Sampling, Resampling, and Warping

COS 426

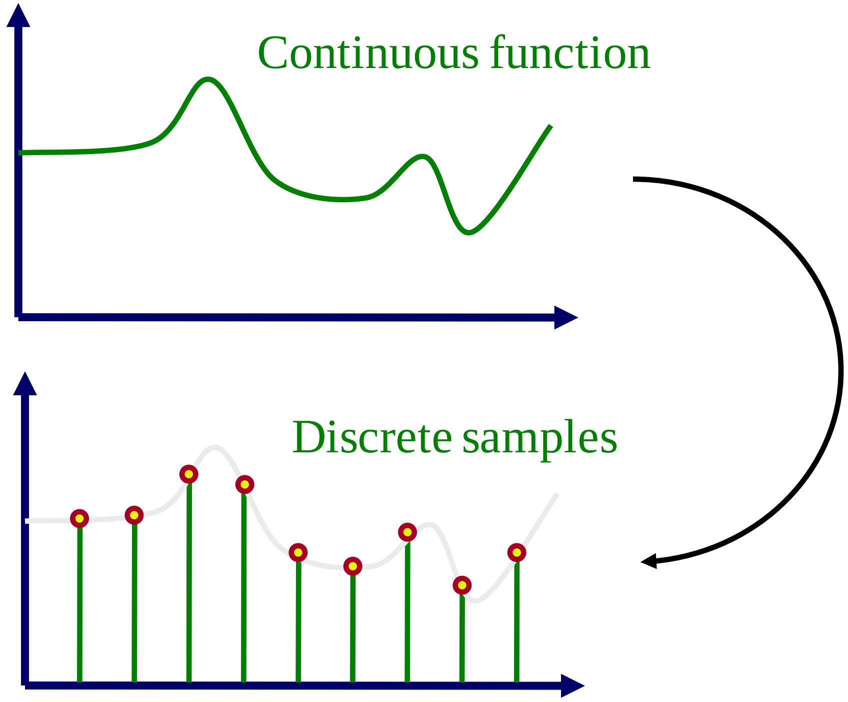
Digital Image Processing

- Changing intensity/color
  - Linear: scale, offset, etc.
  - Nonlinear: gamma, saturation, etc.
  - Add random noise
- Filtering over neighborhoods
  - Blur
  - Detect edges
  - Sharpen
  - Emboss
  - Median
- Moving image locations
  - Scale
  - Rotate
  - Warp
- Combining images
  - Composite
  - Morph
- Quantization
- Spatial / intensity tradeoff
  - Dithering

Digital Image Processing

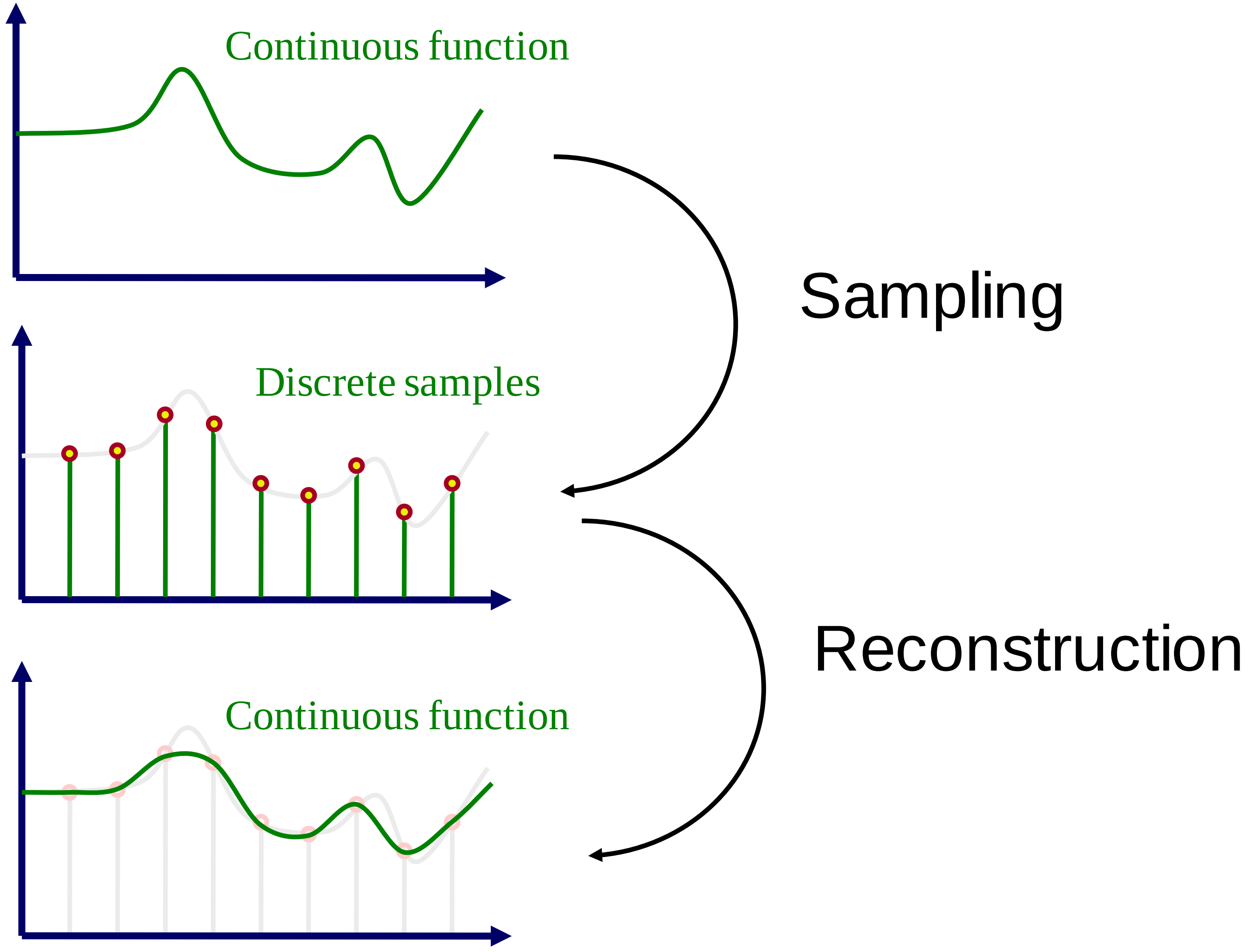
When implementing operations that move pixels, must account for the fact that digital images are sampled versions of continuous ones

Sampling and Reconstruction



Sampling

Sampling and Reconstruction



Sampling and Reconstruction

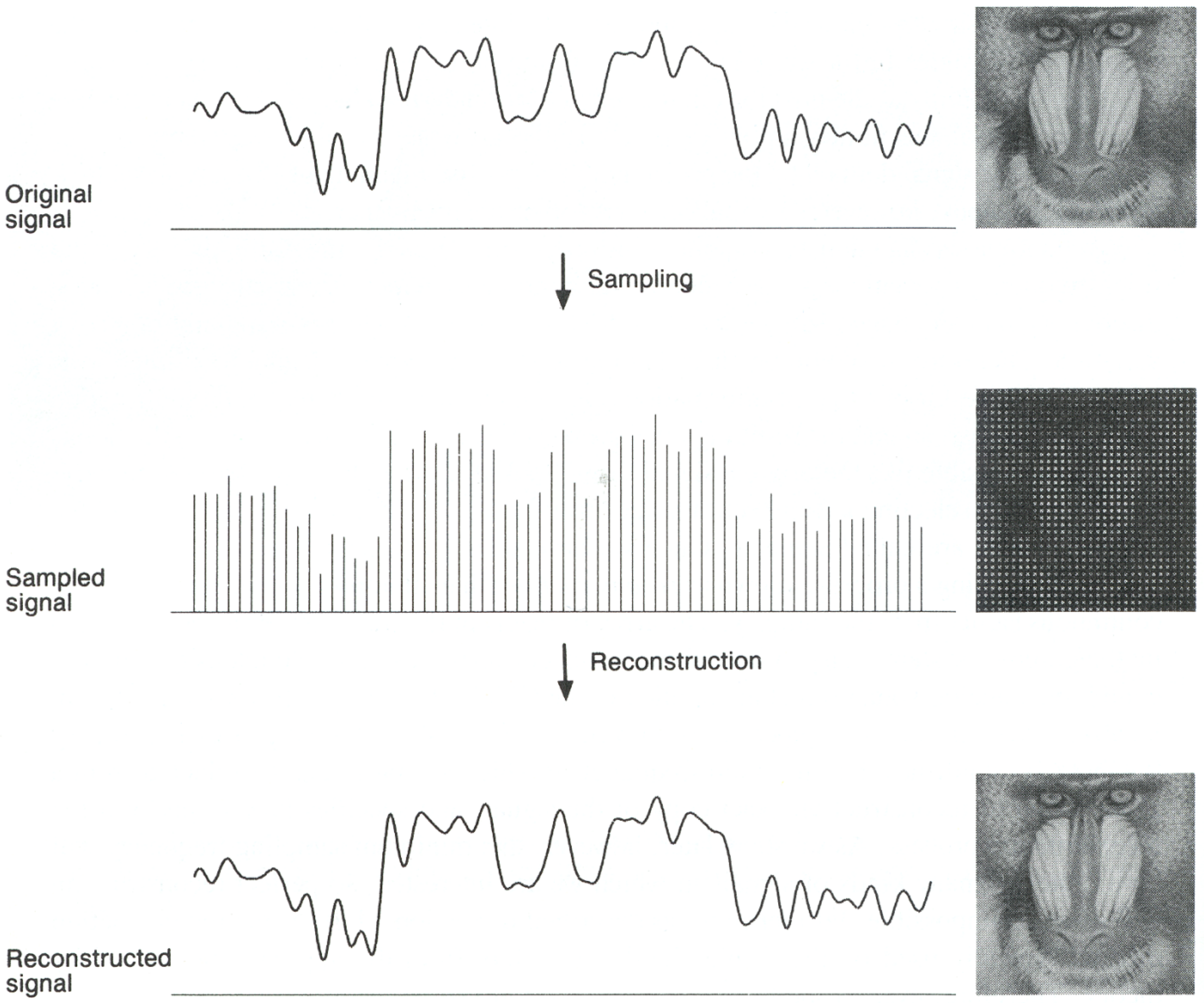
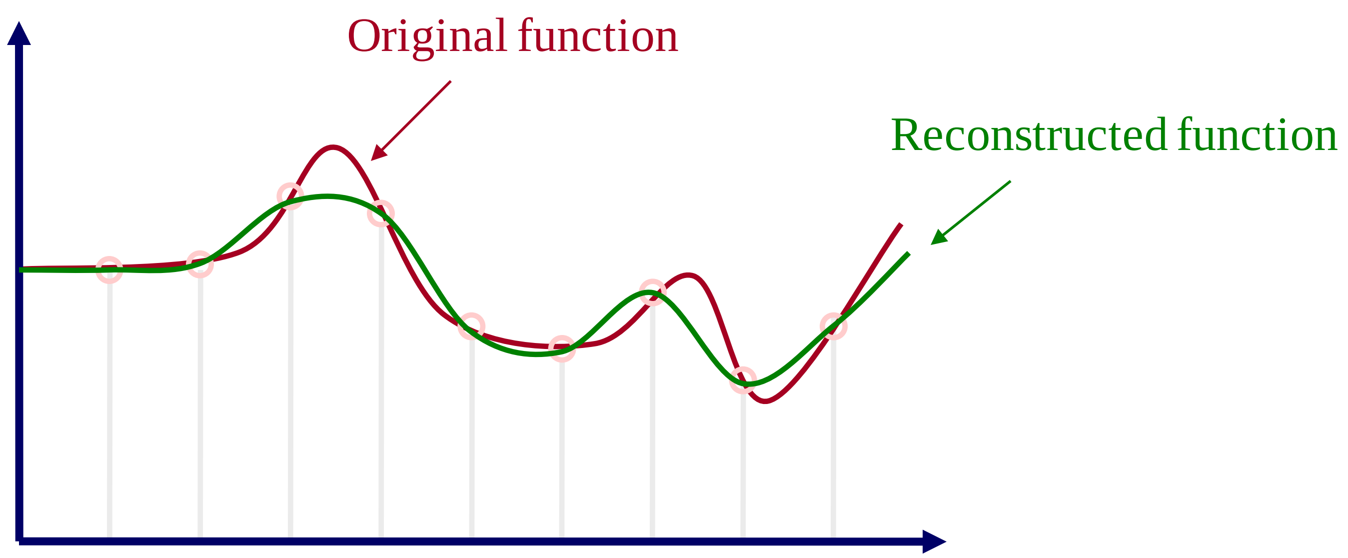


Figure 19.9 FvDFH

Sampling Theory

How many samples are enough?

- 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

What happens when we use too few samples?

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

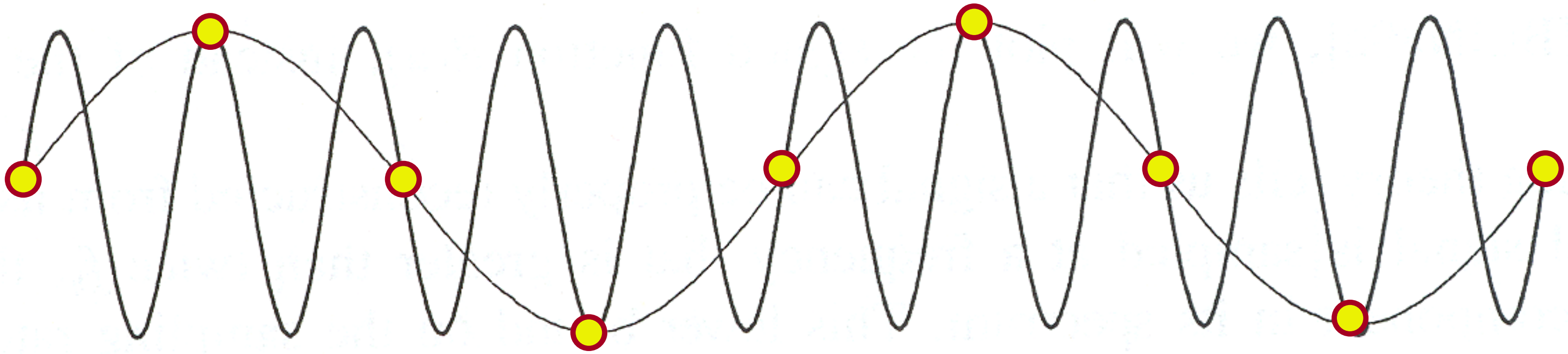
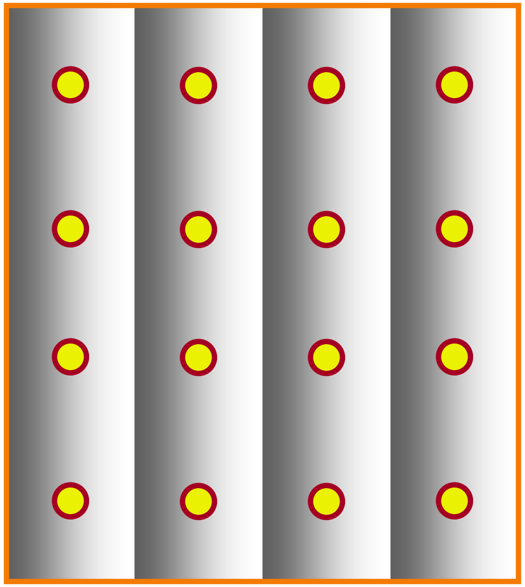


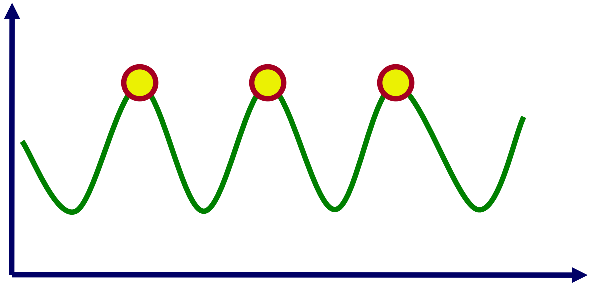
Figure 14.17 FvDFH

Sampling Theory

What happens when we use too few samples?

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





Sampling Theory

What happens when we use too few samples?

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

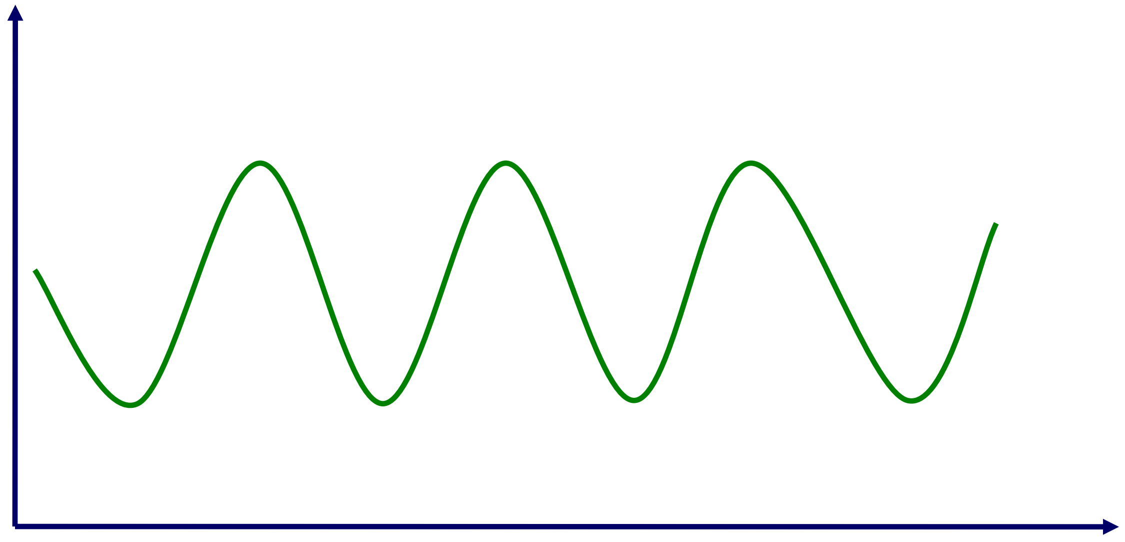
(Barely) adequate sampling

Inadequate sampling

Sampling Theory

How many samples are enough to avoid aliasing?

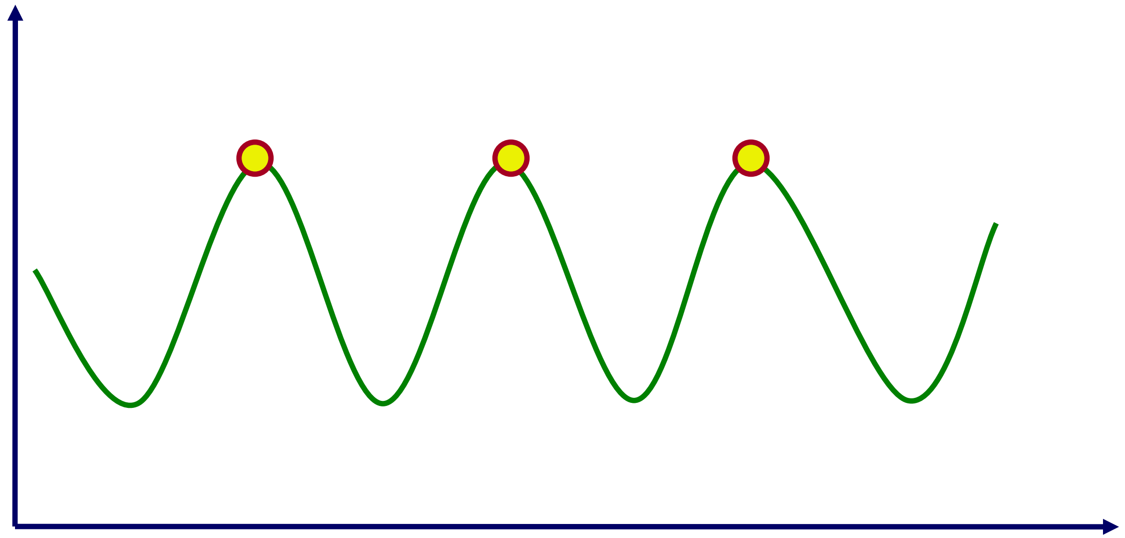
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



Sampling Theory

How many samples are enough to avoid aliasing?

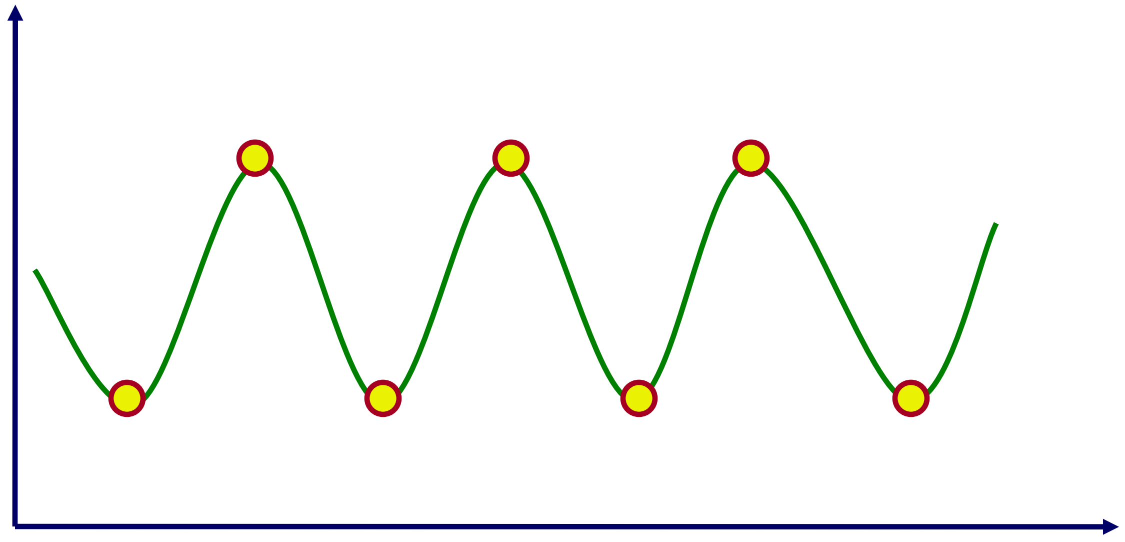
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



Sampling Theory

How many samples are enough to avoid aliasing?

