Image Filtering, Warping and Sampling

Connelly Barnes

CS 4810

University of Virginia

Acknowledgement: slides by Jason Lawrence, Misha Kazhdan, Allison Klein, Tom Funkhouser, Adam Finkelstein and David Dobkin

Outline

- **Image Processing**
- Image Warping
- Image Sampling

Image Processing

- What about the case when the modification that we would like to make to a pixel depends on the pixels around it?
  - Blurring
  - Edge Detection
  - Etc.

Multi-Pixel Operations

- In the simplest case, we define a mask of weights which tells us how the values at adjacent pixels should be combined to generate the new value.

Blurring

- To blur across pixels, define a mask:
  - Whose value is largest at the center pixel
  - Whose entries sum to one


Original


Blur

$$\text{Filter} = \begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Blurring

Pixel(x,y): red = 36
green = 36
blue = 0

$$\text{Filter} = \begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Blurring

Pixel(x,y): red = 36
green = 36
blue = 0

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Blurring

New value for Pixel(x,y).red =
(36 * 1/16) + (109 * 2/16) + (146 * 1/16)
(32 * 2/16) + (36 * 4/16) + (109 * 2/16)
(32 * 1/16) + (36 * 2/16) + (73 * 1/16)

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Blurring

New value for Pixel(x,y).red = 62.69

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \begin{bmatrix} 1/16 & 2/16 & 1/16 \\ 2/16 & 4/16 & 2/16 \\ 1/16 & 2/16 & 1/16 \end{bmatrix}$$

Blurring

New value for Pixel(x,y).red = 63

Original

Blur

$$\begin{bmatrix} \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\ \frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\ \frac{1}{16} & \frac{2}{16} & \frac{1}{16} \end{bmatrix}$$

Blurring

- Repeat for each pixel and each color channel
- Note 1: Keep source and destination separate to avoid “drift”
- Note 2: For boundary pixels, not all neighbors are used, and you need to normalize the mask so that the sum of the values is correct

Blurring

- In general, the mask can have arbitrary size:
  - We can express a smaller mask as a bigger one by padding with zeros.

Original

Blur

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} / 16$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & 4 & 2 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} / 16$$

Blurring

- More non-zero entries to give rise to a wider blur

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & 4 & 2 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} /16 \qquad \begin{bmatrix} 0 & 1 & 2 & 1 & 0 \\ 1 & 2 & 4 & 2 & 1 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \\ 0 & 1 & 2 & 1 & 0 \end{bmatrix} /48$$

Original

Narrow Blur

Wide Blur

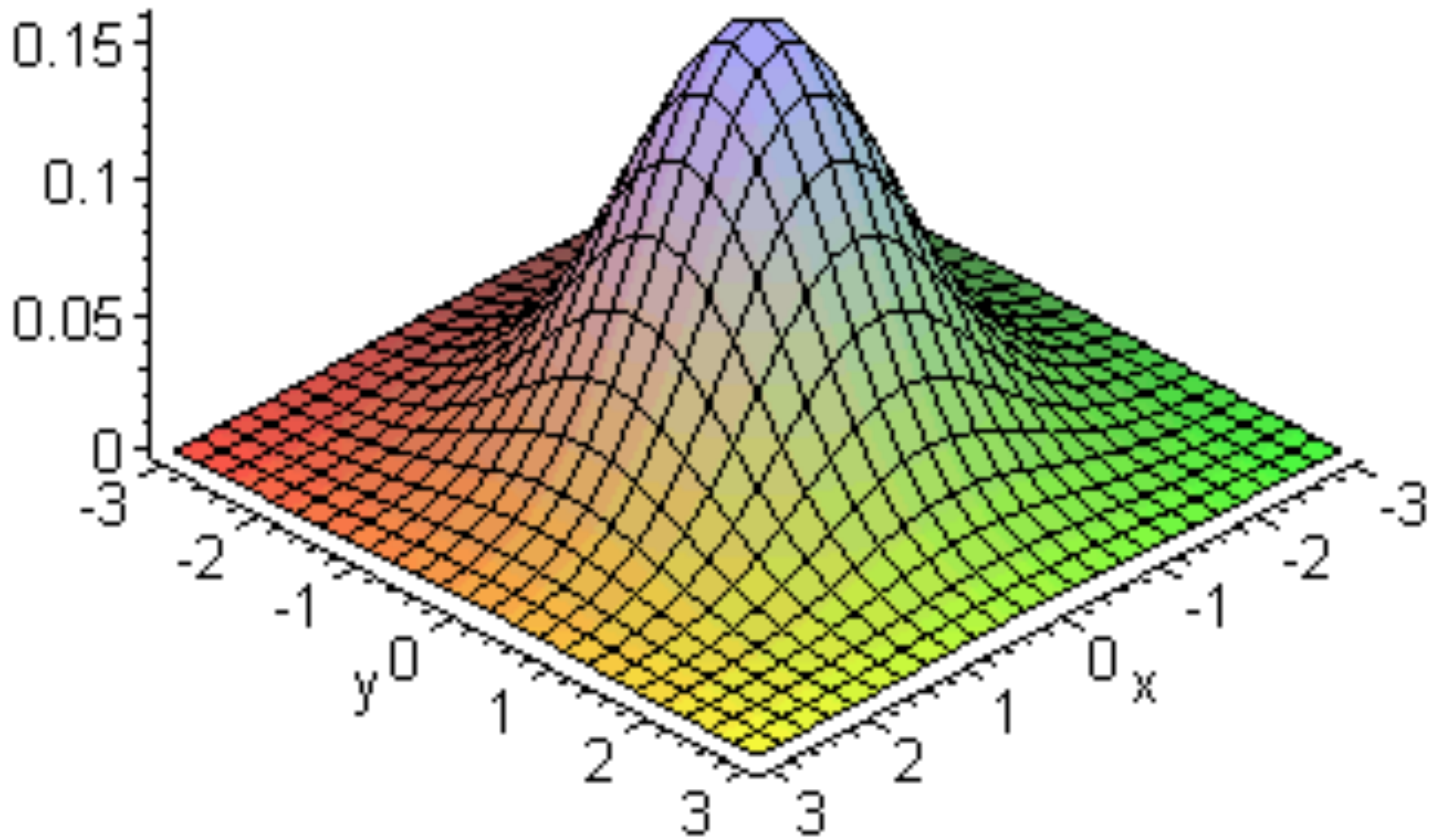
Blurring

- A general way for defining the entries of an nxn blurring mask is to use the values of a Gaussian:

$$\text{Gaussian}[i, j] = \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}$$

  - σ equals the mask radius ("n/2 for an n x n mask")
  - x is i's horizontal distance from center pixel
  - y is j's vertical distance from center pixel
  - Don't forget to normalize!

Bivariate Gaussian Function



$$\mathrm{Gaussian}[i,j] = \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}$$

aka “Normal Distribution”

Edge Detection

- To find the edges in an image, define a mask:
  - Whose value is largest at the center pixel
  - Whose entries sum to zero.
- Edge pixels are those whose value is larger (or smaller) than those of its neighbors.

Original

Highlighted Edges

$$\text{Filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

Pixel(x,y): red = 36
green = 36
blue = 0

$$\text{Filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

Pixel(x,y): red = 36
green = 36
blue = 0

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red =
(36 * -1) + (109 * -1) + (146 * -1)
(32 * -1) + (36 * 8) + (109 * -1)
(32 * -1) + (36 * -1) + (73 * -1)

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red = -285

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	$\boxed{36}$	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

$$\text{Filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red = 0

	X - 1	X	X + 1
Y - 1	36	109	146
Y	32	36	109
Y + 1	32	36	73

Pixel(x,y).red and its red neighbors

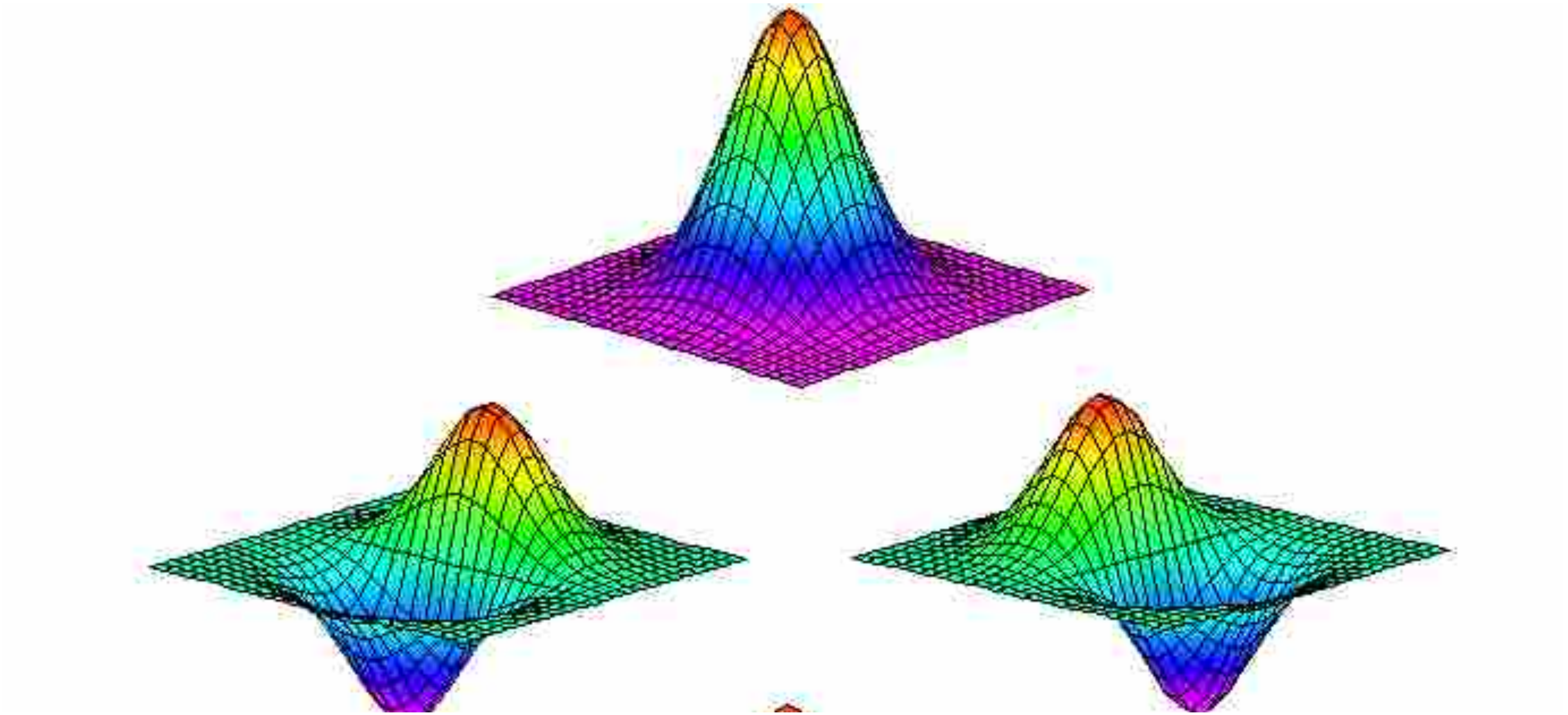
$$\text{Filter} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection

New value for Pixel(x,y).red = 0

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Mask is a Derivative Filter



$$\frac{\partial G}{\partial x} \propto -\frac{x}{\sigma^2} \mathrm{e}^{-\frac{x^2+y^2}{2\sigma^2}}$$

Outline

- Image Processing
- **Image Warping**
- Image Sampling

Image Warping

- Move pixels of image
  - Mapping
  - Resampling

Warp

Source image

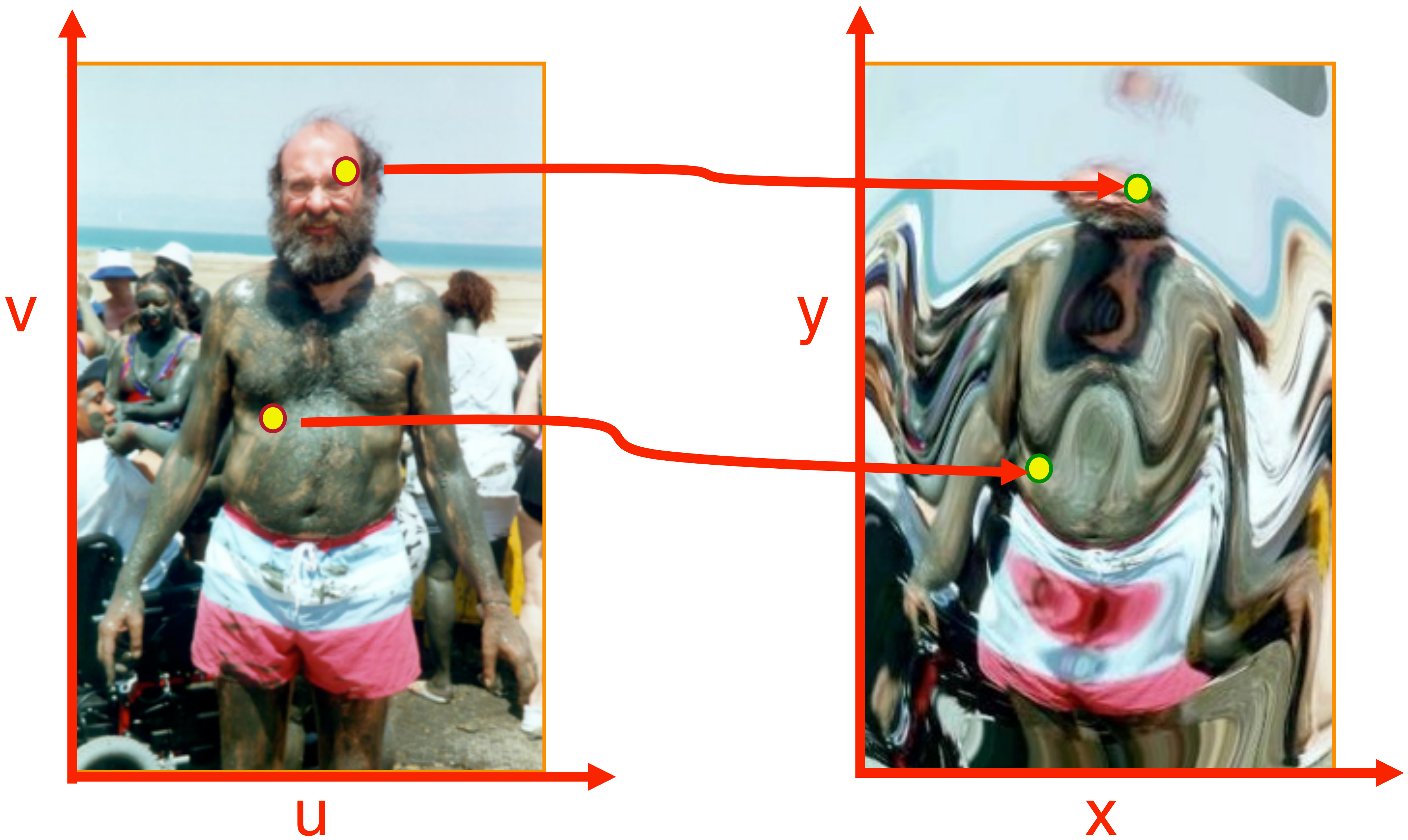
Destination image

Overview

- **Mapping**
  - **Forward**
  - **Reverse**
- Resampling
  - Point sampling
  - Triangle filter
  - Gaussian filter

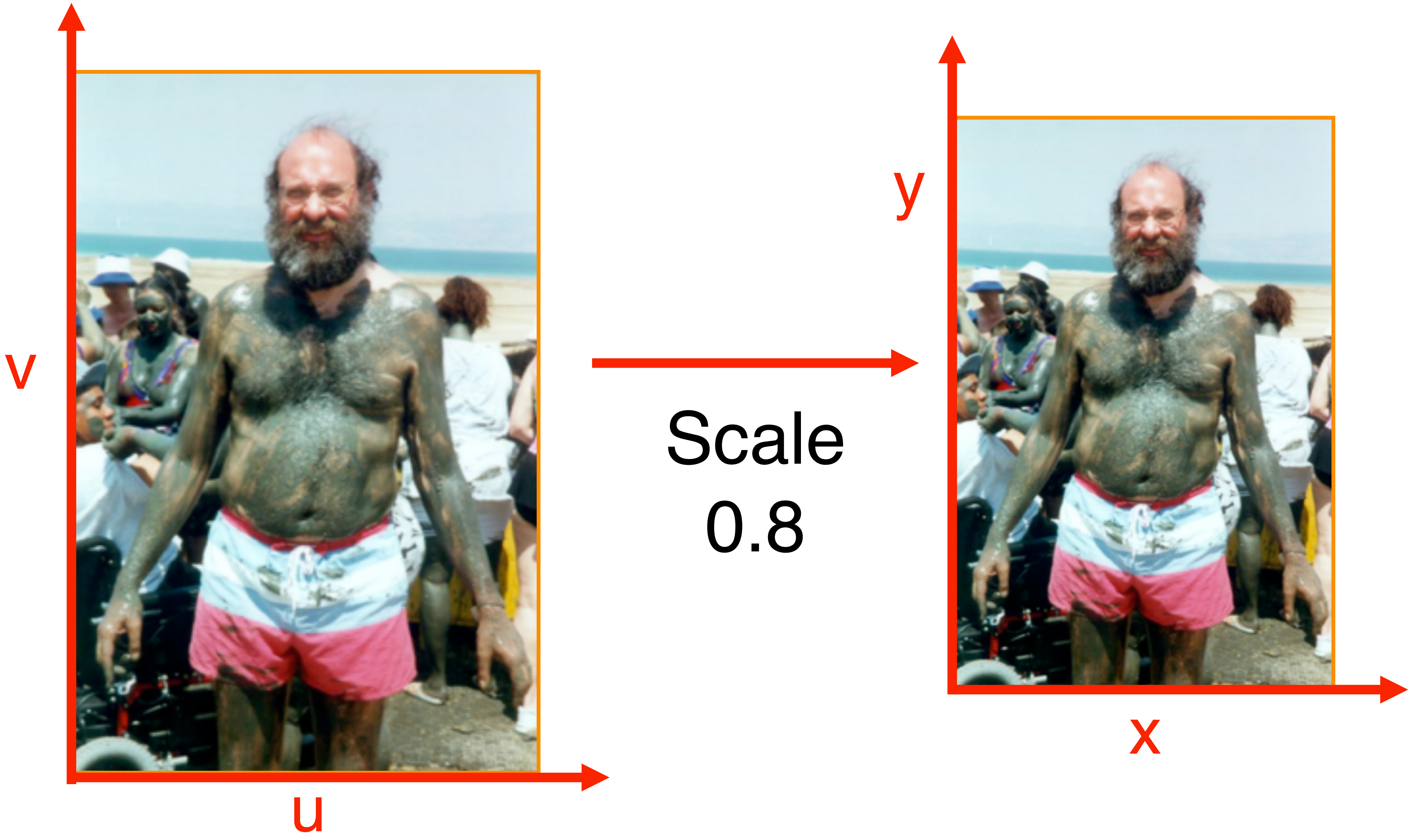
Mapping

- Transformation: describe the destination location (x,y) for every source location (u,v)



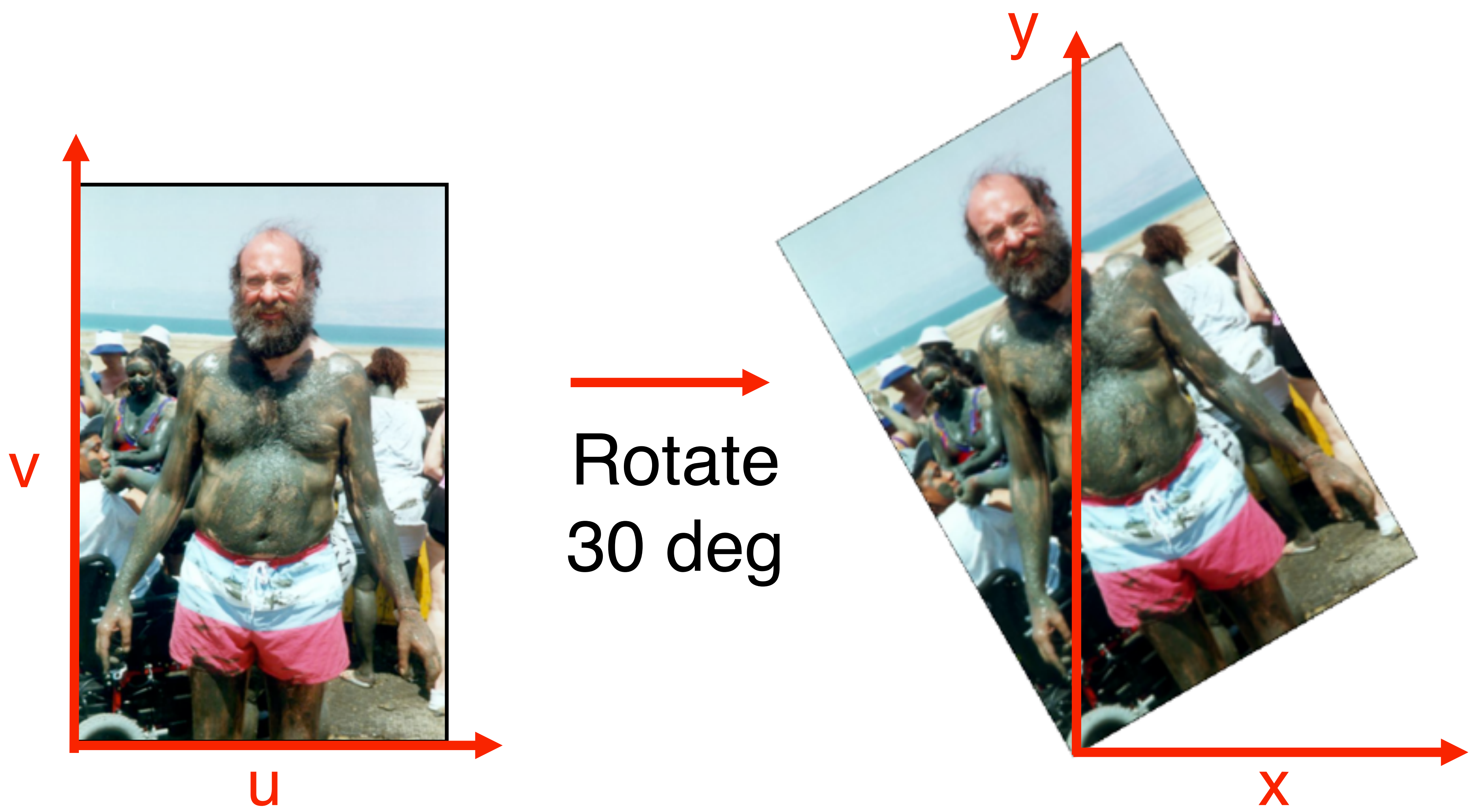
Example Mappings

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



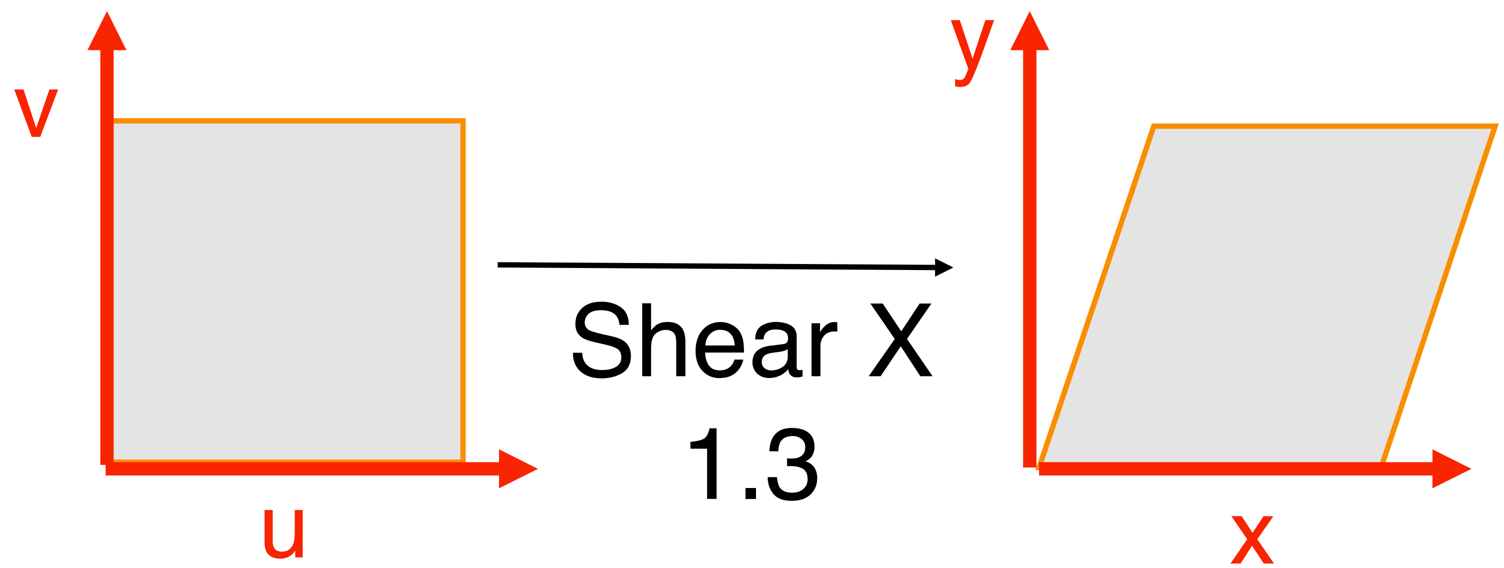
Example Mappings

- Rotate by θ degrees:
  - $x = u\cos\theta - v\sin\theta$
  - $y = u\sin\theta + v\cos\theta$

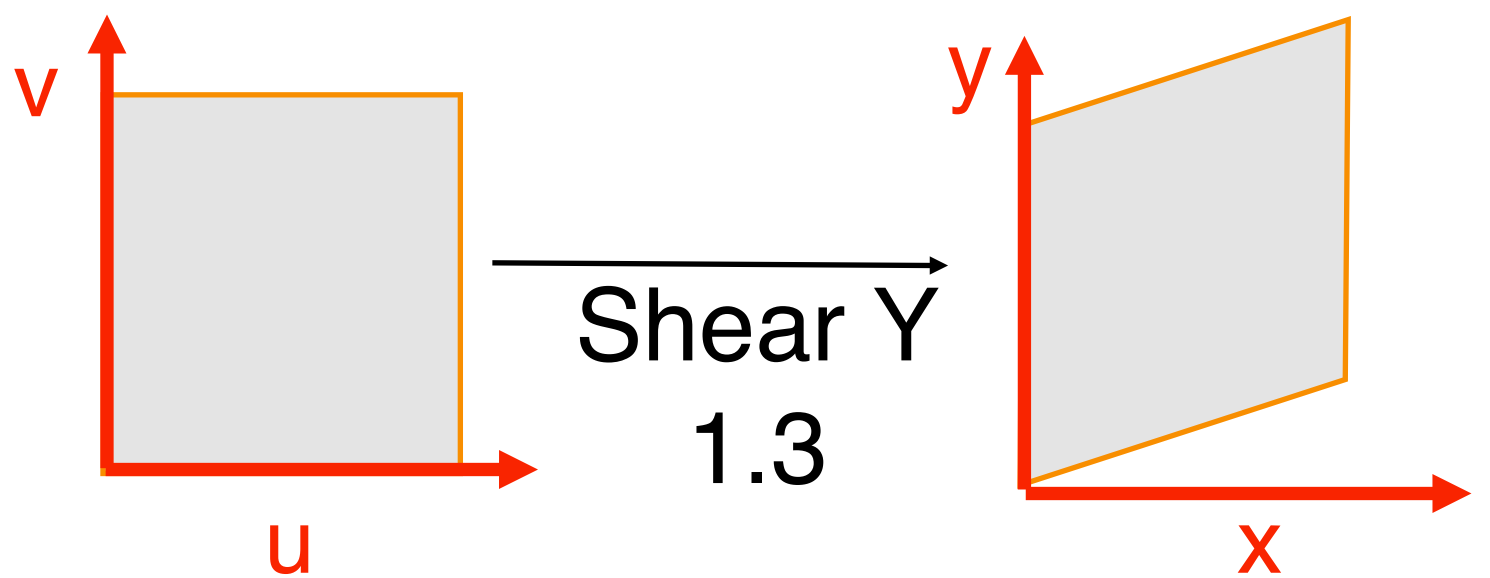


Example Mappings

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

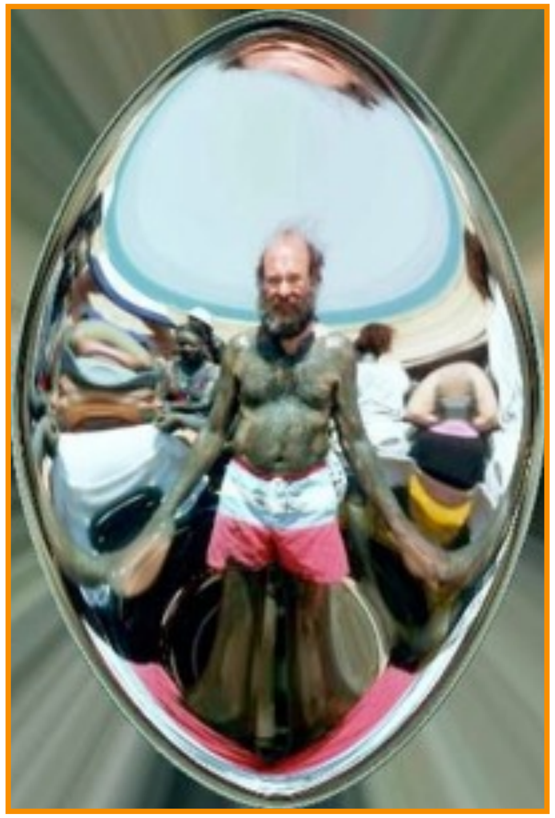


- Shear in Y by factor:
  - x = u
  - y = v + factor * u



Other Mappings

- Any function of u and v:
  - $x = fx(u,v)$
  - $y = fy(u,v)$



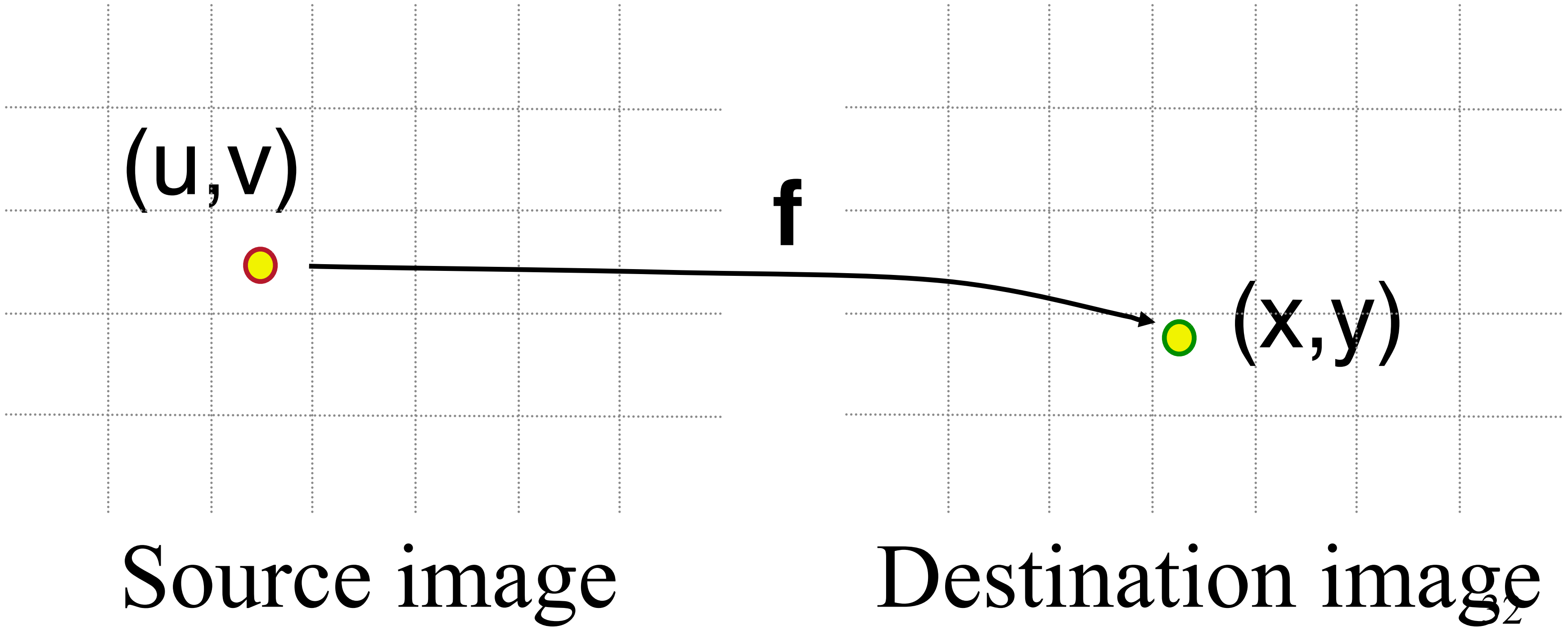
Fish-eye

"Swirl"

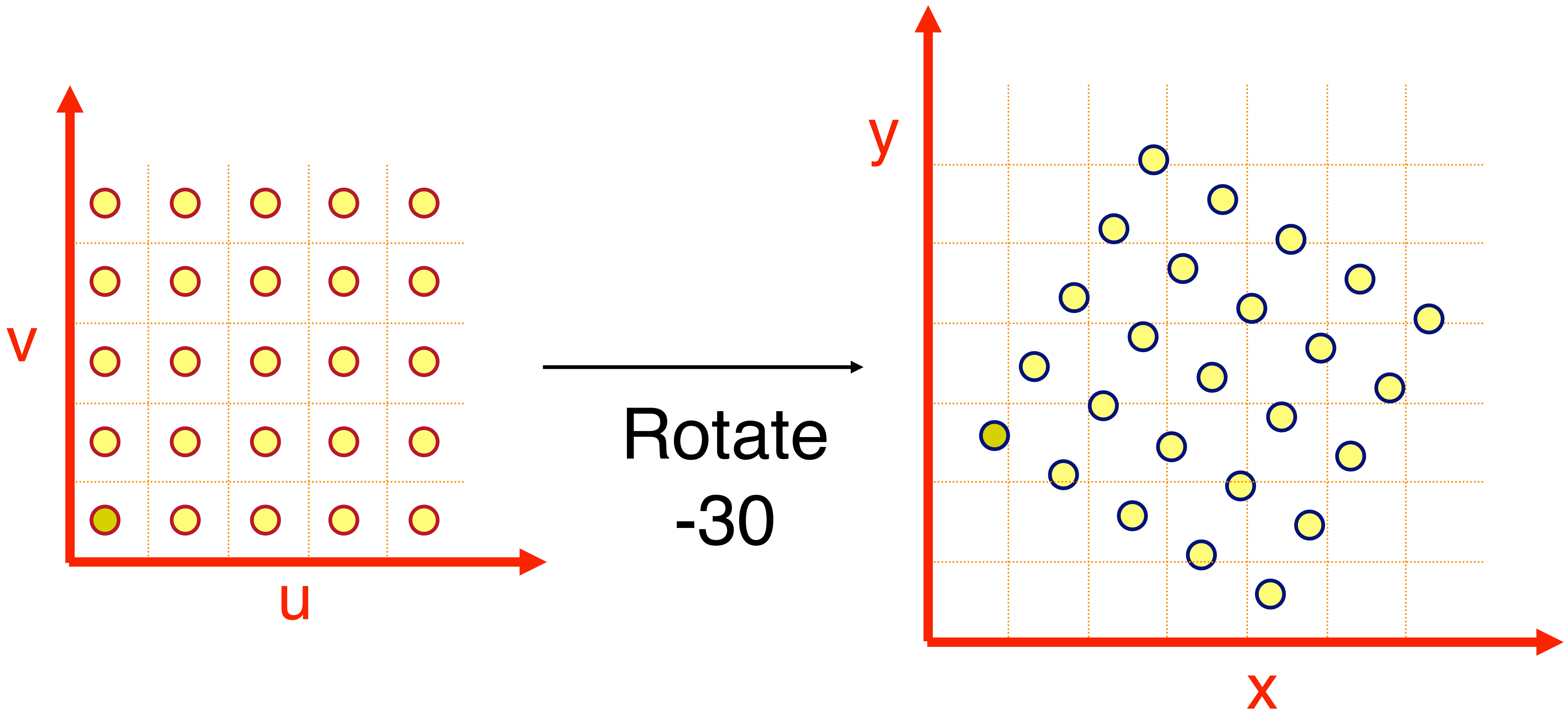
"Rain"

Image Warping Attempt 1 (Forward Mapping)

```
for (int u = 0; u < umax; u++)
  for (int v = 0; v < vmax; v++)
    float x = fx(u,v);
    float y = fy(u,v);
    dst(x,y) = src(u,v);
```

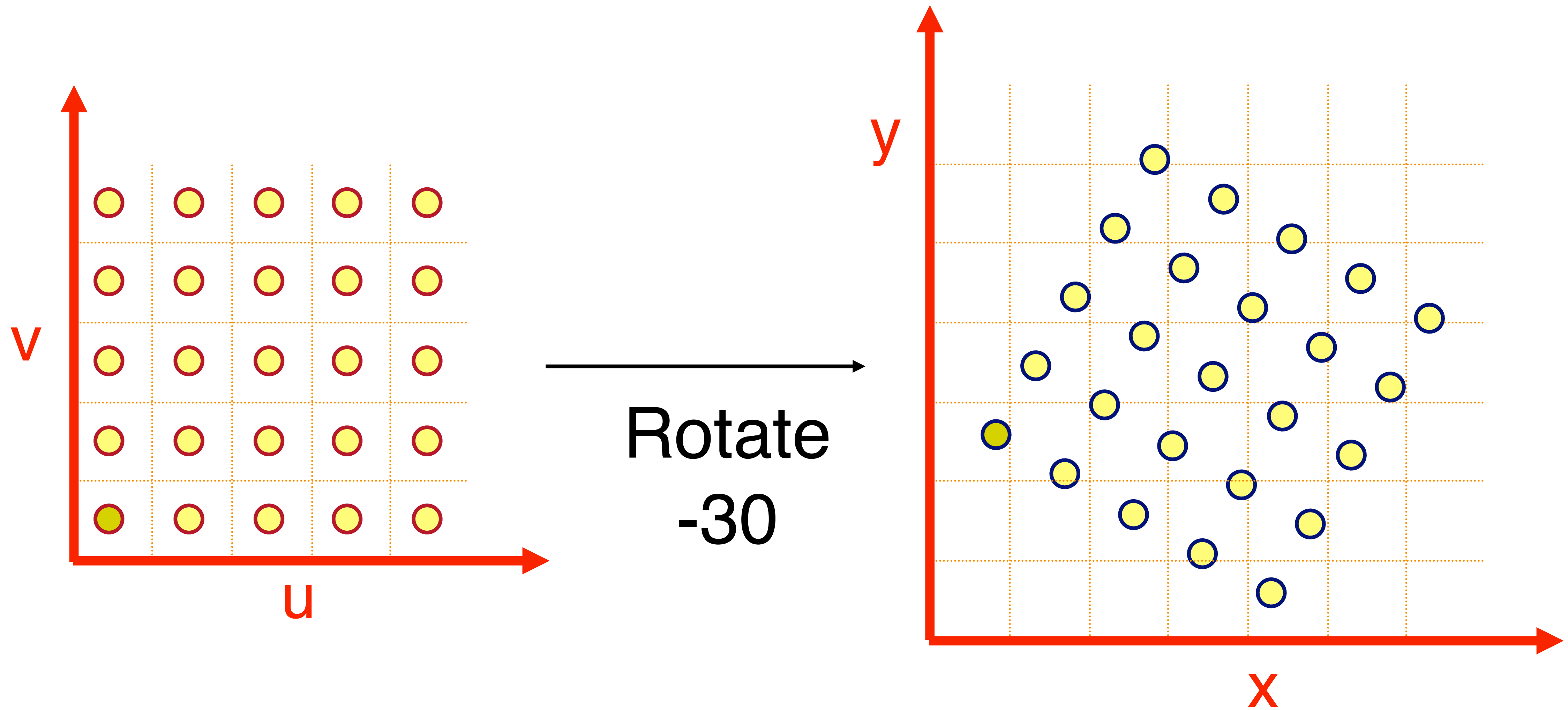


Forward Mapping



Forward Mapping

Many source pixels can map to same destination pixel



Forward Mapping

Some destination pixels may not be covered

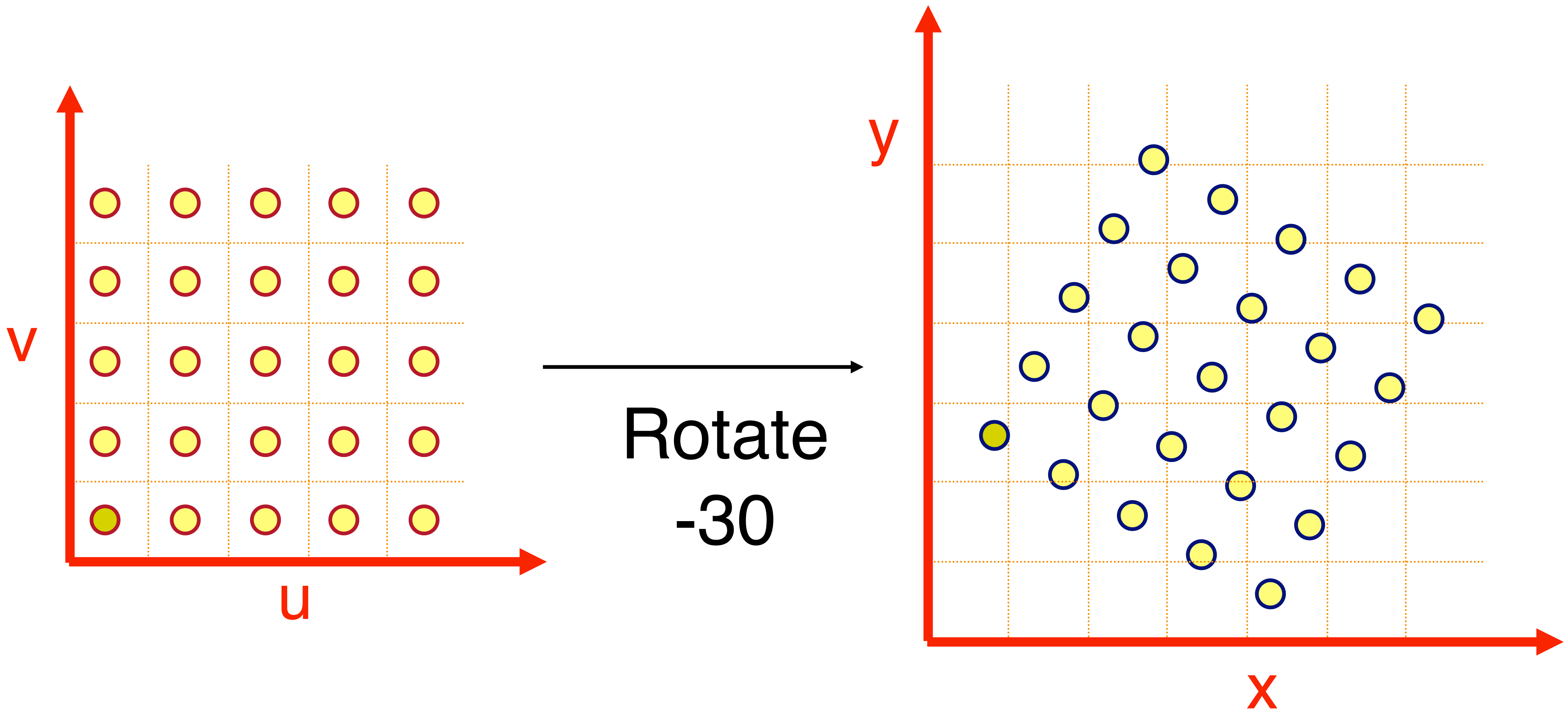
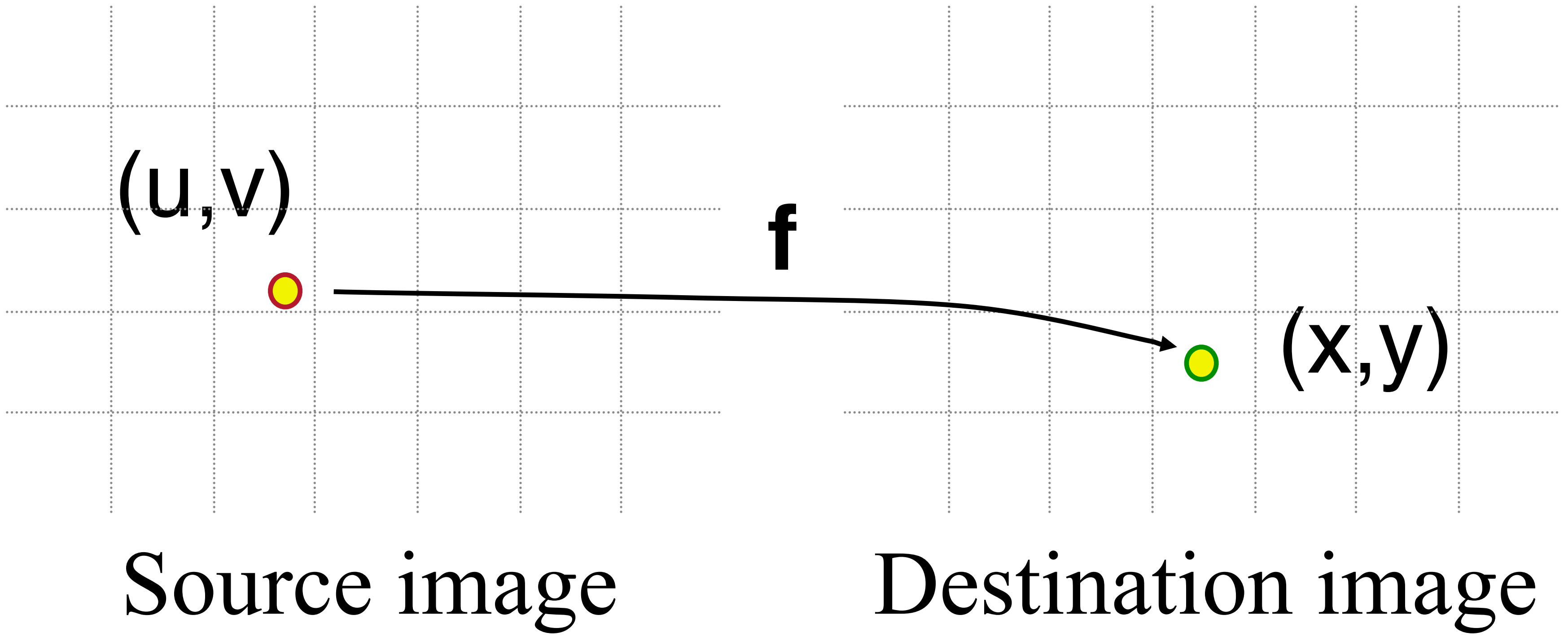


