at frequency $N = 2F$
- $N$ is called the ***Nyquist limit***.

$$\boxed{\text{Nyquist sampling frequency:} \quad f_s = 2 f_{\max}}$$

# The Sampling Theorem

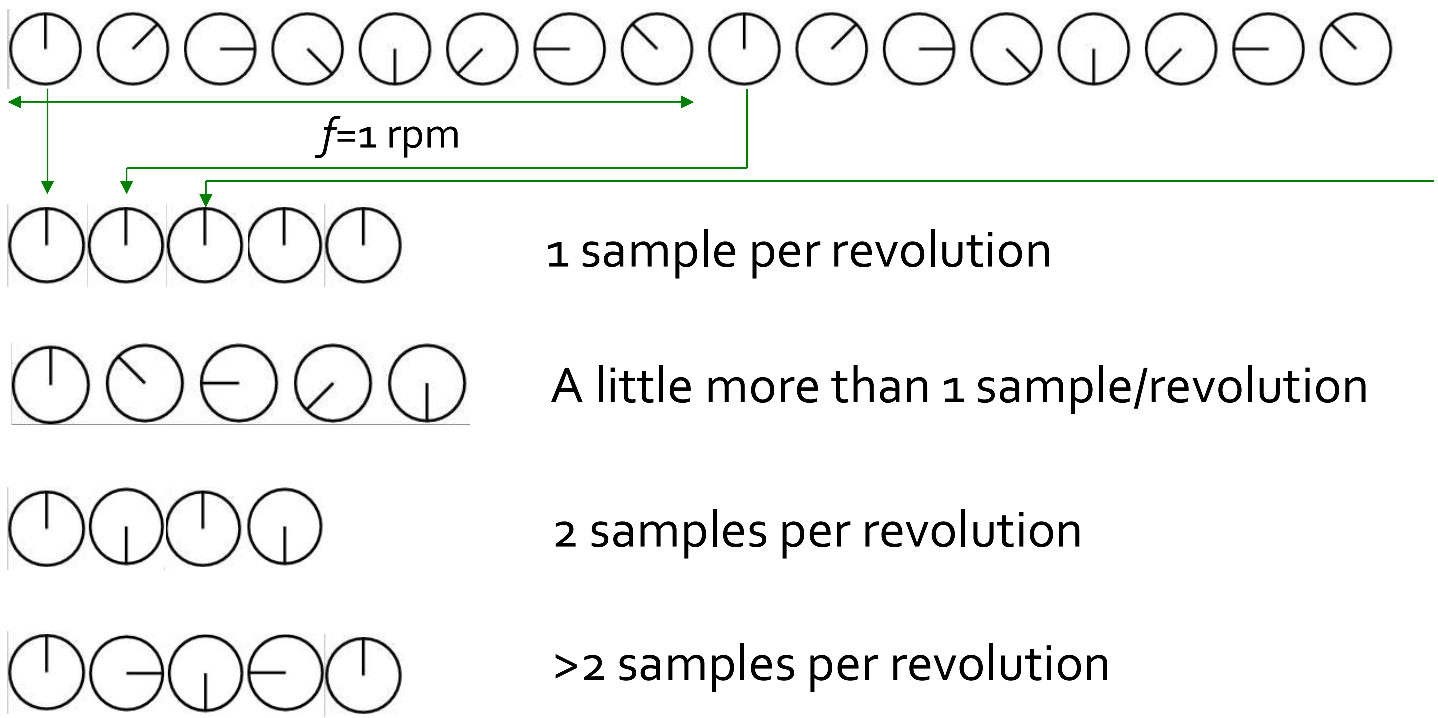


# Backwards Rotating Wheels



# Sampling in Time – Temporal Aliasing

- Wagon-wheel effect




1 sample per revolution

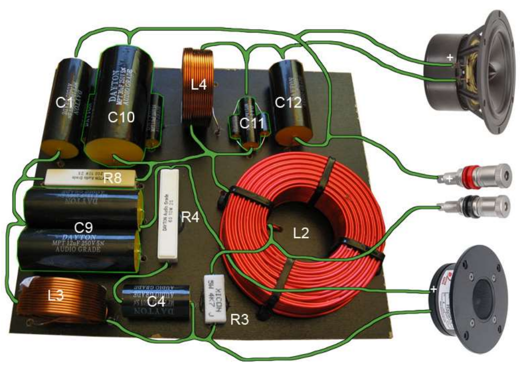
A little more than 1 sample/revolution

2 samples per revolution

>2 samples per revolution

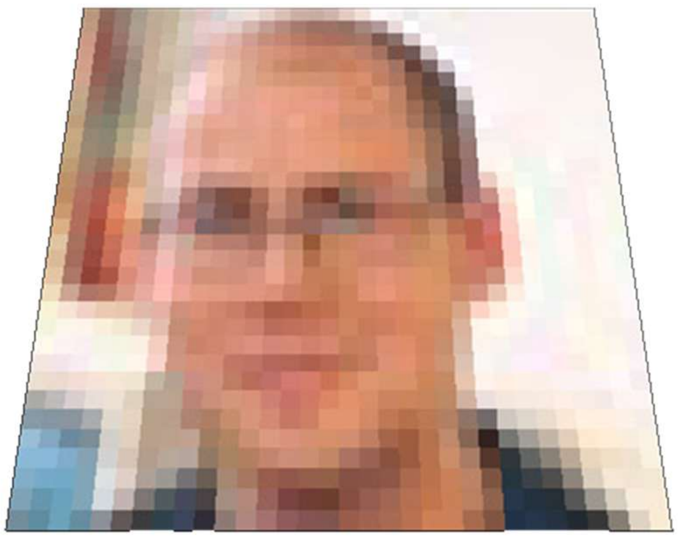
# Sampling in Time – Storing Music

- Human hears up to ~20KHz
- Need to sample with >40KHz
- Actual recording may contain frequencies above hearing range
  - Need to remove those before sampling, otherwise aliasing possible → low pass filter
  - CD:44KHz – filter save margin
- For reproduction continuous wave needed
- Loudspeaker driver may have only certain range
  - Need band-pass filter to avoid aliasing (crossover)

