Image Processing, Warping, and Sampling

Michael Kazhdan

(600.457/657)
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

Stationary/Local Filtering

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

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

Original Blur

Filter = \[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\]
Blurring

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

Filter = \[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\]
Blurring

**Original**

**Filter**

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

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

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

<table>
<thead>
<tr>
<th>Y - 1</th>
<th>X - 1</th>
<th>X</th>
<th>X + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>36</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>109</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>109</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>36</td>
<td>73</td>
</tr>
</tbody>
</table>
### Blurring

The new value for a pixel's red component can be calculated as follows:

\[
\text{New value for Pixel}(x,y).\text{red} = (36 \times \frac{1}{16}) + (109 \times \frac{2}{16}) + (146 \times \frac{1}{16}) + (32 \times \frac{2}{16}) + (36 \times \frac{4}{16}) + (109 \times \frac{2}{16}) + (32 \times \frac{1}{16}) + (36 \times \frac{2}{16}) + (73 \times \frac{1}{16})
\]

The filter used for blurring is:

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

### Table

<table>
<thead>
<tr>
<th>Y - 1</th>
<th>X - 1</th>
<th>X</th>
<th>X + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>109</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>73</td>
<td></td>
</tr>
</tbody>
</table>

**Pixel**$(x,y).\text{red and its red neighbors}$
Blurring

New value for \( \text{Pixel}(x,y).\text{red} = 62.69 \)

\[
\begin{array}{ccc}
X - 1 & X & X + 1 \\
Y - 1 & 36 & 109 & 146 \\
Y & 32 & 36 & 109 \\
Y + 1 & 32 & 36 & 73 \\
\end{array}
\]

\[
\text{Filter} = \begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\]
Blurring

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
- Keep source and destination separate to avoid “drift”.
- 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.
Blurring

• In general, the mask can have arbitrary size:
  - We can have more non-zero entries to give rise to a wider blur.

\[
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 \\
0 & 1 & 2 & 1 & 0 \\
0 & 2 & 4 & 2 & 0 \\
0 & 1 & 2 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{bmatrix}
\quad \frac{1}{16}
\]

\[
\begin{bmatrix}
0 & 1 & 2 & 1 & 0 \\
1 & 2 & 4 & 2 & 1 \\
2 & 4 & 8 & 4 & 2 \\
1 & 2 & 4 & 2 & 1 \\
0 & 1 & 2 & 1 & 0 \\
\end{bmatrix}
\quad \frac{1}{48}
\]
Blurring

• A general way for defining the entries of an $n \times n$ mask is to use the values of a Gaussian:

$$\text{GaussianMask}[i][j] = e^{-\frac{d_i^2 + d_j^2}{2\sigma^2}}$$

- $\sigma$ equals the mask radius ($n/2$ for an $n \times n$ mask)
- $d_i$ is $i$’s horizontal distance from the center pixel
- $d_j$ is $j$’s vertical distance from the center pixel
- Don’t forget to normalize!
Edge Detection

- An edge is a point in the image where the image is “far” from constant.
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 (on average) than those of its neighbors.

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

Filter = \(\frac{1}{8}\) \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1
\end{bmatrix}
\]
Edge Detection

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

Filter = \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 &  8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
Edge Detection

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

\[
(36 \times -1/8) + (109 \times -1/8) + (146 \times -1/8) \\
(32 \times -1/8) + (36 \times 1) + (109 \times -1/8) \\
(32 \times -1/8) + (36 \times -1/8) + (73 \times -1/8)
\]

Filter = \( \frac{1}{8} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \)

<table>
<thead>
<tr>
<th>Y-1</th>
<th>X-1</th>
<th>X</th>
<th>X+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>109</td>
<td>146</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>73</td>
<td></td>
</tr>
</tbody>
</table>

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

Original
Edge Detection

Original

<table>
<thead>
<tr>
<th></th>
<th>X - 1</th>
<th>X</th>
<th>X + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y - 1</td>
<td>36</td>
<td>109</td>
<td>146</td>
</tr>
<tr>
<td>Y</td>
<td>32</td>
<td>36</td>
<td>109</td>
</tr>
<tr>
<td>Y + 1</td>
<td>32</td>
<td>36</td>
<td>73</td>
</tr>
</tbody>
</table>

New value for Pixel(x,y).red = -285/8

Filter = \( \frac{1}{8} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \)
**Edge Detection**