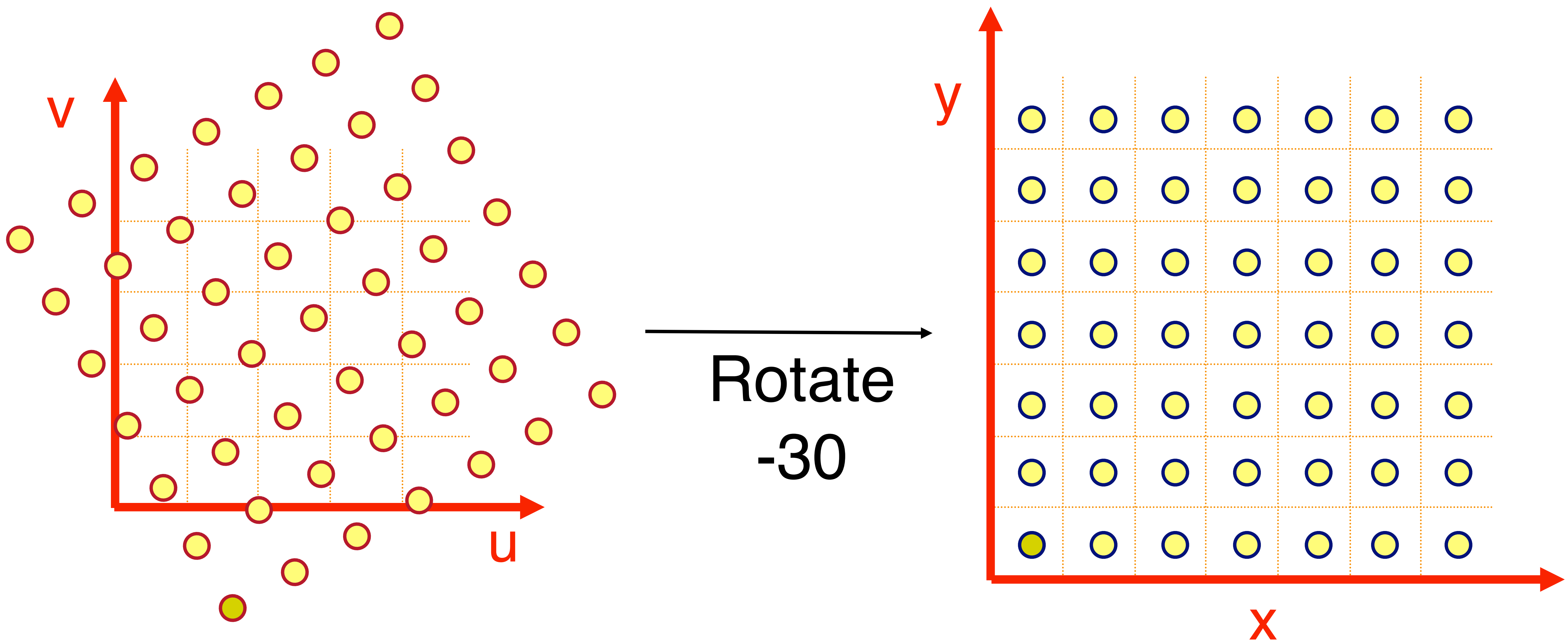
Image Warping Attempt 2 (Reverse Mapping)

```
for (int x = 0; x < xmax; x++)
  for (int y = 0; y < ymax; y++)
    float u = fx^-1(x,y);
    float v = fy^-1(x,y);
    dst(x,y) = src(u,v);
```



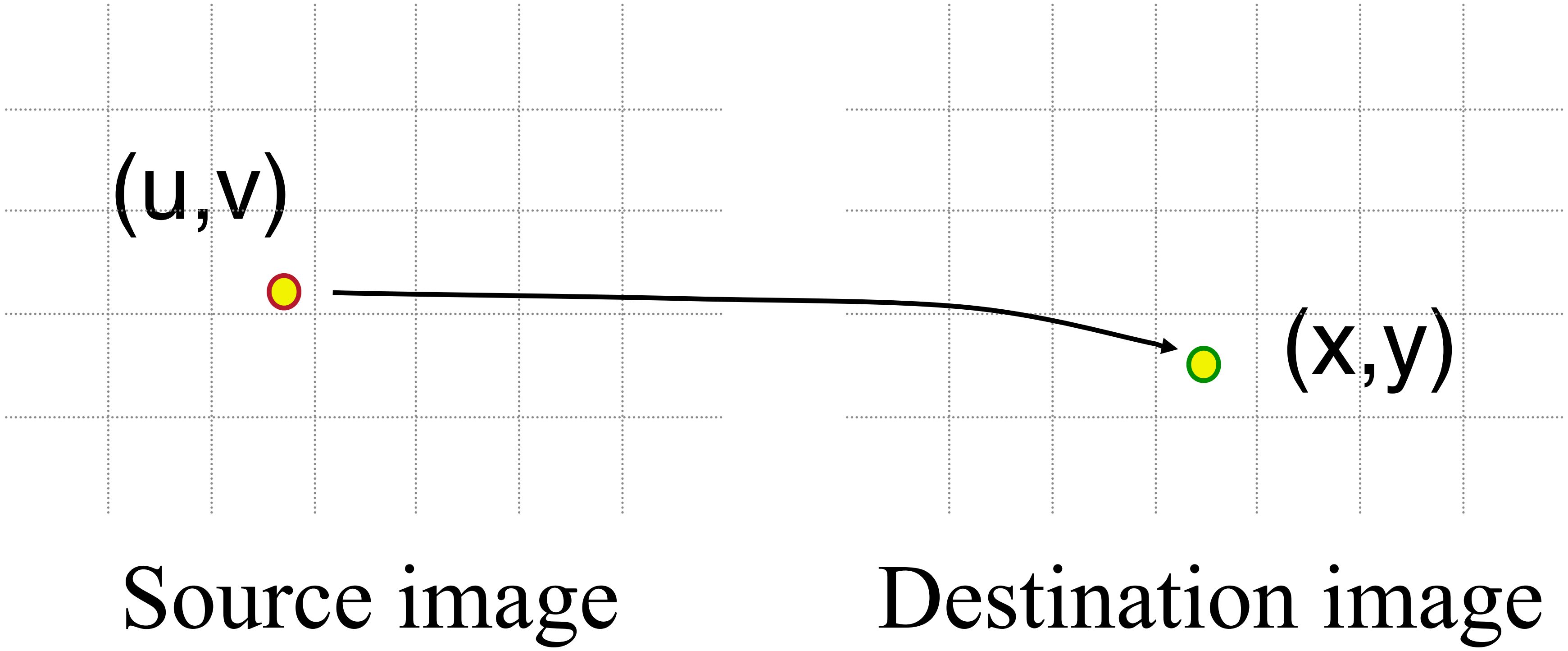
Reverse Mapping – GOOD!

- Iterate over destination image
  - Must resample source
  - May oversample, but much simpler!



Resampling

Issue: (u,v) does not usually have integer coordinates

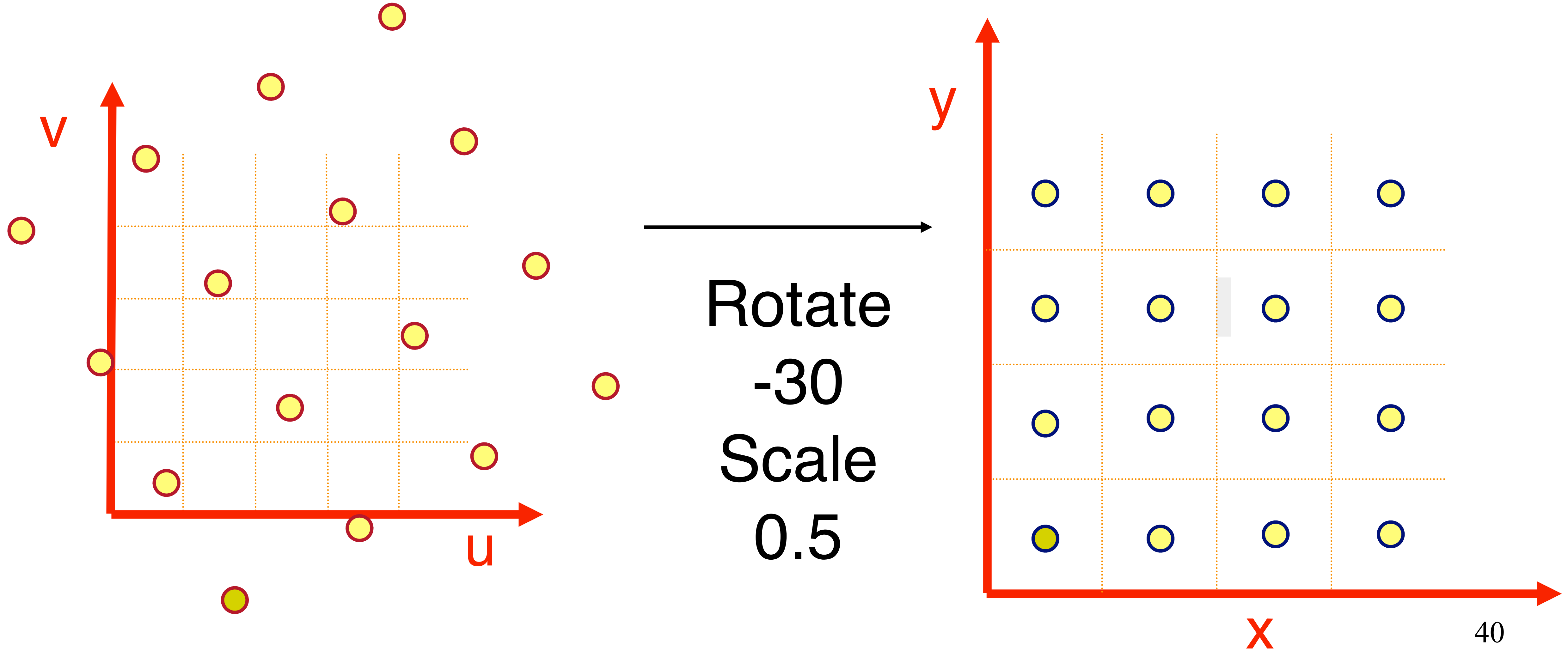


Overview

- Mapping
  - Forward
  - Reverse
- **Resampling**
  - **Nearest Point Sampling**
  - **Bilinear Sampling**
  - **Gaussian Sampling**

Nearest Point Sampling

```
int iu = floor(u+0.5);
int iv = floor(v+0.5);
dst[x,y] = src[iu,iv];
```



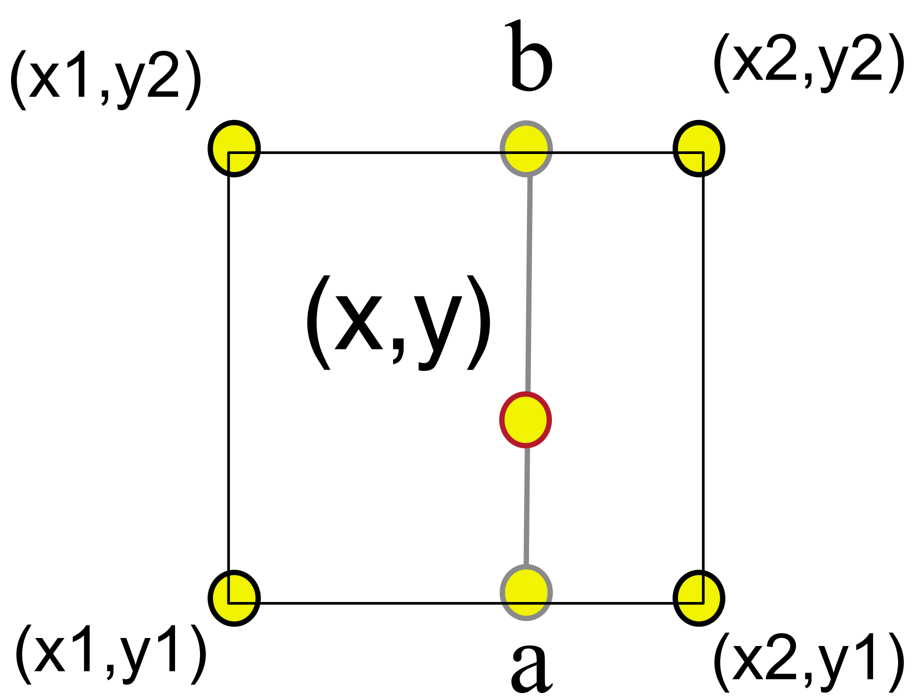
Bilinear Sampling

- Bilinearly interpolate four closest pixels

  a = linear interpolation of src(x1,y1) and src(x2,y1)

  b = linear interpolation of src(x1,y2) and src(x2,y2)

  dst(x,y) = linear interpolation of “a” and “b”




```
x1 = floor( x );
x2 = x1 + 1;
y1 = floor( y );
y2 = y1 + 1;
dx = x - x1;
dy = y - y1;
a = src(x1,y1)*(1-dx) + src(x2,y1)*dx;
b = src(x1,y2)*(1-dx) + src(x2,y2)*dx;
dst(x,y) = a*(1-dy) + b*dy;
```

Bilinear Sampling

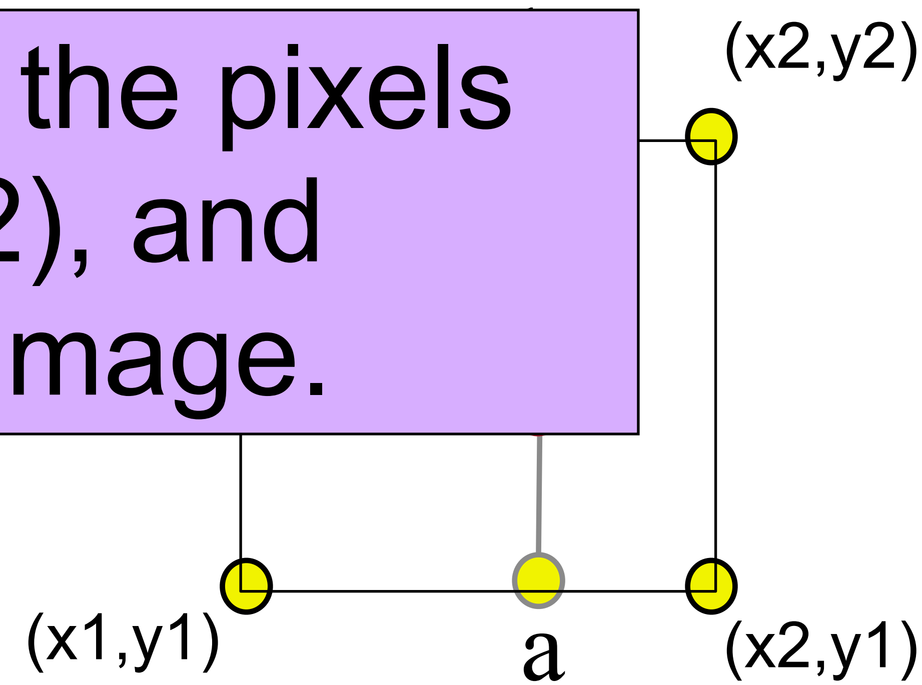
- Bilinearly interpolate four closest pixels

  a = linear interpolation of src(x1,y1) and src(x2,y1)

  b = linear interpolation of src(x1,y2) and src(x2,y2)

  dst(x,y) = linear interpolation of “a” and “b”

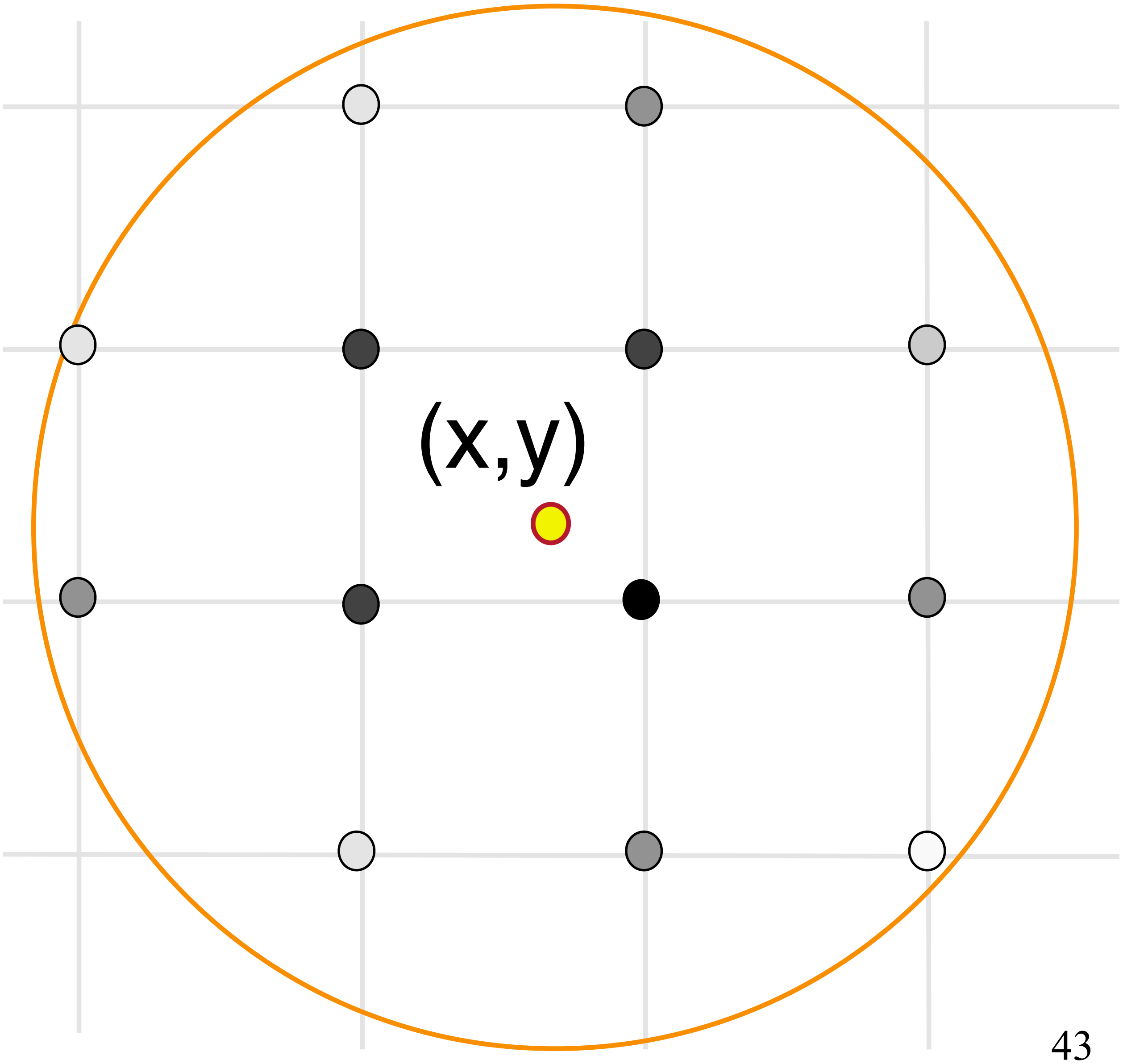
Make sure to test that the pixels (x1,y1), (x2,y2), (x1,y2), and (x2,y1) are within the image.




```
x1 =
x2 =
y1 =
y2 = y1 + 1;
dx = x - x1;
dy = y - y1;
a = src(x1,y1)*(1-dx) + src(x2,y1)*dx;
b = src(x1,y2)*(1-dx) + src(x2,y2)*dx;
dst(x,y) = a*(1-dy) + b*dy;
```

Gaussian Sampling

- Compute weighted sum of pixel neighborhood:
  - The blending weights are the normalized values of a Gaussian function.



Nearest
Neighbor

Bilinear

Gaussian

Trade-offs:

1. Jagged edges versus blurring
2. Computational speed

Gaussian

Image Warping Implementation

```
for (int x = 0; x < xmax; x++)
  for (int y = 0; y < ymax; y++)
    float u = fx-1(x,y);
    float v = fy-1(x,y);
    dst(x,y) = resample_src(u,v,w);
```

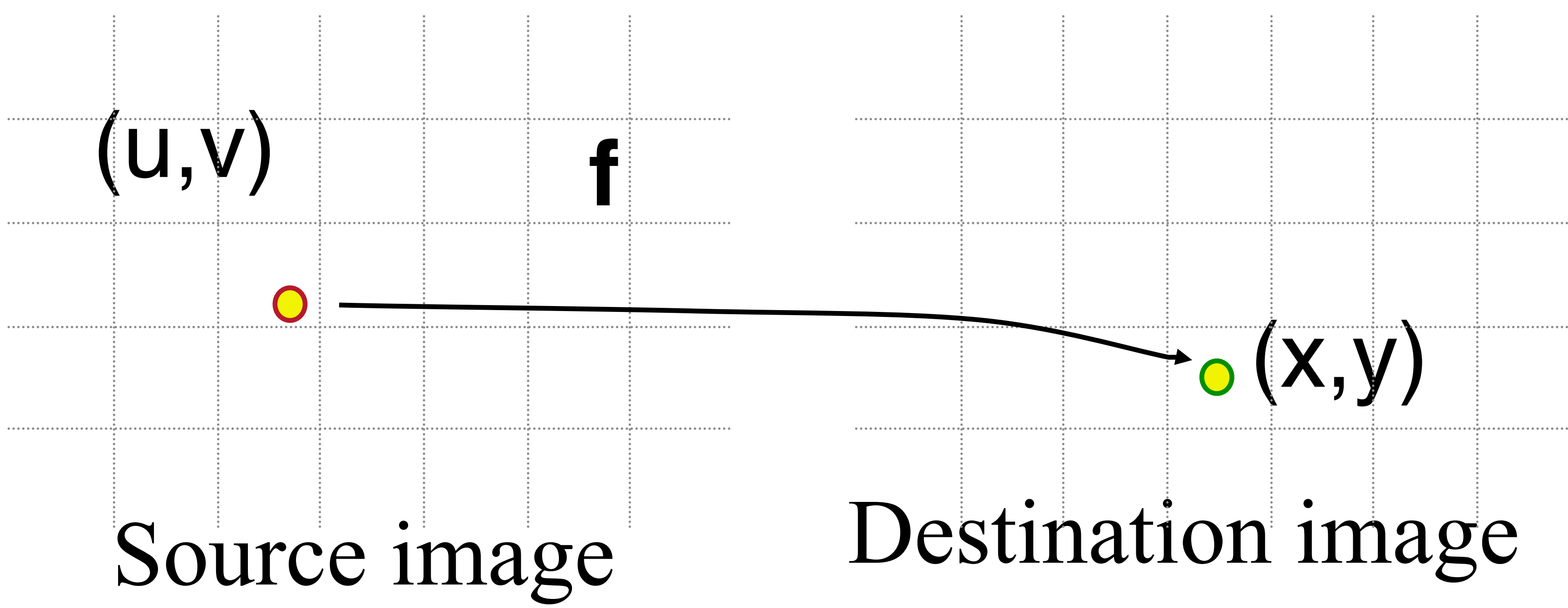
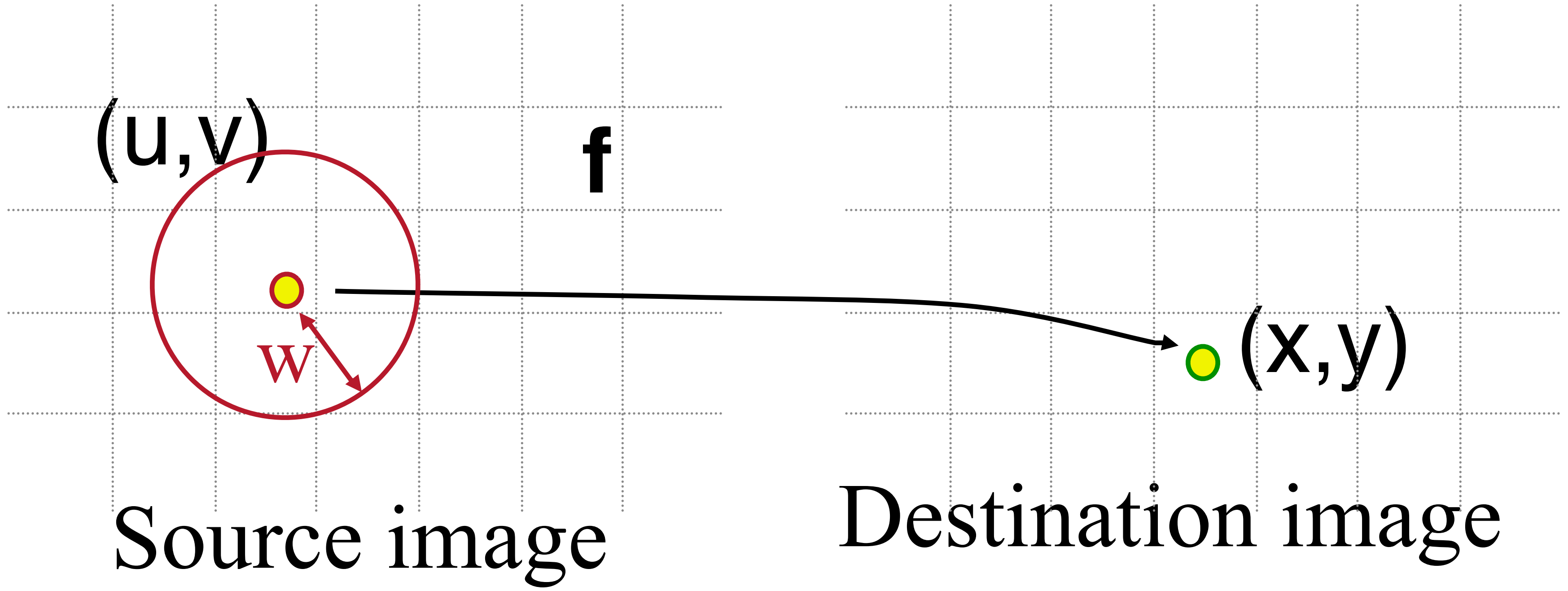


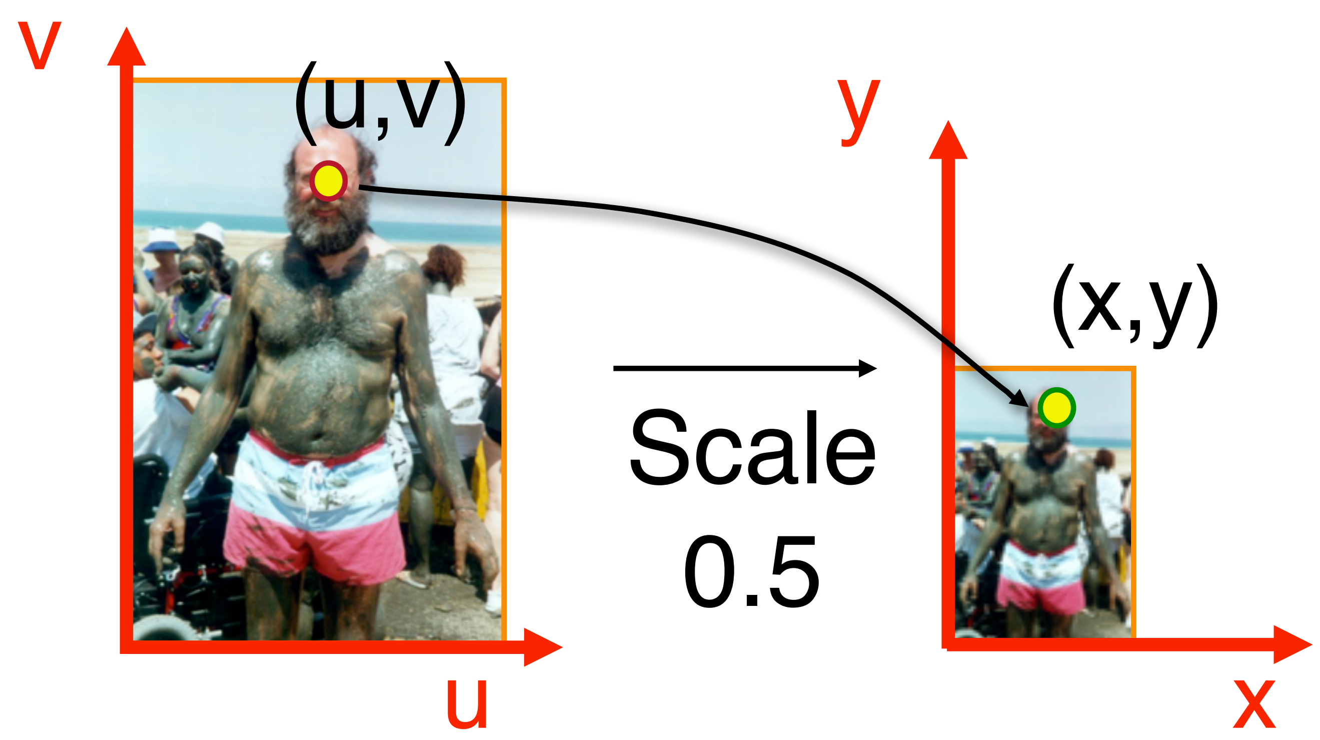
Image Warping Implementation

```
for (int x = 0; x < xmax; x++)
  for (int y = 0; y < ymax; y++)
    float u = fx-1(x,y);
    float v = fy-1(x,y);
    dst(x,y) = resample_src(u,v,w);
```



Example: Scale (src, dst, s)

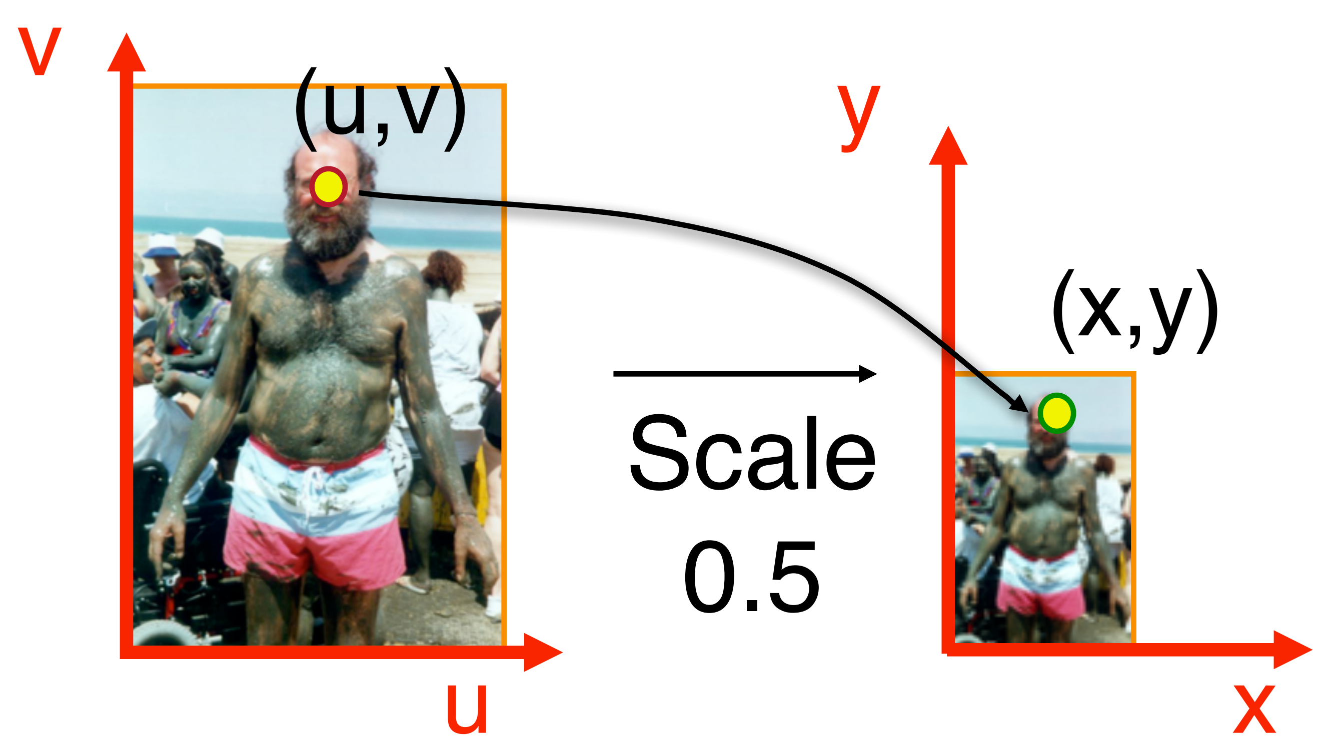
```
float w = ??;
for (int x = 0; x < xmax; x++)
   for (int y = 0; y < ymax; y++)
      float u = x / s;
      float v = y / s;
      dst(x,y) = resample_src(u,v,w);
```



Example: Scale (src, dst, s)

```
float w = ??;
for (int x = 0; x < xmax; x++)
   for (int y = 0; y < ymax; y++)
      float u = x / s;
      float v = y / s;
      dst(x,y) = resample_src(u,v,w);
```

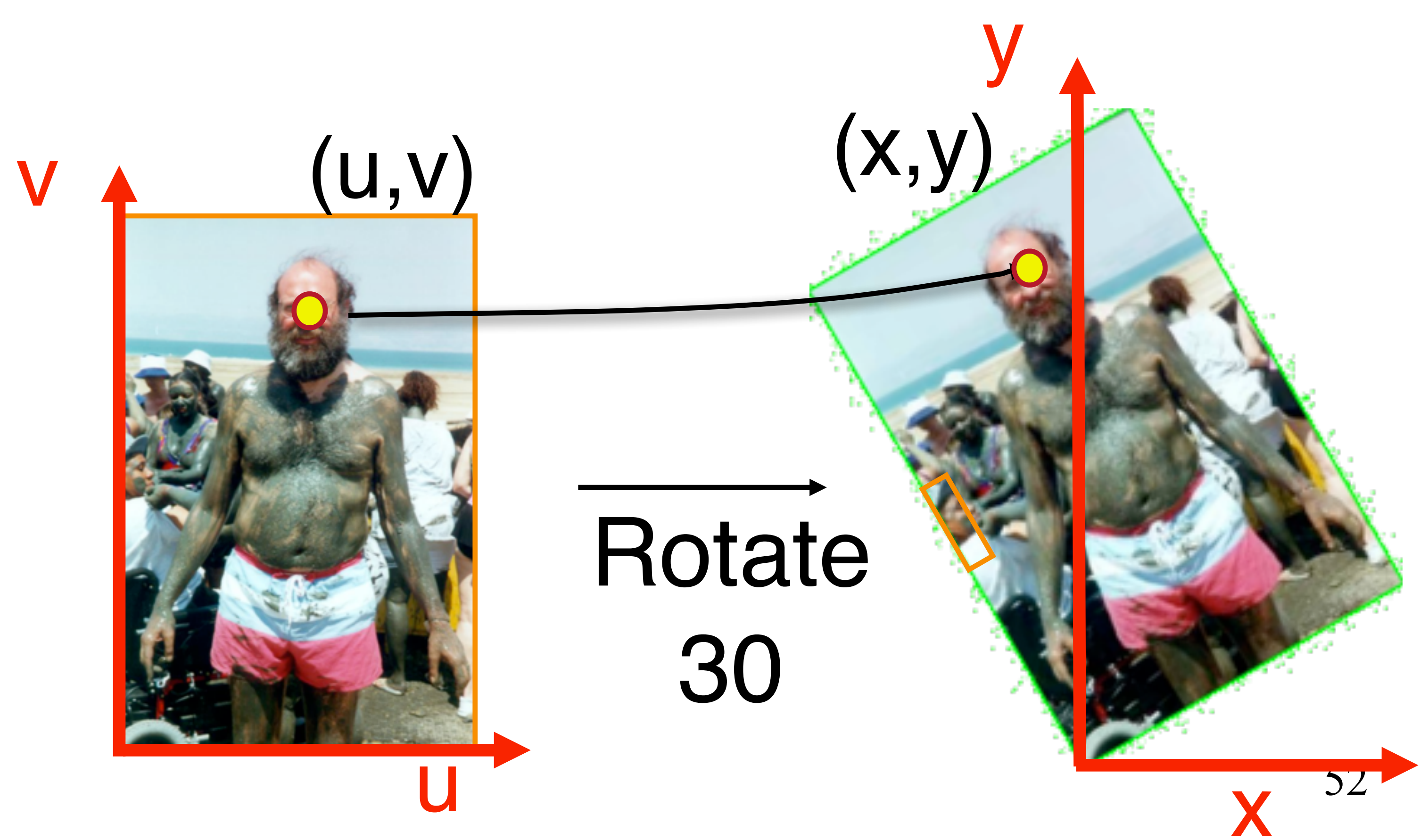
w=1.0/s



Example: Rotate (src, dst, theta)

```
float w = ??;
for (int x = 0; x < xmax; x++)
   for (int y = 0; y < ymax; y++)
      float u = x*cos(-θ) - y*sin(-θ);
      float v = x*sin(-θ) + y*cos(-θ);
      dst(x,y) = resample_src(u,v,w);
```

```
x = ucosθ - vsinθ
y = usinθ + vcosθ
```



