Sampling, Resampling, and Warping

Felix Heide

Princeton University

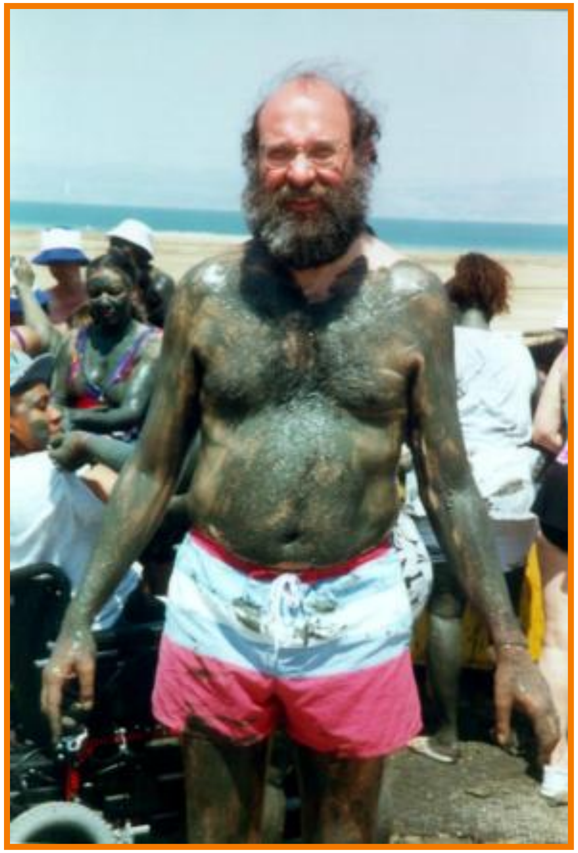
COS 426, Spring 2022

Digital Image Processing

- Changing pixel values
  - Linear: scale, offset, etc.
  - Nonlinear: gamma, saturation, etc.
  - Histogram equalization
- Filtering over neighborhoods
  - Blur & sharpen
  - Detect edges
  - Median
  - Bilateral filter
- Moving image locations
  - Scale
  - Rotate
  - Warp
- Combining images
  - Composite
  - Morph
- Quantization
- Spatial / intensity tradeoff
  - Dithering

Image Warping

- Move pixels of an image



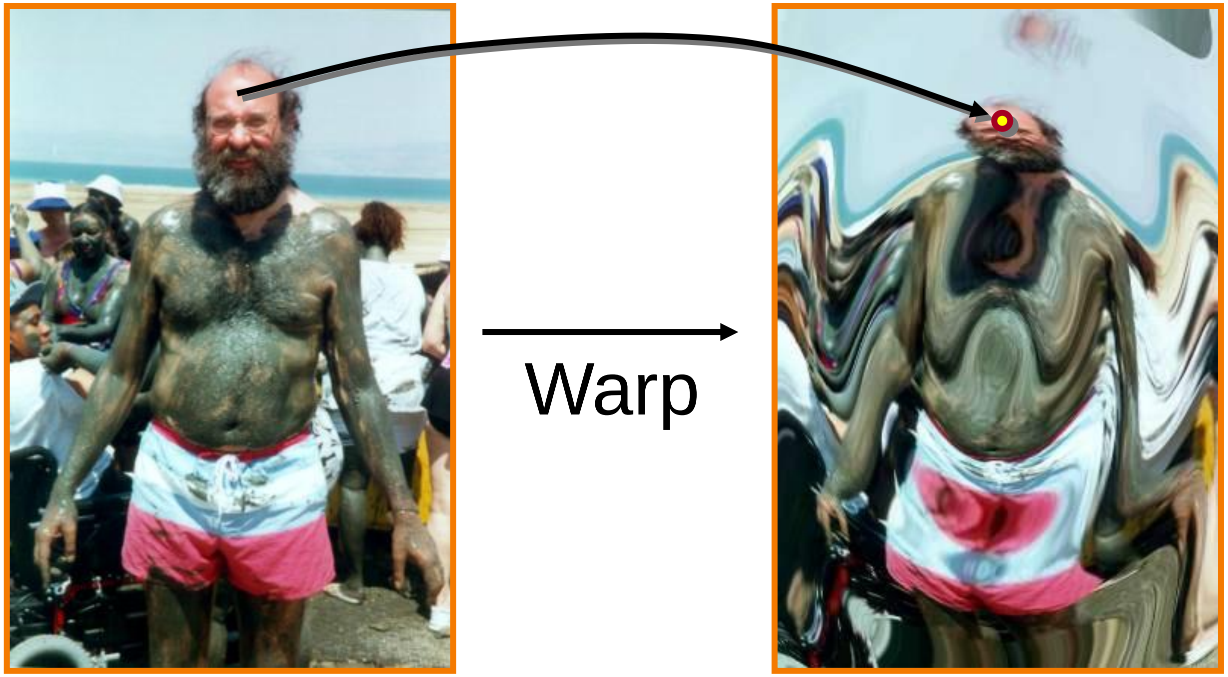
Warp

Source image

Destination image

Image Warping

- Issues:
  - Specifying where every pixel goes (**mapping**)



Source image

Destination image

Image Warping

- Issues:
    - Specifying where every pixel goes (mapping)
    - Computing colors at destination pixels (**resampling**)

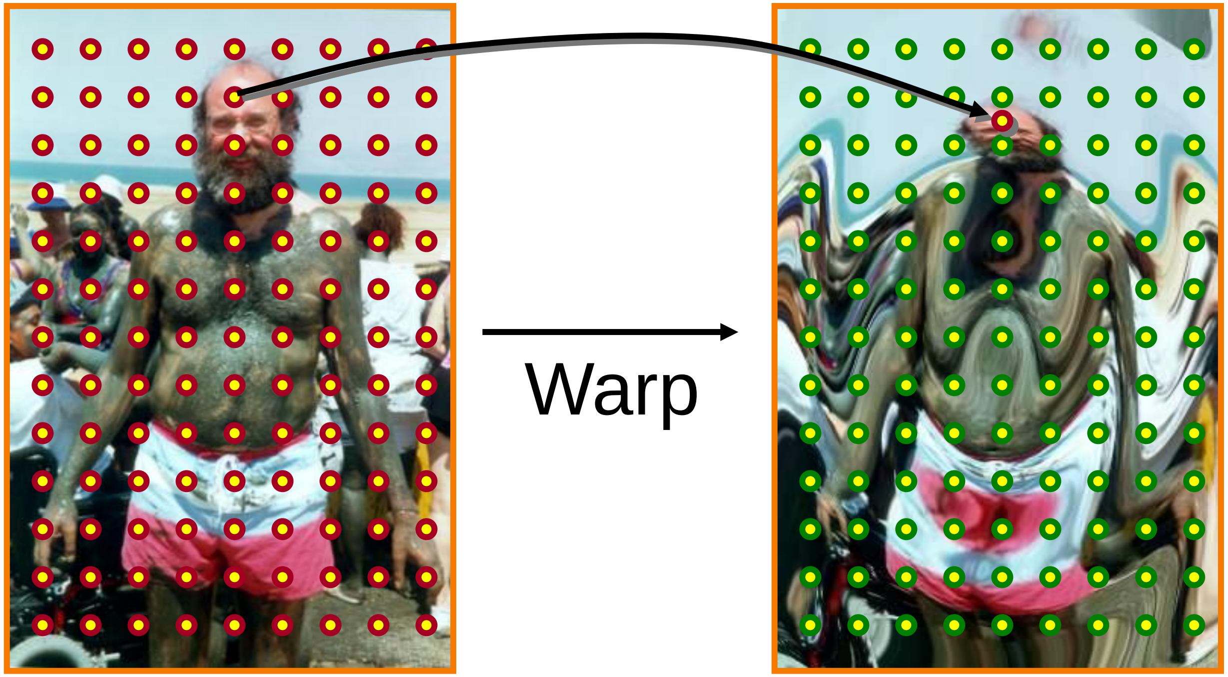
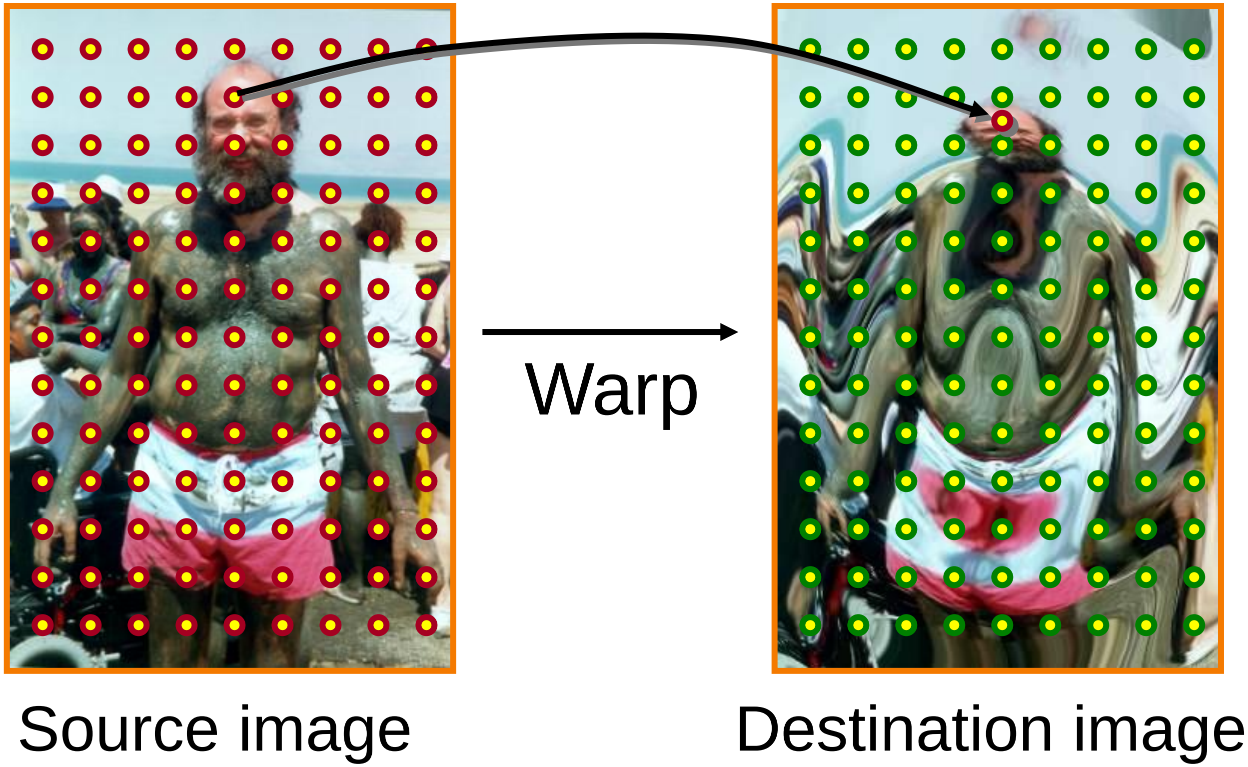


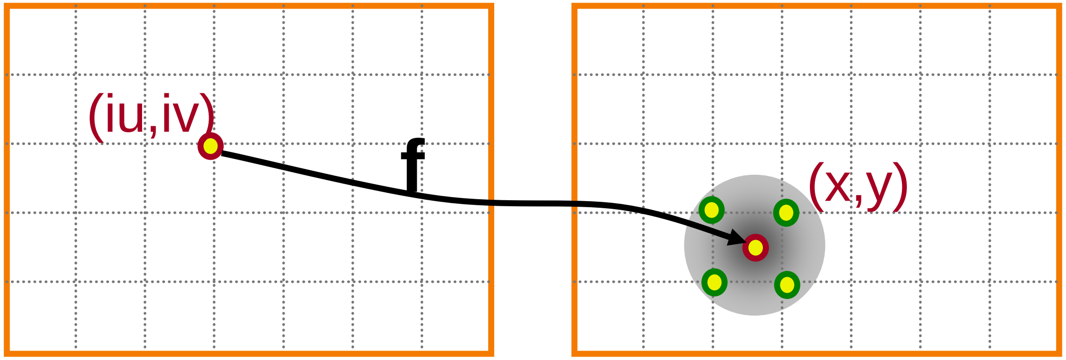
Image Warping

- Issues:
  - ➢ Specifying where every pixel goes (mapping)
  - ○ Computing colors at destination pixels (resampling)



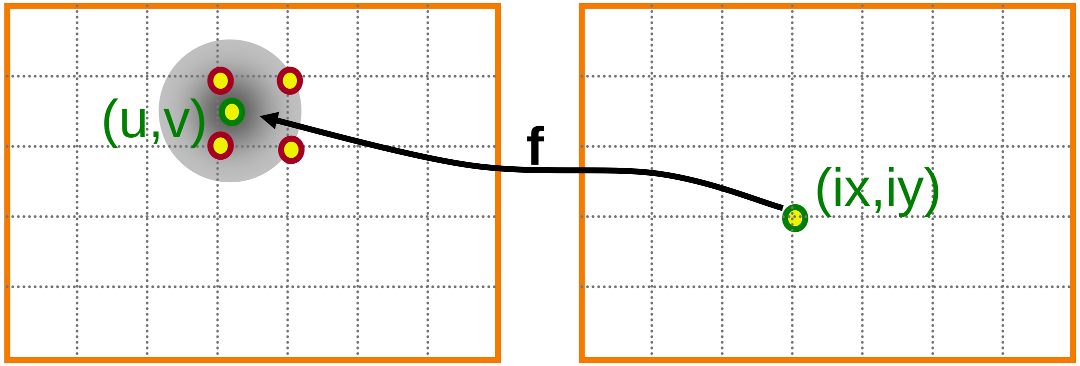
Two Options

- Forward mapping



Source image Destination image

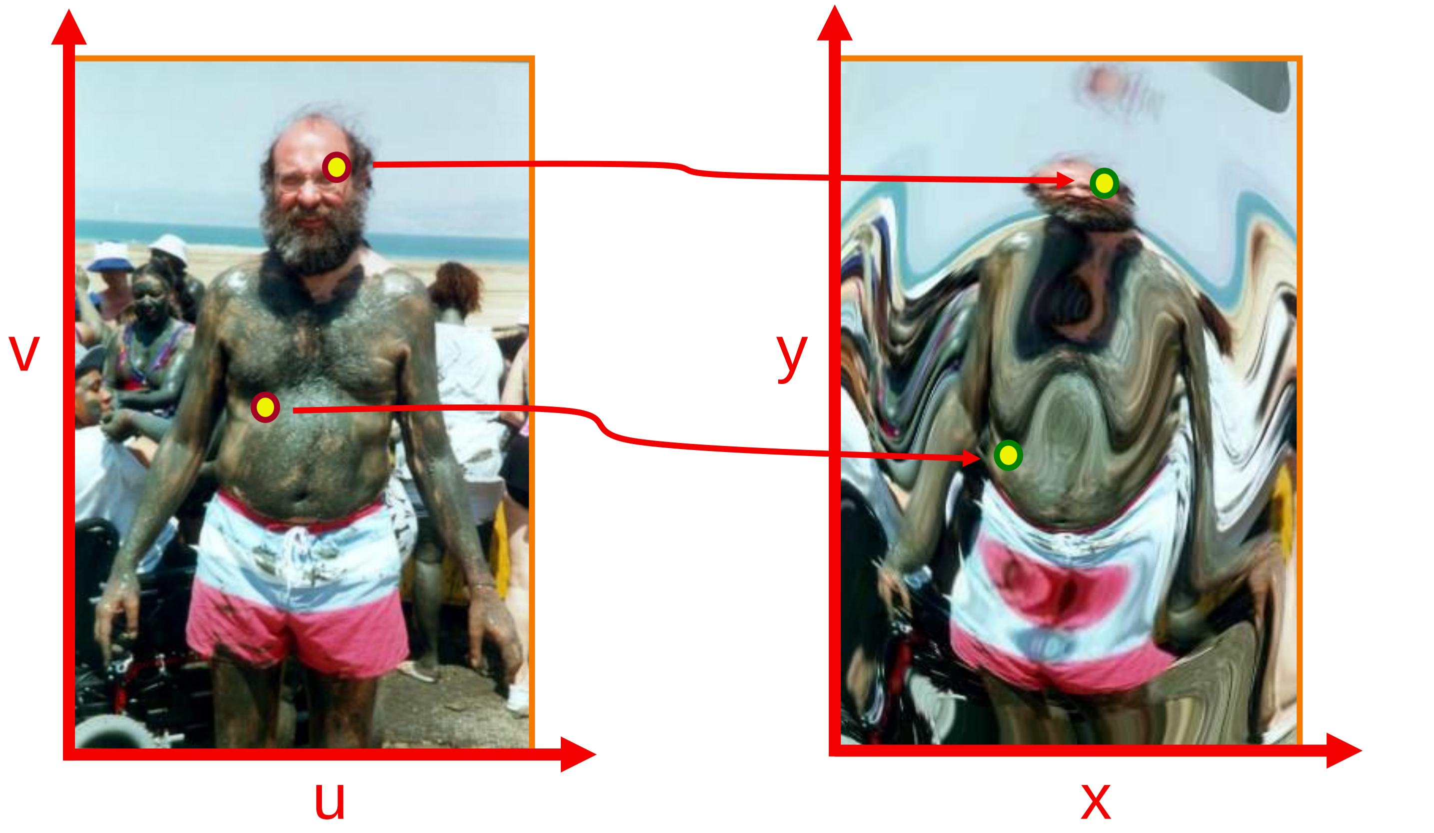
- Reverse mapping



Source image Destination image

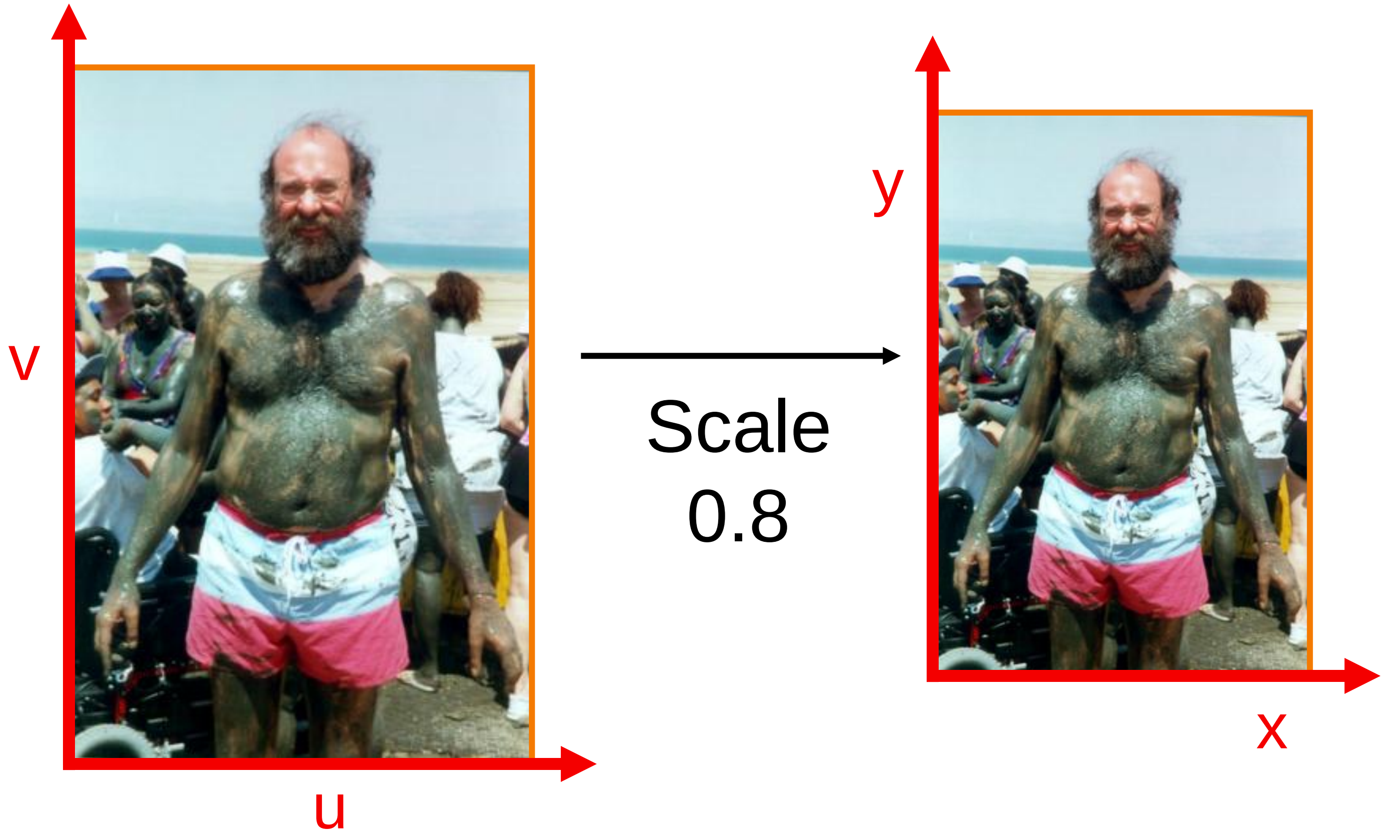
Mapping

- Define transformation
  - Describe the destination (x,y) for every source (u,v) (vice-versa, if reverse mapping)



