Sampling

Aliasing

- Discretization of signals
  - *Space*
  - *Time*
  - *Color*
- High Frequency information appears as low frequency
- Jaggies

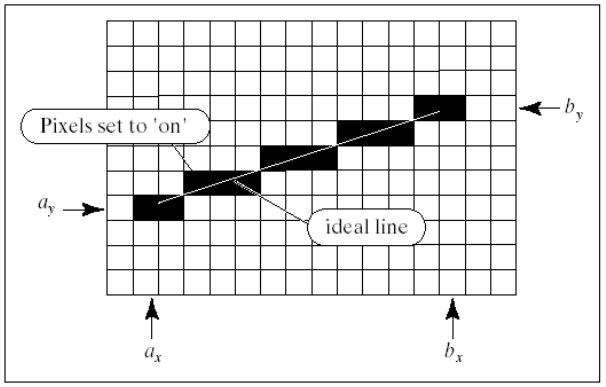


FIGURE 10.23 Drawing a straight-line-segment.

IA-81 Even during an Ice Age, things can get hot for Sid the sloth.

Sampling

Sampling Theorem (Nyquist or Shannon)

To avoid aliasing the sampling frequency must be greater than two times the highest frequency in the signal.

Sampling at frequency equal to highest frequency

Bad

$f_s < 2f$

Sampling exactly equal to highest frequency x 2 may produce good or bad results

Good

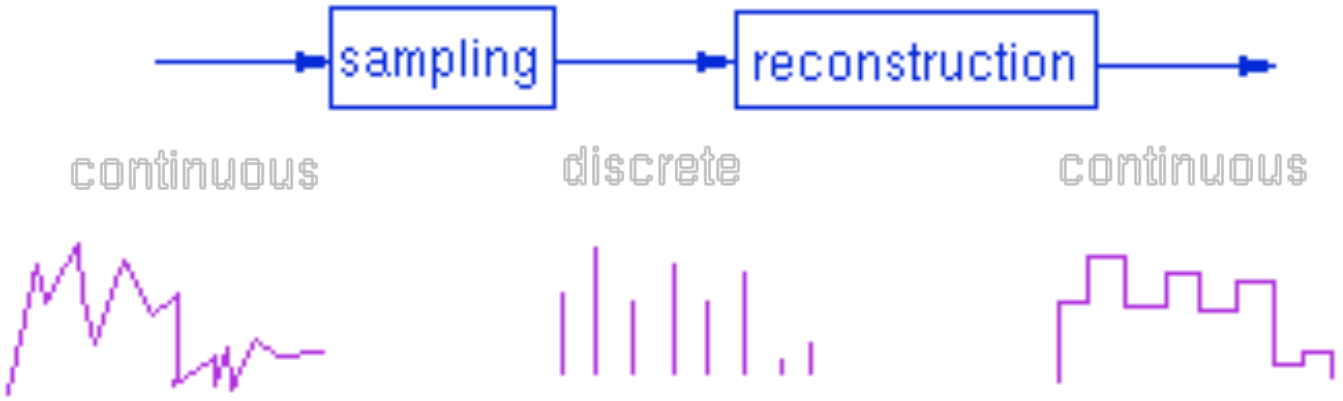
Bad

$f_s = 2f$

Sampling rate strictly greater than highest frequency x 2

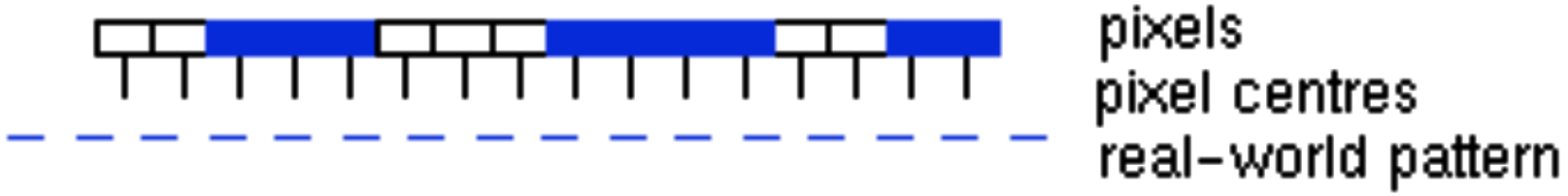
$f_s > 2f$

Sampling and reconstruction

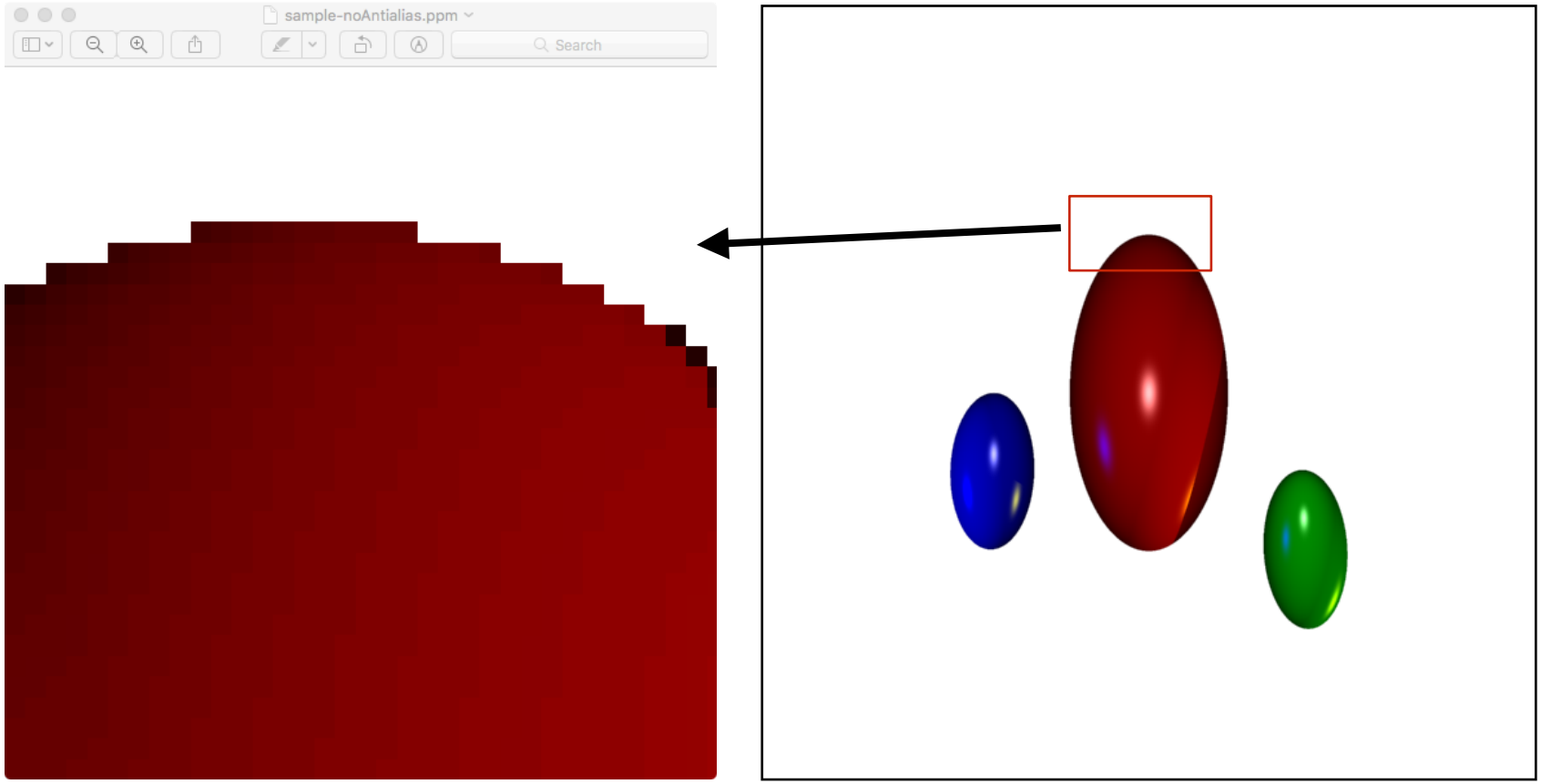


Sampling in graphics

Example: Sampling at pixel centers



Aliasing in Computer Graphics



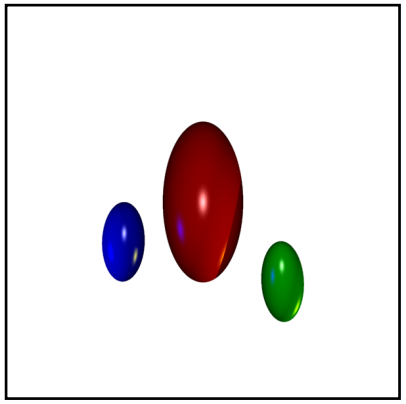
Reducing aliasing

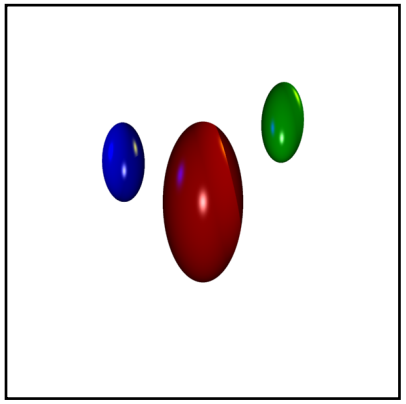
Only one way: Blur the image

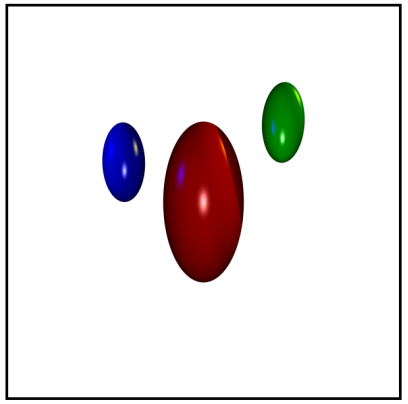
Strategies

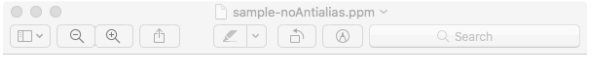
- Prefiltering (Before sampling)
  - *Pixel Coverages*