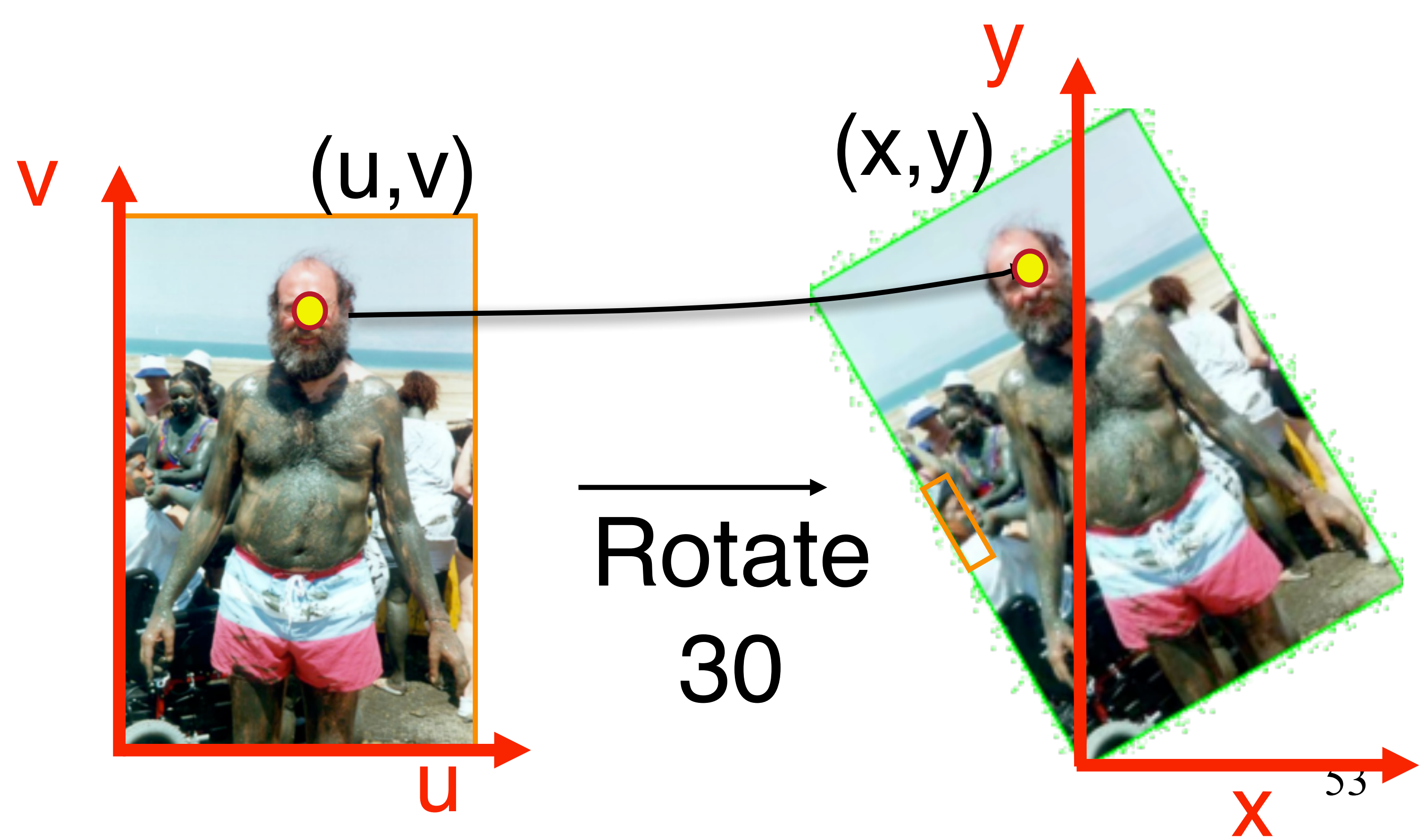
Example: Rotate (src, dst, theta)

```
float w = ??;
for (int x = 0; x < xmax; x++)
  for (int y = 0; y < ymax; y++)
    float u = x*cos(-θ) - y*sin(-θ);
    float v = x*sin(-θ) + y*cos(-θ);
    dst(x,y) = resample_src(u,v,w);
```

w=1.0

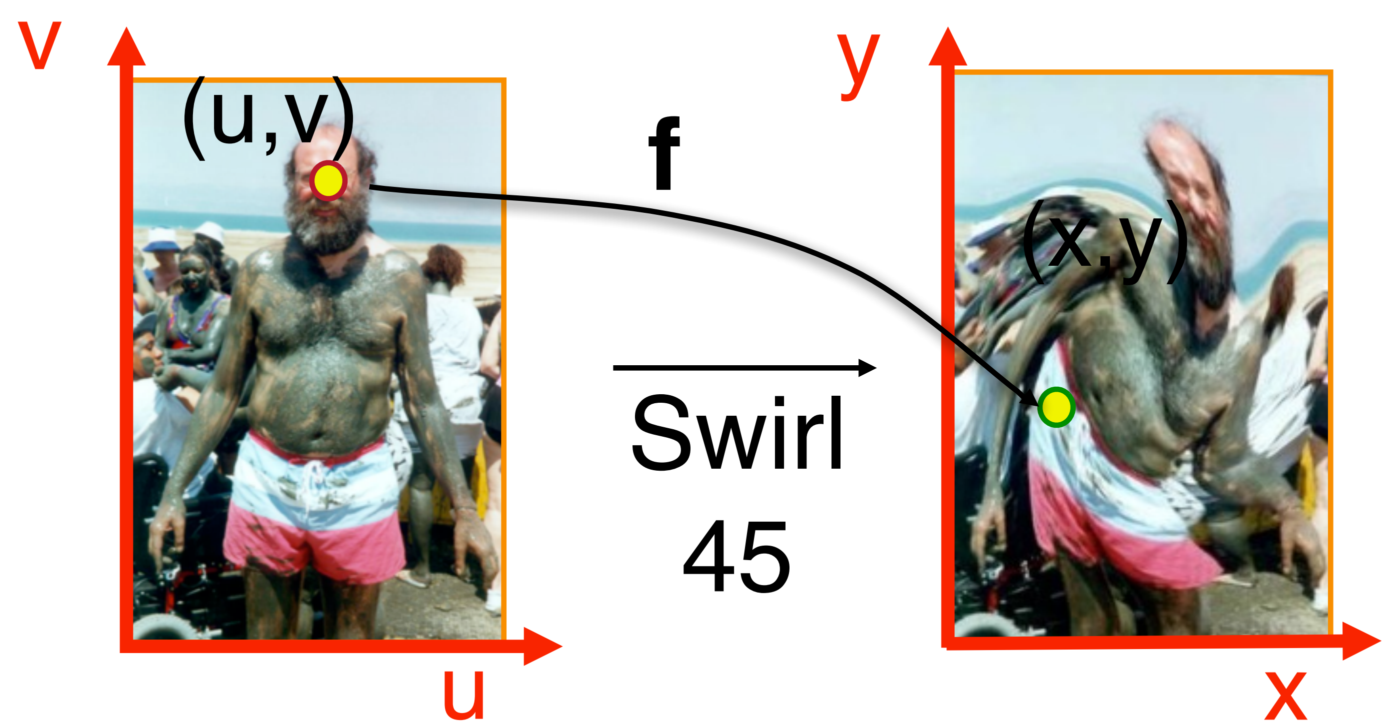
$x = u\cos\theta - v\sin\theta$

$y = u\sin\theta + v\cos\theta$



Example: Swirl (src, dst, theta) ???

```
float w = ??;
for (int x = 0; x < xmax; x++)
  for (int y = 0; y < ymax; y++)
    float u = rot(dist(x,xcenter)*theta);
    float v = rot(dist(y,ycenter)*theta);
    dst(x,y) = resample_src(u,v,w);
```



Outline

- Image Processing
- Image Warping
- **Image Sampling**

Sampling Questions

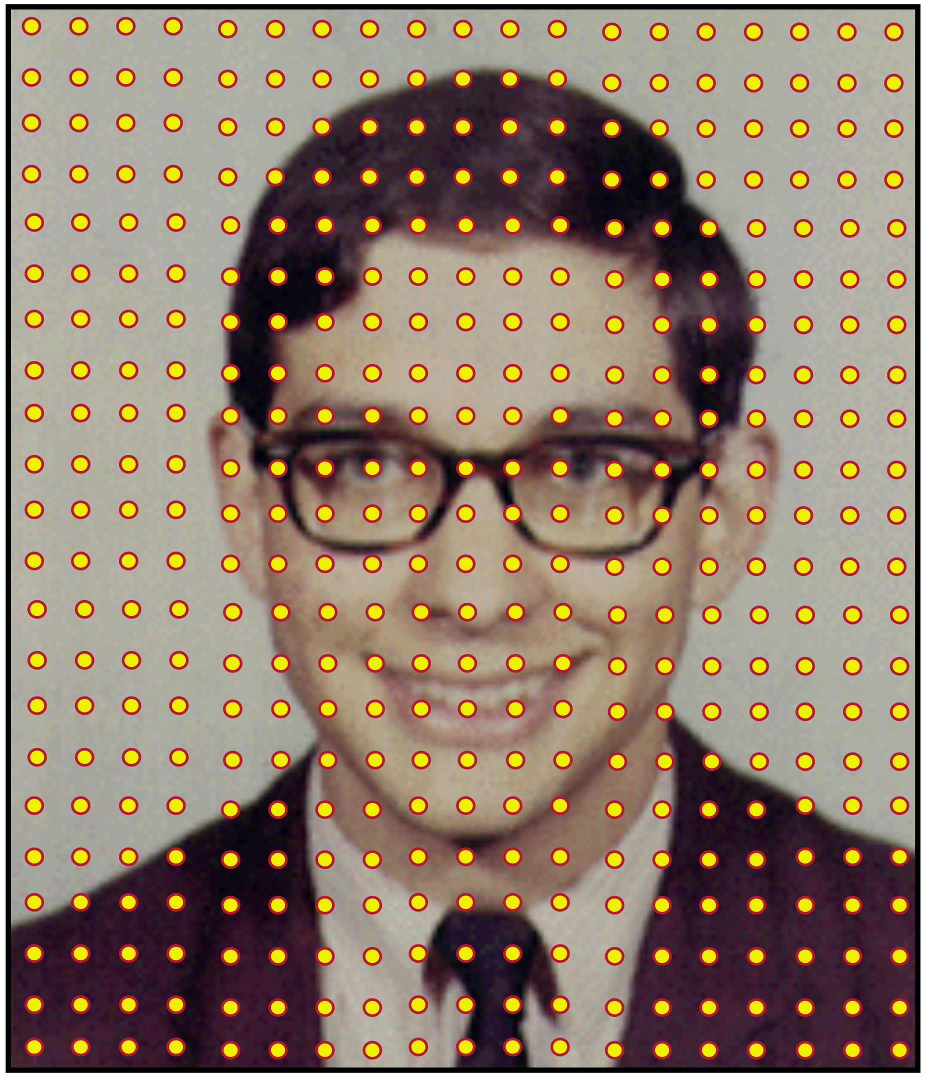
- How should we sample an image:
  - Nearest Point Sampling?
  - Bilinear Sampling?
  - Gaussian Sampling?
  - Something Else?

Image Representation

What is an image?

An image is a discrete collection of pixels, each representing a sample of a continuous function.

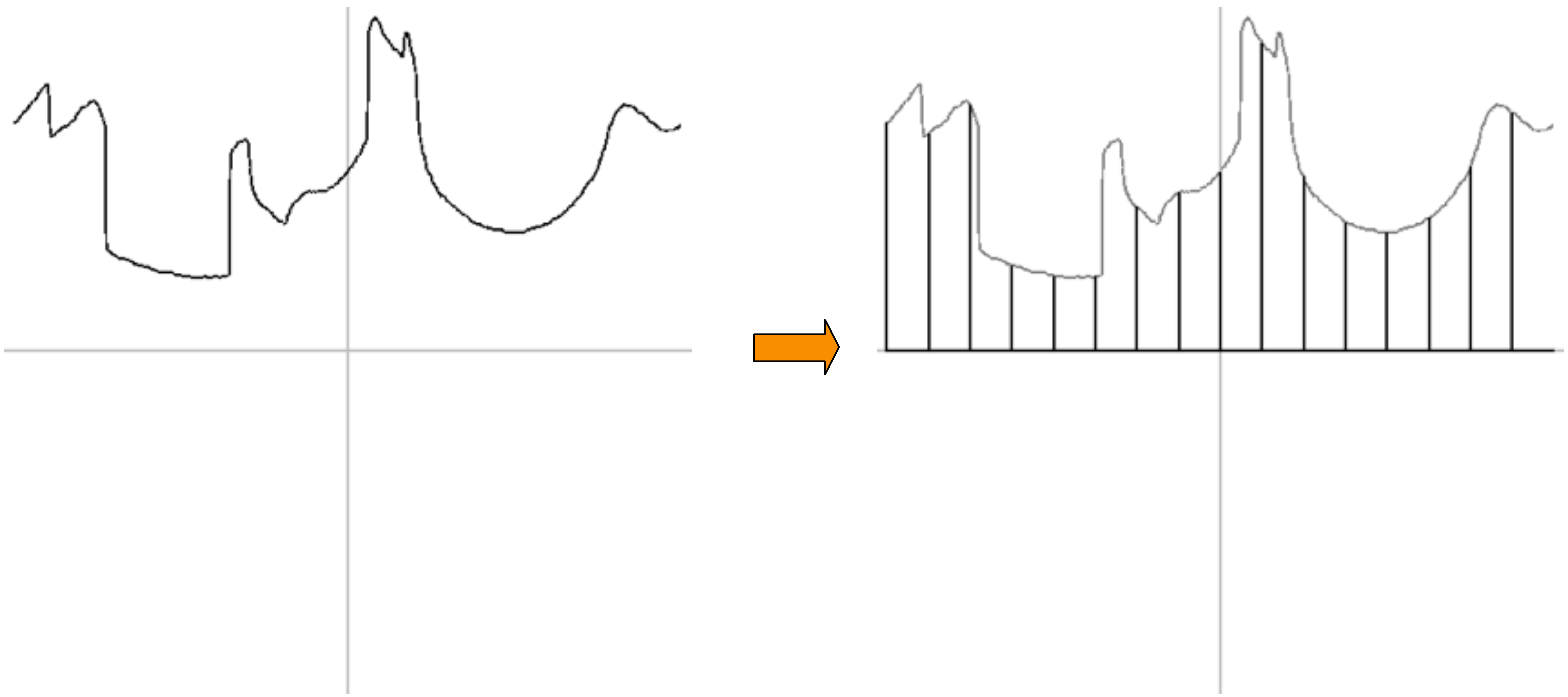
Continuous image



Digital image

Sampling

Let's look at a 1D example:

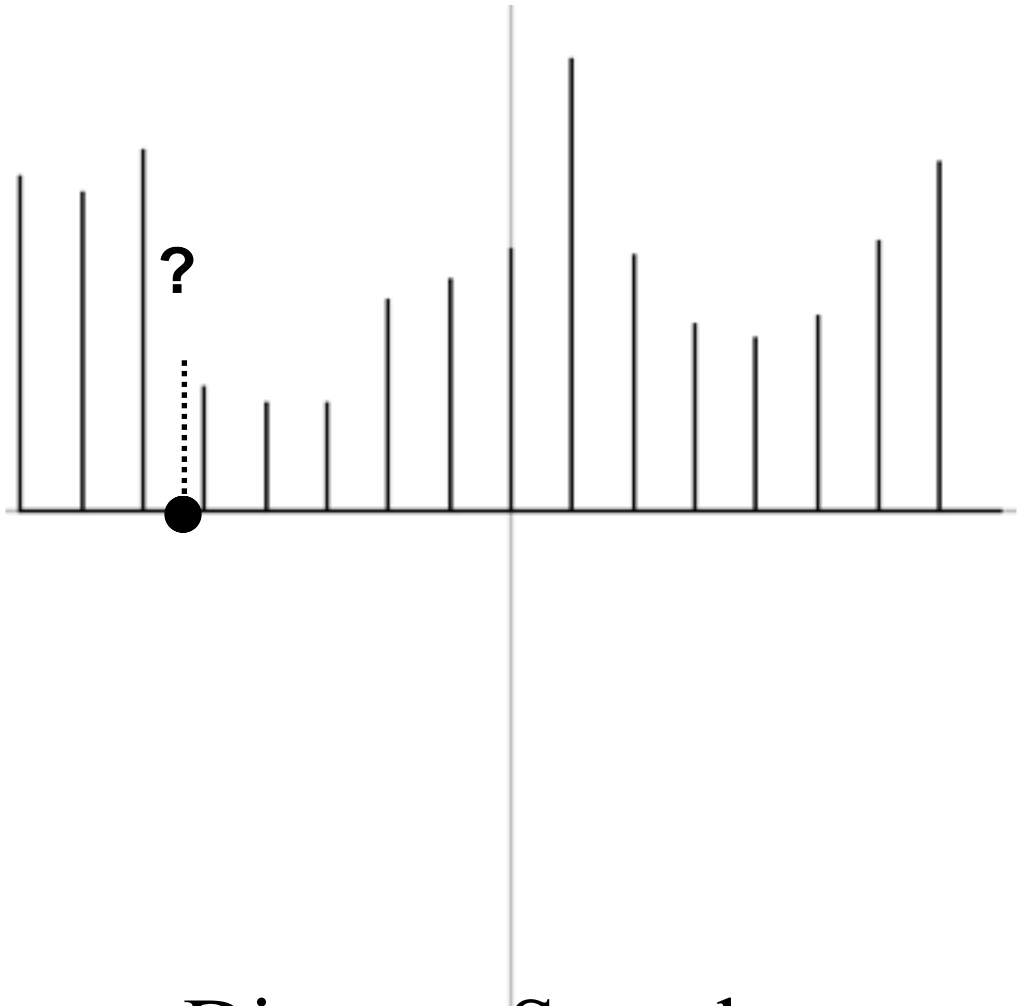


Continuous Function — Discrete Samples

Sampling

At in-between positions, values are undefined.

How do we determine the value of a sample at these locations?



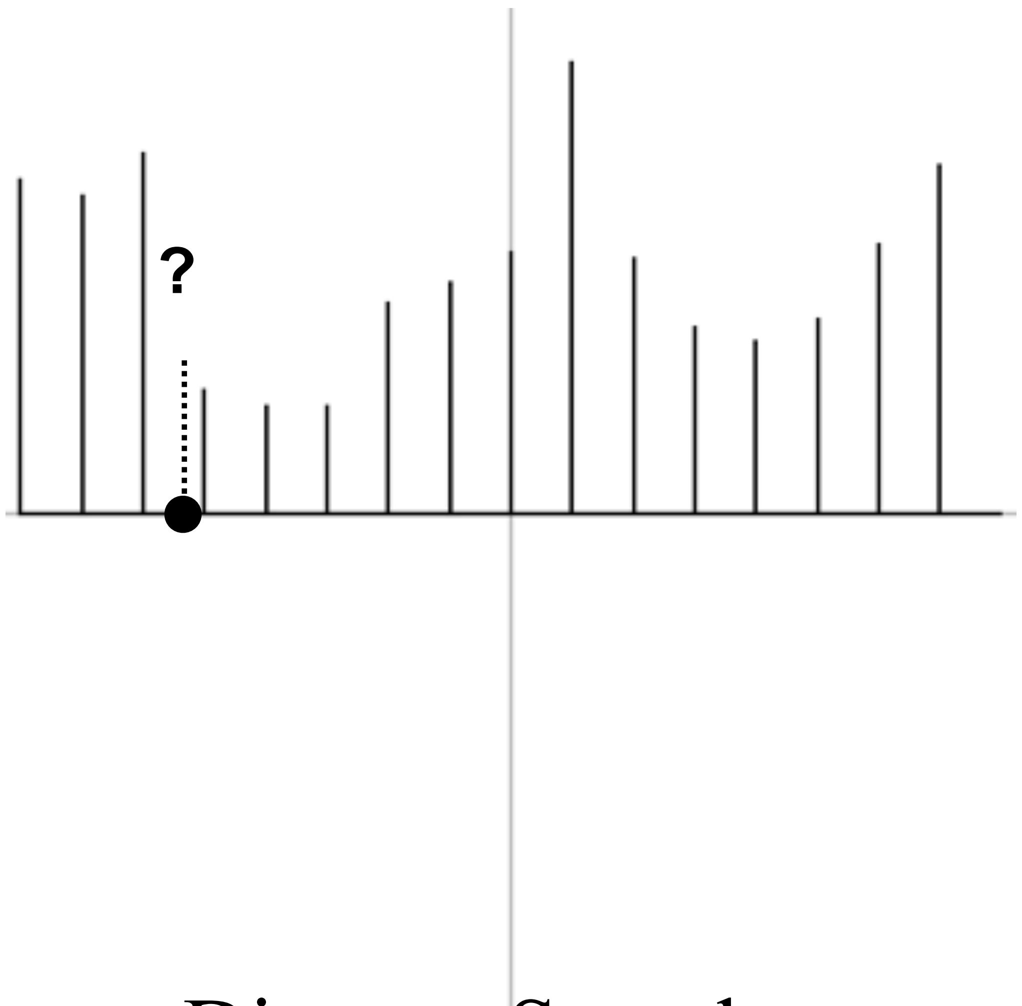
Discrete Samples

Sampling

At in-between positions, values are undefined.

How do we determine the value of a sample at these locations?

We need to reconstruct a continuous function, turning a collection of discrete samples into a 1D function that we can sample at arbitrary locations.



Discrete Samples

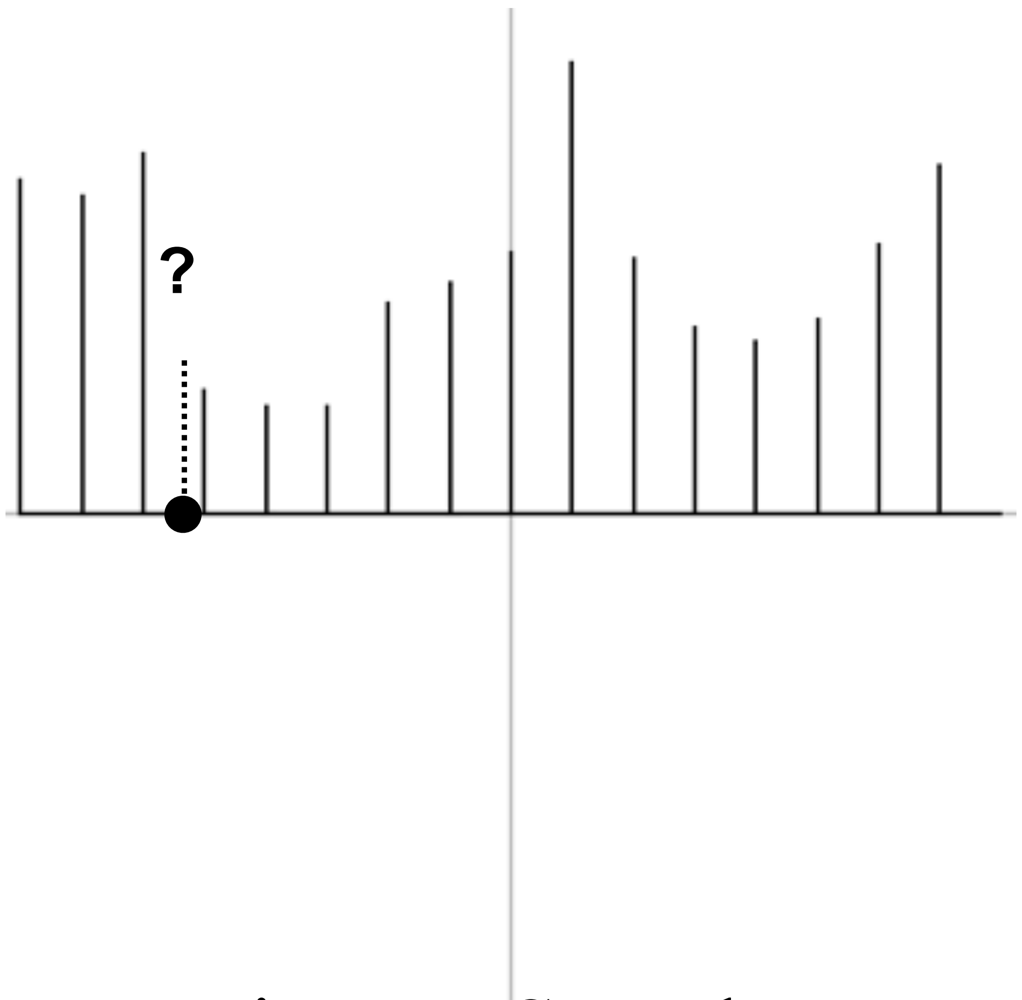
Sampling

At in-between positions, values are undefined.

How do we determine the value of a sample at these locations?

We need to reconstruct a continuous function, turning a collection of discrete samples into a 1D function that we can sample at arbitrary locations.

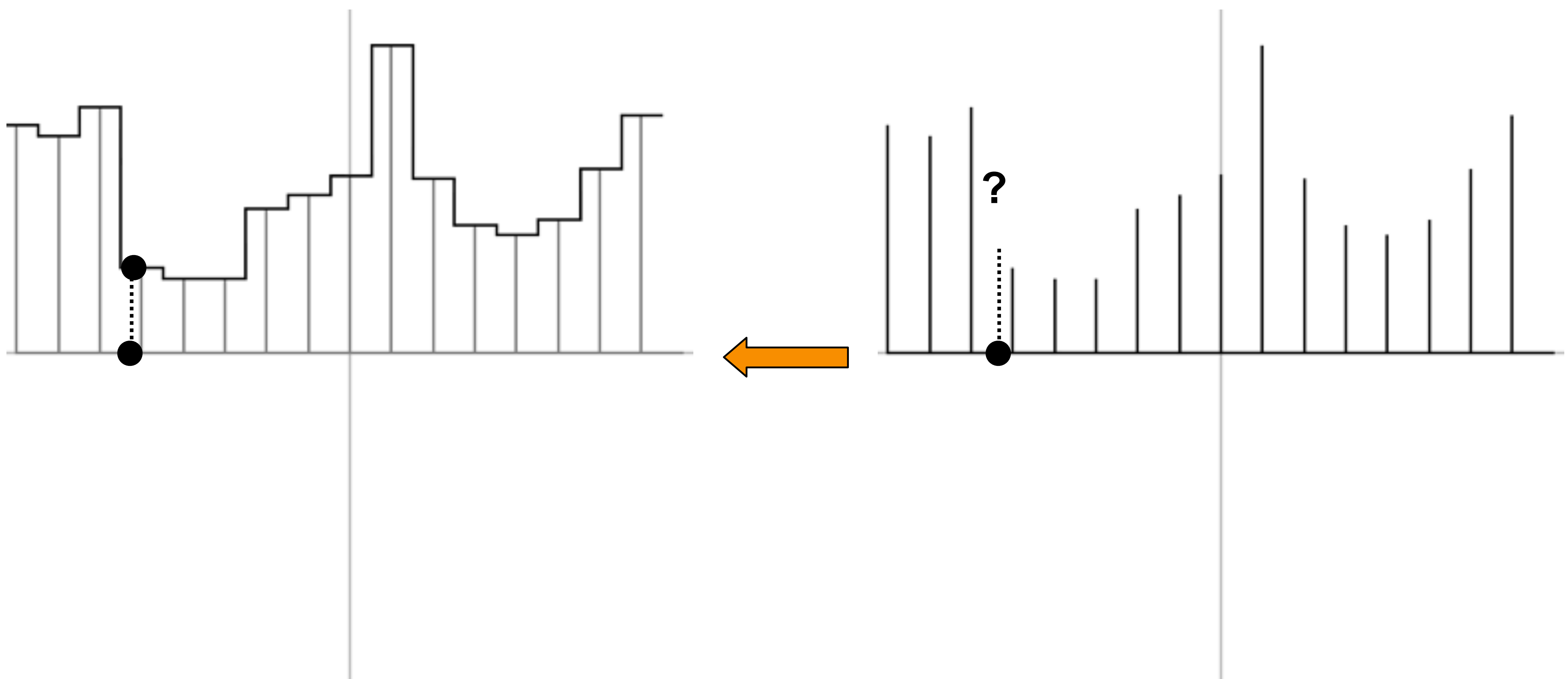
In other words: "How do we define the in-between values?"



Discrete Samples

Nearest Point Sampling

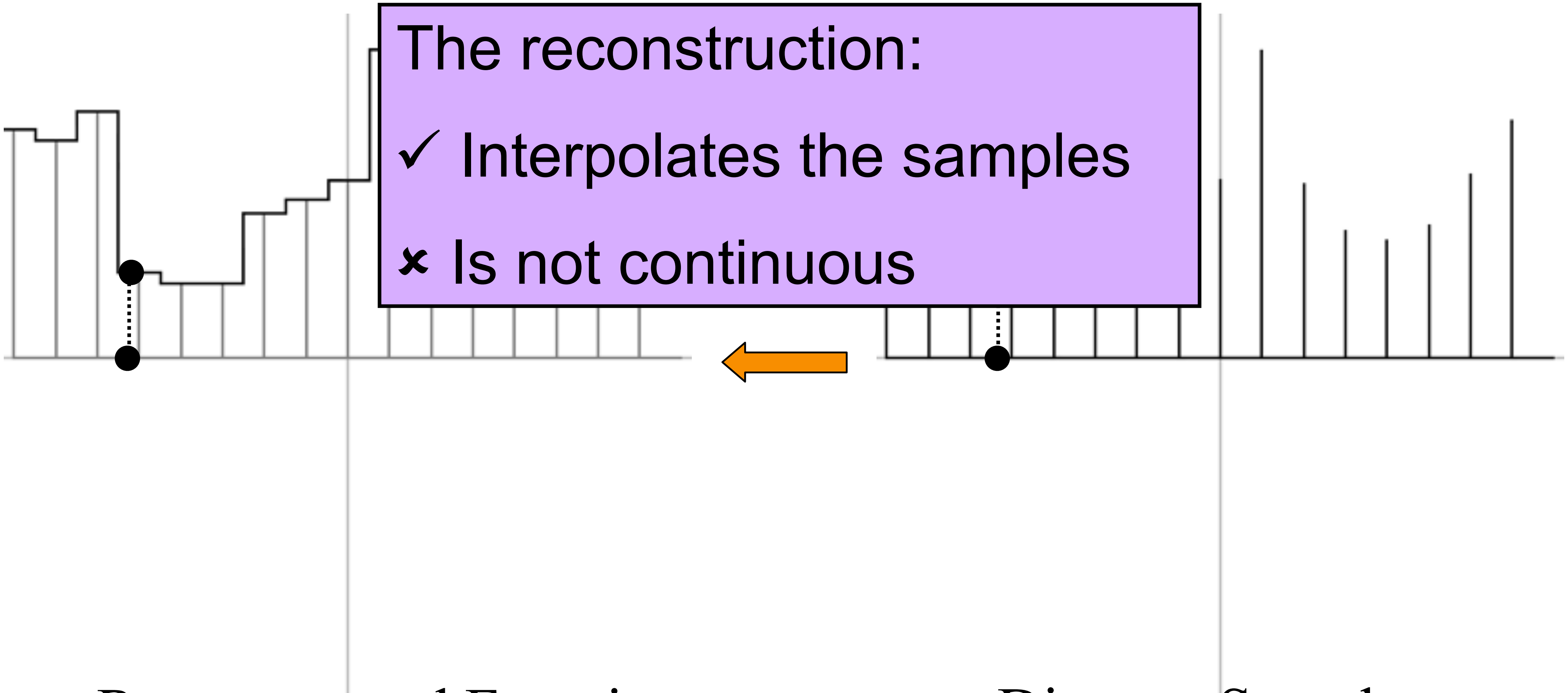
The value at a point is the value of the closest discrete sample.



Reconstructed Function Discrete Samples

Nearest Point Sampling

The value at a point is the value of the closest discrete sample.



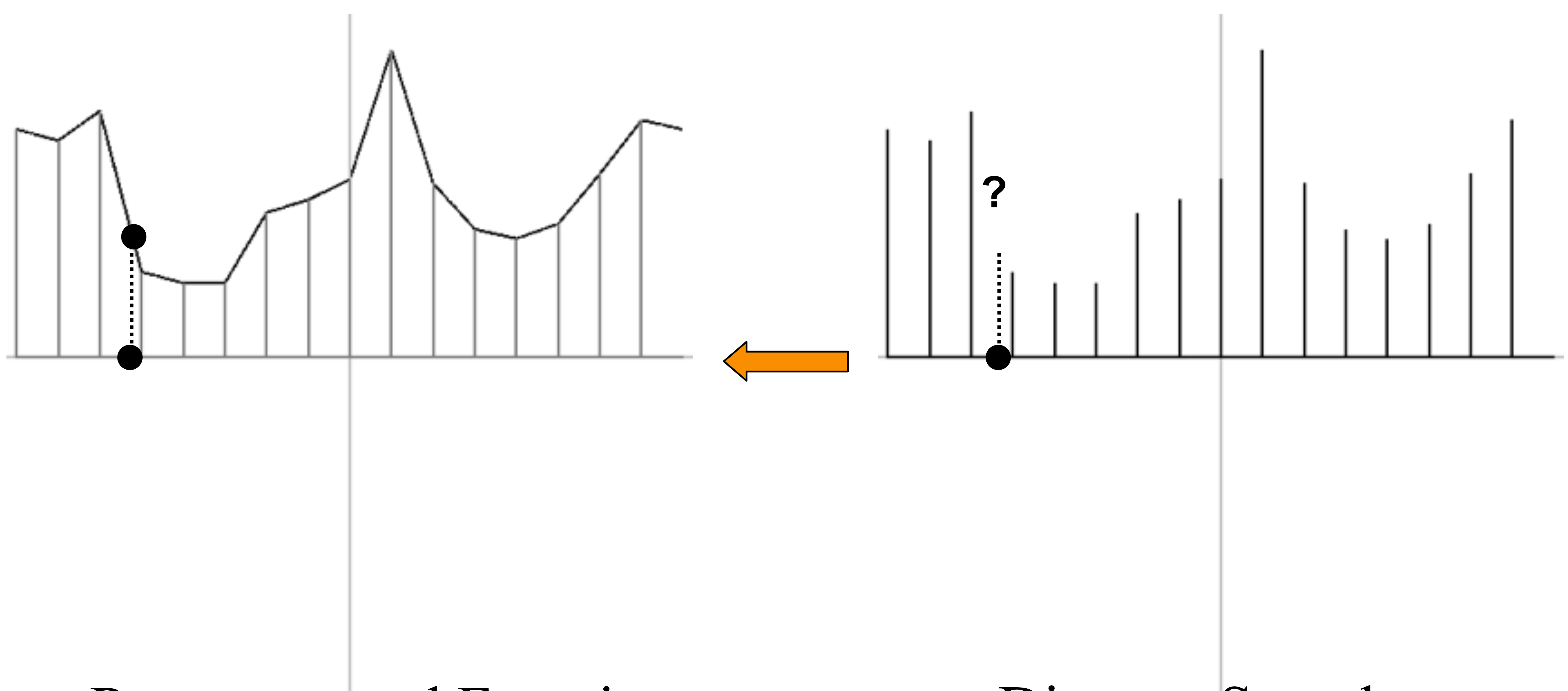
The reconstruction:

✓ Interpolates the samples

✗ Is not continuous

Bilinear Sampling

The value at a point is the (bi)linear interpolation of the two surrounding samples.

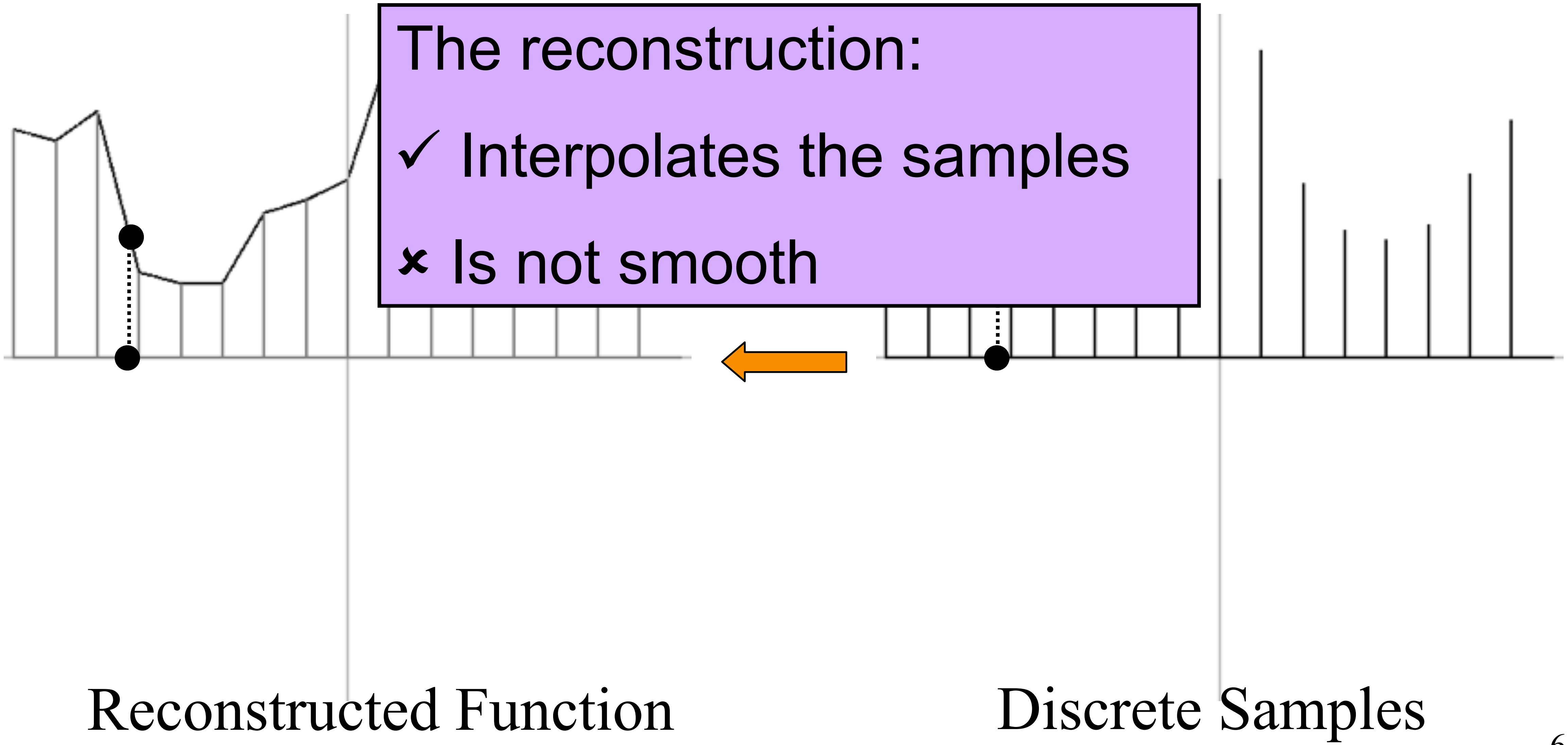


Reconstructed Function

Discrete Samples

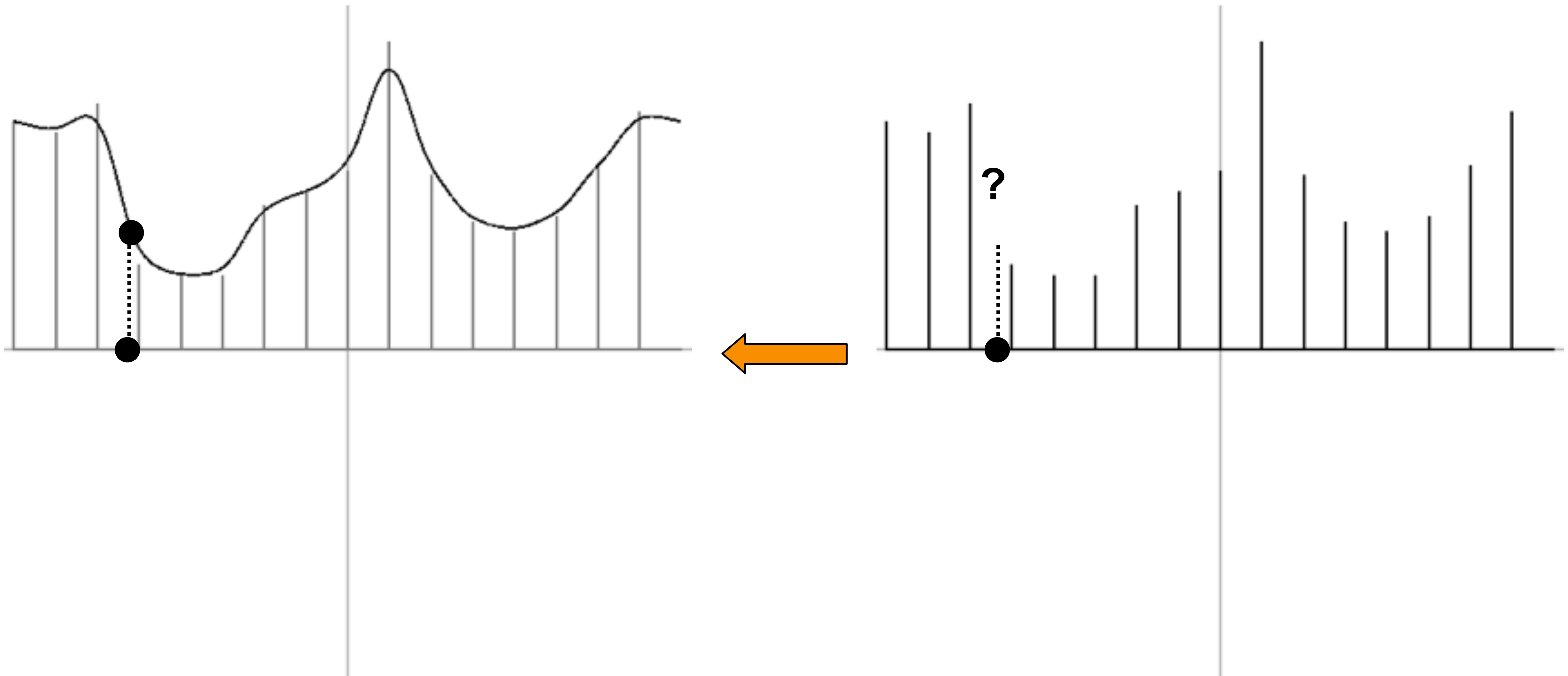
Bilinear Sampling

The value at a point is the (bi)linear interpolation of the two surrounding samples.



Gaussian Sampling

The value at a point is the Gaussian average of the surrounding samples.

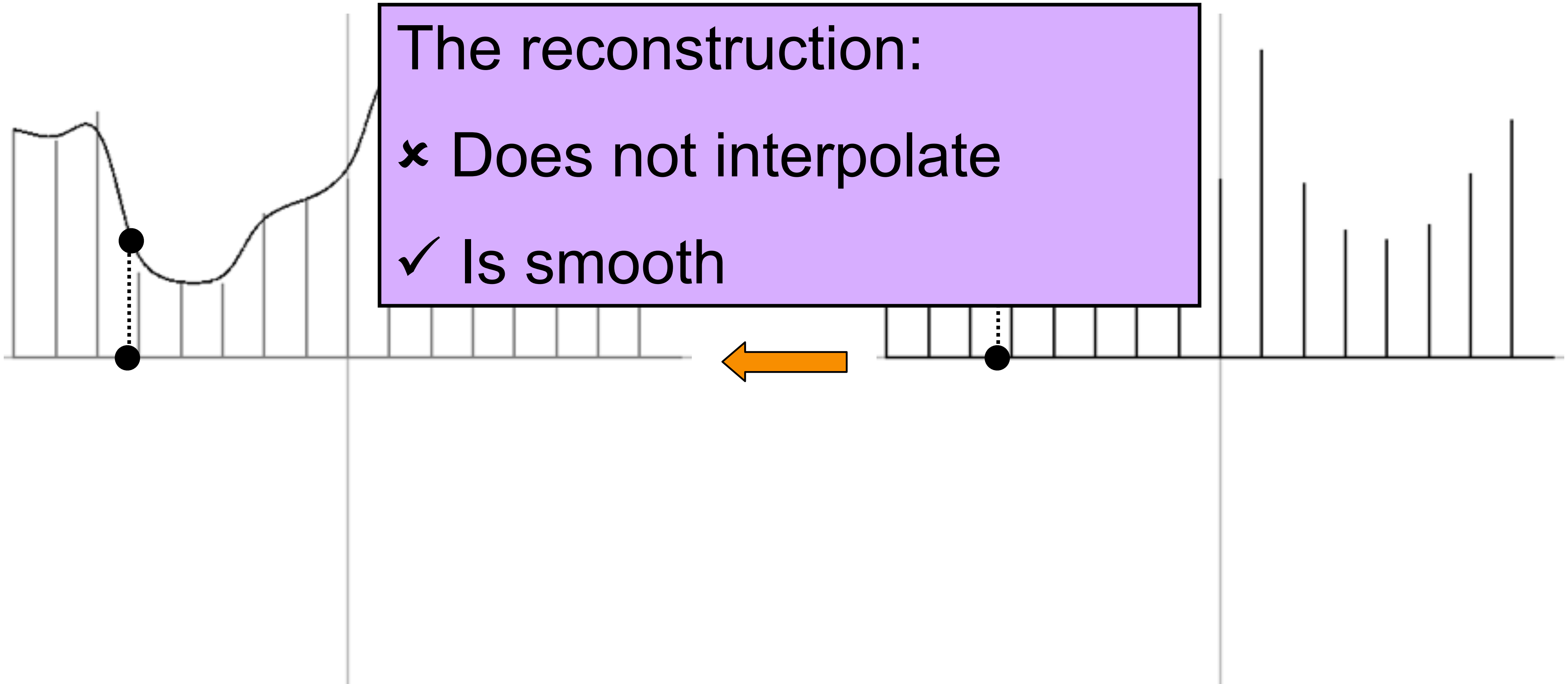


Reconstructed Function

Discrete Samples

Gaussian Sampling

The value at a point is the Gaussian average of the surrounding samples.



The reconstruction:

- ✗ Does not interpolate
- ✓ Is smooth

Image Sampling

- How do we reconstruct a function from a collection of samples?