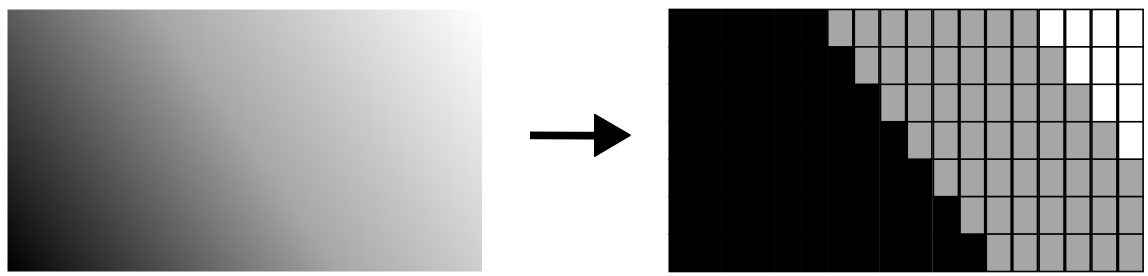


# Aliasing in Computer Graphics

# Aliasing from too Bad Resolution



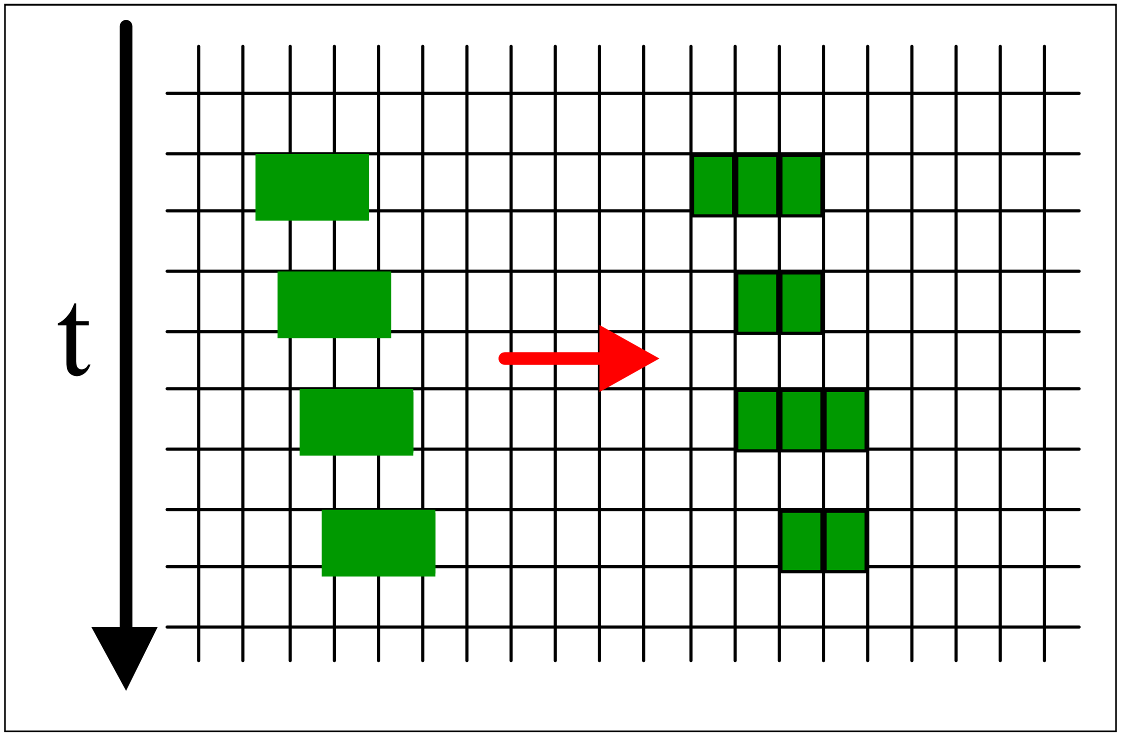
# Aliasing from too Few Colors



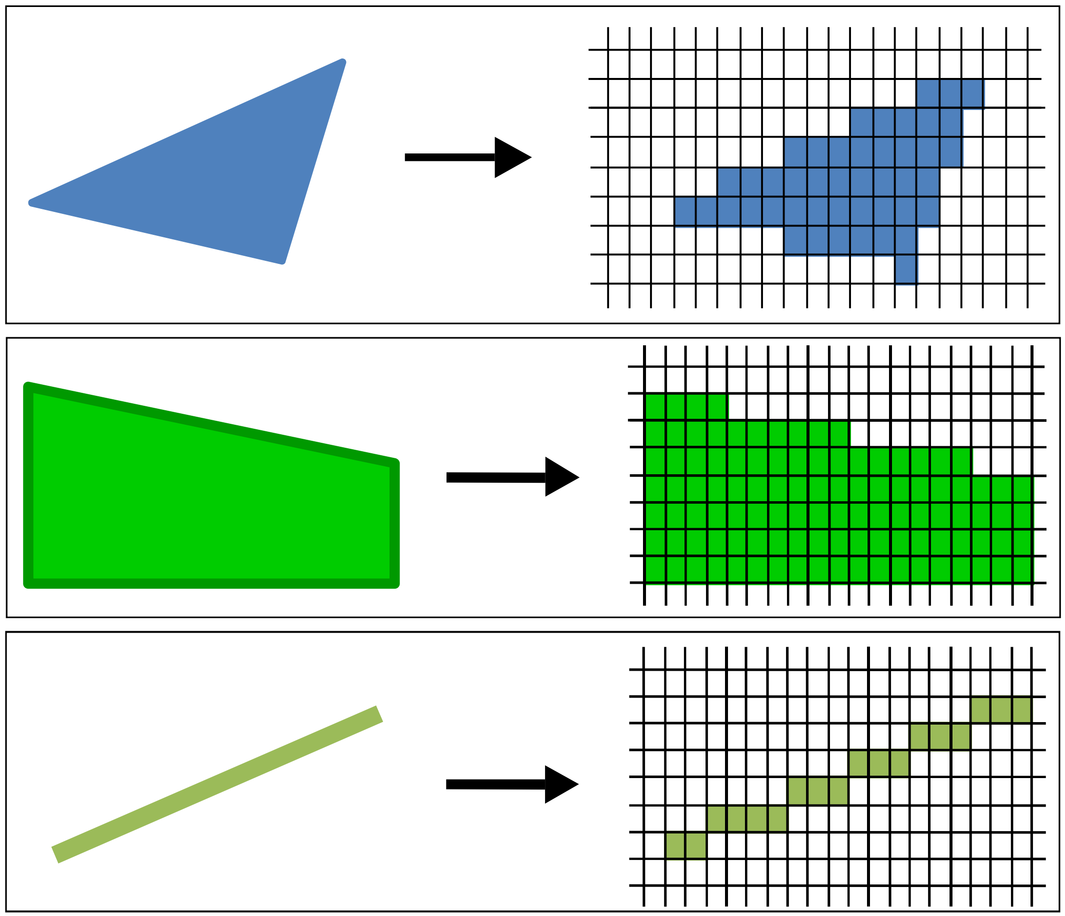
artificial color borders can appear

# Aliasing in Animations

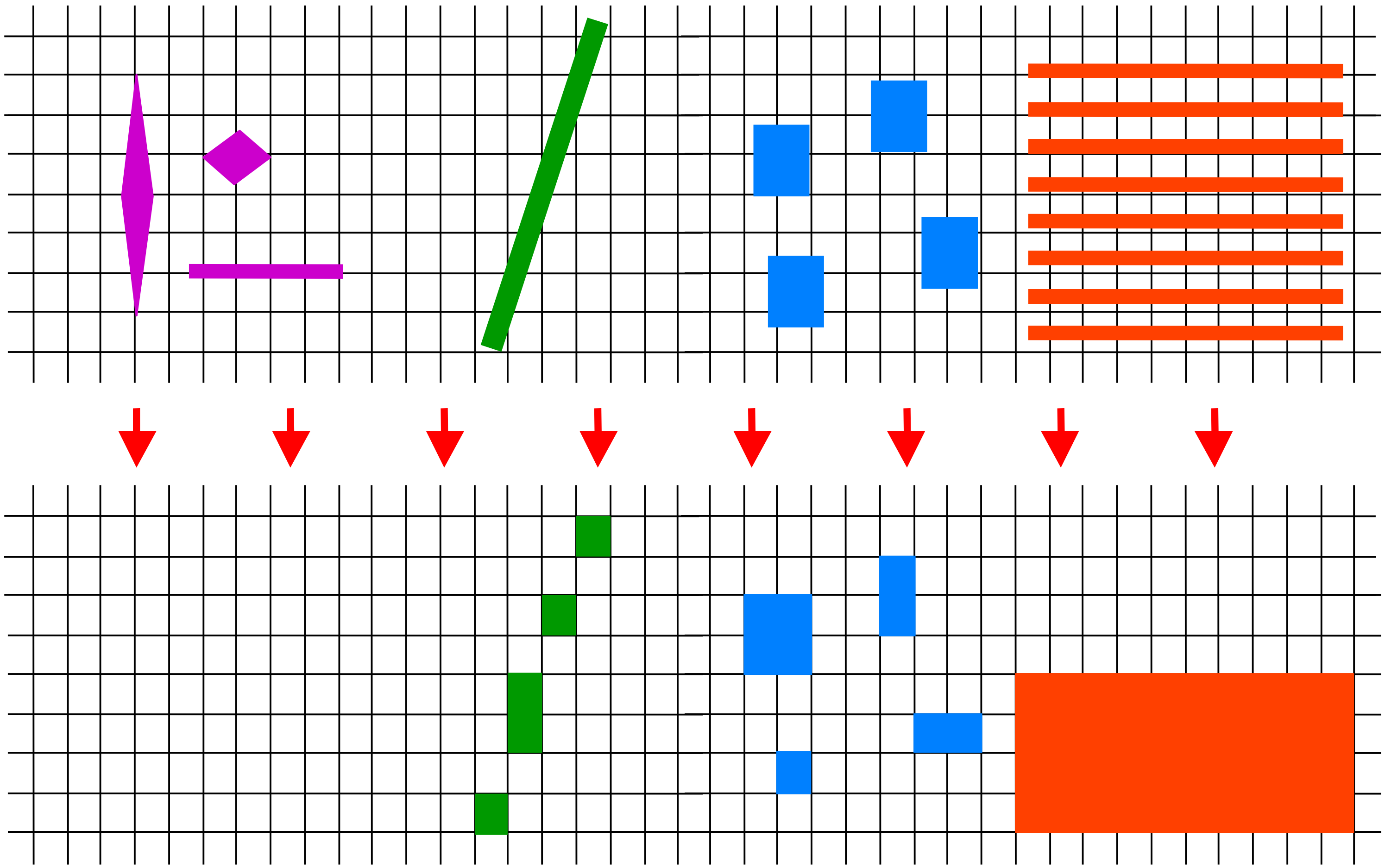
- Jumping images
- “worming”



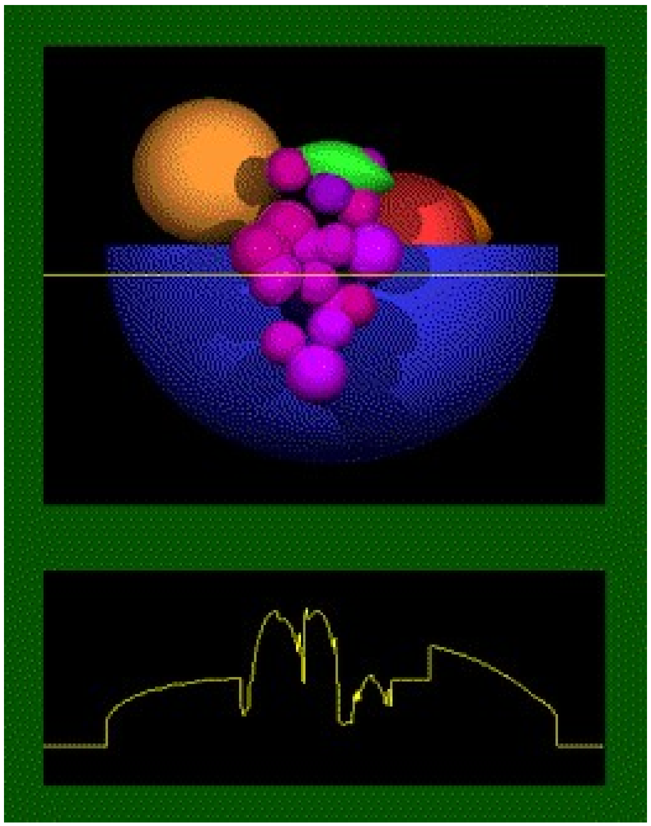
# Aliasing from Geometric Errors



# Aliasing from Geometric Errors



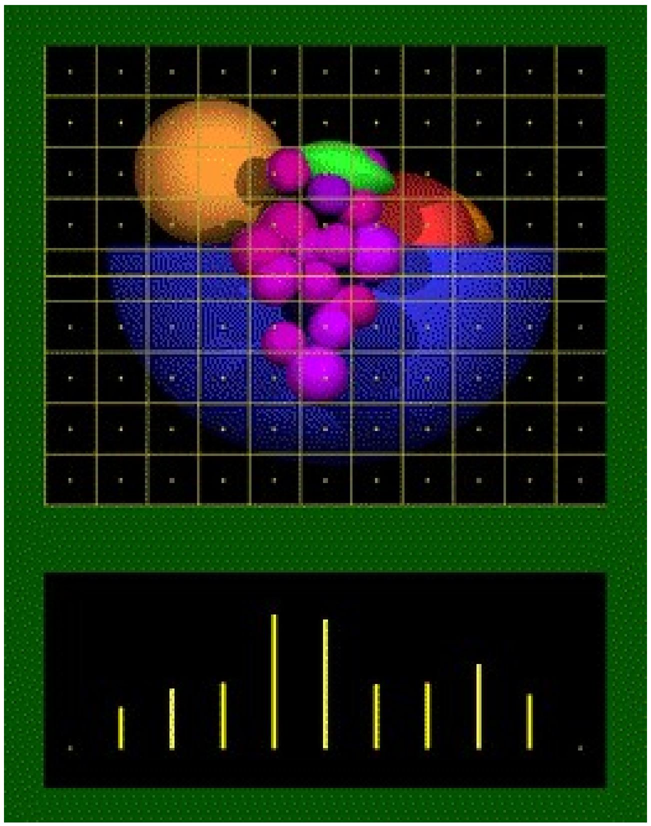
# Aliasing and Point Samping



Original Scene

Luminosity

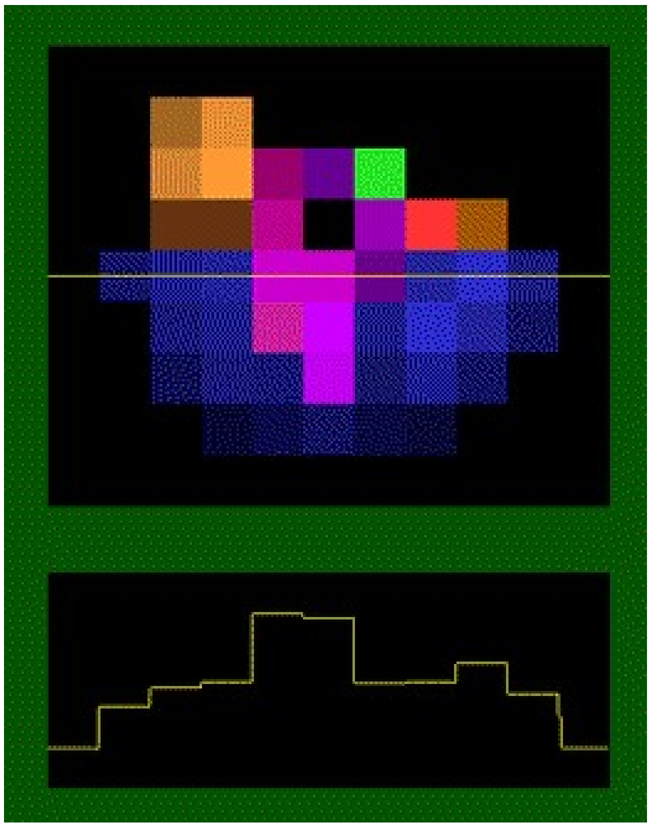
# Aliasing and Point Samping



Pixel Sampling

Samples

# Aliasing and Point Samping



Displayed Image

Luminosity

# Anti-Aliasing

# Solutions against Aliasing?

- **1. improve the devices**
  - Higher resolution
  - More color levels
  - Faster image sequence

expensive
or
incompatible

- **2. improve the images = *anti-aliasing***
  - Post-processing
  - Pre-filtering !

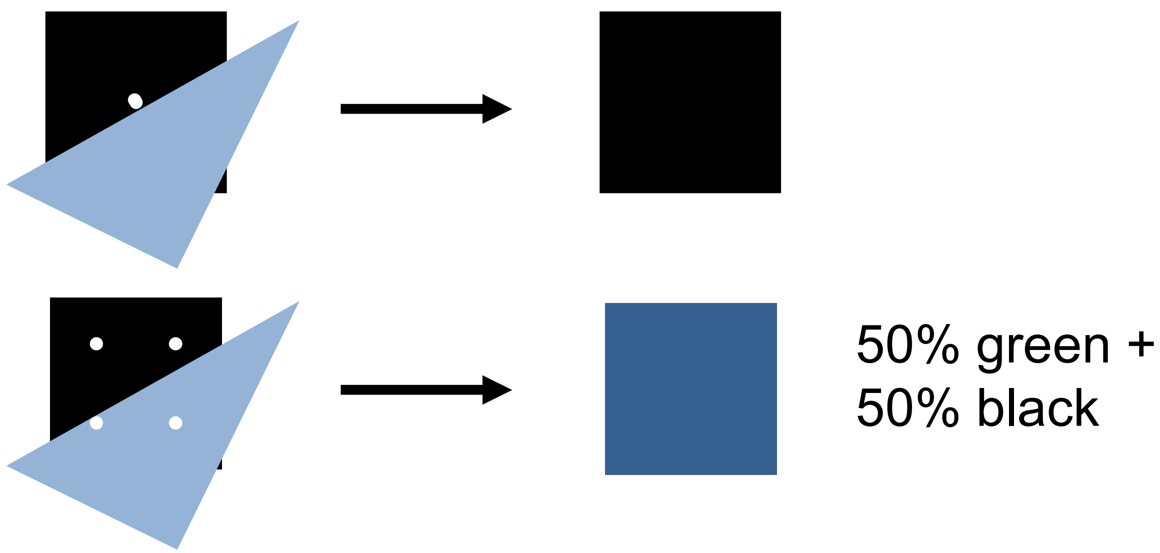
software

# Anti-Aliasing

More Samples

# Screen-based Anti-Aliasing

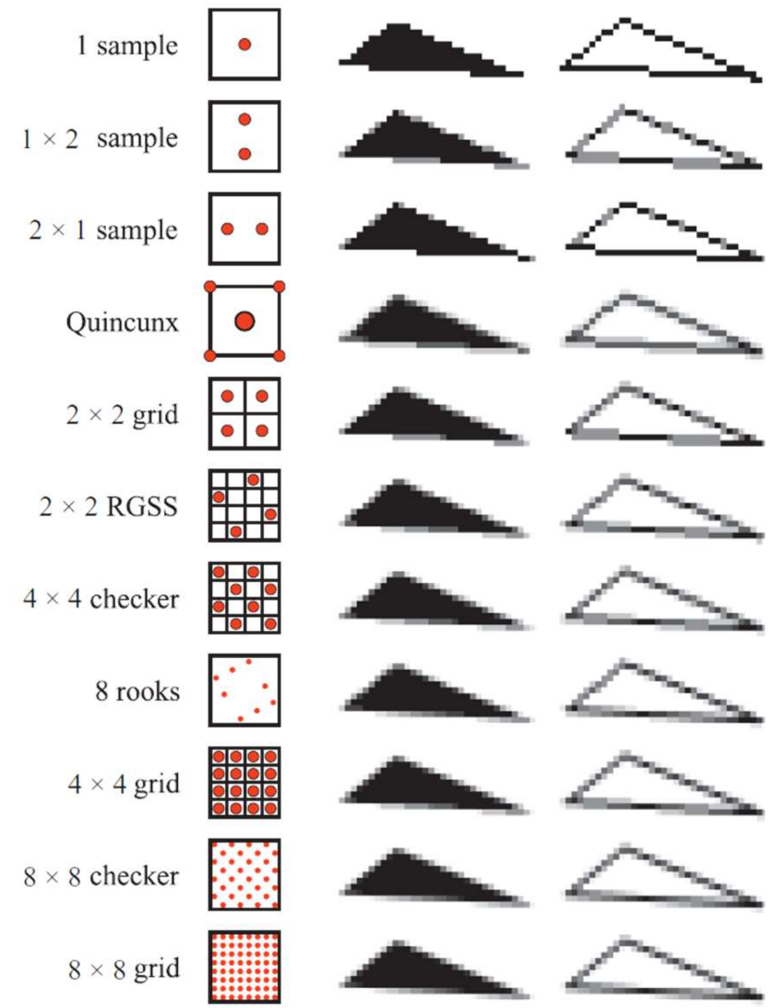
- Hard case: edge has infinite frequency
- Supersampling: use more than one sample per pixel and average
- More samples → better results → more work