  - *Computationally expensive especially for non-polygonal objects*
- Postfiltering (After sampling)
  - *Weighted average of samples*

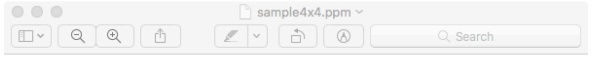
Aliasing in Computer Graphics

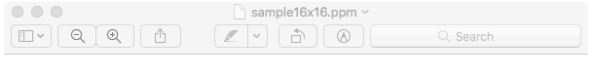


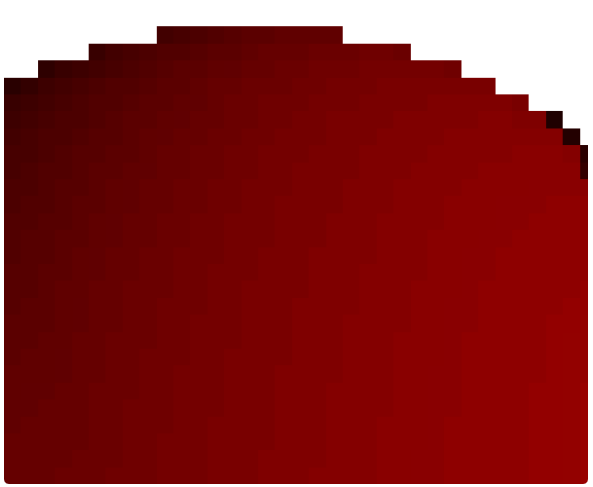


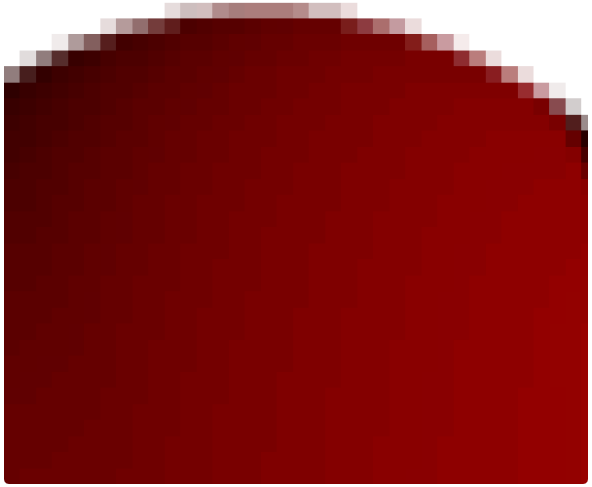


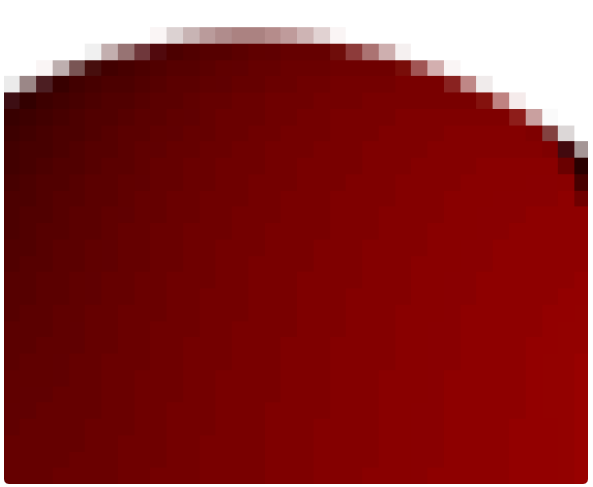












Filtering

Filtering of a function f(x) using convolution with a kernel g(x)

$$P(x) = \int_{-s}^{s} f(x-u)g(u)du$$

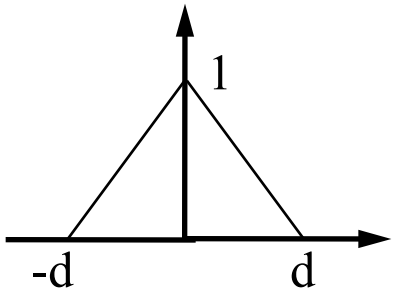
where s can be infinity or a finite horizon

Discreet form

$$P(n) = \sum_{u=-m}^{m} f[n-u]g[u]$$

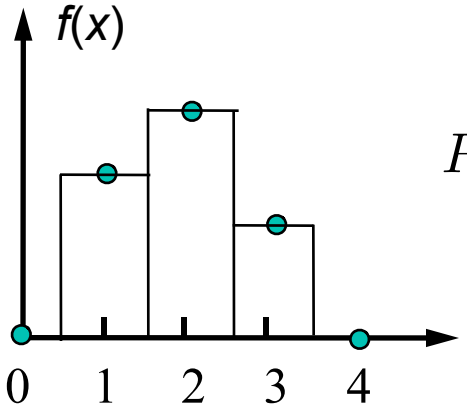
Example

Filtering of a step function using convolution with tent kernel g (linear interpolation)

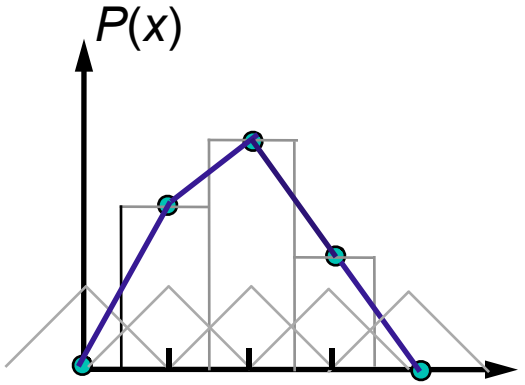


$$g(u) = \begin{cases} 1 - \frac{u}{d} & u \le |d| \\ 0 & u > |d| \end{cases}$$

Tent kernel, for the example *d* =1



$$P(x) = \int_{-s}^{s} f(x-u)g(u)du$$



Size of the kernel

Filtering of a function f(x) using convolution with a kernel g(x)

$$P(x) = \int_{-s}^{s} f(x-u)g(u)du$$

The extent of the kernel (-s,s) defines how many neighbors of x contribute to the value of P(x)