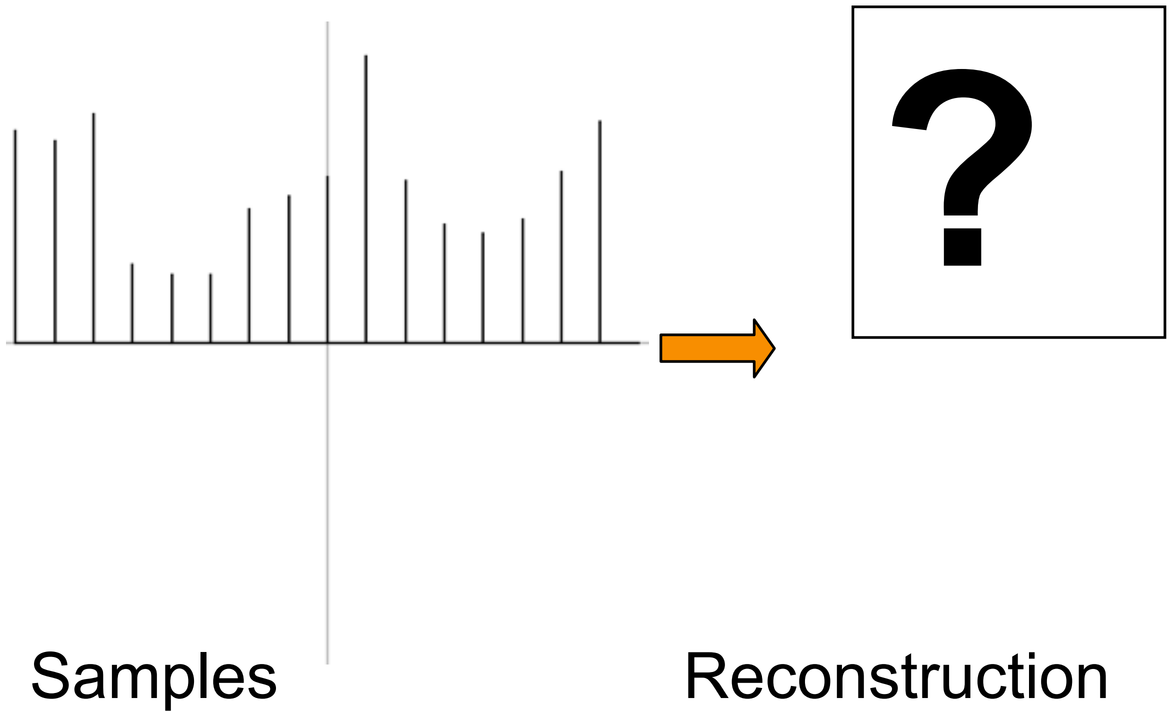


Image Sampling

- How do we reconstruct a function from a collection of samples?
- To answer this question, we need to understand what kind of information the samples contain.

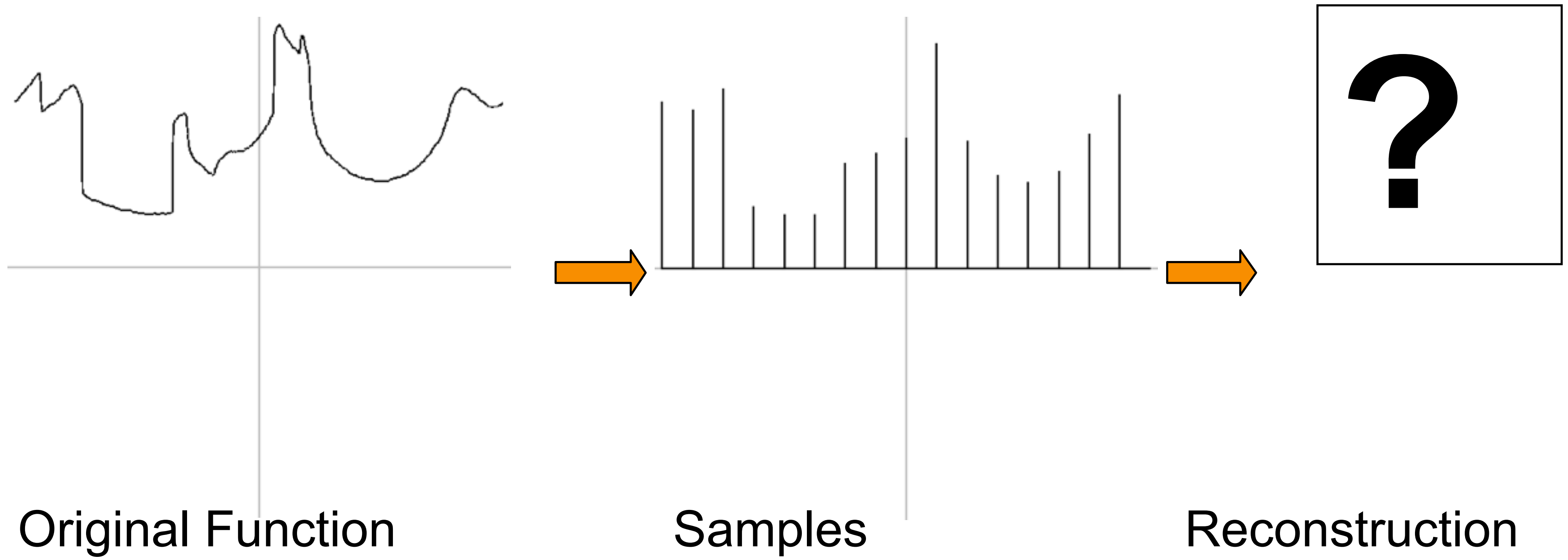
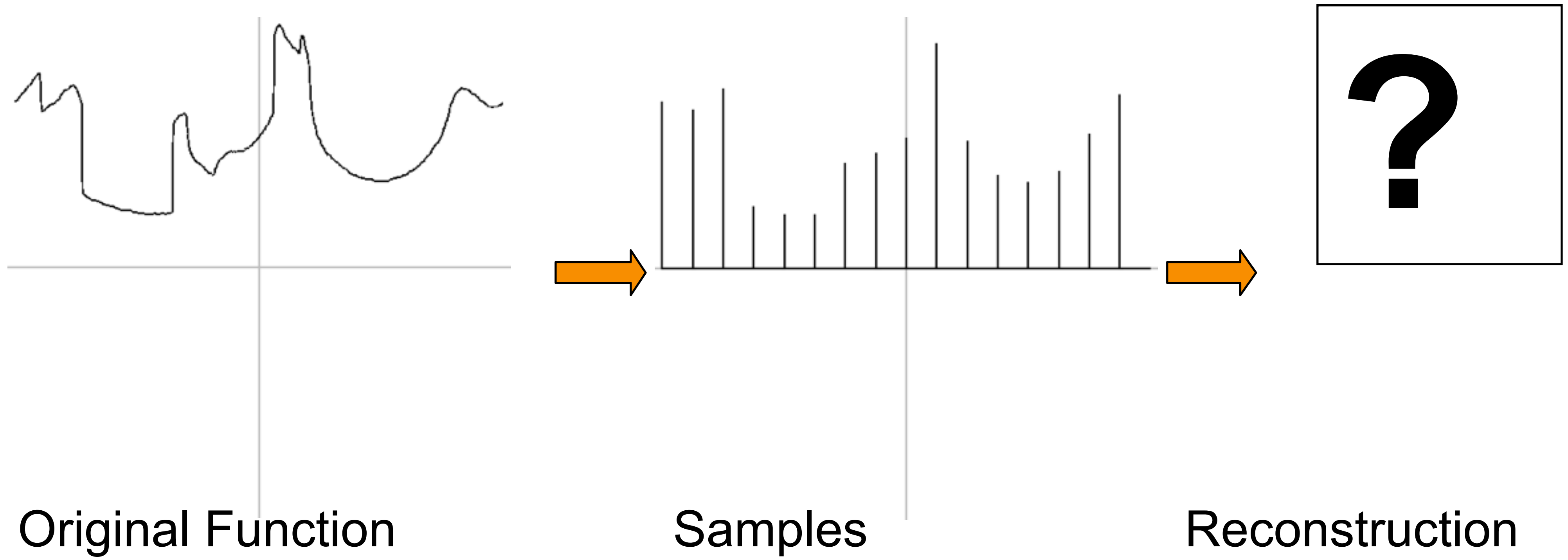


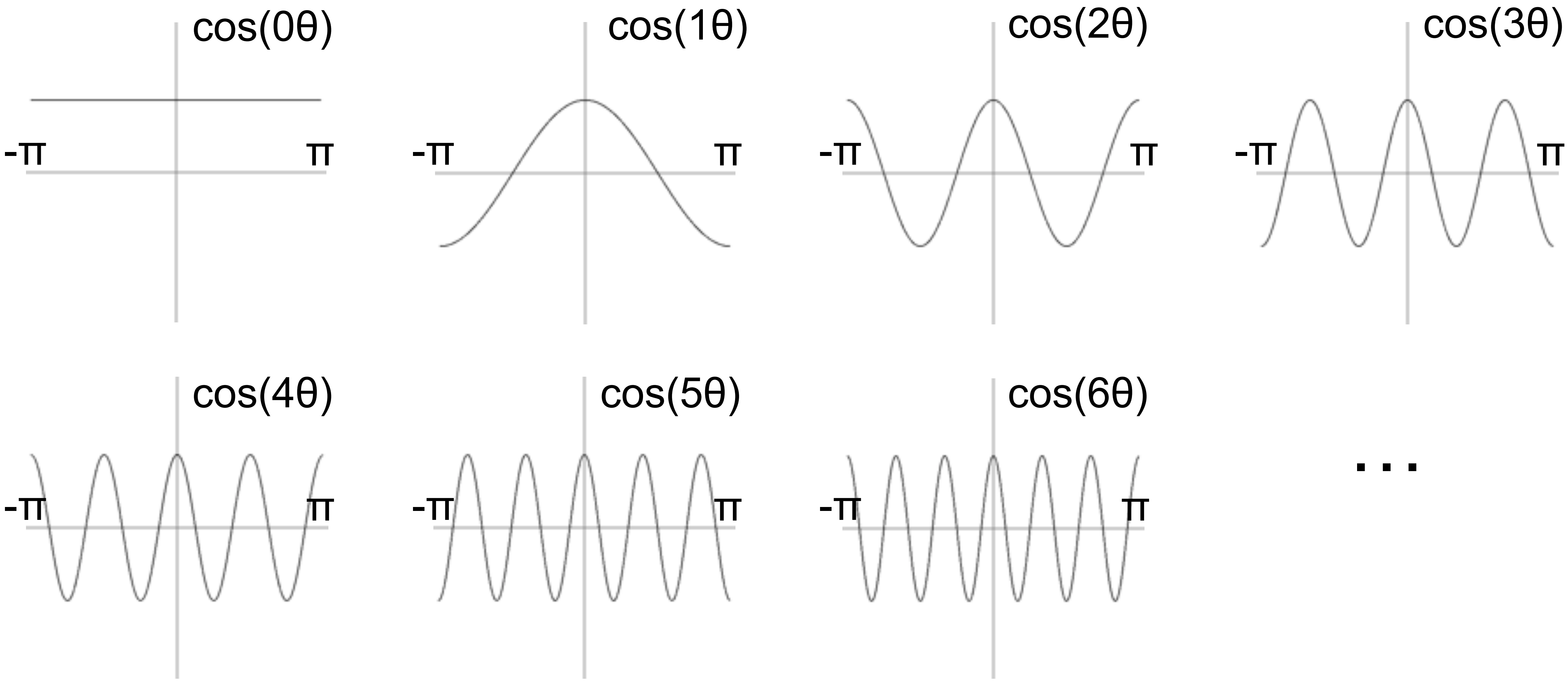
Image Sampling

- How do we reconstruct a function from a collection of samples?
- To answer this question, we need to understand what kind of information the samples contain.
- Signal processing helps us understand this better.



Fourier Analysis

- Fourier analysis provides a way for expressing (or approximating) any signal as a sum of scaled and shifted cosine functions.

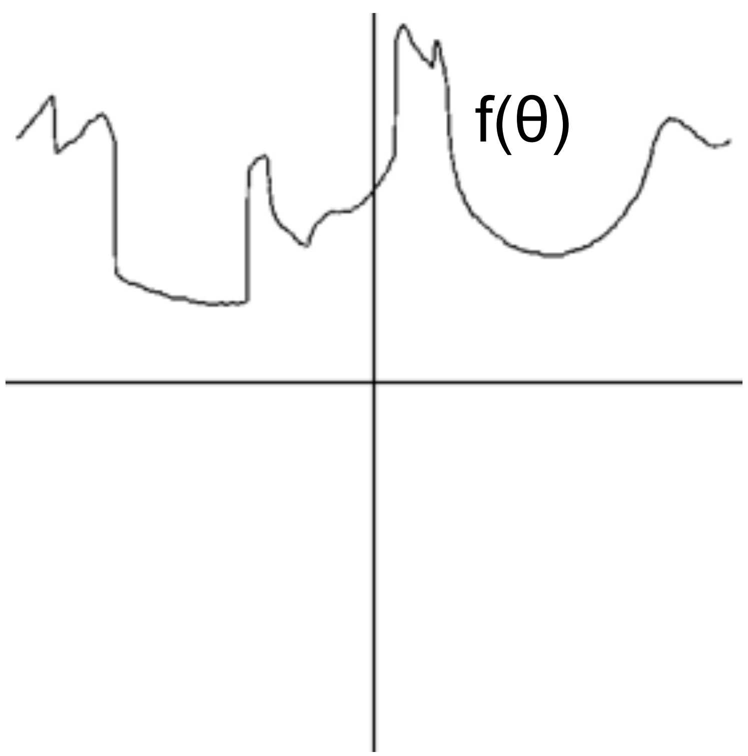


The Building Blocks for the Fourier Decomposition

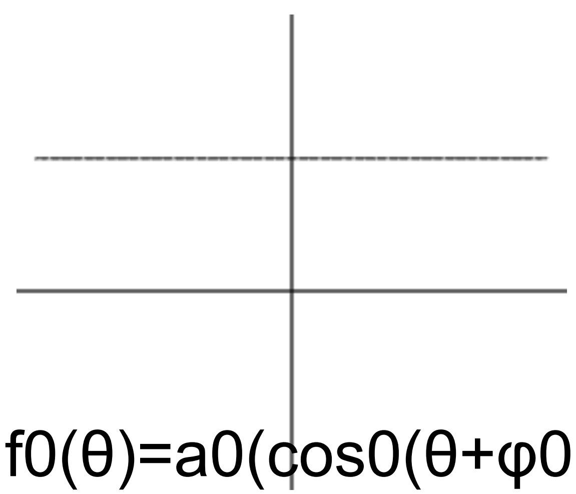
Fourier Analysis

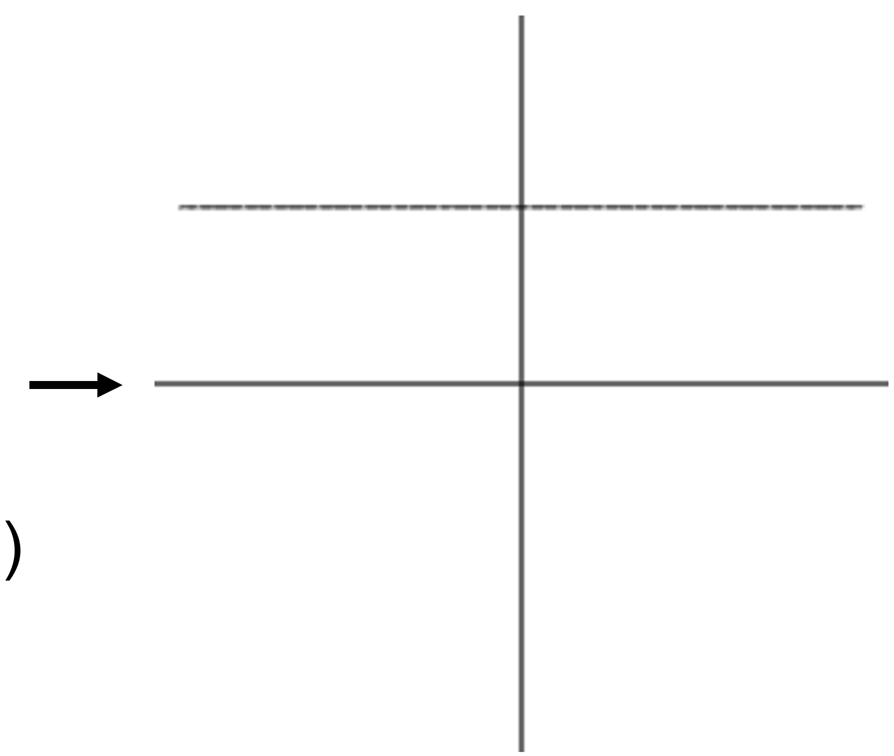
- As higher frequency components are added to the approximation, finer details are captured.

Initial Function



0th Order Approximation



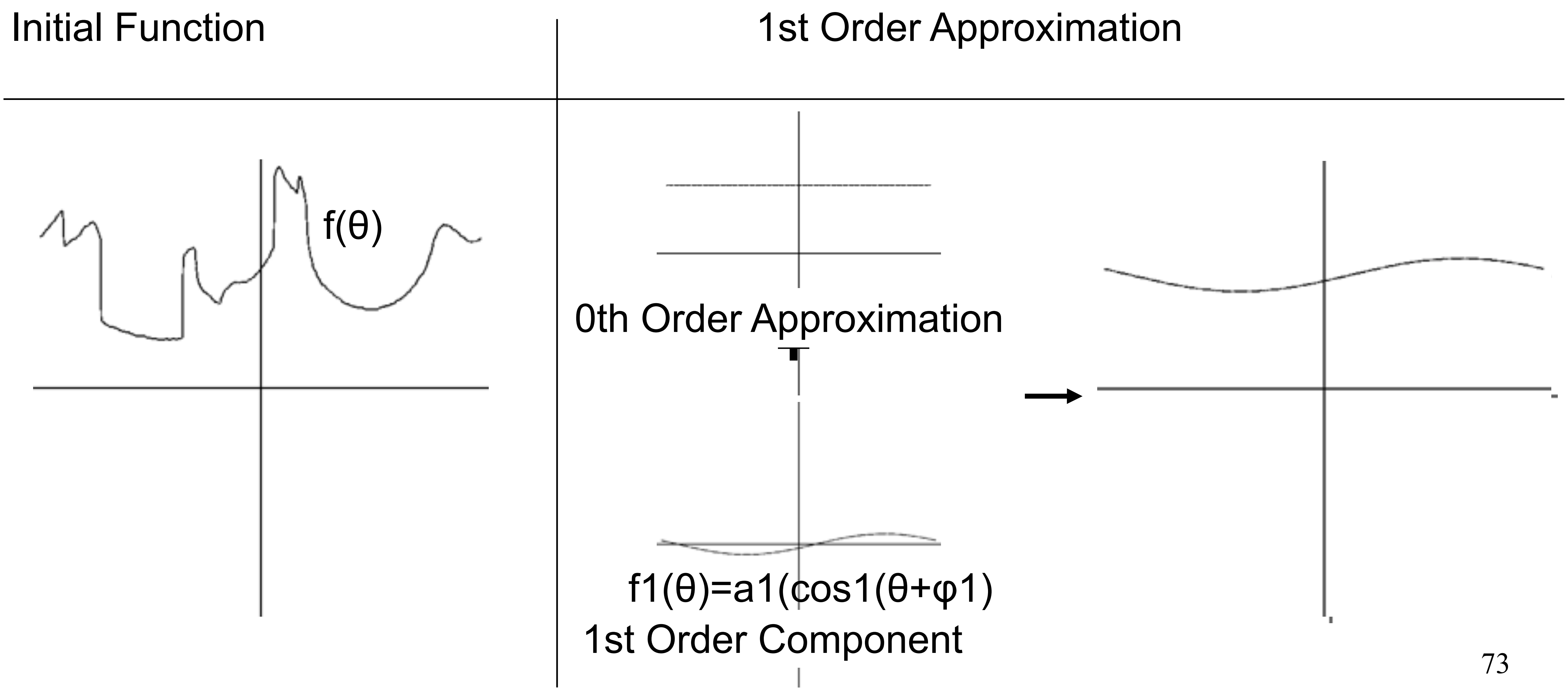


$f_0(\theta)=a_0(\cos 0(\theta+\varphi_0)$

0th Order Component

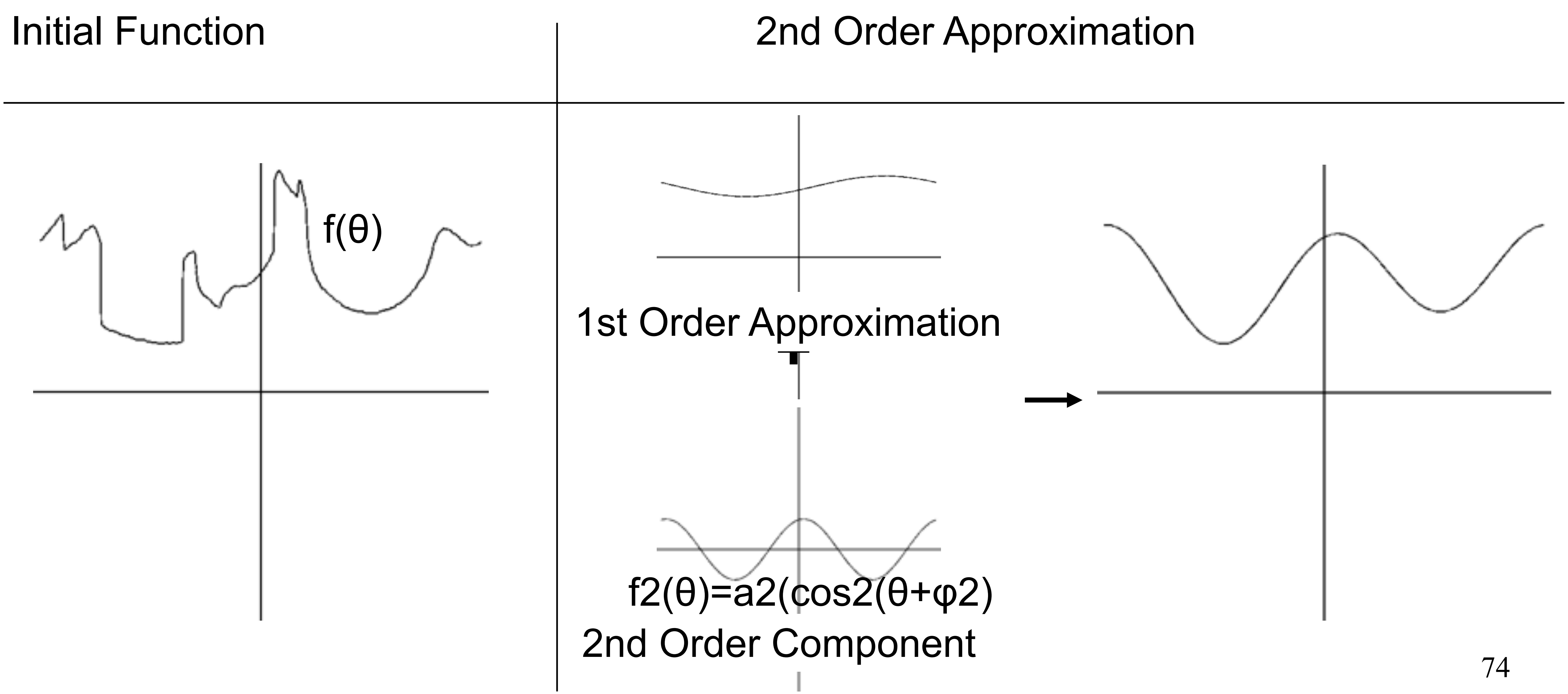
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



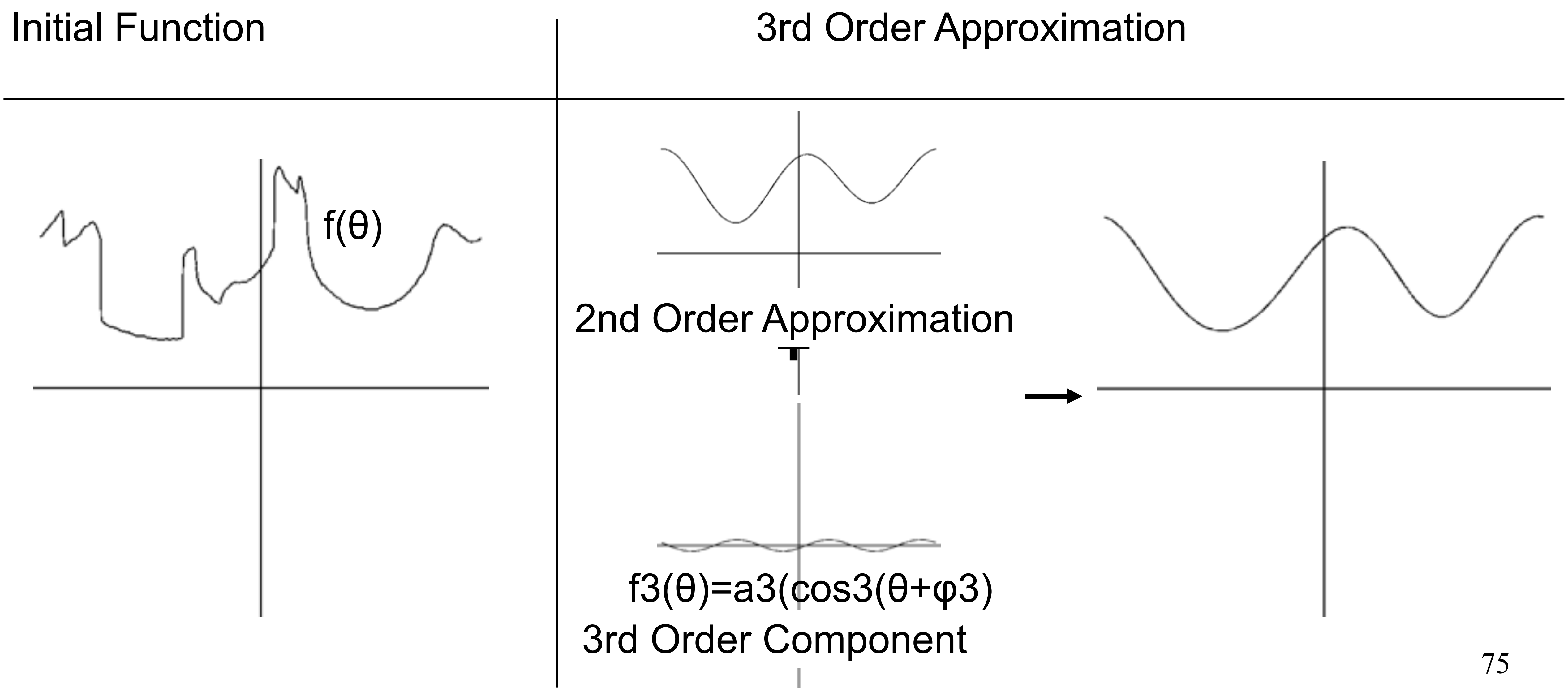
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



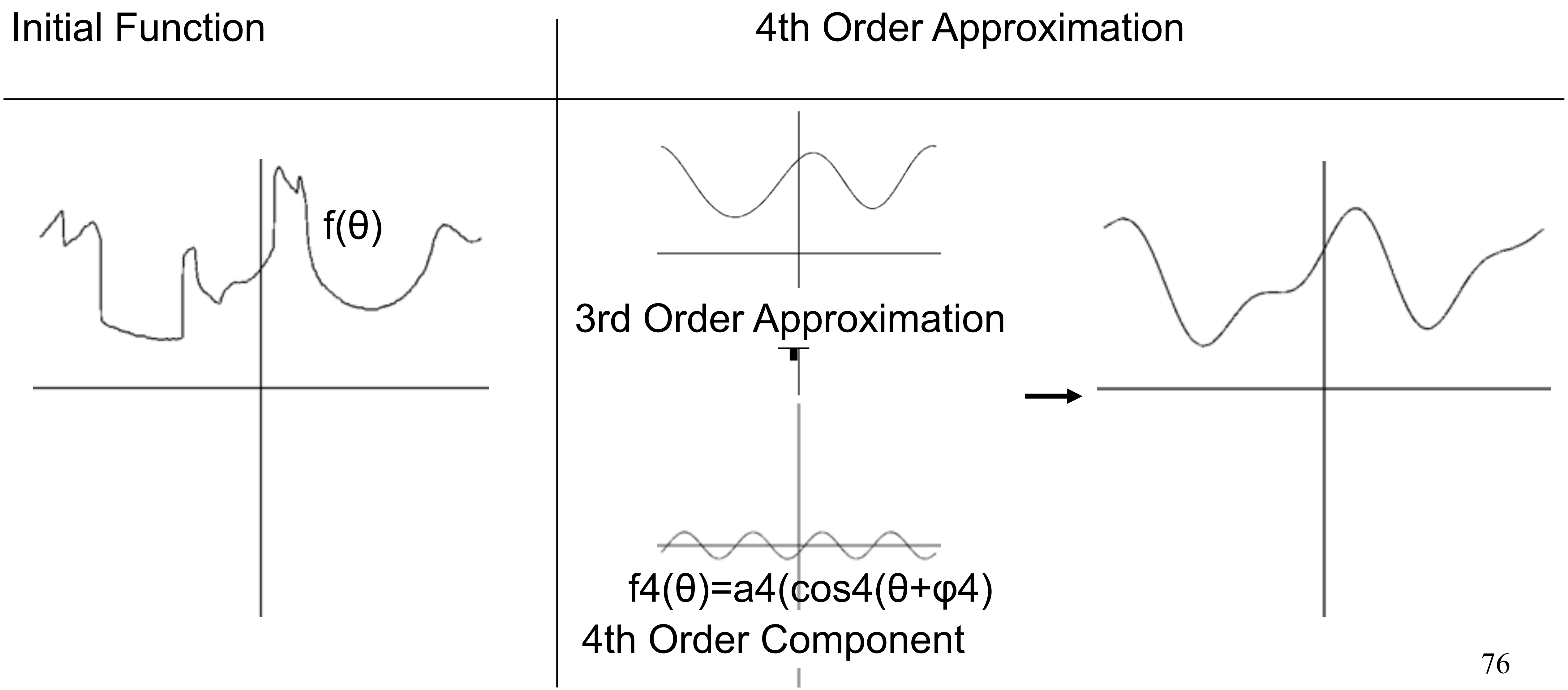
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



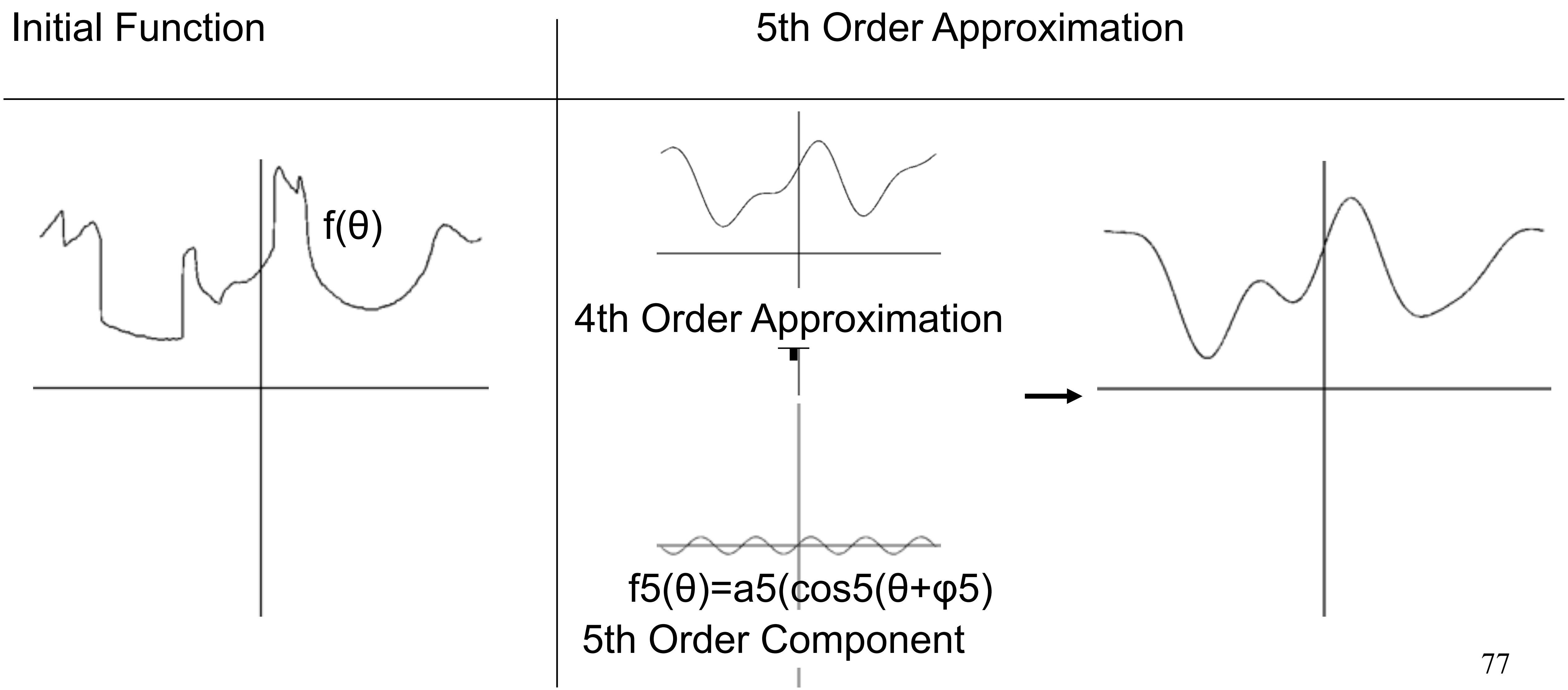
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



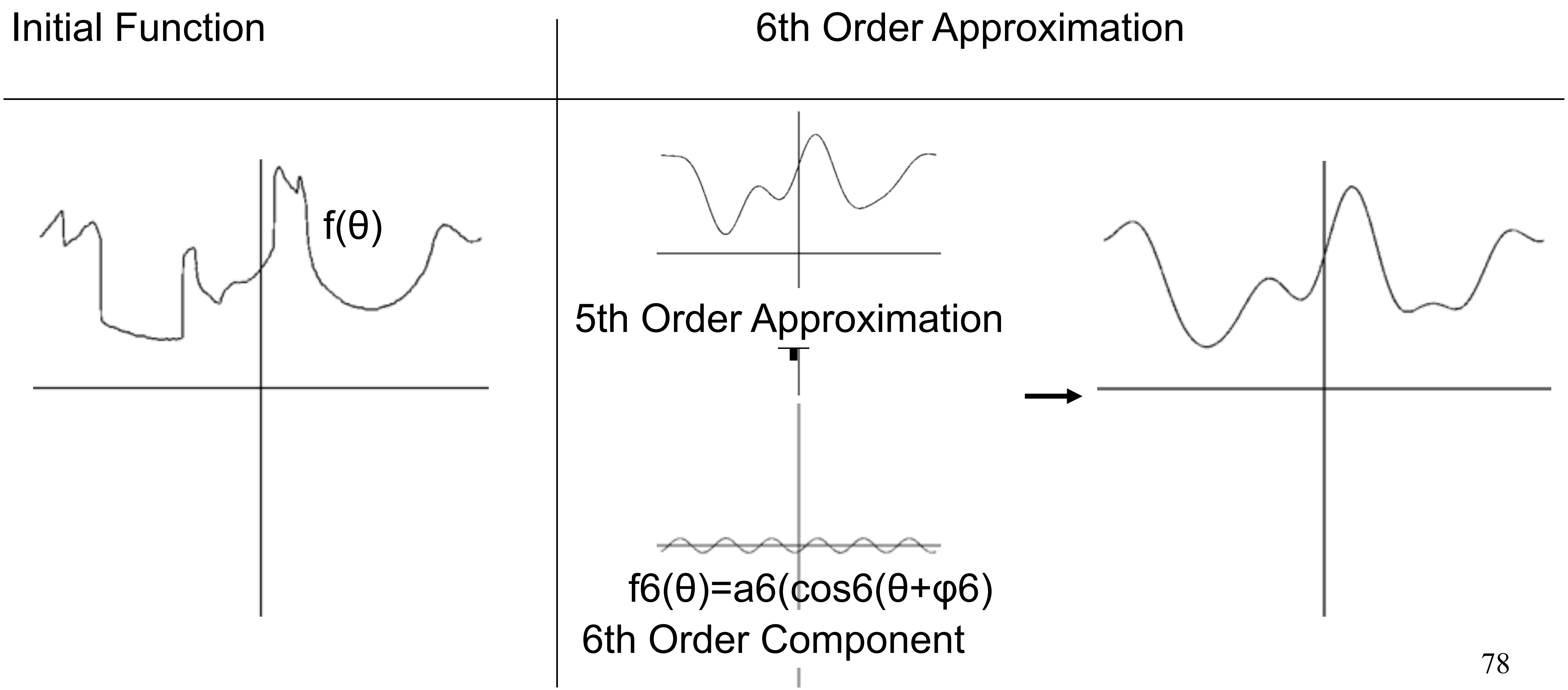
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



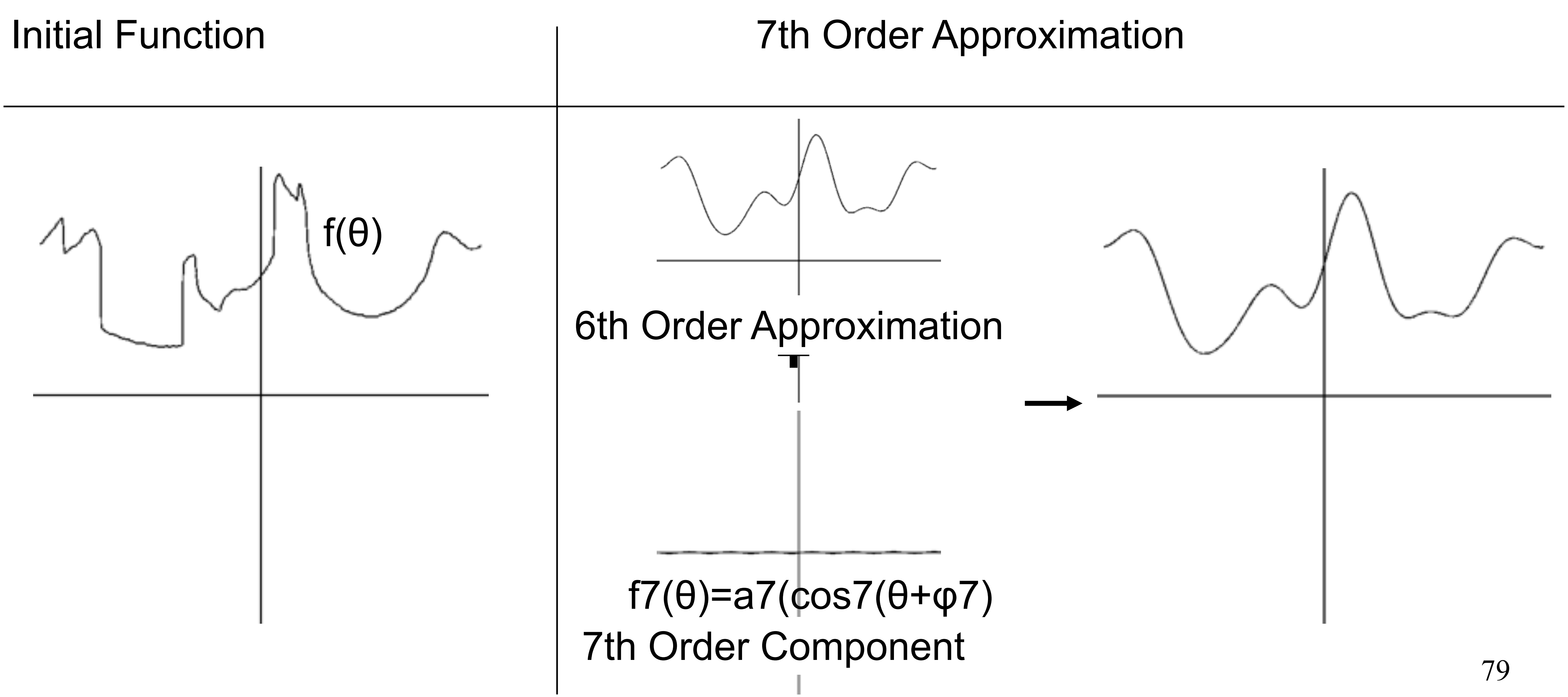
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



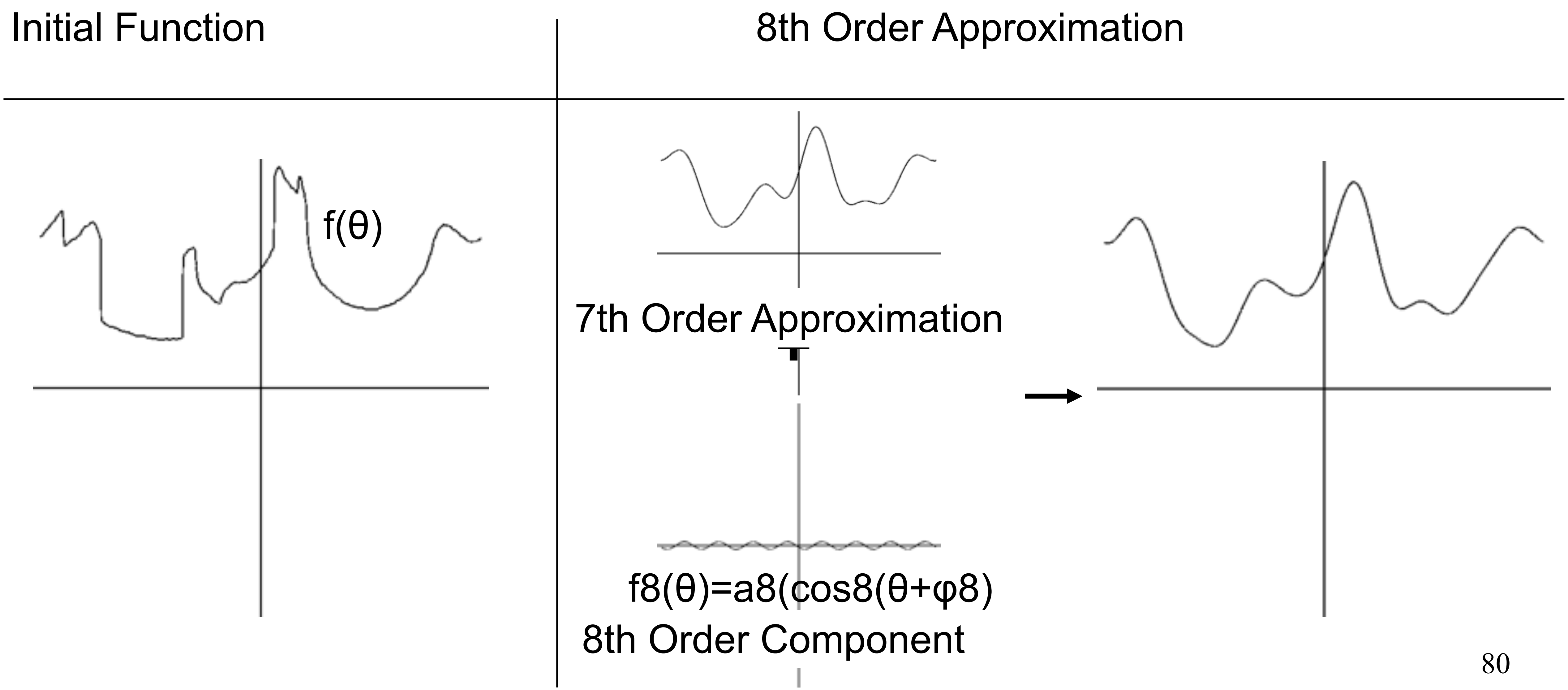
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