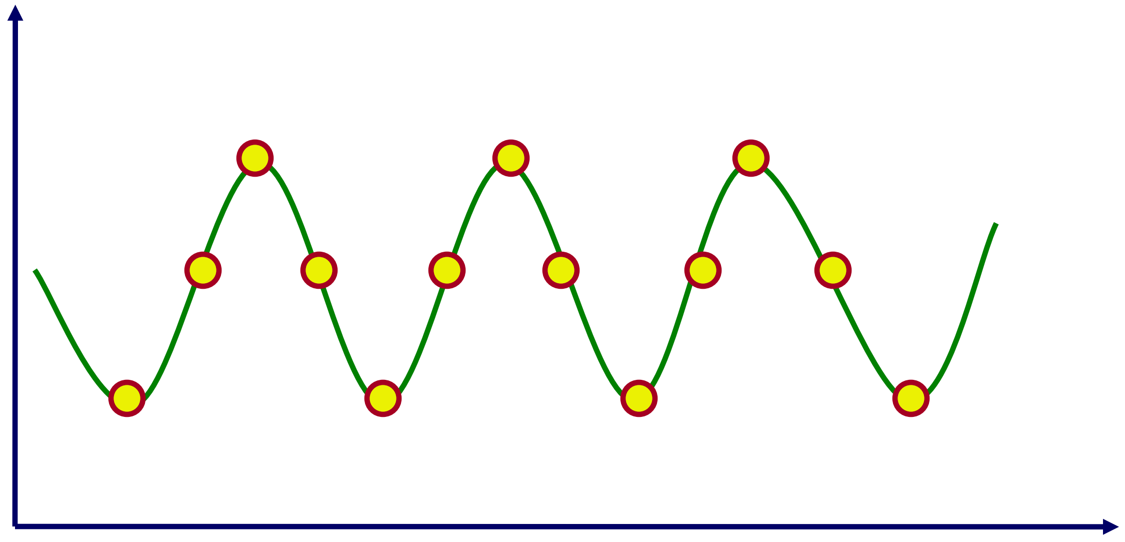
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



Sampling Theory

How many samples are enough to avoid aliasing?

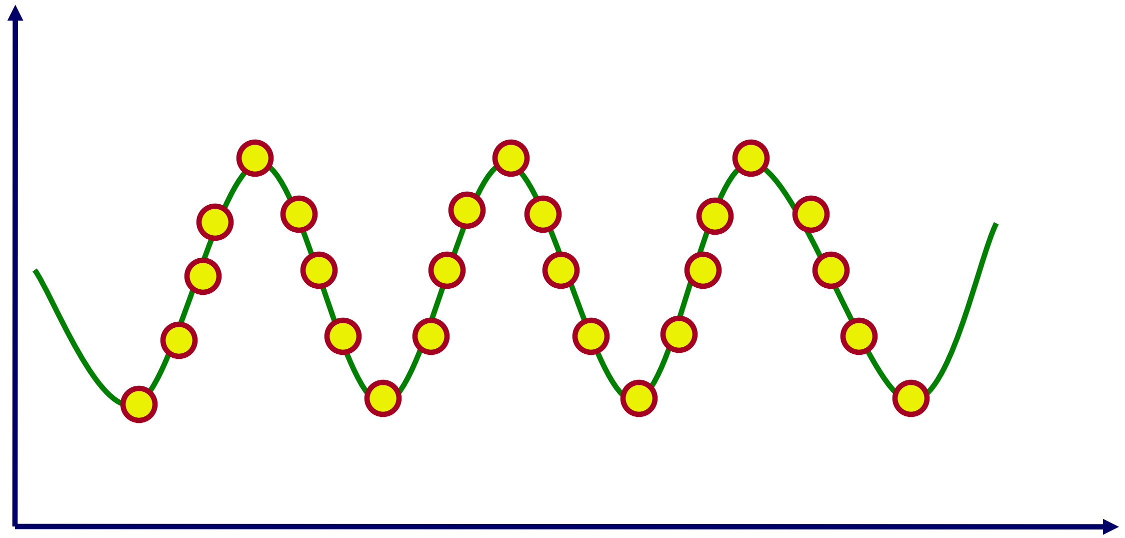
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



Sampling Theory

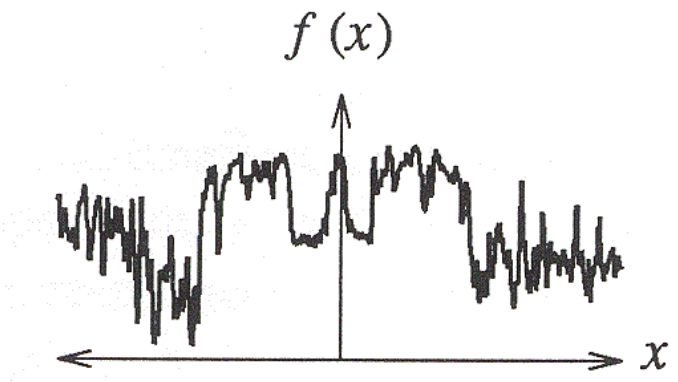
How many samples are enough to avoid aliasing?

- 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?

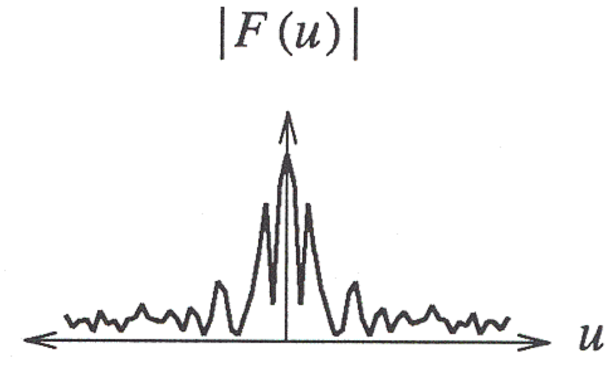


Spectral Analysis

- Spatial domain:
  - Function: f(x)
  - Filtering: convolution

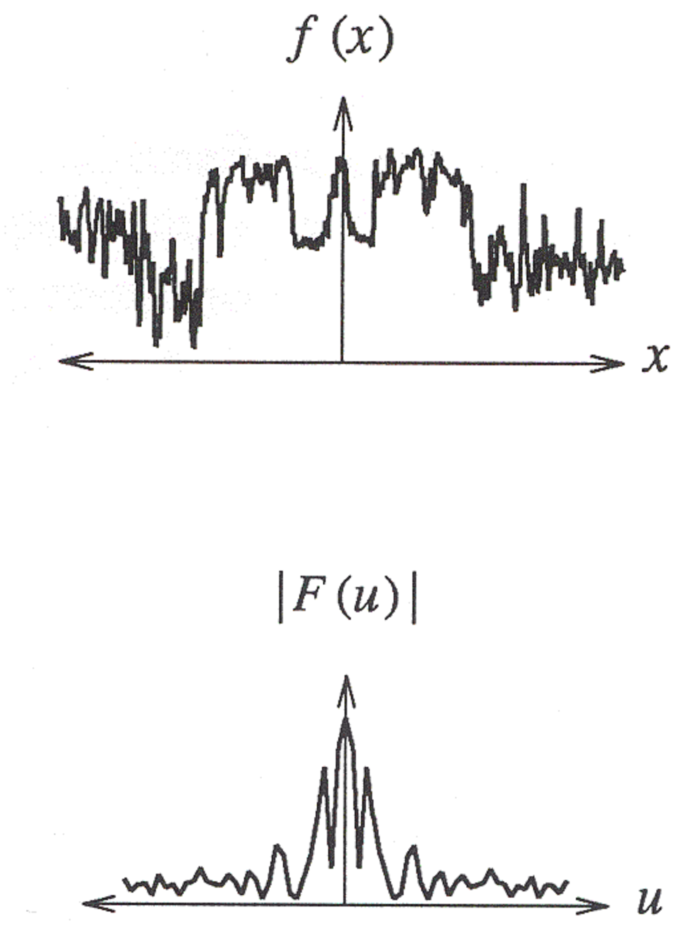


- Frequency domain:
  - Function: F(u)
  - Filtering: multiplication



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

Fourier Transform



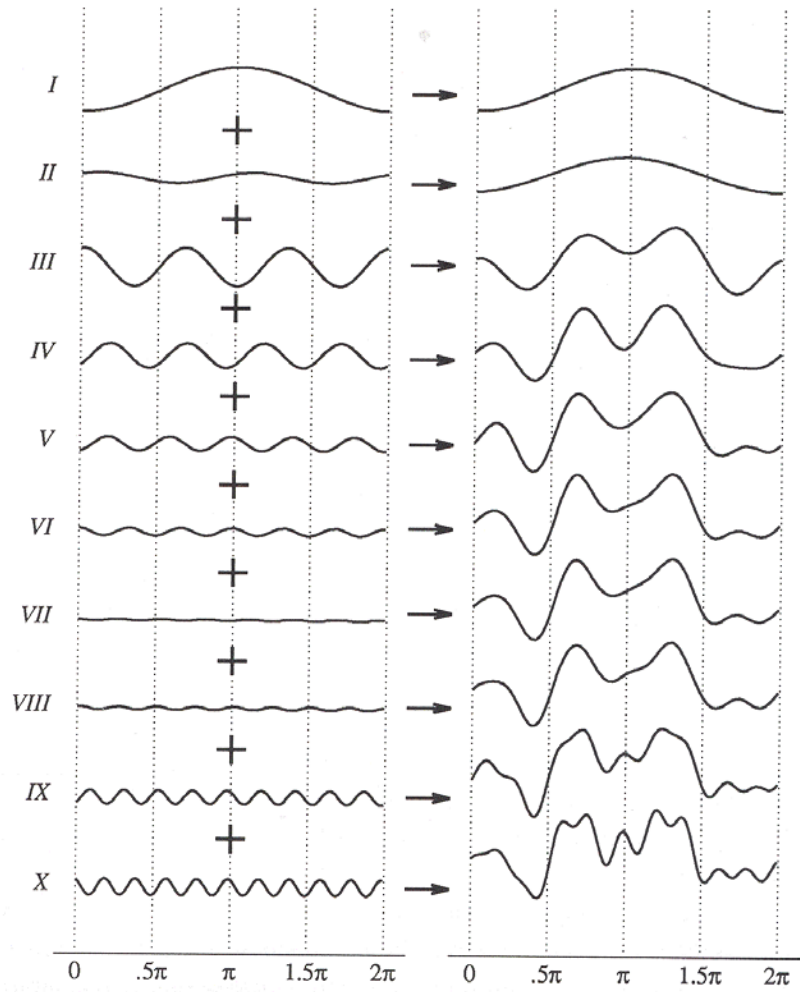


Figure 2.6 Wolberg

Fourier Transform

- Fourier transform:

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

- Inverse Fourier transform:

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

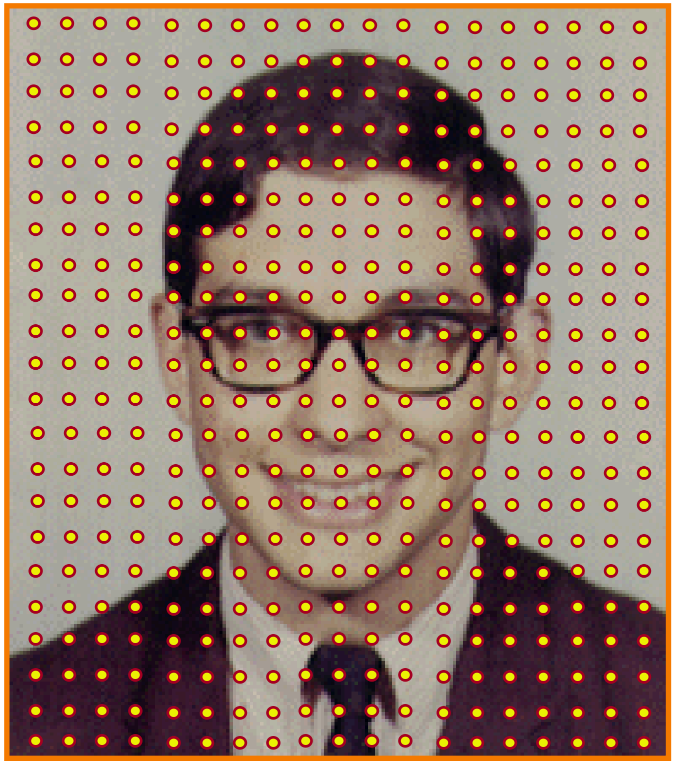
Sampling Theorem

- A signal can be reconstructed from its samples, iff the original signal has no content >= 1/2 the sampling frequency - Shannon
- The minimum sampling rate for bandlimited function is called the "Nyquist rate"

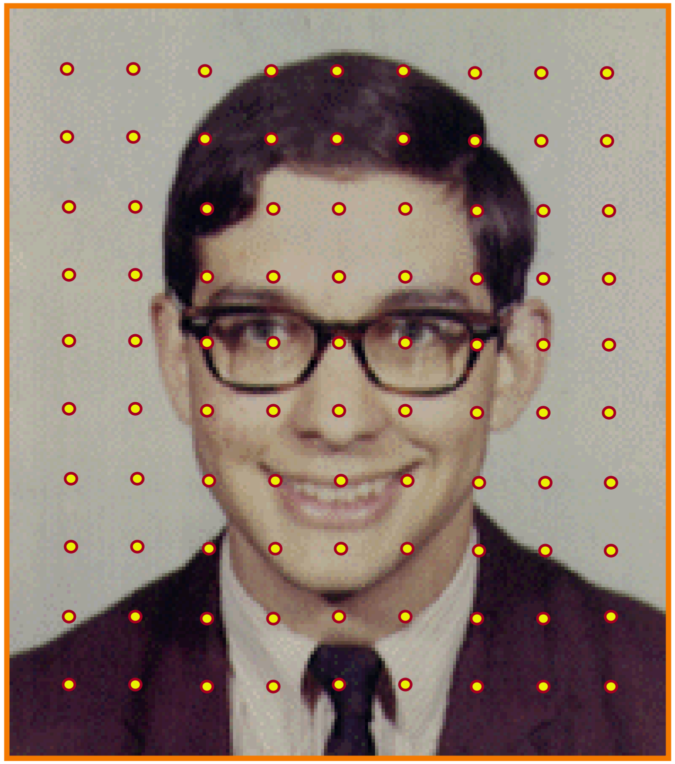
A signal is bandlimited if its highest frequency is bounded. The frequency is called the bandwidth.

Image Processing

- Consider reducing the image resolution



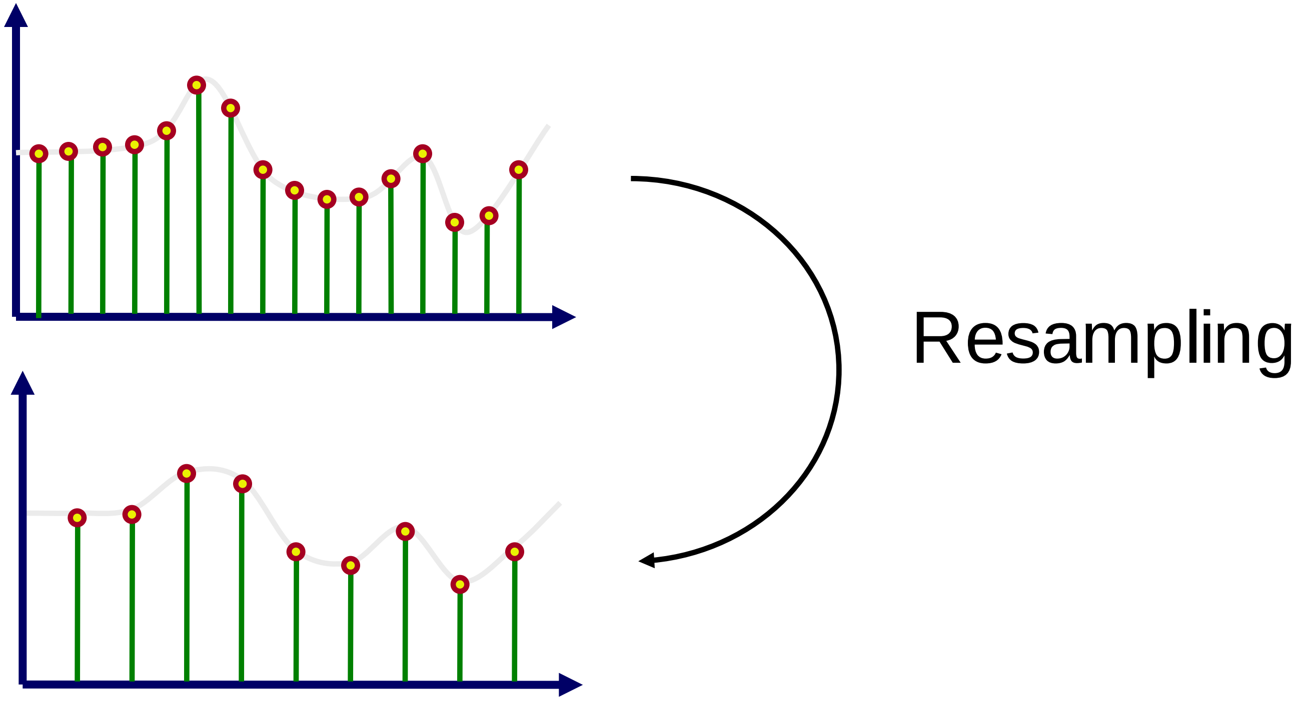
Original image



1/4 resolution

Image Processing

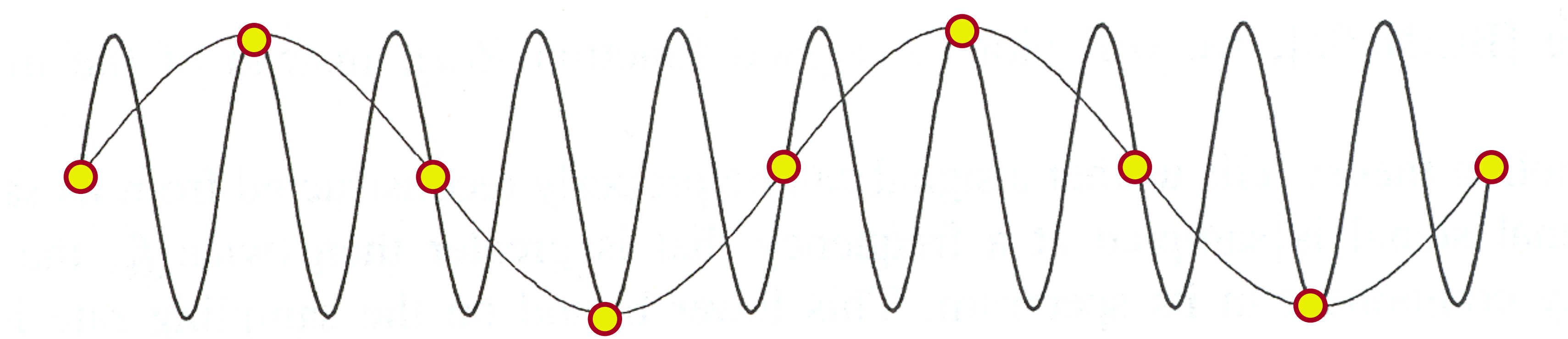
- Image processing is a resampling problem



Sampling Theorem

- A signal can be reconstructed from its samples, iff the original signal has no content >= 1/2 the sampling frequency - Shannon

Aliasing will occur if the signal is under-sampled

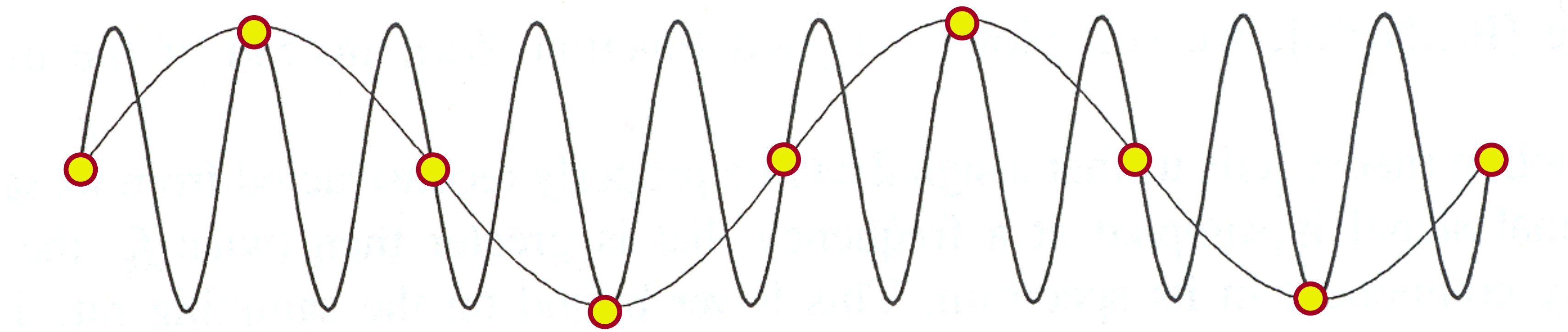


Under-sampling

Figure 14.17 FvDFH

Aliasing

- In general:
  - Artifacts due to under-sampling or poor reconstruction
- Specifically, in graphics:
  - Spatial aliasing
  - Temporal aliasing