Pre-filtering

Unweighted Area sampling

- Use average intensity of square pixel area

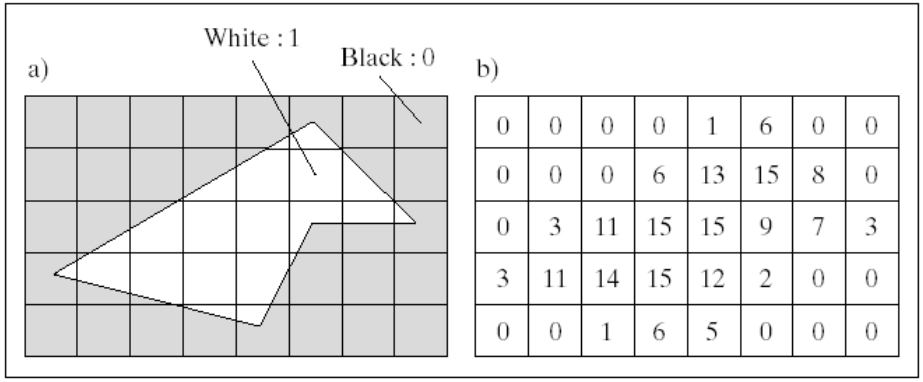


FIGURE 10.49 Using the fraction of the pixel area covered by the object.

Black 0, White 1 (15)

Pixel value: coverage*15

Incremental Polygon Antialiasing

Bresenham's Algorithm provides the dotted pixels (boundary)

- Incremental area calculation

Inside pixels 1

Outside pixels 0

Boundary pixels fractions based on coverage

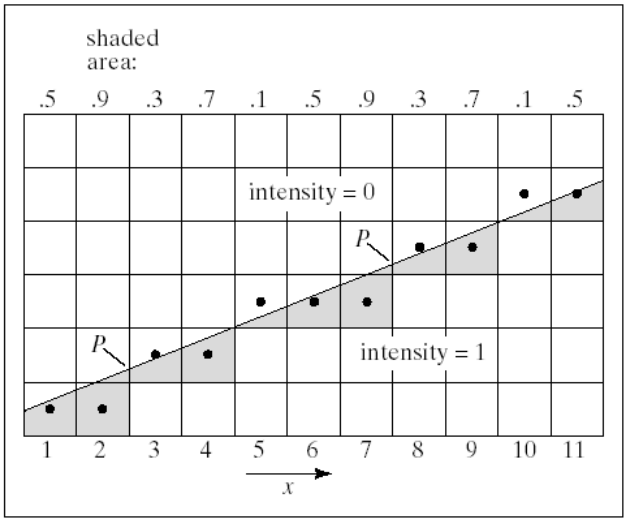
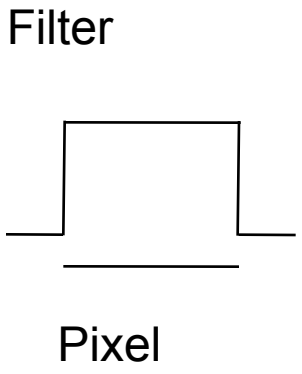


FIGURE 10.50 Example of scan conversion with antialiasing.

Box Filter

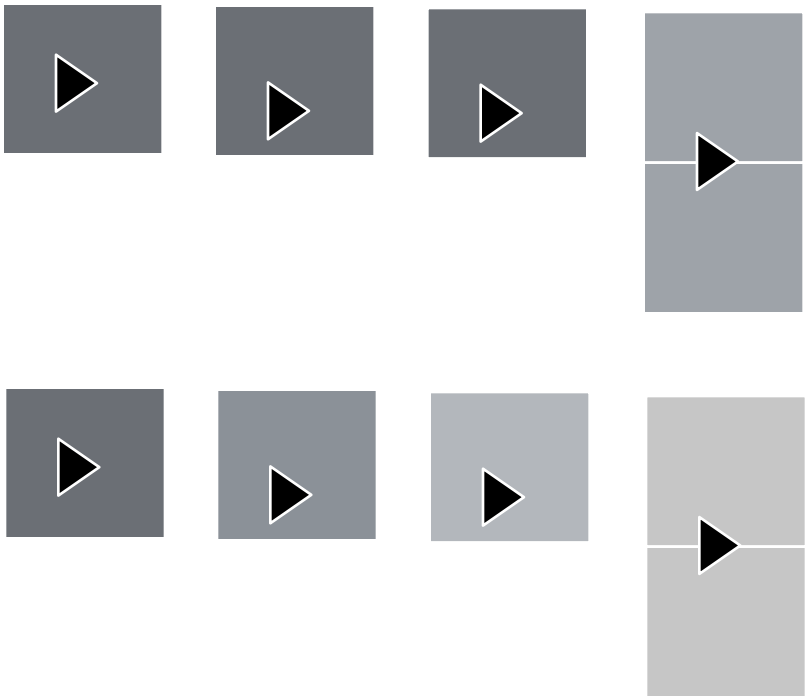
Area coverage approach corresponds to a box filter