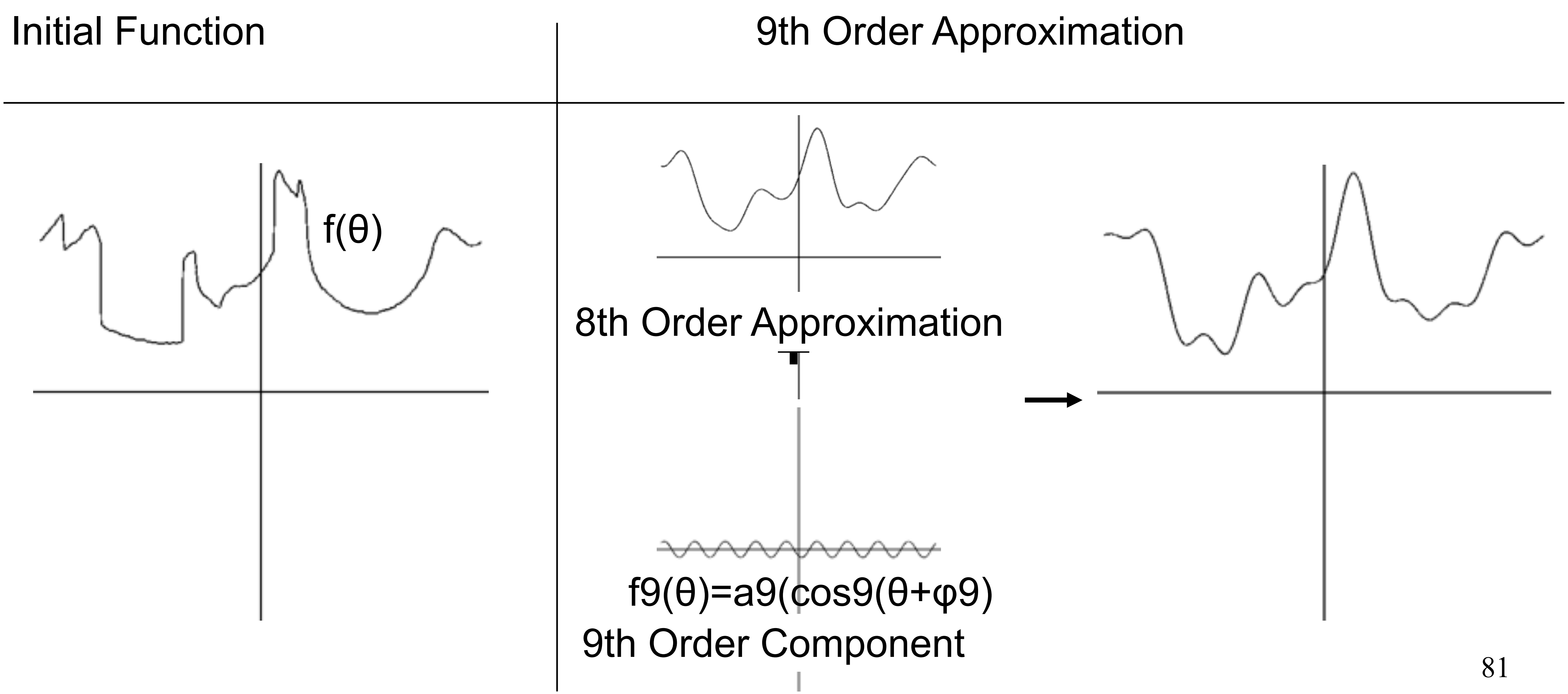
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



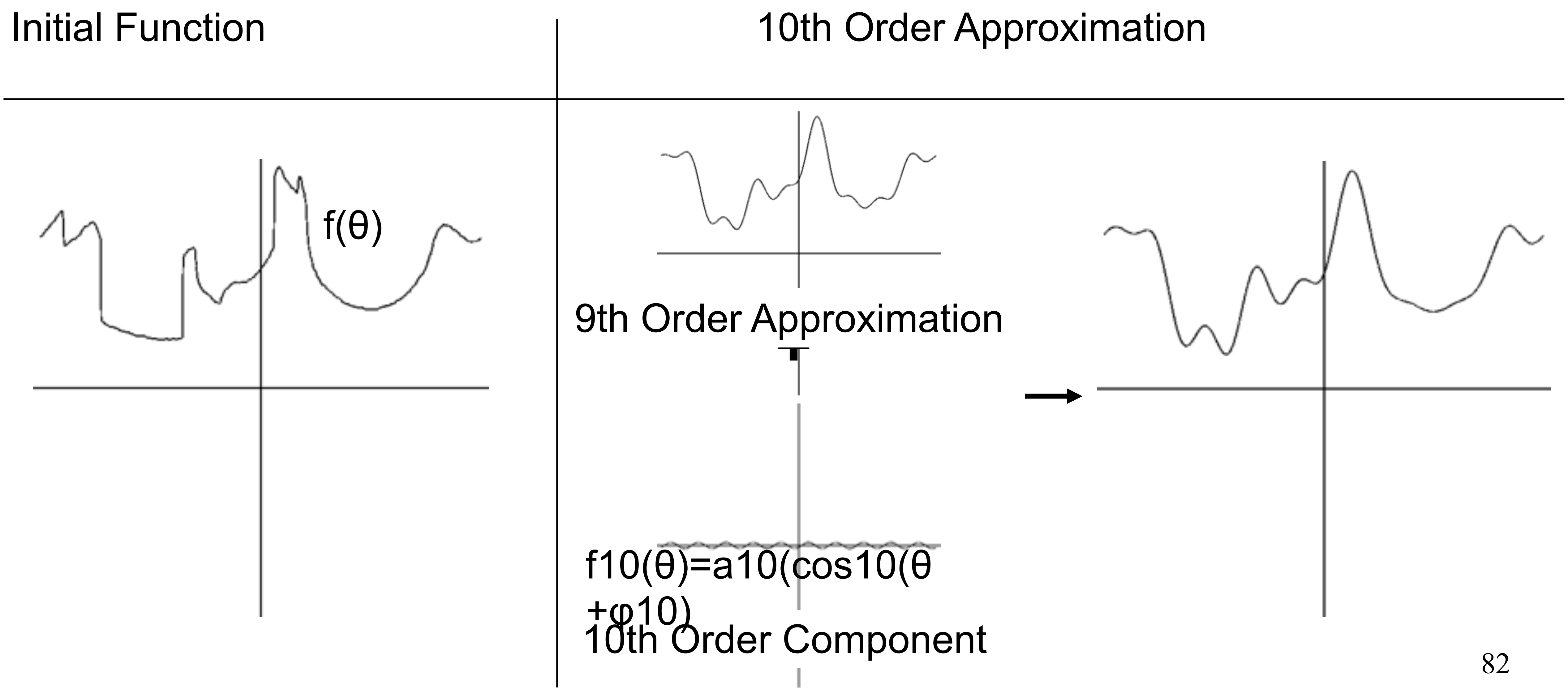
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



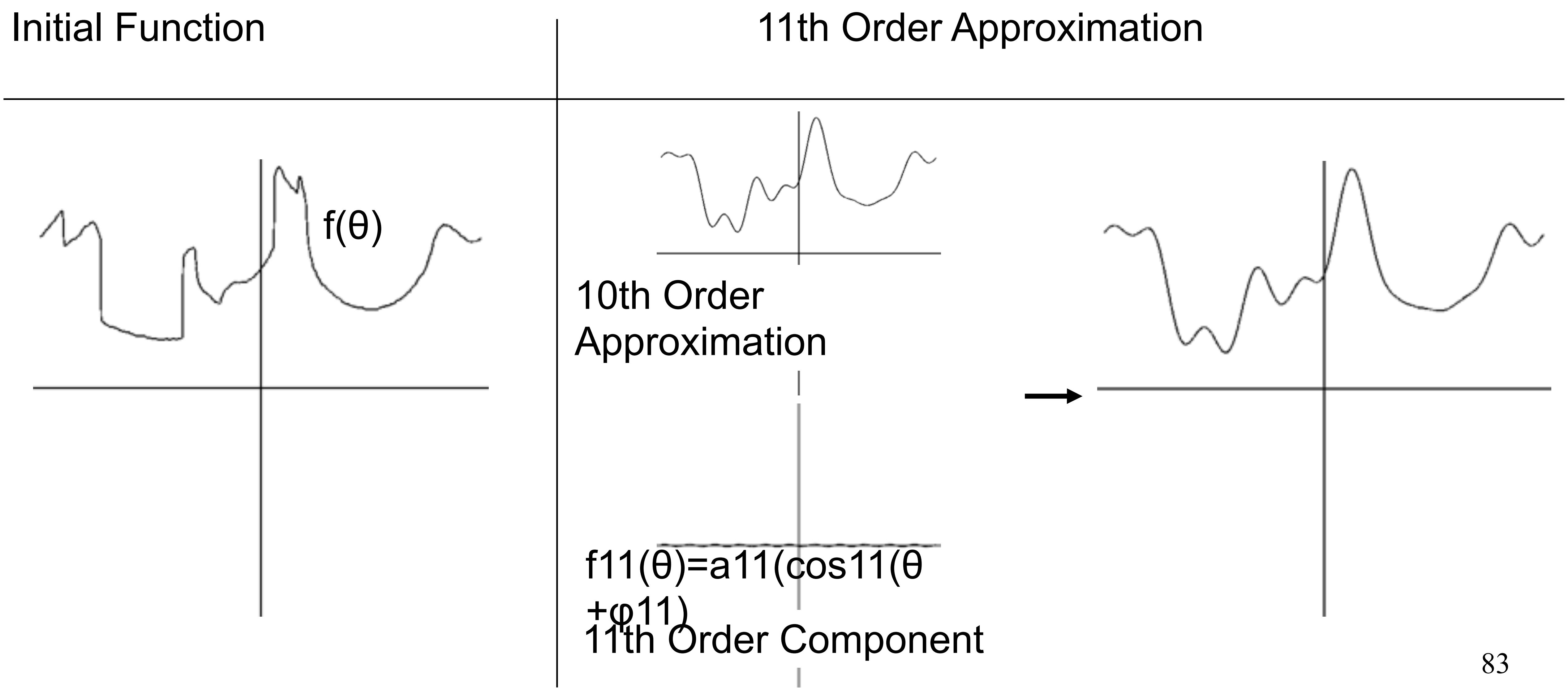
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



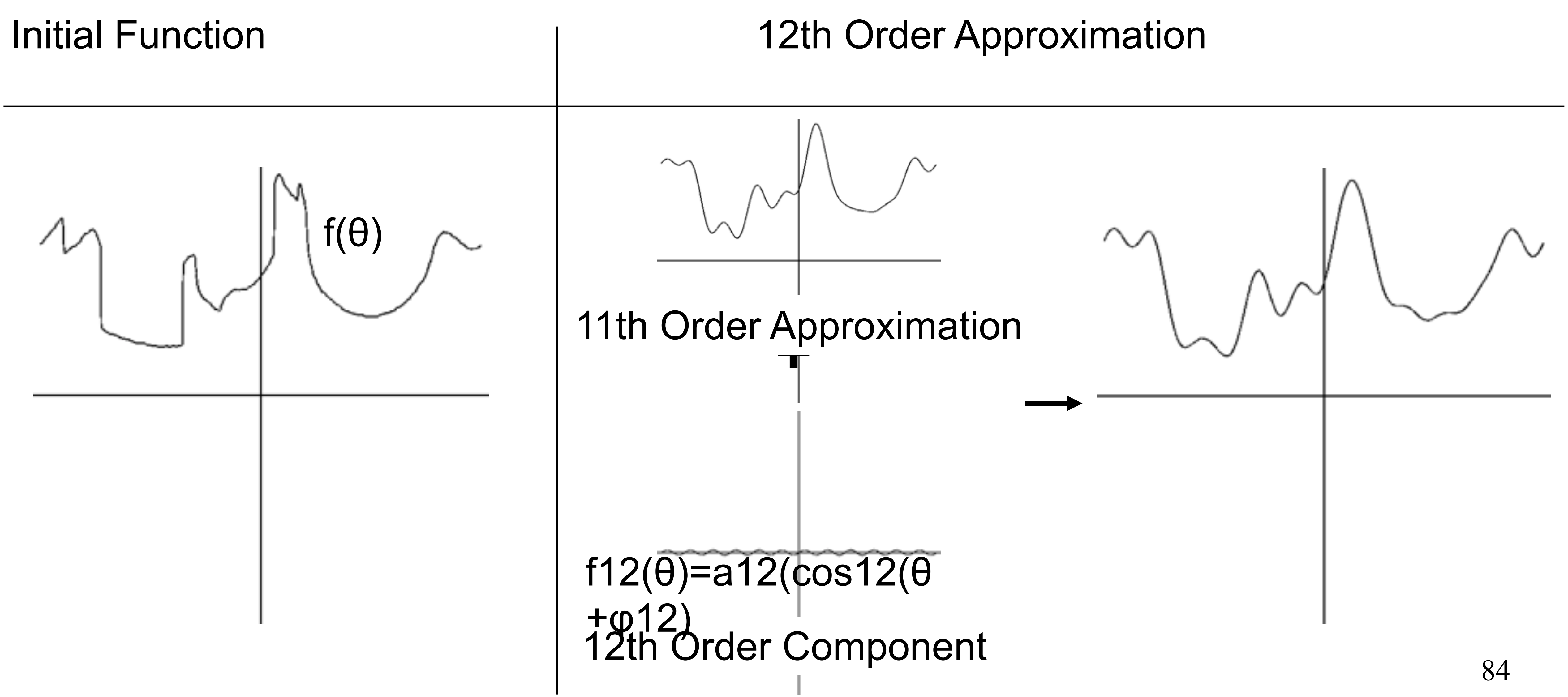
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



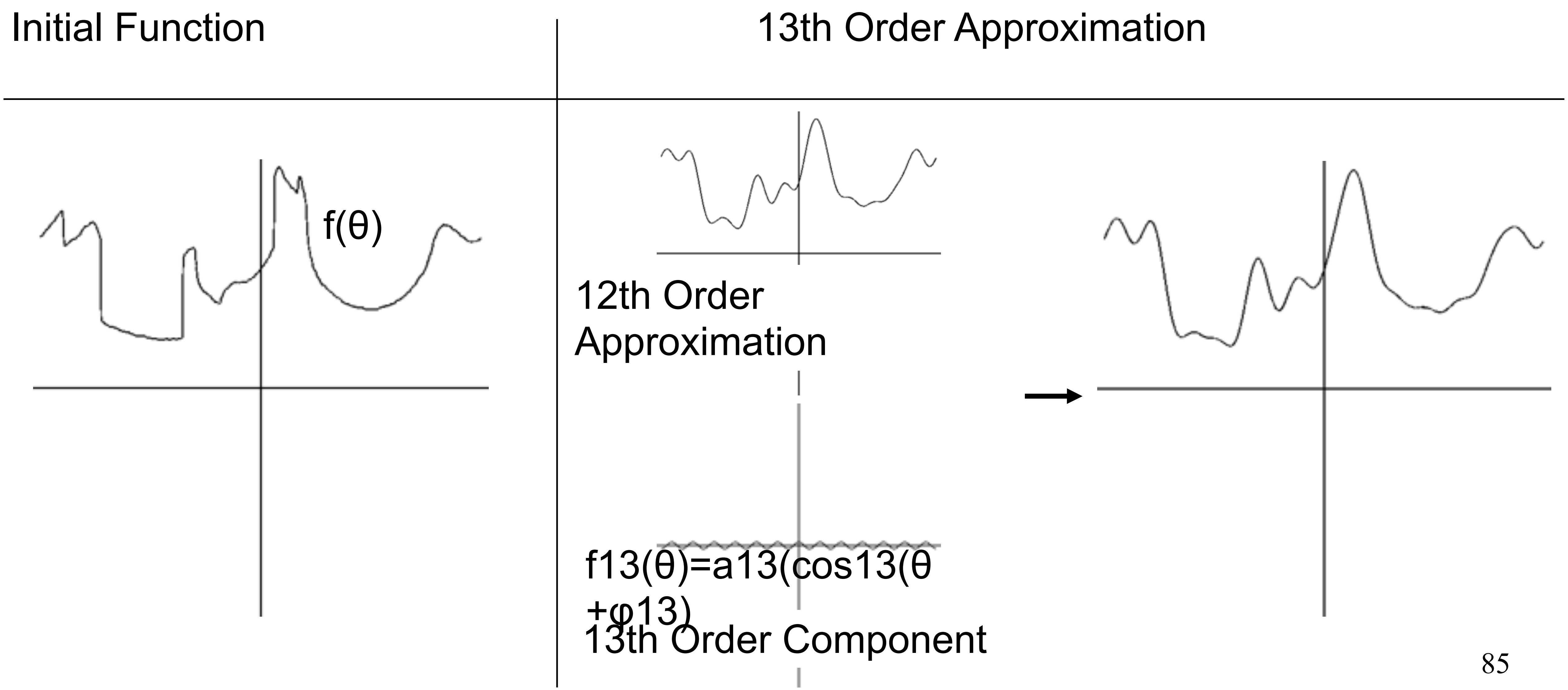
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



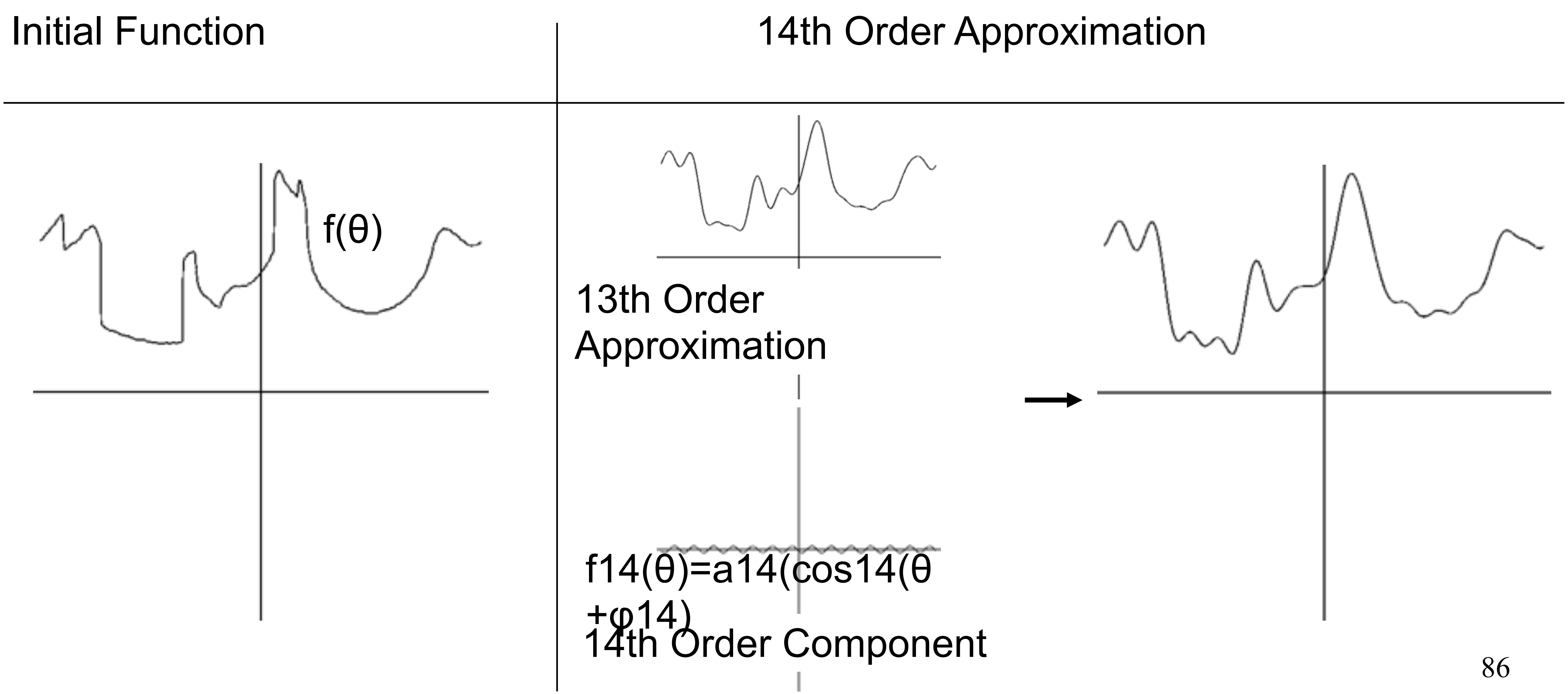
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



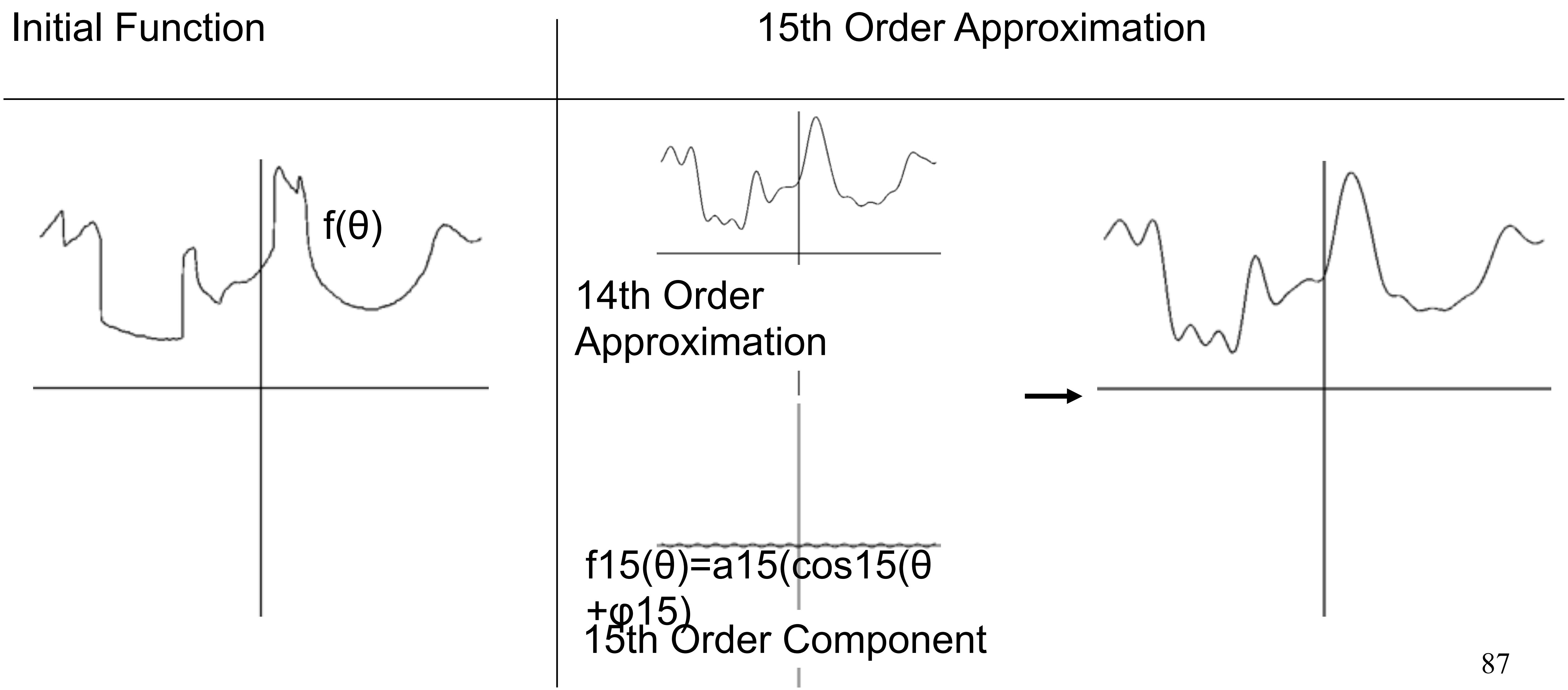
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



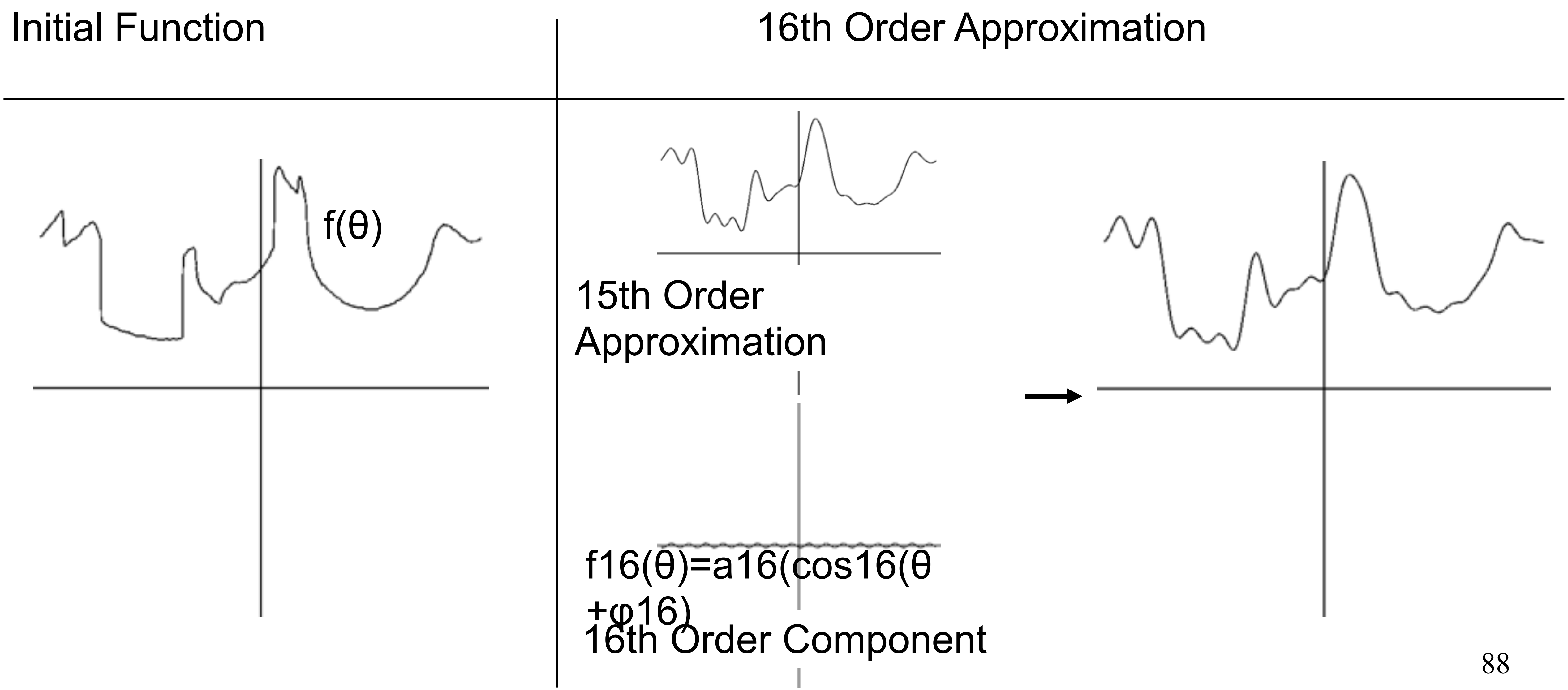
Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.



Fourier Analysis

- As higher frequency components are added to the approximation, finer details are captured.

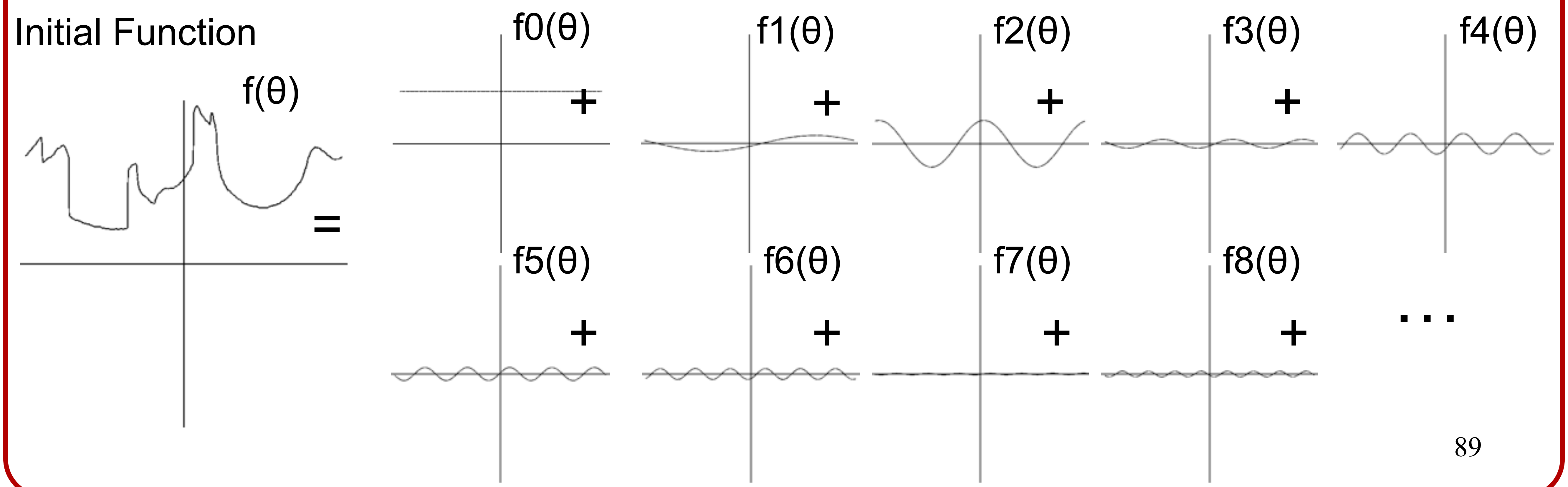


Fourier Analysis

- Combining all of the frequency components together, we get the initial function.

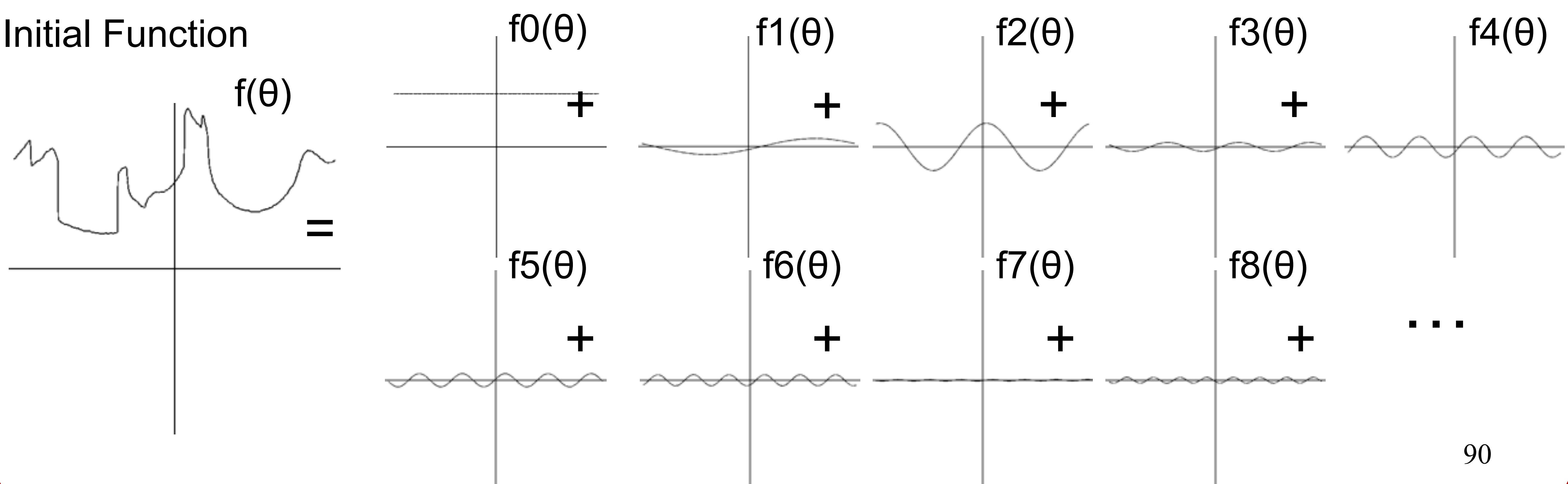
$$\boxed{f(\theta) = \sum_{k=0}^{\infty} f_k(\theta) = \sum_{k=0}^{\infty} a_k \cos(k(\theta + \phi_k))}$$

ak: amplitude of the kth frequency component
φk: shift of the kth frequency component



Question

- As higher frequency components are added to the approximation, finer details are captured.
- If we have n samples, what is the highest frequency that can be represented?



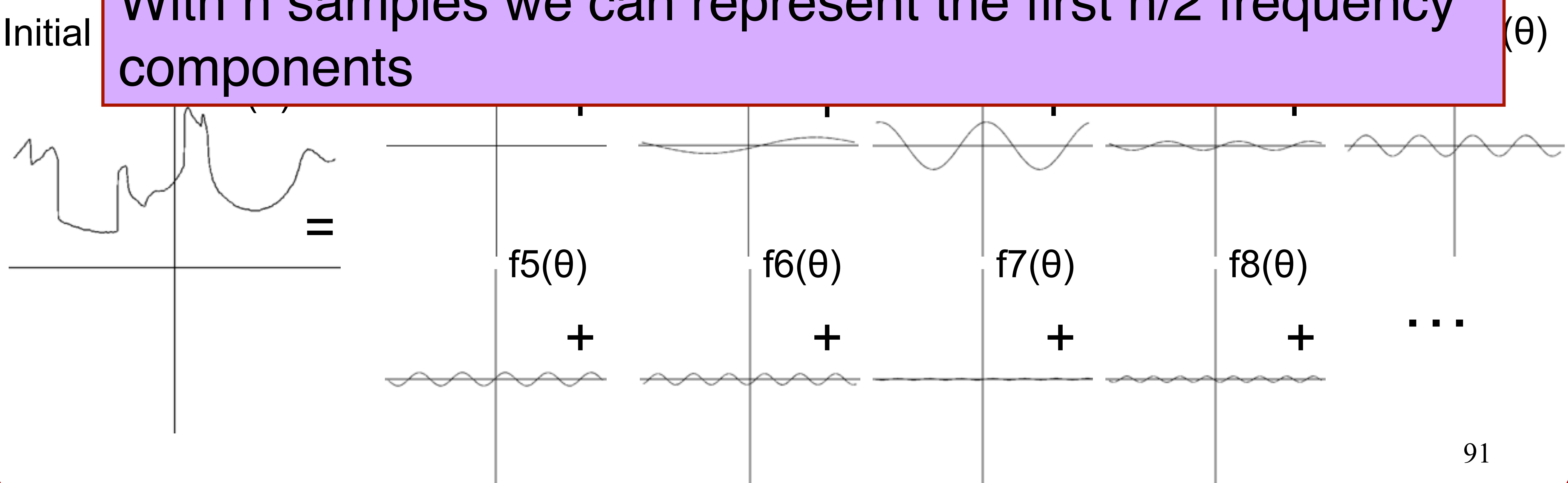
Question

- As higher frequency components are added to the approximation, finer details are captured.
- If we have n samples, what is the highest frequency that can be represented?

Each frequency component has two degrees of freedom:

- Amplitude
- Shift

With n samples we can represent the first n/2 frequency components



Sampling Theorem

- A signal can be reconstructed from its samples, if the original signal has no frequencies above 1/2 the sampling frequency – Shannon's Theorem
- The minimum sampling rate for band-limited function is called the "Nyquist rate"

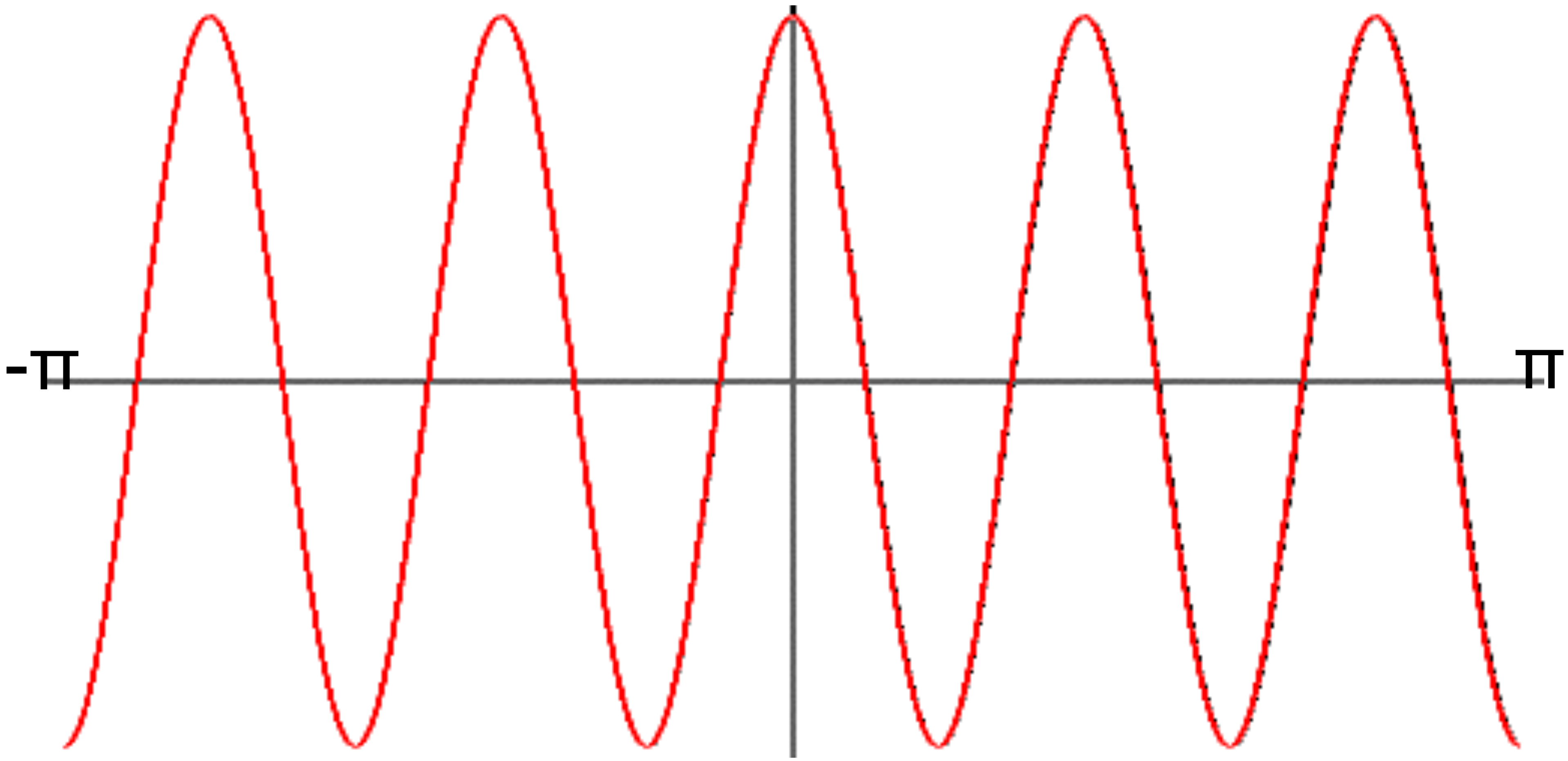
A signal is band-limited if its highest non-zero frequency is bounded.
The frequency is called the bandwidth.

Question

- What if we have only n samples and we try to reconstruct a function with frequencies larger than the Nyquist frequency (n/2)?

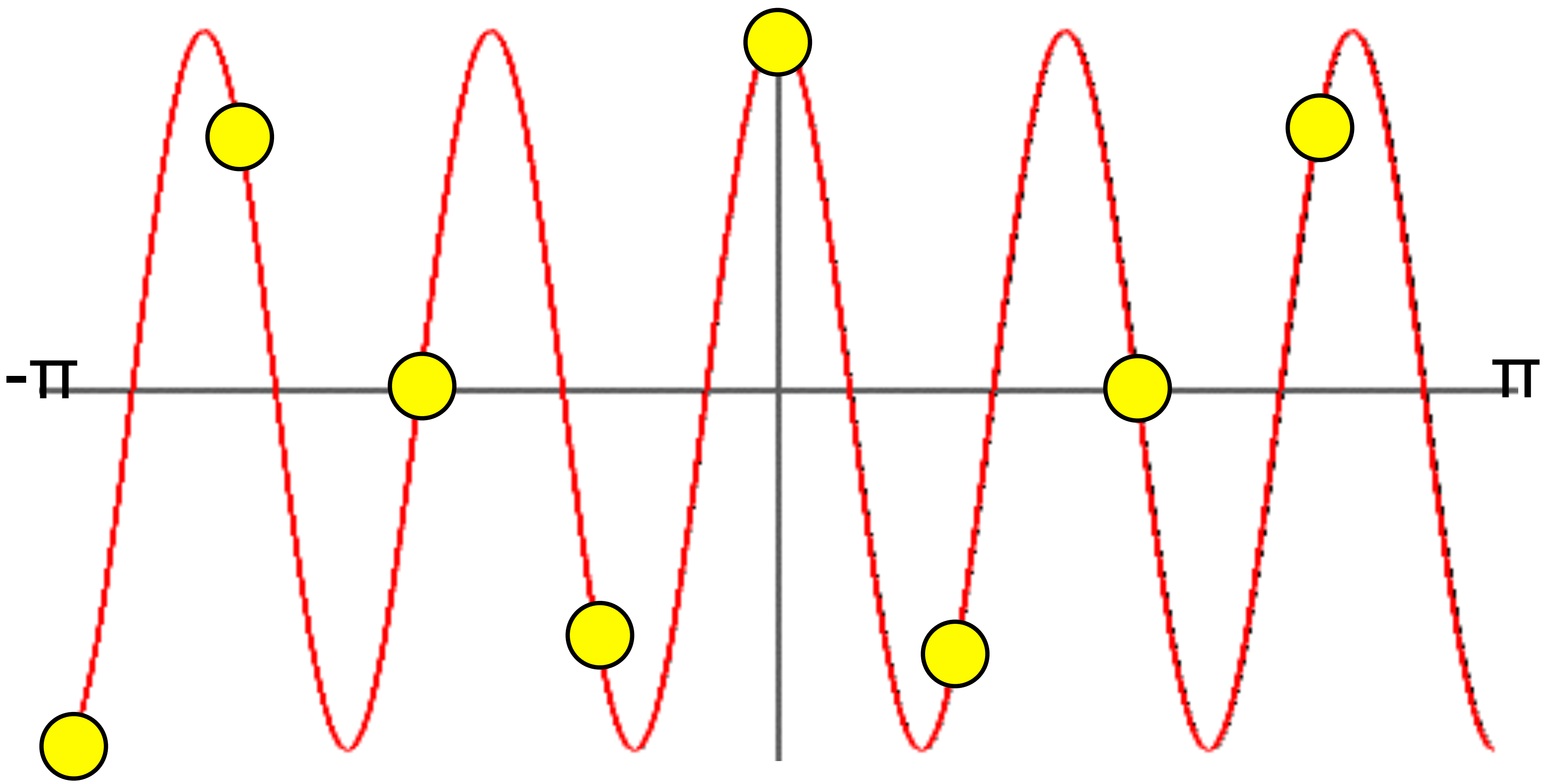
Aliasing

- When a high-frequency signal is sampled with insufficiently many samples, it will be perceived as a lower-frequency signal. This masking of higher frequencies as lower ones is referred to as aliasing.