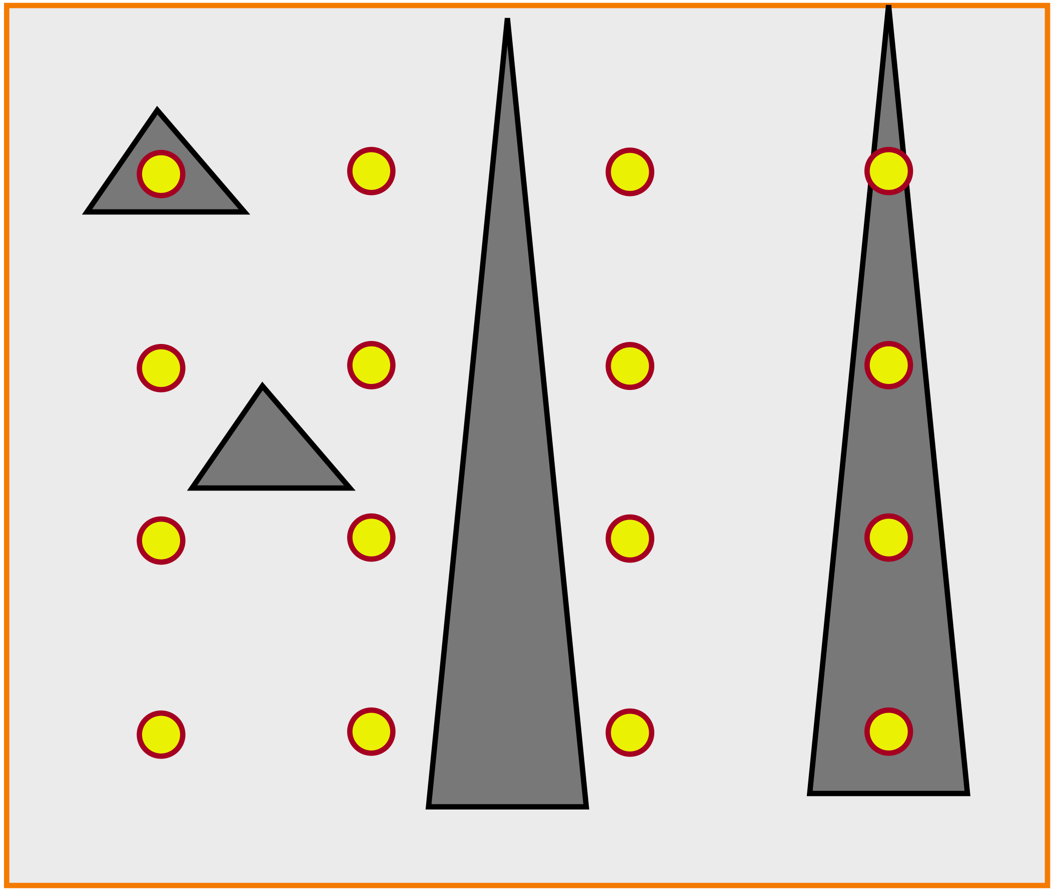


Under-sampling

Figure 14.17 FvDFH

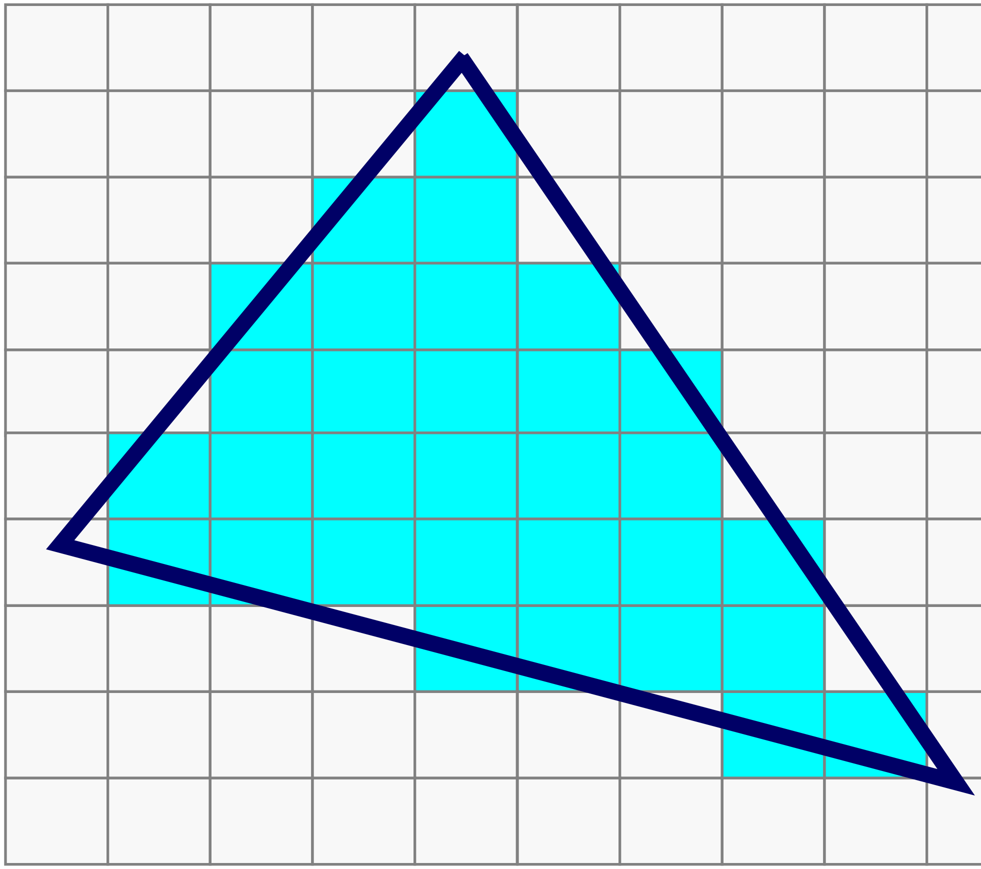
Spatial Aliasing

Artifacts due to limited spatial resolution



Spatial Aliasing

Artifacts due to limited spatial resolution

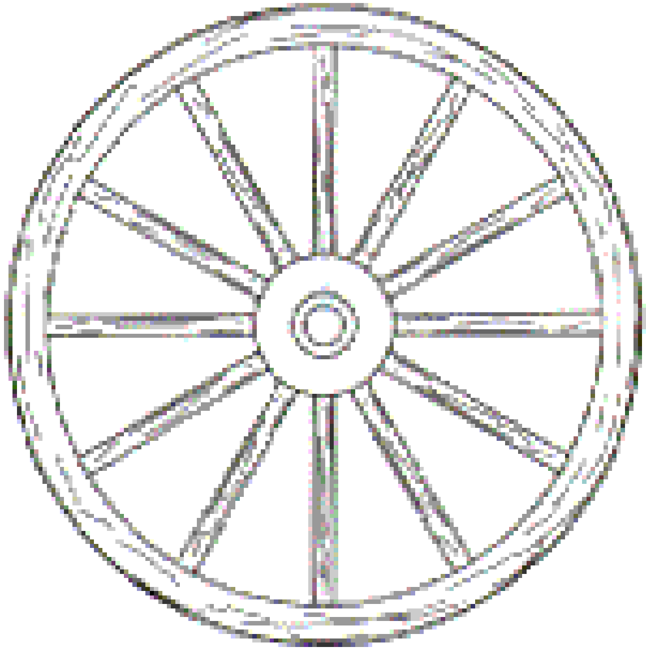


"Jaggies"

Temporal Aliasing

Artifacts due to limited temporal resolution

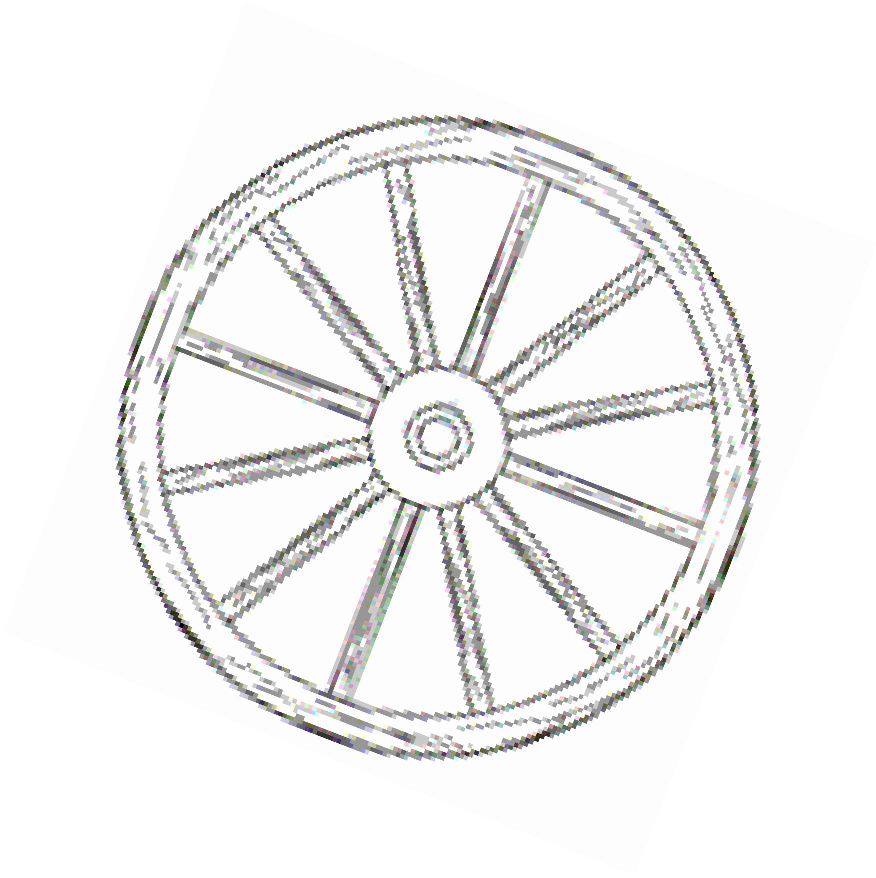
- Strobing
- Flickering



Temporal Aliasing

Artifacts due to limited temporal resolution

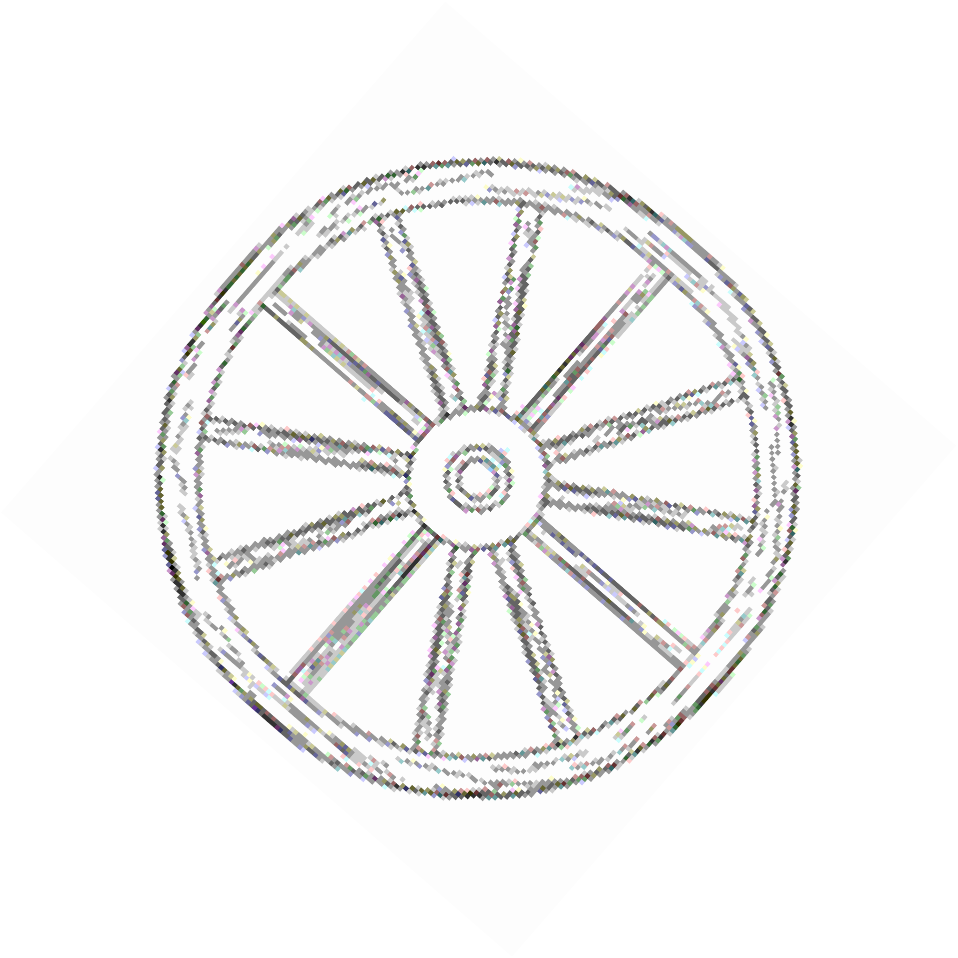
- Strobing
- Flickering



Temporal Aliasing

Artifacts due to limited temporal resolution

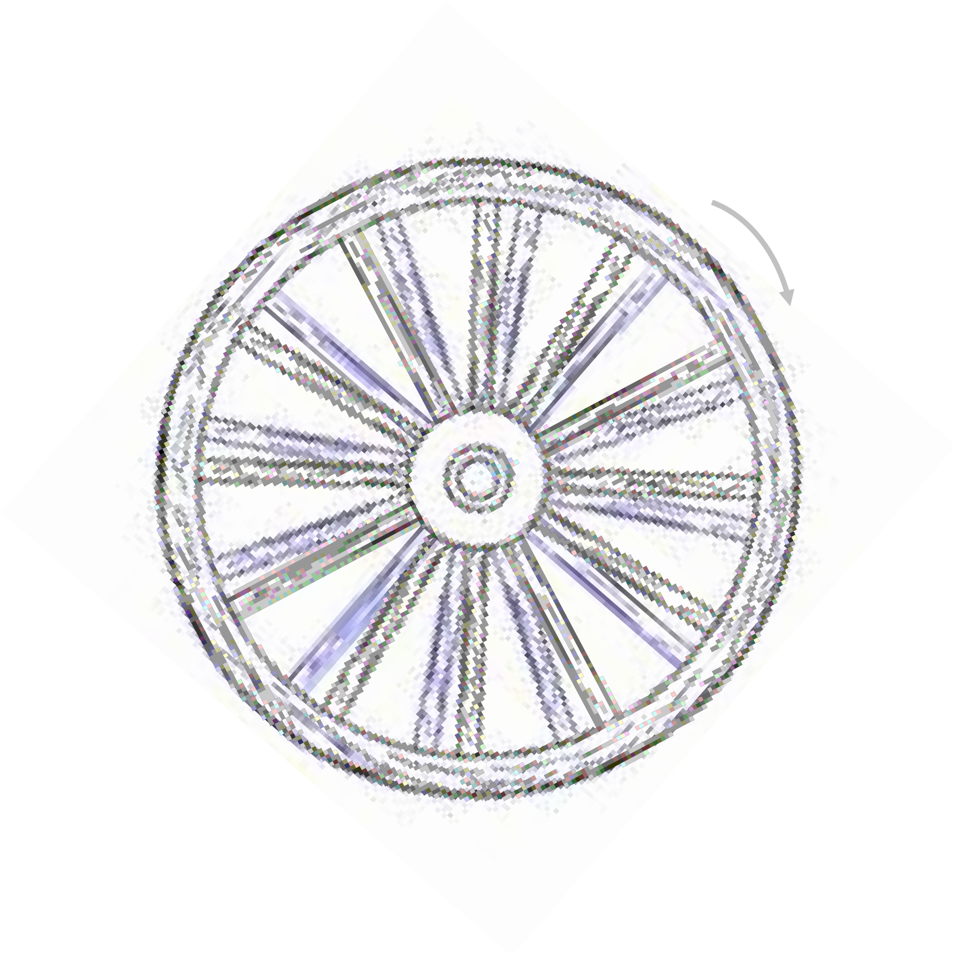
- Strobing
- Flickering



Temporal Aliasing

Artifacts due to limited temporal resolution

- Strobing
- Flickering



Antialiasing

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

Image Processing

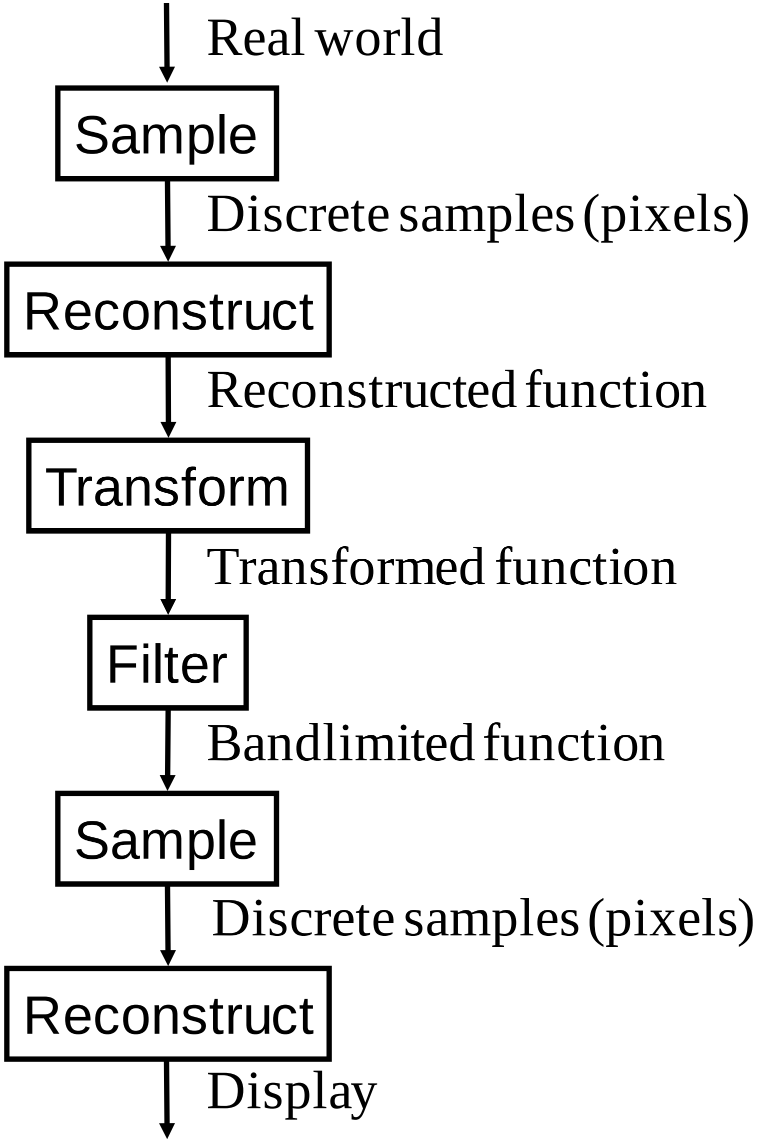


Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

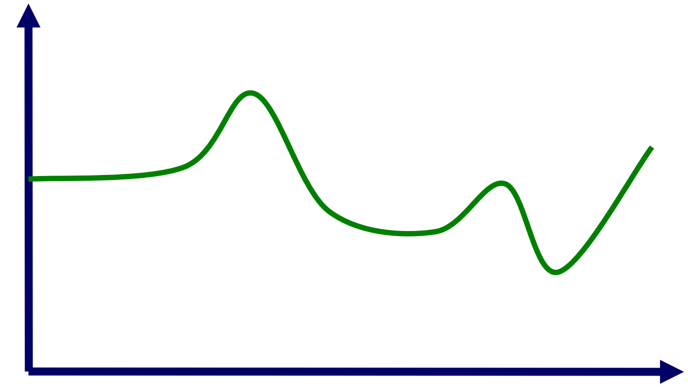
Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display



Continuous Function

Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

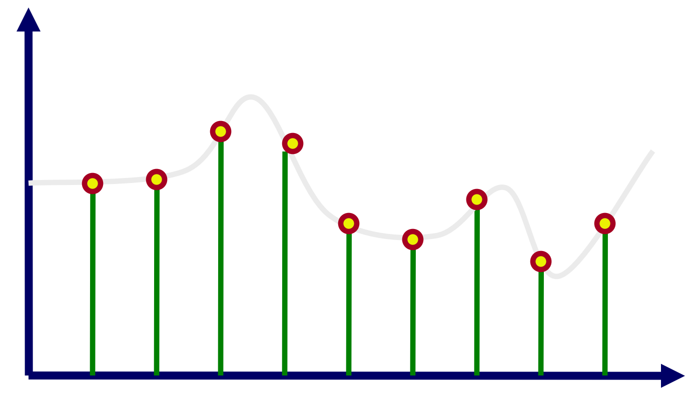
Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display



Discrete Samples

Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

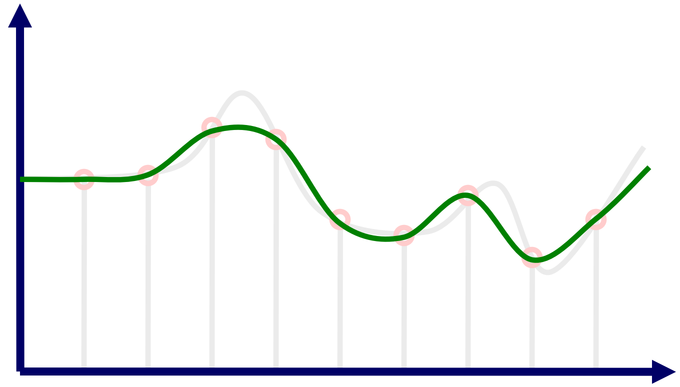
Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display