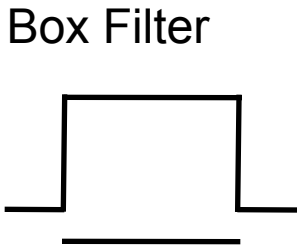


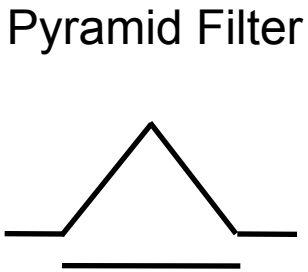
$$P(x) = \int_{-s}^{s} f(x-u)g(u)du$$

Problem with Box filter

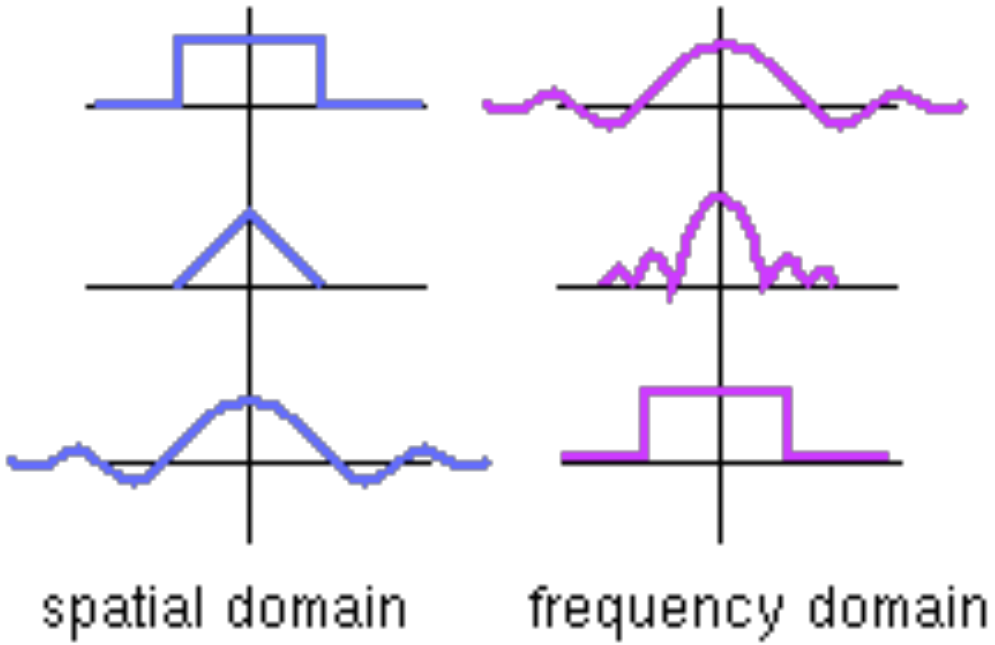
Area Coverage Independent of position



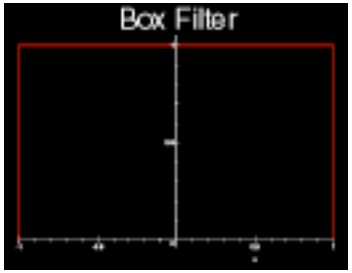


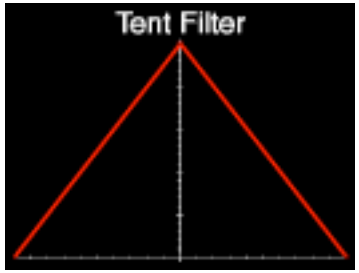


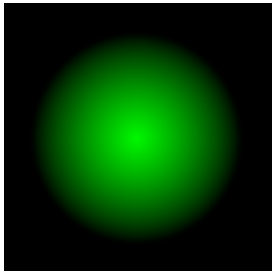
Filter Kernels

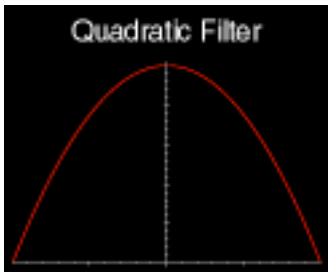


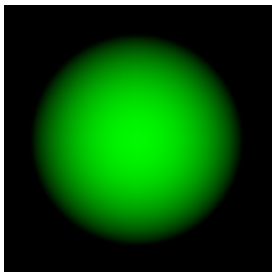
Filter visualization

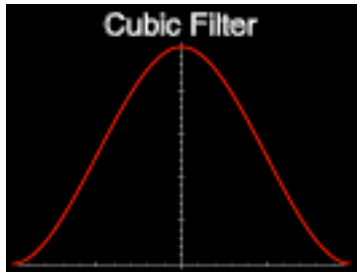


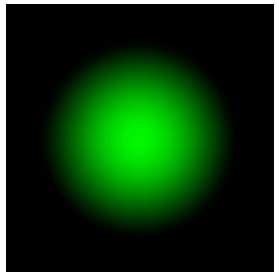












Images
Courtesy of Josh Grant

Postfiltering

Super sampling

- Take many samples
- Combine them

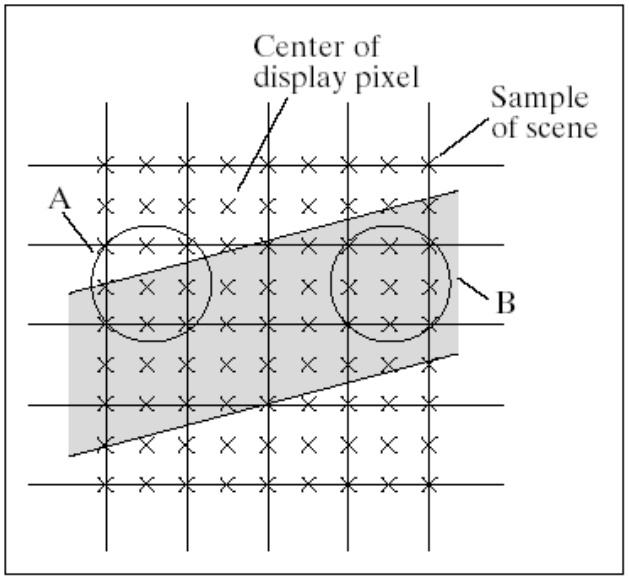
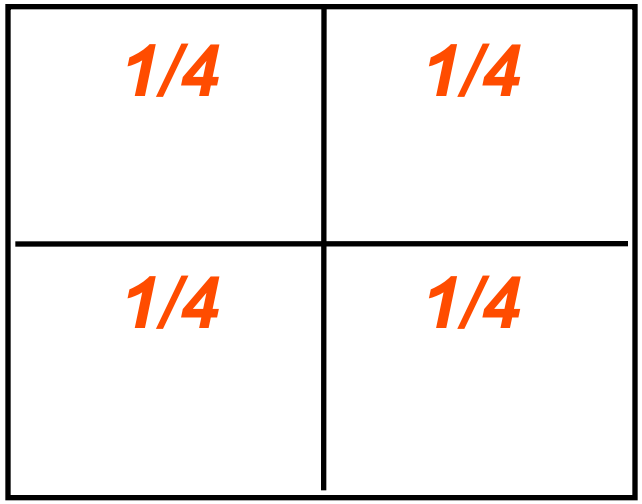


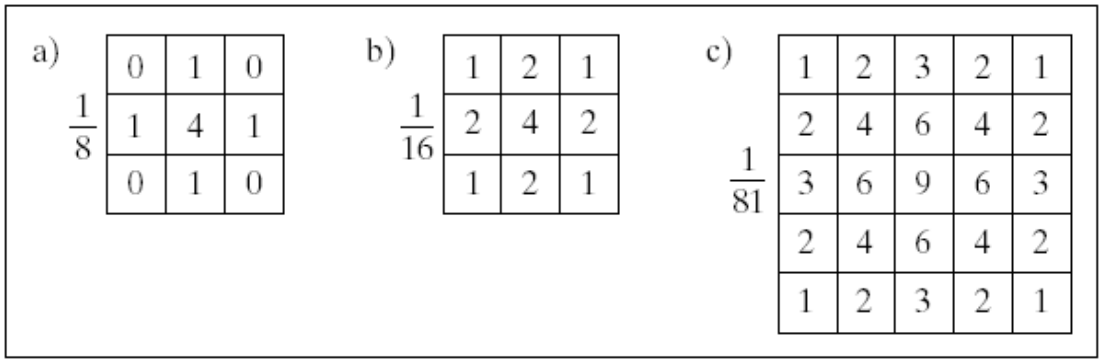
FIGURE 10.51 Antialiasing using supersampling.

Box Filter (Discreet version)



Bartlett window

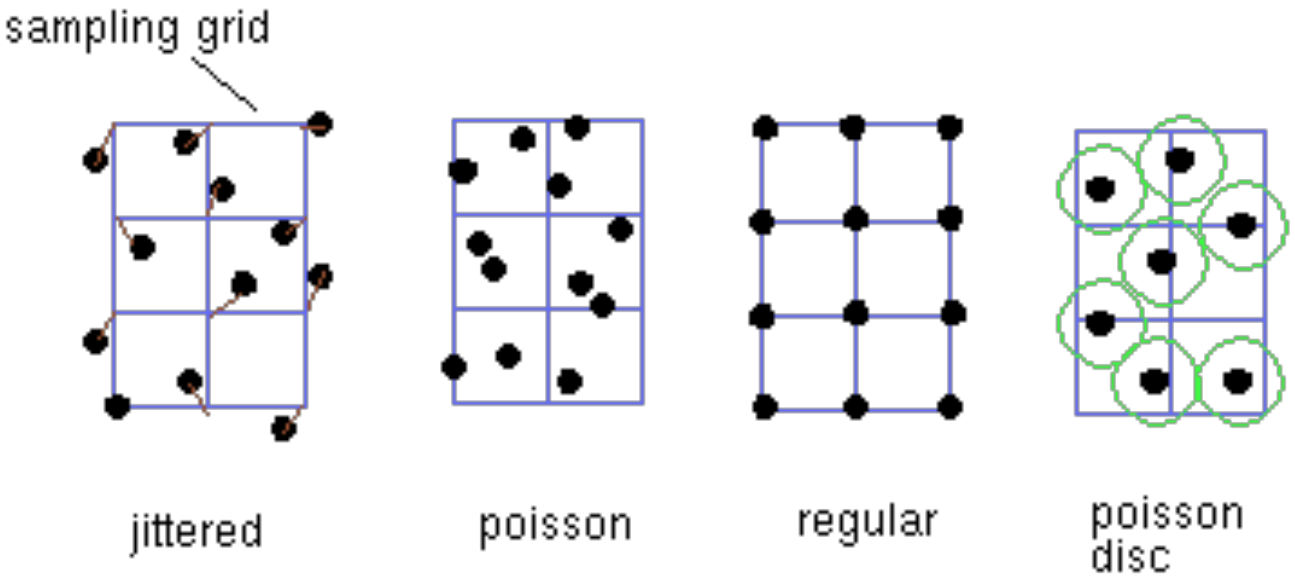
FIGURE 10.55 Examples of window functions.