# Supersampling: Different Schemes

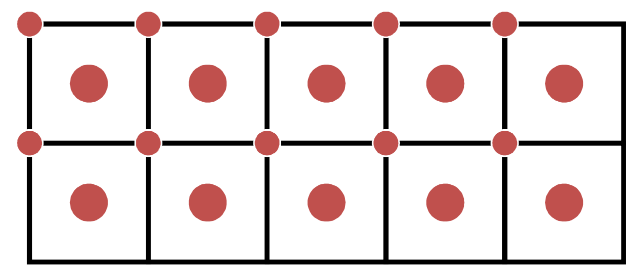
$$\mathbf{p}(x, y) = \sum_{i=1}^{n} w_i \mathbf{c}(i, x, y)$$

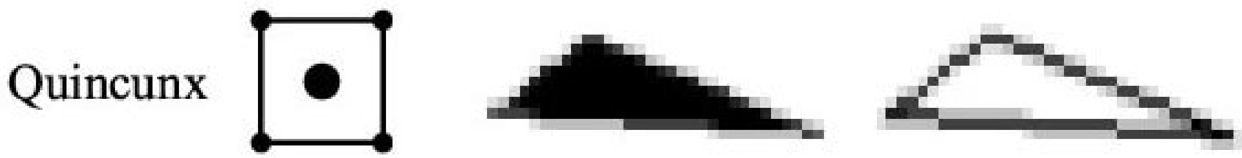
- $w_i$ are the weights in [0,1]
- $n$ is the number of samples taken per pixel
- $\mathbf{c}(i,x,y)$ is the color of sample $i$ inside pixel



# Quincunx

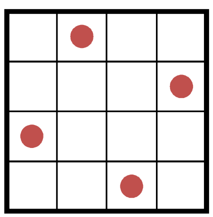
- Sharing of corner samples
- w1=0.5, w2=0.125, w3=0.125, w4=0.125, w5=0.125
- All corner samples have same weight
- Is available since NVIDIA GeForce3 in HW

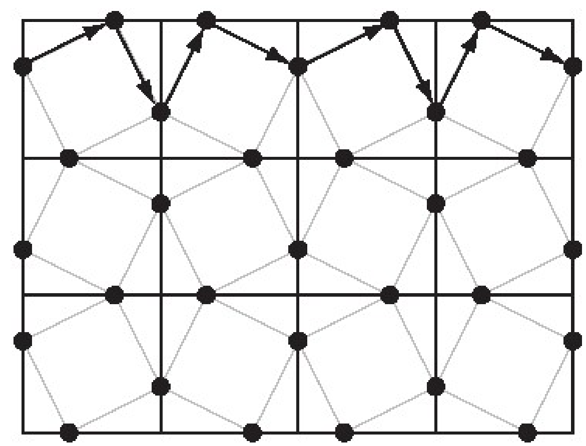




# FLIPQUAD

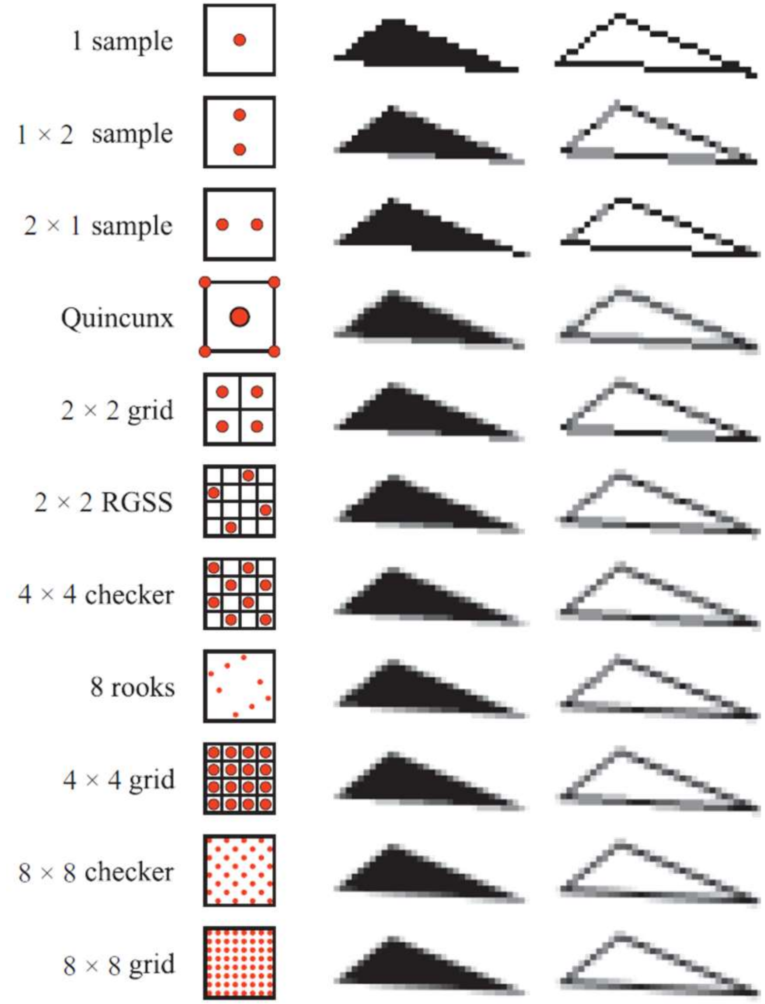
- Weights: 0.25 per sample
- Performs better than Quincunx



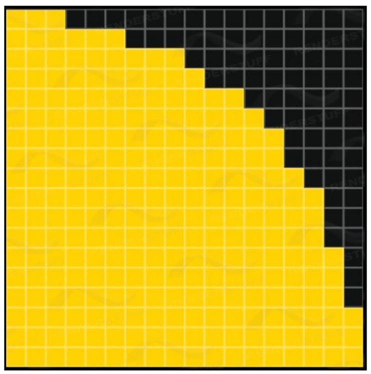
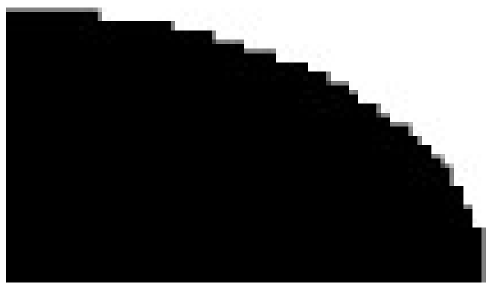
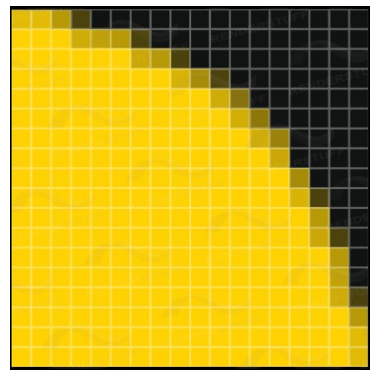
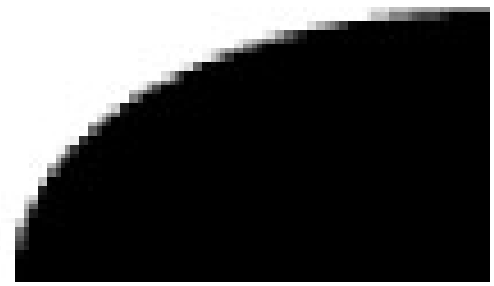


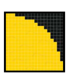
# Screen-based Anti-Aliasing

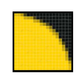
- Use more samples per pixel
  - Render at higher resolution
    - **Storage** goes up quadratically
  - Apply sampling pattern
  - Weight and sum
- Shifts the Nyquist limit higher
  - **Doesn't eliminate aliasing!**
- Operate only on output samples of pipeline
- No knowledge of objects being rendered
- A.k.a. post-filtering