Reconstructed Function

Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

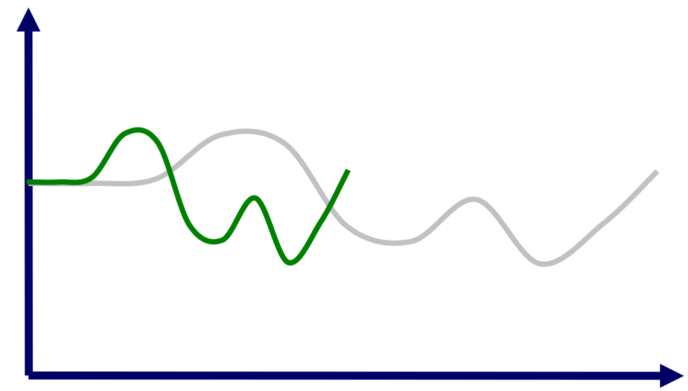
Bandlimited function

Sample

Discrete samples (pixels)

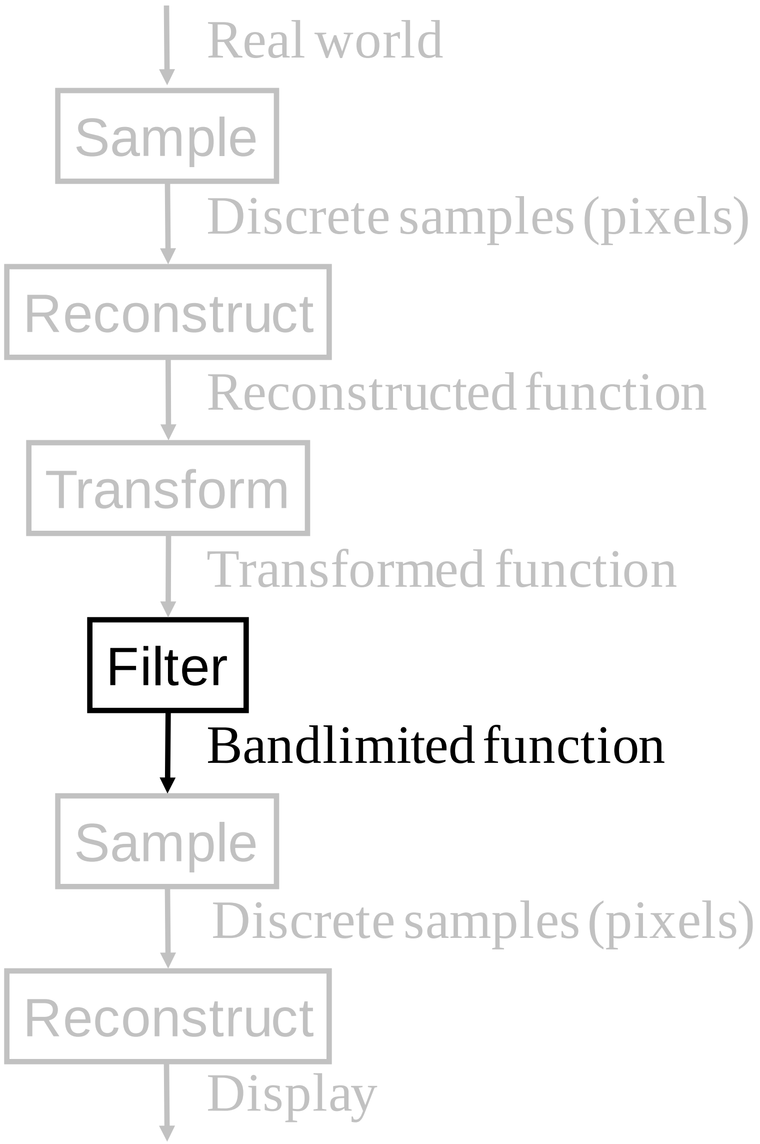
Reconstruct

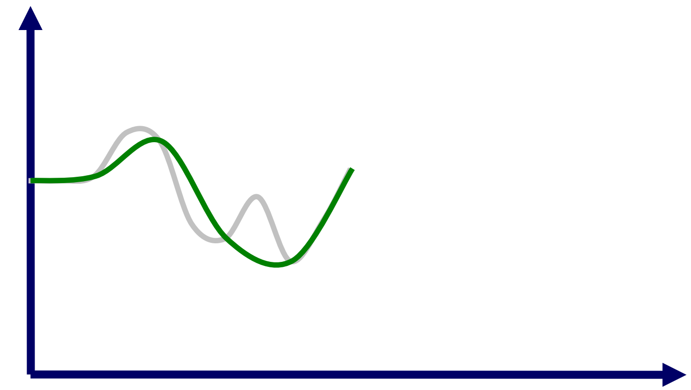
Display



Transformed Function

Image Processing





Bandlimited Function

Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

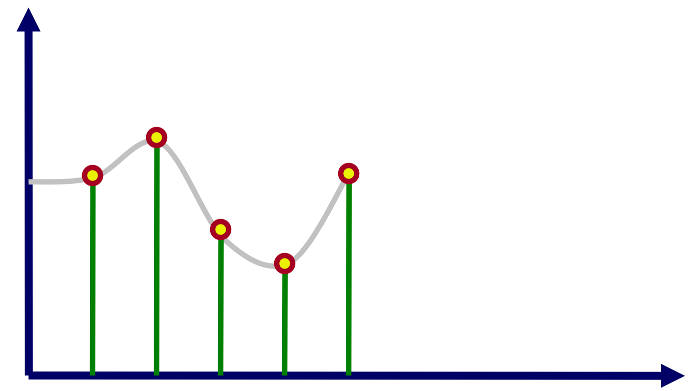
Bandlimited function

Sample

Discrete samples (pixels)

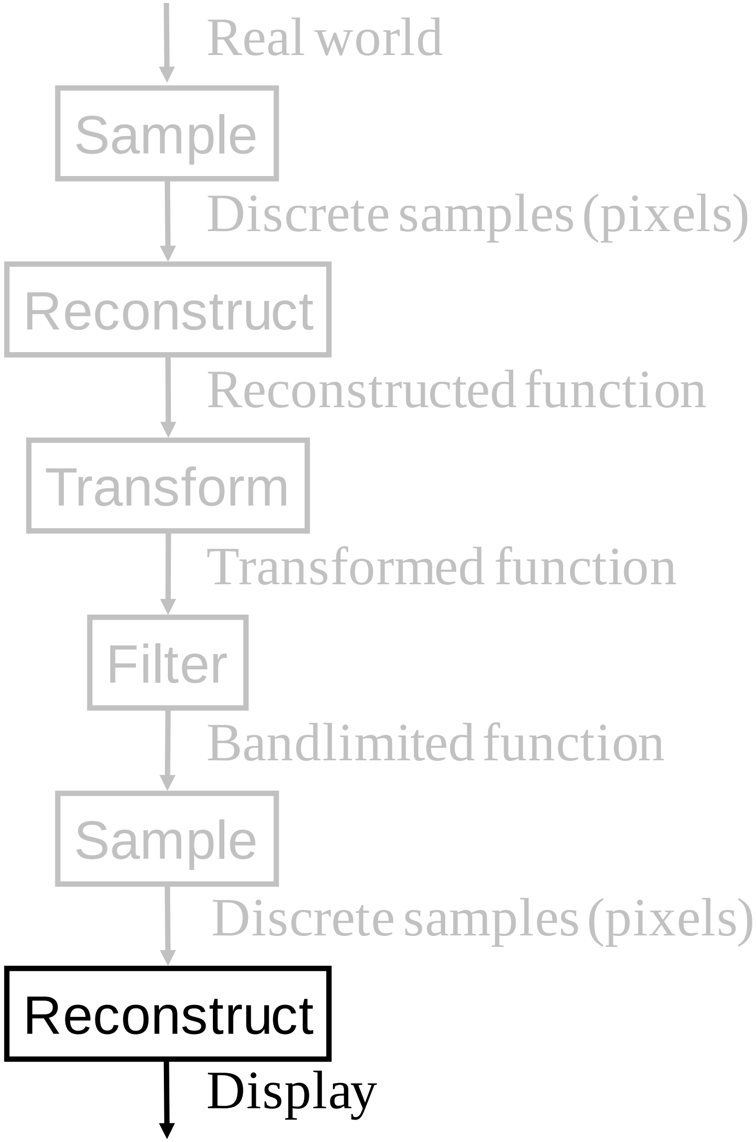
Reconstruct

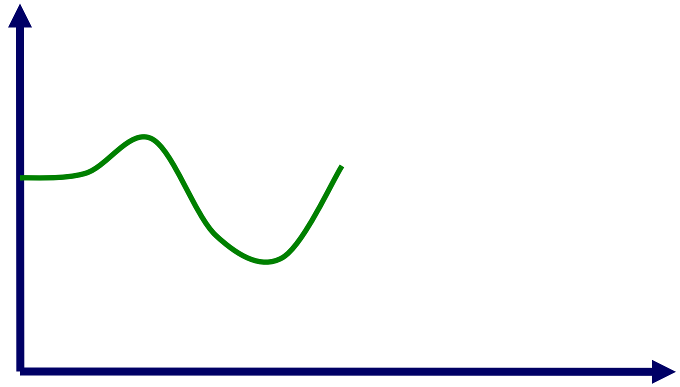
Display



Discrete samples

Image Processing

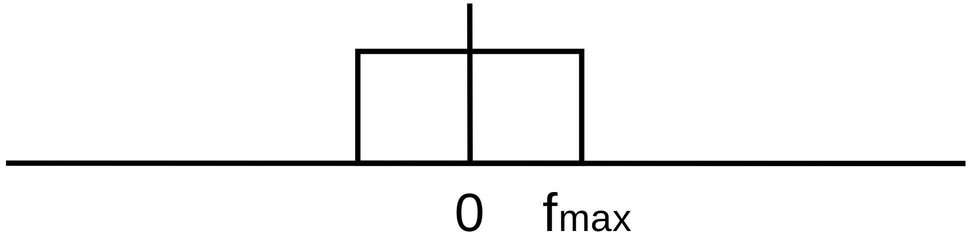




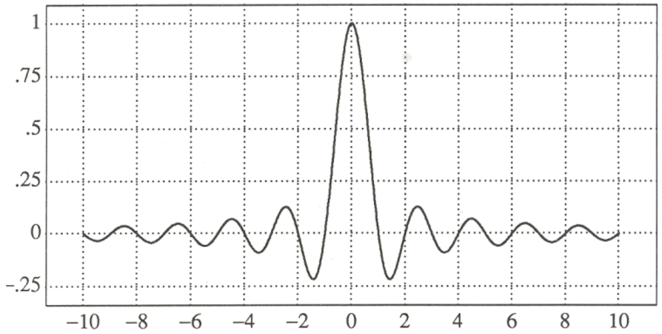
Display

Ideal Bandlimiting Filter

- Frequency domain



- Spatial domain

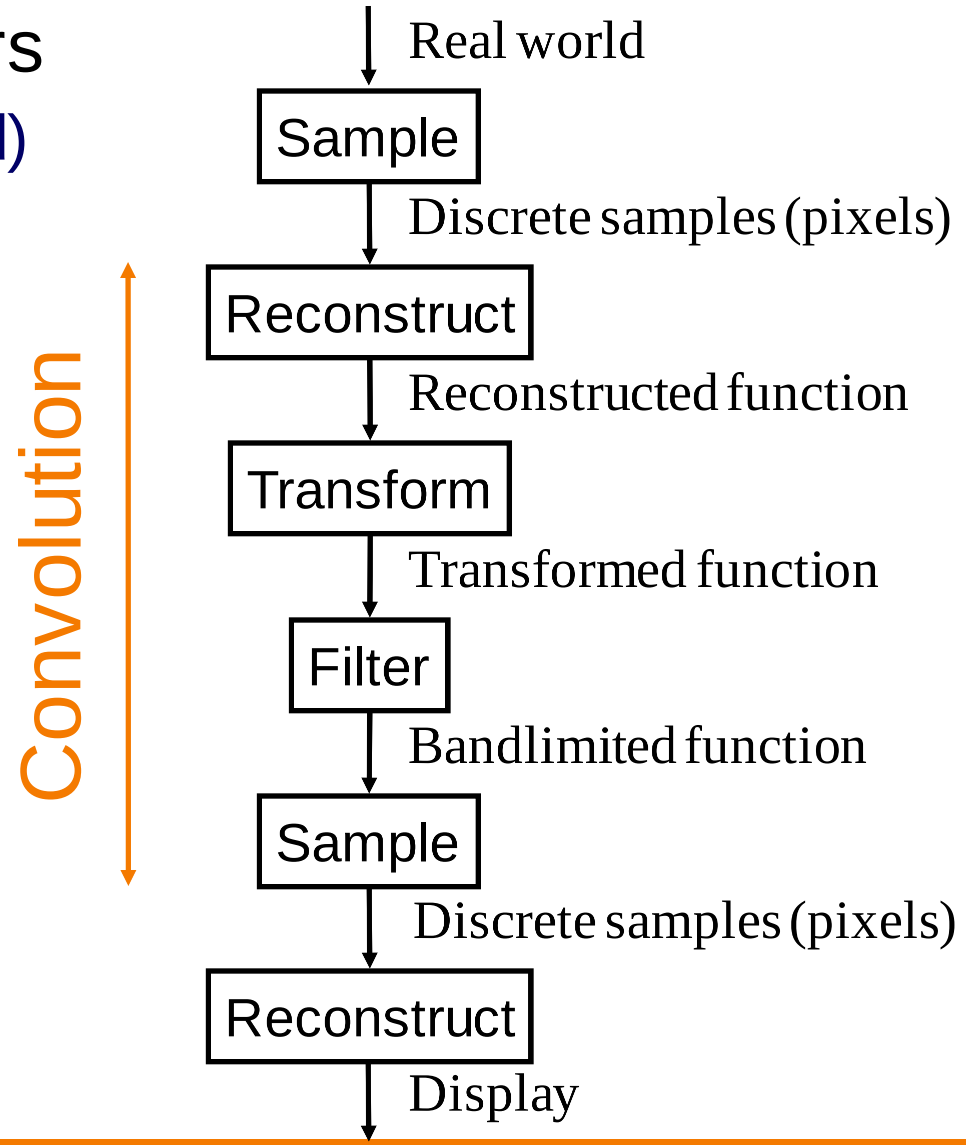


$$Sinc(x) = \frac{\sin \pi x}{\pi x}$$

Figure 4.5 Wolberg

Practical Image Processing

- Finite low-pass filters
  - Point sampling (bad)
  - Box filter
  - Triangle filter
  - Gaussian filter