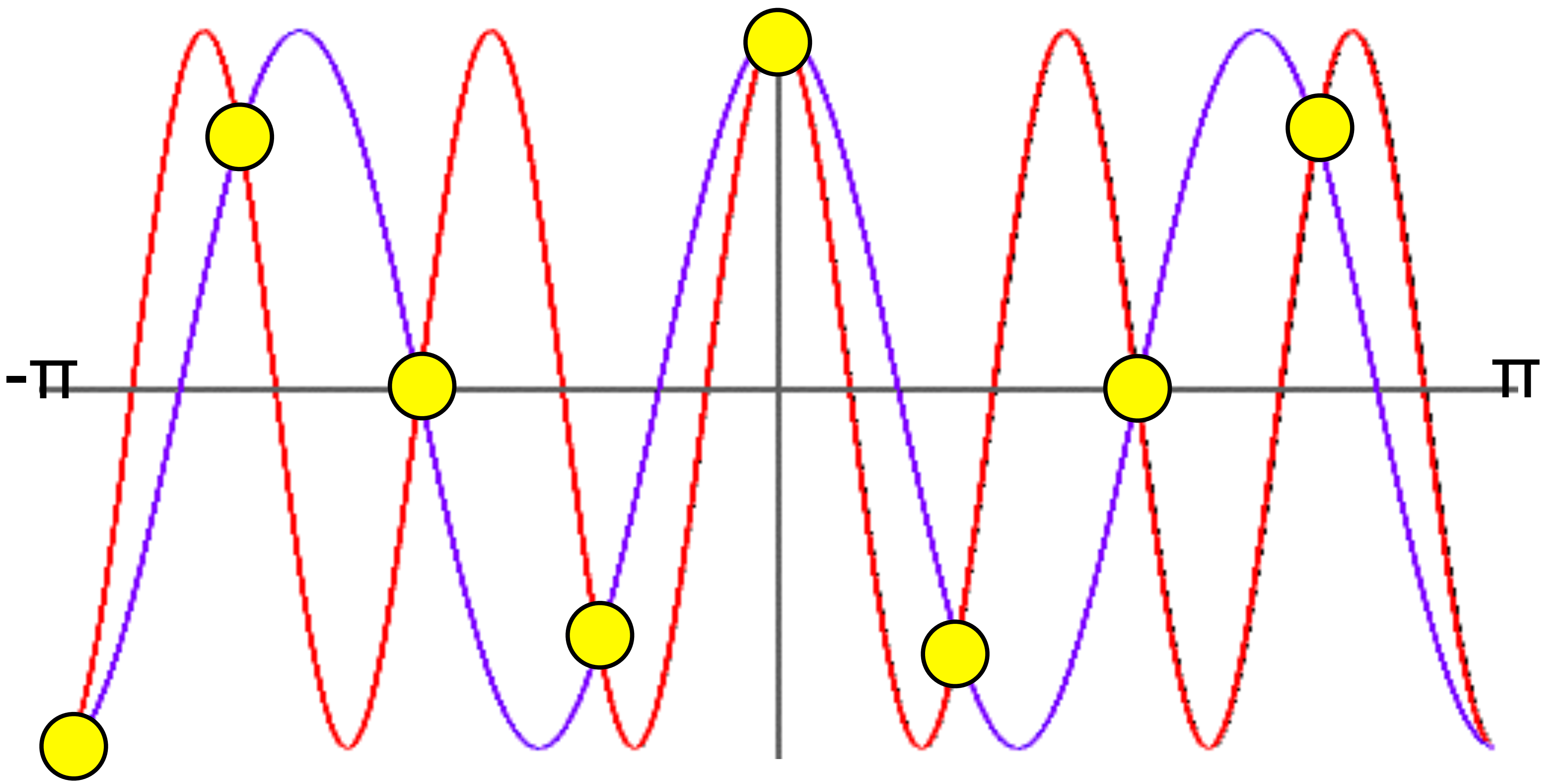
Aliasing

- When a high-frequency signal is sampled with insufficiently many samples, it will be perceived as a lower-frequency signal. This masking of higher frequencies as lower ones is referred to as aliasing.



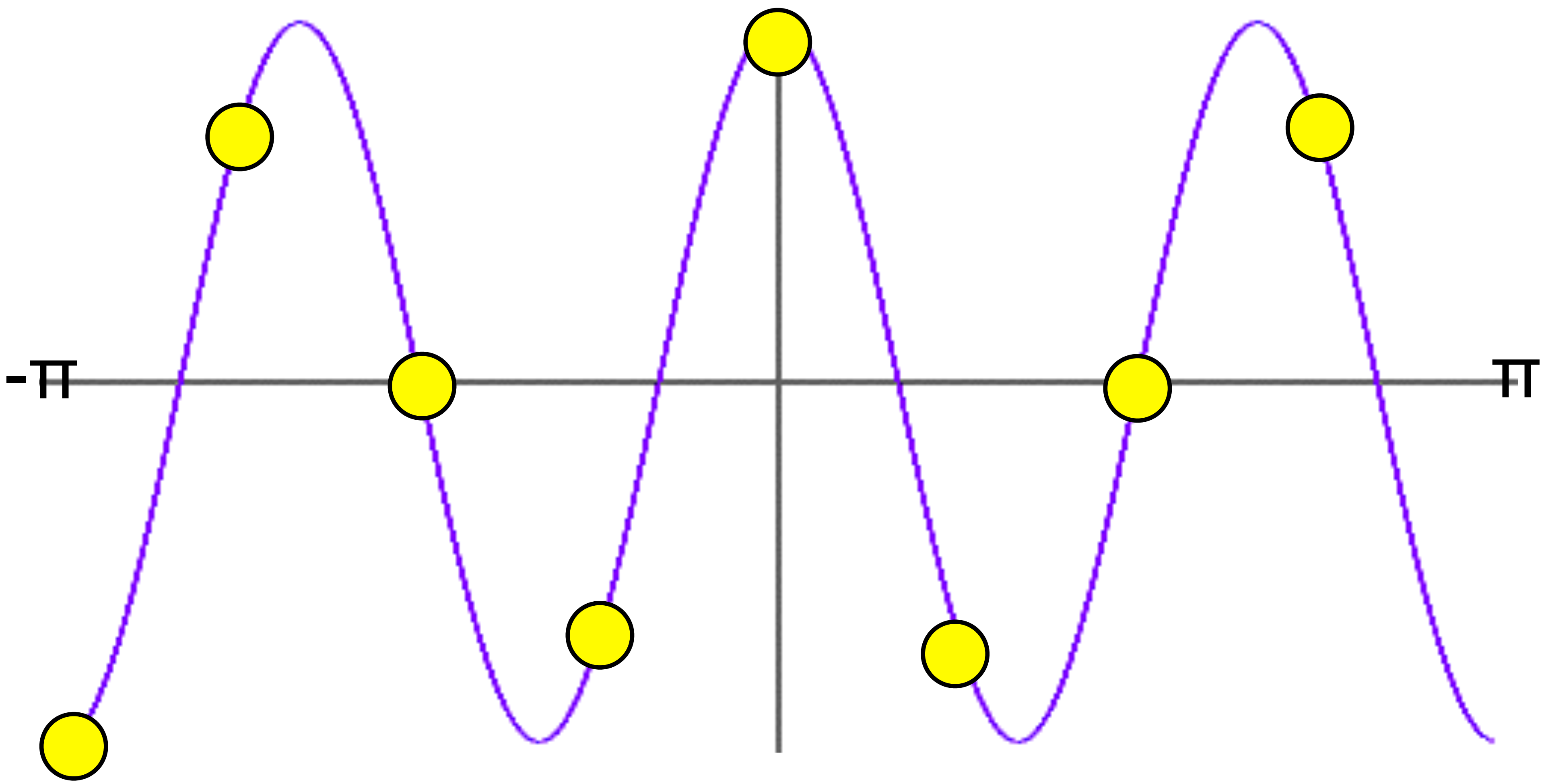
Aliasing

- When a high-frequency signal is sampled with insufficiently many samples, it will be perceived as a lower-frequency signal. This masking of higher frequencies as lower ones is referred to as aliasing.



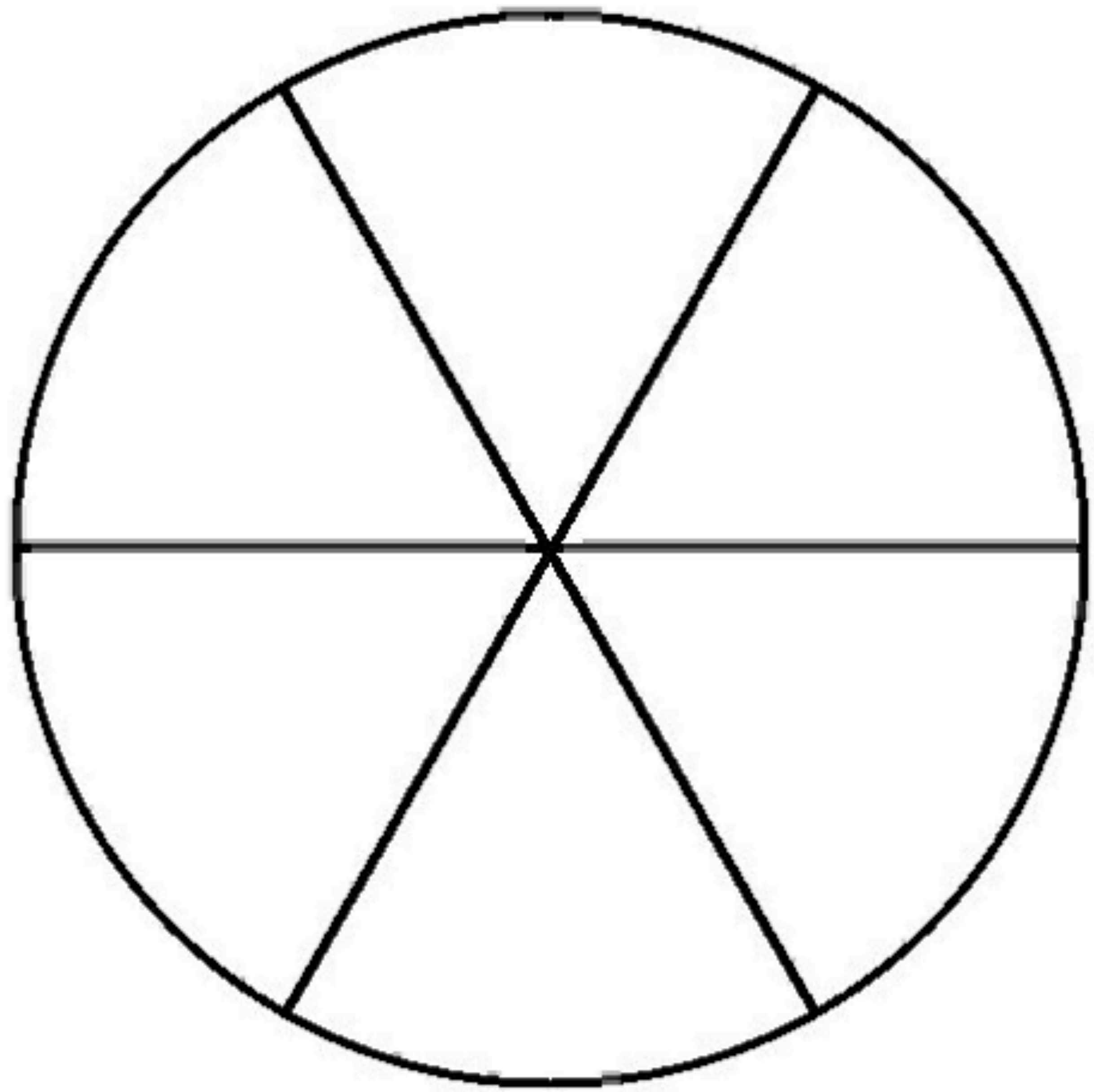
Aliasing

- When a high-frequency signal is sampled with insufficiently many samples, it will be perceived as a lower-frequency signal. This masking of higher frequencies as lower ones is referred to as aliasing.



Temporal Aliasing

- Artifacts due to limited temporal resolution



Nearest
Neighbor

Sampling

- There are two problems:
  - You don’t have enough samples to correctly reconstruct your high-frequency information
  - You corrupt the low-frequency information because the high-frequencies mask themselves as lower ones.

Anti-Aliasing

Two possible ways to address aliasing:

- Sample at higher rate
- Pre-filter to form band-limited signal

Anti-Aliasing

Two possible ways to address aliasing:

- Sample at higher rate
  - Not always possible
  - Still rendering to fixed resolution
- Pre-filter to form band-limited signal

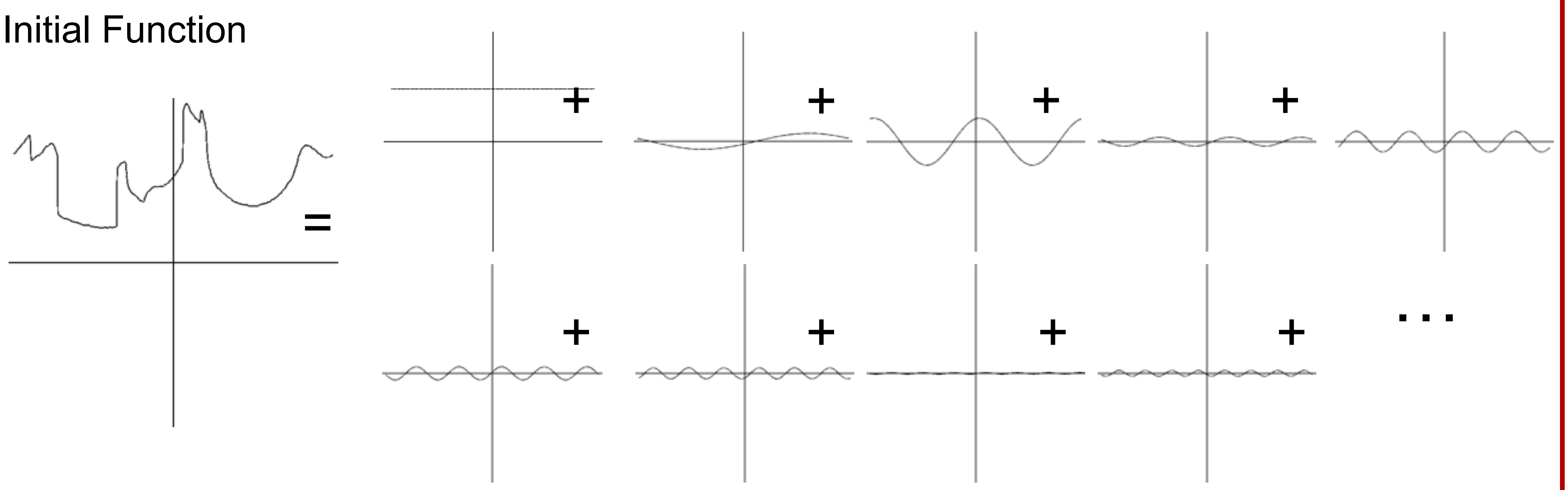
Anti-Aliasing

Two possible ways to address aliasing:

- Sample at higher rate
- Pre-filter to form a band-limited signal
  - You still don't get your high frequencies, but at least the low frequencies are uncorrupted.

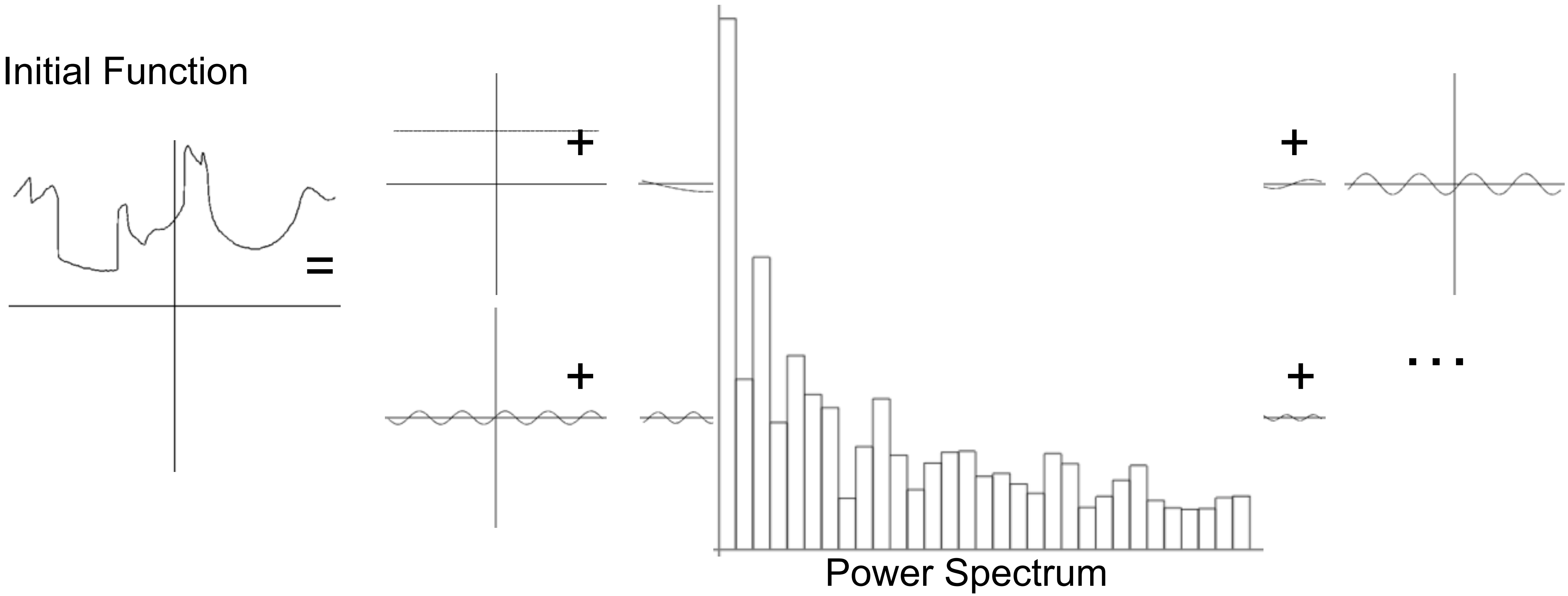
Fourier Analysis

- If we just look at how much information each frequency contributes, we obtain the power spectrum of the signal:



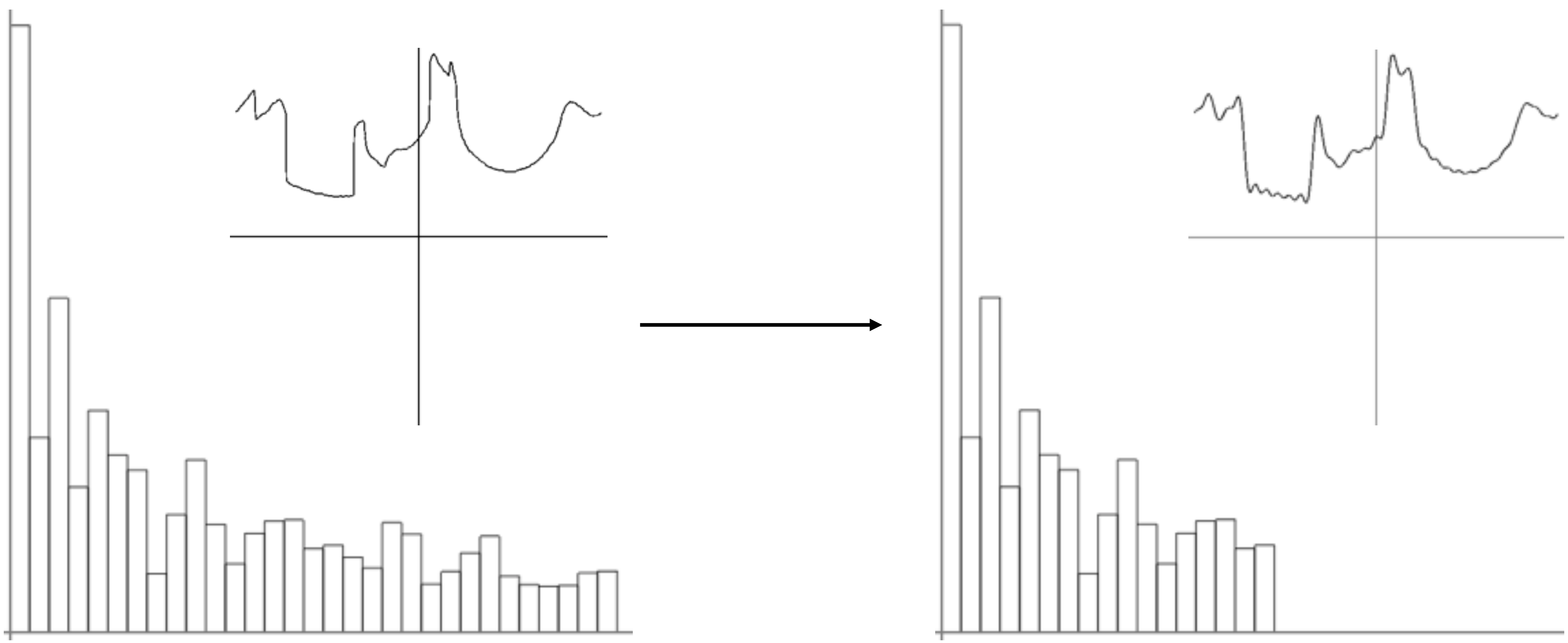
Fourier Analysis

- If we just look at how much information each frequency contributes, we obtain the power spectrum of the signal:



Pre-Filtering

- Band-limit by discarding the high-frequency components of the Frequency decomposition.

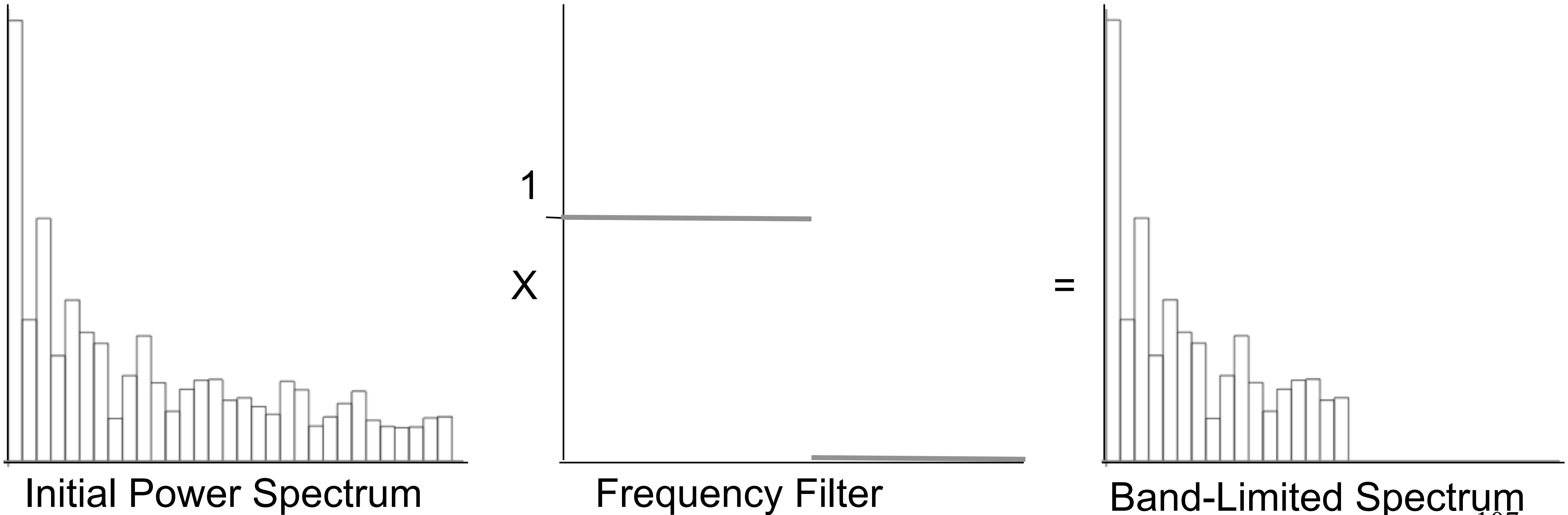


Initial Power Spectrum

Band-Limited Power Spectrum

Pre-Filtering

- Band-limit by discarding the high-frequency components of the Fourier decomposition.
- We can do this by multiplying the frequency components by a 0/1 function:



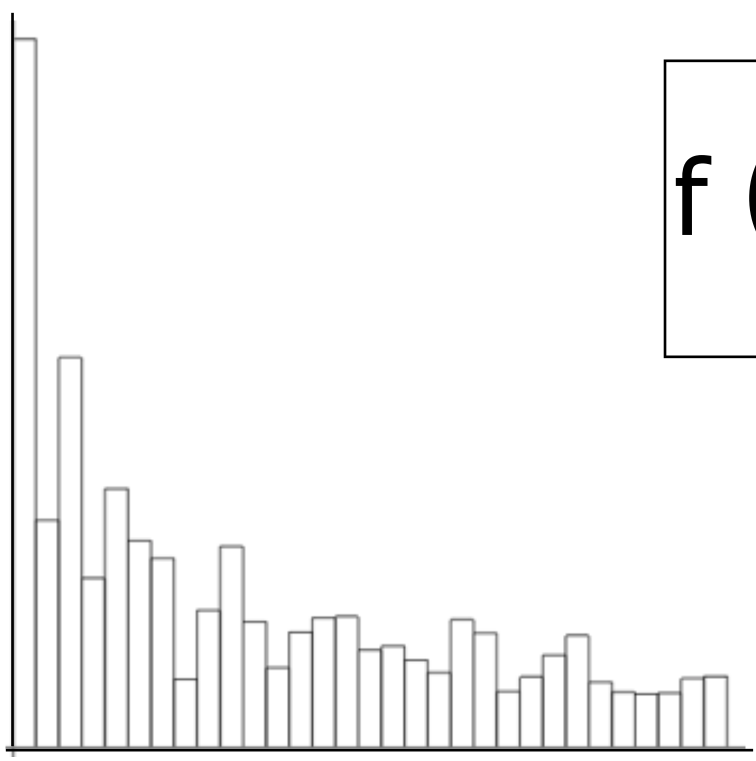
Pre-Filtering

- Band-limit by discarding the high-frequency components of the Fourier decomposition.
- We can do this by multiplying the frequency components by a 0/1 function:

$$f(\theta) = \sum_{k=0}^{\infty} a_k \cos\left(k\left(\theta + \phi_k\right)\right)$$

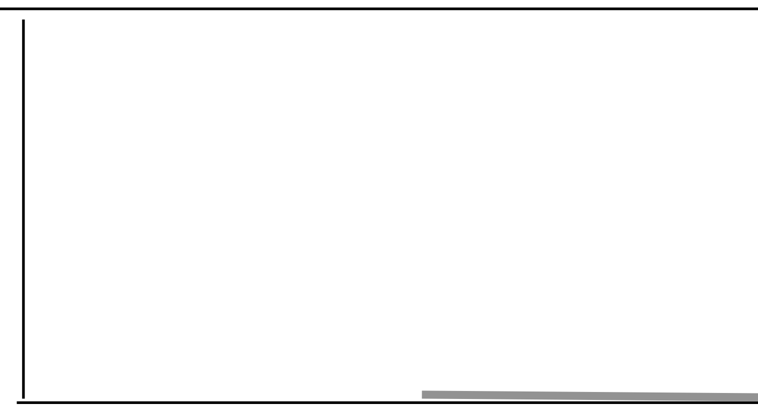
$$\Downarrow$$

$$f(\theta) = \sum_{k=0}^{n/2} a_k \cos\left(k\left(\theta + \phi_k\right)\right)$$



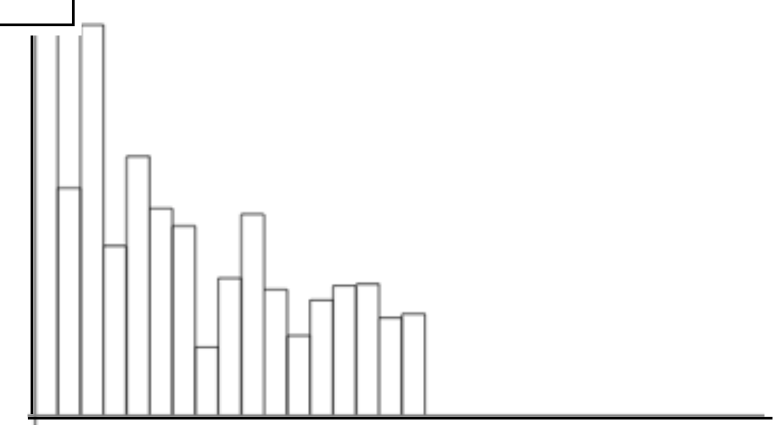
Initial Power Spectrum

X



Frequency Filter

=



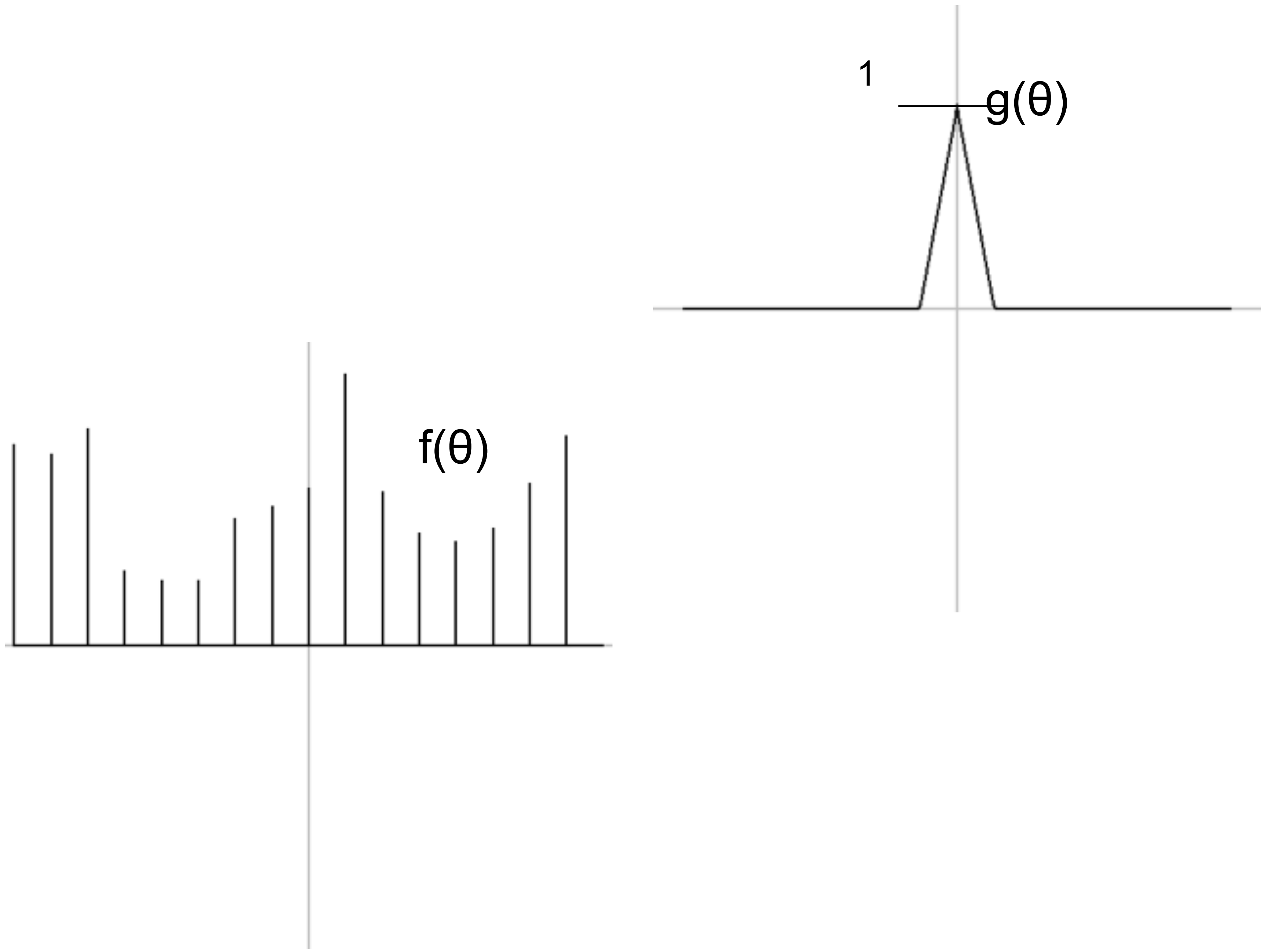
Band-Limited Spectrum

Fourier Theory

- A fundamental fact from Fourier theory is that multiplication in the frequency domain is equivalent to convolution in the spatial domain.

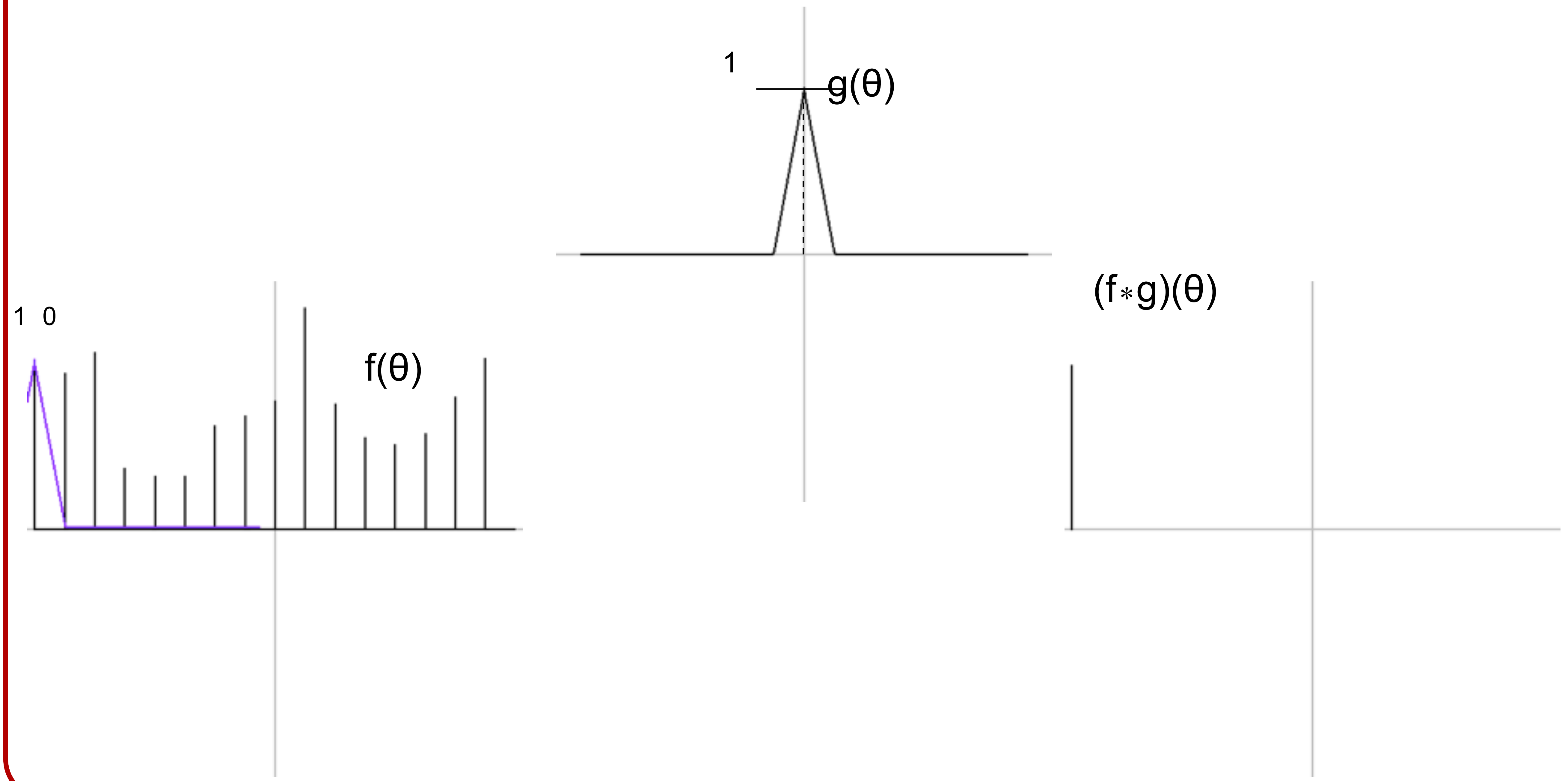
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.



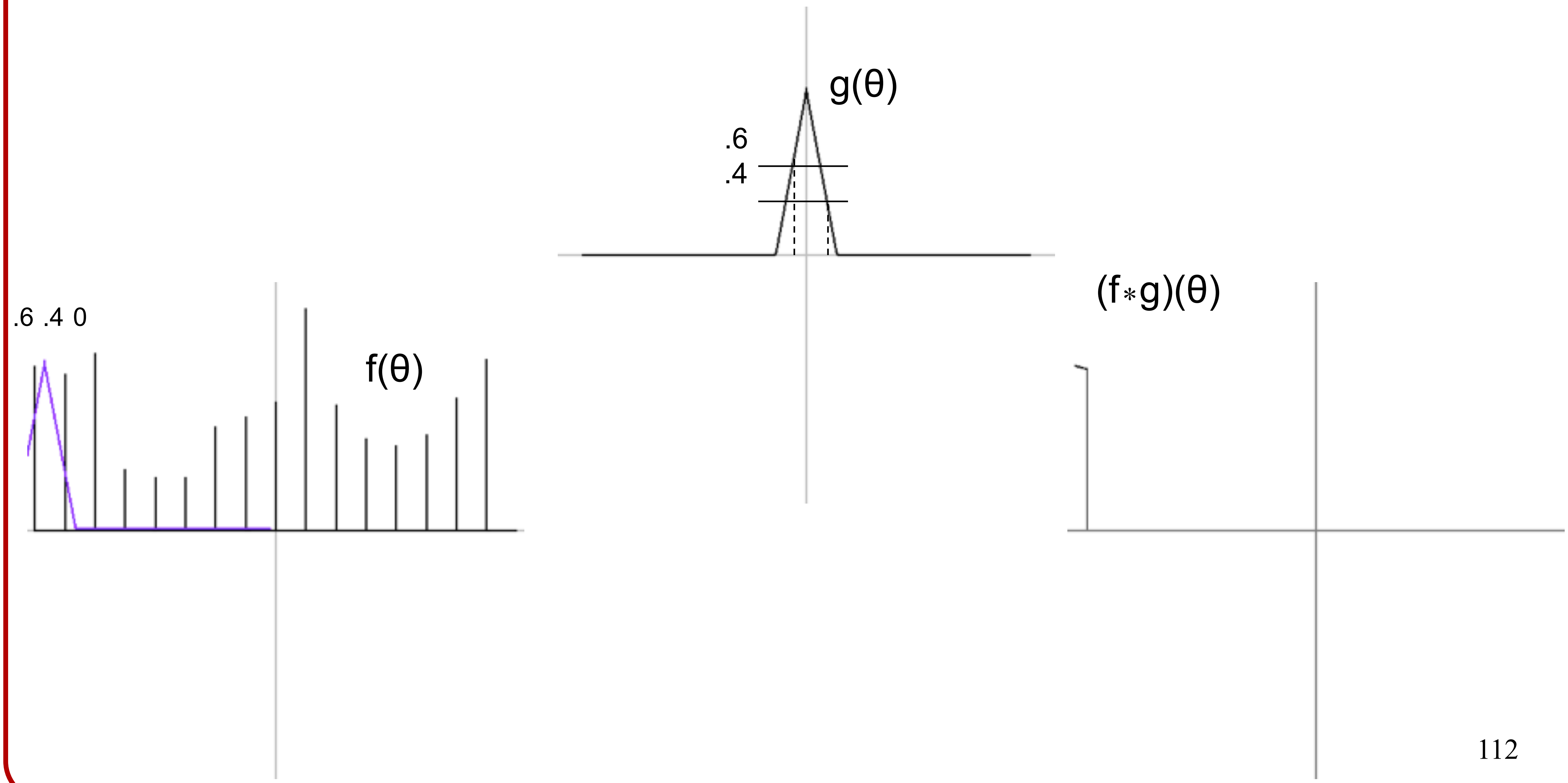
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.



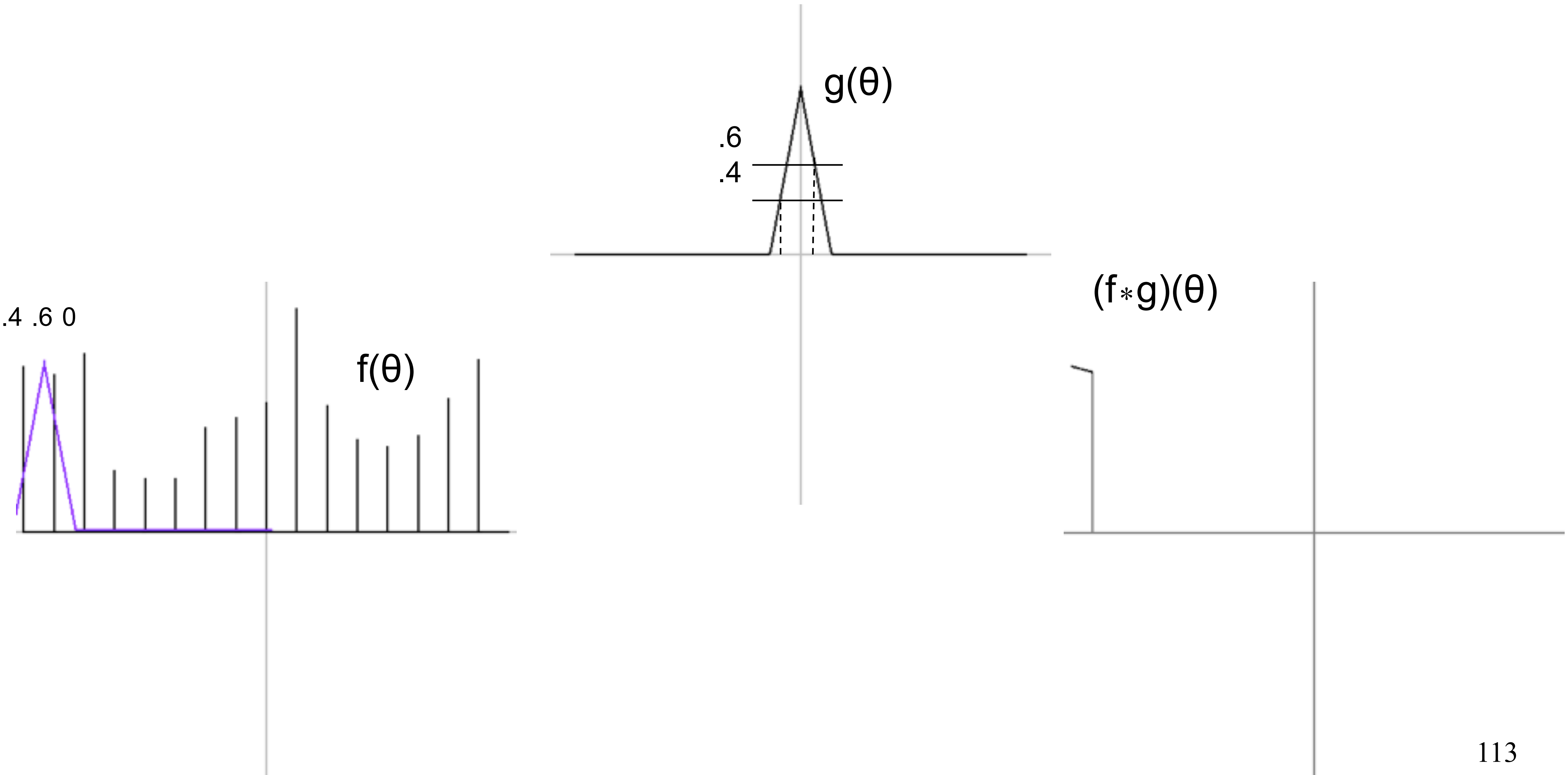
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.