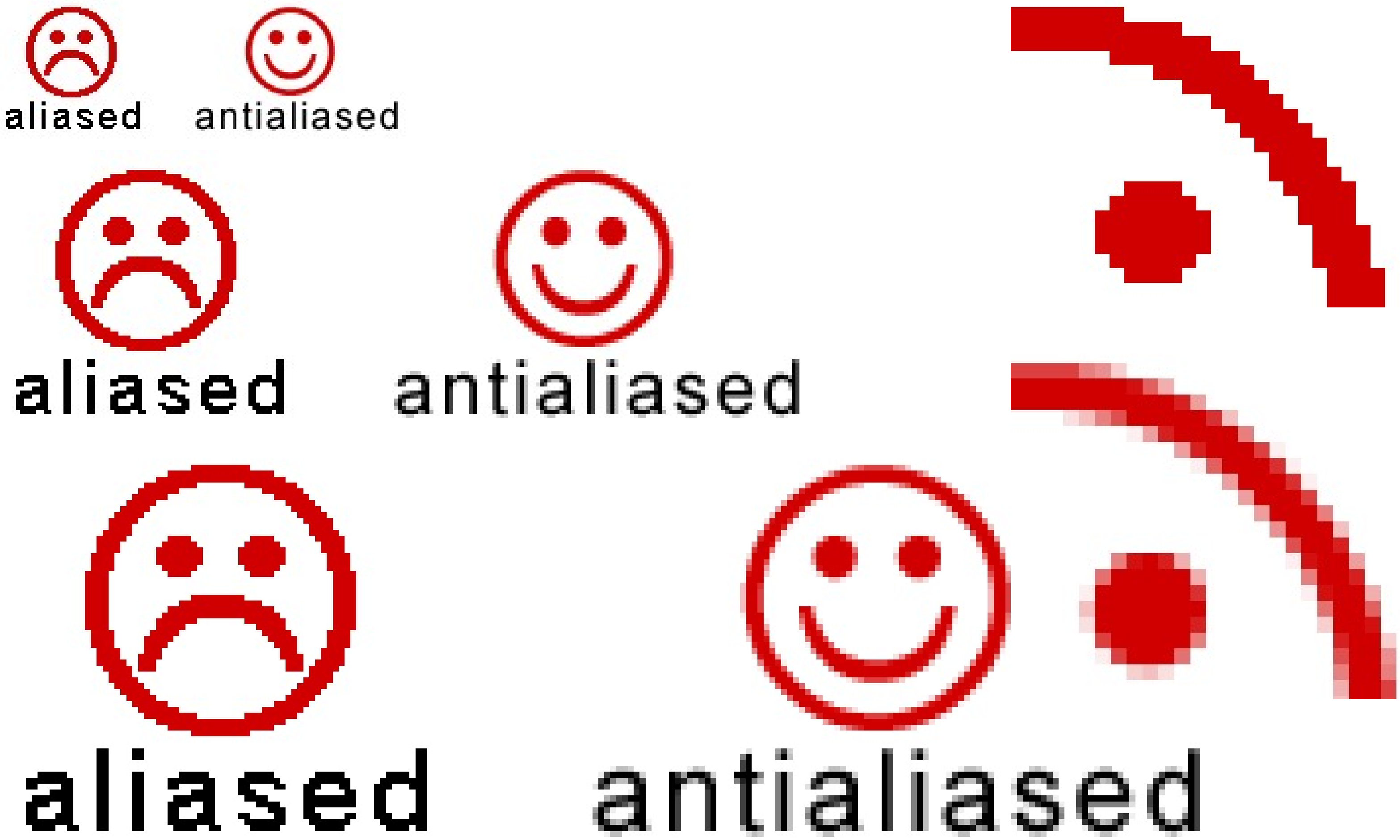
# Anti-Aliasing Examples

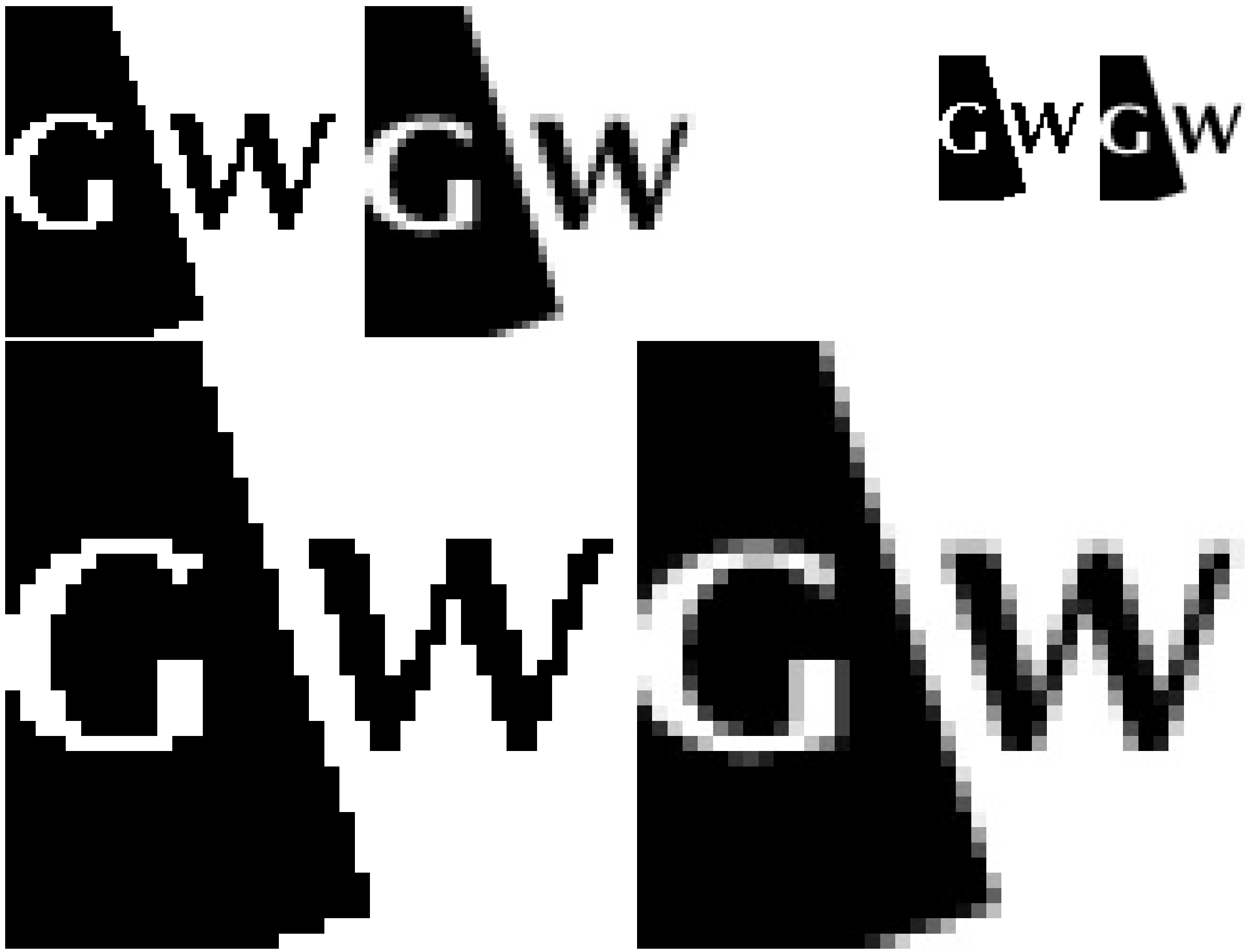





# Anti-Aliasing Examples

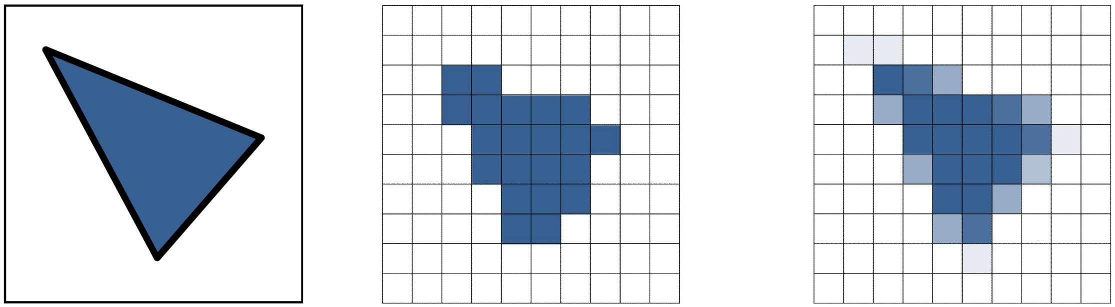


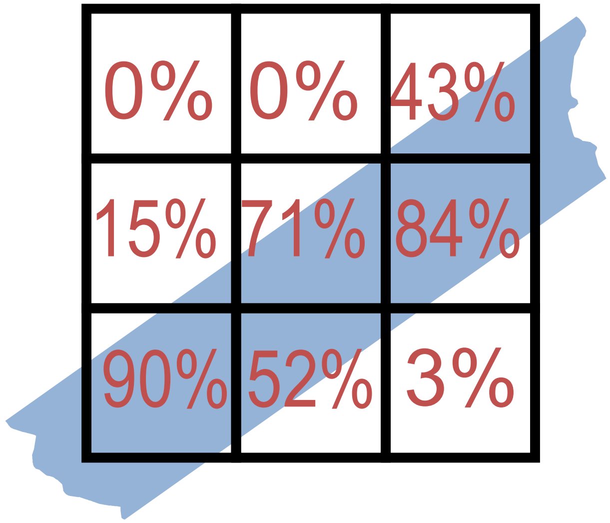
# Anti-Aliasing Examples



# Area Sampling

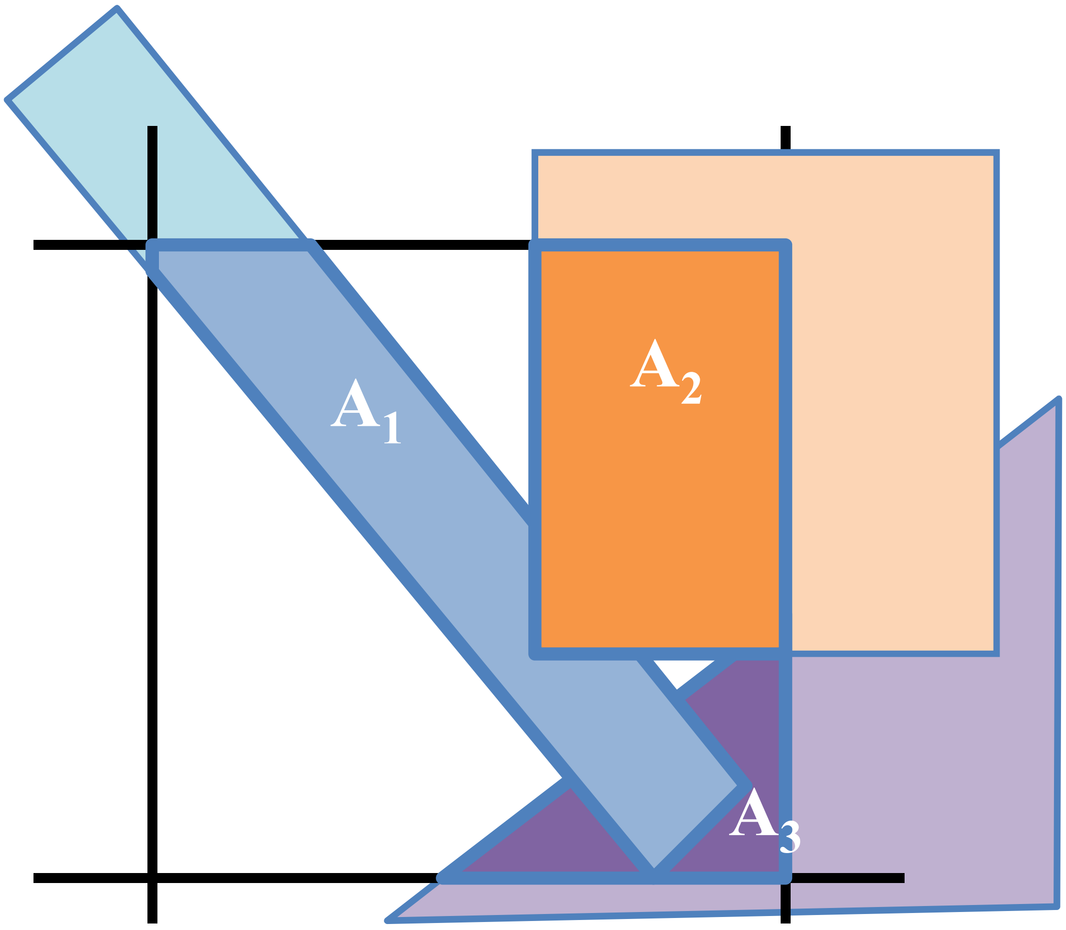
- Calculate the pixel coverage exactly
- Can be done with incremental schemes





# Catmull's Algorithm

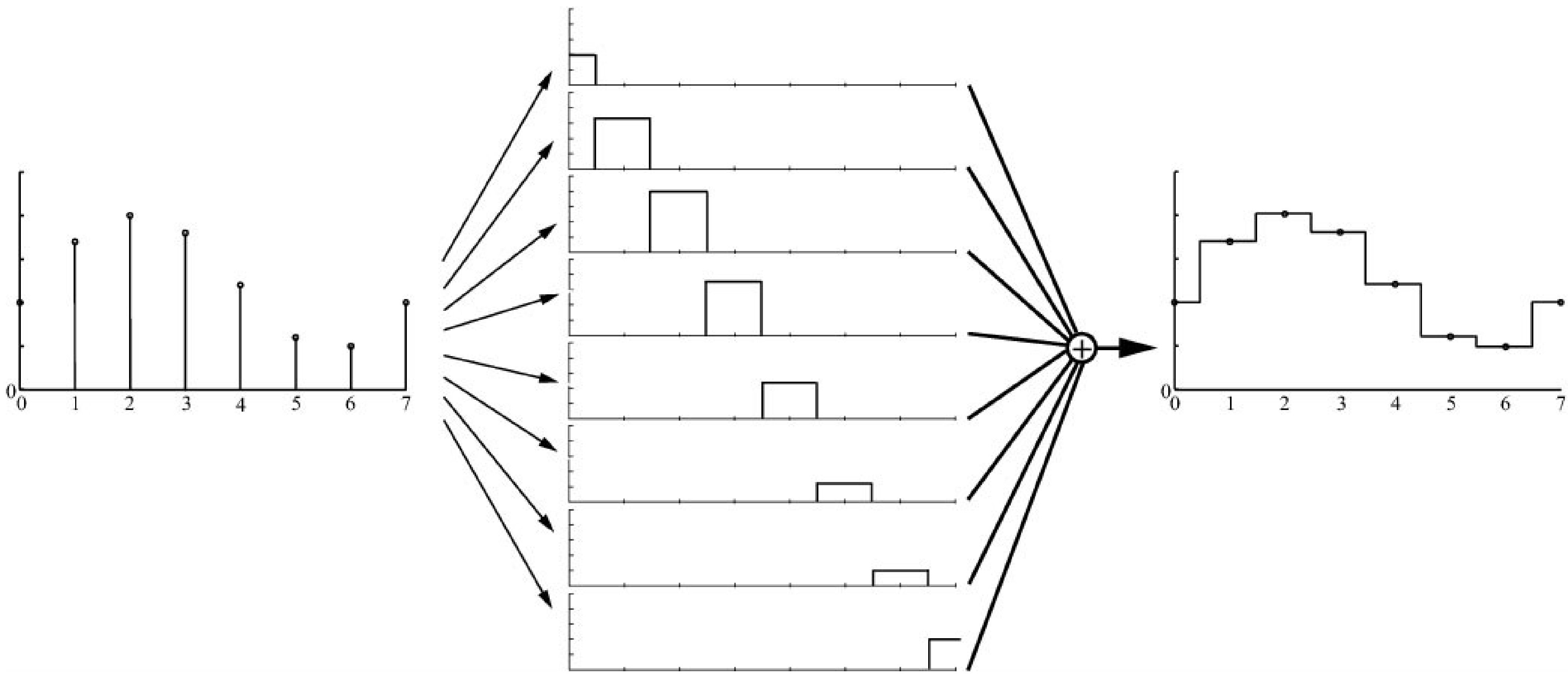
- Find fragment areas (Visibility!)
- Multiply by fragment colors
- Sum for final pixel color