Example: Scaling

- Resample with triangle or Gaussian filter

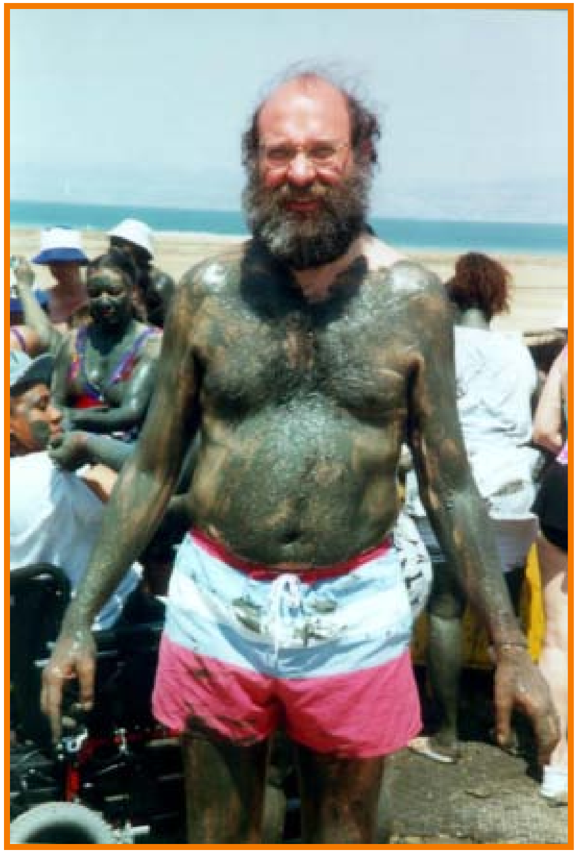
Original

1/4X resolution

4X resolution

General Image Warping

- Move pixels of an image



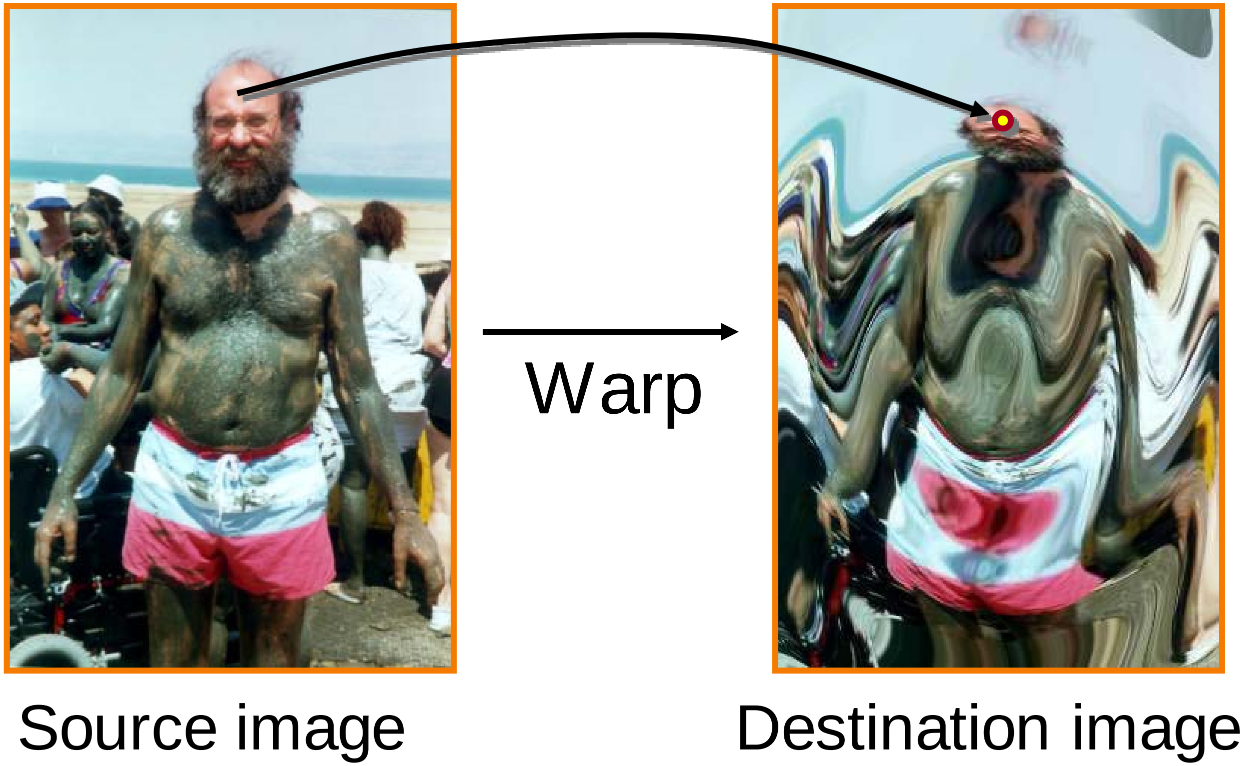
Source image

Warp

Destination image

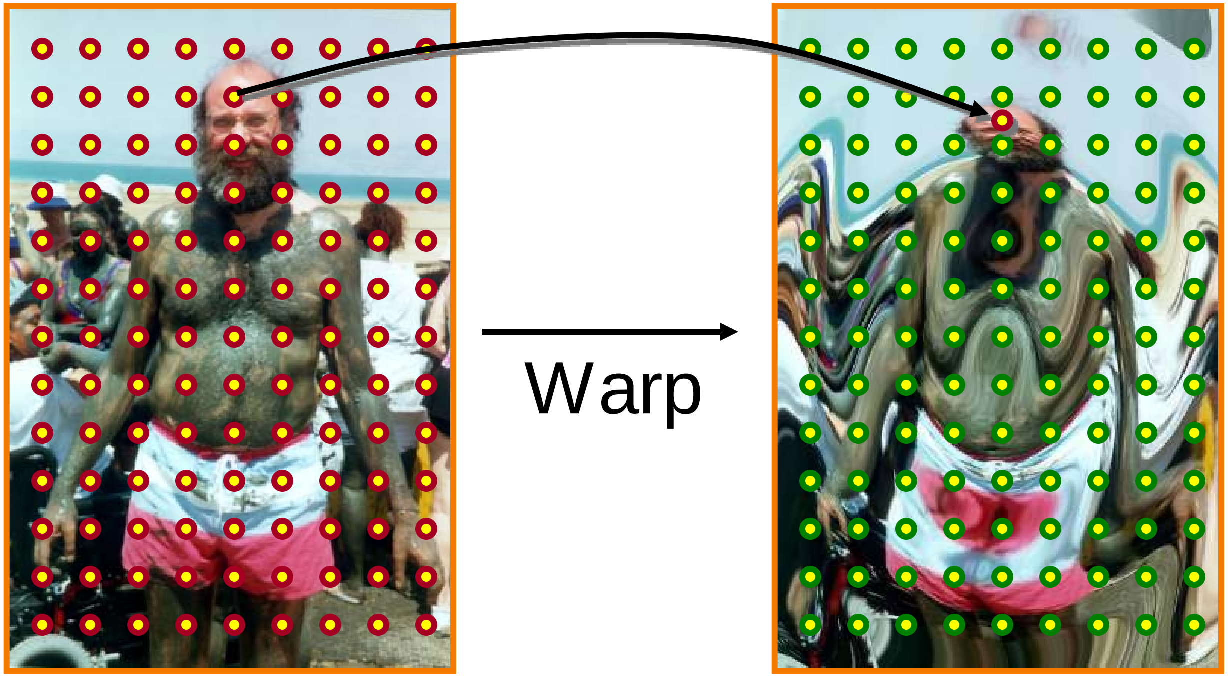
General Image Warping

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



General Image Warping

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

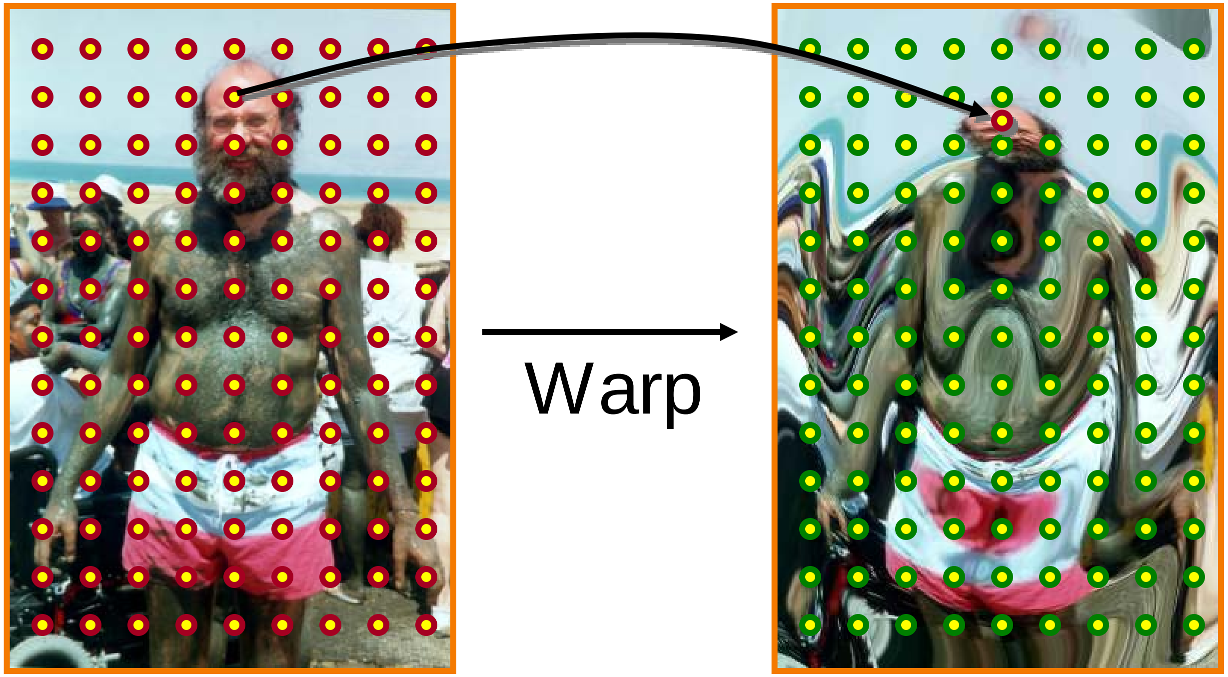


Source image

Destination image

General Image Warping

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

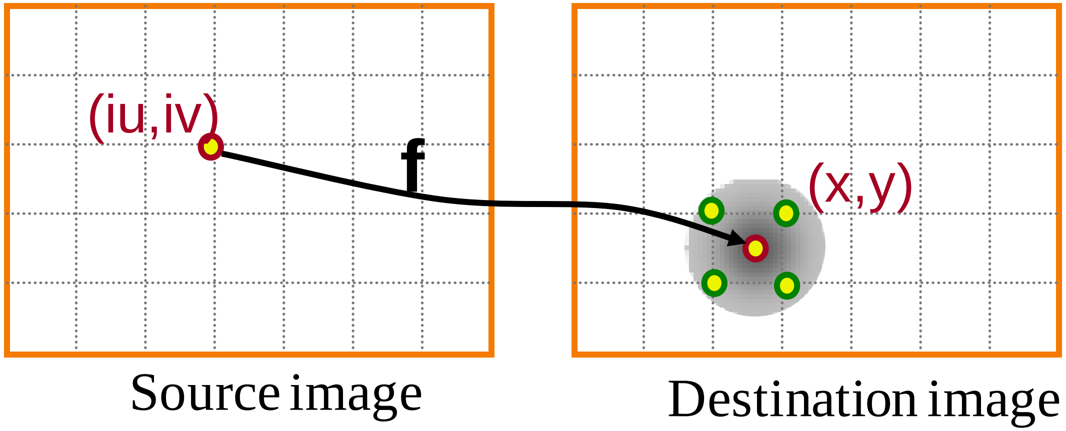


Source image

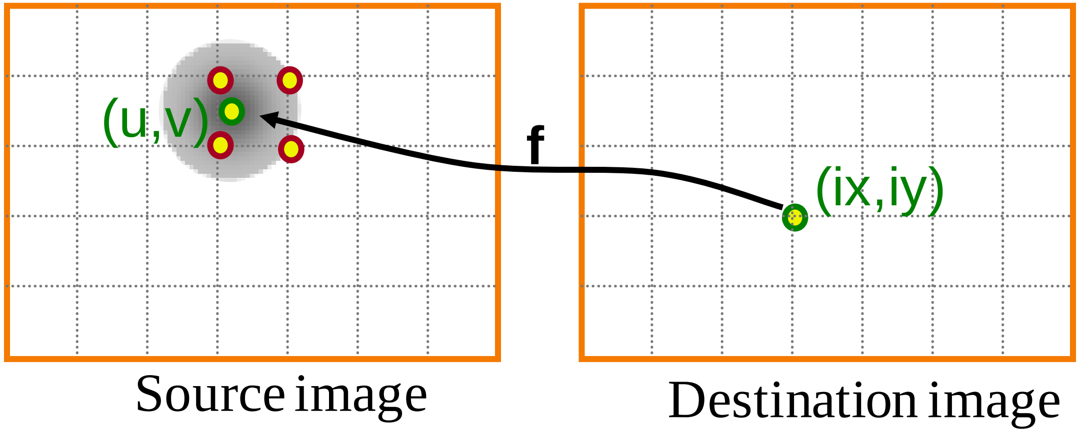
Destination image

Two Options

- Forward mapping

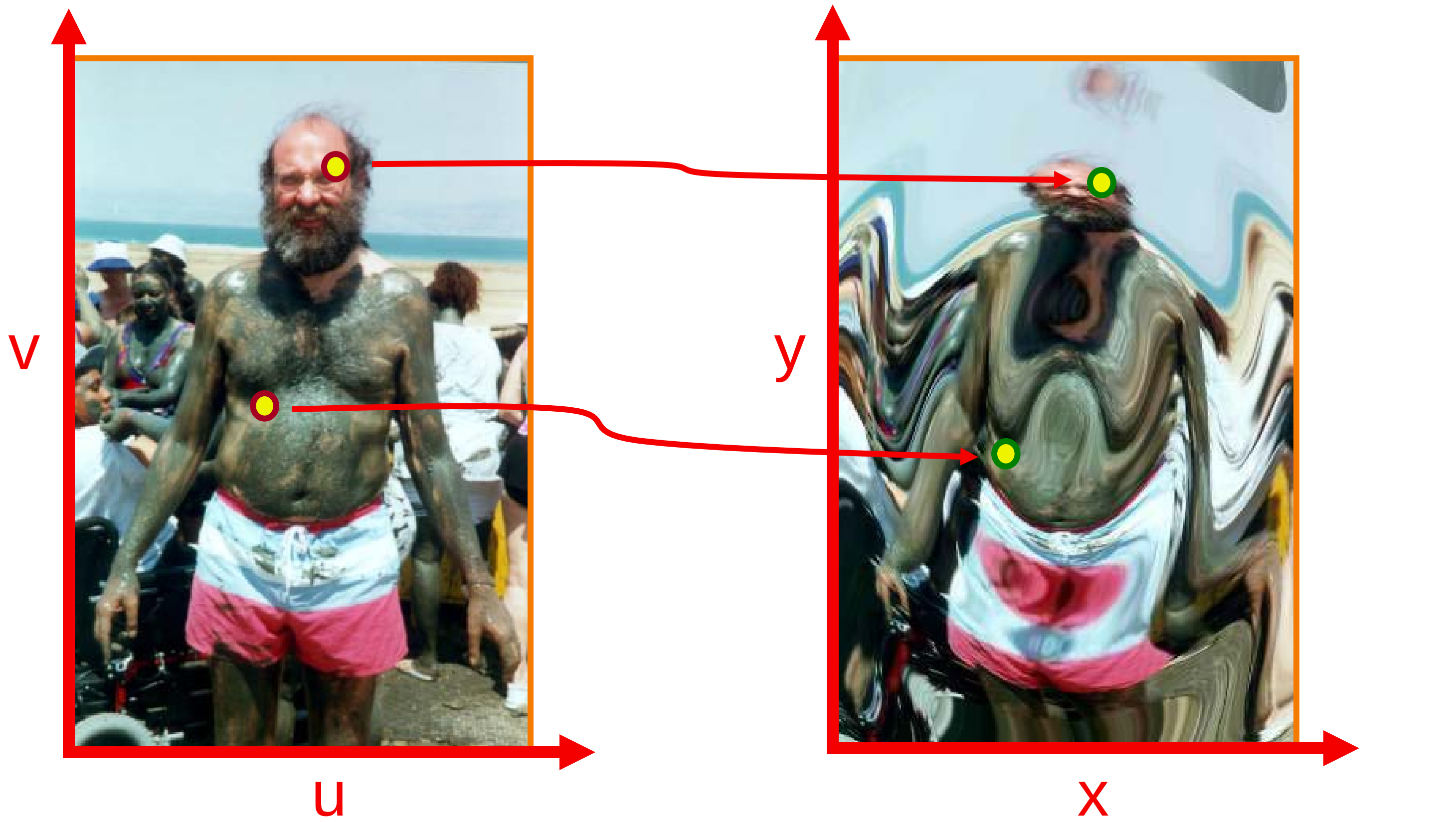


- Reverse mapping



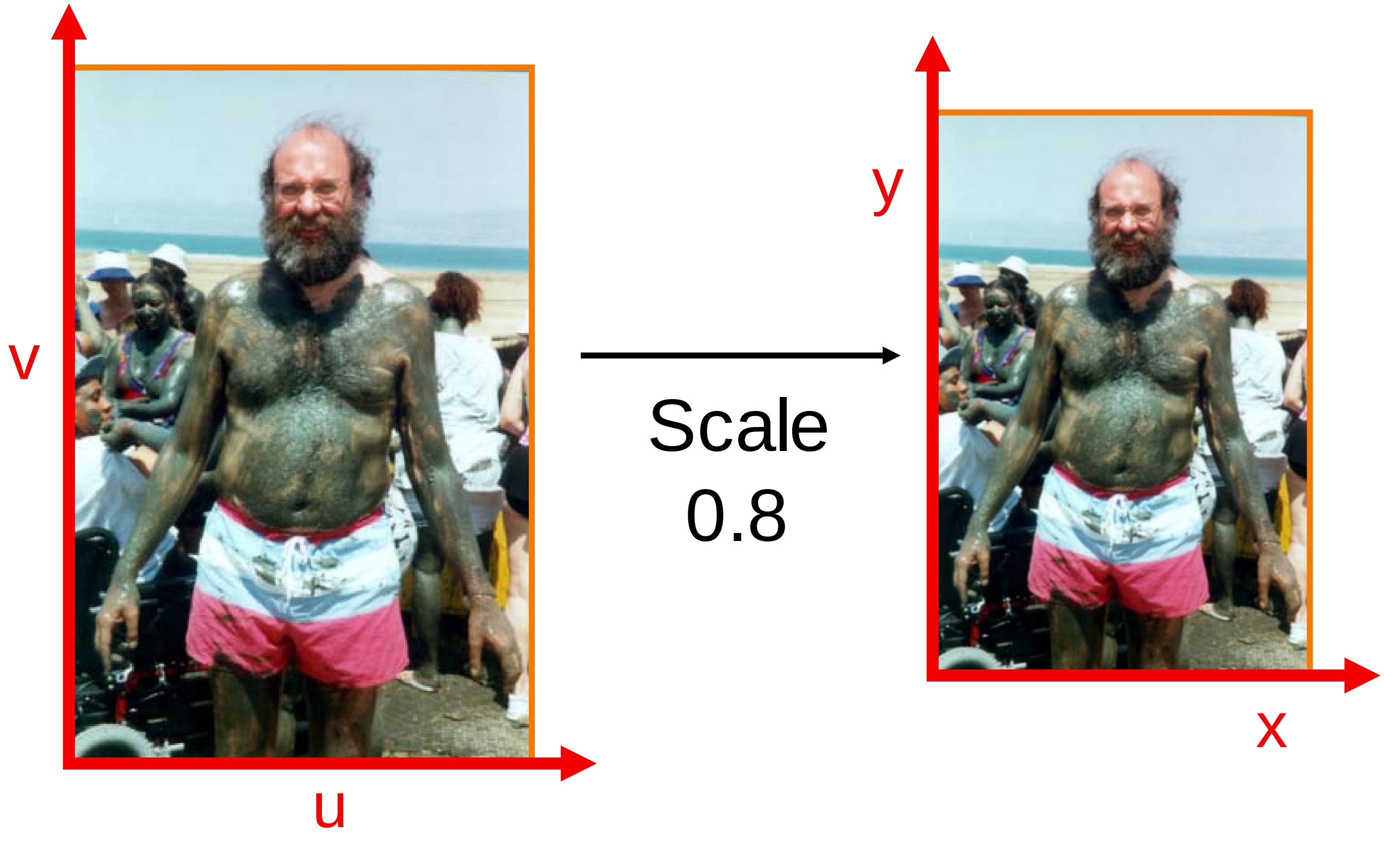
Mapping

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



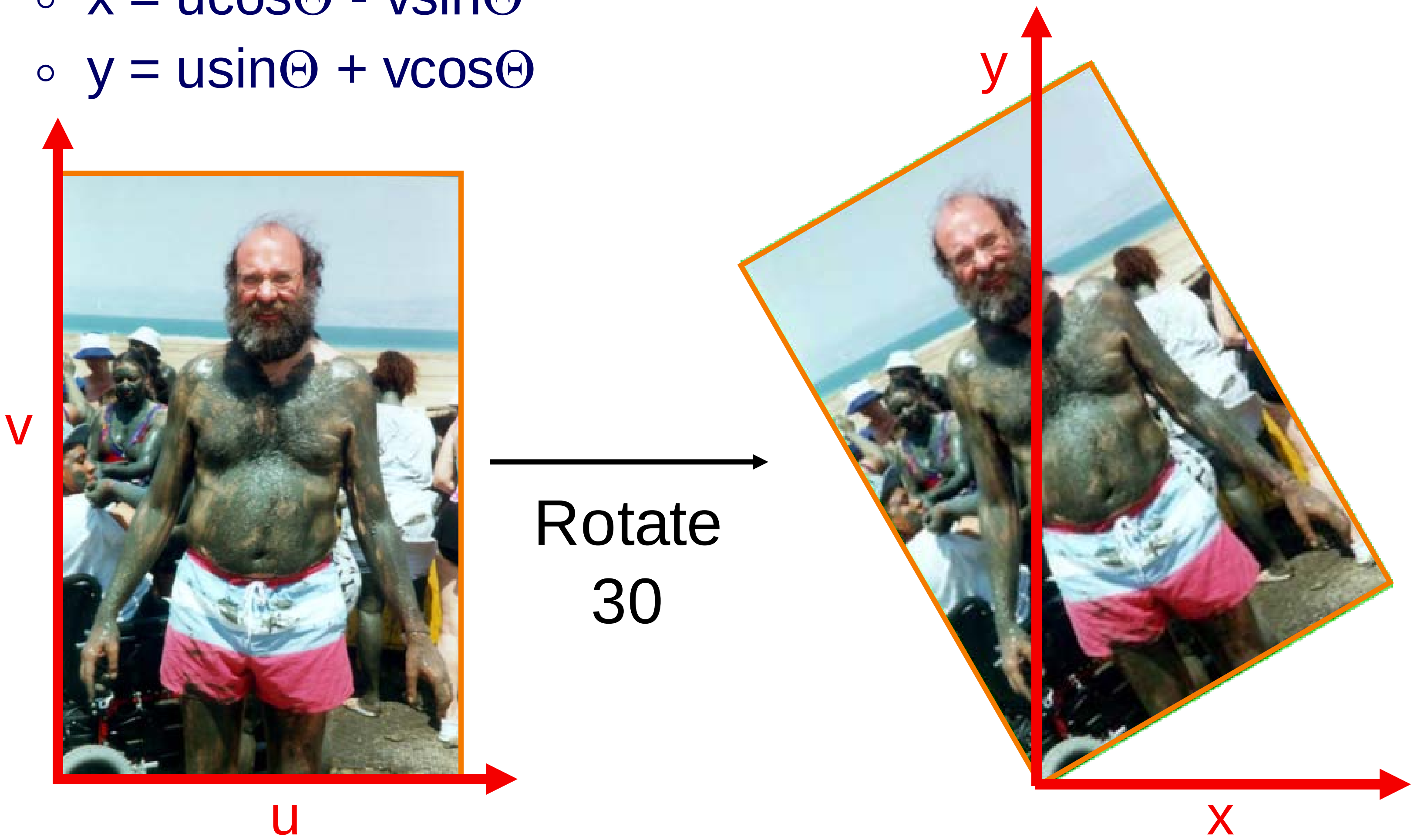
Parametric Mappings

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



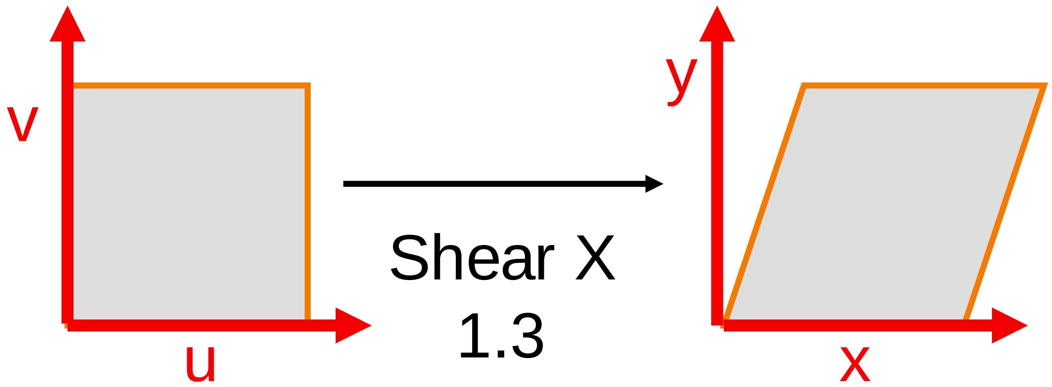
Parametric Mappings

- Rotate by Θ degrees:
  - $x = u\cos\Theta - v\sin\Theta$
  - $y = u\sin\Theta + v\cos\Theta$

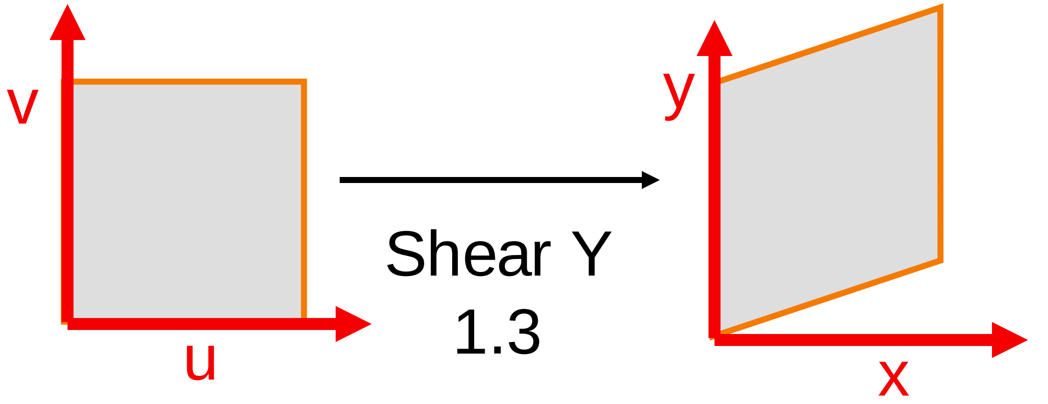


Parametric Mappings

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

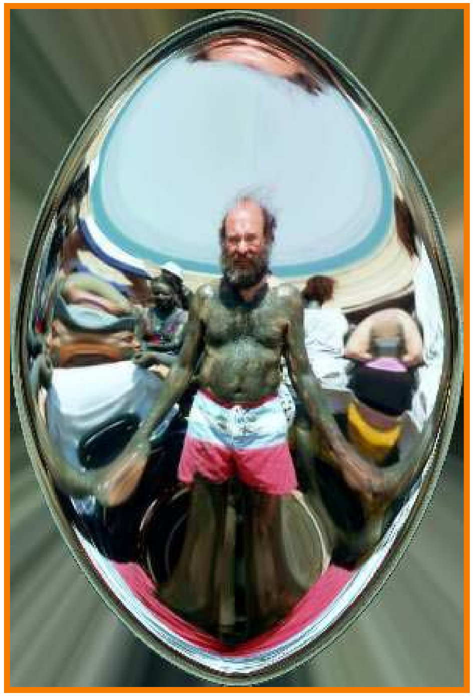


- Shear in Y by *factor:*
  - x = u
  - y = v + *factor* * u



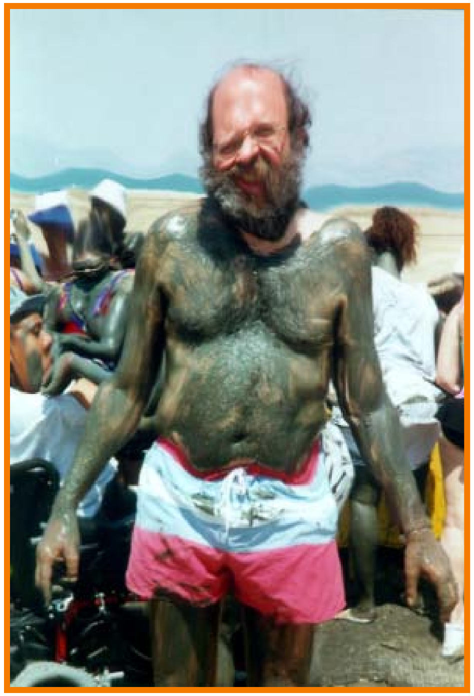
Other Parametric Mappings

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



Fish-eye

"Swirl"