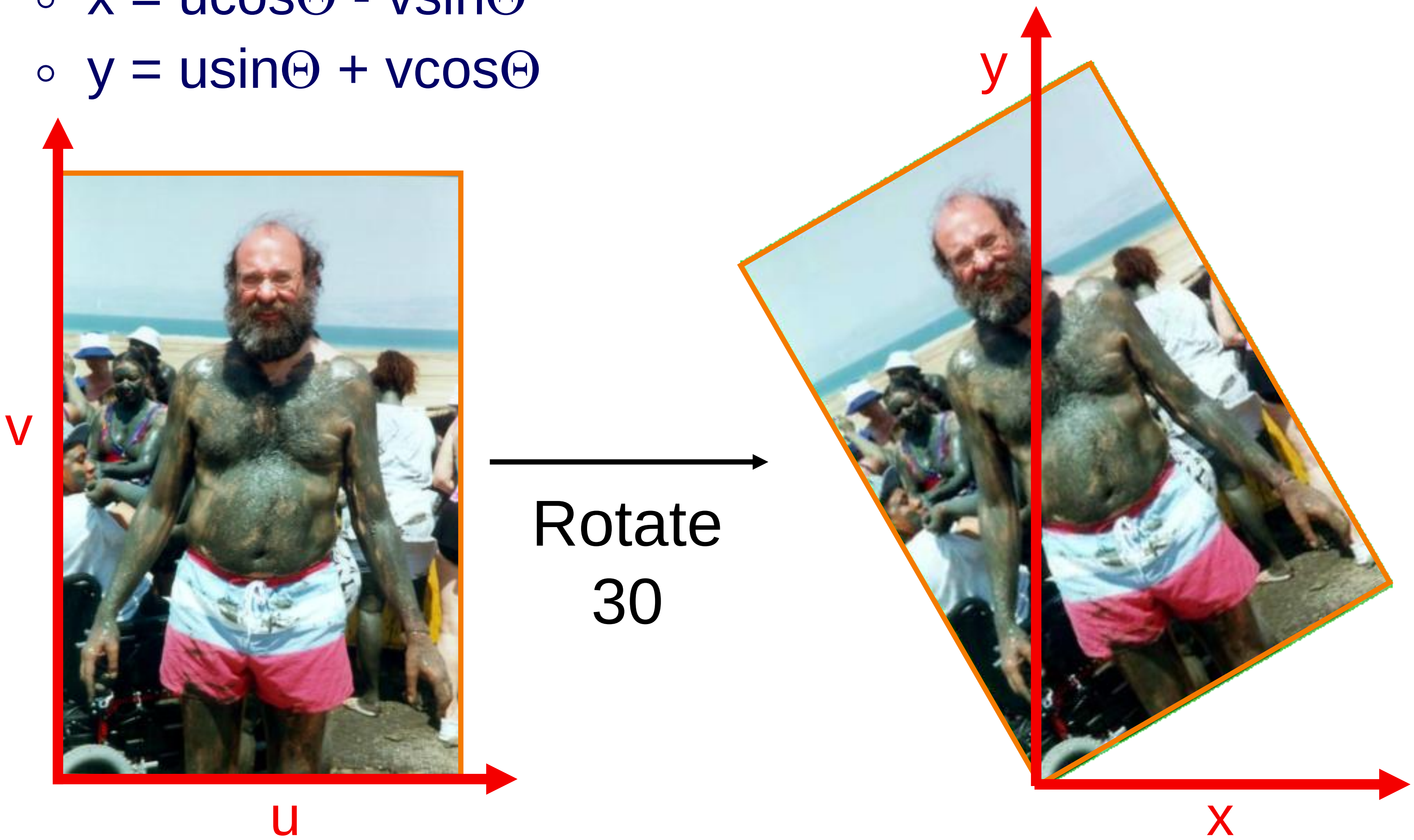
Parametric Mappings

- Scale by *factor*:
  - x = *factor* * u
  - y = *factor* * v



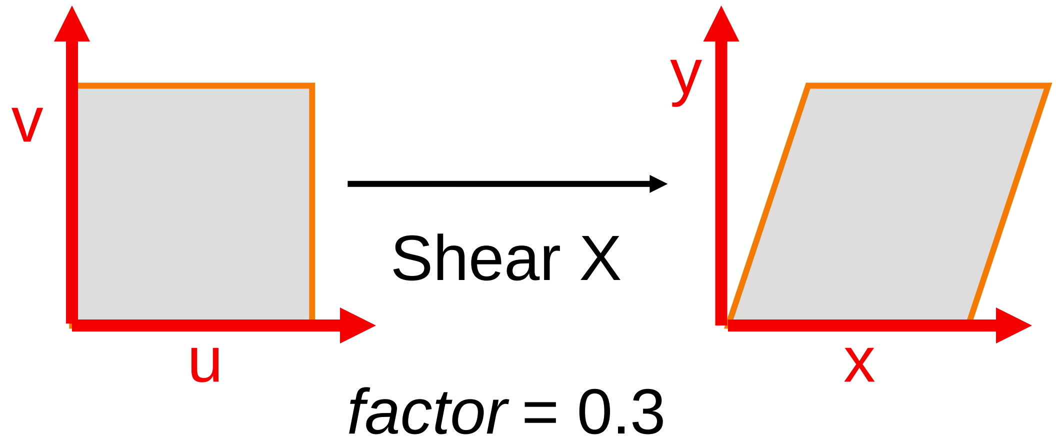
Parametric Mappings

- Rotate by Θ degrees:
  - x = ucosΘ - vsinΘ
  - y = usinΘ + vcosΘ

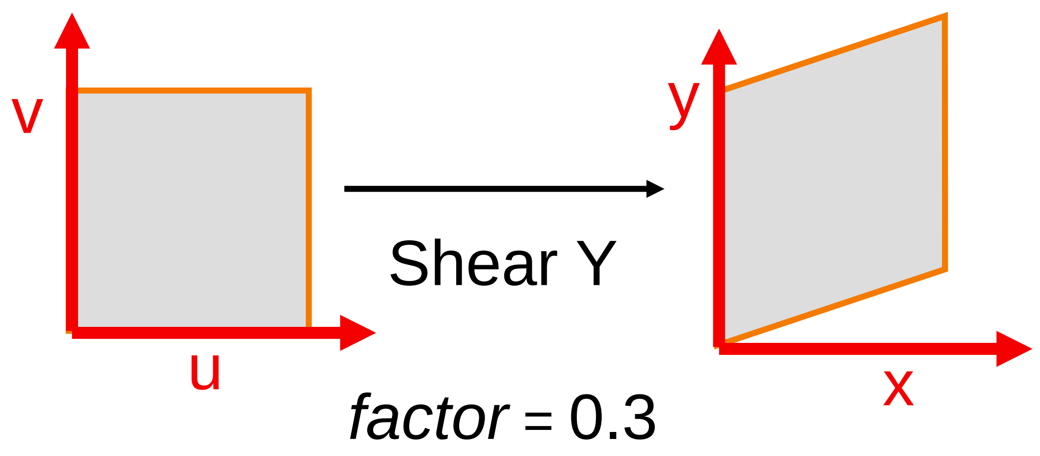


Parametric Mappings

- Shear in X by *factor:*
  - x = u + *factor* * v
  - y = v

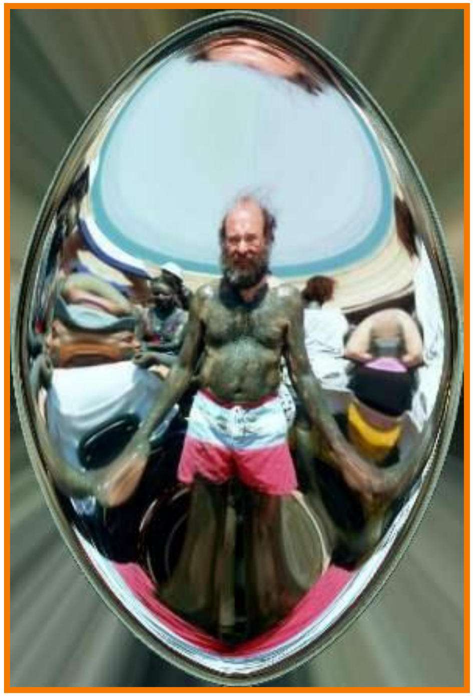


- Shear in Y by *factor:*
  - x = u
  - y = v + *factor* * u



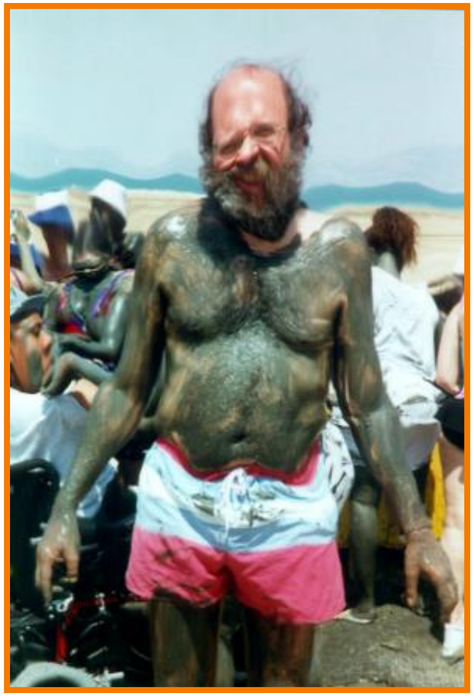
Other Parametric Mappings

- Any function of u and v:
  - $x = f_x(u,v)$
  - $y = f_y(u,v)$



Fish-eye

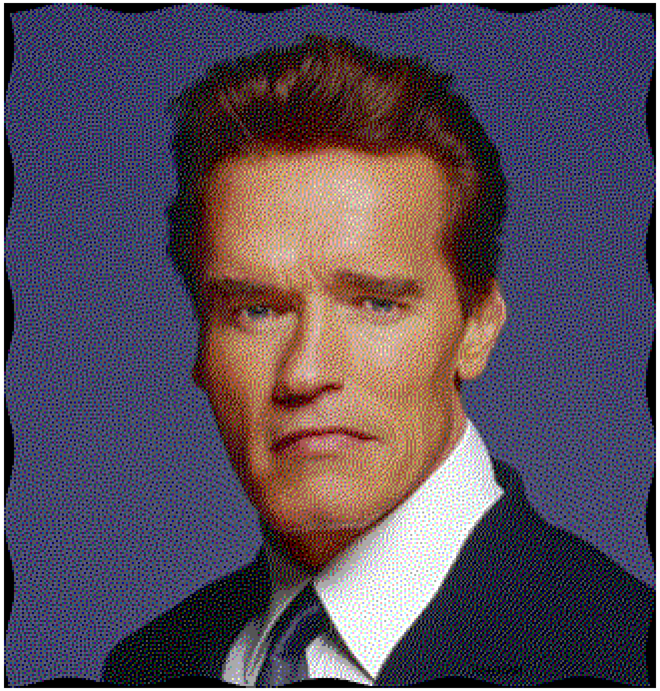
"Swirl"



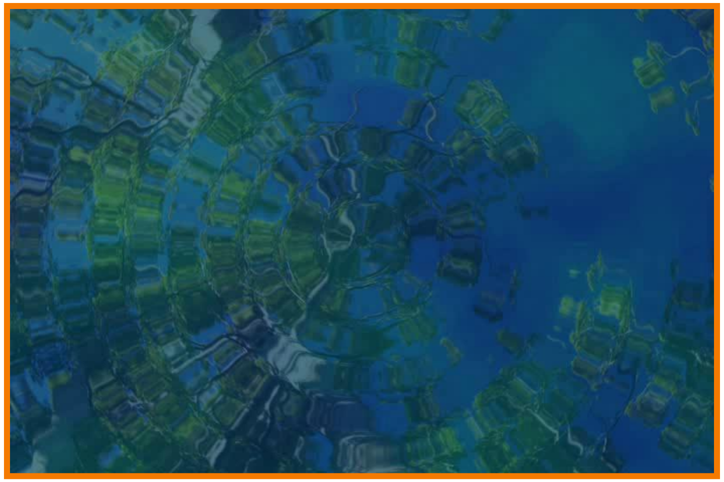
"Rain"

Animated COS426 Examples

Sid Kapur



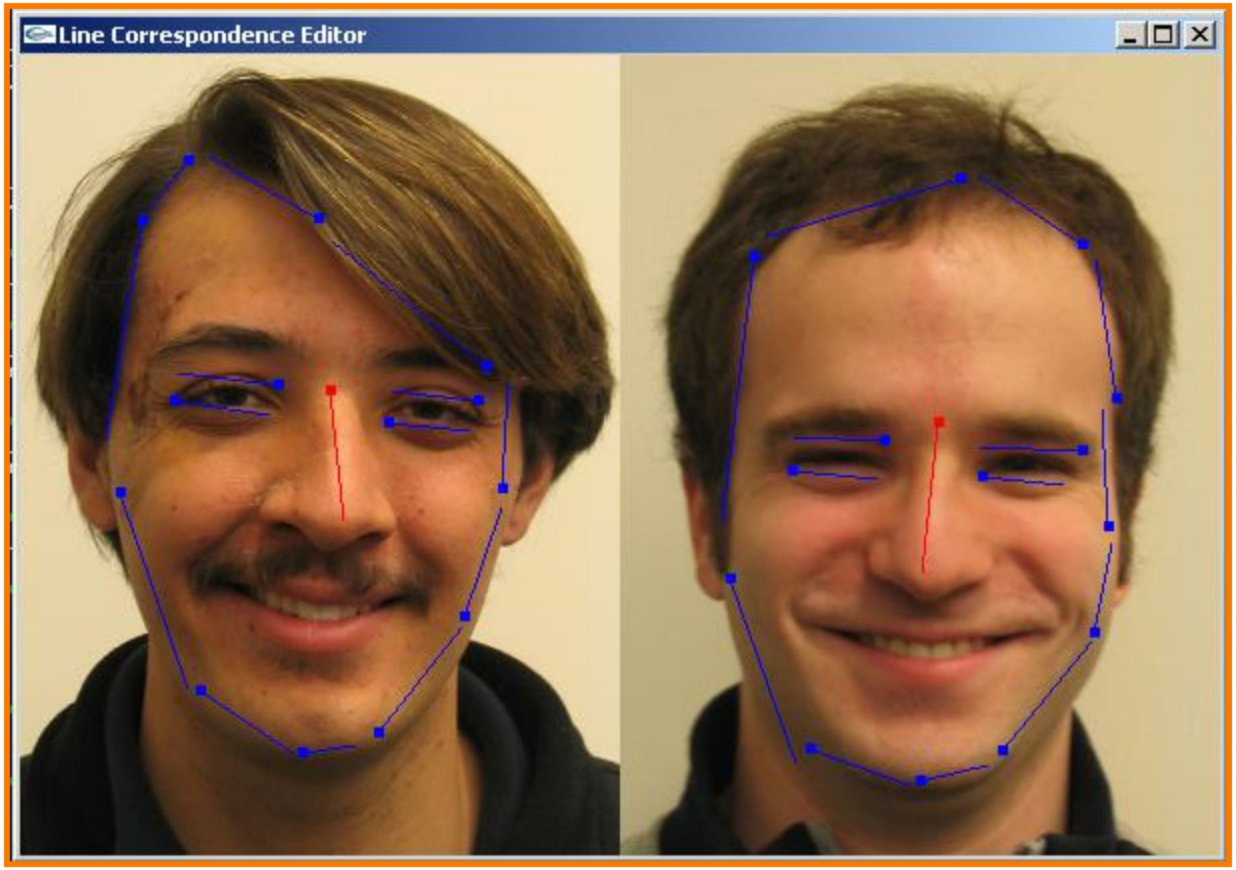
Eirik Bakke



Michael Oranato

Line Correspondence Mappings

[Beier&Neeley'92] use pairs of lines to specify warp



(more on this in next lecture)

Implies Correspondence Mappings

- Mappings implied by correspondences:
  - A ↔ A'
  - B ↔ B'
  - C ↔ C'

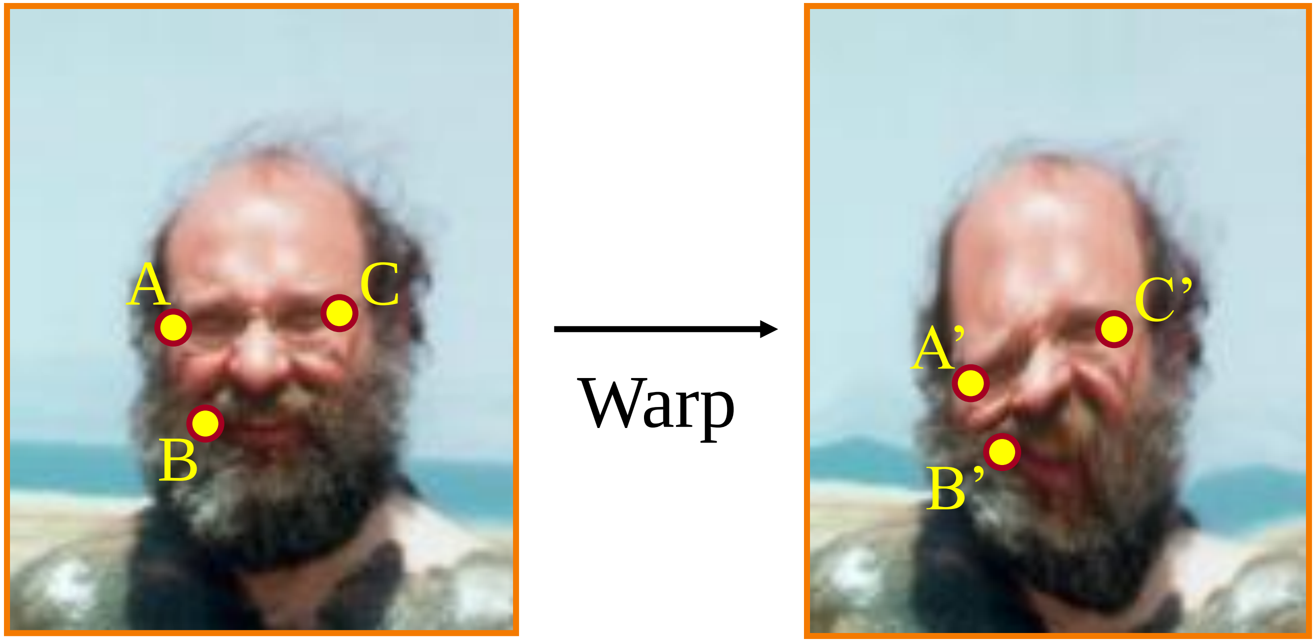
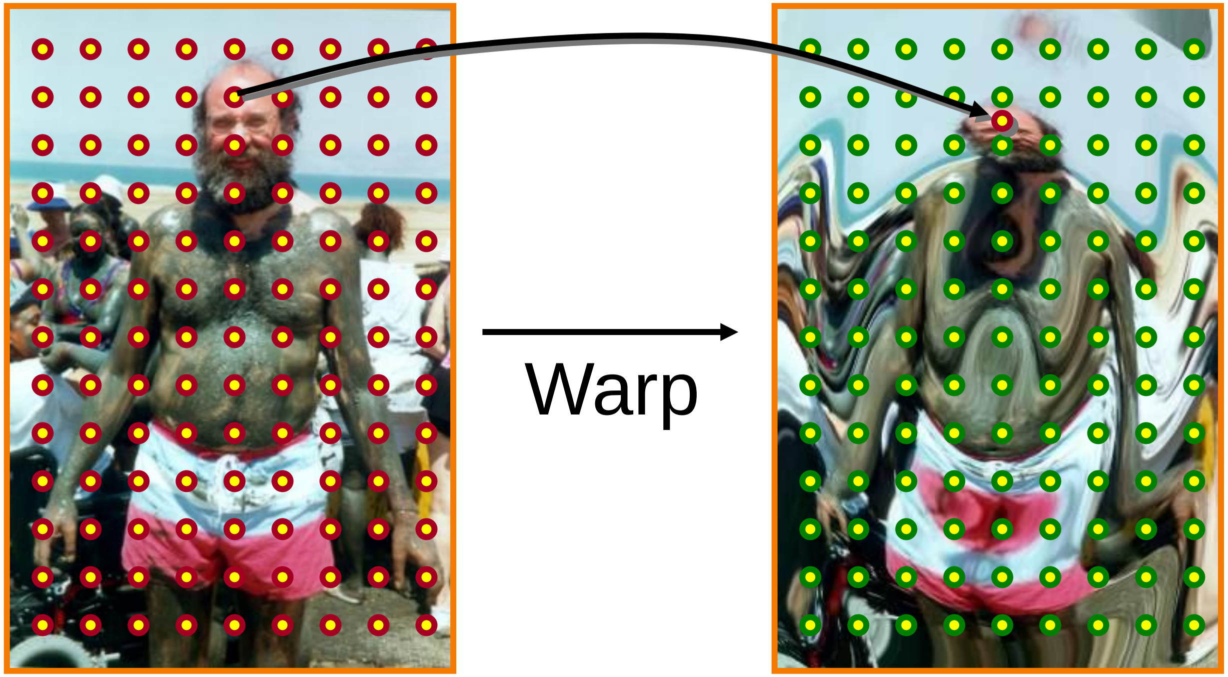