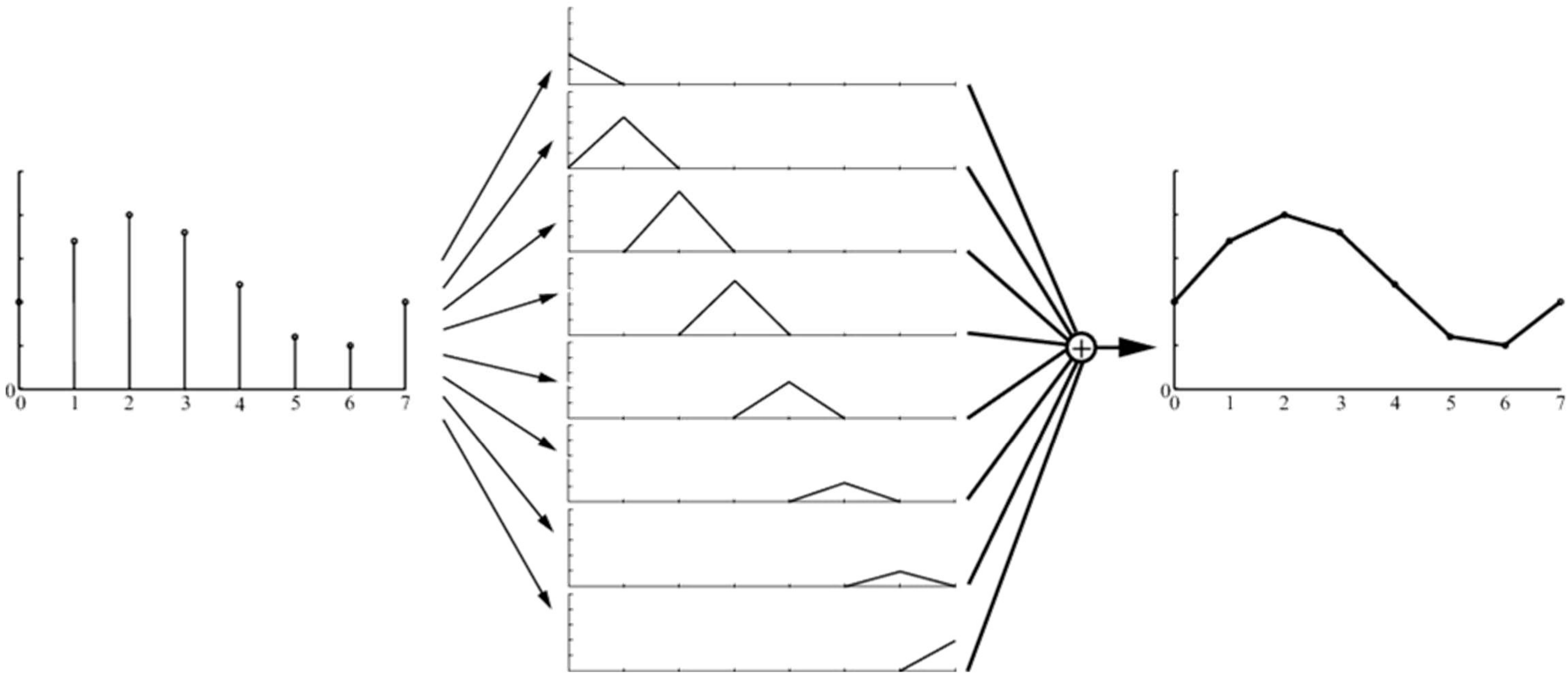
# Anti-Aliasing

Better Reconstruction

# Box Filter – Nearest Neighbor

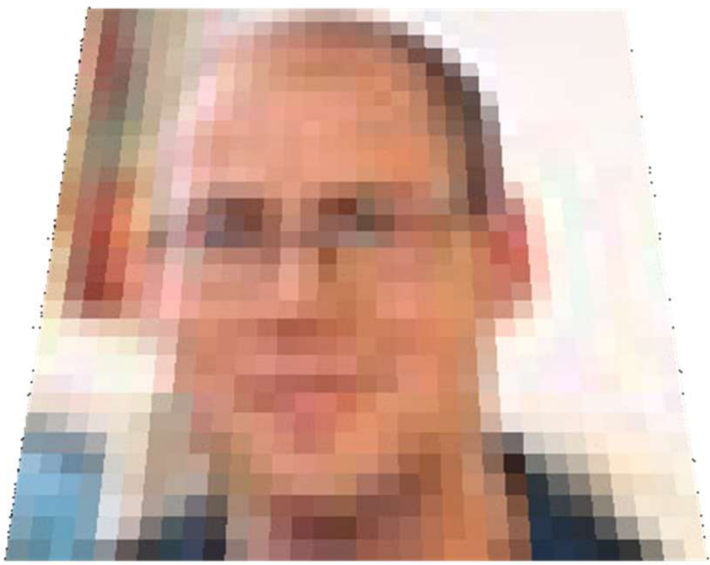


# Tent Filter – Linear Interpolation

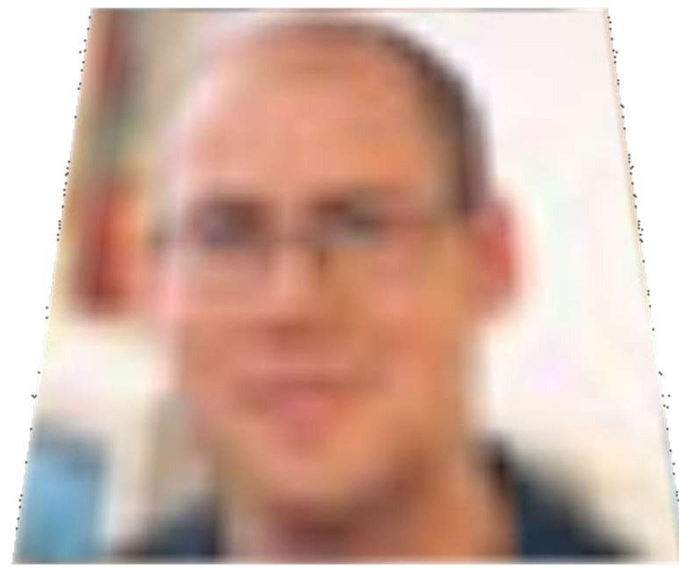


# Reconstruction

Nearest neighbour (box)

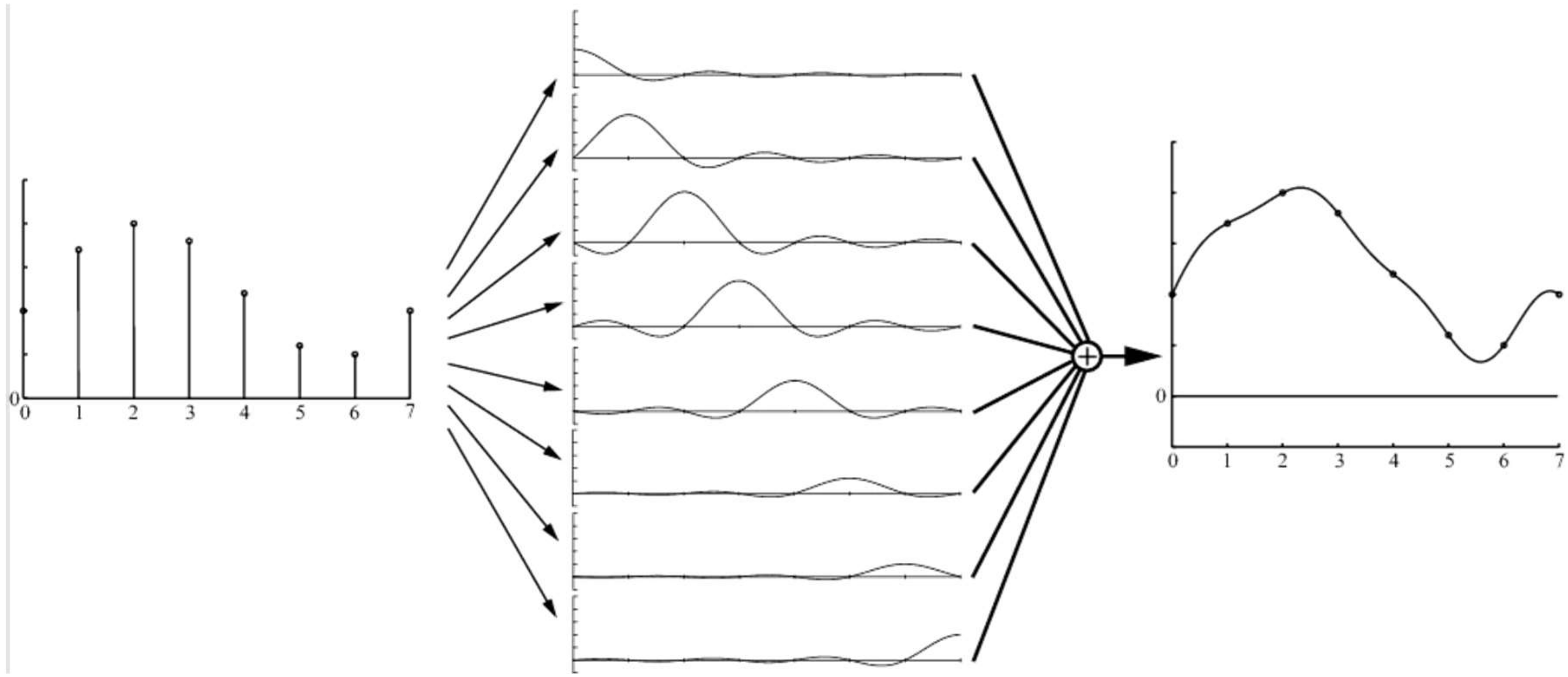


Linear (tent)



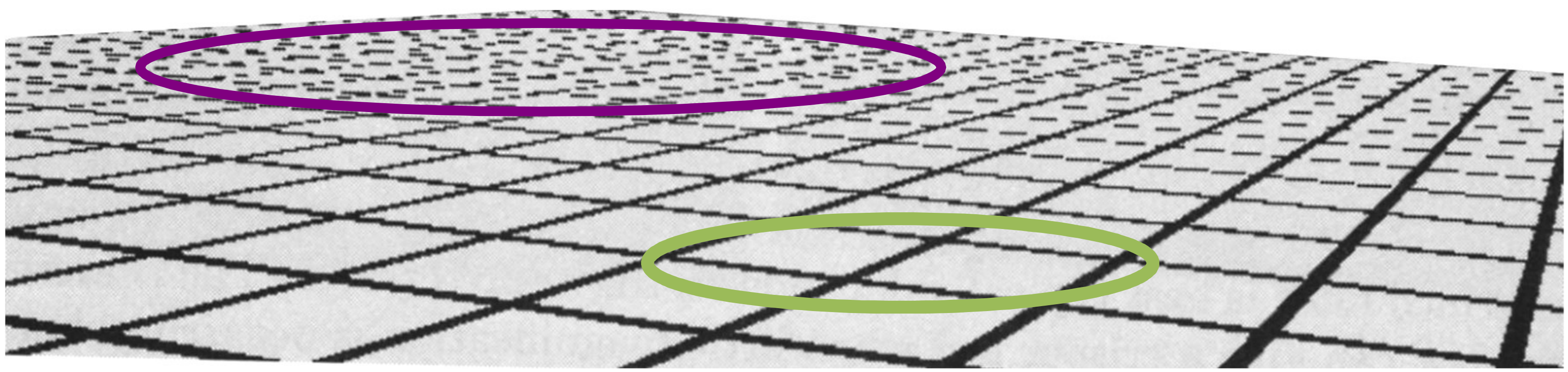
# Reconstruction with Sinc Filter

- In theory, the ideal filter
- Not practical (infinite extension, negative)
- Practical version use filtering window



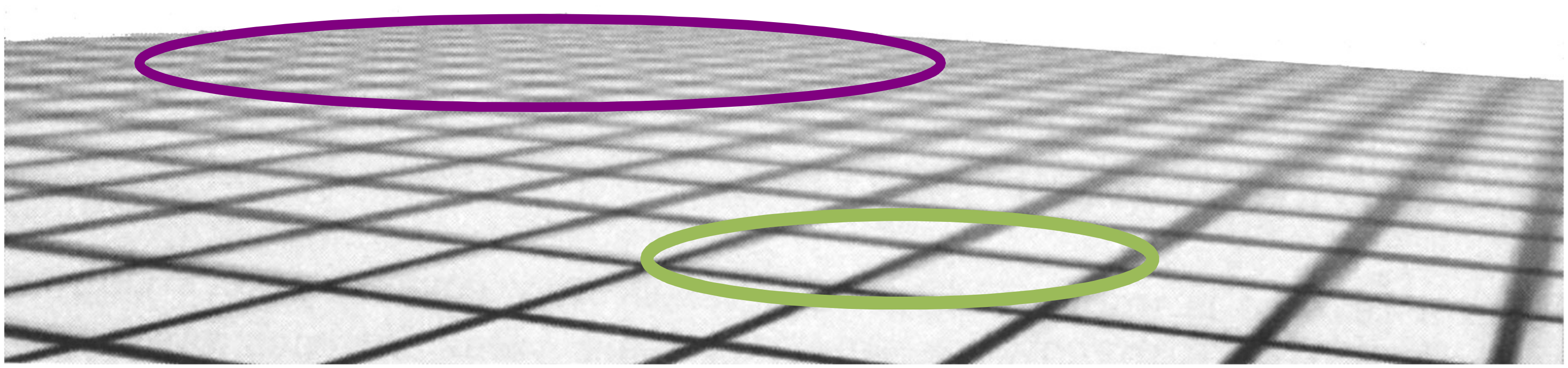
# Antialiasing and Texture Mapping

- Texture mapping is uniquely harder
  - Coherent textures present pathological artifacts
    - Magnification
    - Minification
  - Correct filter shape changes



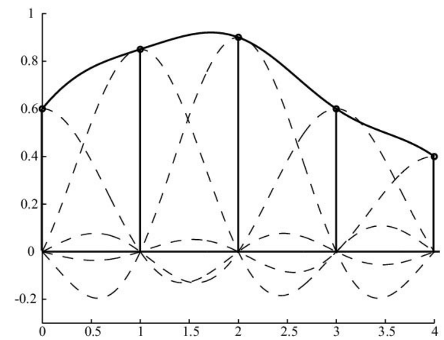
# Antialiasing and Texture Mapping

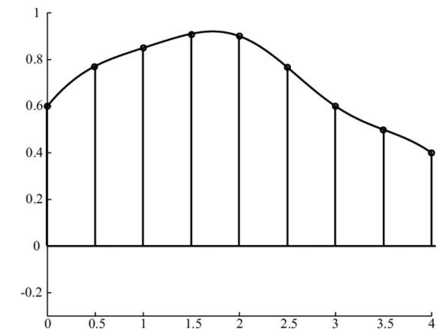
- Texture mapping is uniquely easier
  - Textures are known ahead of time
  - They can thus be prefiltered



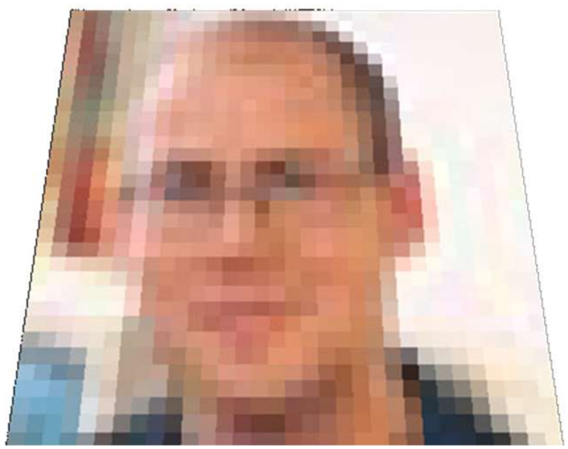
# Texture Magnification

- What does the theory say…



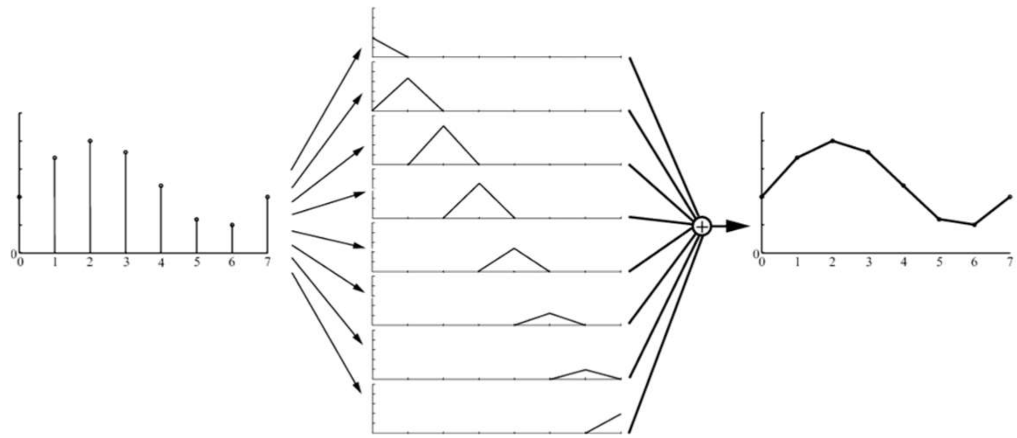


- sinc($x$) is not feasible in real time
- Box filter (nearest-neighbour)
  - Poor quality