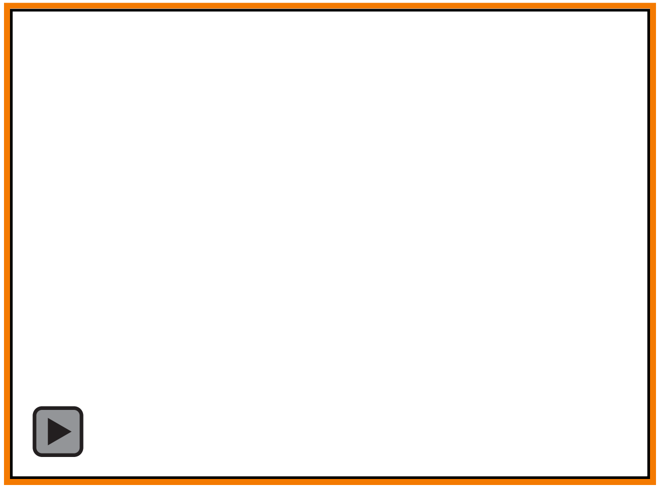
"Rain"

COS426 Examples

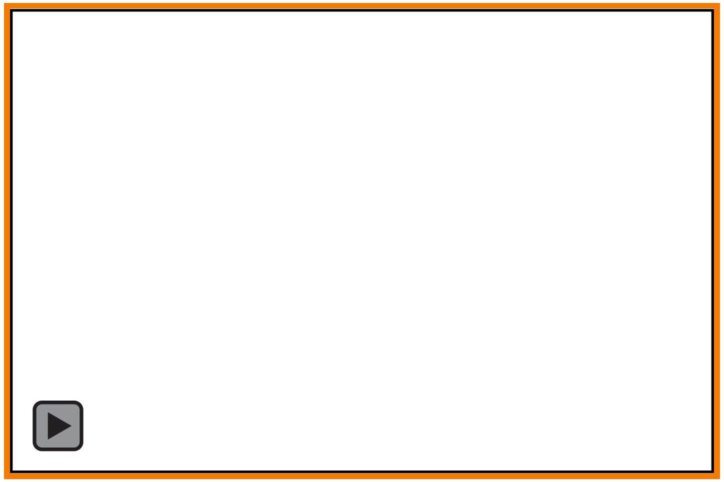
Aditya Bhaskara

Wei Xiang

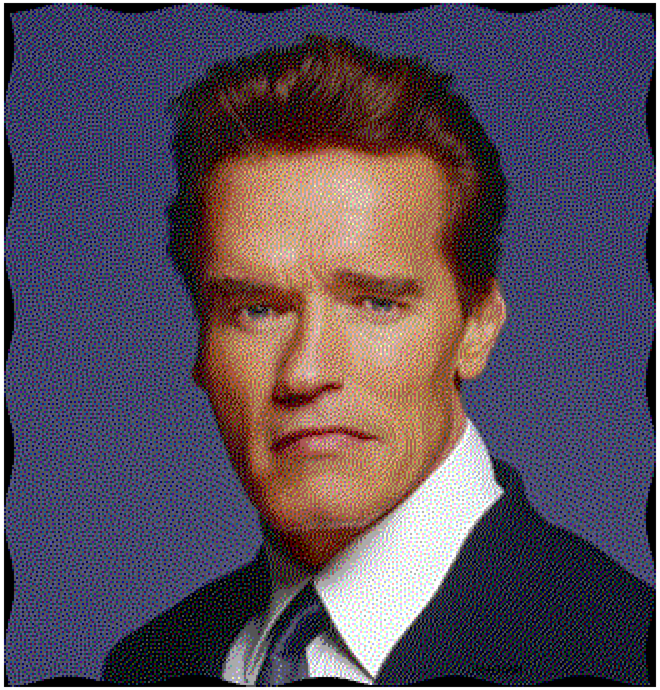
More COS426 Examples



Sid Kapur



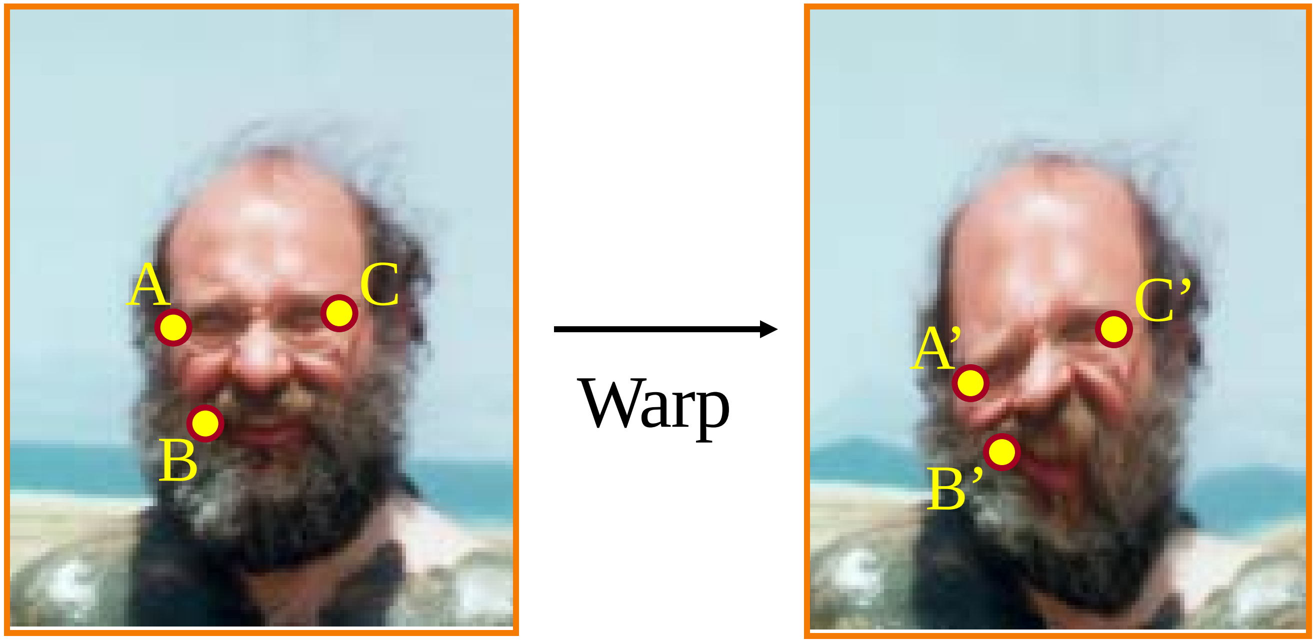
Michael Oranato



Eirik Bakke

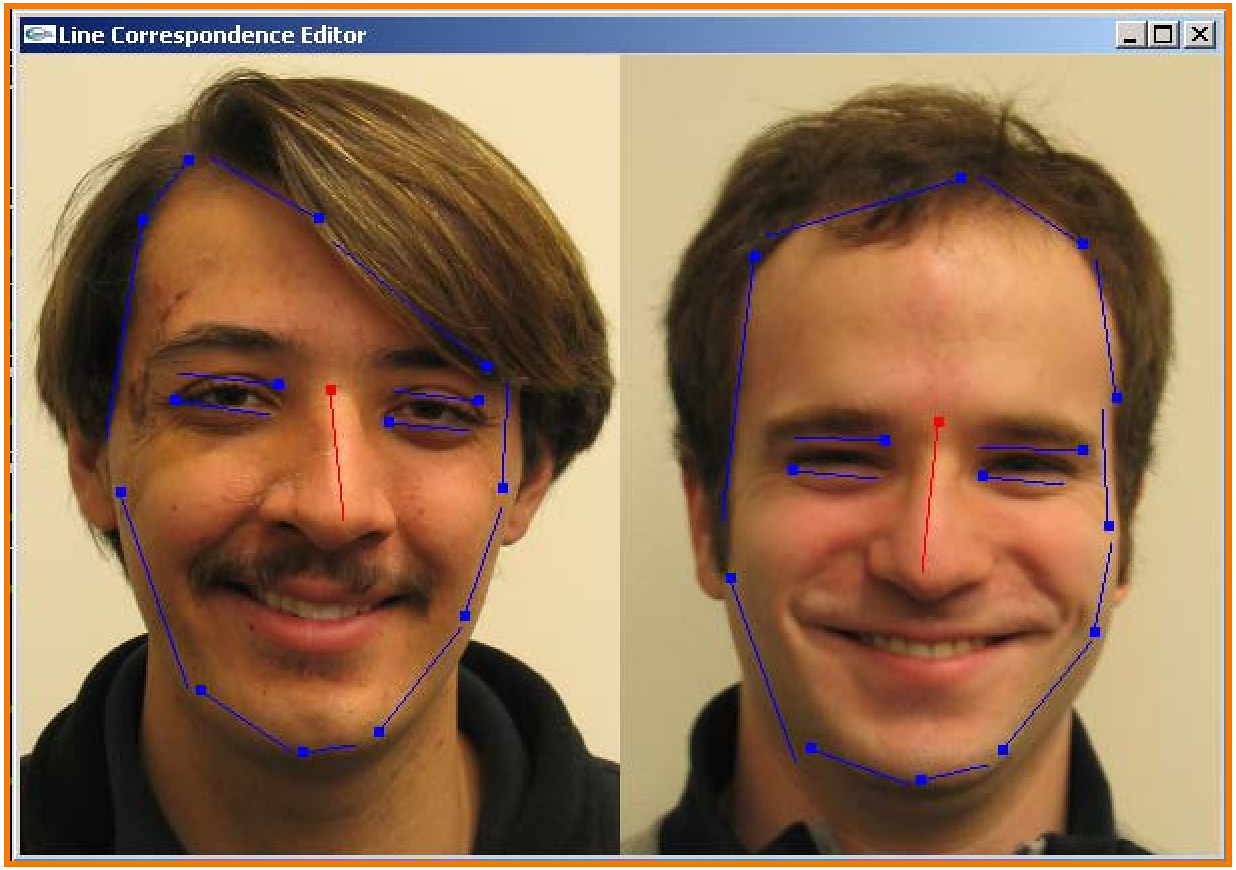
Point Correspondence Mappings

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



Line Correspondence Mappings

- Beier & Neeley use pairs of lines to specify warp



Beier & Neeley
SIGGRAPH 92

Image Warping

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

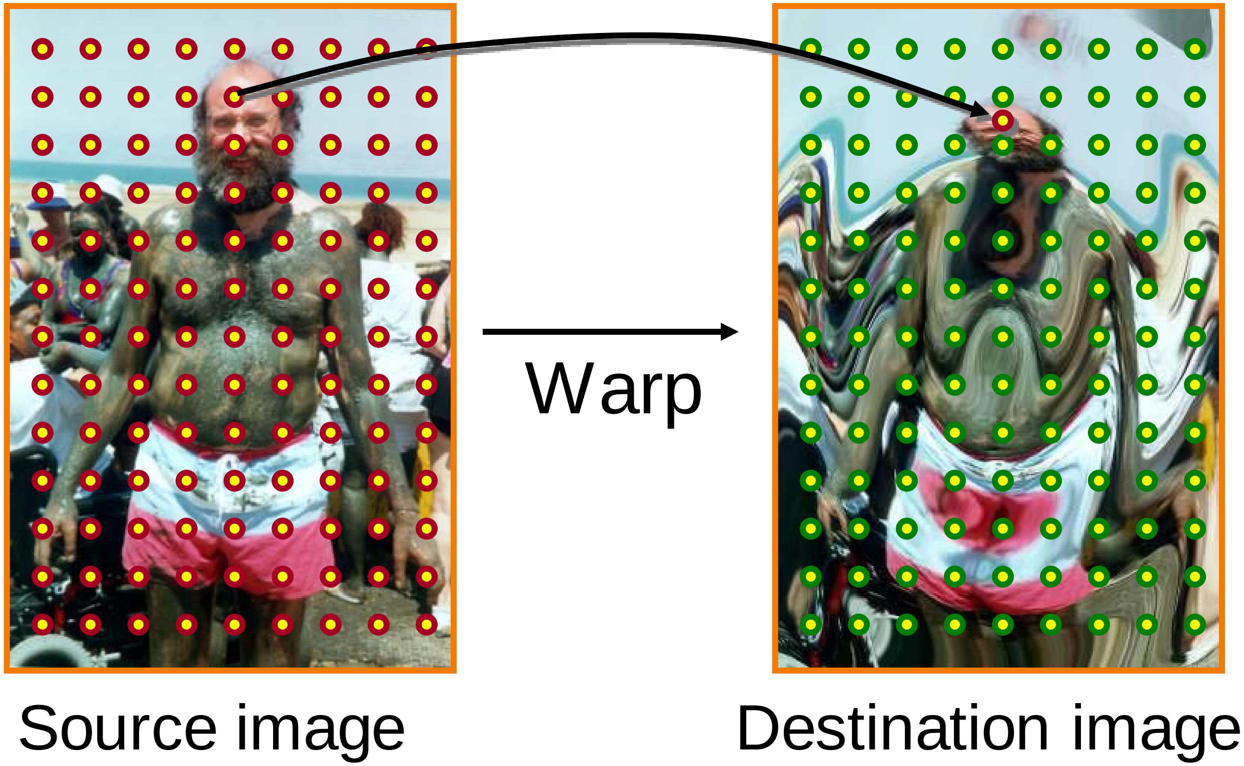
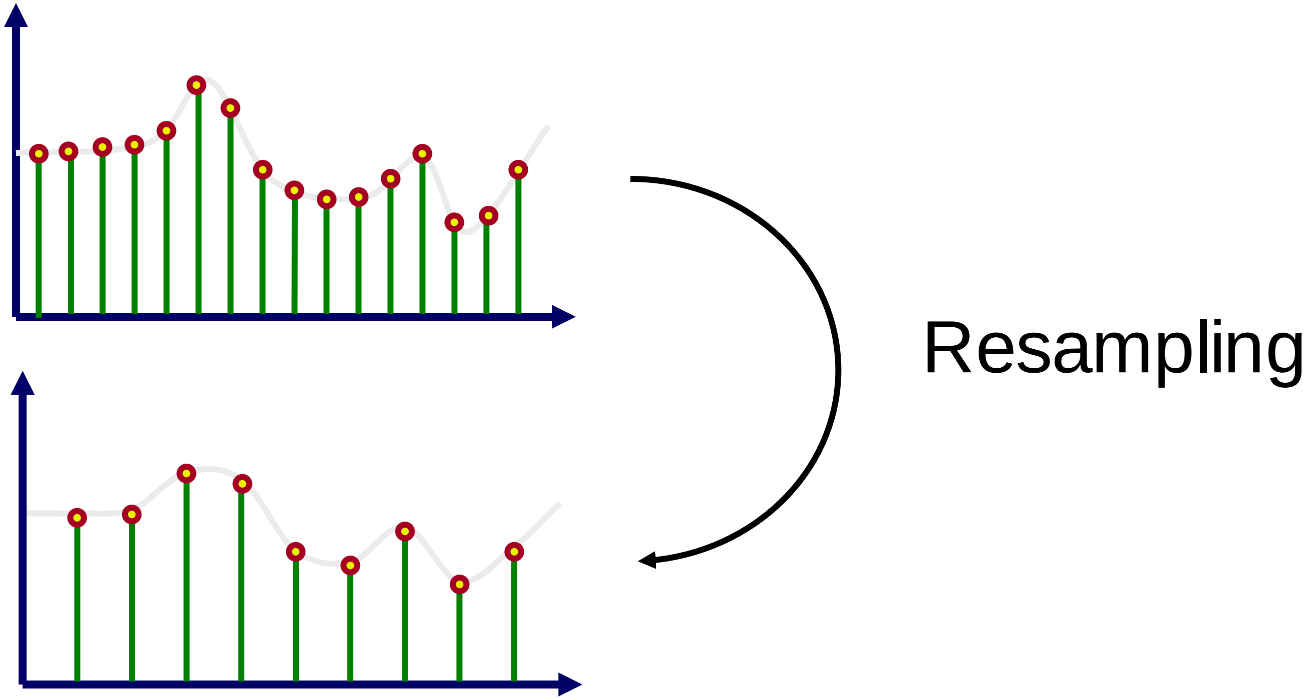


Image Warping

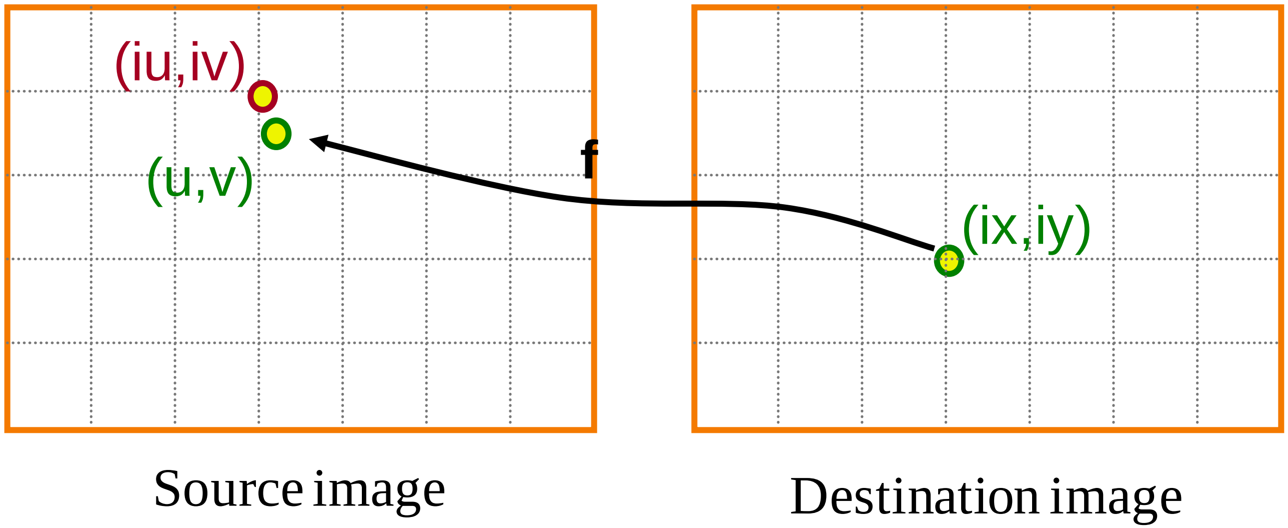
- Image warping requires resampling of image



Point Sampling

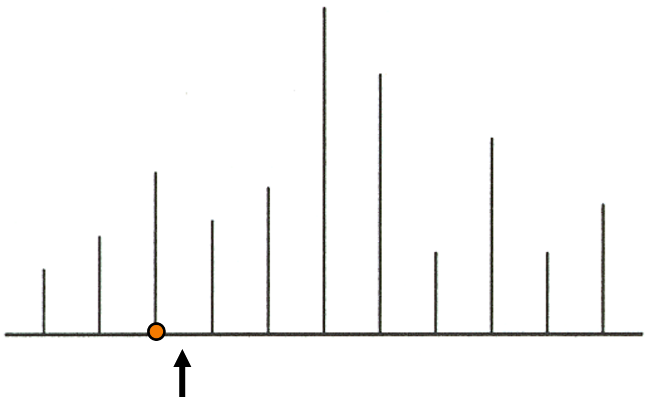
- Possible (poor) resampling implementation:

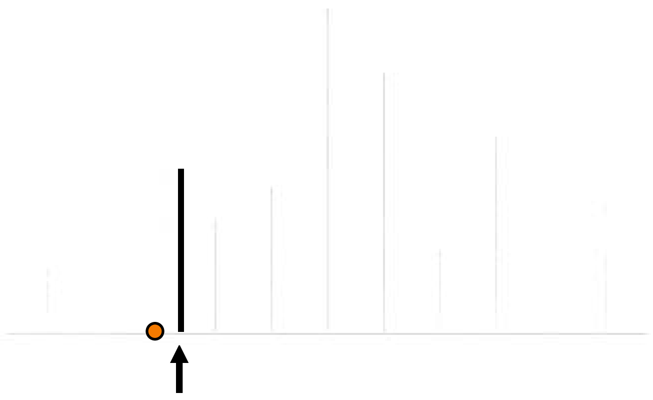
```
float Resample(src, u, v, k, w) {
  int iu = round(u);
  int iv = round(v);
  return src(iu,iv);
}
```



Point Sampling

- Use nearest sample





Input

Output

Point Sampling

Point Sampled: Aliasing!