All weights add up to 1

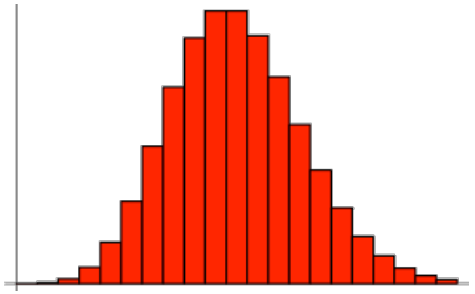
Stochastic supersampling

High frequency noise



Poisson Distribution (Aside)

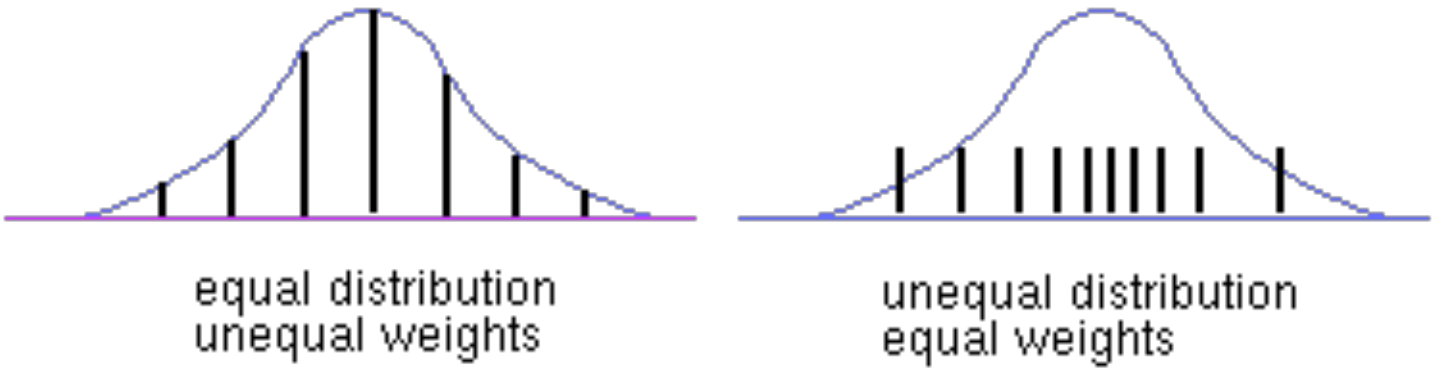
$$P(\lambda, n) = \lambda^n \frac{e^{-\lambda}}{n!}$$



- Lambda is the expected rate of occurrence of an event in a given time interval (or area,volume etc)
- *P* is the probability that the event will happen *n* times in the given time interval (or area, volume etc)
- The mean and variance of the distribution are both lambda

Importance Sampling

Location vs density



Height indicates weight

Scene antialiasing

- Draw the scene multiple times each time from a slightly different point of view. Blend the results

- Raytracing: Shoot multiple rays

- xxGL: Enable Screen Space Multi-Sampling

Multi-sampling

General Idea

- Take a number of samples in screen space
- Somehow combine them

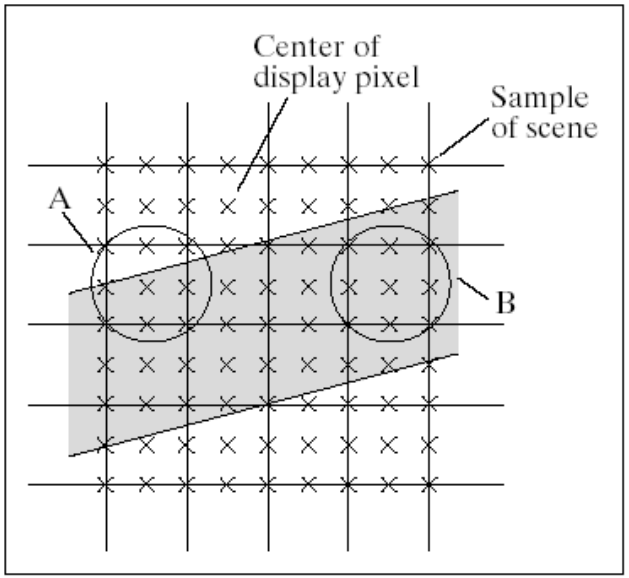


FIGURE 10.51 Antialiasing using supersampling.

WebGL 2.0

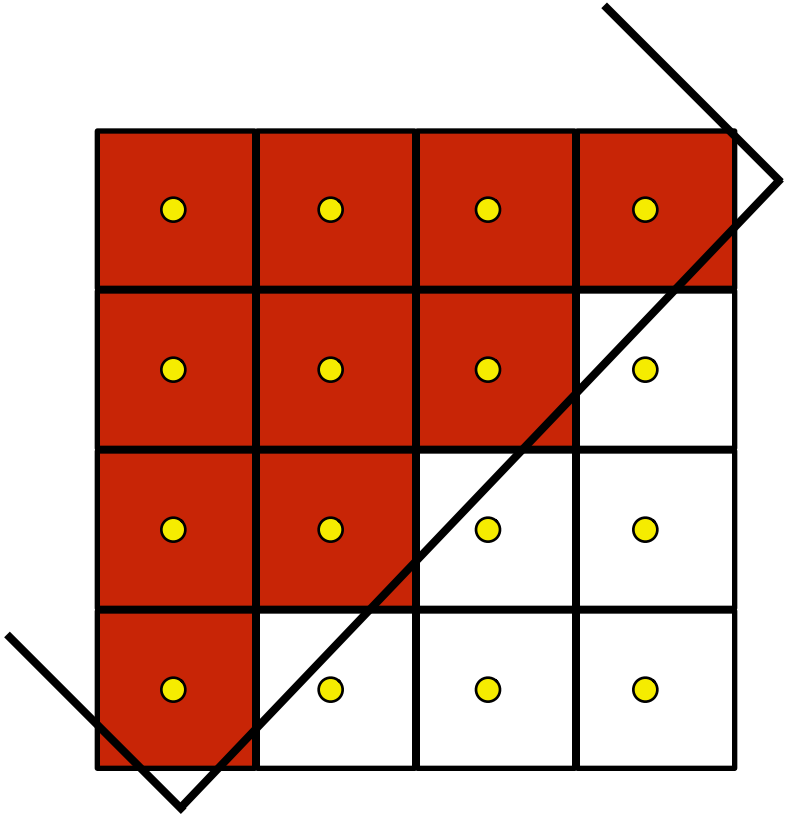
- var gl = canvas.getContext('webgl2', { antialias: true });
- Non standard behaviour
- Renderbuffers are antialiased though and can be used to indirectly apply multisampling
- See: https://stackoverflow.com/questions/50255282/webgl2-has-anti-alias-automatically-built-in

Multisample Anti-Aliasing (MSAA) in xxGL

- Rasterizer produces more than one sample per fragment
- Default behavior
  - *The fragment shader runs once per fragment if any sample is within the polygon*
  - *The result is scaled based on how many of the samples fall within the polygon (default behaviour)*