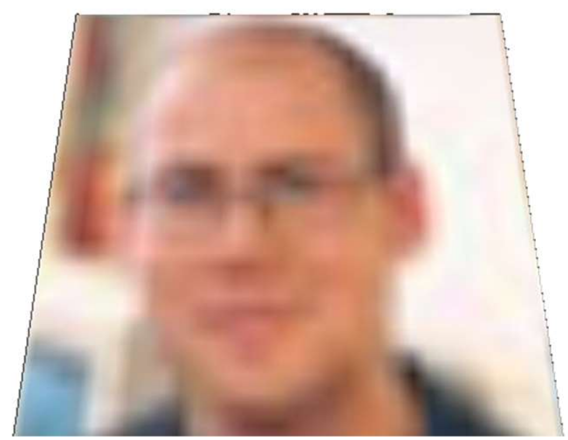


# Texture Magnification

- Tent filter is feasible!
- Linear interpolation

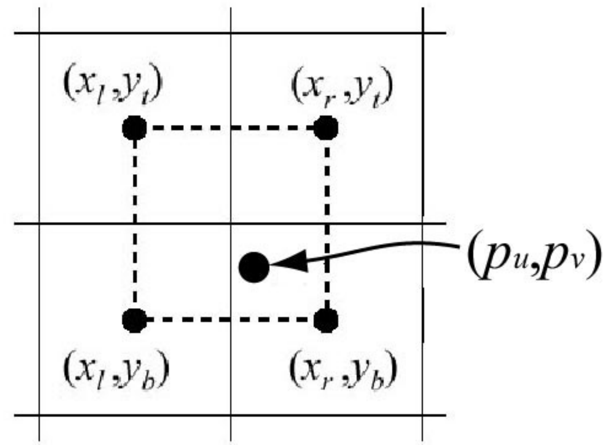


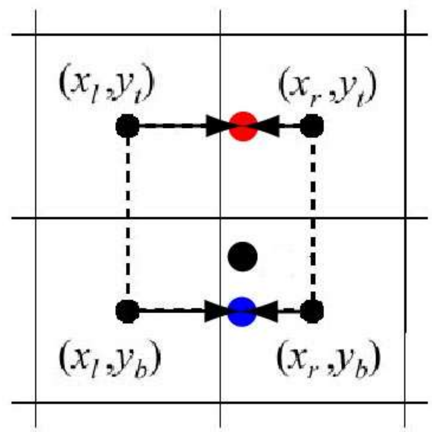
- Simple in 1D
  - (1-t)*color0+t*color1
- How about 2D?

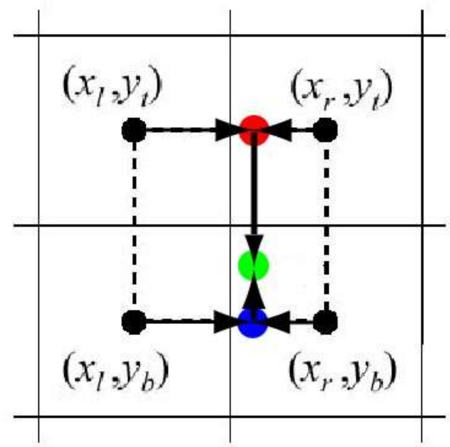


# Bilinear Interpolation

- Texture coordinates $(p_u,p_v)$ in [0,1]
- Texture image size: n*m texels
- Nearest neighbour would access: ( floor(n*u), floor(m*v) )
- Interpolate 1D in x & y

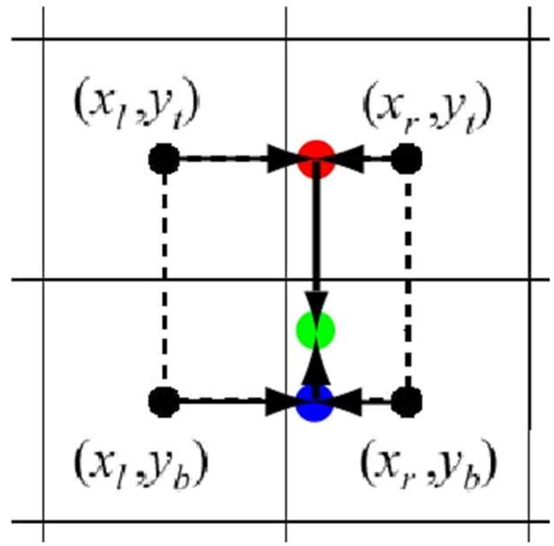


# Bilinear Interpolation

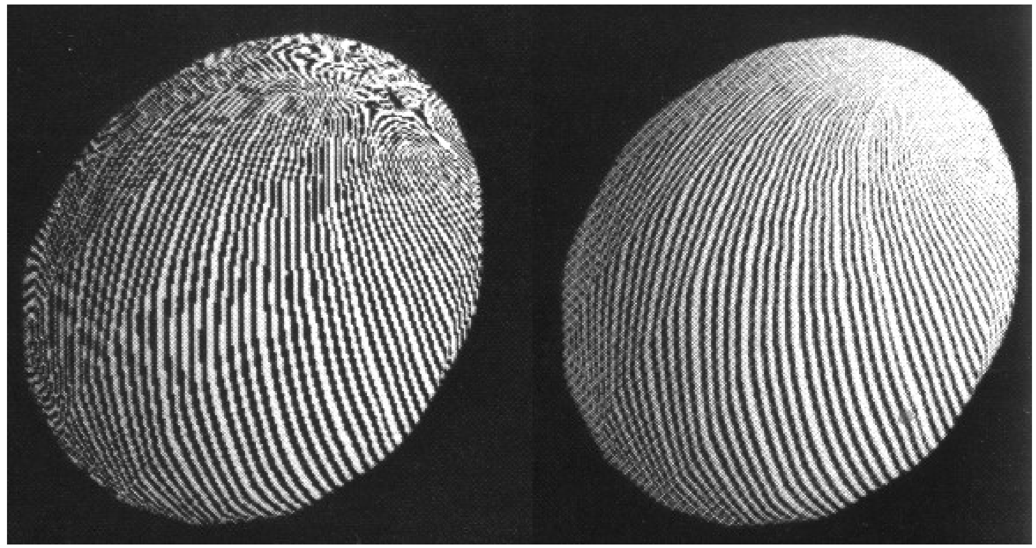
- $\mathbf{t}(u,v)$ accesses the texture map
- $\mathbf{b}(u,v)$ filtered texel

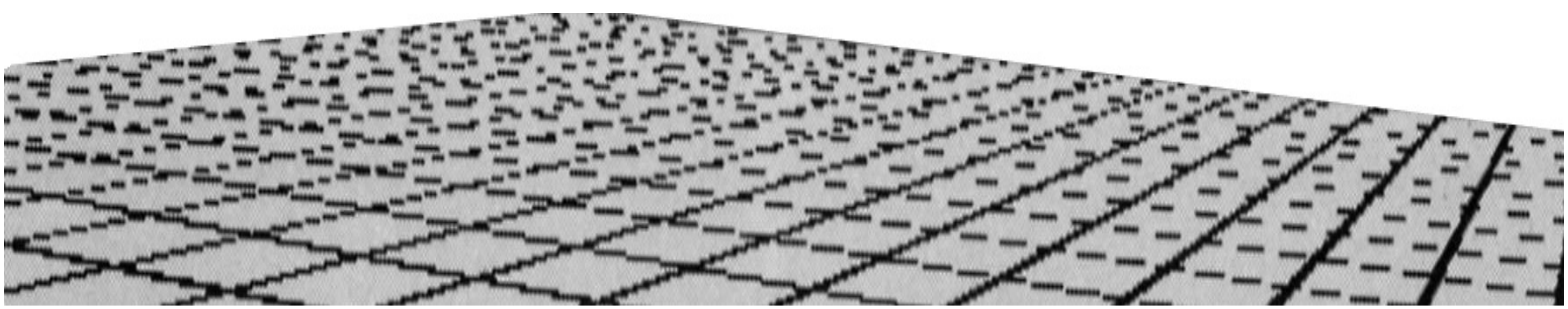



$$(u', v') = (p_u - \lfloor p_u \rfloor, p_v - \lfloor p_v \rfloor).$$

$$\begin{aligned}\mathbf{b}(p_u, p_v) =& (1-u')(1-v')\mathbf{t}(x_l, y_b) + u'(1-v')\mathbf{t}(x_r, y_b) \\ &+ (1-u')v'\mathbf{t}(x_l, y_t) + u'v'\mathbf{t}(x_r, y_t).\end{aligned}$$

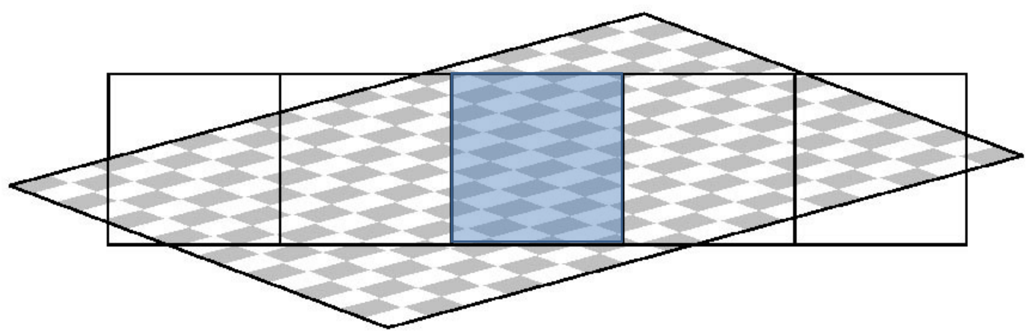
# Texture Minification





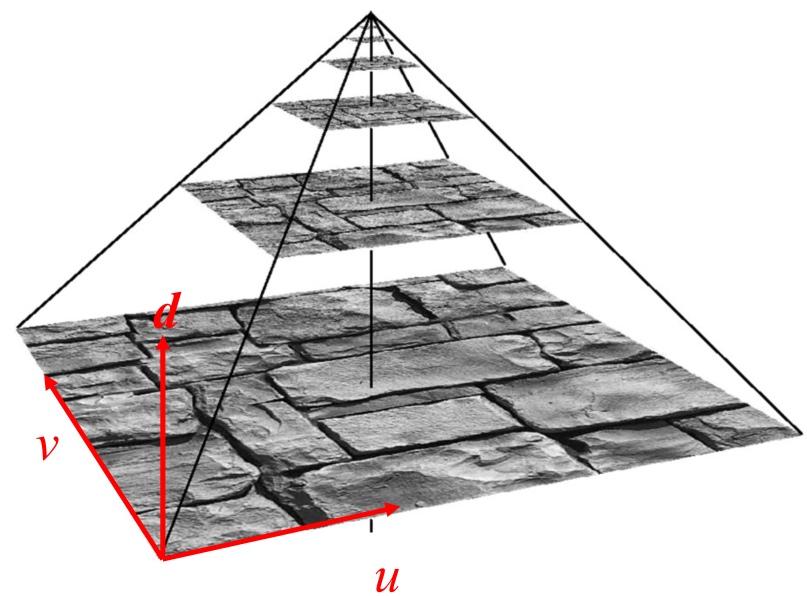
# Texture Minification

- What does a pixel ”see”?
- Theory (*sinc*) is too expensive
- Cheaper: average of texels inside a pixel
  - Still expensive
- MIP-maps – another level of approximation
  - Pre-filter texture maps…



# MIP-Mapping (Multum In Parvum)

- Image pyramid
- Half width and height when going upwards
- Average over 4 ”parent texels” to form ”child texel”

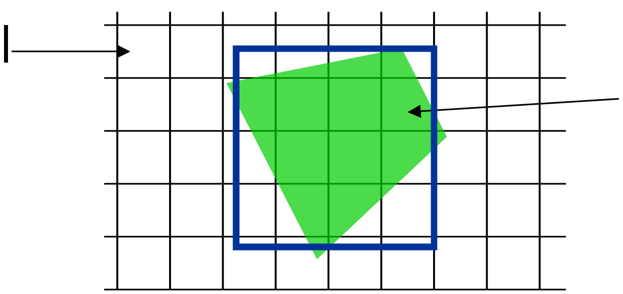



- Depending on amount of minification, determine which image to fetch from
- How do compute $d$ ?