Correctly Bandlimited

Image Resampling Pipeline

- Ideal resampling requires correct filtering to avoid artifacts
- Reconstruction filter especially important when magnifying
- Bandlimiting filter especially important when minifying

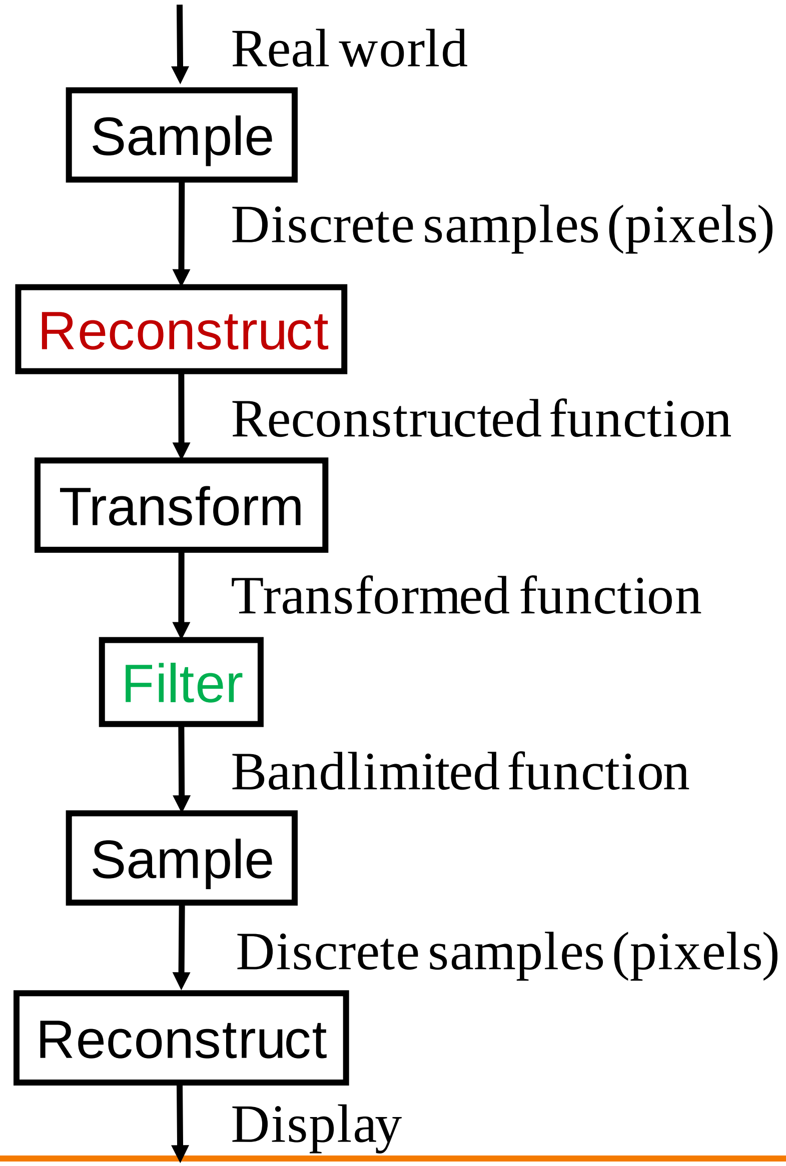
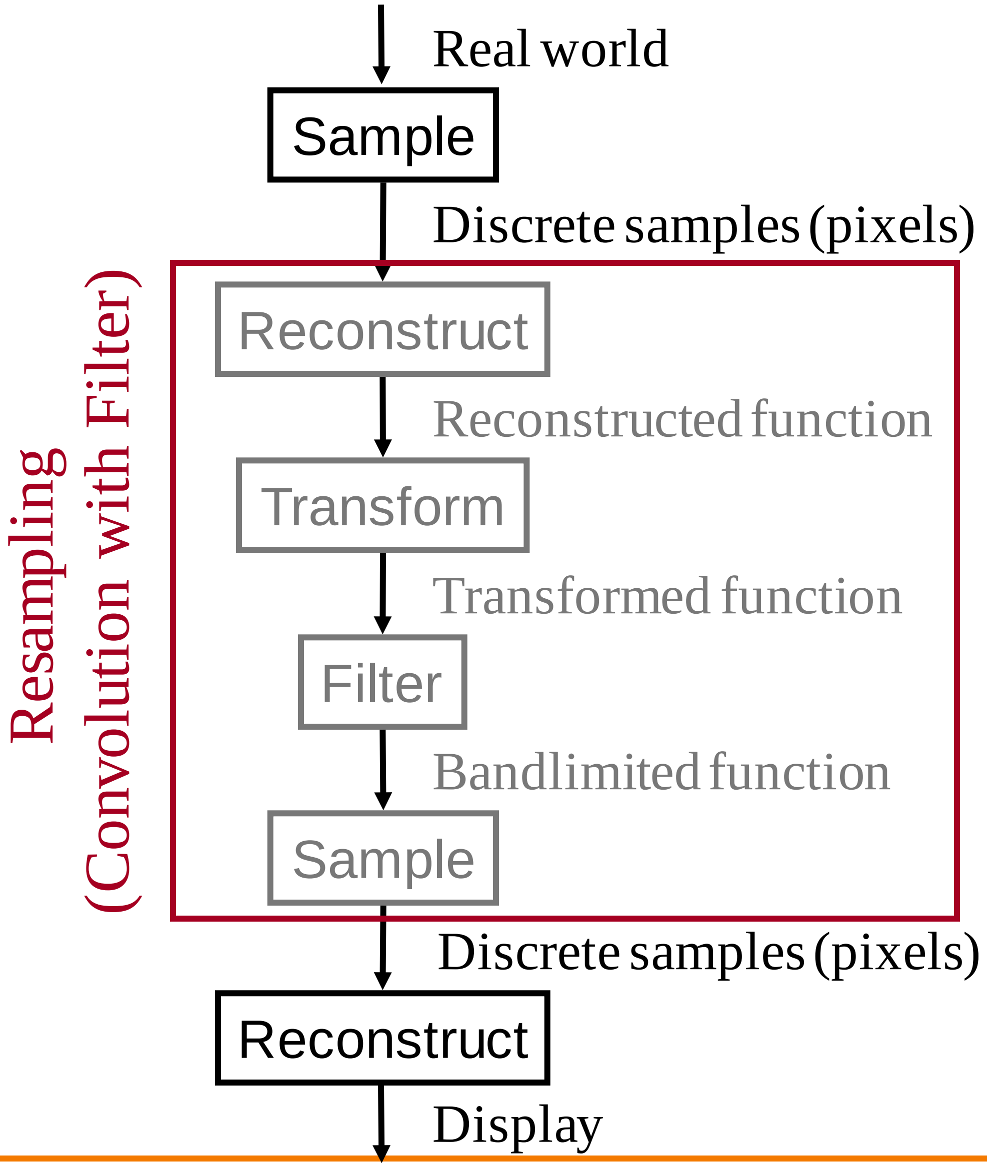


Image Resampling Pipeline

- In practice:
  Resampling with
  low-pass filter
  in order to reduce
  aliasing artifacts
  when minifying



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;
}
```

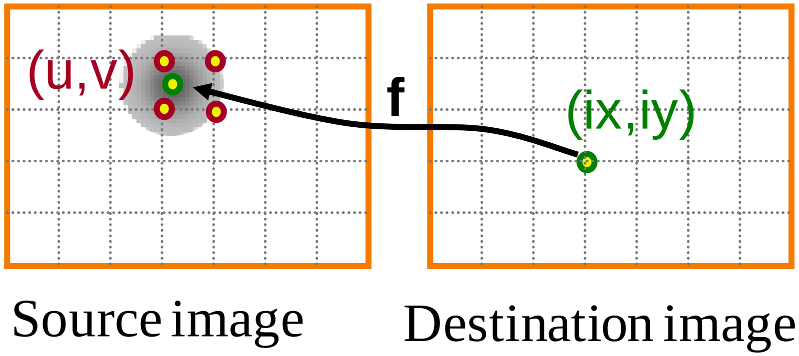
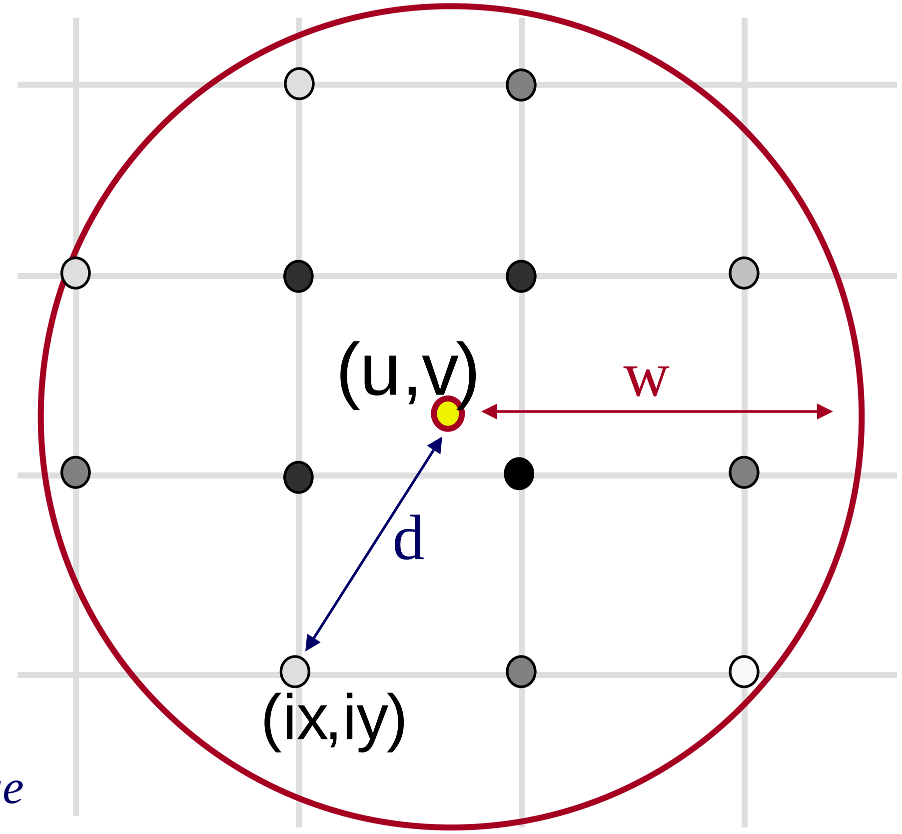


Image Resampling

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

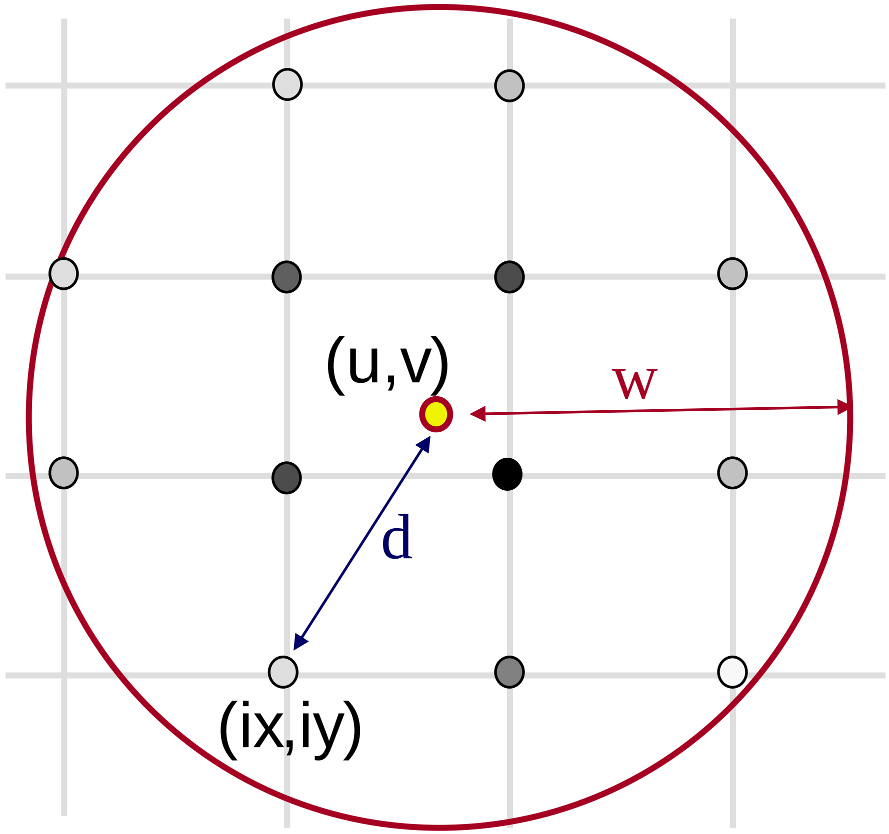
```
dst(ix,iy) = 0;
for (ix = u-w; ix <= u+w; ix++)
  for (iy = v-w; iy <= v+w; iy++)
    d = dist (ix,iy)↔(u,v)
    dst(ix,iy) += k(ix,iy)*src(ix,iy);
```

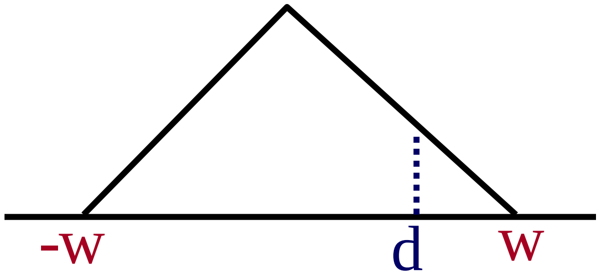


k(ix,iy) represented by gray value

Image Resampling

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





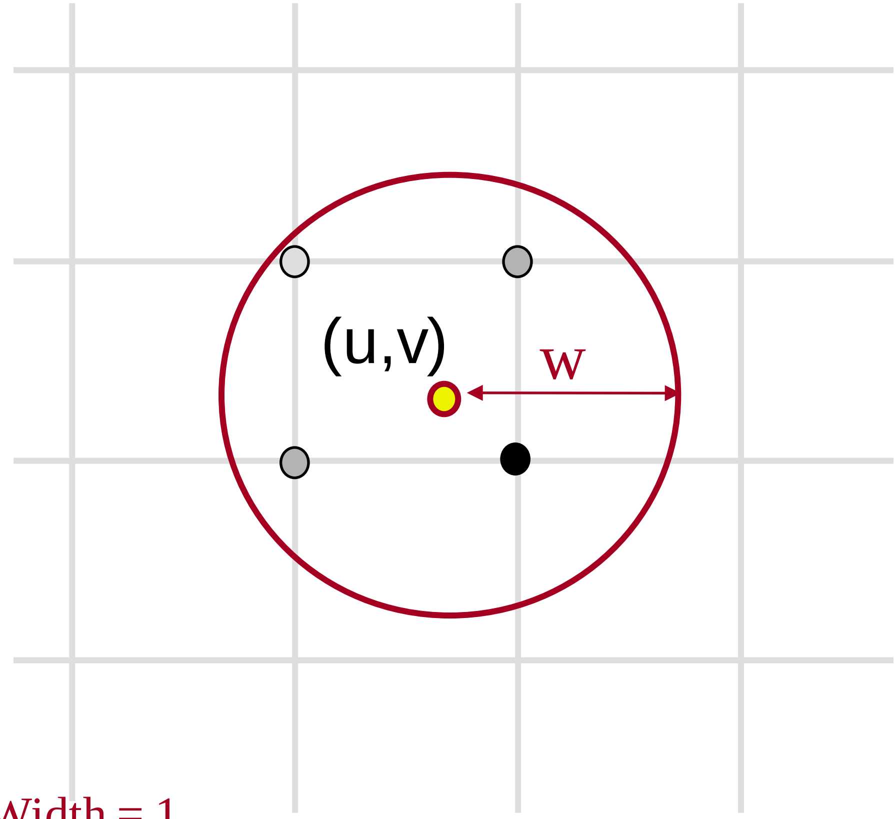
Triangle filter

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

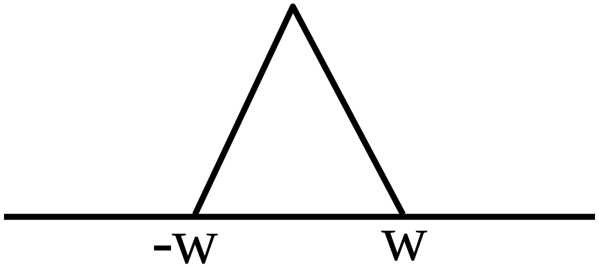
Filter Width = 2

Image Resampling

- For isotropic Triangle and Gaussian filters, k(ix,iy) is function of d and w
  - Filter width chosen based on scale factor (or blur)



Filter Width = 1

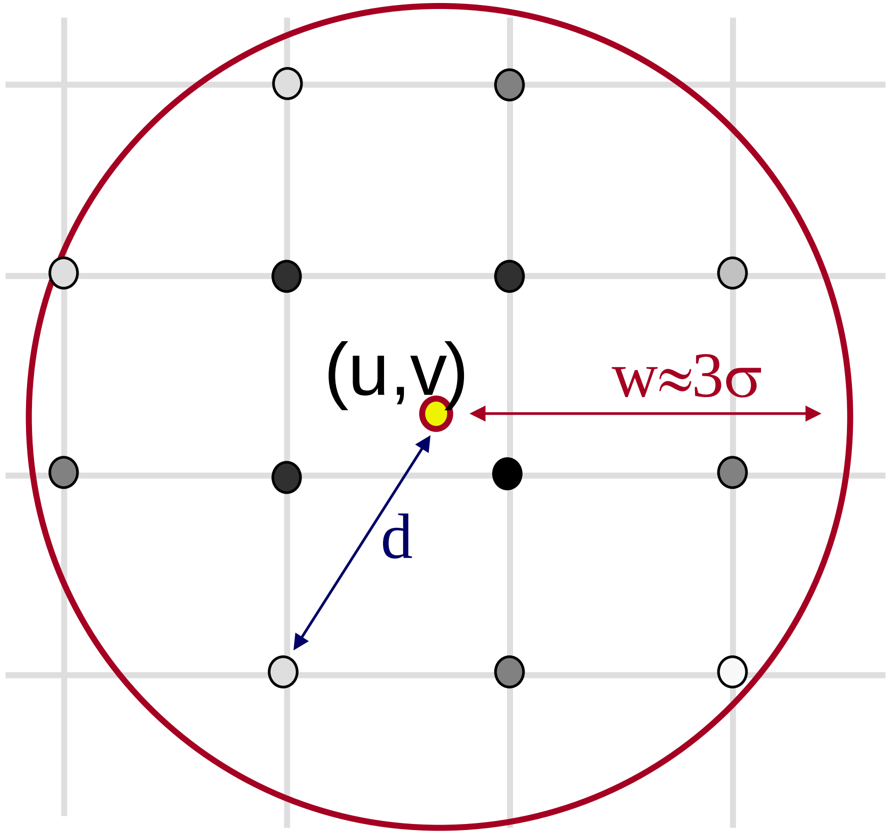


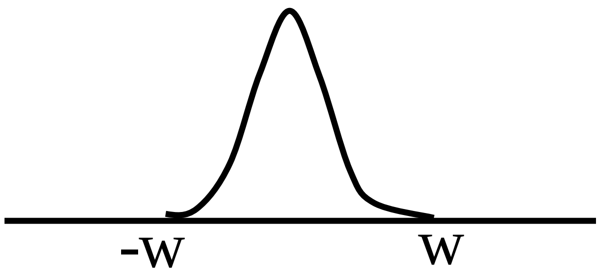
Width of filter affects blurriness

Gaussian Filtering

- Kernel is Gaussian function

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



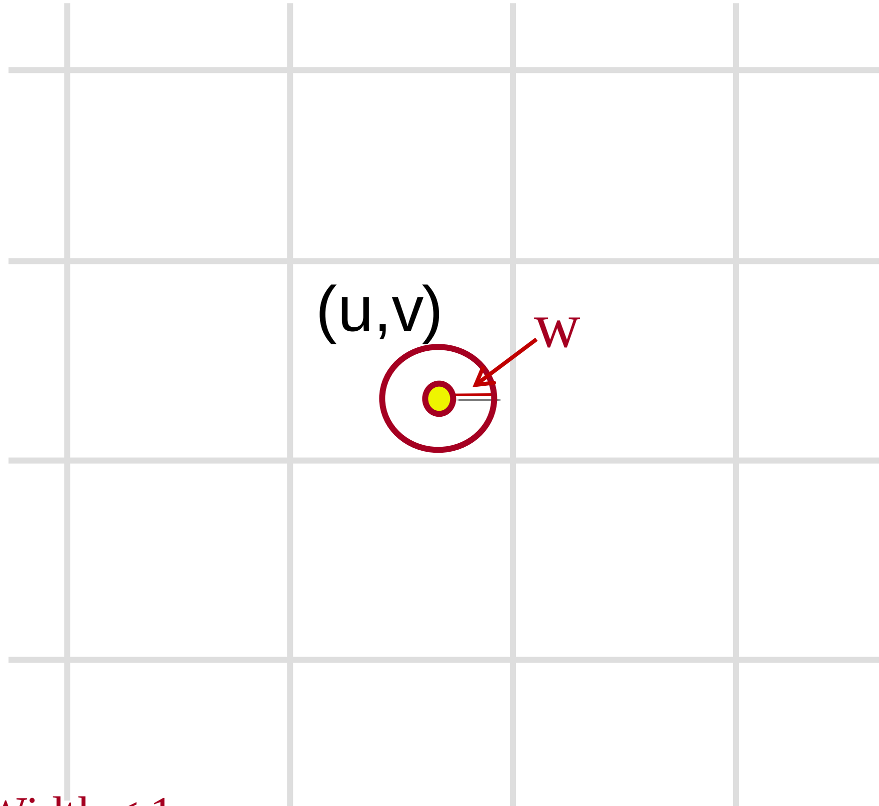


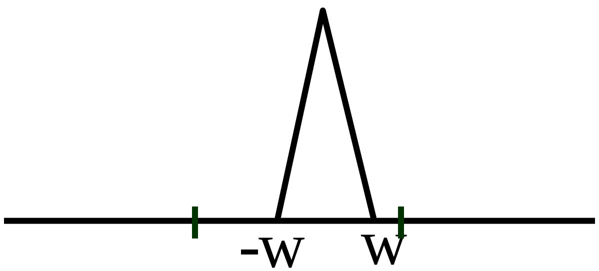
Gaussian Function

- 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?



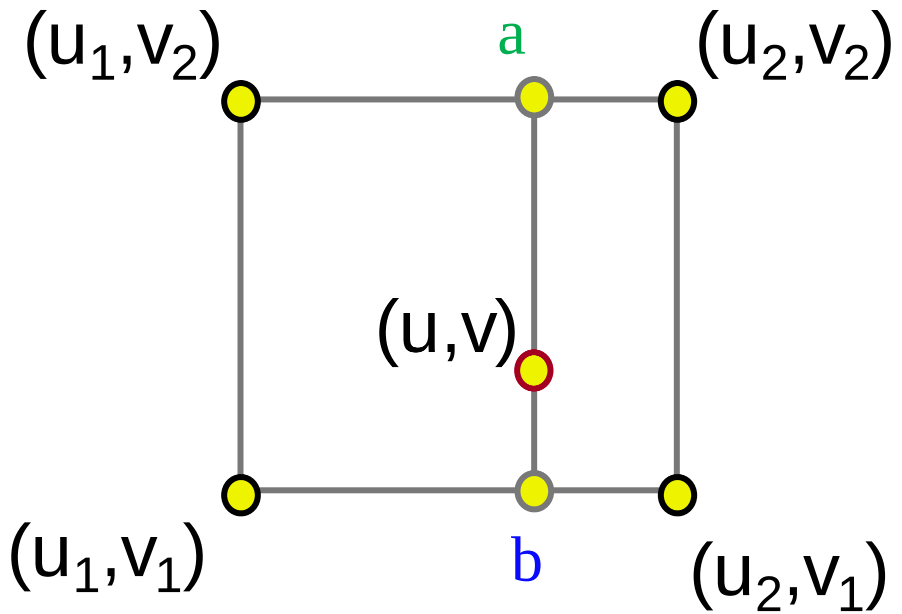


Triangle filter

Filter Width < 1

Image Resampling (with width < 1)

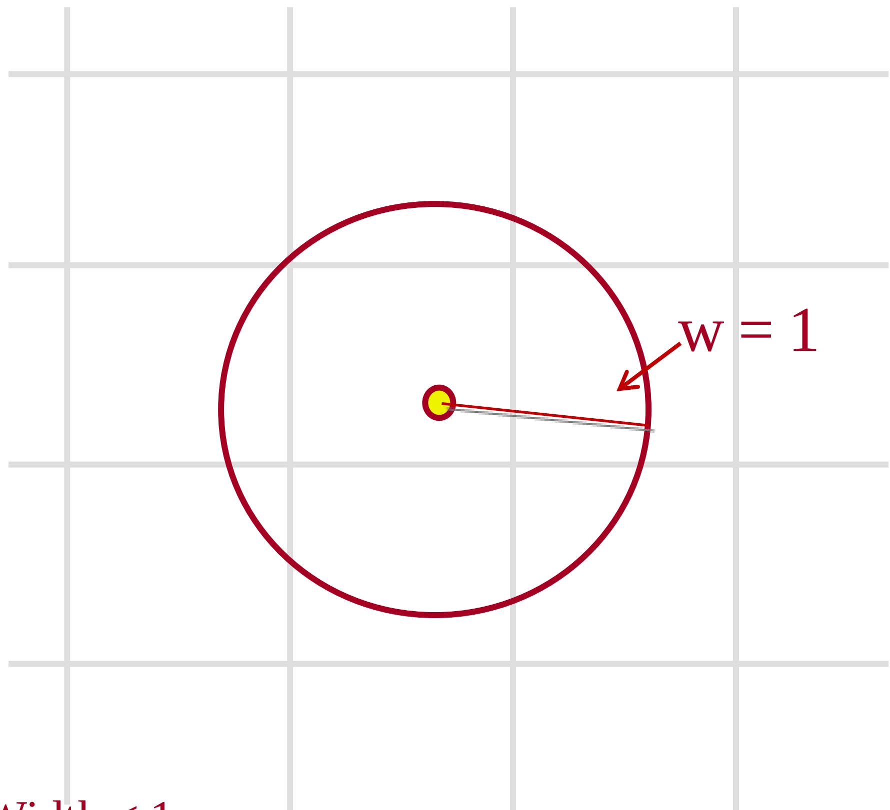
- Reconstruction filter: Bilinearly interpolate four closest pixels
  - a = linear interpolation of $src(u_1,v_2)$ and $src(u_2,v_2)$
  - b = linear interpolation of $src(u_1,v_1)$ and $src(u_2,v_1)$
  - dst(x,y) = linear interpolation of "a" and "b"

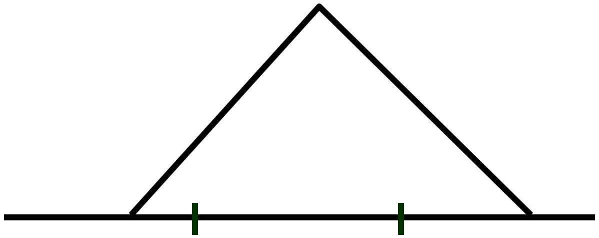


Filter Width < 1

Image Resampling (with width < 1)

- Alternative: force width to be at least 1



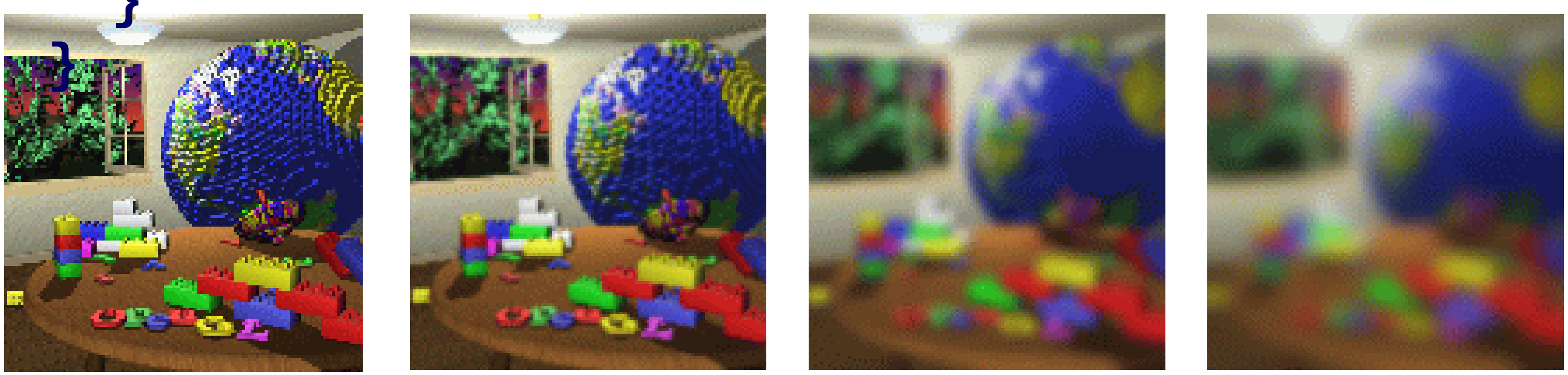


Filter Width < 1

Putting it All Together

- Possible implementation of image blur:

```
Blur(src, dst, sigma) {
  w ≈ 3*sigma;
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float u = ix;
      float v = iy;
      dst(ix,iy) = Resample(src,u,v,k,w);
    }
  }
}
```

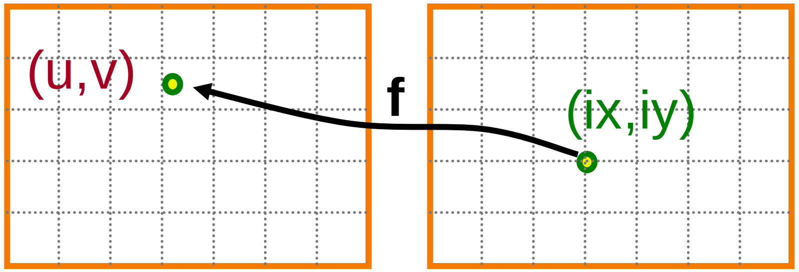


Increasing sigma

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);
    }
  }
}
```

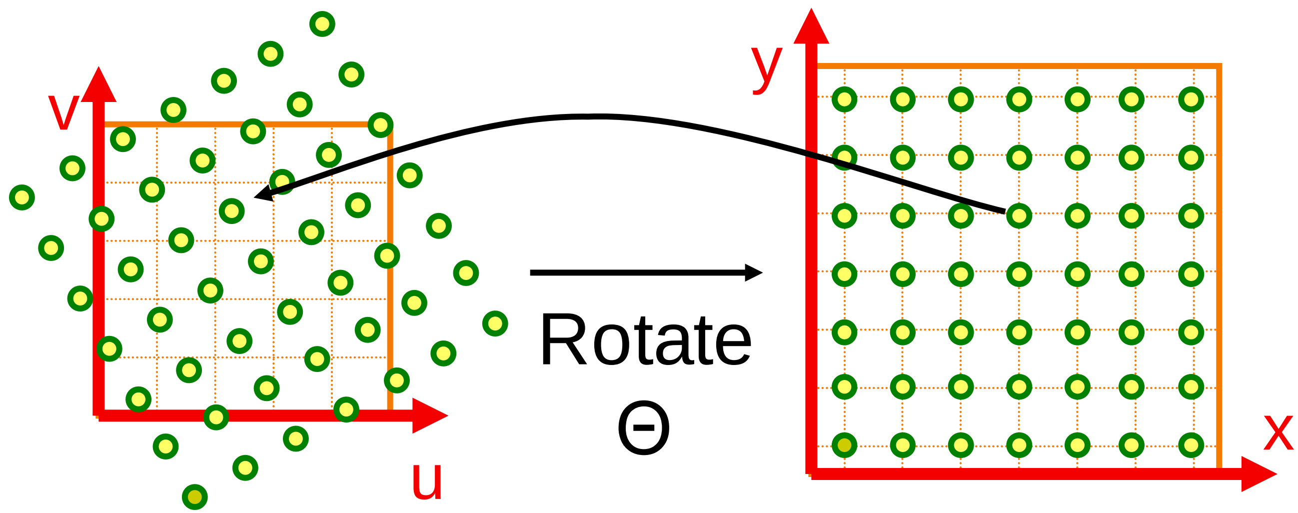


Source image Destination image

Putting it All Together

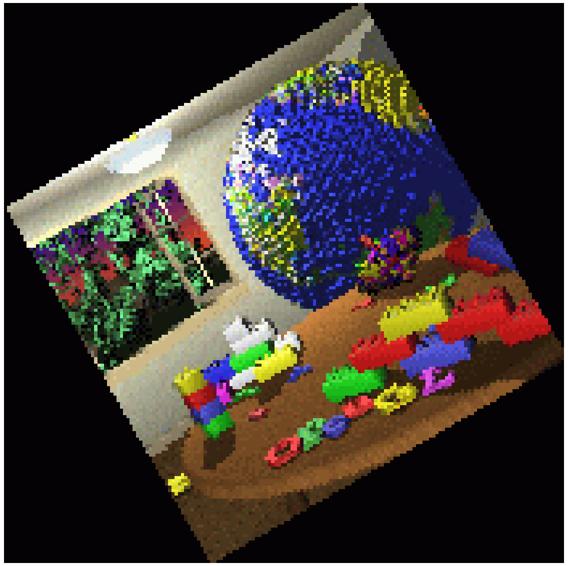
- Possible implementation of image rotation:

```
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);
      }
   }
}
```

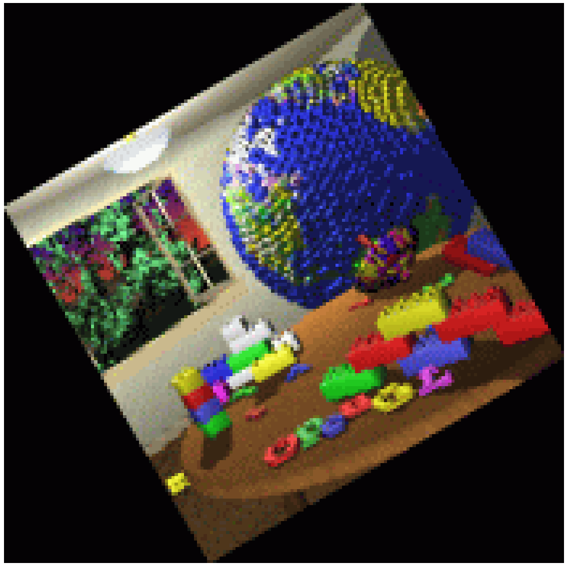


Sampling Method Comparison

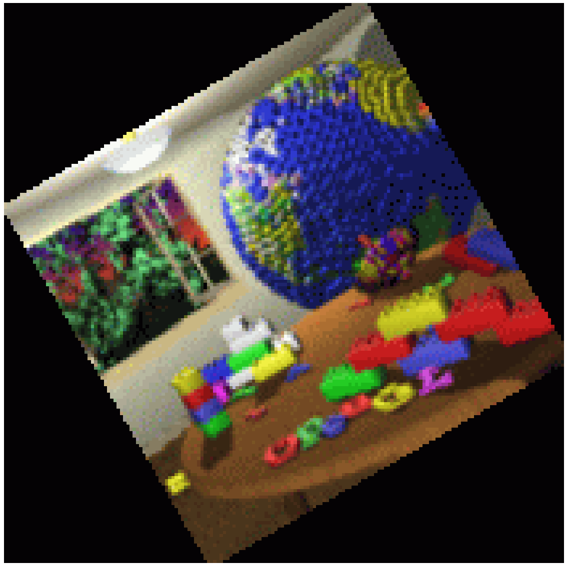
- Trade-offs
  - Aliasing versus blurring
  - Computation speed



Point



Triangle

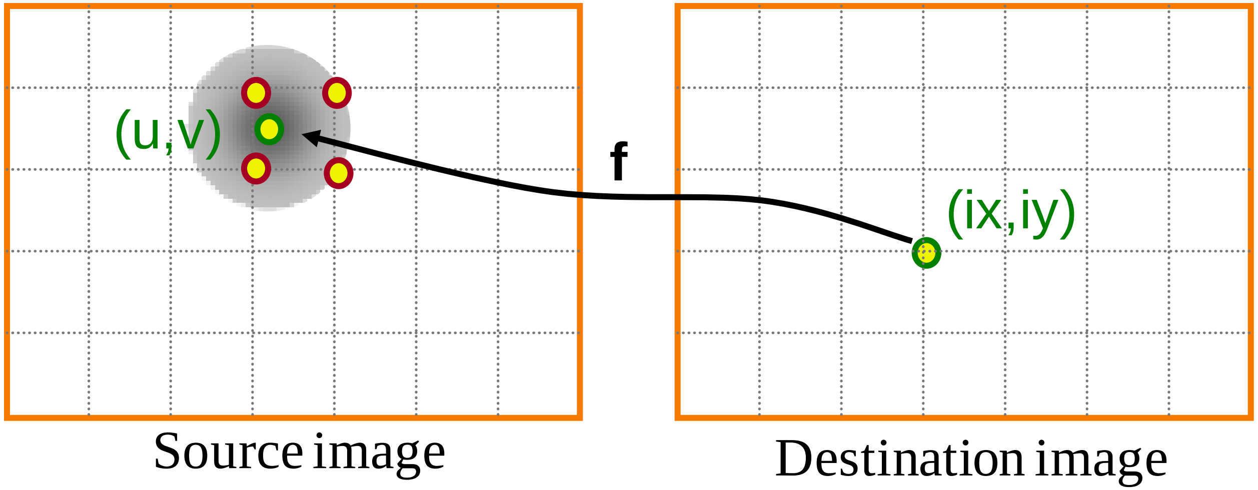


Gaussian

Forward vs. Reverse Mapping

- Reverse mapping:

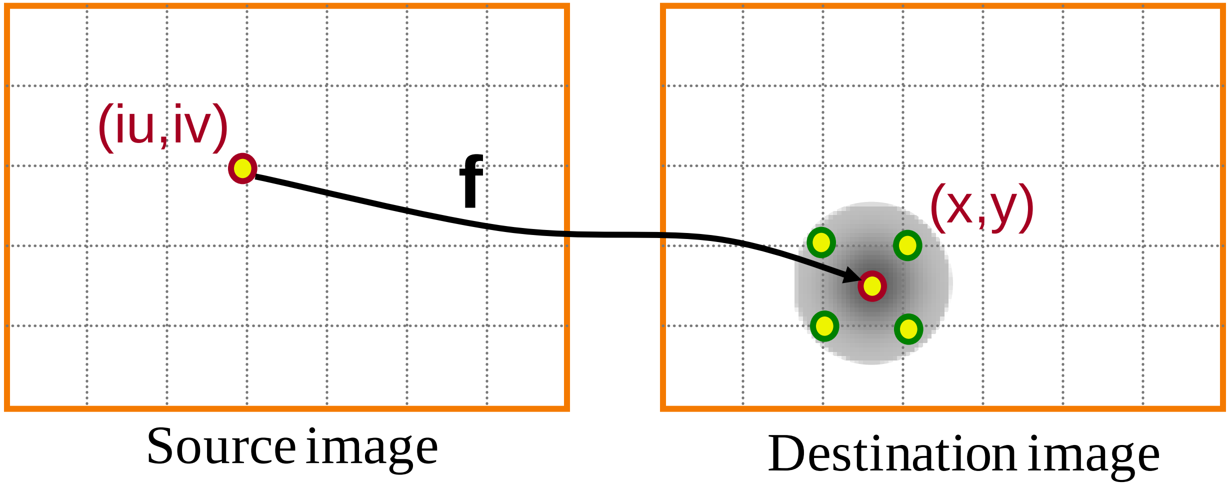
```
Warp(src, dst) {
  for (int ix = 0; ix < xmax; ix++) {
    for (int iy = 0; iy < ymax; iy++) {
      float w ≈ 1 / scale(ix, iy);
      float u = fx-1(ix,iy);
      float v = fy-1(ix,iy);
      dst(ix,iy) = Resample(src,u,v,w);
    }
  }
}
```



Forward vs. Reverse Mapping

- Forward mapping:

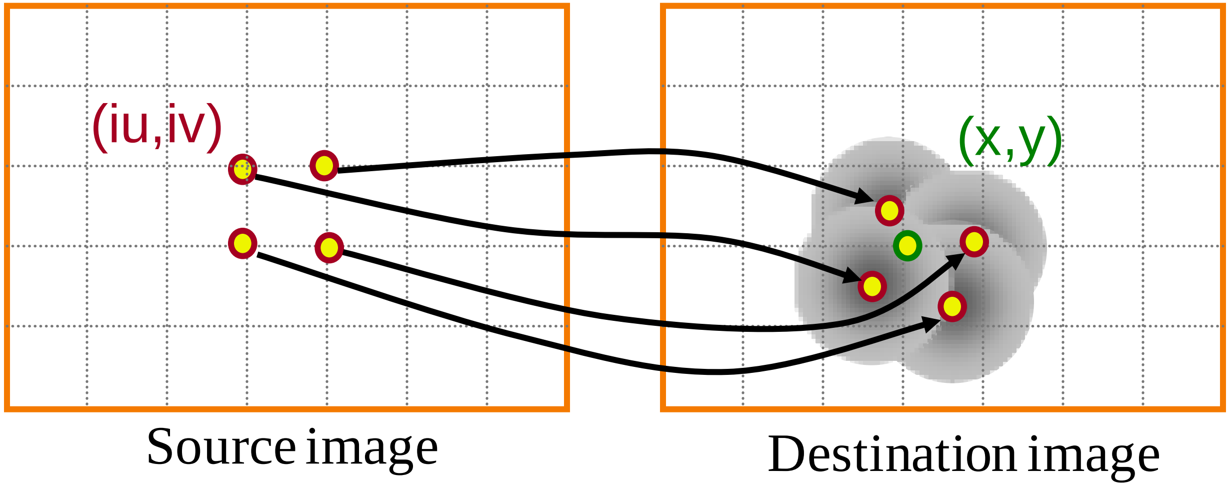
```
Warp(src, dst) {
  for (int iu = 0; iu < umax; iu++) {
    for (int iv = 0; iv < vmax; iv++) {
      float x = fx(iu,iv);
      float y = fy(iu,iv);
      float w ≈ 1 / scale(x, y);
      Splat(src(iu,iv),x,y,k,w);
    }
  }
}
```



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);
    }
  }
}
```

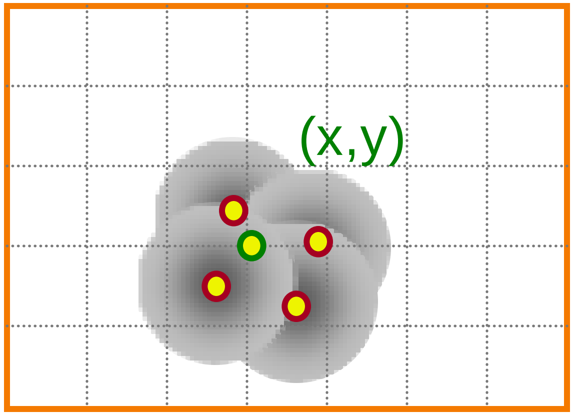


Forward vs. Reverse Mapping

- 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?

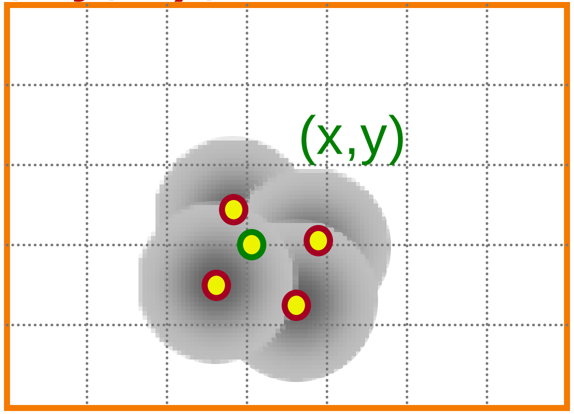


Destination image

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 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
  - Parametric vs. correspondences
- 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 intensity/color
  - Linear: scale, offset, etc.
  - Nonlinear: gamma, saturation, etc.
  - Add random noise
- Filtering over neighborhoods
  - Blur
  - Detect edges
  - Sharpen
  - Emboss
  - Median
- Moving image locations
  - Scale
  - Rotate
  - Warp
- Combining images
  - Composite
  - Morph
- Quantization
- Spatial / intensity tradeoff
  - Dithering