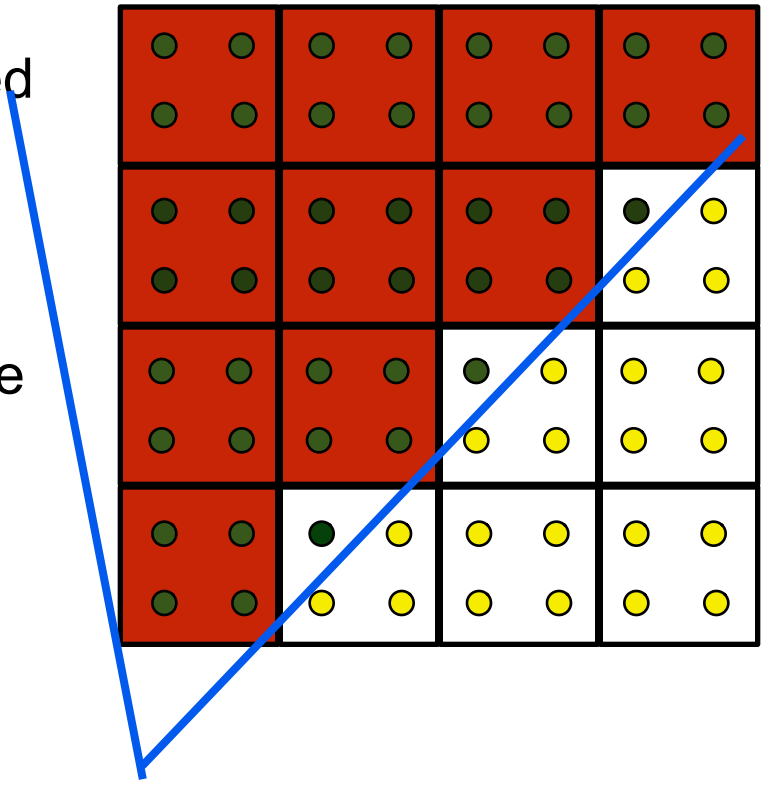
No Multi-sampling

- Only the centre of the pixel is considered
- If the centre is in the polygon the fragment shader is executed
- The values for the interpolated variables are interpolated at the **center** of each pixel by the rasterizer



4xMSAA in xxGL

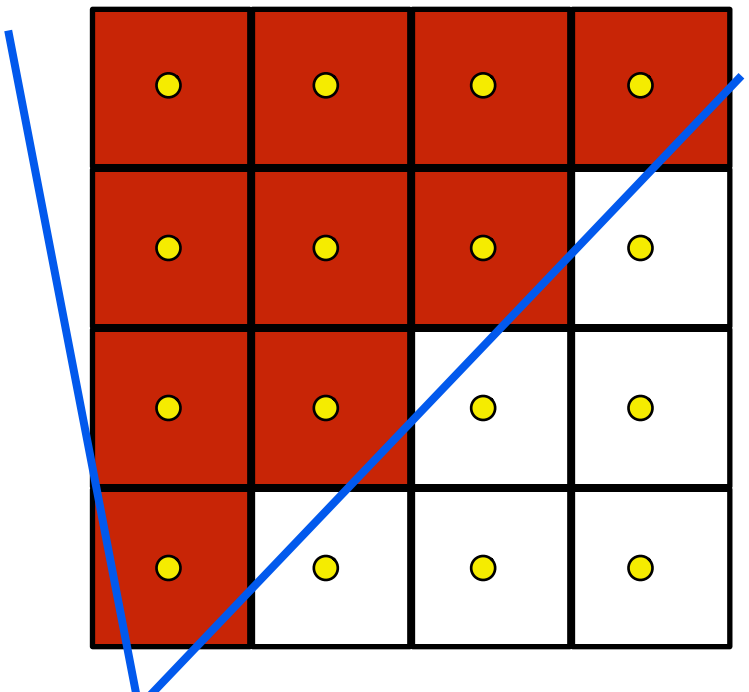
- Samples at various locations within pixels
- The values for the interpolated variables are interpolated at the center of each pixel
- Therefore they can be interpolated for values outside of the polygon
- Is that a problem?



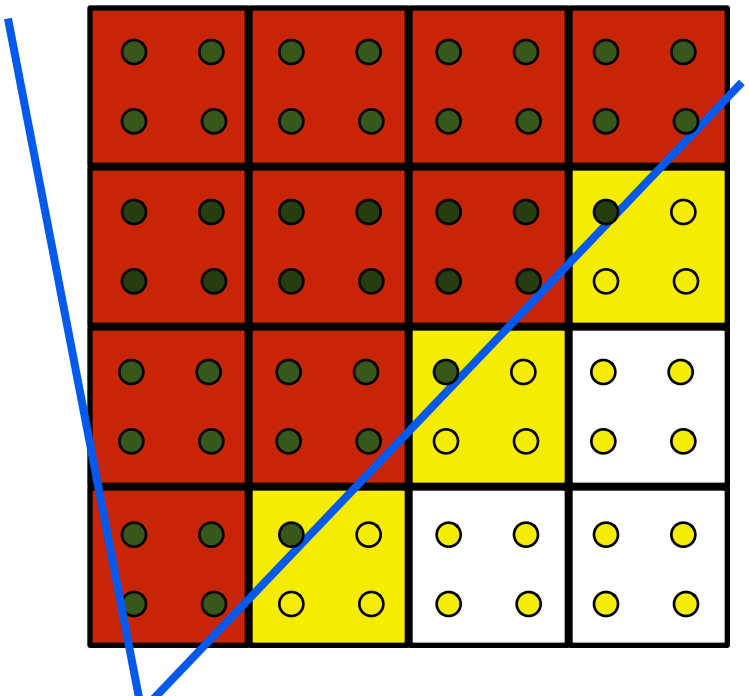
Interpolating outside the vertex attribute range

- Let's say that a fragment shader assigns a red colour if the texture coordinates are <=1 and yellow if >1. For example in a clamp to border situation.

Without



With 4x multi-sampling



Interpolation Qualifiers

Inside the polygon

- Qualifier: `centroid in vec2 TexCoord;`

On the sample itself

- Qualifier: `sample in vec2 TexCoord;`
- In that case the fragment shader is executed once per sample

The qualifier has to appear both in the vertex and the fragment shader

Setting up multi-sampling

Window toolkit based - GLFW:

```
glfwWindowHint(GLFW_SAMPLES, 8);
```

Operating system/Driver override

- Operating system and driver specific

From OpenGL:

```
GLint bufs, samples;
glGetIntegerv(GL_SAMPLE_BUFFERS, &bufs);
glGetIntegerv(GL_SAMPLES, &samples);
printf("MSAA: buffers = %d samples = %d\n", bufs,
          samples);
glEnable(GL_MULTISAMPLE);
```

Use multi-sampling

Nothing else really to do

- Vertex Shader using “centroid”

```
#version 400
layout (location = 0) in vec3 VertexPosition;
layout (location = 1) in vec3 VertexNormal;
layout (location = 2) in vec2 VertexTexCoord;

centroid out vec2 TexCoord;

uniform mat4 MVP;

void main()
{
    TexCoord = VertexTexCoord;
    gl_Position = MVP * vec4(VertexPosition,1.0);
}
```

Fragment Shader

Must also use “centroid”

```
#version 400

centroid in vec2 TexCoord;

layout( location = 0 ) out vec4 FragColor;

void main()
{
    vec3 yellow = vec3(1.0,1.0,0.0);
    vec3 blue = vec3(0.0,0.0,1.0);
    vec3 color = blue;
    if( TexCoord.x > 1.0 )
       color = yellow;
    FragColor = vec4( color , 1.0 );
}
```