# Computing *d* for MIP-Mapping

- Approximate quad with square
- Gives overblur!

texel



screen pixel projected
to texture space is quadrilateral

$A = \text{approximative area of quadrilateral}$

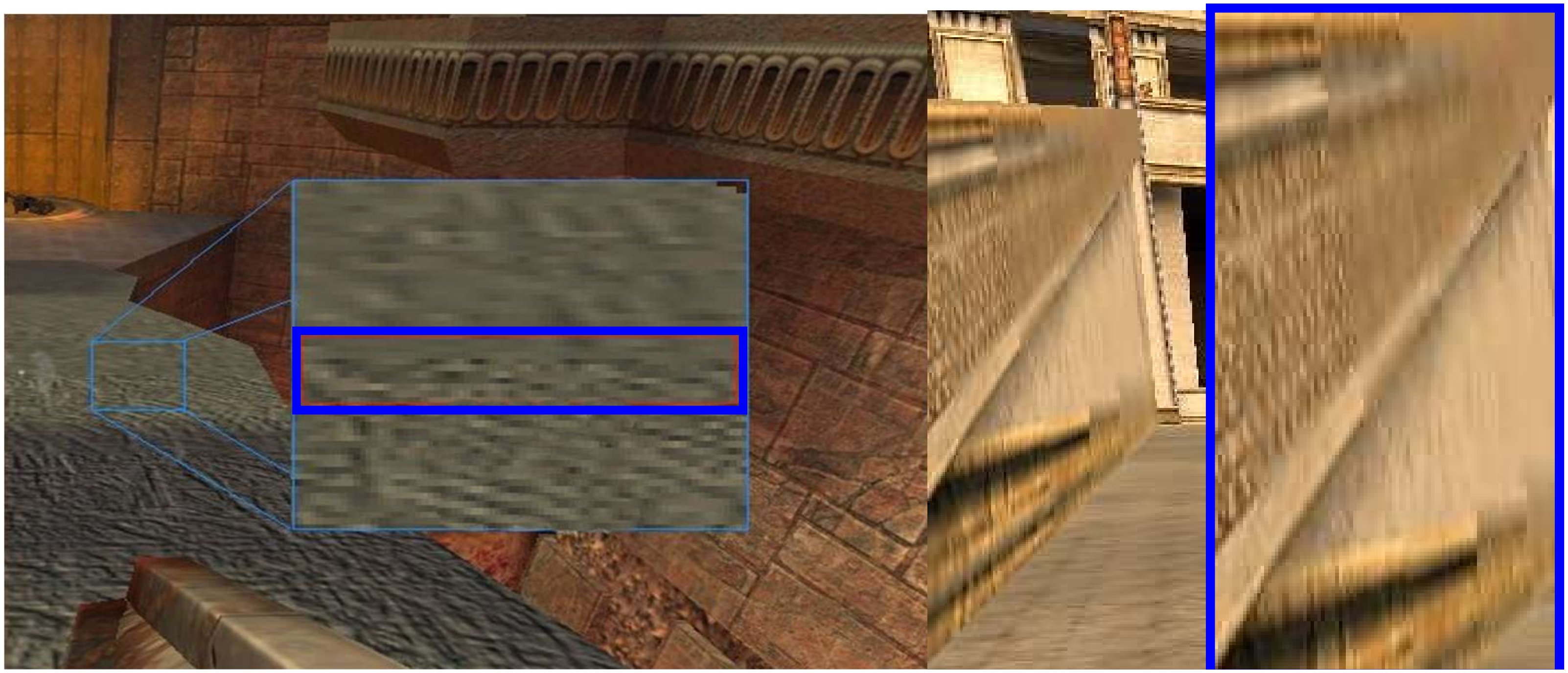
$b = \sqrt{A}$

$d = \log_2 b$

# MIP-Mapping

- Take value from texture $d_0 = trunc(d)$
  - Use nearest mip map texture
  - Use *bilinear interpolation*

*"mip-map level"*

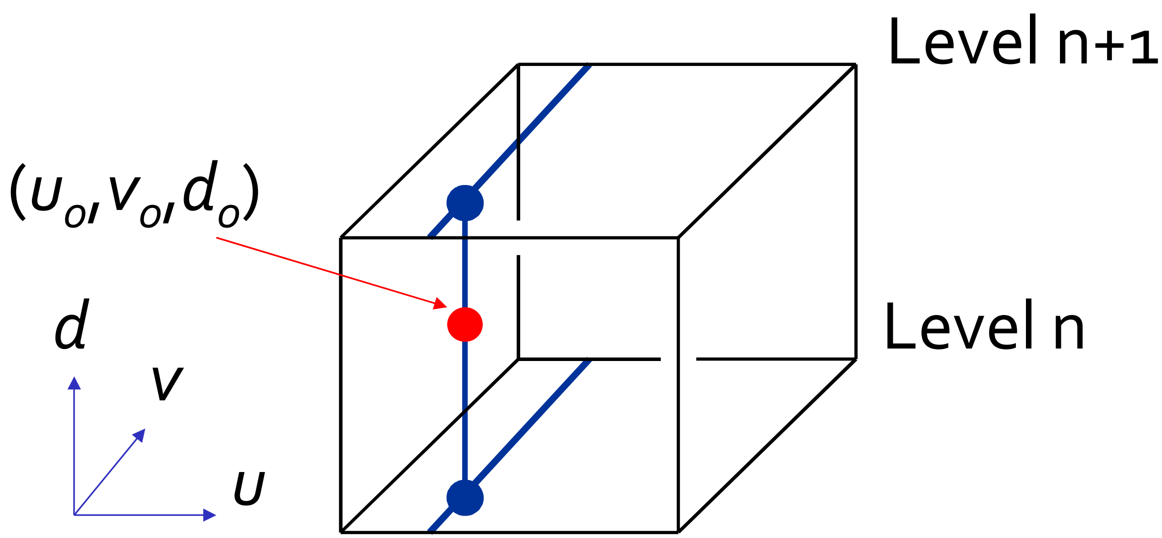


# MIP-Mapping

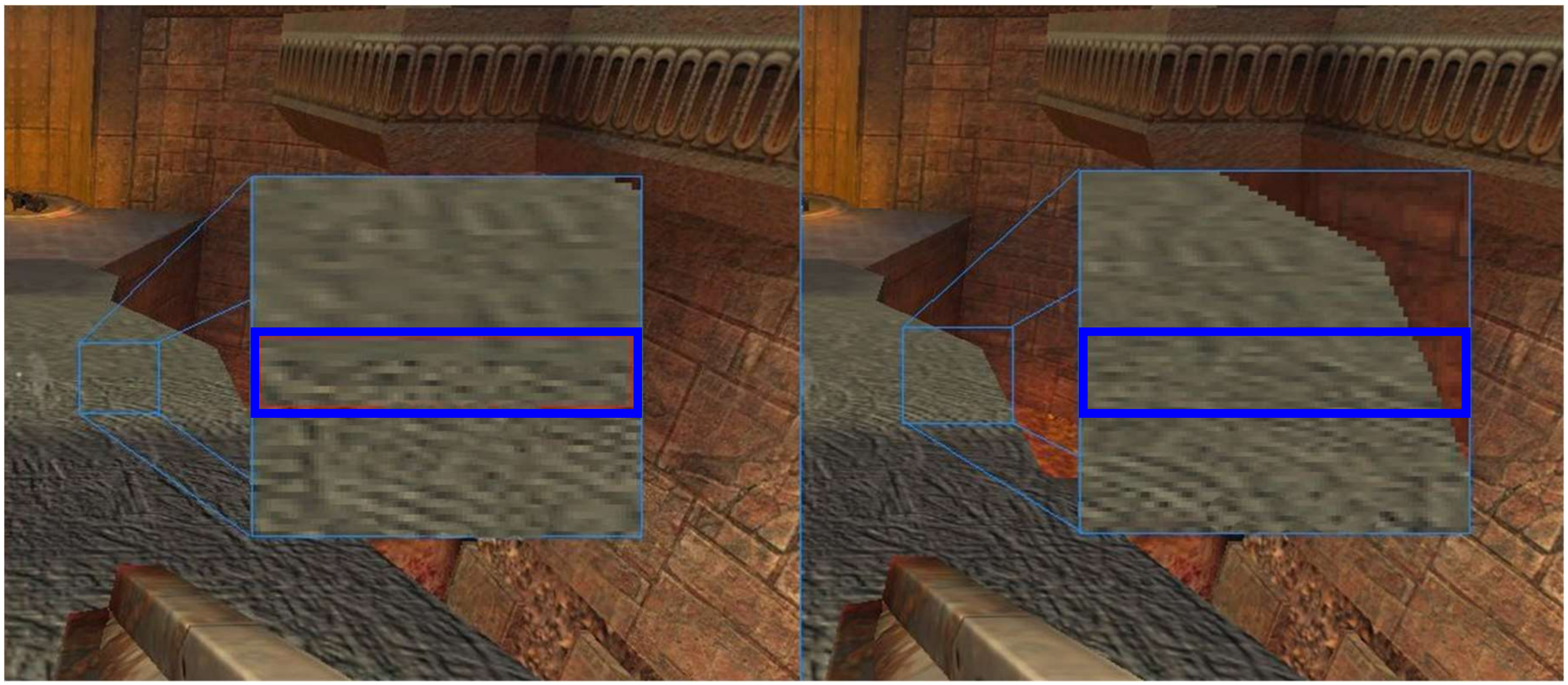
- Trilinear interpolation:
  - Linear interpolation between successive mip-maps
  - Avoids “mip-banding" (but doubles texture lookups)

# MIP-Mapping

- Bilinear interpolation in each of the images
- Interpolate between those bilinear values
  - Gives trilinear interpolation
- Constant time filtering: 8 texel accesses

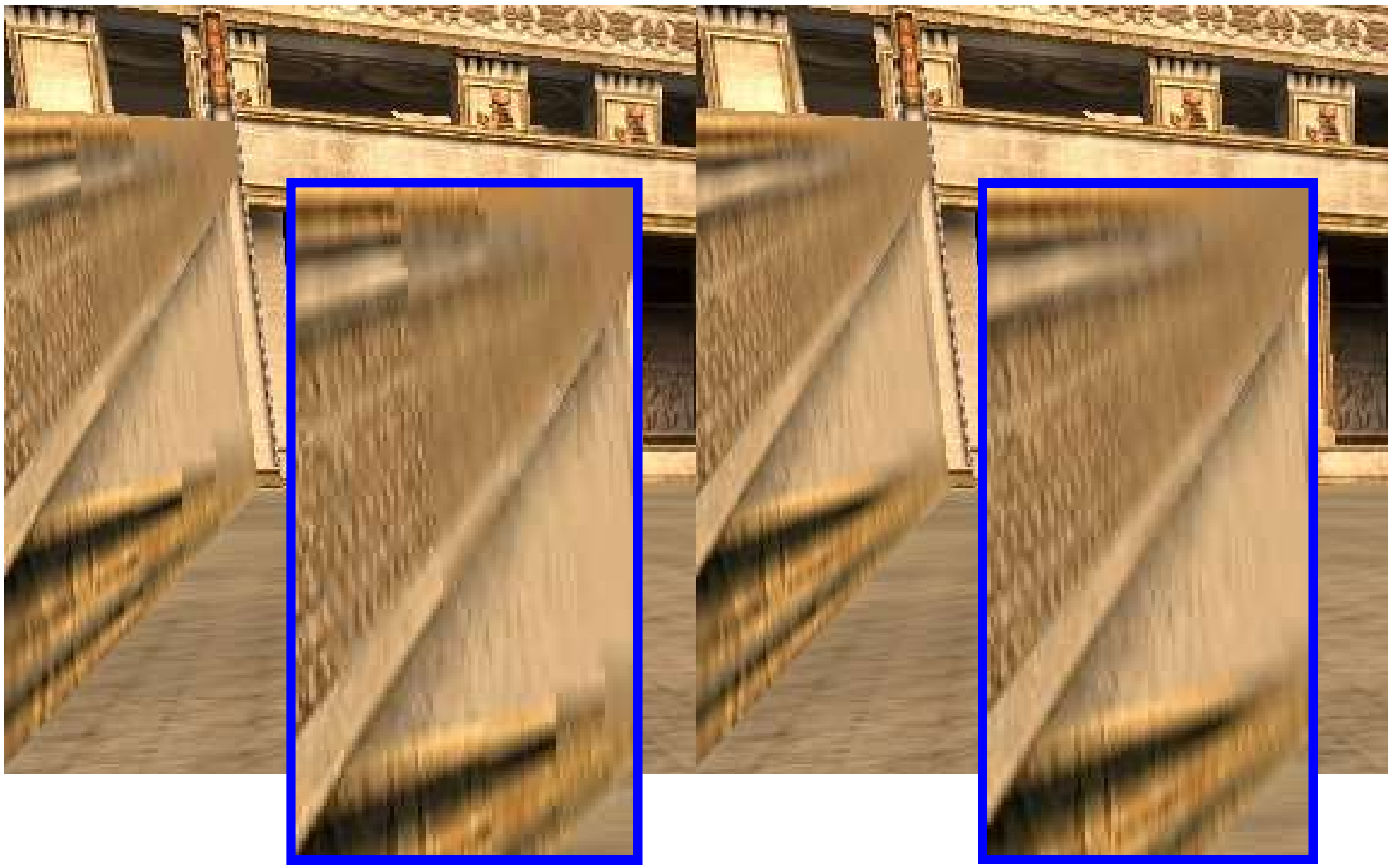


# MIP-Mapping



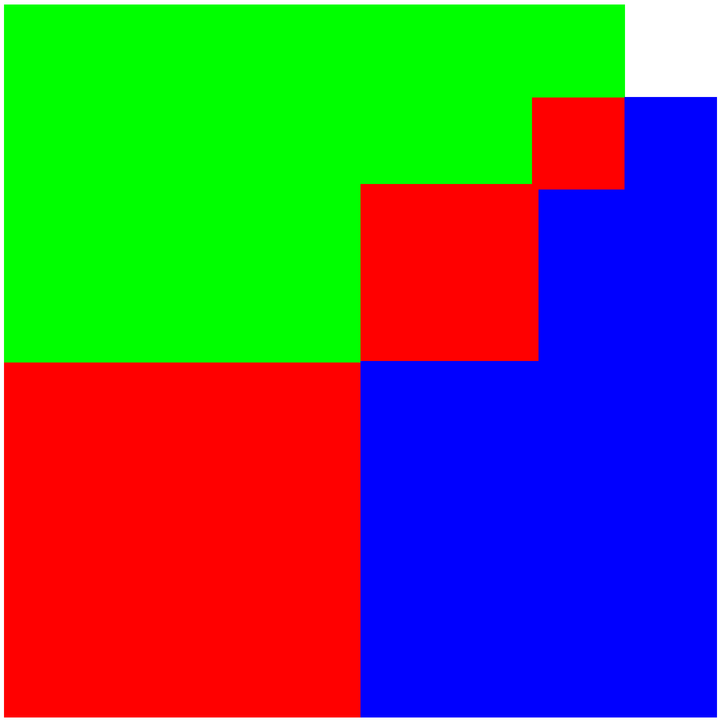
*Bilinear interpolation* *Trilinear interpolation*