Image Warping

- Issues:
  - Specifying where every pixel goes (mapping)
  - Computing colors at destination pixels (resampling)



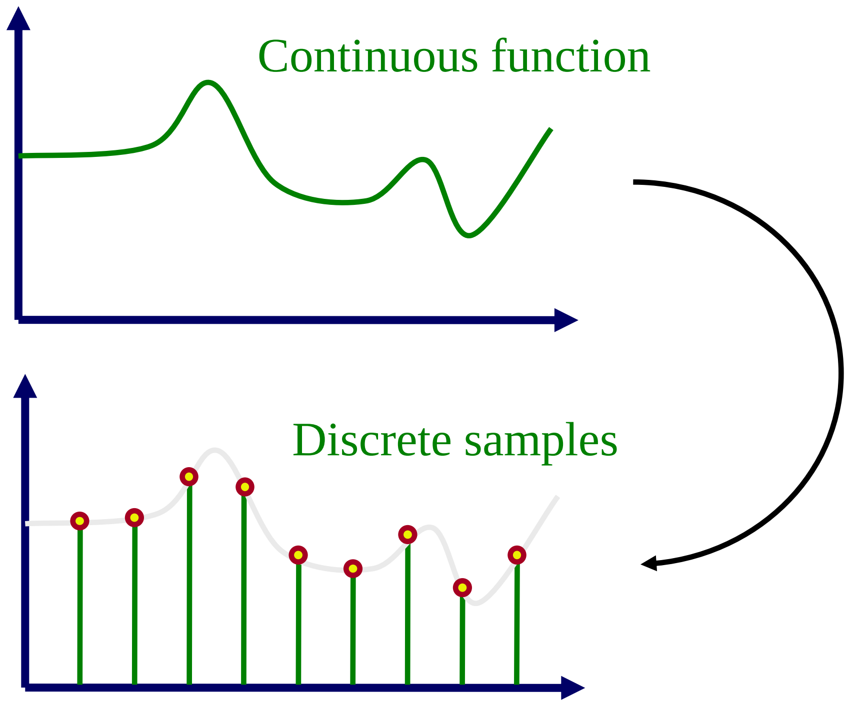
Source image

Destination image

Digital Image Processing

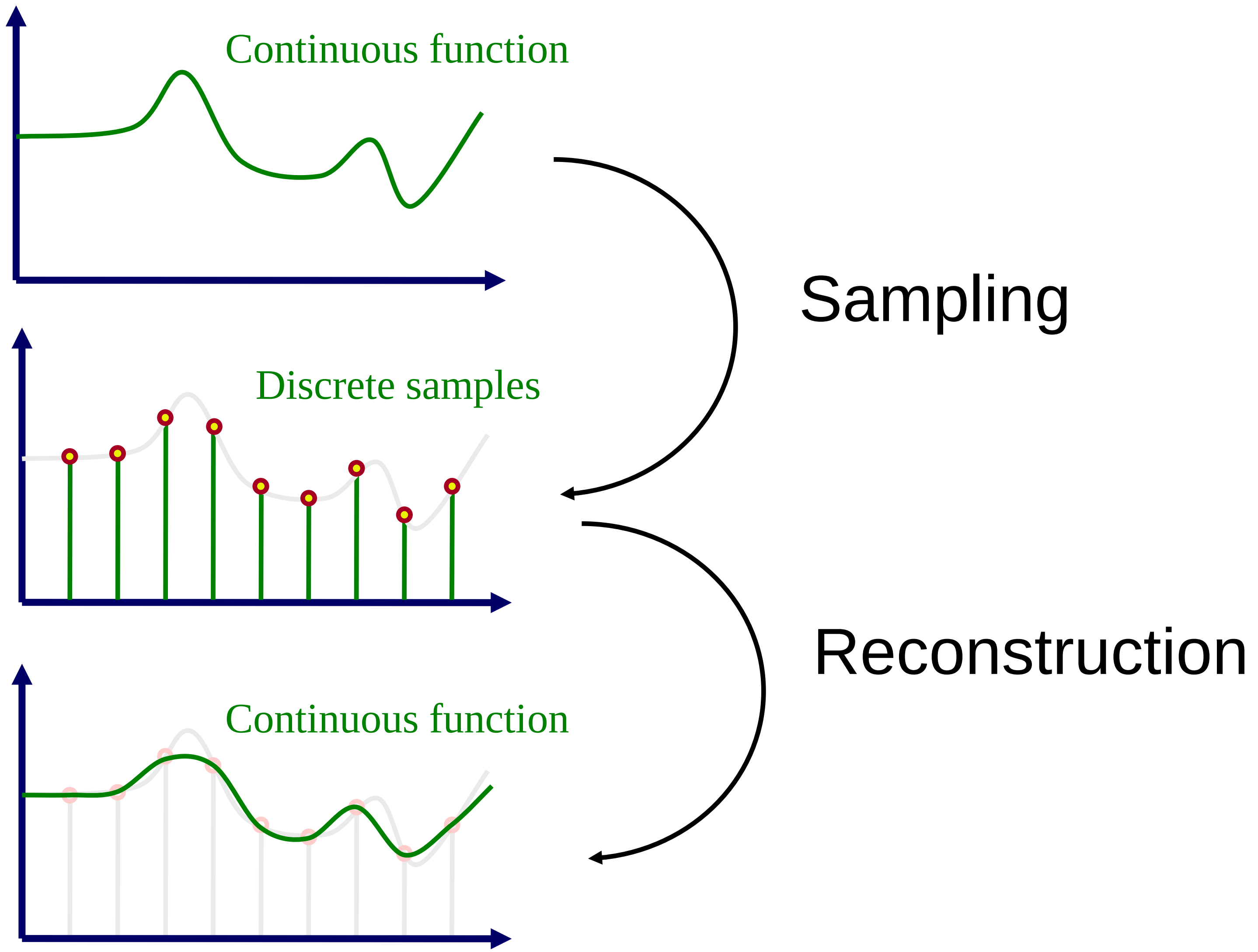
When implementing operations that move pixels, must account for the fact that digital images are sampled versions of continuous ones

Sampling and Reconstruction



Sampling

Sampling and Reconstruction



2D Example

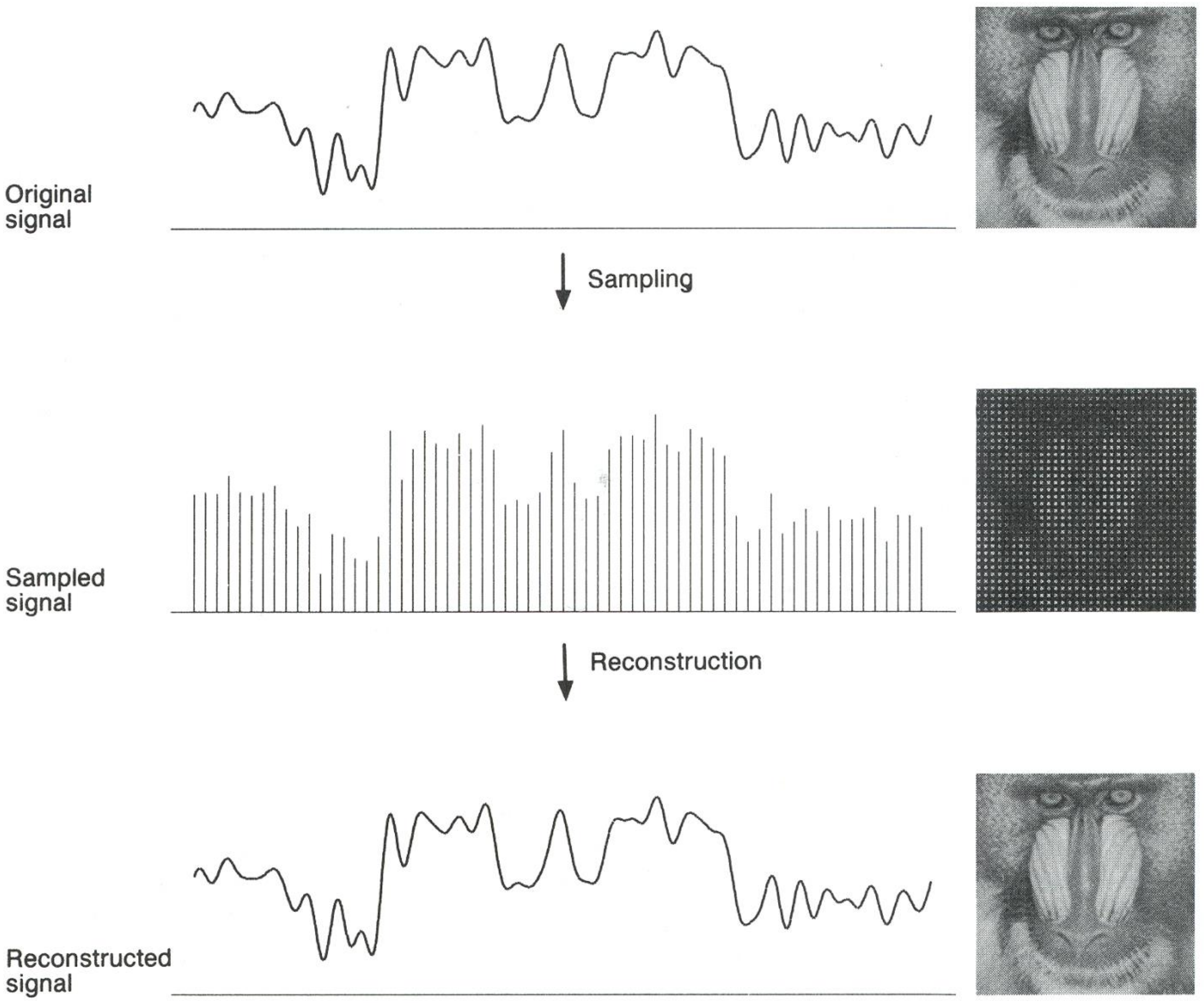


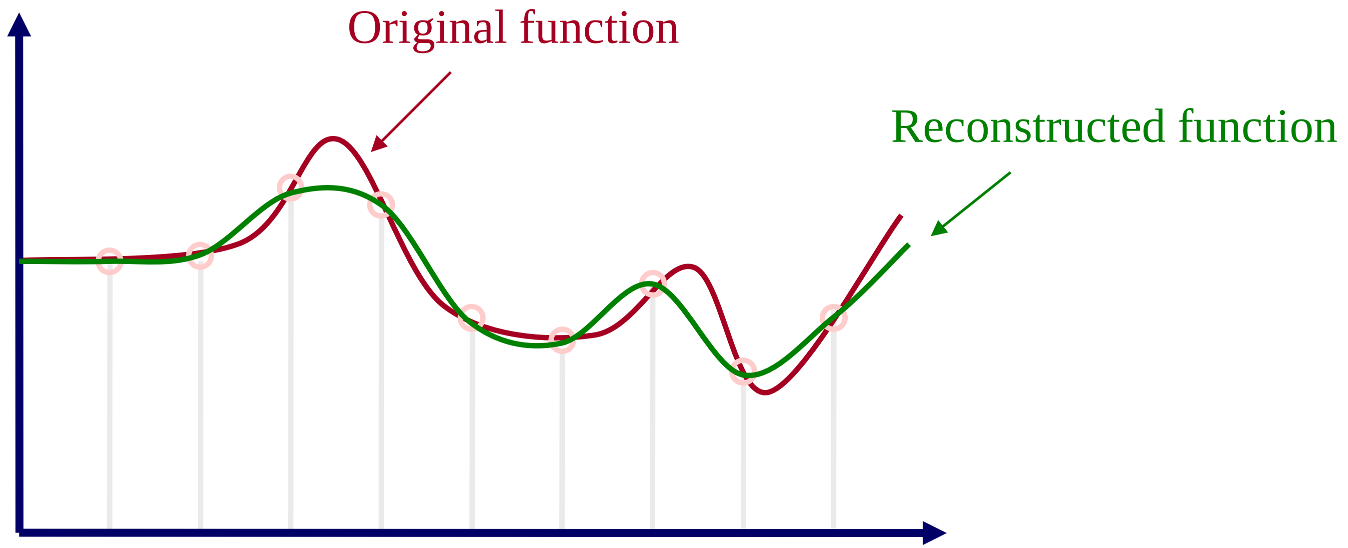
Figure 19.9 FvDFH

Sampling Theory

How many samples are enough?

- How many samples needed to represent a signal?
- What can be reconstructed for a given sampling rate?

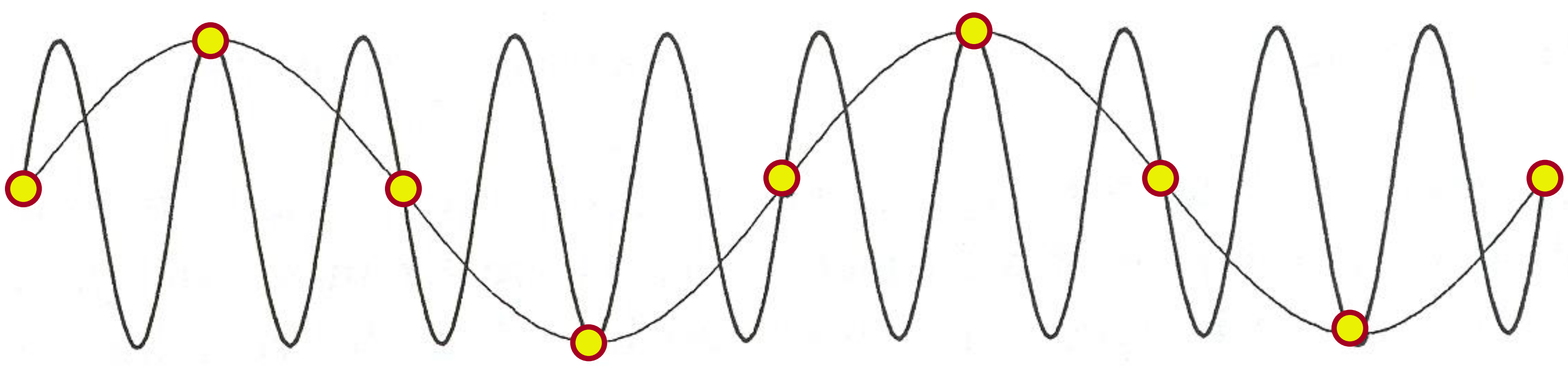
What happens when we use too few samples?



Sampling Theory

What happens when we use too few samples?

- **Aliasing:** high frequencies masquerade as low ones



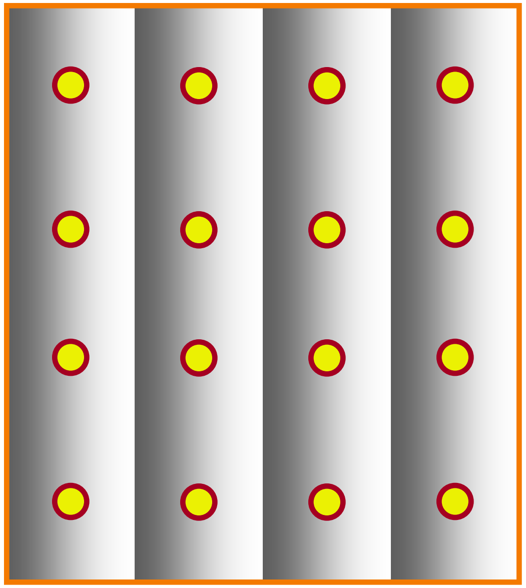
Specifically, in graphics:

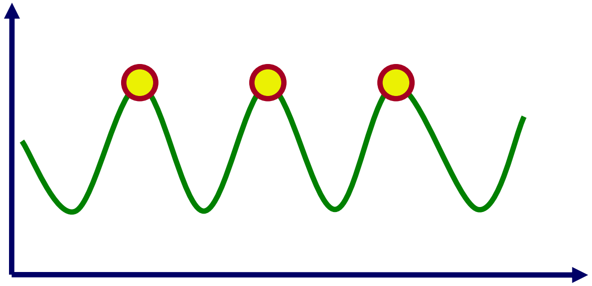
- Spatial aliasing
- Temporal aliasing

Figure 14.17 FvDFH

Spatial Aliasing

Artifacts due to limited spatial resolution

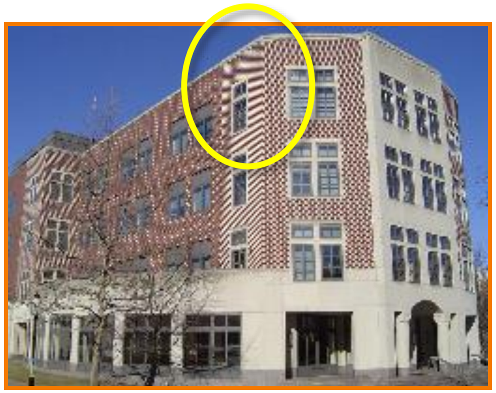




Aliased Frequencies

Artifacts due to limited spatial resolution

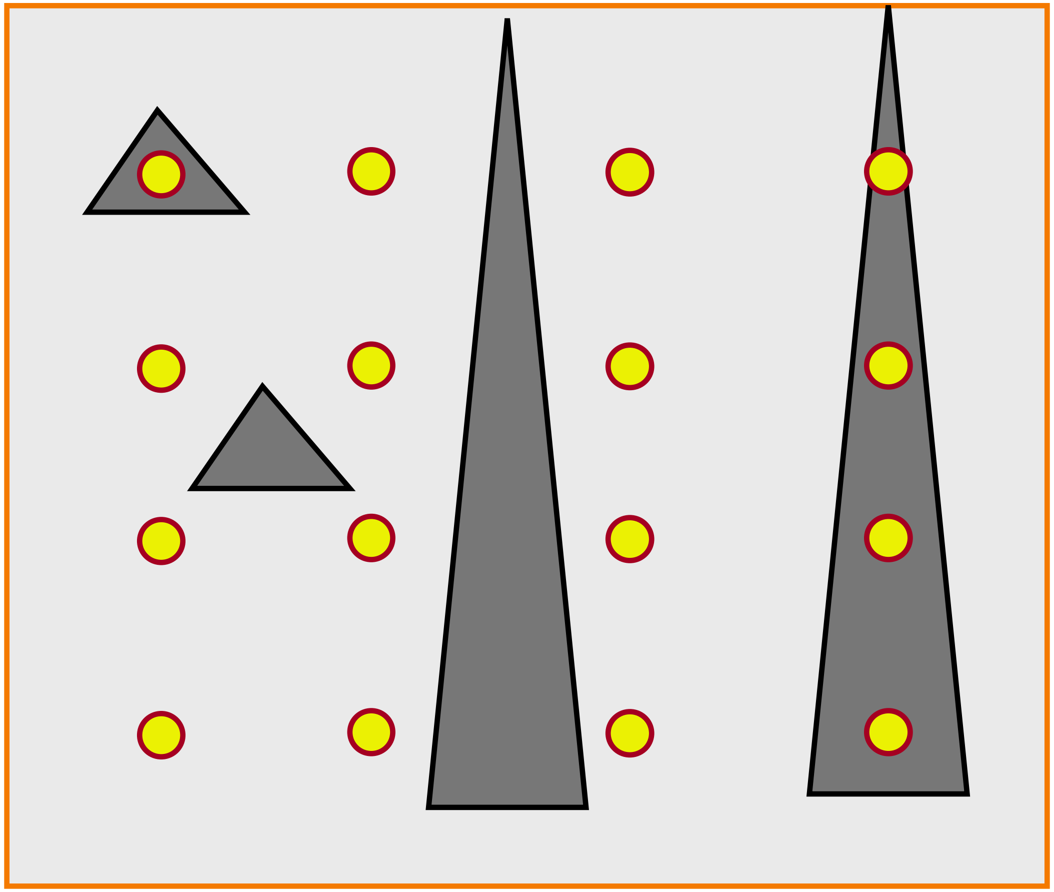
(Barely) adequate sampling



Inadequate sampling

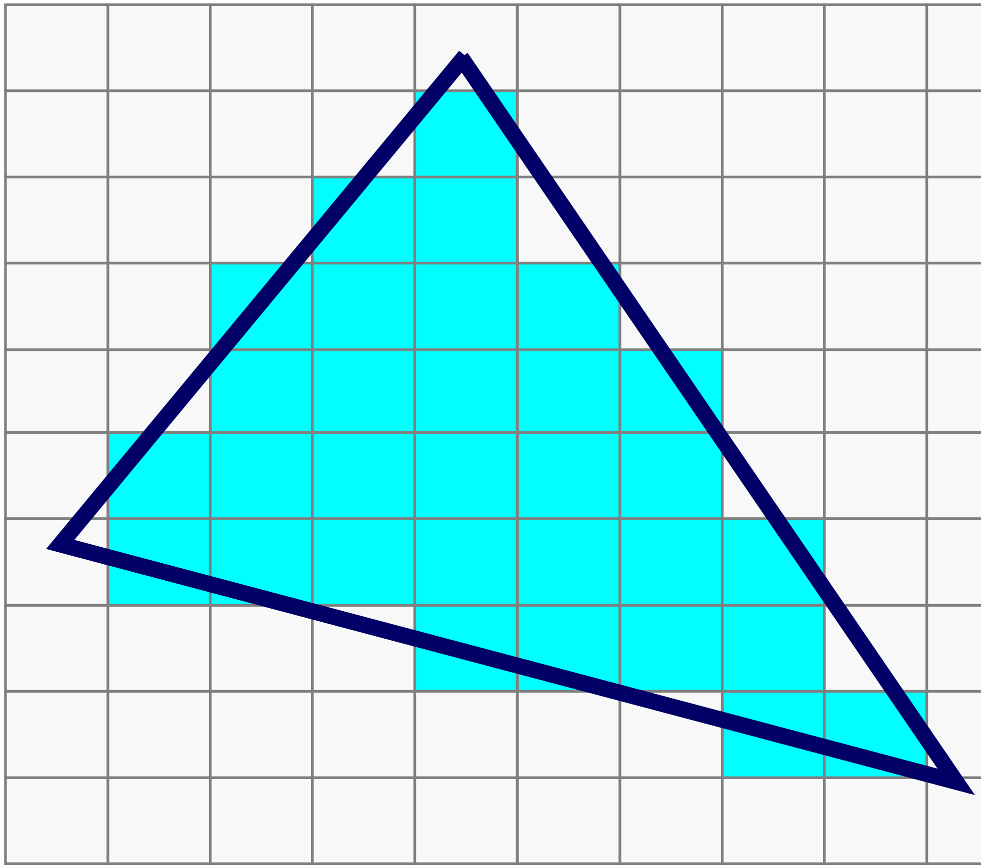
Missing Image Detail

Artifacts due to limited spatial resolution



Missing Image Detail

Artifacts due to limited spatial resolution

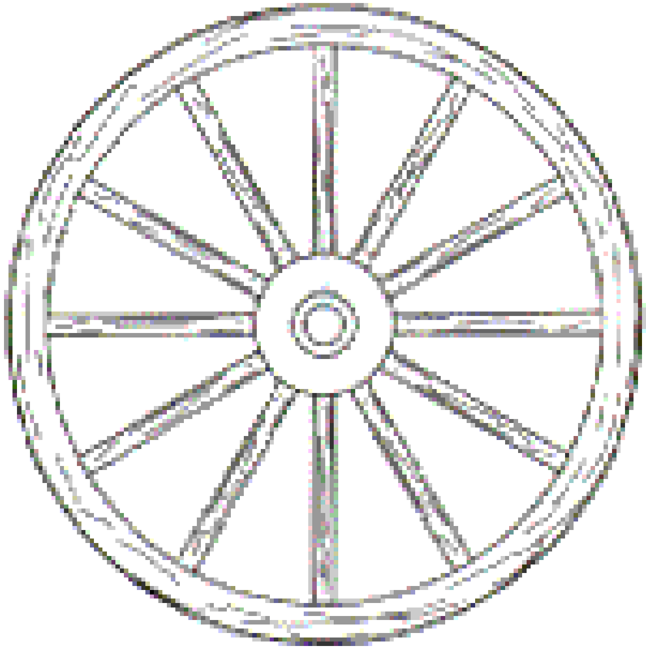


“Jaggies”

Temporal Aliasing

Artifacts due to limited temporal resolution

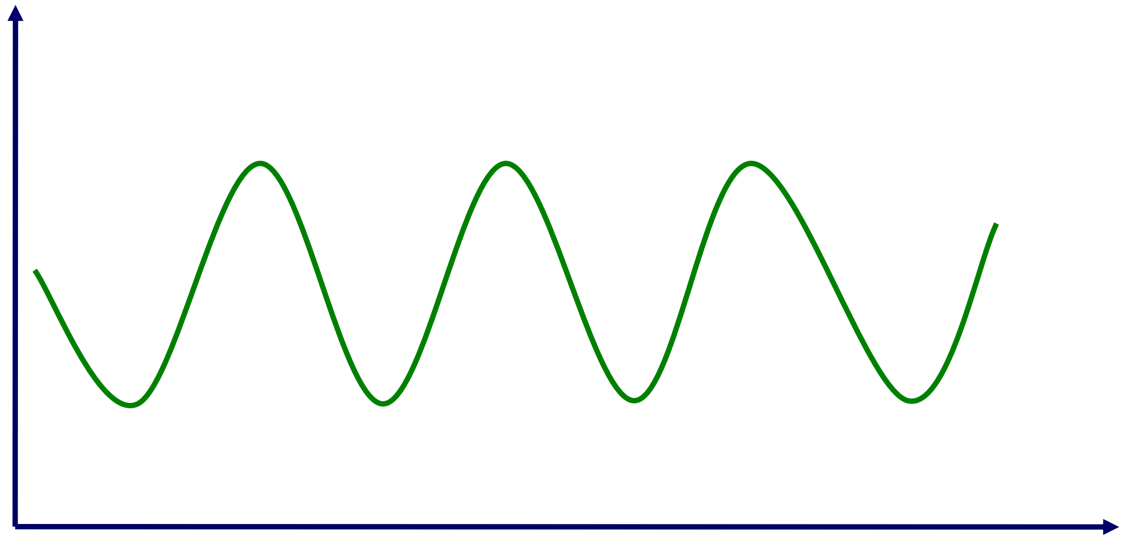
- Strobing (Wagon Wheel Effect)
- Flickering



Sampling Theory

How many samples are enough to avoid aliasing?

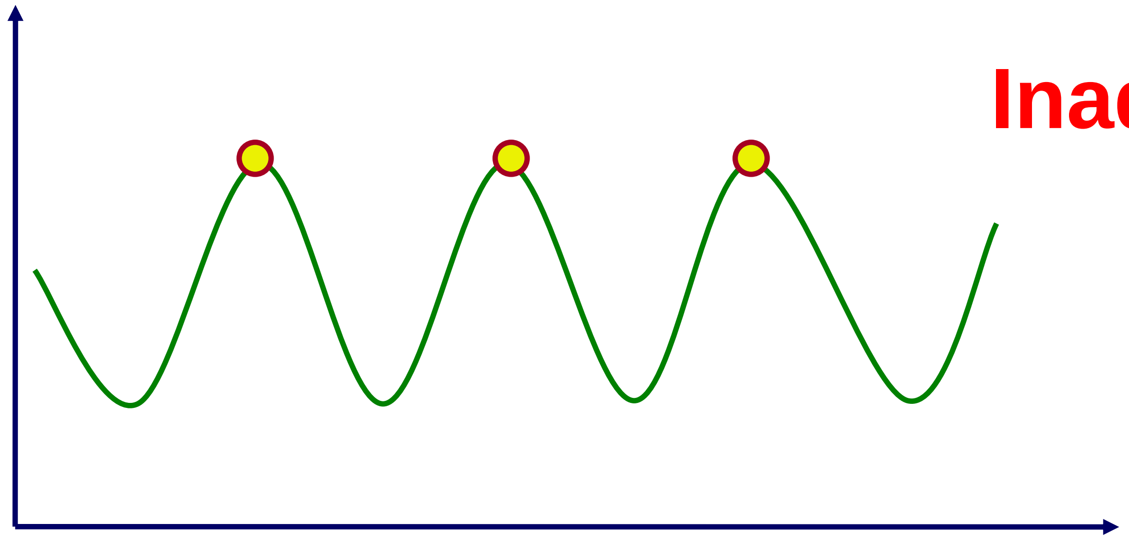
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



Sampling Theory

How many samples are enough to avoid aliasing?

- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?

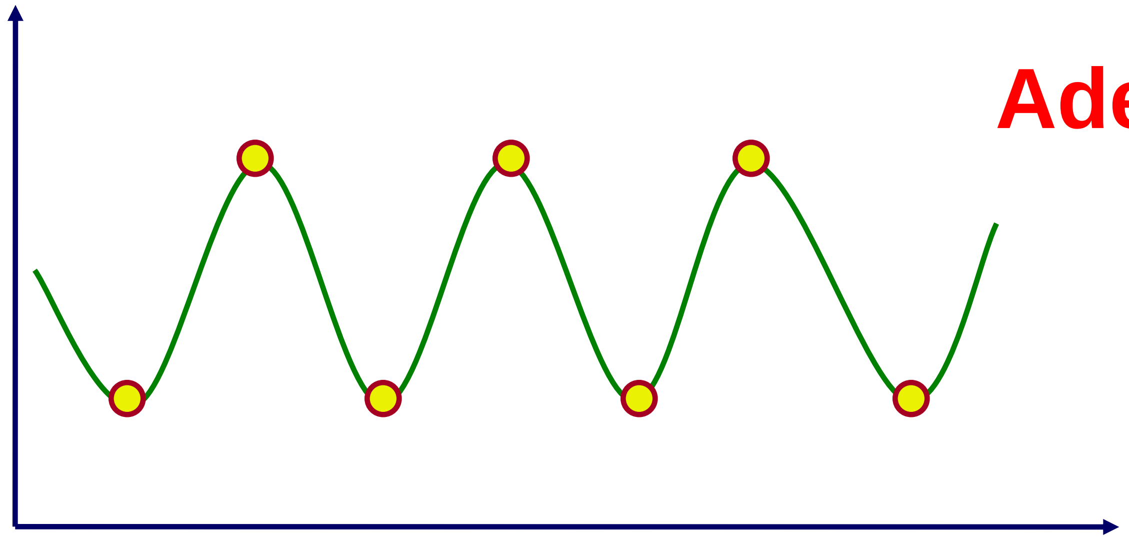


Inadequate

Sampling Theory

How many samples are enough to avoid aliasing?

- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?

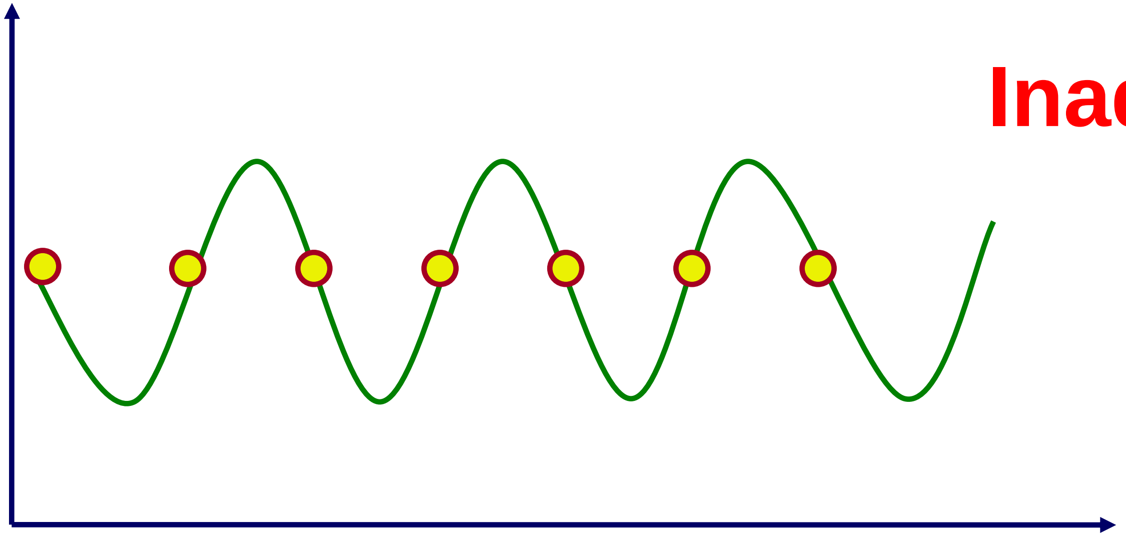


Adequate?

Sampling Theory

How many samples are enough to avoid aliasing?

- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?

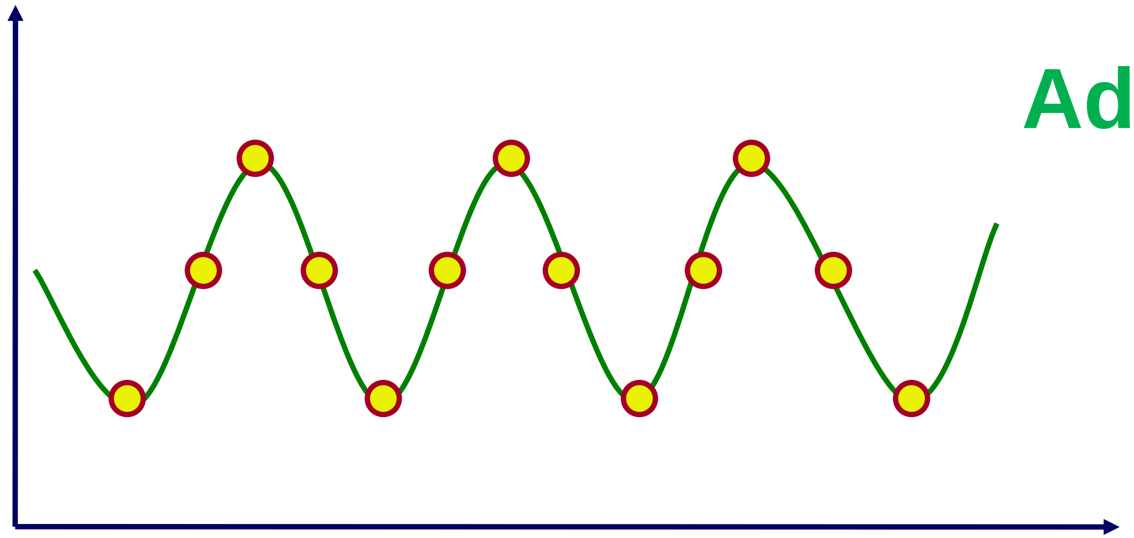


Inadequate

Sampling Theory

How many samples are enough to avoid aliasing?

- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?

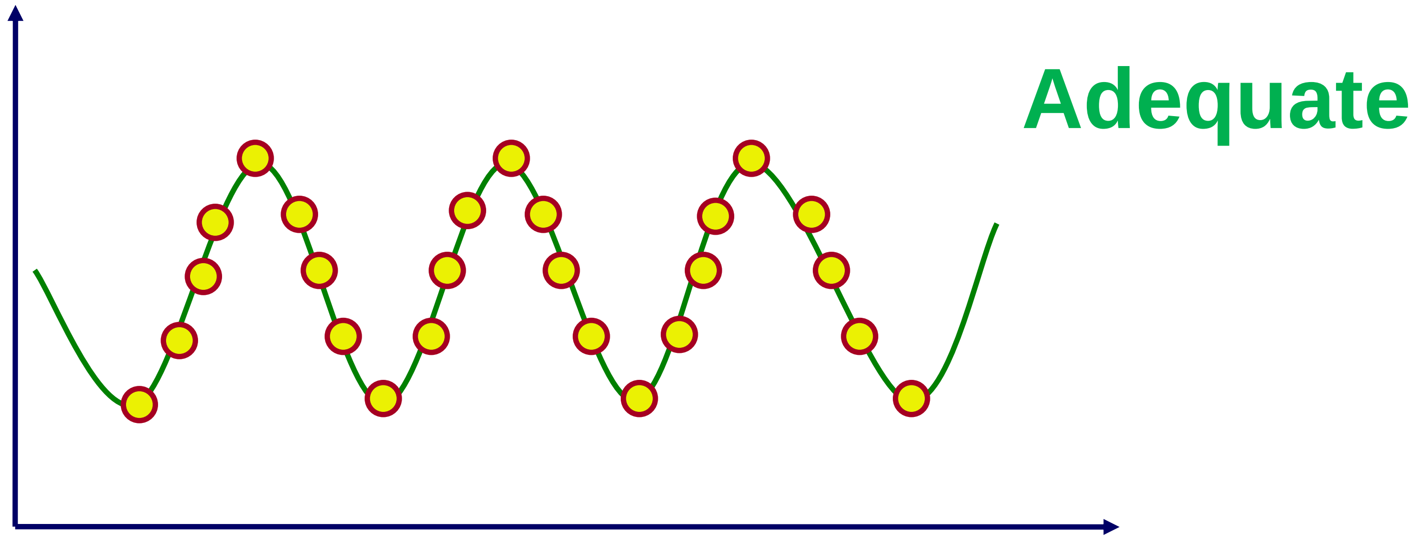


Adequate

Sampling Theory

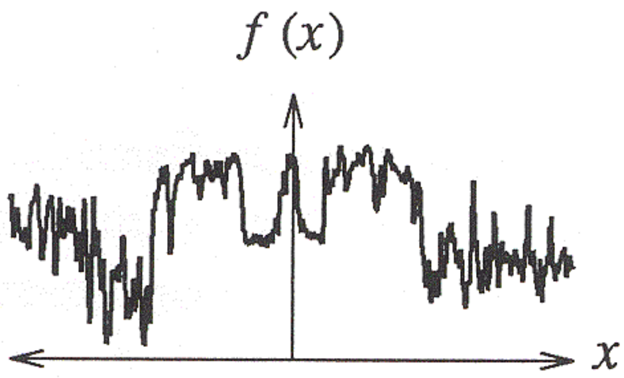
How many samples are enough to avoid aliasing?

- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?

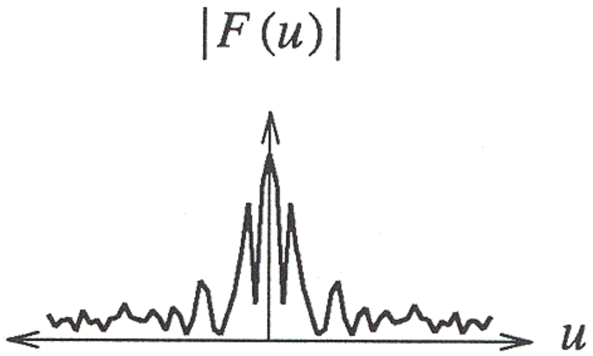


Spectral Analysis

- Spatial domain:
  - Function: f(x)

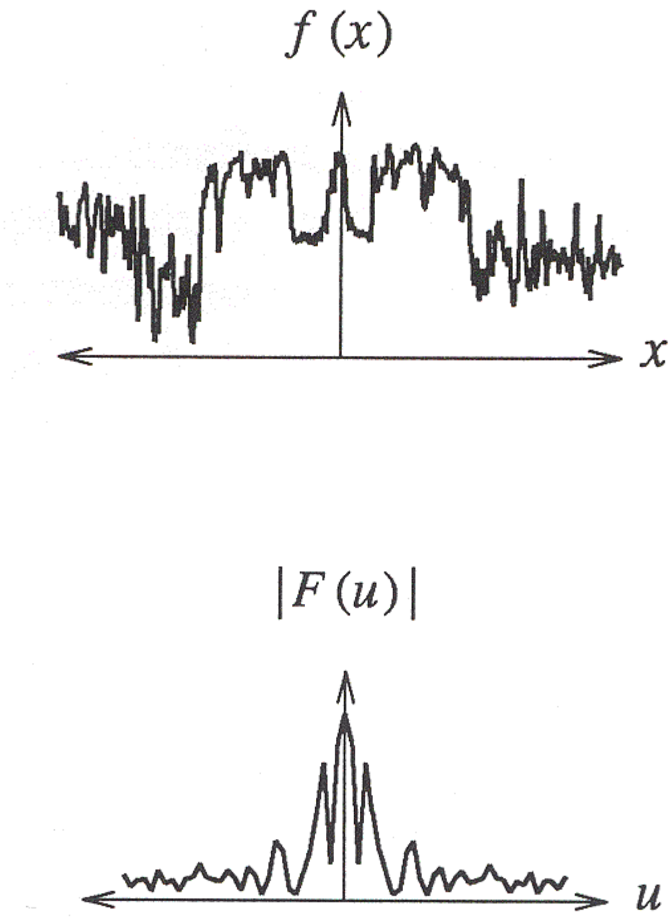


- Frequency domain:
  - Function: F(u)



Any signal can be written as a sum of periodic functions.