Texture Antialiasing

Pixels have area

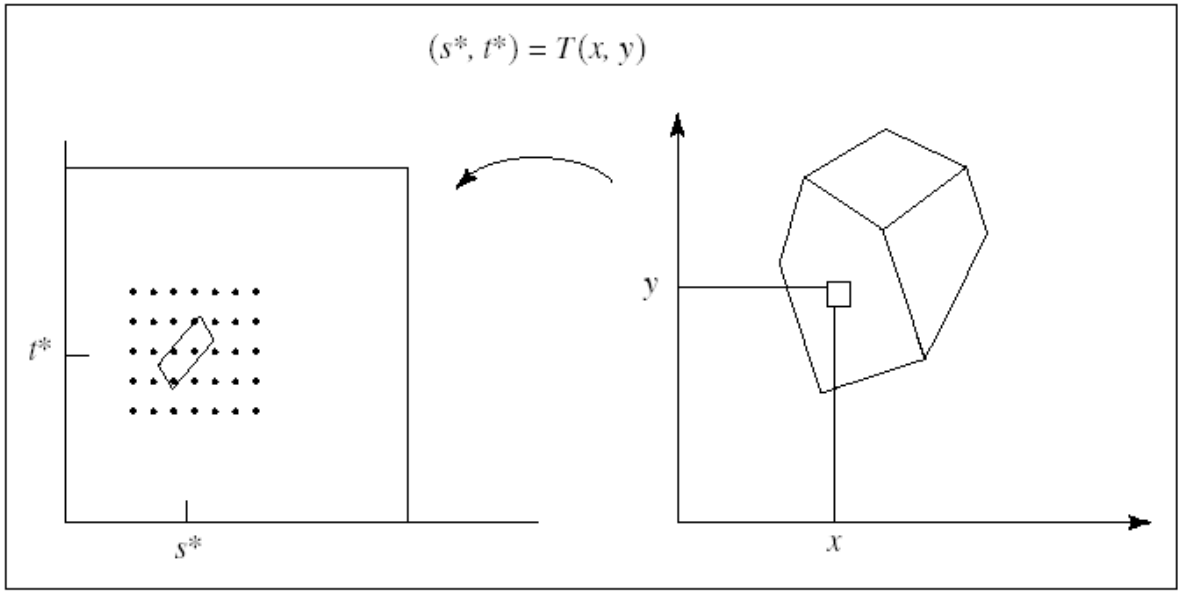
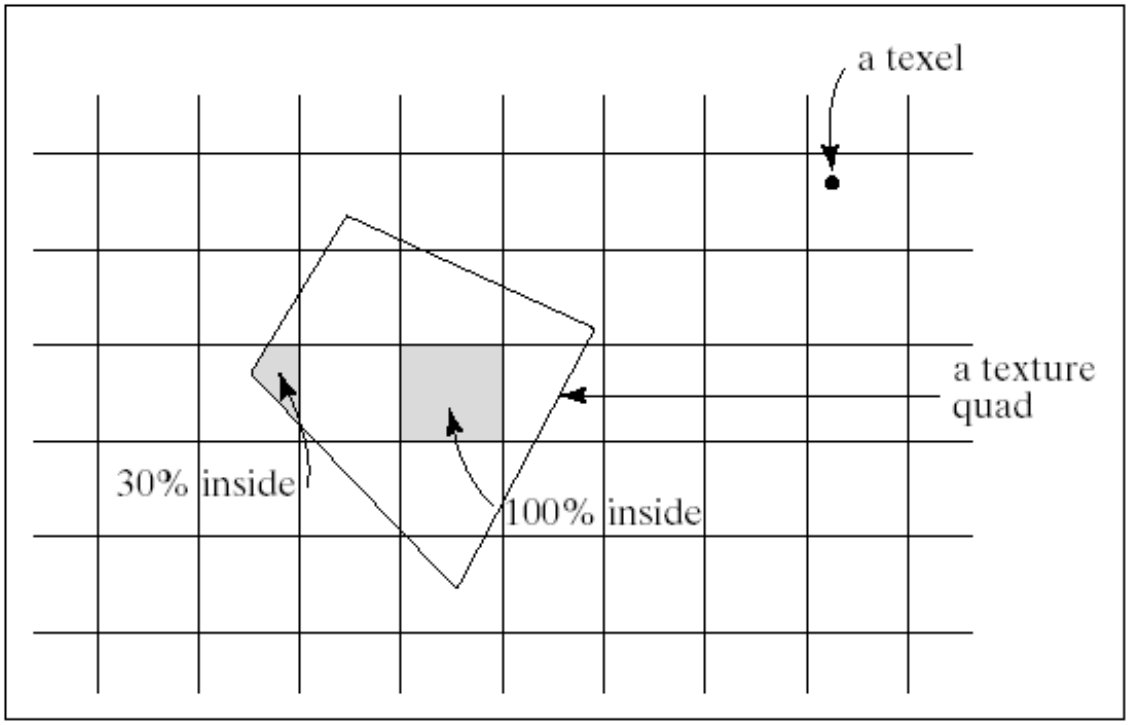


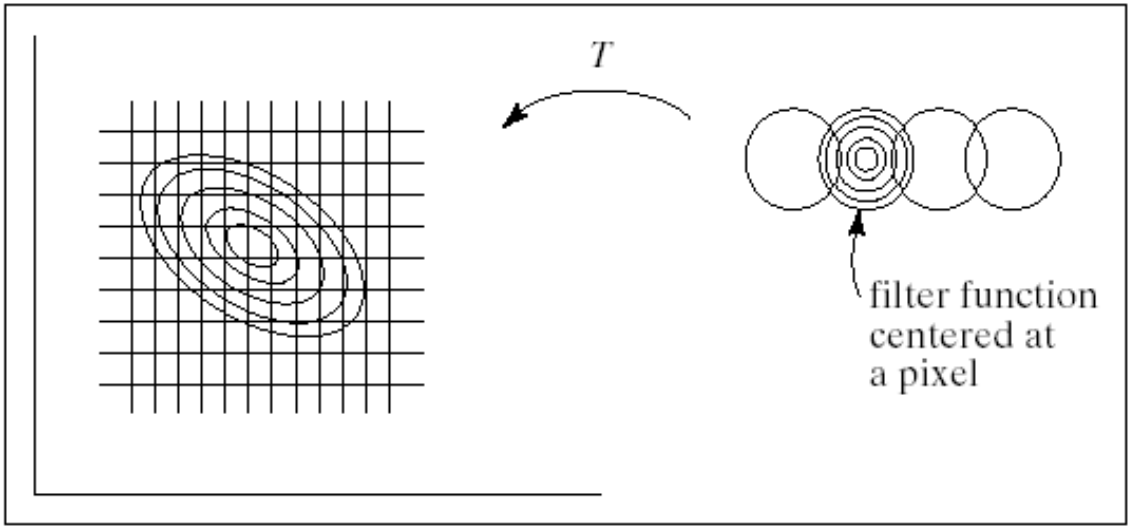
FIGURE 10.57 Cause of aliasing in rendering texture.

Area coverage

Two costly



Elliptical weighted average



Stochastic sampling

Average = $1/N_k$ texture(s + a_k, t+b_k)

Where a_k, b_k are small random quantities and N_k the number of samples.