# MIP-Mapping



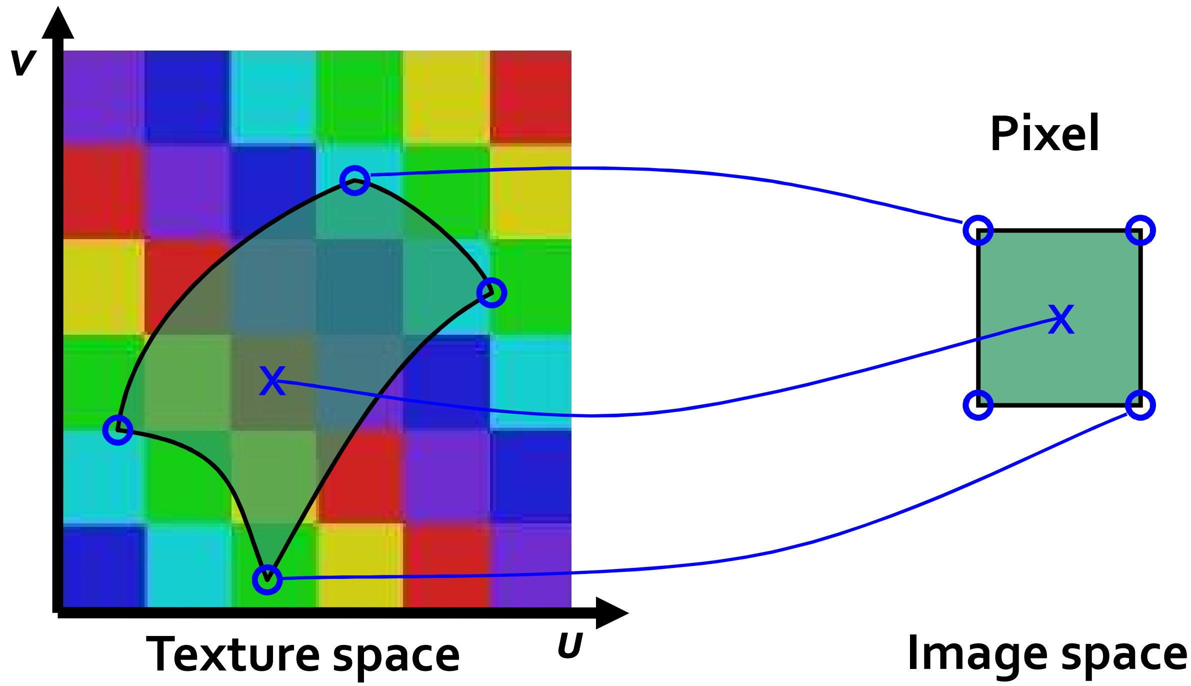
# MIP-Mapping: Memory Requirements

- Not twice the number of bytes…!
- Rather 33% more – not that much



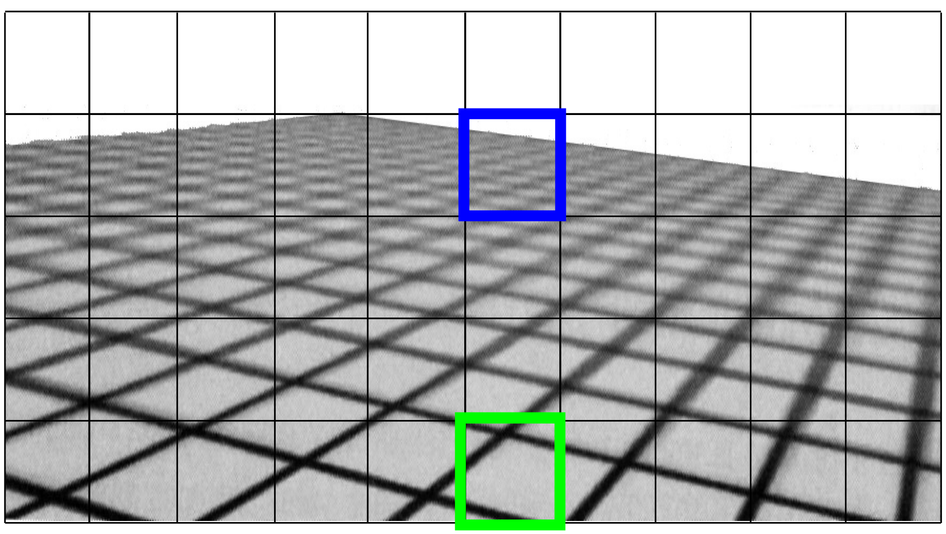
# Texture Anti-Aliasing

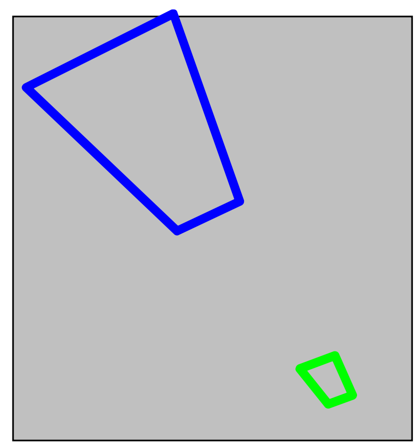
- Weighted mean of the pixel area projected into texture space



# Anisotropic Filtering

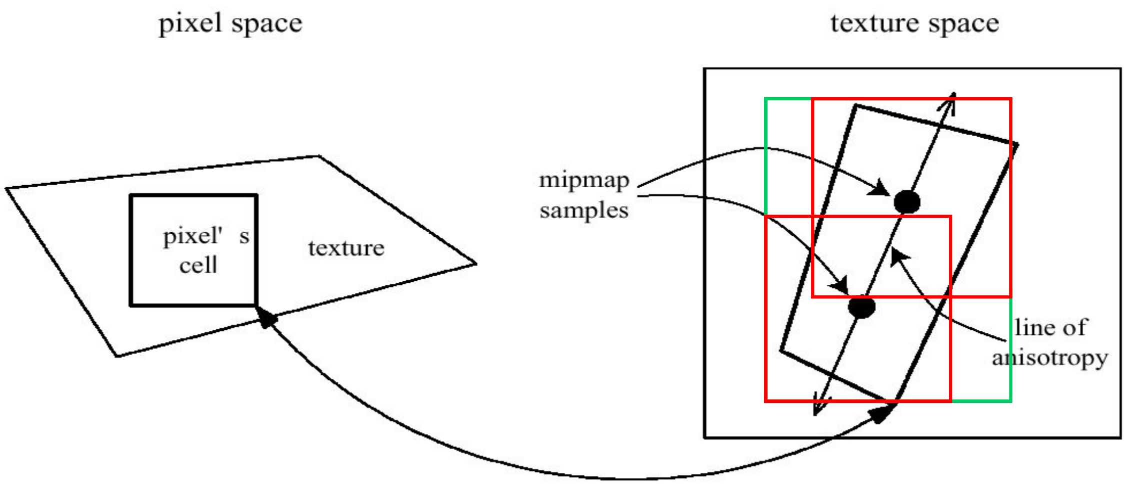
- View dependent filter kernel
- Implementation: *summed area table*, "*RIP Mapping*", "*footprint assembly*" , "***sampling***"
- Sampling idea: sample quadrilateral multiple times



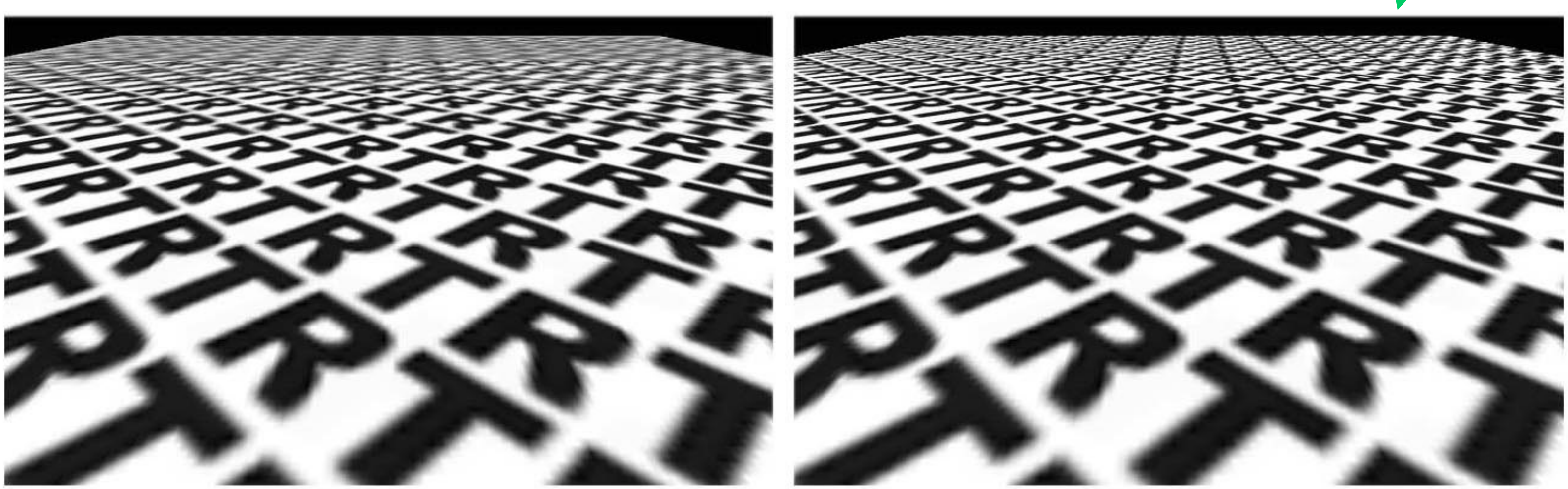


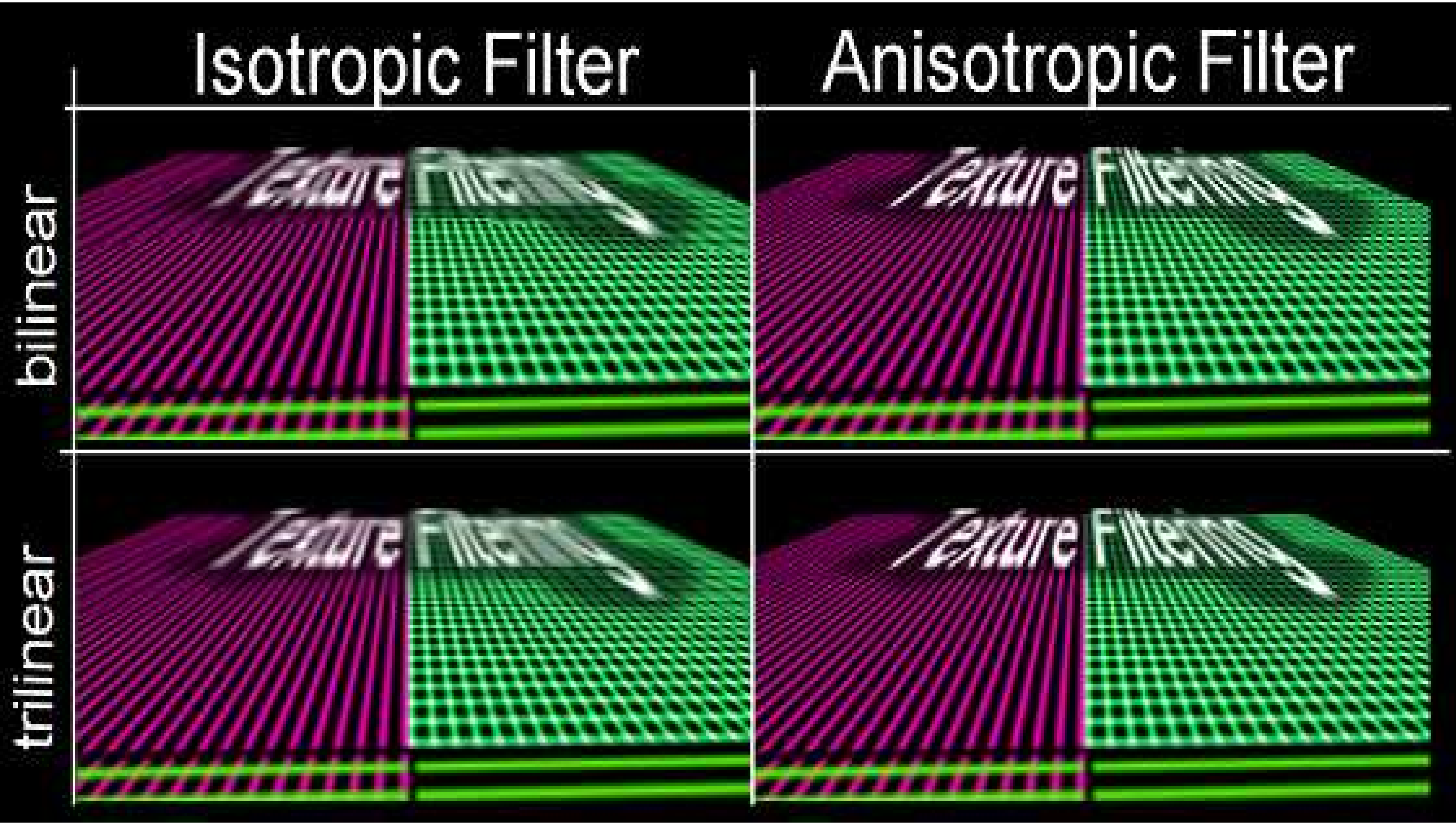
Texture space

# Anisotropic Filtering




16 samples





Isotropic Filter
Anisotropic Filter
bilinear
trilinear
Texture Filtering
Texture Filtering
Texture Filtering
Texture Filtering