Fourier Transform



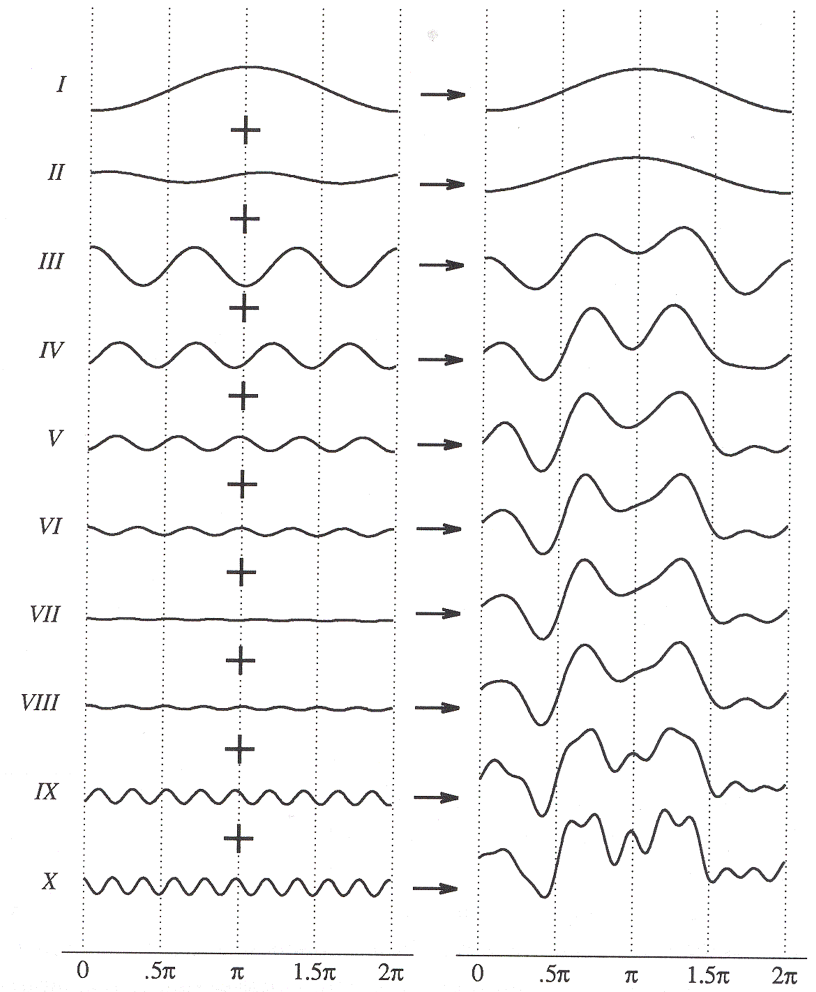


Figure 2.6 Wolberg

Fourier Transform

- Fourier transform:

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi xu} dx$$

- Inverse Fourier transform:

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{+i2\pi ux} du$$

Sampling Theorem

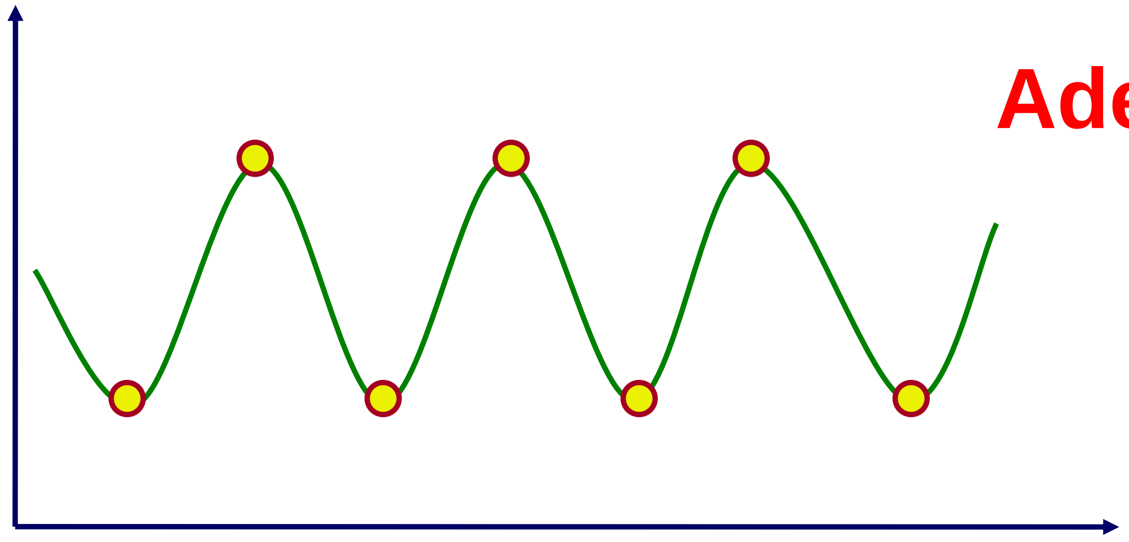
- A signal can be reconstructed from its samples iff it has no content ≥ ½ the sampling frequency
  – Shannon
- The minimum sampling rate for a bandlimited function is called the "Nyquist rate"
- Sampling rate must be > 2 bandwidth.

A signal is *bandlimited* if its highest frequency is bounded. That frequency is called the bandwidth.

Why > ?

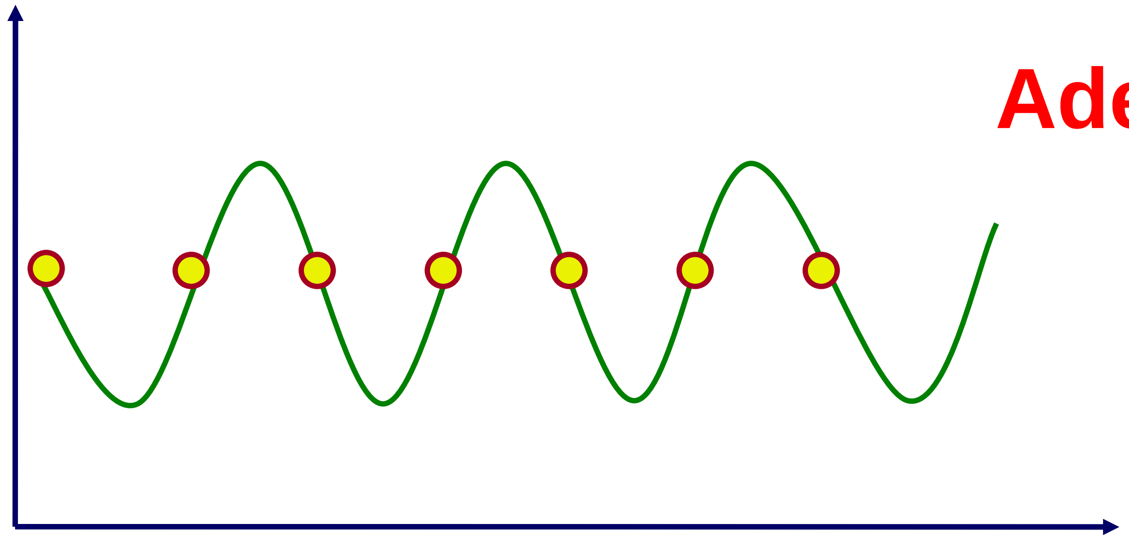
Sampling rate must be > 2 bandwidth

Adequate?



Why > ?

Sampling rate must be > 2 bandwidth



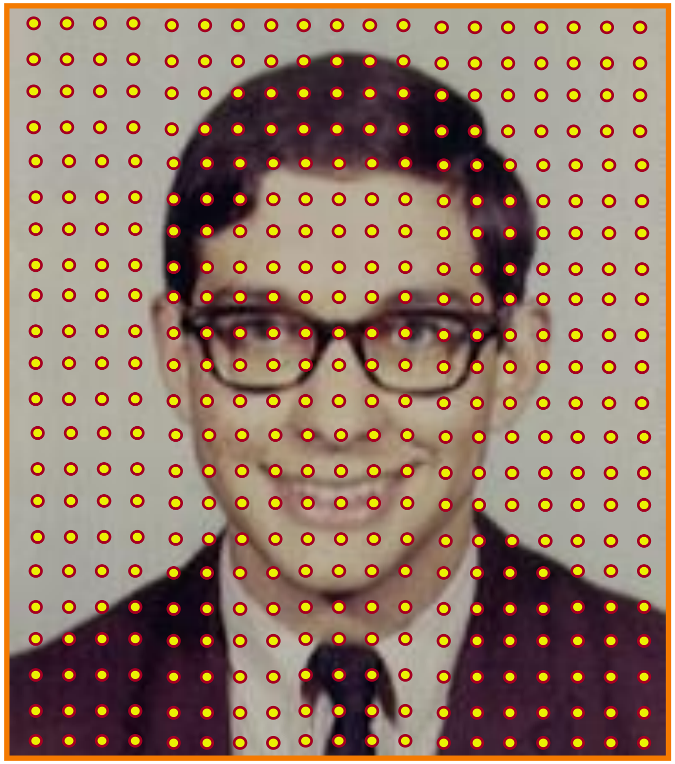
Adequate?

Antialiasing

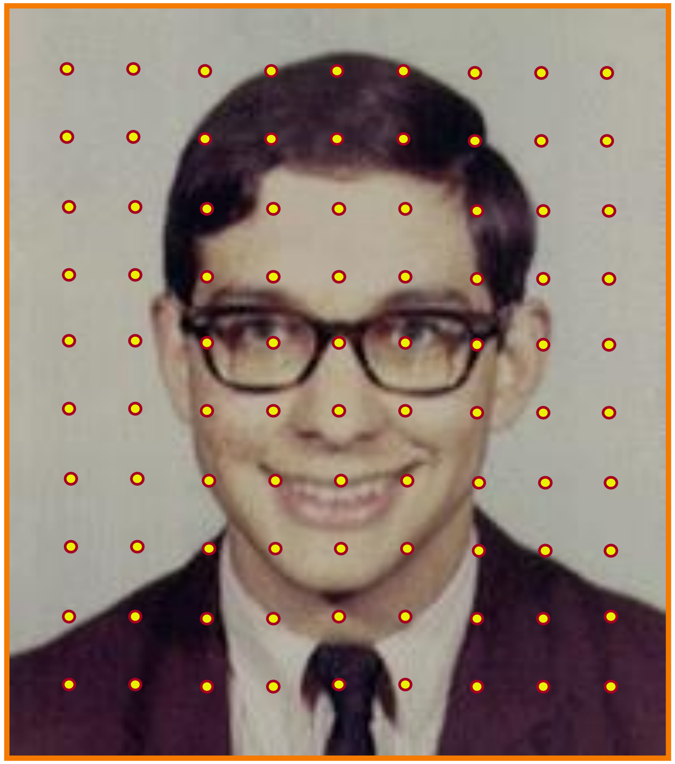
- Option: Sample at higher rate
  - Not always possible
  - Doesn't always solve the problem
- Option: Pre-filter to form bandlimited signal
  - Use low-pass filter to limit signal to < 1/2 sampling rate
  - **Trades blurring for aliasing**

Image Processing

Consider the task of scaling the image
(or, equivalently, reducing resolution)



Original image



1/4 resolution

Image Processing

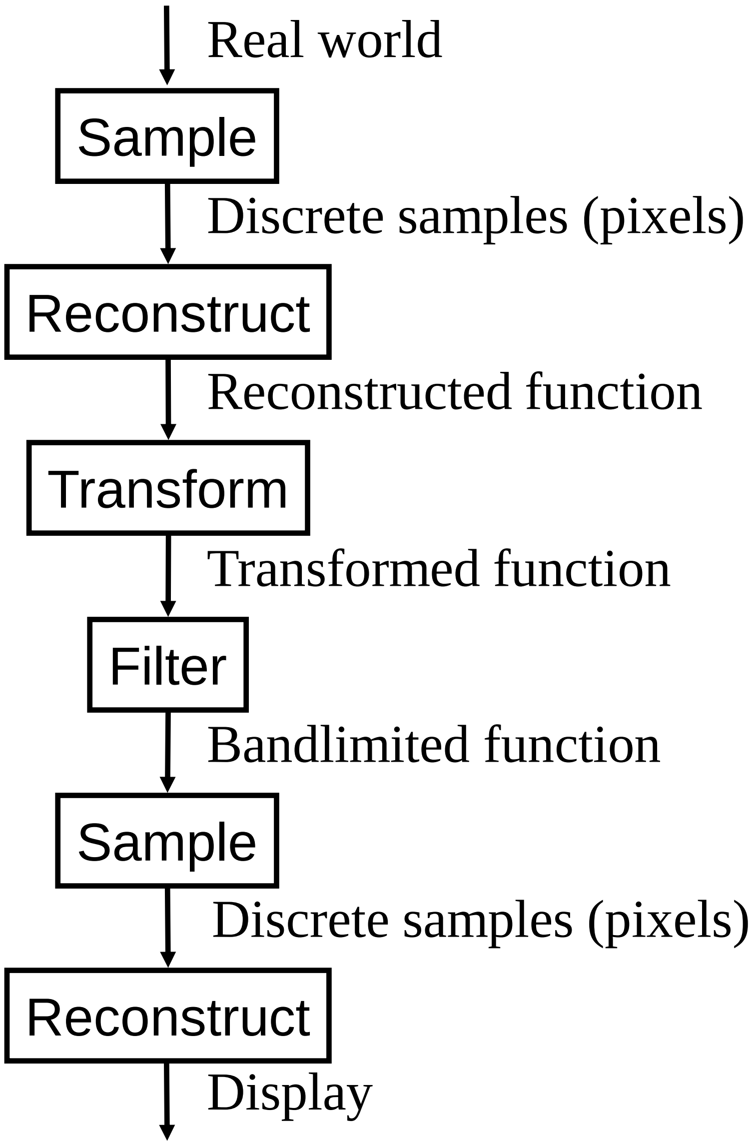


Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

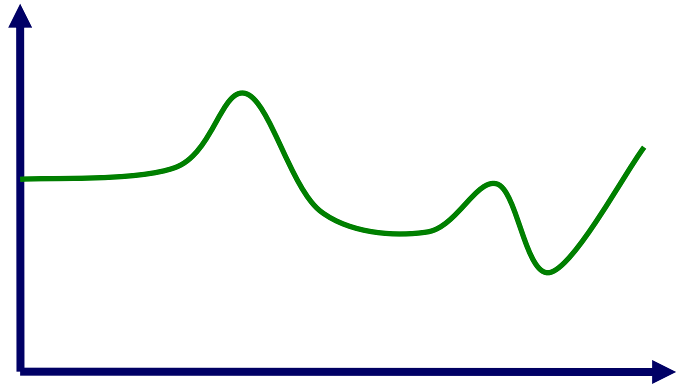
Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display



Continuous Function

Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

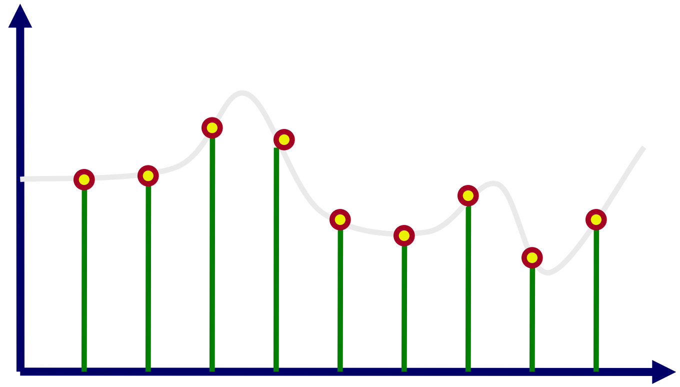
Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display



Discrete Samples

Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

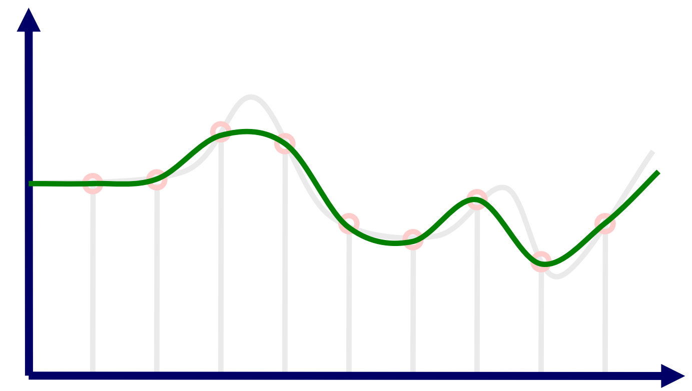
Bandlimited function

Sample

Discrete samples (pixels)