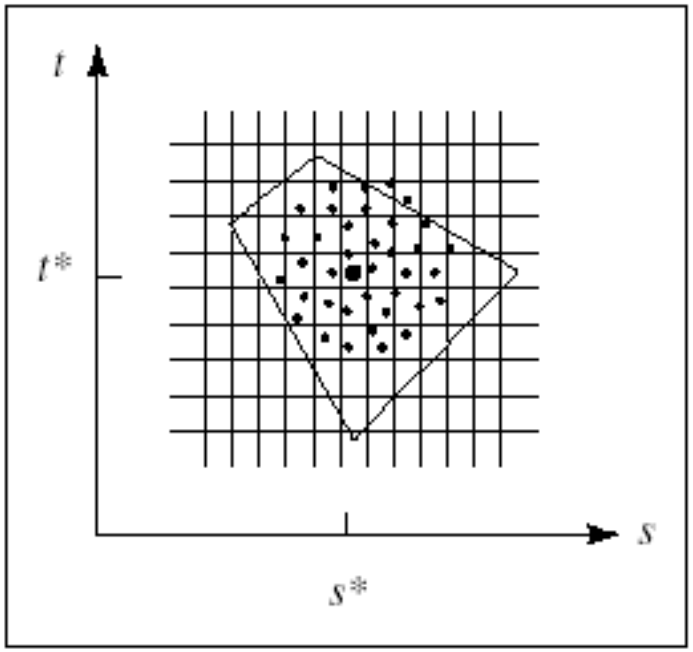


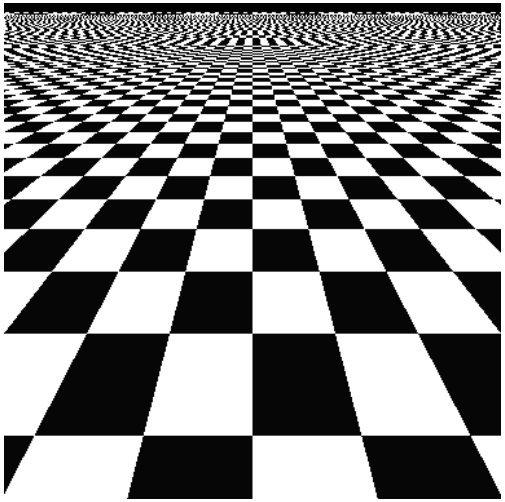
FIGURE 10.60 Antialiasing using stochastic sampling.

Examples

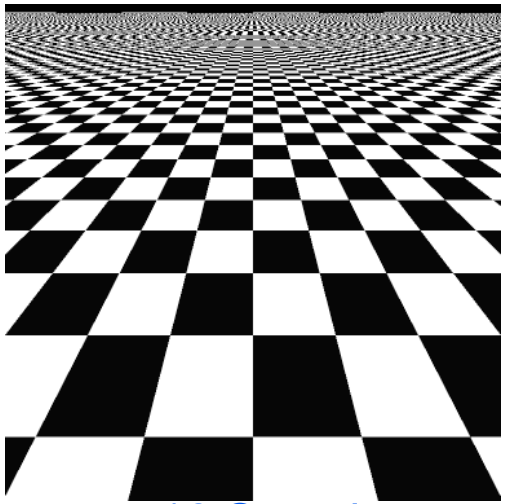
From: www.hpl.hp.com/research/mmsl/projects/graphics/antialiasing/index.html

Box vs Tent filtering over regular grids

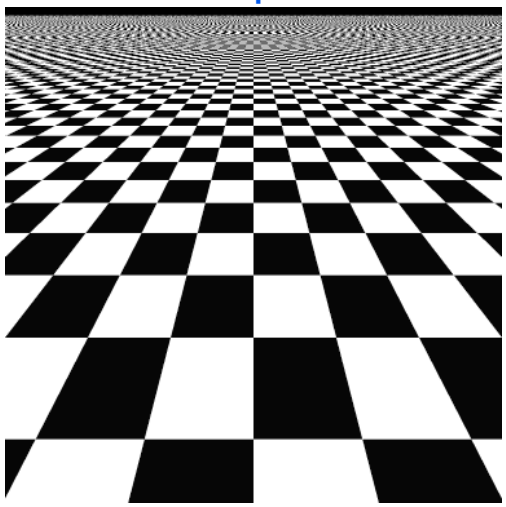
1 Sample



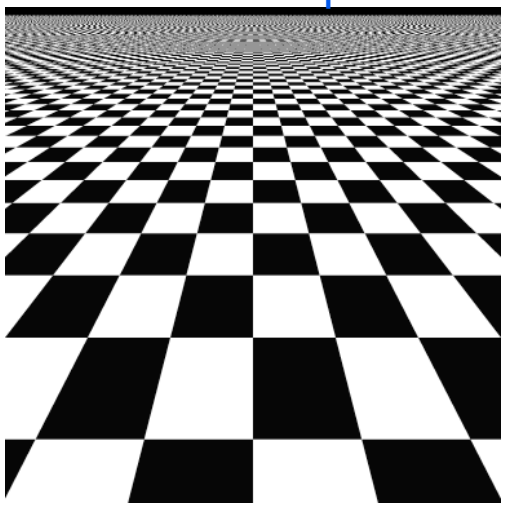
4 Samples



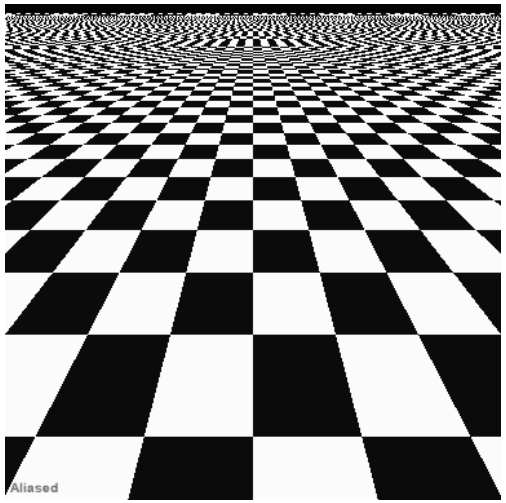
9 Samples

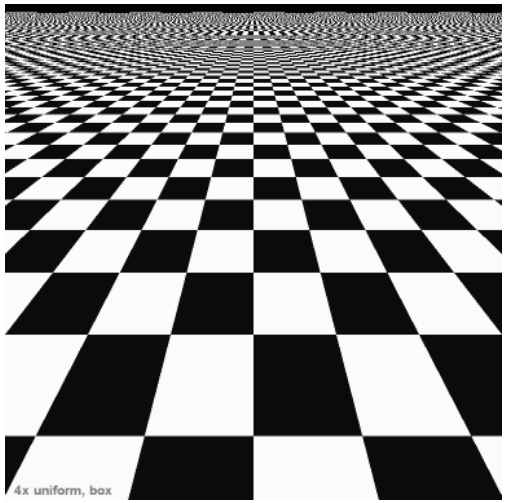


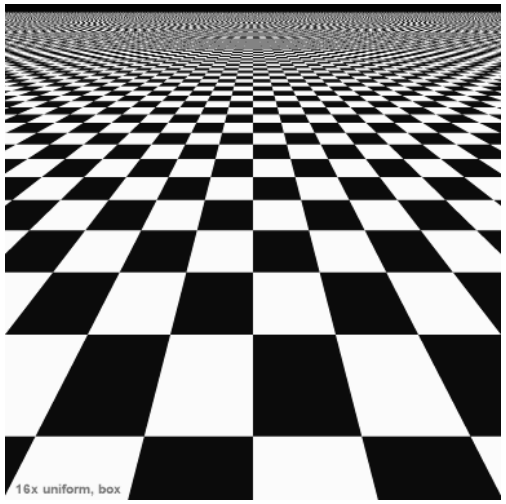
16 Samples

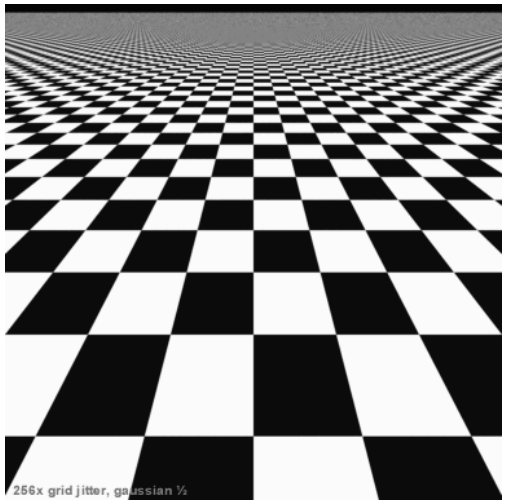


Box vs Tent filtering over regular grids --- Animation









Human Eye

- Our visual system is more sensitive to aliasing effects than noise
- Lens acts as a filter
- Photoreceptors are non-uniformly distributed, effectively creating a non-uniform sampling effect