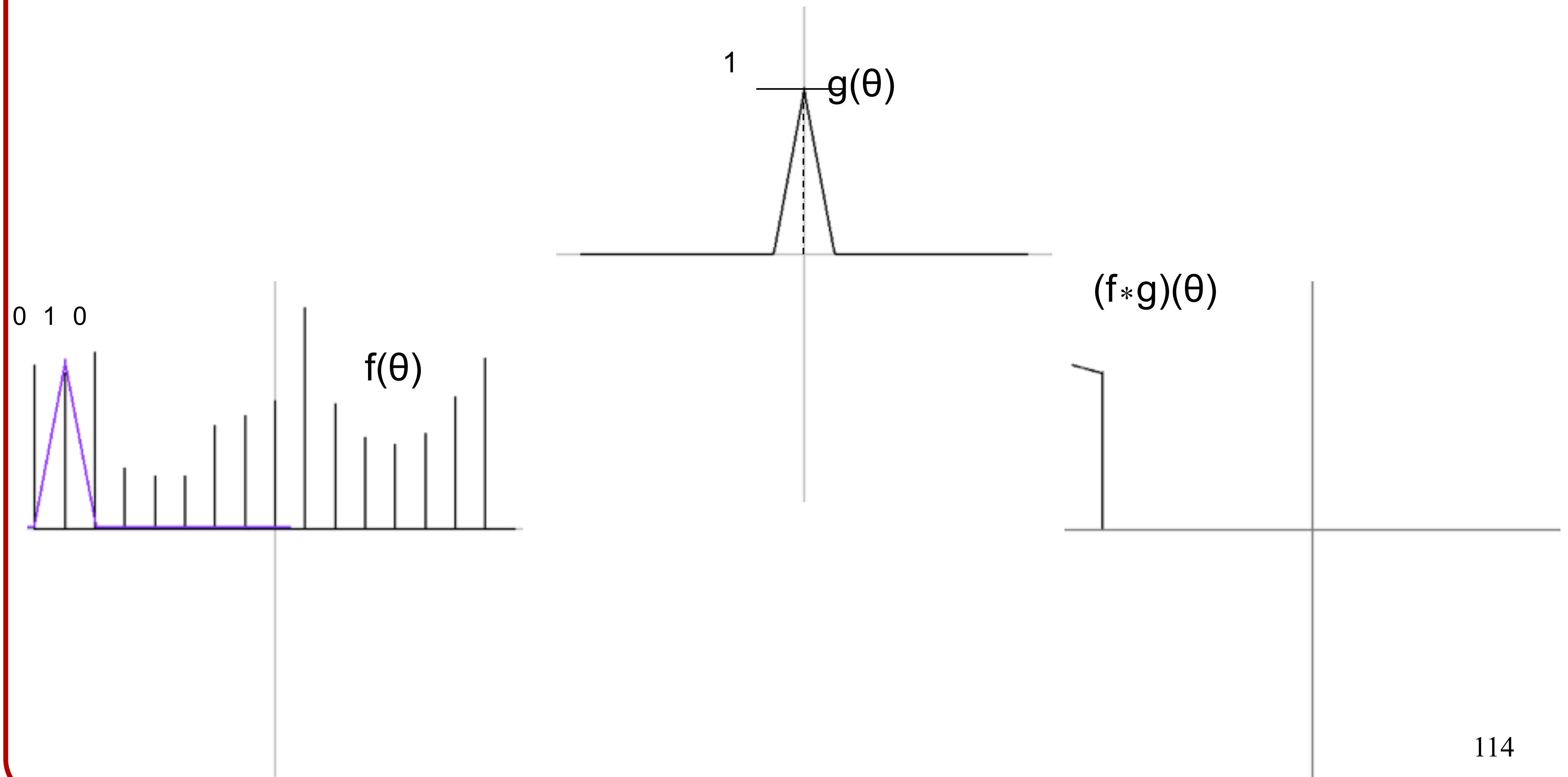
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.



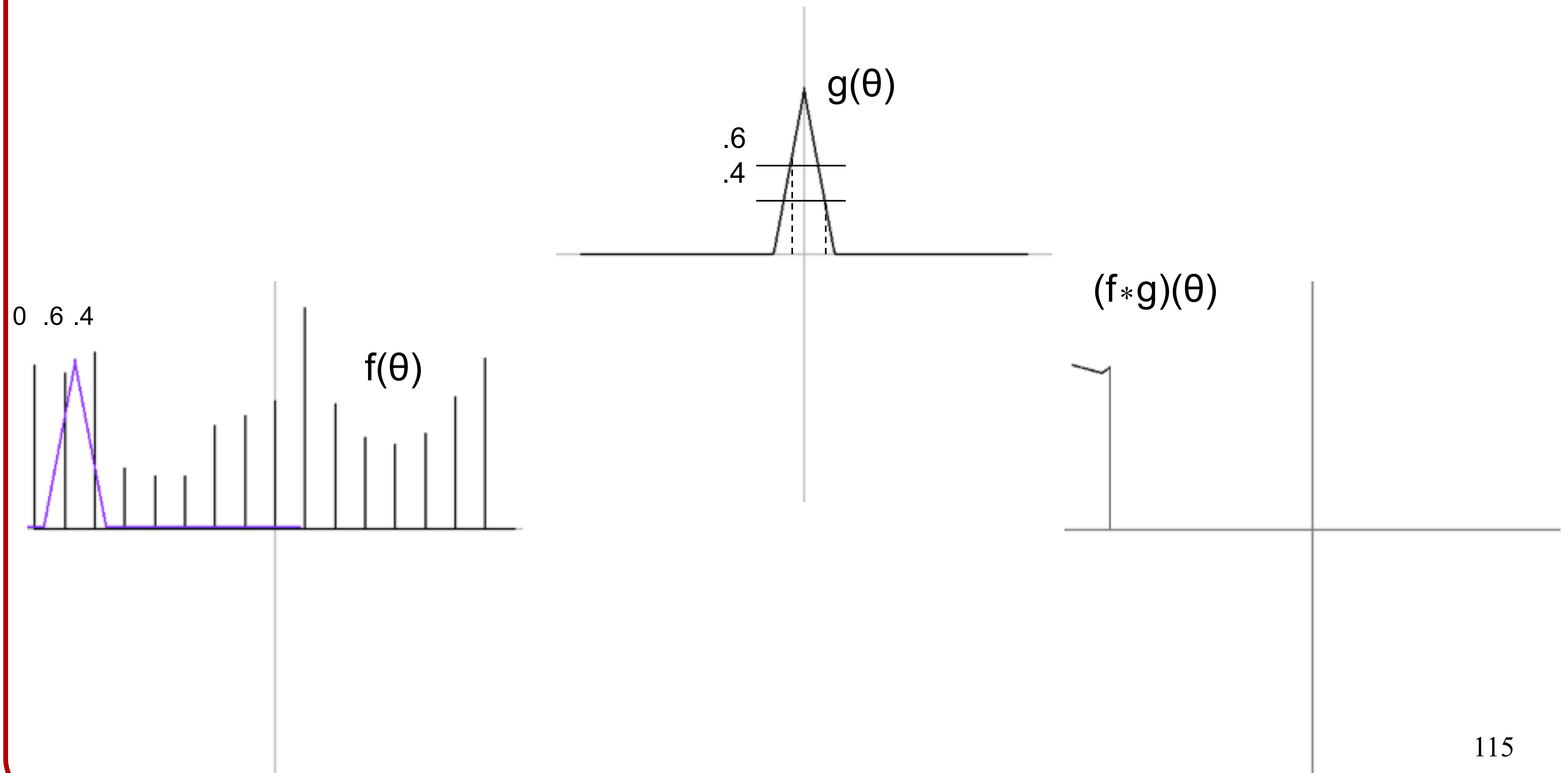
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.



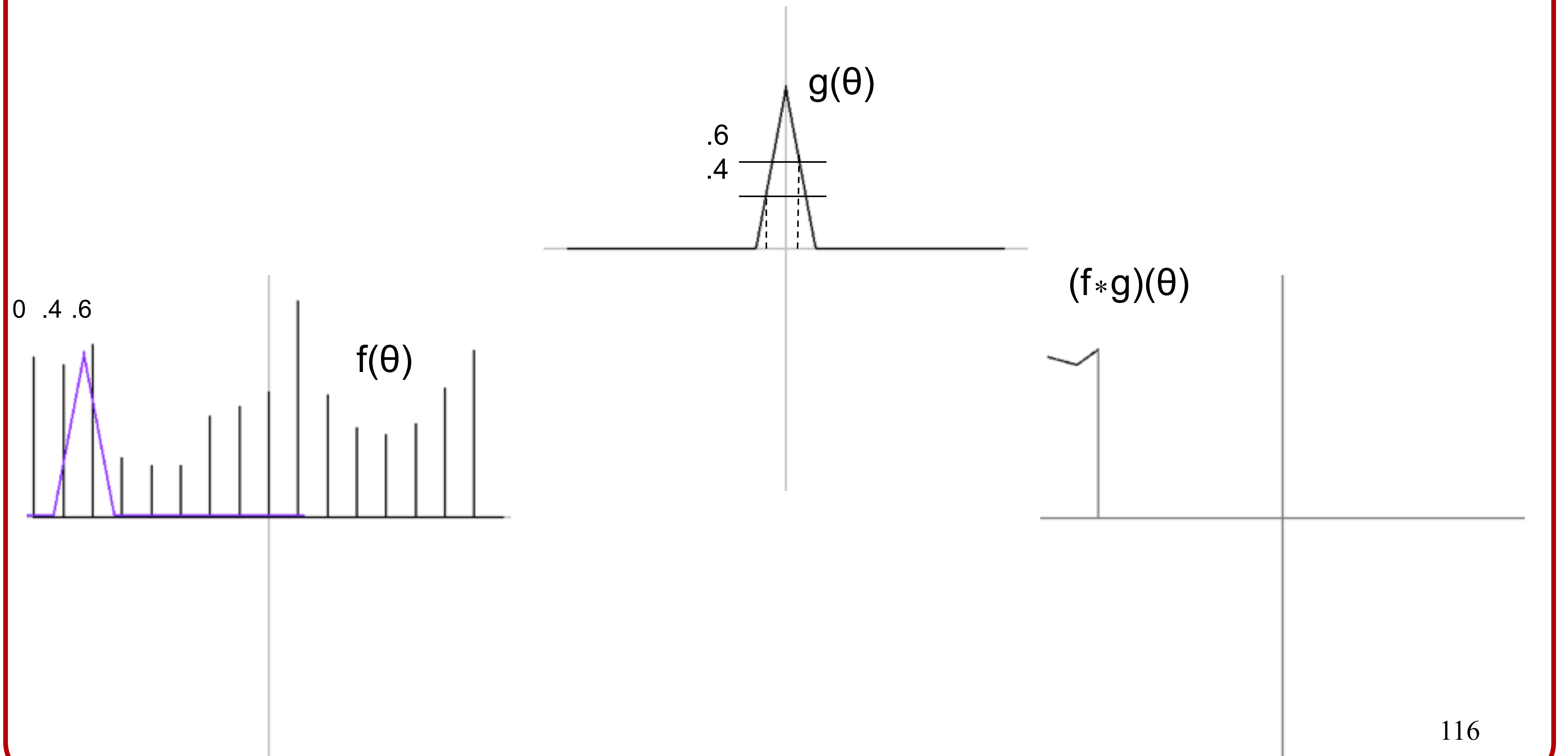
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.



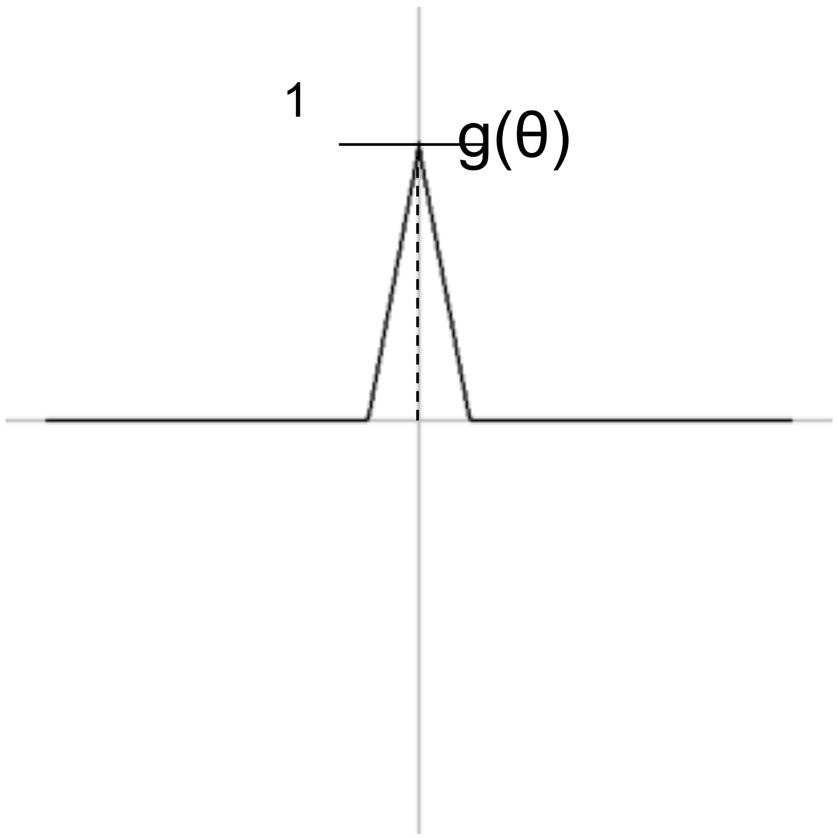
Convolution

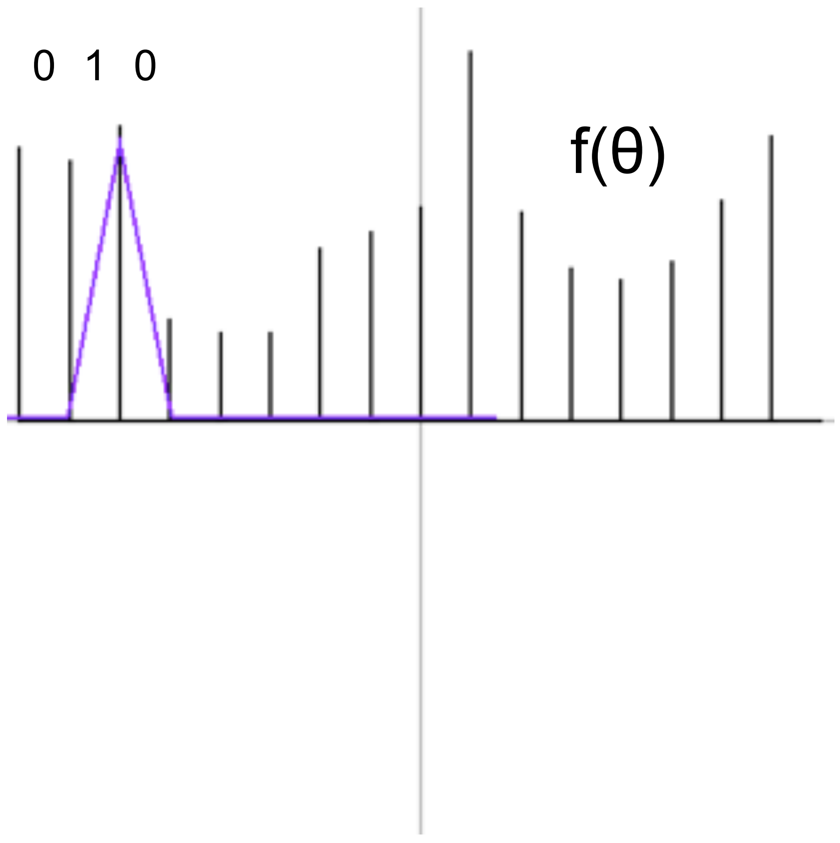
- To convolve two functions f and g, we resample the function f using the weights given by g.

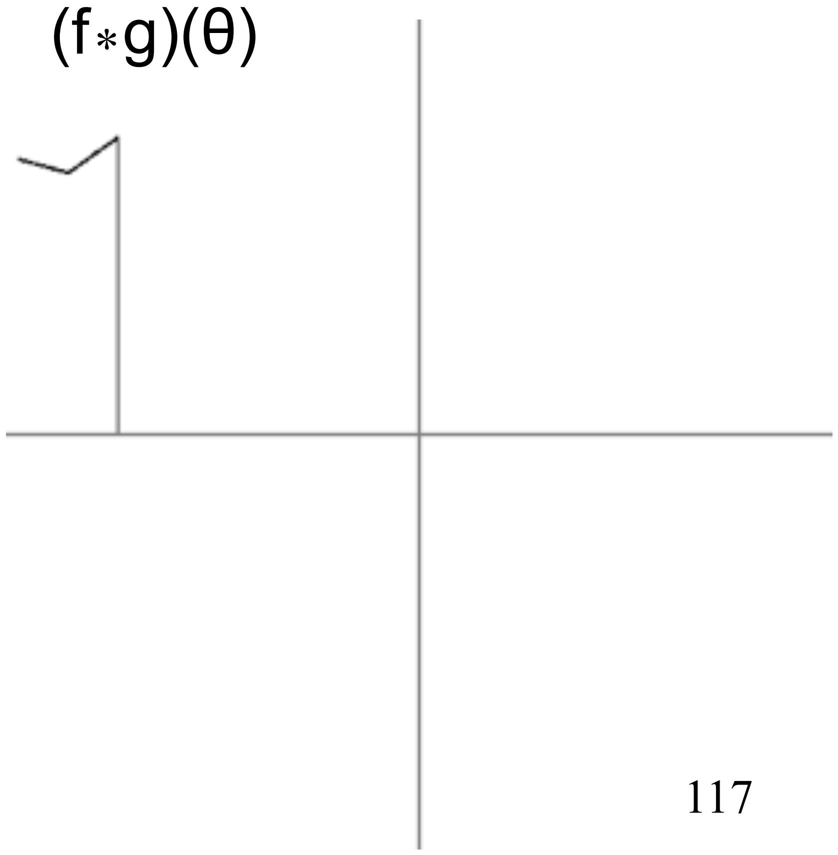


Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.

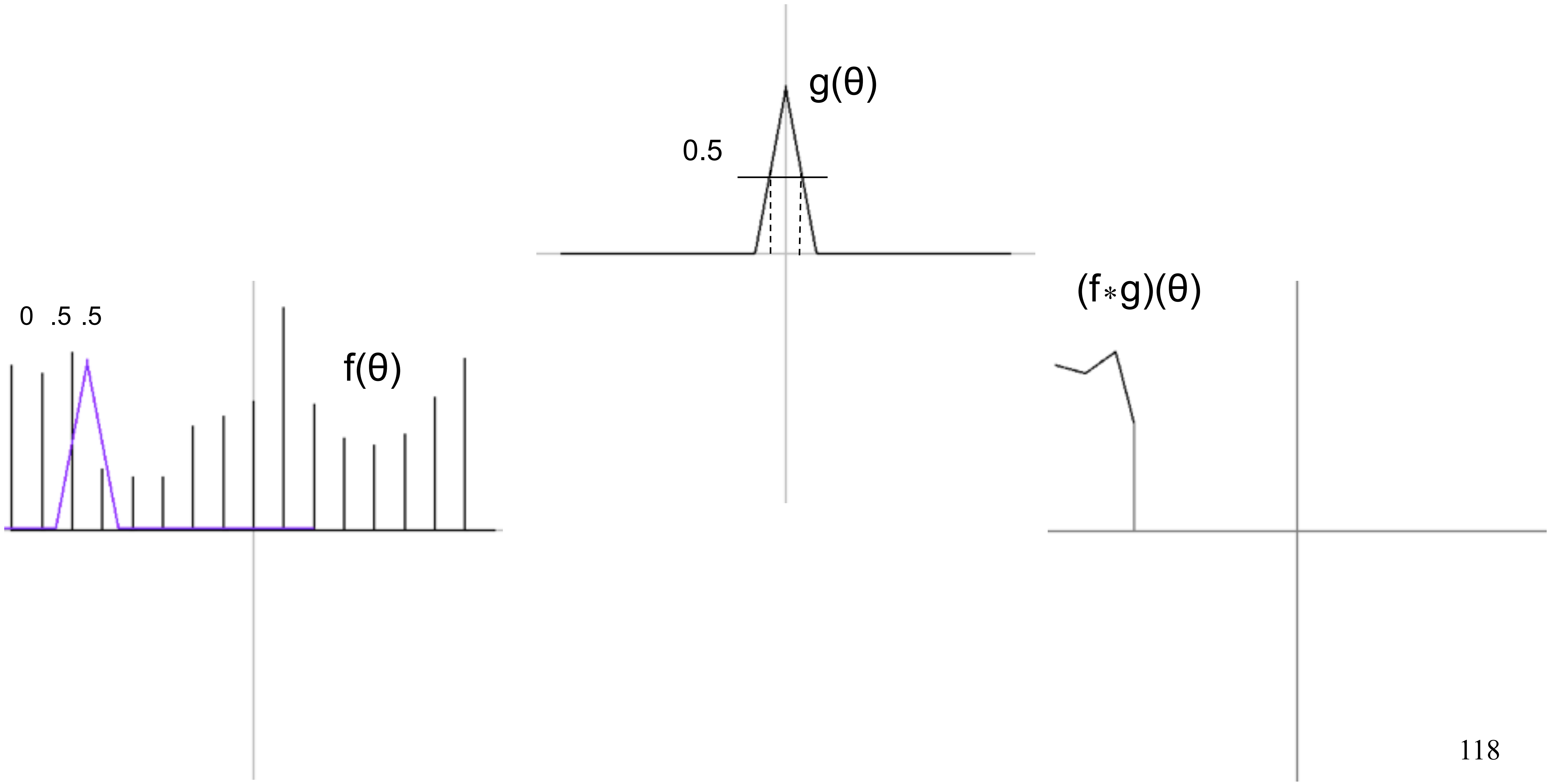






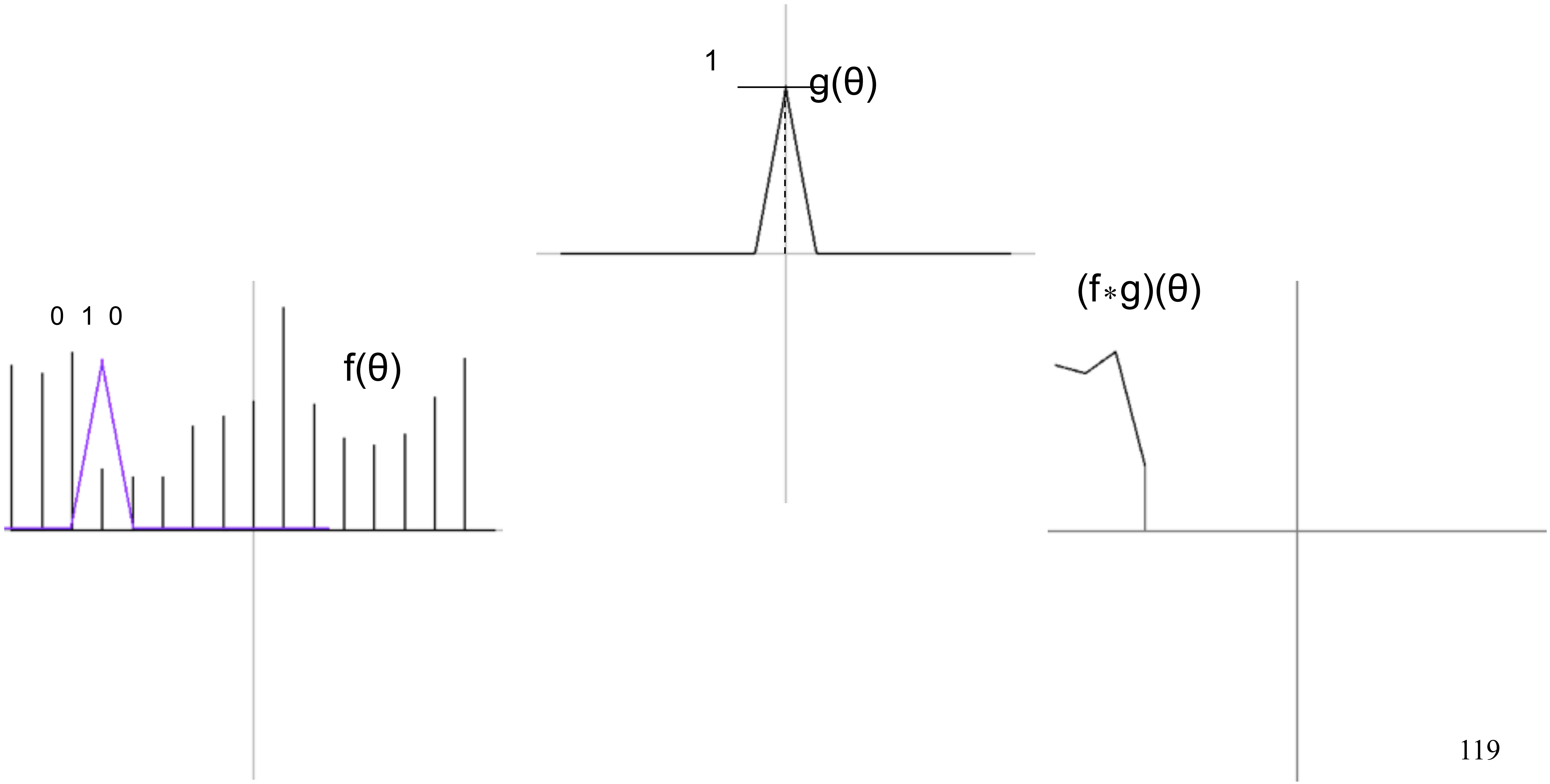
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.



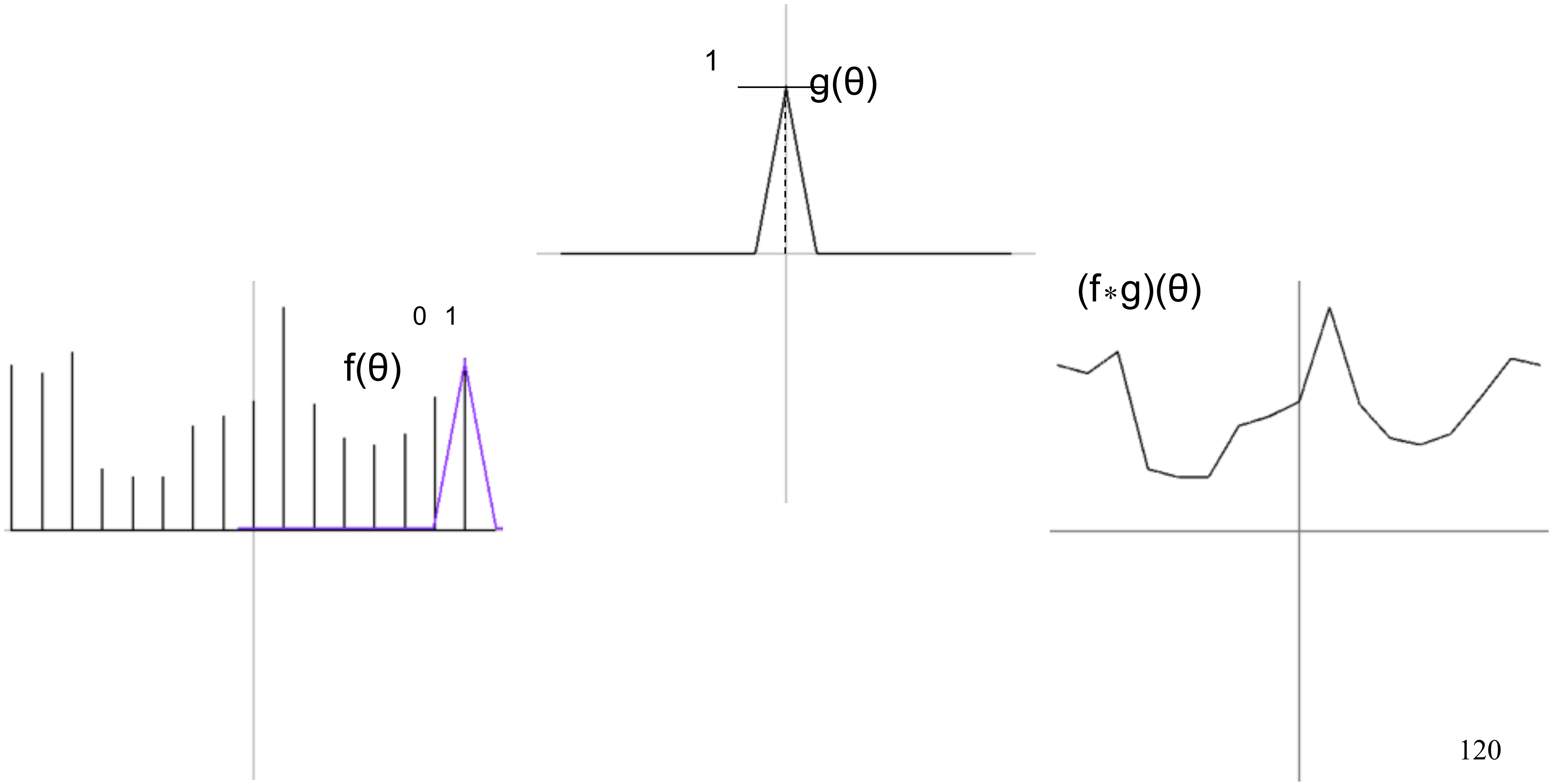
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.



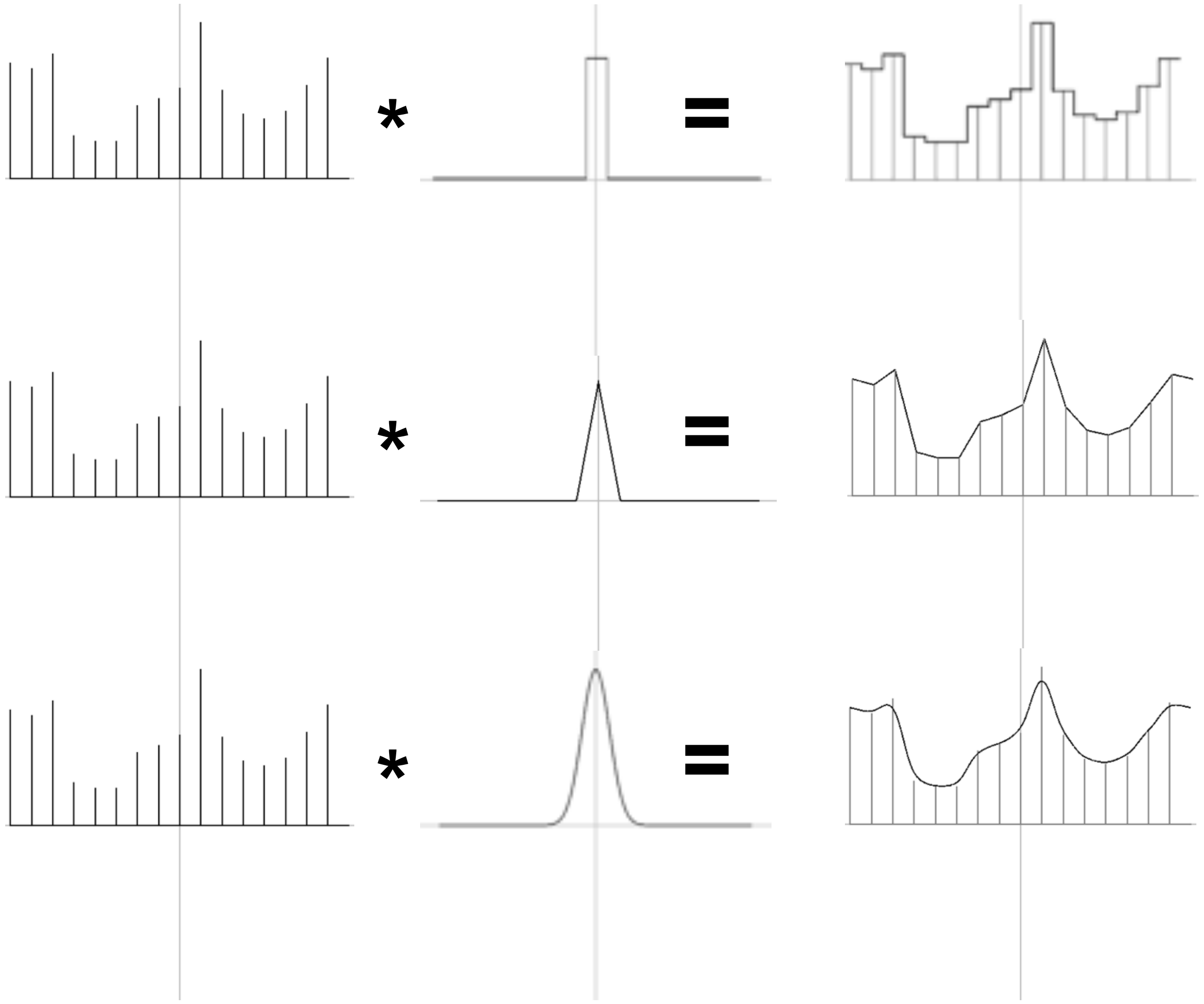
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.



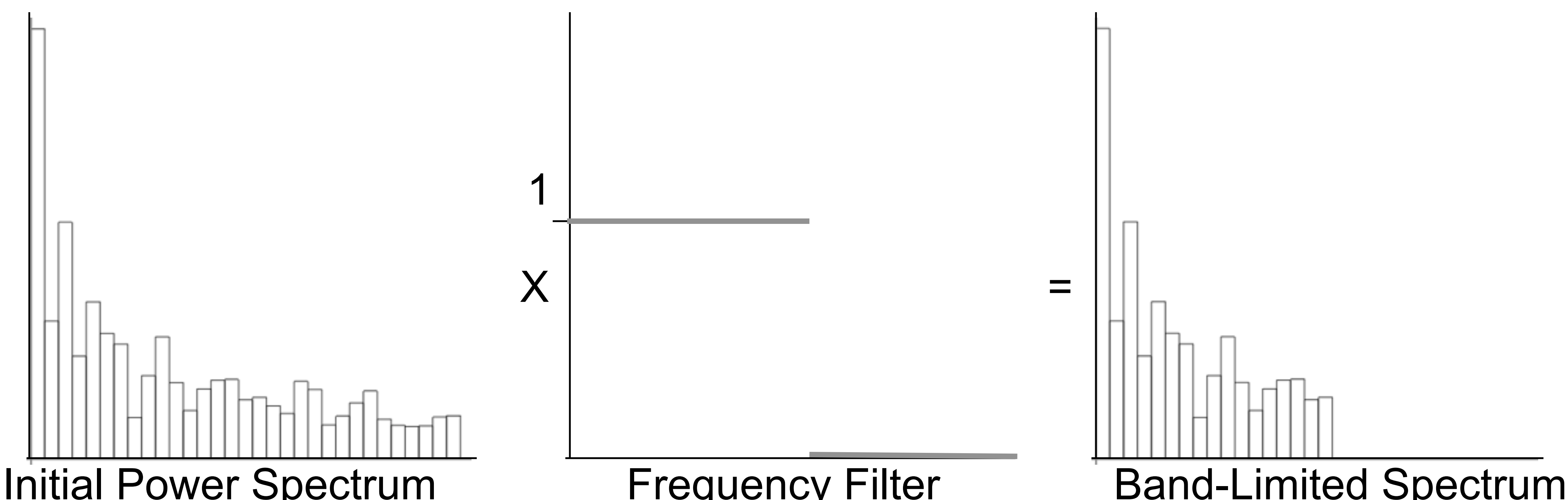
Convolution

- To convolve two functions f and g, we resample the function f using the weights given by g.
- Nearest point, bilinear, and Gaussian interpolation are just convolutions with different filters.

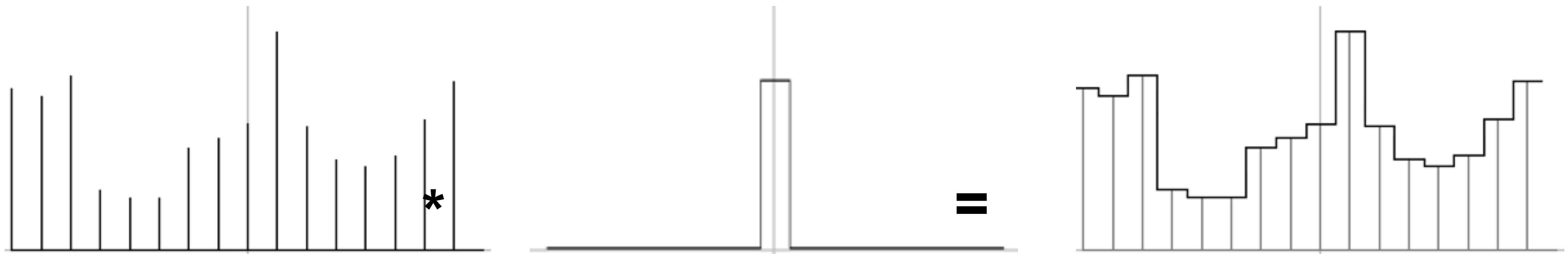


Convolution

- Recall that convolution in the spatial domain is the equal to multiplication in the frequency domain.
- In order to avoid aliasing, we need to convolve with a filter whose power spectrum has value:
  - 1 at low frequencies
  - 0 at high frequencies