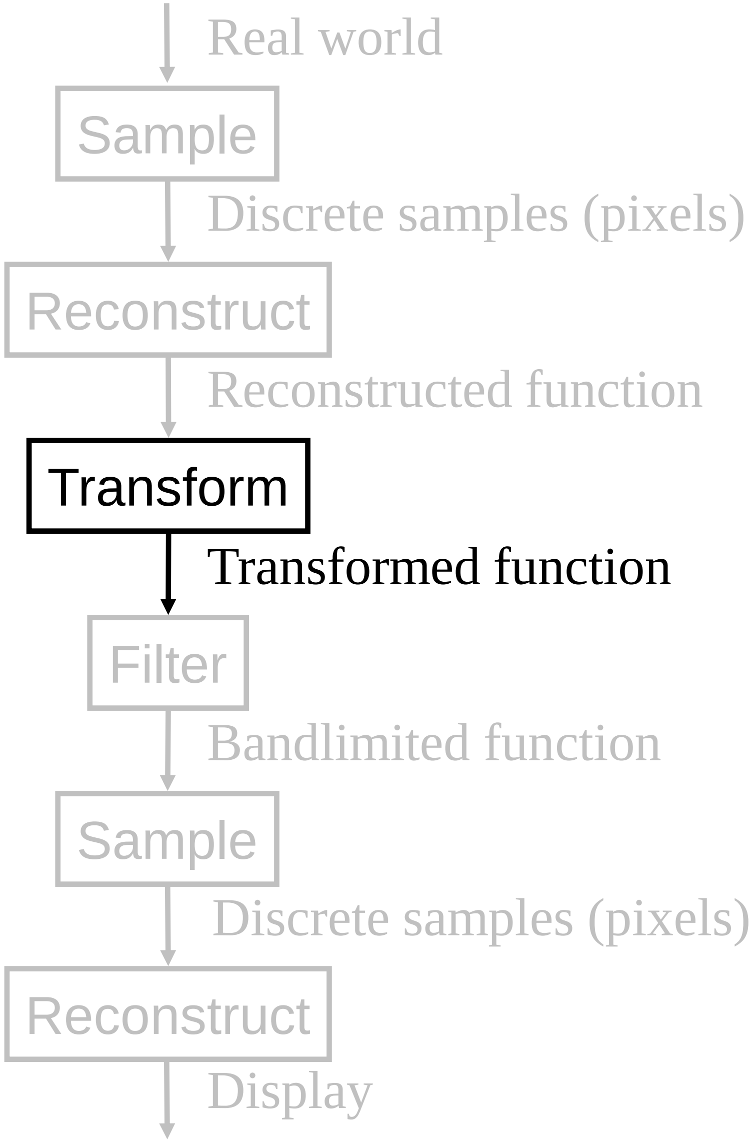
Reconstruct

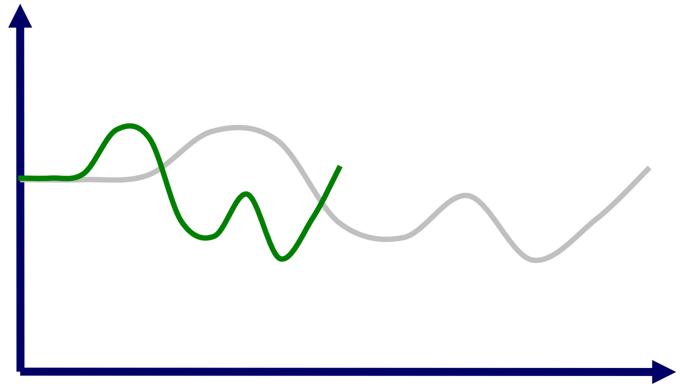
Display



Reconstructed Function

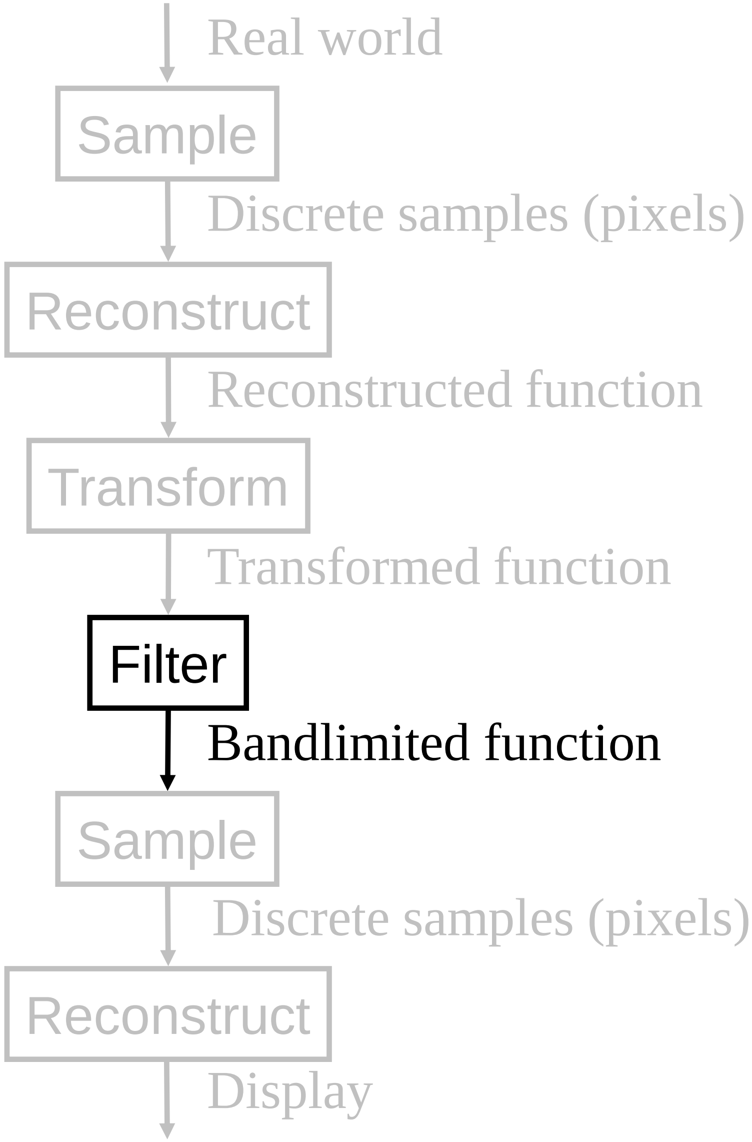
Image Processing

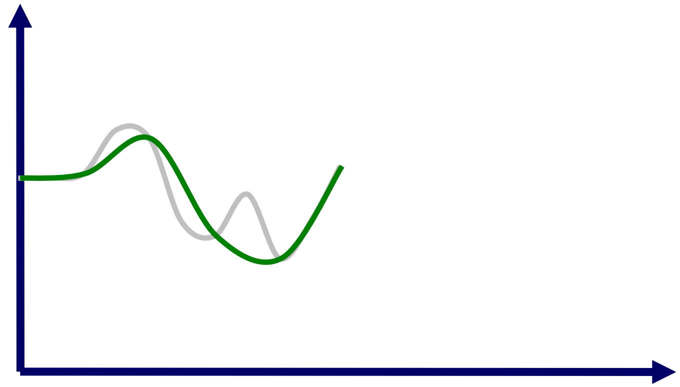




Transformed Function

Image Processing





Bandlimited Function

Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

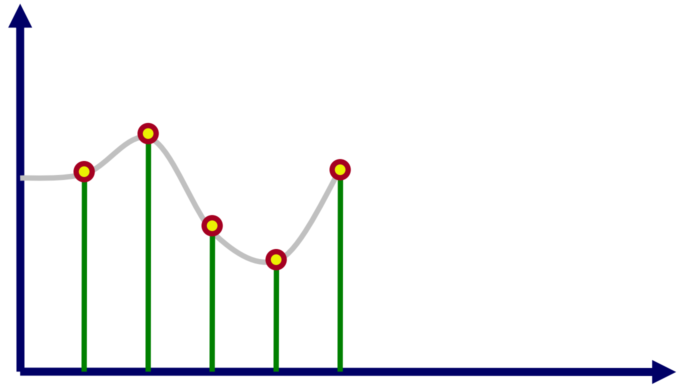
Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display



Discrete samples

Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

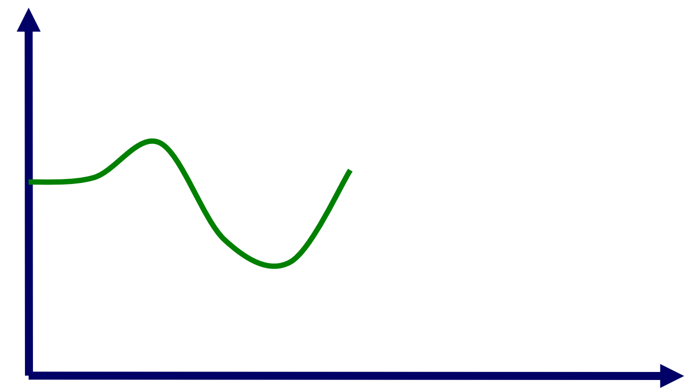
Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display



Display

Image Processing

Real world

↓

Sample

↓ Discrete samples (pixels)

Reconstruct

↓ Reconstructed function

Transform

↓ Transformed function

Filter

↓ Bandlimited function

Sample

↓ Discrete samples (pixels)

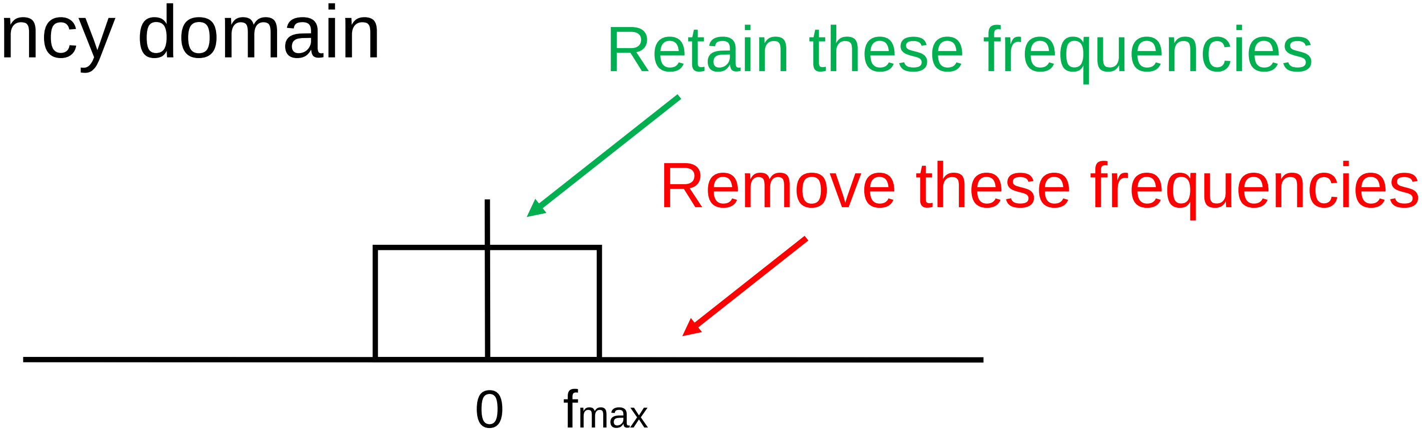
Reconstruct

↓ Display

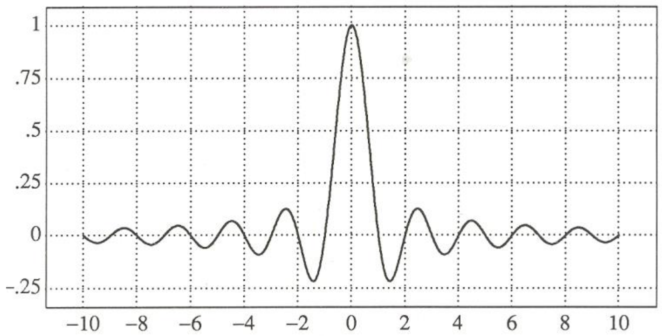
- Reconstruction filter especially important when magnifying
- Bandlimiting filter especially important when minifying

Ideal Image Processing Filter

- Frequency domain



- Spatial domain



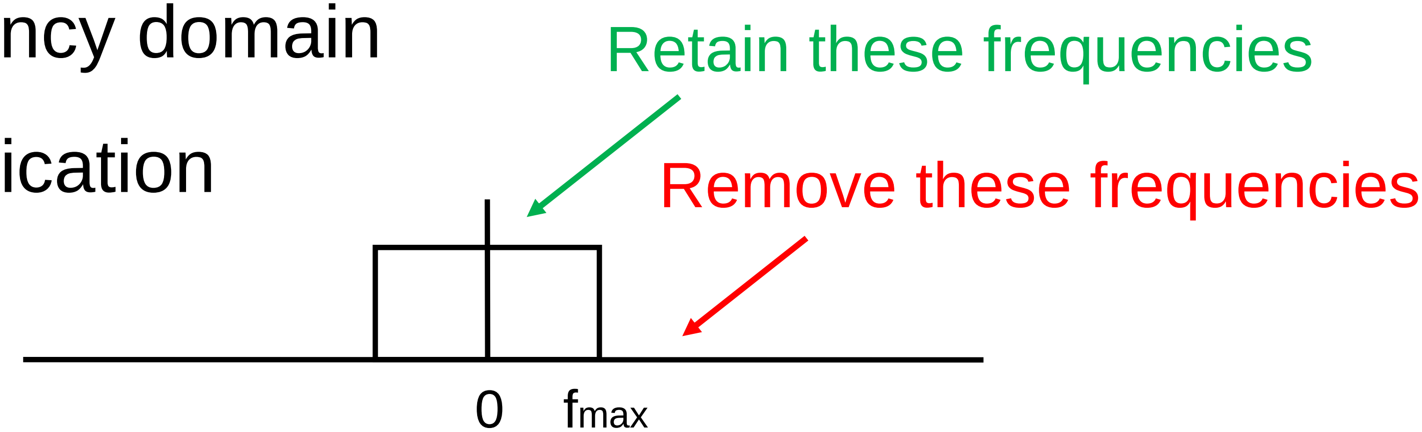
$$Sinc(x) = \frac{\sin \pi x}{\pi x}$$

Figure 4.5 Wolberg

Ideal Image Processing Filter

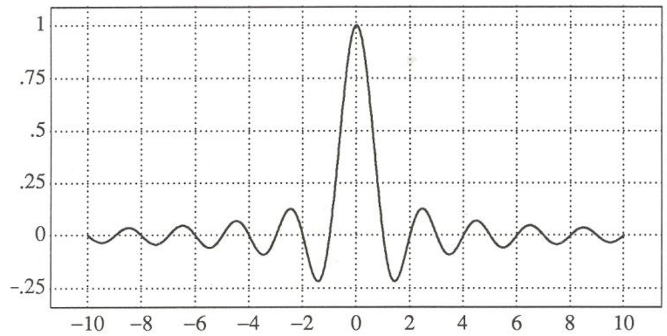
- Frequency domain

→ multiplication



- Spatial domain

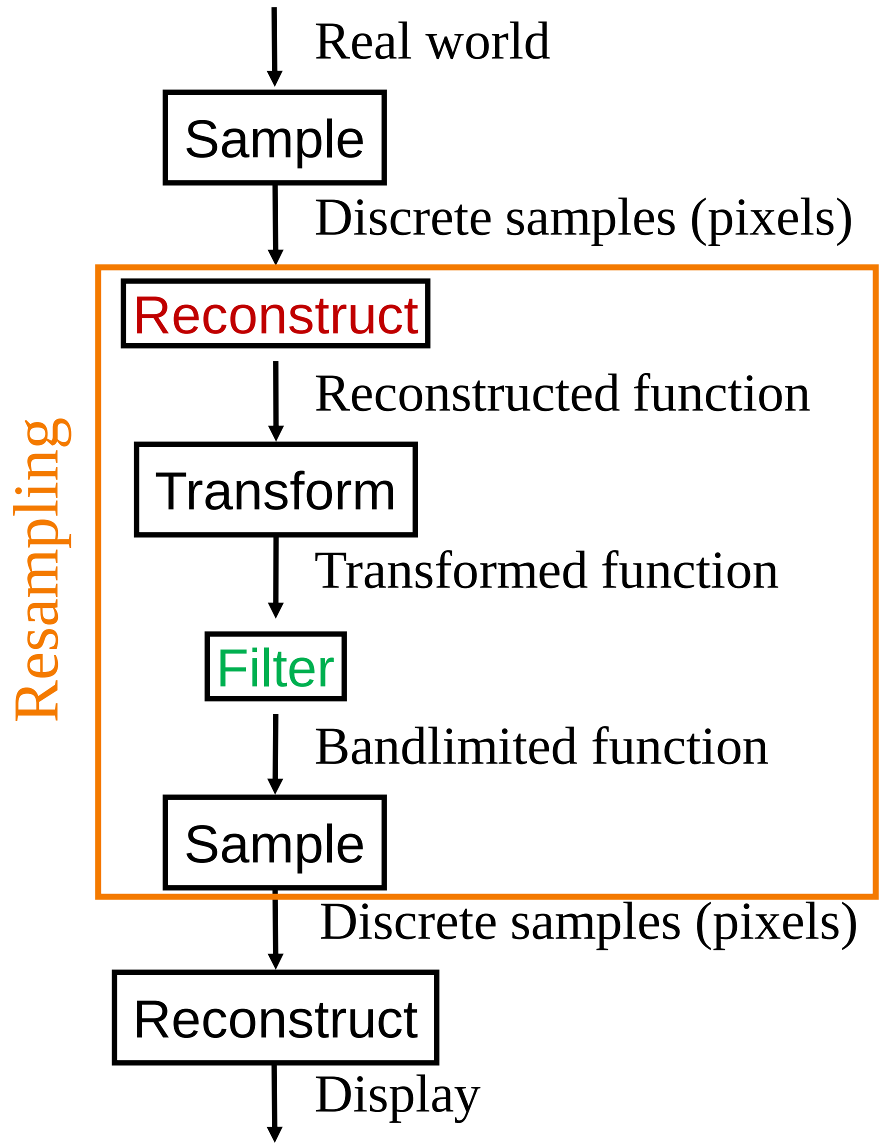
→ convolution!



$$Sinc(x) = \frac{\sin \pi x}{\pi x}$$

Figure 4.5 Wolberg

Practical Image Processing

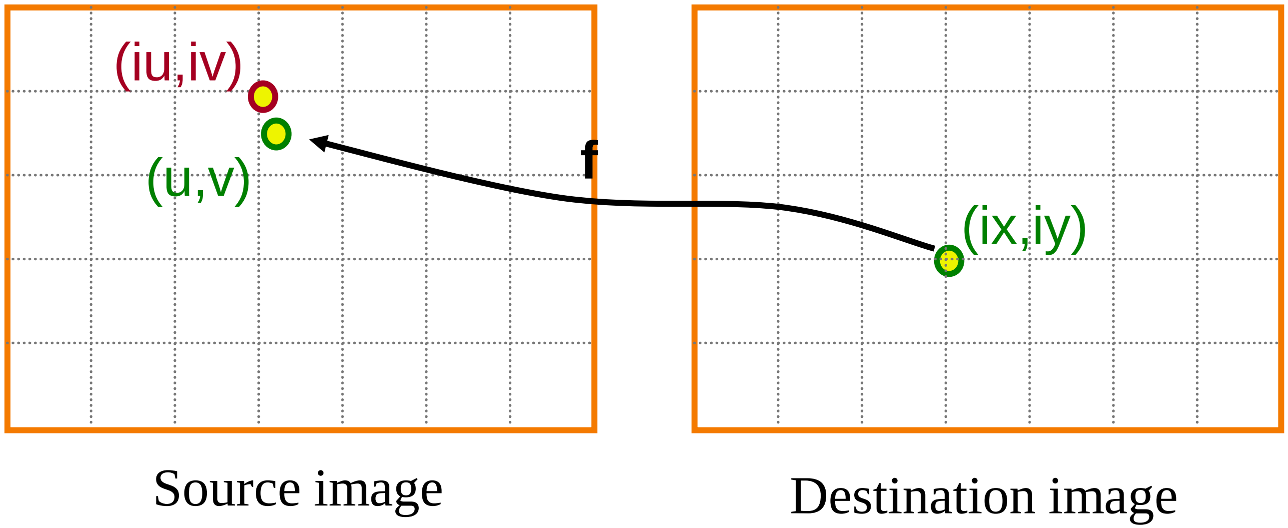


- Resampling: effectively (discrete) convolution to prevent artifacts
- Finite low-pass filters
  - Point sampling (bad)
  - Box filter
  - Triangle filter
  - Gaussian filter

Point Sampling

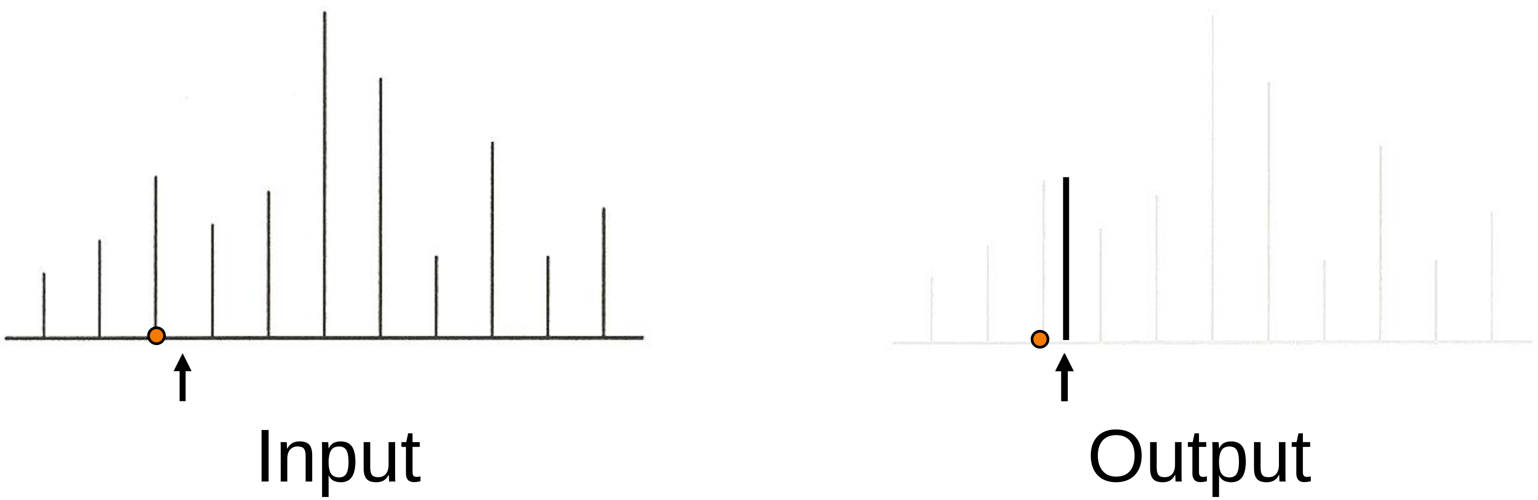
- Possible (poor) resampling implementation:

```
float Resample(src, u, v, k, w) {
   int iu = round(u);
   int iv = round(v);
   return src(iu,iv);
}
```



Point Sampling

- Use nearest sample



Point Sampling

Point Sampled: Aliasing!