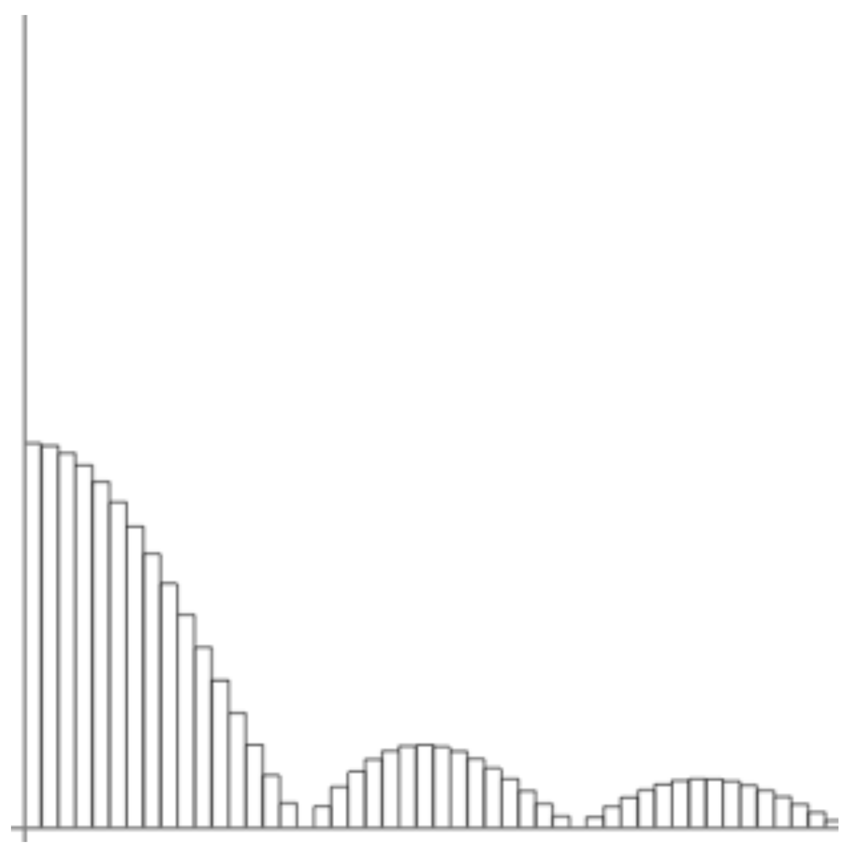
Nearest Point Convolution



Discrete Samples

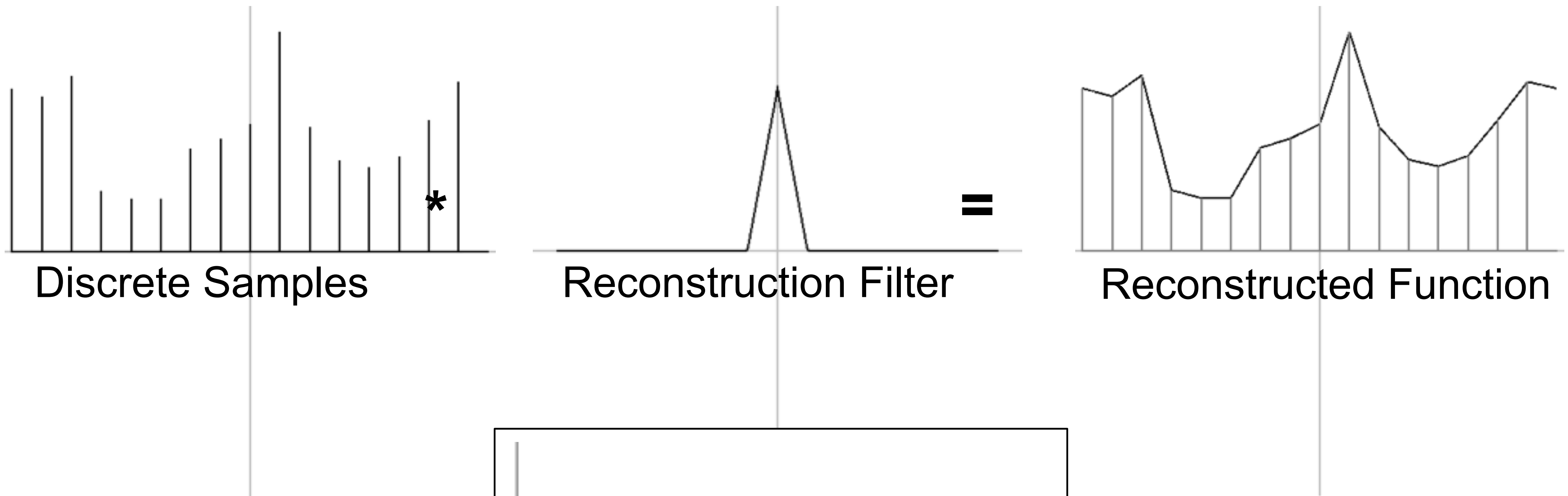
Reconstruction Filter

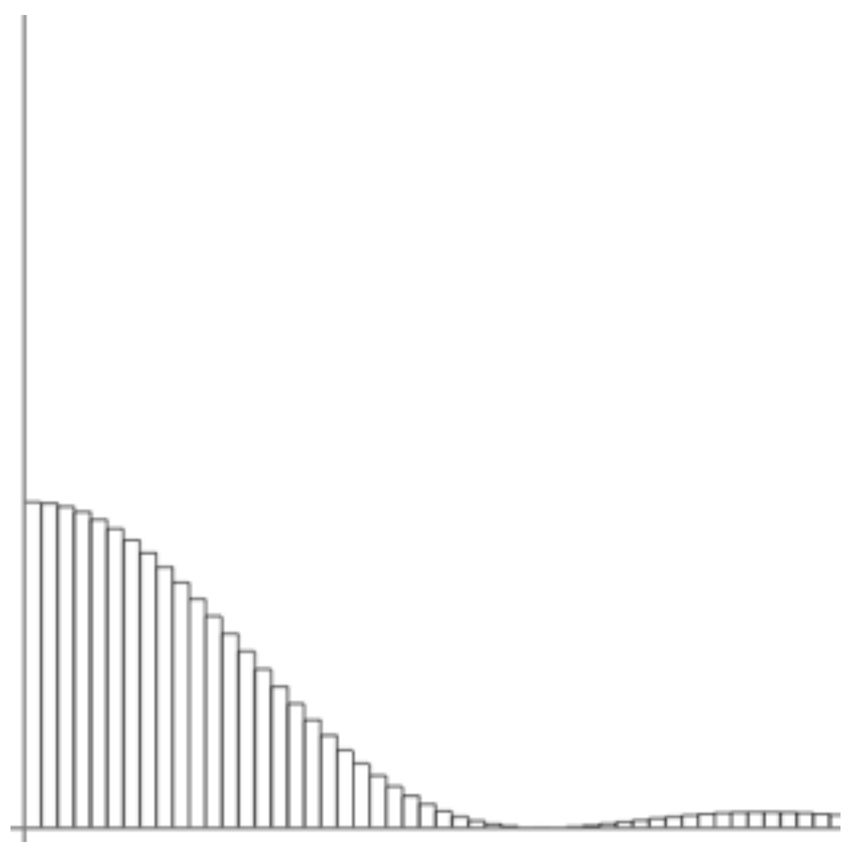
Reconstructed Function



Filter Spectrum

Bilinear Convolution

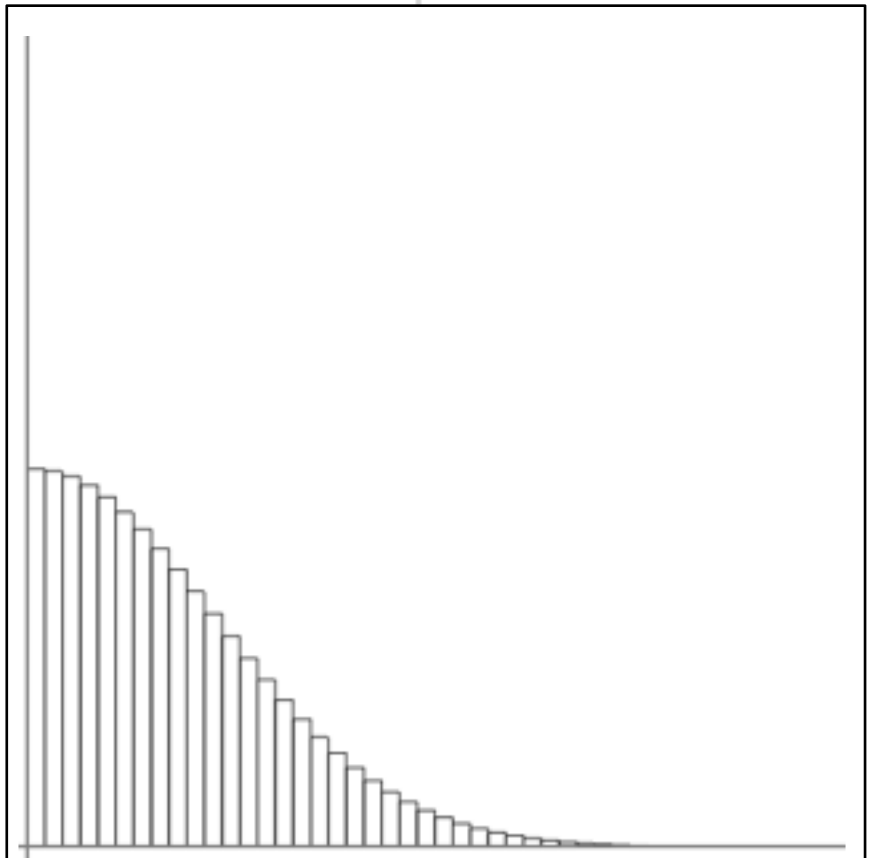




Filter Spectrum

Gaussian Convolution

Discrete Samples Reconstruction Filter Reconstructed Function



Filter Spectrum

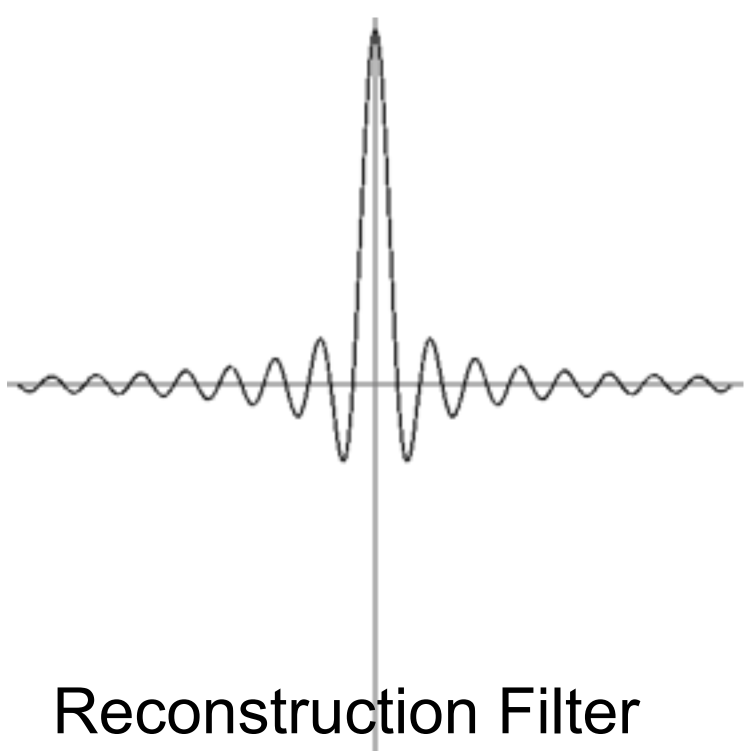
Convolution

- The ideal filter for avoiding aliasing has a power spectrum with values:
  - 1 at low frequencies
  - 0 at high frequencies
- The sinc function has such a power spectrum and is referred to as the ideal reconstruction filter:

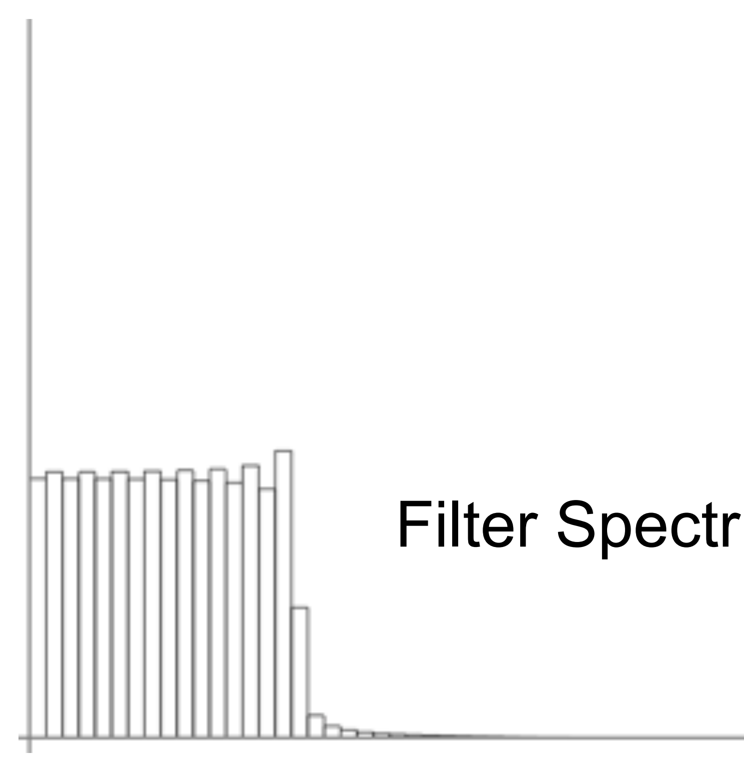
$$\boxed{\operatorname{sinc}(\theta) = \begin{cases} \dfrac{\sin(\theta)}{\theta} & \text{if } \theta \neq 0 \\ 1 & \text{if } \theta = 0 \end{cases}}$$

The Sinc Filter

- The ideal filter for avoiding aliasing has a power spectrum with values:
  - 1 at low frequencies
  - 0 at high frequencies
- The sinc function has such a power spectrum and is referred to as the ideal reconstruction filter:



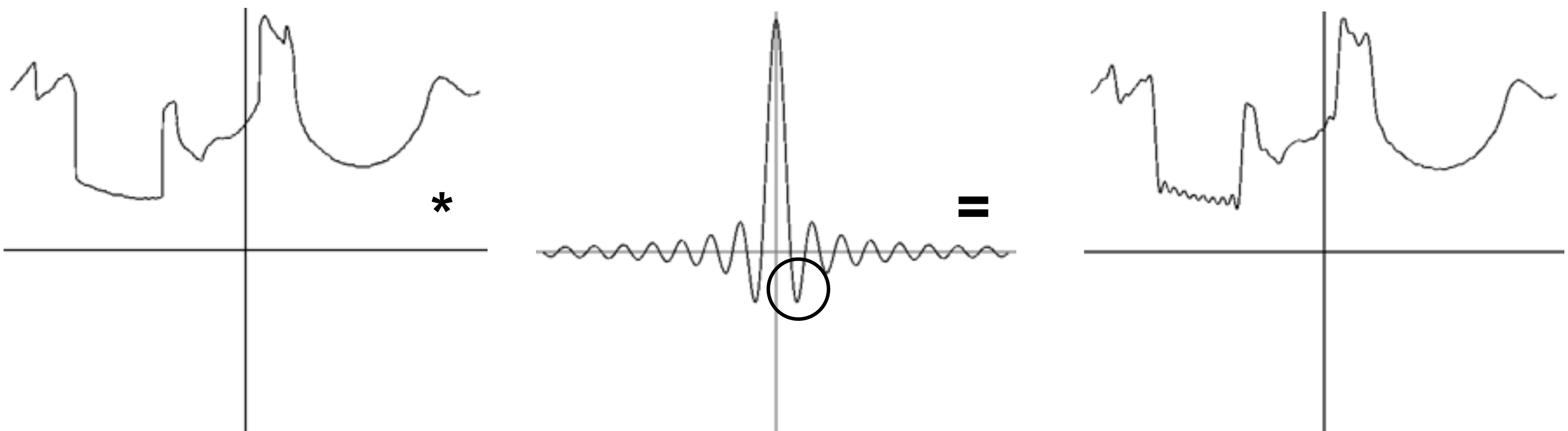
Reconstruction Filter



Filter Spectrum

The Sinc Filter

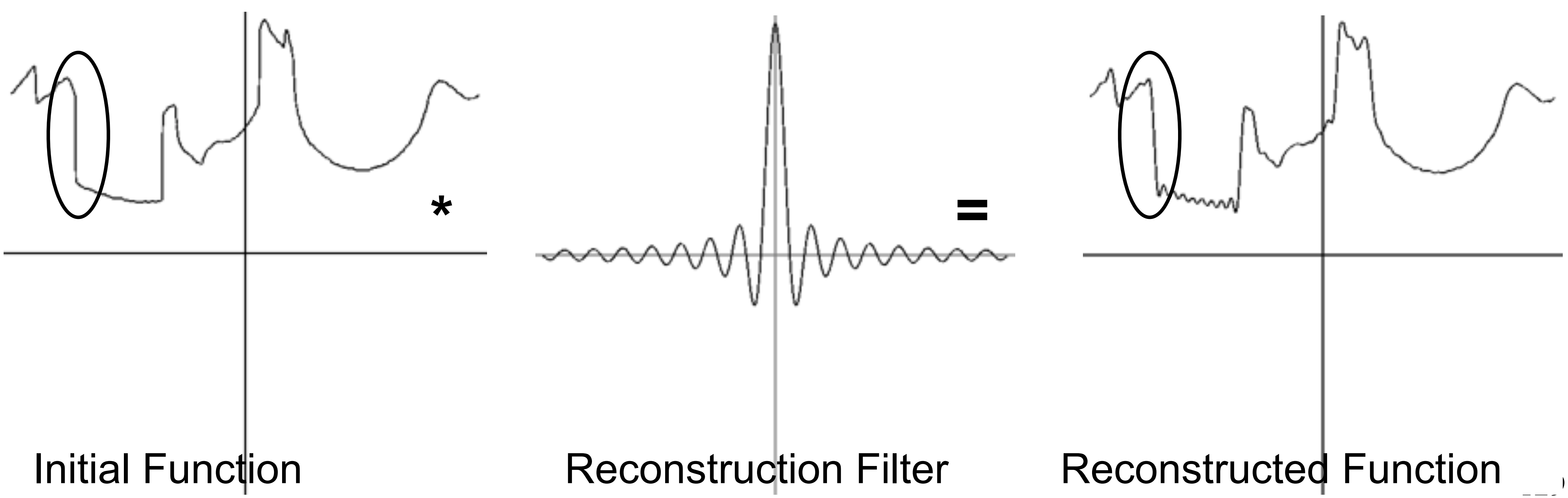
- Limitations:
  - Has negative values, giving rise to negative weights in the interpolation.
  - The discontinuity in the frequency domain (power spectrum) results in ringing artifacts known as the Gibbs Phenomenon.



Initial Function Reconstruction Filter Reconstructed Function

The Sinc Filter

- Limitations:
  - Has negative values, giving rise to negative weights in the interpolation.
  - The discontinuity in the frequency domain (power spectrum) results in ringing artifacts near spatial discontinuities, known as the Gibbs Phenomenon.



Summary

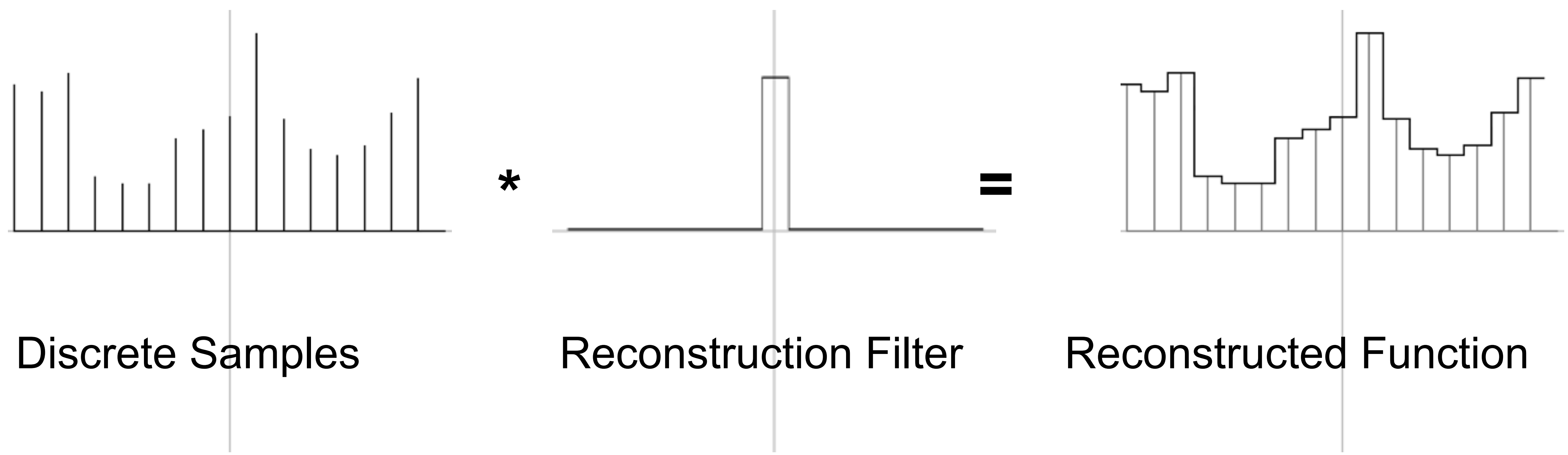
There are different ways to sample an image:

- Nearest Point Sampling
- Linear Sampling
- Gaussian Sampling
- Sinc Sampling

These methods have advantages and disadvantages.

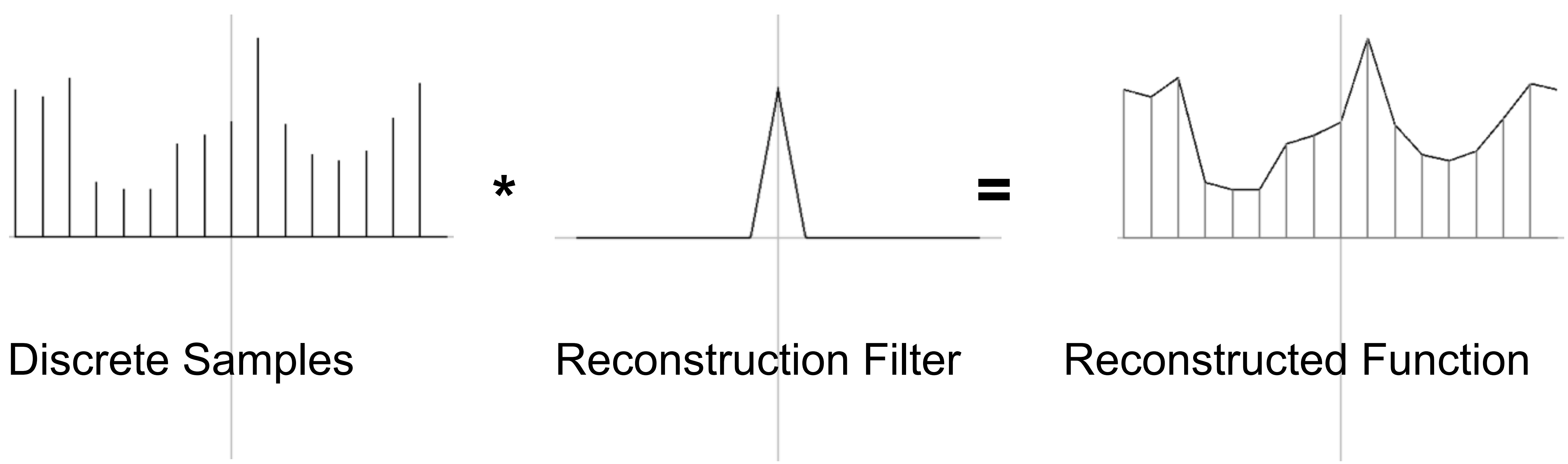
Summary – Nearest

- ✓ Can be implemented efficiently because the filter is non-zero in a very small region.
- ? Interpolates the samples.
- ✗ Is discontinuous.
- ✗ Does not address the aliasing problem, giving bad results when a signal is under-sampled.



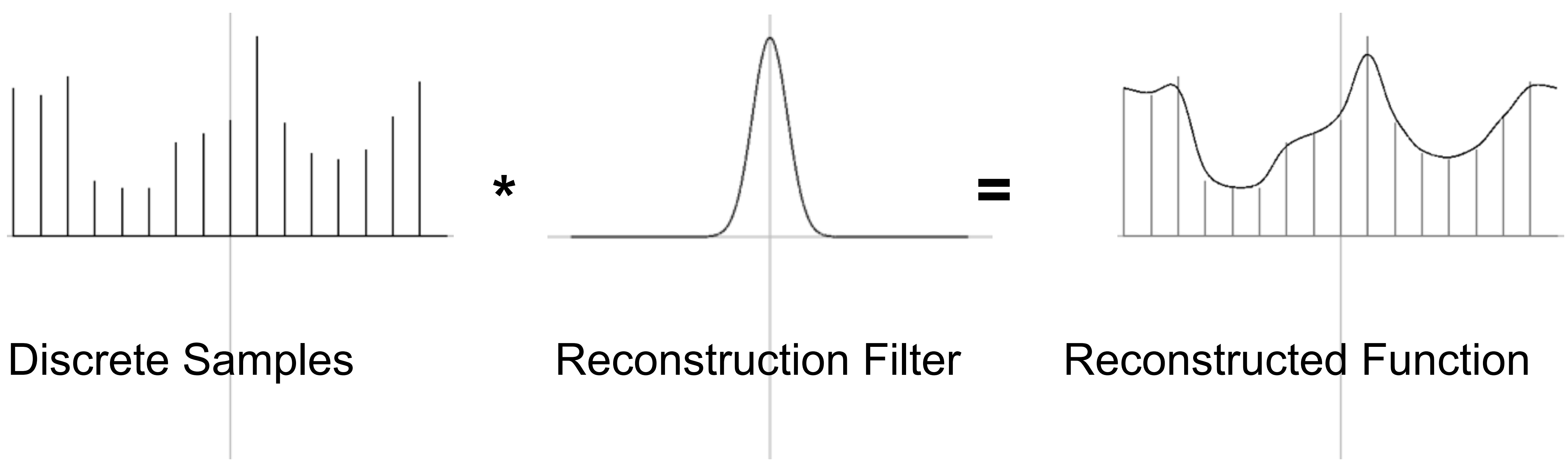
Summary – Linear

- ✓ Can be implemented efficiently because the filter is non-zero in a very small region.
- ? Interpolates the samples.
- ✗ Is not smooth.
- ✗ Partially addresses the aliasing problem, but can still give bad results when a signal is under-sampled.



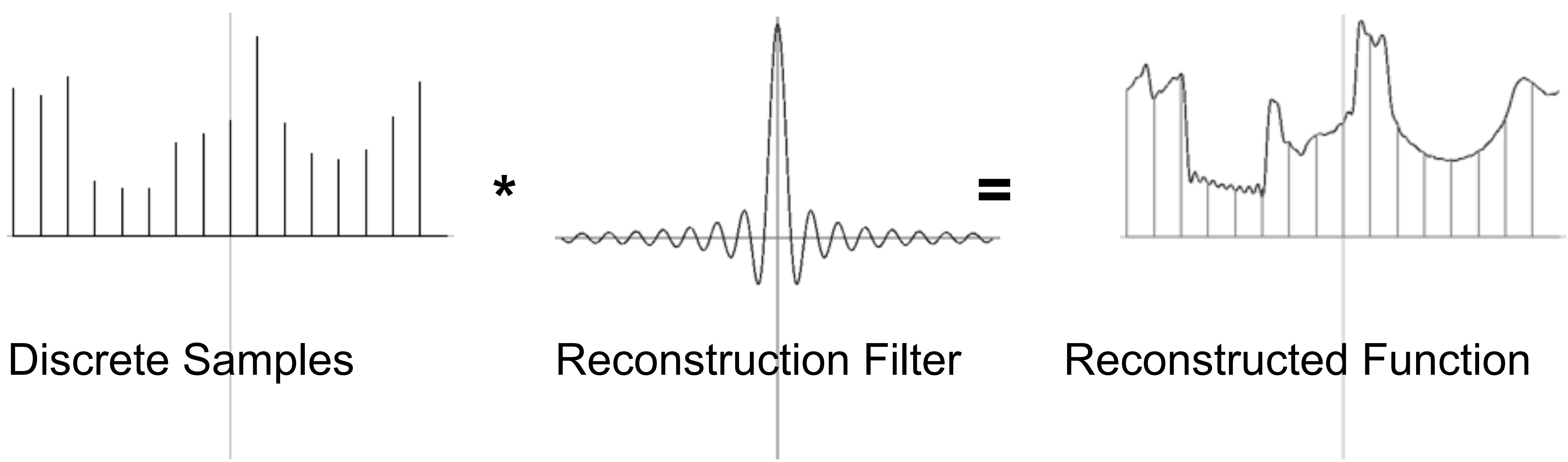
Summary – Gaussian

- ✗ Is slow to implement because the filter is non-zero in a large region.
- ? Does not interpolate the samples.
- ✓ Is smooth.
- ✓ Addresses the aliasing problem by killing off the high frequencies.



Summary – Sinc

- ✗ Is slow to implement because the filter is non-zero in a large region.
- ? Does not interpolate the samples.
- ✗ Assigns negative weights.
- ✗ Ringing at discontinuities.
- ✓ Addresses the aliasing problem by killing off the high frequencies.



Summary

Question:

- Is it good if a reconstruction method is interpolating? (Consider the case when you are down-scaling an image?)

Summary

It appears that we have been mixing the sampling problem with the reconstruction problem.

However, our motivation for the choice of filter is the same in both cases. We want a filter whose spectrum goes to zero so that:

- Sampling: High frequency samples are killed off, the signal becomes band-limited, and we can sample discretely.
- Reconstruction: We do not end up reconstructing a function with high frequency components.

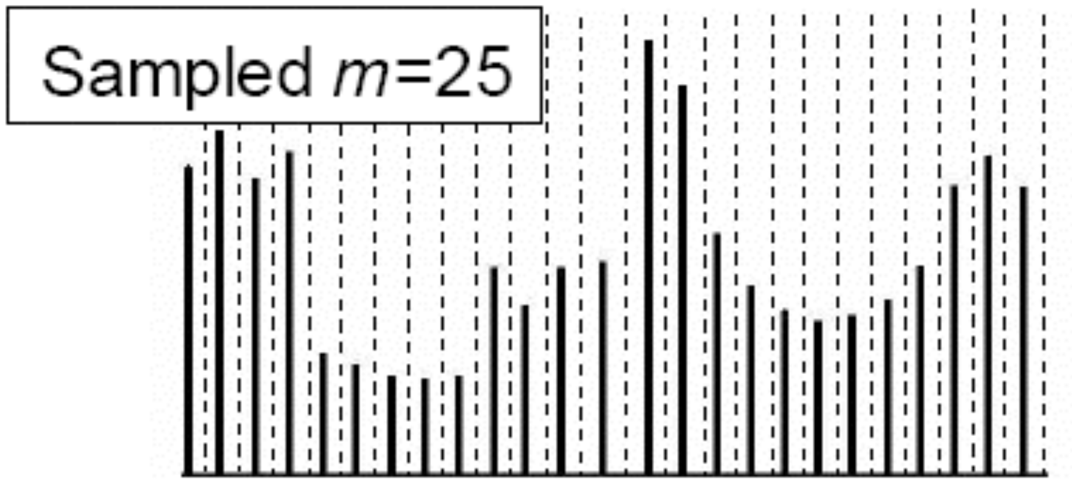
Image Sampling

Given a signal sampled at m positions, if we would like to re-sample at n positions we need to:

1. Reconstruct a function with maximum non-zero frequency no larger than min(m/2,n/2).
2. Sample the reconstructed function at the n positions.

Image Sampling

Example:



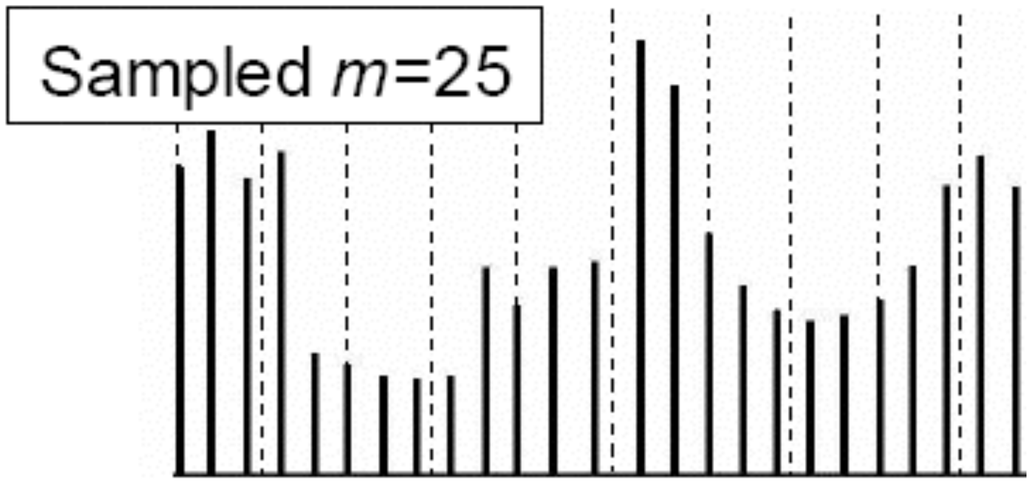
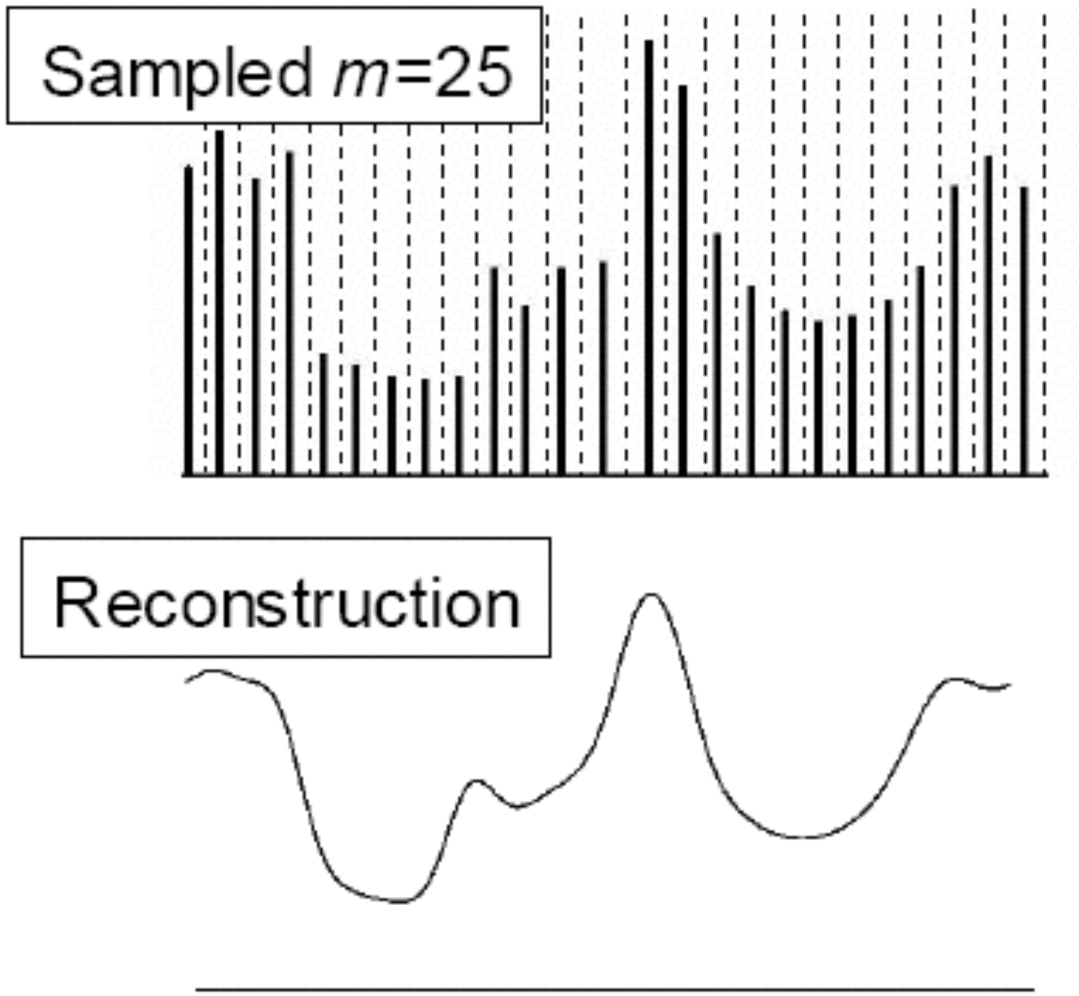


Image Sampling

Example:



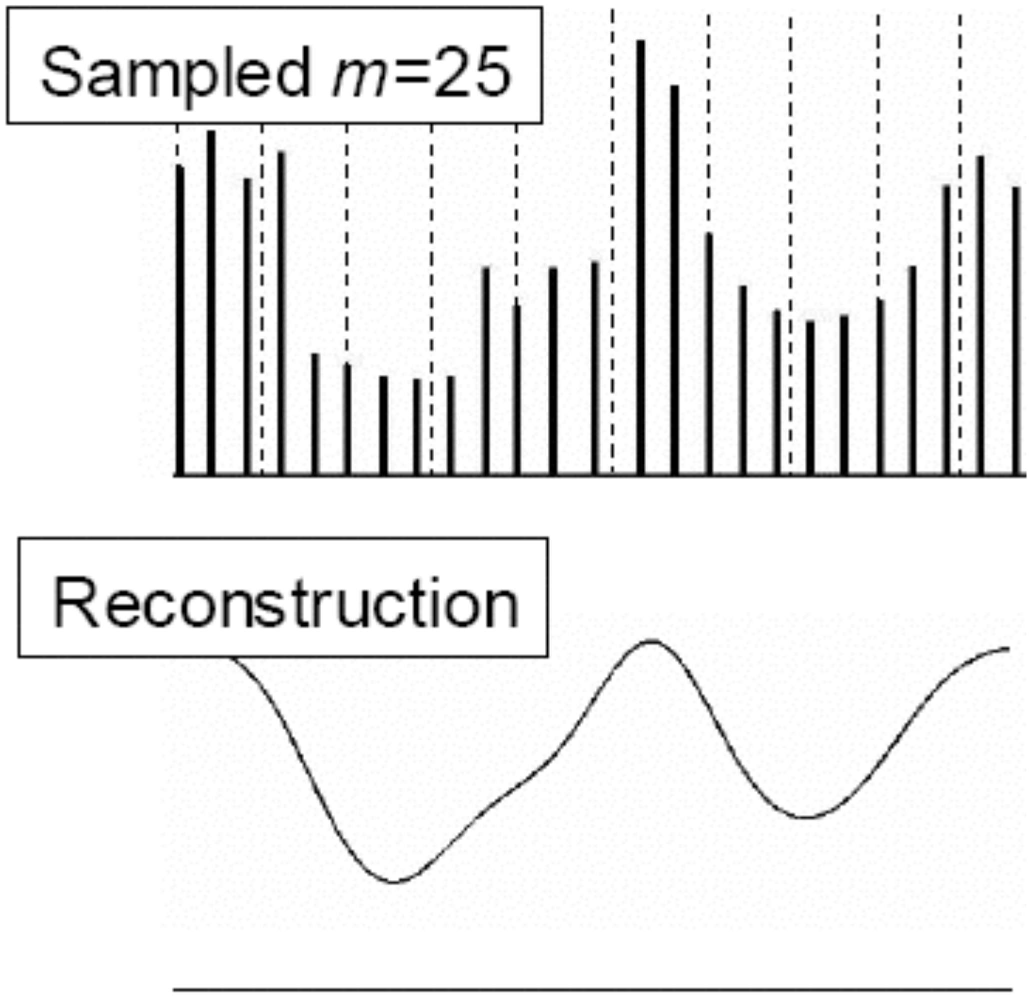
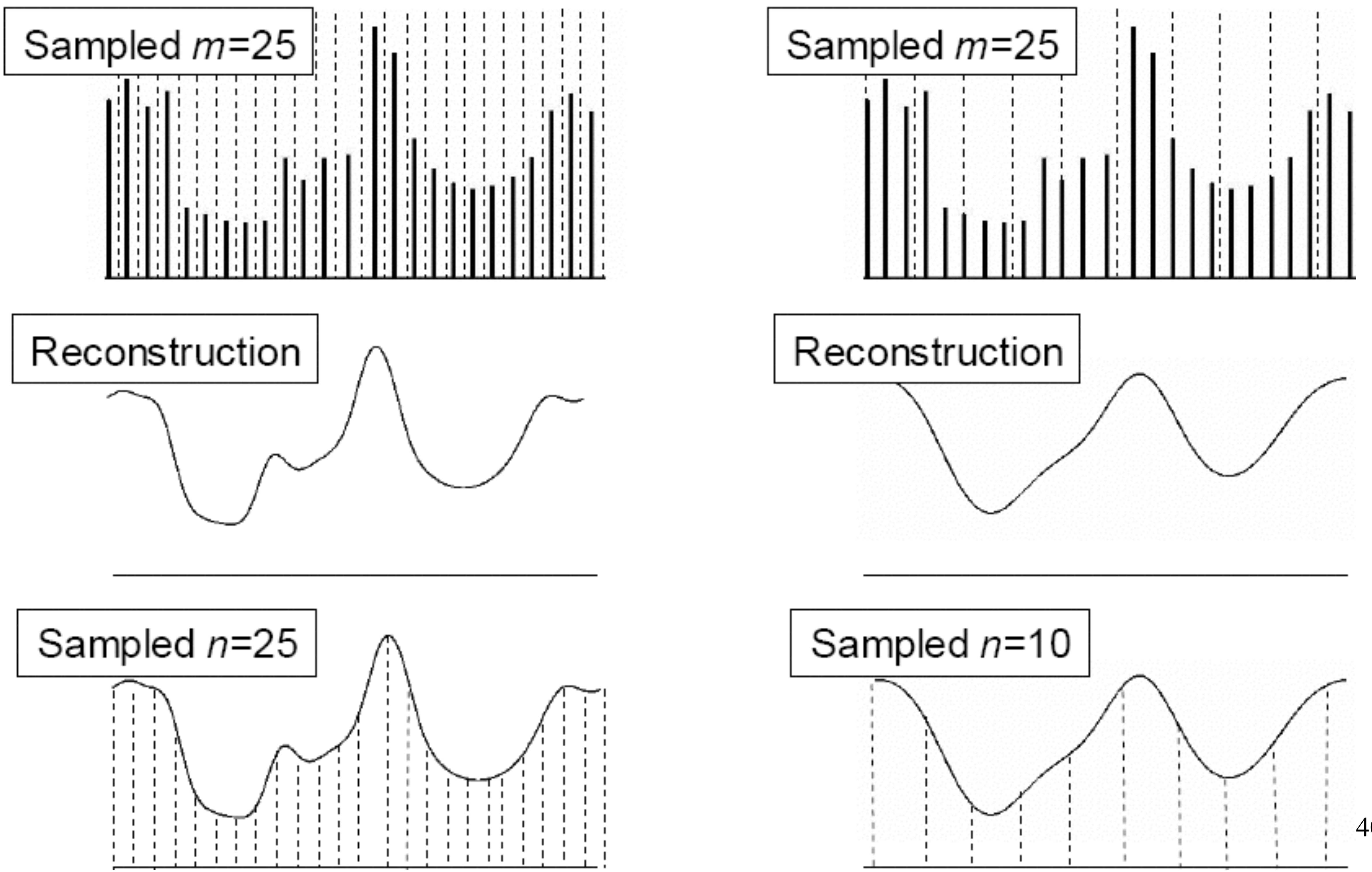


Image Sampling

Example:



Gaussian Sampling

Recall:

To avoid aliasing, we kill off high-frequency components, by convolving with a function whose power spectrum is zero at high frequencies.

We use a Gaussian for function reconstruction and sampling because it smoothly kills of the high frequency components.

Gaussian Sampling

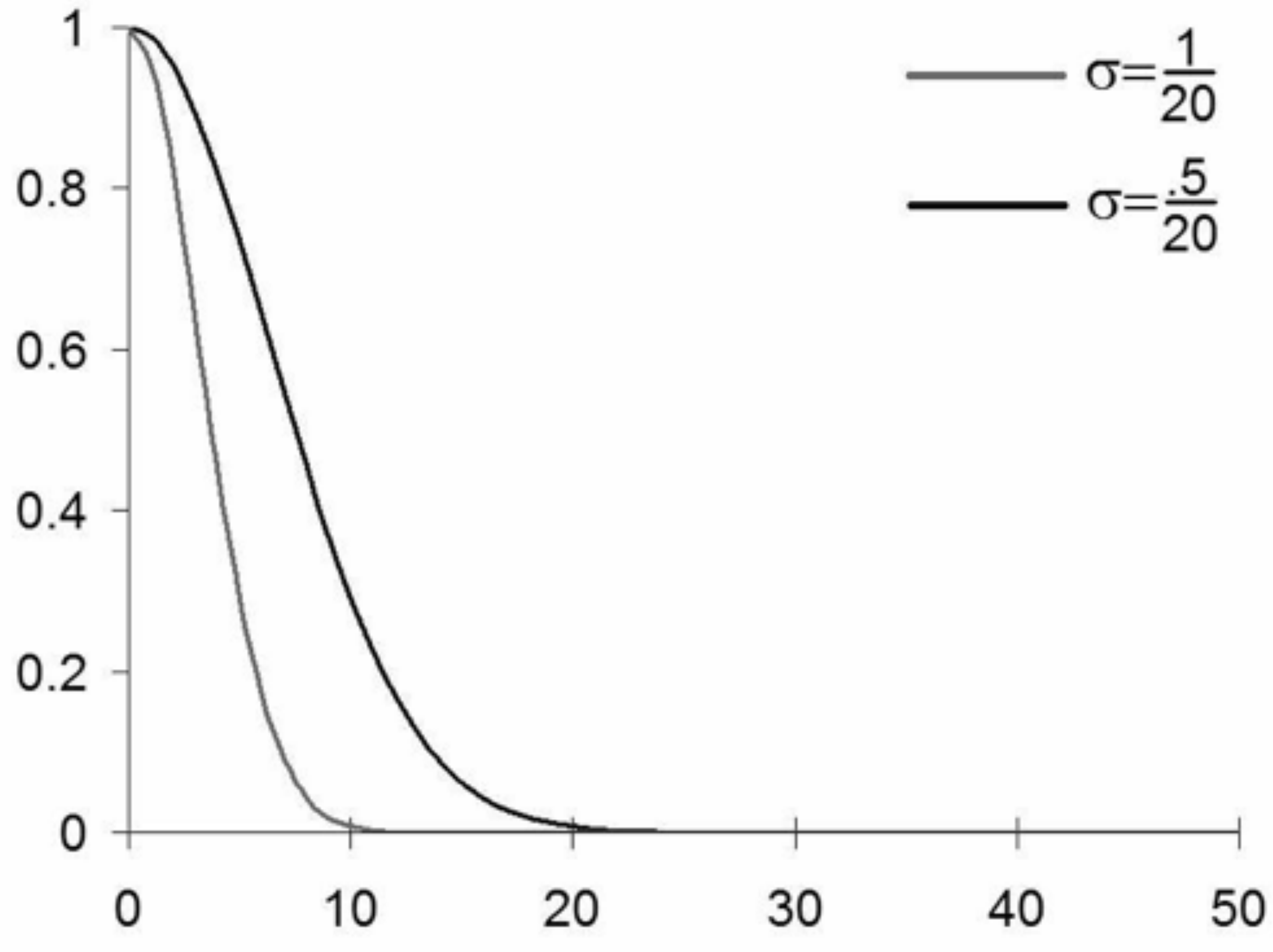
Q: What variance Gaussian should we use?

A: The variance of the Gaussian should be between 0.5 and 1.0 times the distance between samples.

Gaussian Sampling

Q: What variance Gaussian should we use?

A: The variance of the Gaussian should be between 0.5 and 1.0 times the distance between samples.



Power spectra of the Gaussians used for reconstructing and sampling a function with 20 samples

Gaussian Sampling

Scaling Example:

Q: Suppose we have data represented by 20 samples that we would like to down-sample to 5 samples. What variance should we use?

Gaussian Sampling

Scaling Example:

Q: Suppose we have data represented by 20 samples that we would like to down-sample to 5 samples. What variance should we use?

A: The distance between two adjacent samples in the final array corresponds to a distance of 4 units in the initial array.
The variance of the Gaussian should be between 2.0 and 4.0.

Gaussian Sampling

Scaling Example:

Q: Suppose we have data represented by 20 samples that we would like to up-sample to 40 samples. What variance should we use?

Gaussian Sampling

Scaling Example:

Q: Suppose we have data represented by 20 samples that we would like to up-sample to 40 samples. What variance should we use?

A: Because the initial samples can’t represent frequencies higher than 10, we shouldn’t use a Gaussian with smaller variance since this would introduce high-frequency components into the reconstruction. The variance of the Gaussian should remain between 0.5 and 1.0.