Correctly Bandlimited

Resampling with Filter

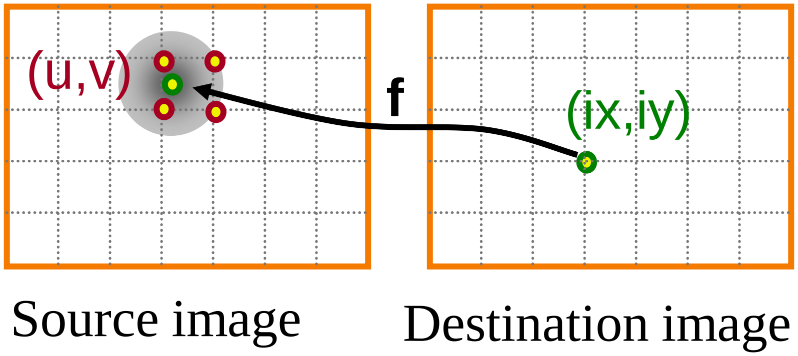
- Output is weighted average of inputs:

```
float Resample(src, u, v, k, w)
{
  float dst = 0;
  float ksum = 0;
  int ulo = u - w; etc.
  for (int iu = ulo; iu < uhi; iu++) {
    for (int iv = vlo; iv < vhi; iv++) {

              Iterate over neighborhood 2w x 2w

    }
  }

}
```



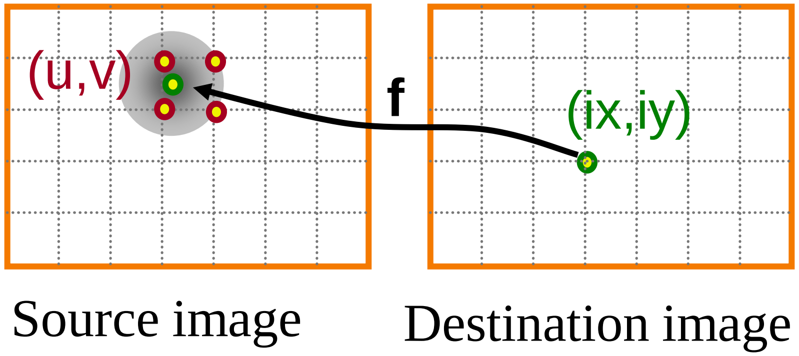
Resampling with Filter

- Output is weighted average of inputs:

```
float Resample(src, u, v, k, w)
{
  float dst = 0;
  float ksum = 0;
  int ulo = u - w; etc.
  for (int iu = ulo; iu < uhi; iu++) {
    for (int iv = vlo; iv < vhi; iv++) {
      dst += k(u,v,iu,iv,w) * src(u,v)

    }
  }

}
```

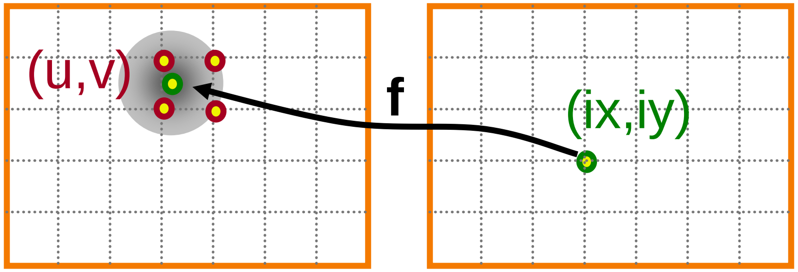


Resampling with Filter

- Output is weighted average of inputs:

```
float Resample(src, u, v, k, w)
{
  float dst = 0;
  float ksum = 0;
  int ulo = u - w; etc.
  for (int iu = ulo; iu < uhi; iu++) {
    for (int iv = vlo; iv < vhi; iv++) {
      dst += k(u,v,iu,iv,w) * src(u,v)
      ksum += k(u,v,iu,iv,w);
    }
  }

}
```

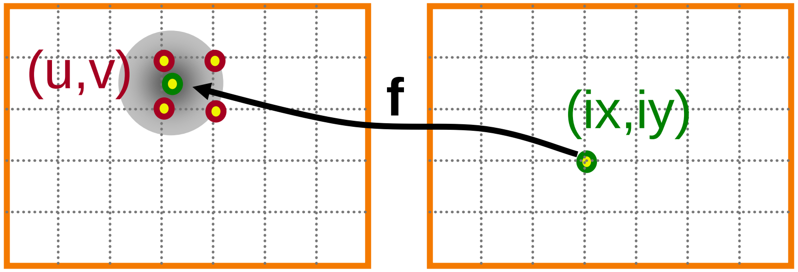


Source image Destination image

Resampling with Filter

- Output is weighted average of inputs:

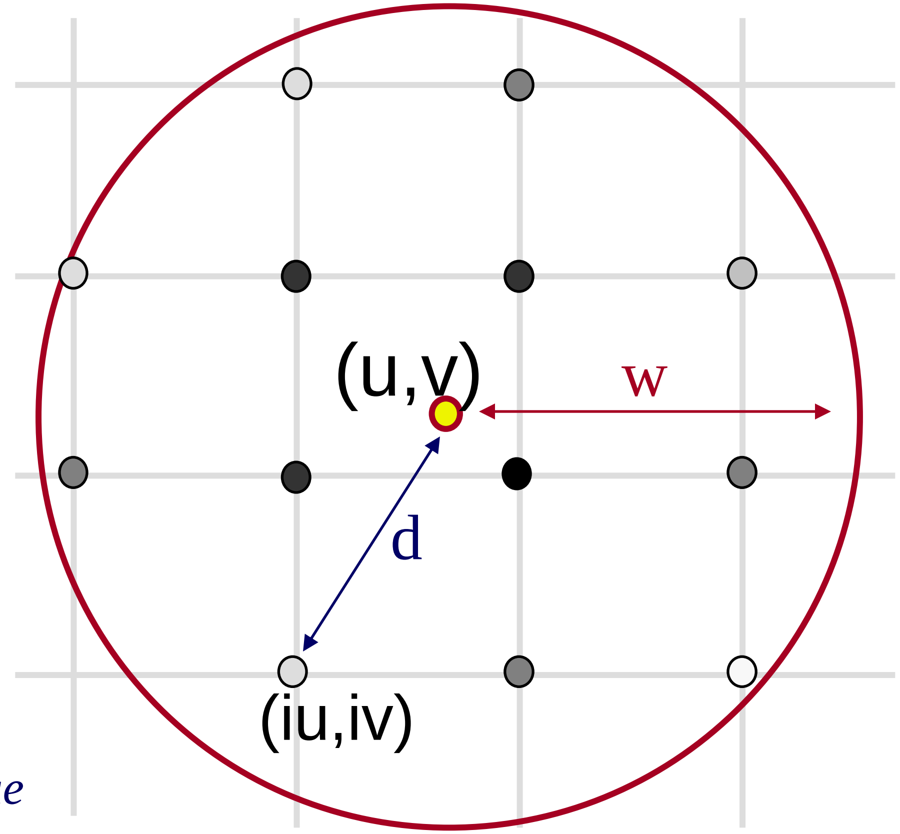
```
float Resample(src, u, v, k, w)
{
  float dst = 0;
  float ksum = 0;
  int ulo = u - w; etc.
  for (int iu = ulo; iu < uhi; iu++) {
    for (int iv = vlo; iv < vhi; iv++) {
      dst += k(u,v,iu,iv,w) * src(u,v)
      ksum += k(u,v,iu,iv,w);
    }
  }
  return dst / ksum;
}
```



Source image Destination image

Image Resampling

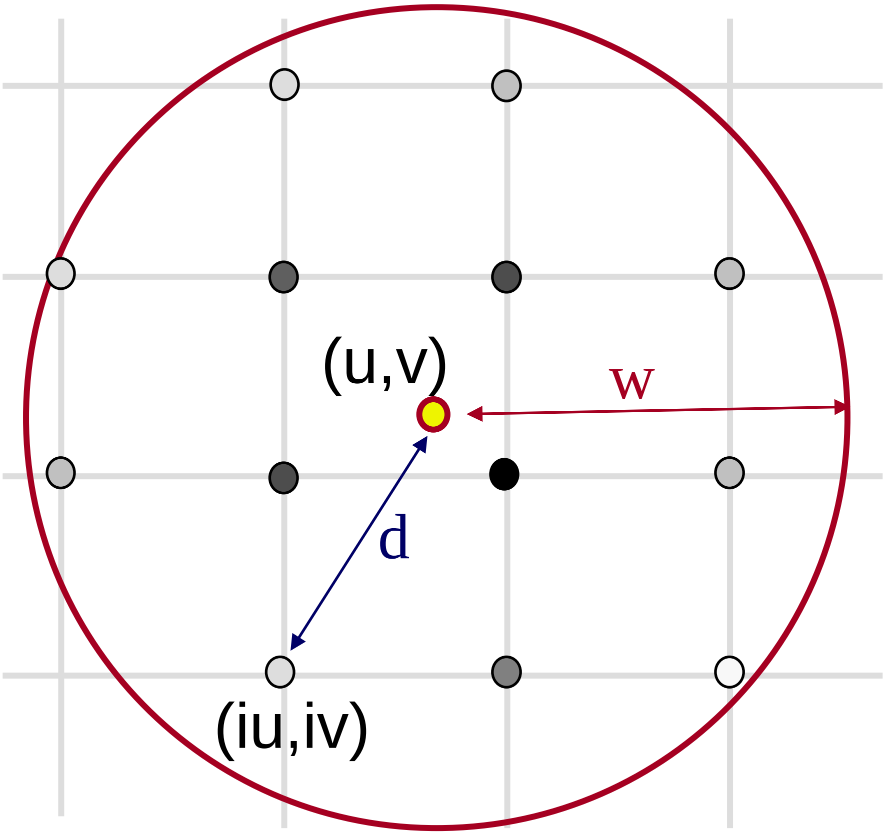
- Compute weighted sum of pixel neighborhood
  - Output is weighted average of input, where weights are normalized values of filter kernel (k)

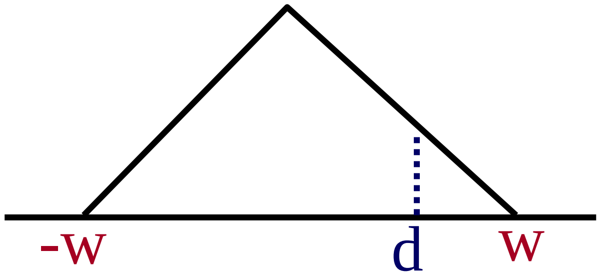


k(iu,iv) represented by gray value

Image Resampling

- For isotropic Triangle and Gaussian filters, k(iu,iu) is function of d and w





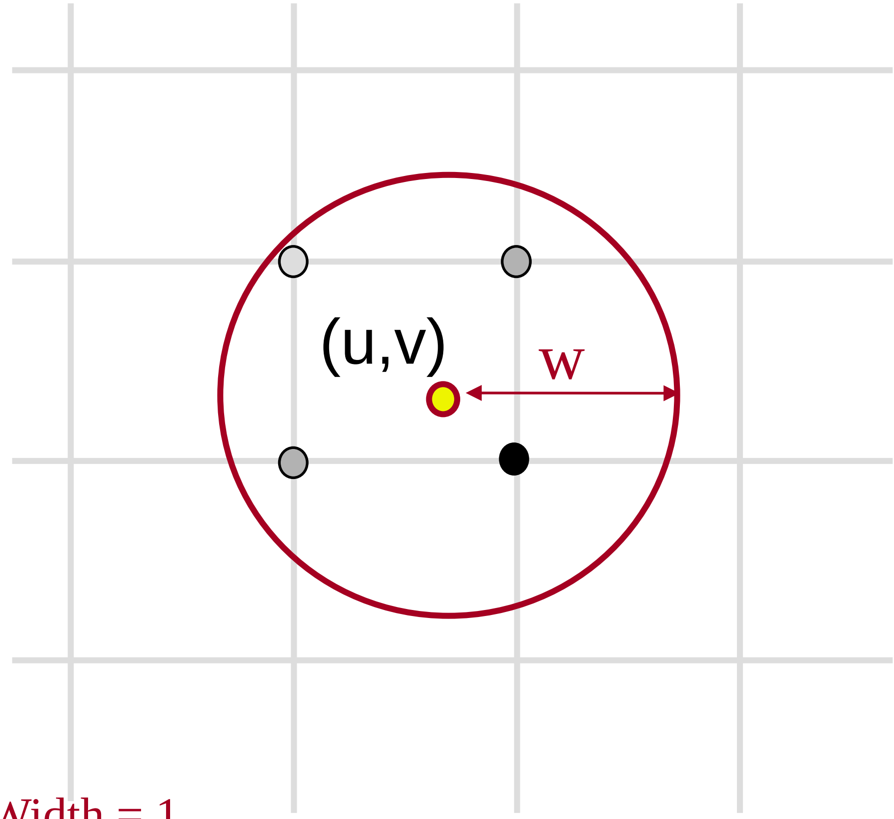
Triangle filter

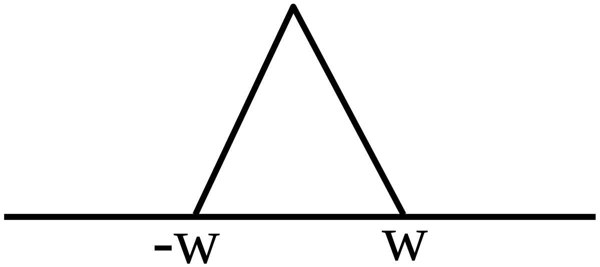
$$k(i,j)=\max(1 - d/w, 0)$$

Filter Width = 2

Image Resampling

- For isotropic Triangle and Gaussian filters,
  Filter width chosen based on scale factor
  (large for minification, small for magnification)





Triangle filter

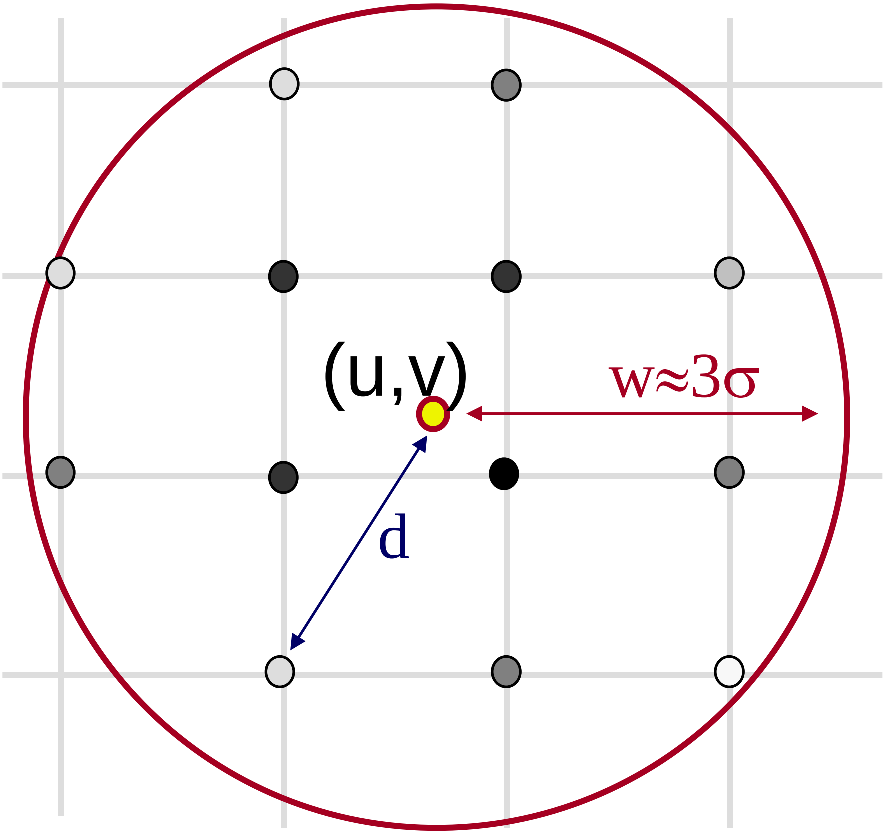
Width of filter affects blurriness

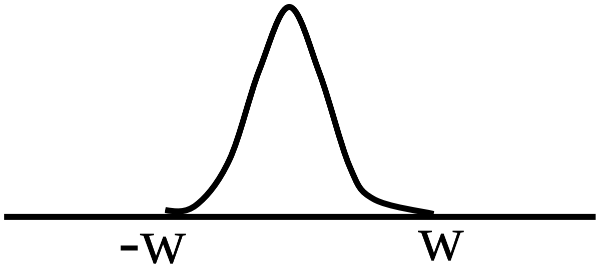
Filter Width = 1

Gaussian Filtering

- Kernel is Gaussian function

$$G(d,\sigma) = e^{-d^2/(2\sigma^2)}$$



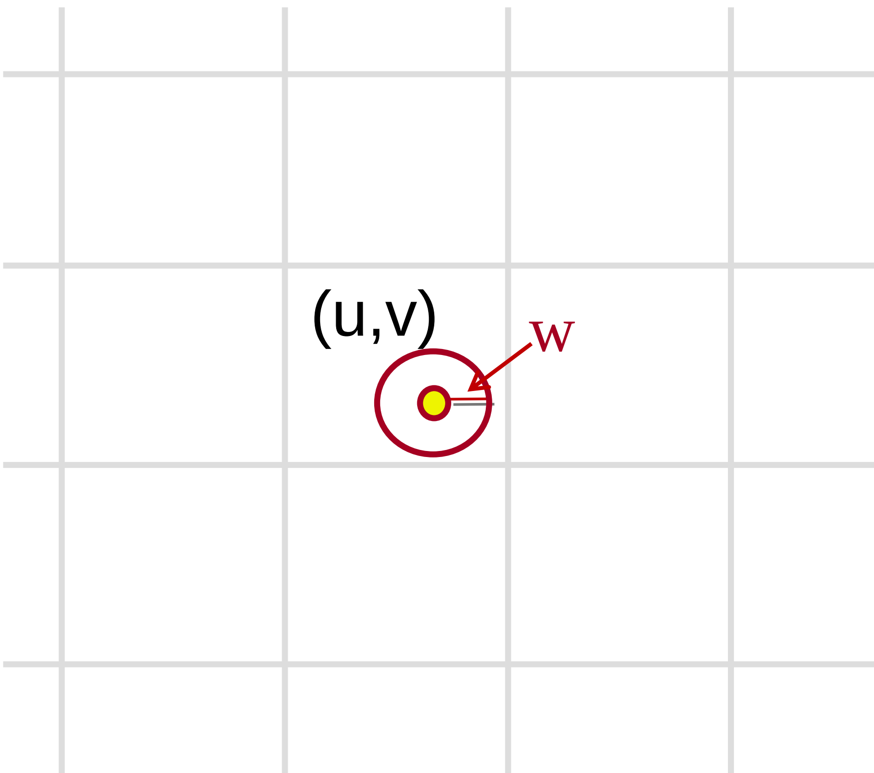


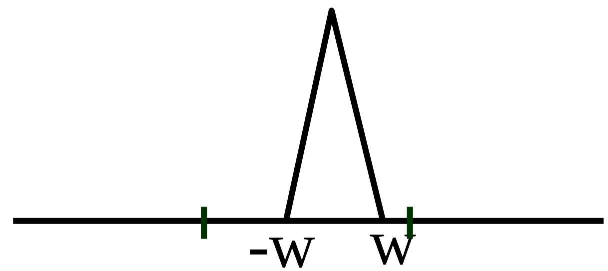
Gaussian Function

- Drops off quickly, but never gets to exactly 0
- In practice: compute out to w ~ 2.5σ or 3σ

Image Resampling

- What if width (w) is smaller than sample spacing?



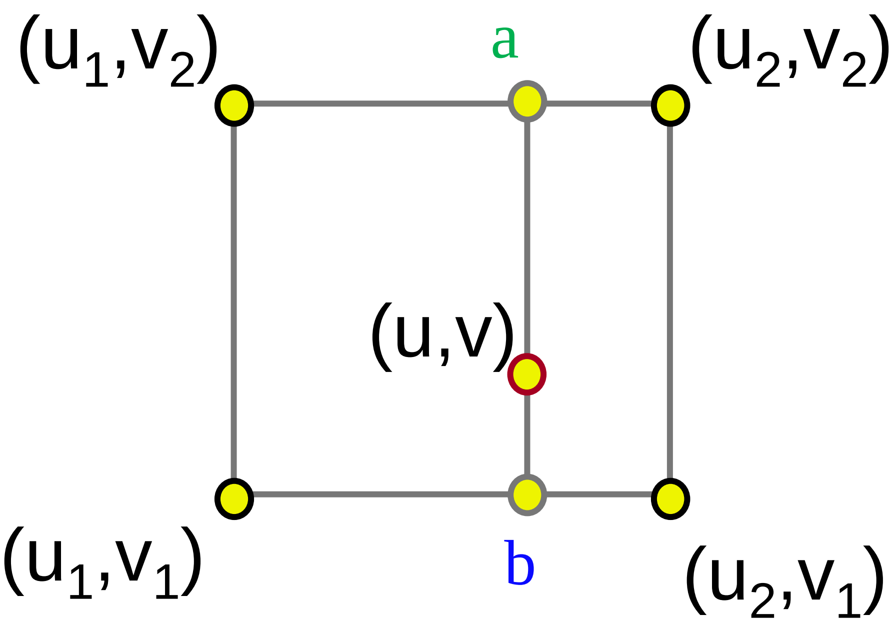


Triangle filter

Filter Width < 1

Image Resampling (with width < 1)

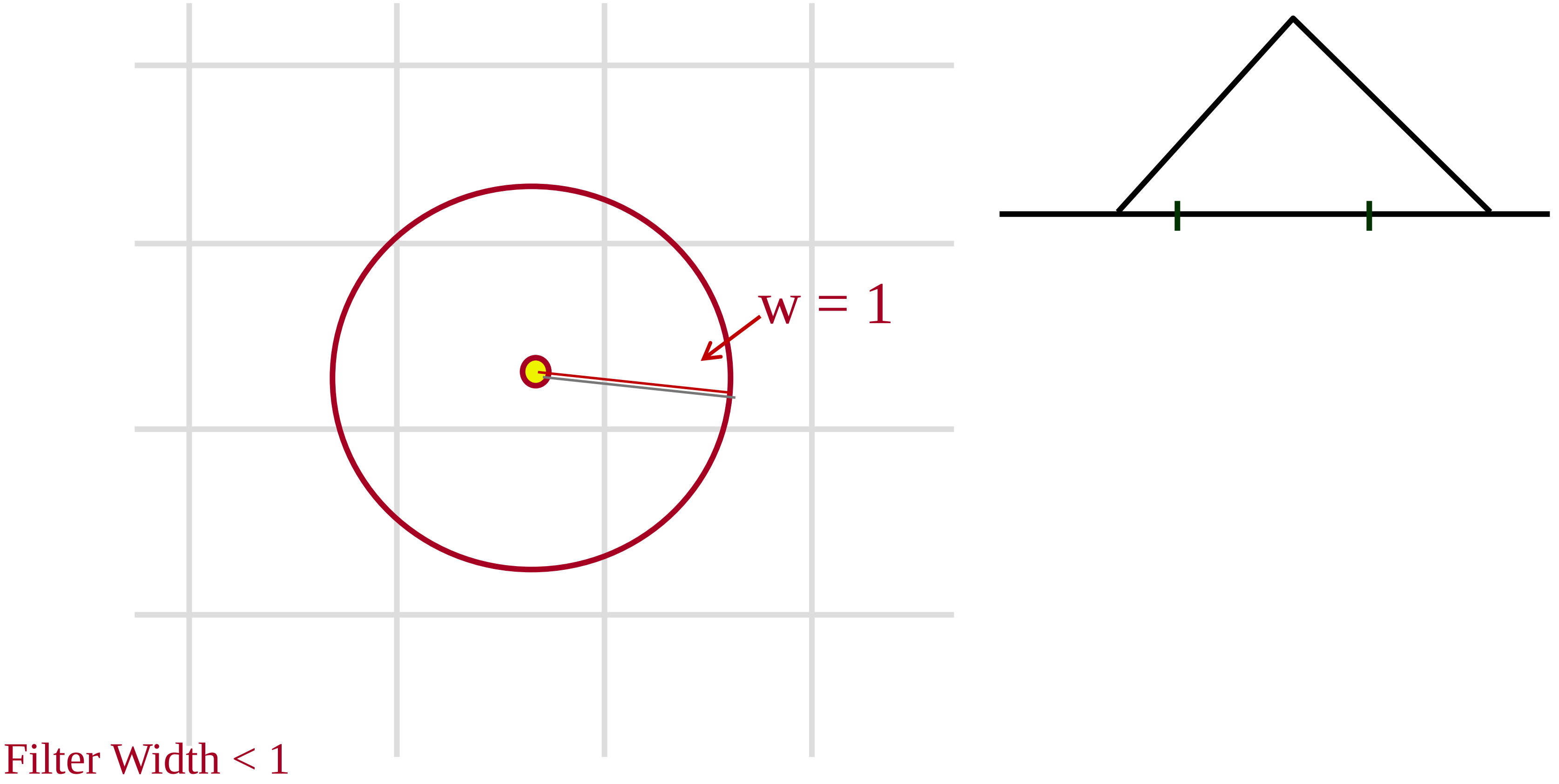
- Reconstruction filter:
  bilinear interpolation of four closest pixels
  - a = linear interpolation of src(u_1,v_2) and src(u_2,v_2)
  - b = linear interpolation of src(u_1,v_1) and src(u_2,v_1)
  - dst(x,y) = linear interpolation of "a" and "b"



Filter Width < 1

Image Resampling (with width < 1)

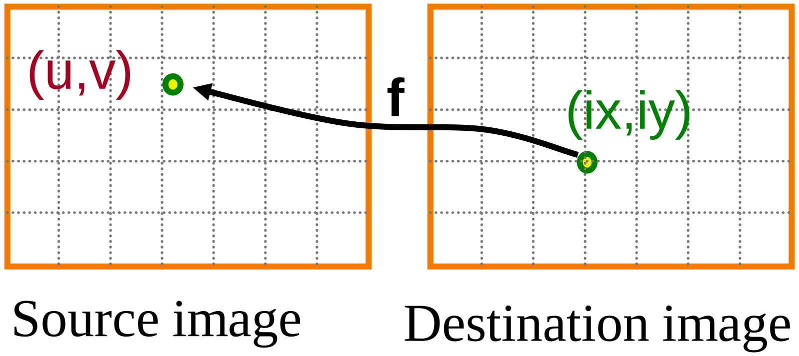
- Alternative: force width to be at least 1



Putting it All Together

- Possible implementation of image scale:

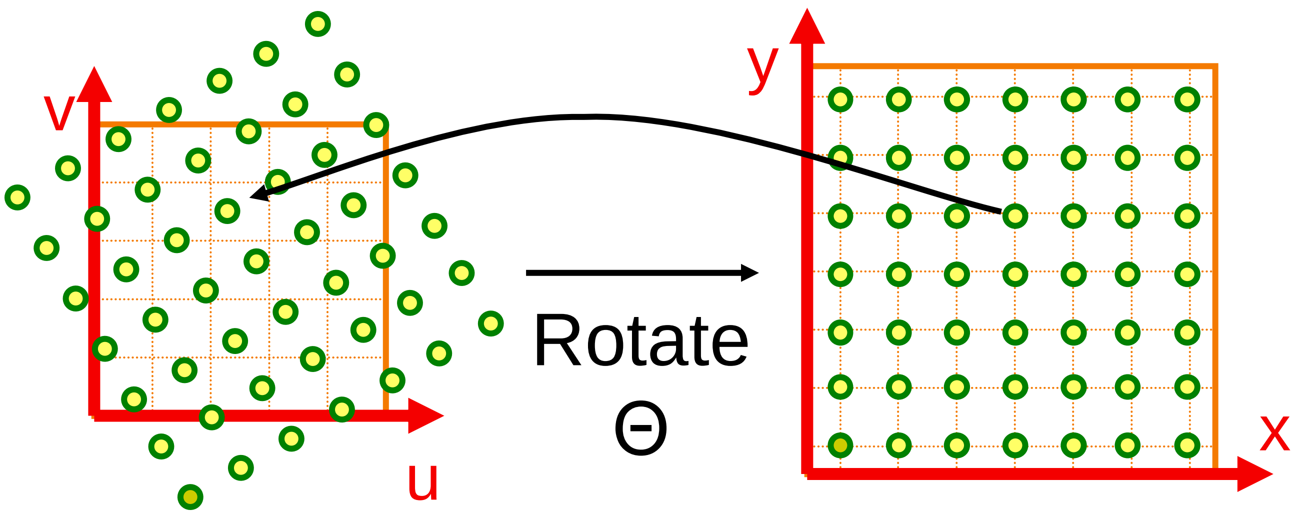
```
Scale(src, dst, sx, sy) {
  w ≈ max(1/sx,1/sy);
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float u = ix / sx;
      float v = iy / sy;
      dst(ix,iy) = Resample(src,u,v,k,w);
    }
  }
}
```



Putting it All Together

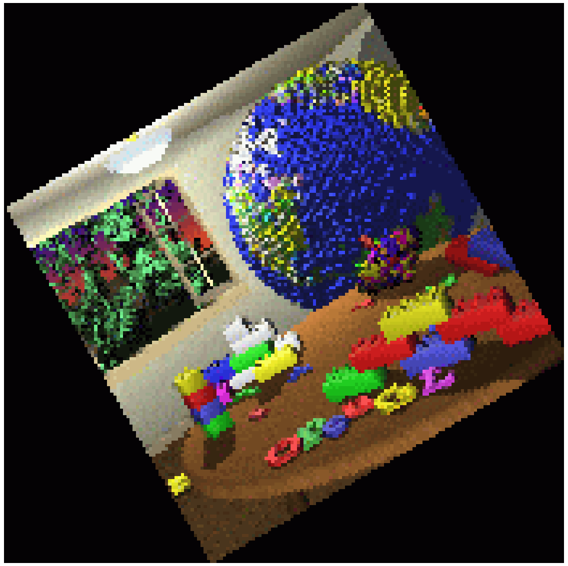
- Possible implementation of image rotation:

```
Rotate(src, dst, Θ) {
  w ≈ 1
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float u = ix*cos(-Θ) - iy*sin(-Θ);
      float v = ix*sin(-Θ) + iy*cos(-Θ);
      dst(ix,iy) = Resample(src,u,v,k,w);
    }
  }
}
```

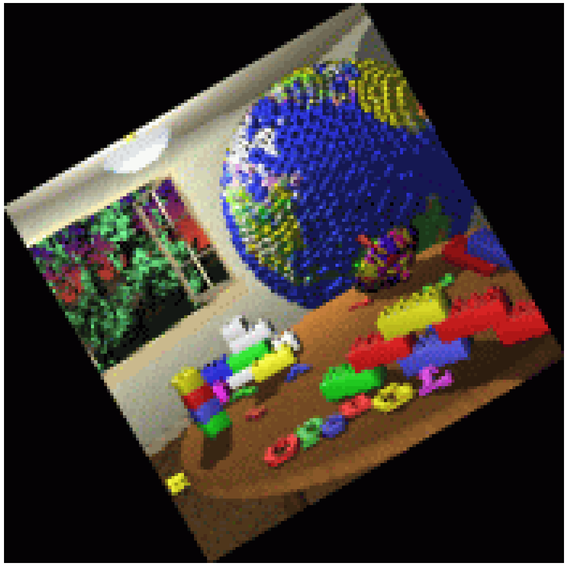


Sampling Method Comparison

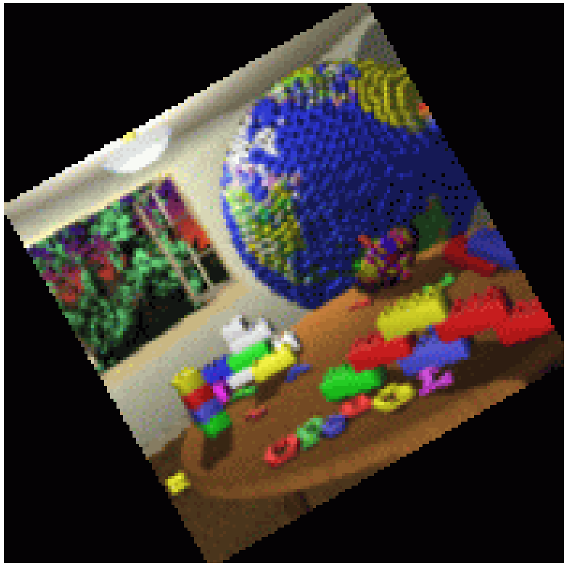
- Trade-offs
  - Aliasing versus blurring
  - Computation speed



Point



Triangle

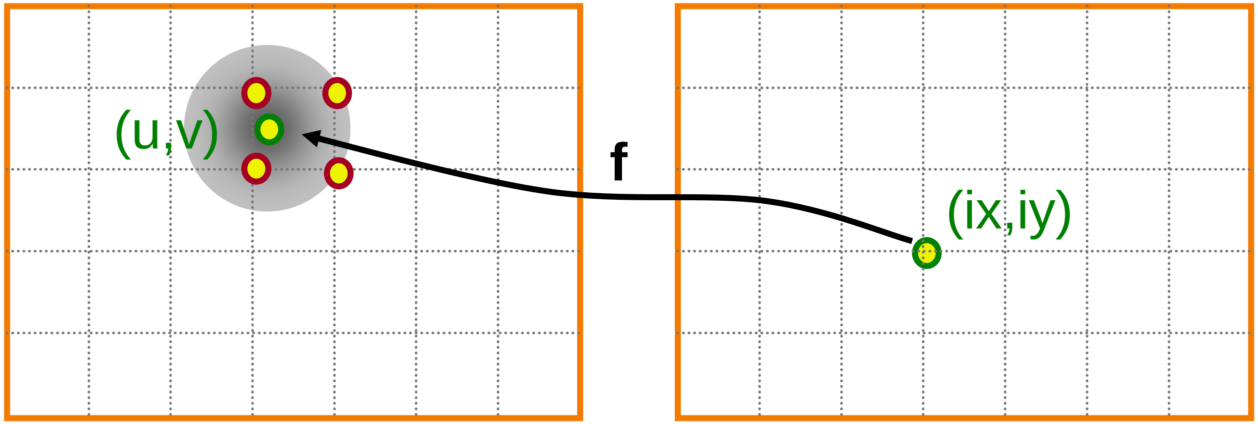


Gaussian

Forward vs. Reverse Mapping

- Reverse mapping:

```
Warp(src, dst) {
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float w ≈ 1 / scale(ix, iy);
      float u = fx^-1(ix,iy);
      float v = fy^-1(ix,iy);
      dst(ix,iy) = Resample(src,u,v,w);
    }
  }
}
```



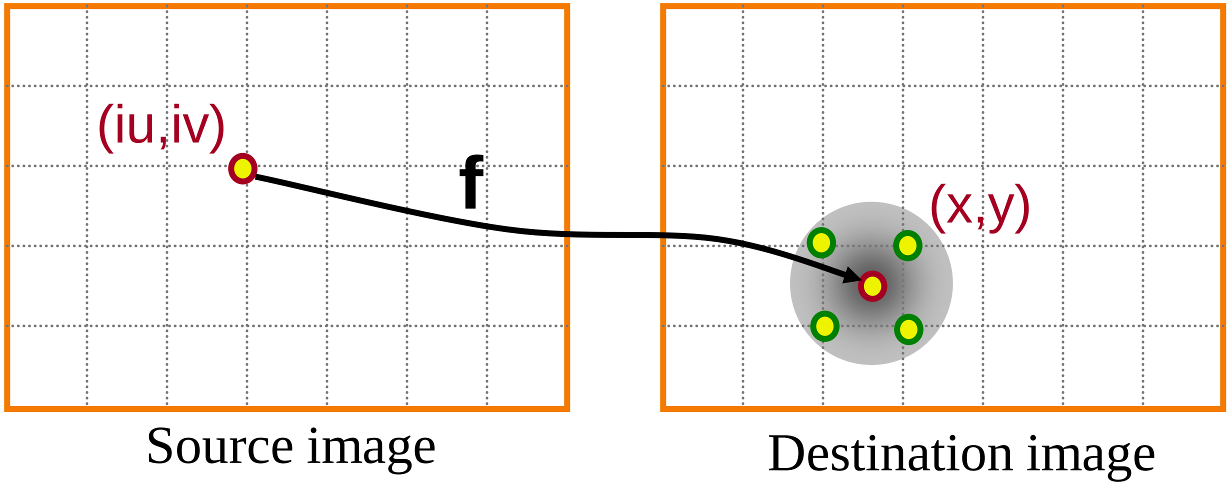
Source image

Destination image

Forward vs. Reverse Mapping

- Forward mapping:

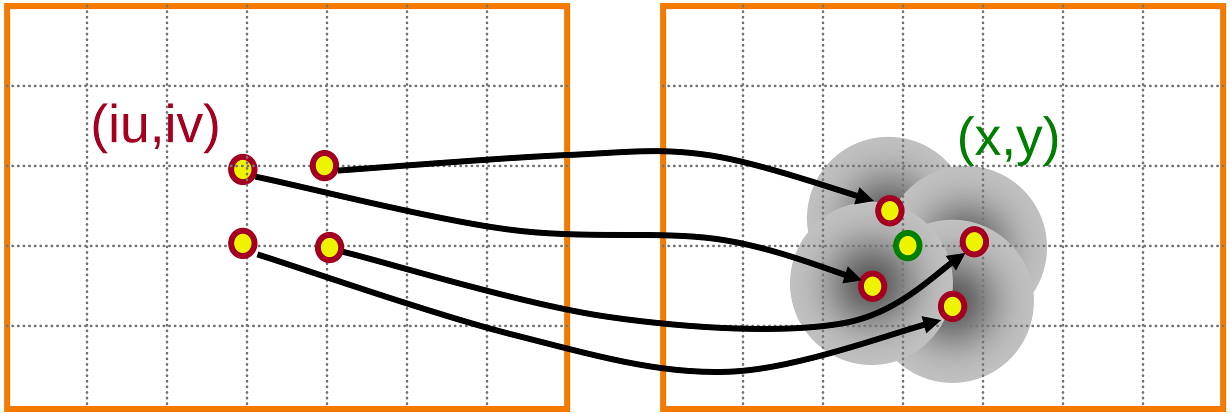
```
Warp(src, dst) {
  for (int iu = 0; iu < umax; iu++) {
    for (int iv = 0; iv < vmax; iv++) {
      float x = fx(iu,iv);
      float y = fy(iu,iv);
      float w ≈ 1 / scale(x, y);
      Splat(src(iu,iv),x,y,k,w);
    }
  }
}
```



Forward vs. Reverse Mapping

- Forward mapping:

```
Warp(src, dst) {
  for (int iu = 0; iu < umax; iu++) {
    for (int iv = 0; iv < vmax; iv++) {
      float x = fx(iu,iv);
      float y = fy(iu,iv);
      float w ≈ 1 / scale(x, y);
      Splat(src(iu,iv),x,y,k,w);
    }
  }
}
```



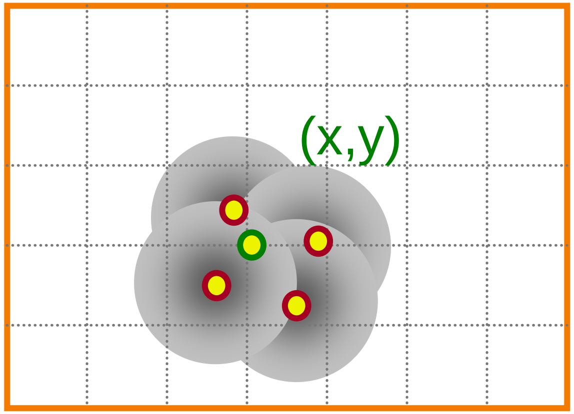
Source image   Destination image

Forward vs. Reverse Mapping

- Forward mapping:

```
for (int iu = 0; iu < umax; iu++) {
  for (int iv = 0; iv < vmax; iv++) {
    float x = fx(iu,iv);
    float y = fy(iu,iv);
    float w ≈ 1 / scale(x, y);
    for (int ix = xlo; ix <= xhi; ix++) {
      for (int iy = ylo; iy <= yhi; iy++) {
        dst(ix,iy) += k(x,y,ix,iy,w) * src(iu,iv);
      }
    }
  }
}
```

Problem?

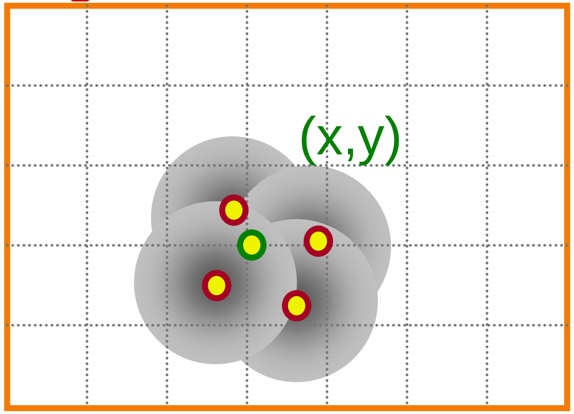


Destination image

Forward vs. Reverse Mapping

- Forward mapping:

```
for (int iu = 0; iu < umax; iu++) {
  for (int iv = 0; iv < vmax; iv++) {
    float x = fx(iu,iv);
    float y = fy(iu,iv);
    float w ≈ 1 / scale(x, y);
    for (int ix = xlo; ix <= xhi; ix++) {
      for (int iy = ylo; iy <= yhi; iy++) {
        dst(ix,iy) += k(x,y,ix,iy,w) * src(iu,iv);
        ksum(ix,iy) += k(x,y,ix,iy,w);
      }
    }
  }
}
for (ix = 0; ix < xmax; ix++)
  for (iy = 0; iy < ymax; iy++)
    dst(ix,iy) /= ksum(ix,iy)
```



Destination image

Forward vs. Reverse Mapping

- Tradeoffs:
  - Forward mapping:
    - Requires separate buffer to store weights
  - Reverse mapping:
    - Requires inverse of mapping function, random access to original image

Summary

- Mapping
  - Forward vs. reverse
- Sampling, reconstruction, resampling
  - Frequency analysis of signal content
  - Filter to avoid undersampling: point, triangle, Gaussian
  - Reduce visual artifacts due to aliasing
    - » Blurring is better than aliasing

Next Time…

- Changing pixel values
  - Linear: scale, offset, etc.
  - Nonlinear: gamma, saturation, etc.
  - Histogram equalization
- Filtering over neighborhoods
  - Blur & sharpen
  - Detect edges
  - Median
  - Bilateral filter
- Moving image locations
  - Scale
  - Rotate
  - Warp
- Combining images
  - Composite
  - Morph
- Quantization
- Spatial / intensity tradeoff
  - Dithering