Original

```

<table>
<thead>
<tr>
<th>X - 1</th>
<th>X</th>
<th>X + 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>36</td>
<td>109</td>
<td>146</td>
</tr>
<tr>
<td>Y - 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>Y + 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

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

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

Filter = \[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
Edge Detection

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

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

Note: Edge values are not colors, so we have to rescale/remap for visualization.
Outline

• Image Processing
• Image Warping
• Image Sampling
Image Warping

- Move pixels of image
  - Mapping
  - Resampling
Overview

• Mapping
  ○ Forward
  ○ Inverse

• Resampling
  ○ Point sampling
  ○ Triangle filter
  ○ Gaussian filter
Mapping

• Define transformation
  ◦ Describe the destination \((x, y) = \Phi(u, v)\) for every location \((u, v)\) in the source
Example Mappings

- Scale by $\sigma$:
  - $\Phi(u, v) = (\sigma u, \sigma v)$

Scale $\sigma = 0.8$
Example Mappings

• Rotate by $\theta$ degrees:
  \[ \Phi(u, v) = (u \cos \theta - v \sin \theta, u \sin \theta + v \cos \theta) \]
Example Mappings

- Shear in $x$ by $\sigma_x$:
  \[ \Phi(u, v) = (u + \sigma_x \cdot v, v) \]

- Shear in $y$ by $\sigma_y$:
  \[ \Phi(u, v) = (u, v + \sigma_y \cdot u) \]
Other Mappings

- Any function of $u$ and $v$:
  - $\Phi(u, v) = \ldots$

Fish-eye

“Swirl”

“Rain”
Image Warping Implementation I

- Forward mapping:

\[
\text{for( int } v=0 ; v<v_{\text{max}} ; v++ ) \\
\text{for( int } u=0 ; u<u_{\text{max}} ; u++ ) \\
\text{float } (x,y) = \Phi(u,v); \\
\text{dst}(x,y) = \text{src}(u,v);
\]
Forward Mapping

• Iterate over source image
Forward Mapping – BAD!

- Iterate over source image

Multiple source pixels can map to same destination pixel
Forward Mapping – BAD!

- Iterate over source image

Multiple source pixels can map to same destination pixel

Some destination pixels may not be covered
Image Warping Implementation II

• Inverse mapping:

```c
for( int y=0 ; y<ymax ; y++ )
    for( int x=0 ; x<xmax ; x++ )
        float (u,v) = Φ⁻¹(x,y);
        dst(x,y) = src(u,v);
```

![Diagram showing source and destination images with inverse mapping](image)
Reverse Mapping – GOOD!

• Iterate over destination image
  ◦ Must resample source
  ◦ May oversample, but much simpler!
Resampling

- Evaluate source image at arbitrary \((u, v)\)

\((u, v)\) does not usually have integer coordinates
Overview

• Mapping
  ◦ Forward
  ◦ Inverse

• Resampling
  ◦ Nearest Point Sampling
  ◦ Bilinear Sampling
  ◦ Gaussian Sampling
Nearest Point Sampling

- Take value at closest pixel:
  
  $$\text{int } iu = \text{floor}(u+0.5);$$
  $$\text{int } iv = \text{floor}(v+0.5);$$
  $$\text{dst}(x,y) = \text{src}(iu,iv);$$
Bilinear Sampling

- Bilinearly interpolate four closest source pixels

\[ \text{dst}(x, y) = \text{Weighted average of source at } (u_1, v_1), (u_2, v_1), (u_1, v_2), \text{ and } (u_2, v_2) \]
Linear Sampling

- Linearly interpolate two closest source pixels

\[ \text{dst}(x) = \text{linear interpolation of } u_1 \text{ and } u_2 \]

\[
\begin{align*}
    u_1 &= \text{floor}(u); \\
    u_2 &= u_1 + 1; \\
    du &= u - u_1; \\
    \text{dst}(x) &= \text{src}(u_1) \times (1 - du) + \text{src}(u_2) \times du;
\end{align*}
\]
Bilinear Sampling

- Bilinearly interpolate four closest source pixels

\[ a = \text{linear interpolation of } \text{src}(u_1, v_1) \text{ and } \text{src}(u_2, v_1) \]
\[ b = \text{linear interpolation of } \text{src}(u_1, v_2) \text{ and } \text{src}(u_2, v_2) \]
\[ \text{dst}(x, y) = \text{linear interpolation of } a \text{ and } b \]

\[
\begin{align*}
  u_1 &= \text{floor}(u) \text{, } u_2 = u_1 + 1; \\
  v_1 &= \text{floor}(v) \text{, } v_2 = v_1 + 1; \\
  du &= u - u_1; \\
  a &= \text{src}(u_1,v_1)*(1-du) + \text{src}(u_2,v_1)*(du); \\
  b &= \text{src}(u_1,v_2)*(1-du) + \text{src}(u_2,v_2)*du; \\
  dv &= v - v_1; \\
  \text{dst}(x,y) &= a*(1-dv) + b*dv;
\end{align*}
\]
Bilinear Sampling

- Bilinearly interpolate four closest source pixels
  
  \[ a = \text{linear interpolation of } \text{src}(u_1, v_1) \text{ and } \text{src}(u_2, v_1) \]
  
  \[ b = \text{linear interpolation of } \text{src}(u_1, v_2) \text{ and } \text{src}(u_2, v_2) \]
  
  \[ \text{dst}(x, y) = \text{linear interpolation of } a \text{ and } b \]

\[
\begin{align*}
  u_1 &= \text{floor}(u), \ u_2 = u_1 + 1; \\
  v_1 &= \text{floor}(v), \ v_2 = v_1 + 1; \\
  du &= u - u_1; \\
  dv &= v - v_1; \\
  a &= \text{src}(u_1, v_1) \times (1 - du) \\
      &\quad + \text{src}(u_2, v_1) \times du; \\
  b &= \text{src}(u_1, v_2) \times (1 - dv) \\
      &\quad + \text{src}(u_2, v_2) \times dv; \\
  \text{dst}(x, y) &= a \times (1 - dv) + b \times dv;
\end{align*}
\]

Make sure to test that the pixels \((u_1, v_1), (u_2, v_2), (u_1, v_2), \) and \((u_2, v_1)\) are within the image.
Gaussian Sampling

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

\[(u, v)\]

Diagram showing a grid with a point \((u, v)\) and a normalized Gaussian function.
Filtering Methods Comparison

- Trade-offs
  - Jagged edges versus blurring
  - Computation speed

- Nearest
- Bilinear
- Gaussian
Image Warping Implementation

- Inverse mapping:

```c
for( int y=0 ; y<ymax ; y++ )
    for( int x=0 ; x<xmax ; x++ )
        float (u,v) = \Phi^{-1}(x,y);
        dst(x,y) = resample_src(u,v,w);
```

\( \Phi \)
Image Warping Implementation

- Inverse mapping:

```plaintext
define: for(int y=0; y<y_max; y++)
    for(int x=0; x<x_max; x++)
        float (u,v) = \Phi^{-1}(x,y);
        dst(x,y) = resample_src(u,v,w);
```

![Diagram](Image)
Example: Scale

\[ \text{Scale}( \text{src}, \text{dst}, \sigma ) : \]

\[
\begin{align*}
\text{float } w & \equiv \text{?} ; \\
\text{for( int } y=0 ; y<y_{\text{max}} ; y++ ) & \\
& \text{for( int } x=0 ; x<x_{\text{max}} ; x++ ) \\
& \quad \text{float } (u,v) = (x,y) / \sigma ; \\
& \quad \text{dst}(x,y) = \text{resample}_\text{src}(u,v,w) ;
\end{align*}
\]
Example: Rotate

Rotate(src, dst, \theta):

float w \equiv 1;
for( int y=0 ; y<y_{\text{max}} ; y++ )
    for( int x=0 ; x<x_{\text{max}} ; x++ )
        float (u,v) = ( x \cos(-\theta) - y \sin(-\theta), x \sin(-\theta) + y \cos(-\theta) );
        dst(x,y) = \text{resample\_src}(u,v,w);

\theta = 30

x = u \cos \theta - v \sin \theta
y = u \sin \theta + v \cos \theta
Example: Fun

Swirl( src , dst , θ ):

float w ≡ ?;
for( int y=0 ; y<ymax ; y++ )
    for( int x=0 ; x<xmax ; x++ )
        float (u,v) =
            rot( (xc,yc),(x,y),
                dist((x,y)-(xc,yc))*theta);
        dst(x,y) = resample_src(u,v,w);

(u,v) → Swirl → (x,y)

v

x

y
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?

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

The reconstruction:
- Interpolates the samples
- Is not smooth
Gaussian Sampling

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

Typically this is done in two steps:

1. Reconstruct a continuous function from input samples.
2. Sample a continuous function at a fixed resolution.

Challenge:

Reconstruction is an under-constrained problem.

⇒ To make headway, we need to define what makes a reconstruction good.
Image Sampling

Typically this is done in two steps:

1. Reconstruct a continuous function from input samples.
2. Sample a continuous function at a fixed resolution.

Challenge:

To make headway, we need to define what makes a reconstruction good.

Key Idea:

Of all possible reconstructions, we want the one that is smoothest (has lowest frequencies).

Signal processing helps us formulate this precisely.
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

\[ f_0(\theta) = a_0 \cdot \cos(0 \cdot (\theta + \phi_0)) \]

0th Order Component
Fourier Analysis

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

Initial Function

$\begin{align*}
\text{0th Order Approximation} \\
\text{1st Order Approximation} \\
f_1(\theta) &= a_1 \cdot \cos(1 \cdot (\theta + \phi_1)) \\
\end{align*}$
Fourier Analysis

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

Initial Function

$\mathbf{f}(\theta)$

$\mathbf{f}(\theta) = a_2 \cdot \cos(2 \cdot (\theta + \phi_2))$

2nd Order Component
Fourier Analysis

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

\[ f_3(\theta) = a_3 \cdot \cos(3 \cdot (\theta + \phi_3)) \]

3rd Order Component
Fourier Analysis

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

Initial Function

$\approx f_3(\theta)$

$\approx f_4(\theta)$

$4^{th}$ Order Approximation

$\approx f_4(\theta) = a_4 \cdot \cos(4 \cdot (\theta + \phi_4))$

$3^{rd}$ Order Approximation

$\approx f_3(\theta) + f_4(\theta)$
Fourier Analysis

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

\[ f_5(\theta) = a_5 \cdot \cos(5 \cdot (\theta + \phi_5)) \]

5th Order Component
Fourier Analysis

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

\[
f_6(\theta) = a_6 \cdot \cos(6 \cdot (\theta + \phi_6))
\]

Initial Function | 6\textsuperscript{th} Order Approximation
---|---
\[f(\theta)\] | 
\[\text{5\textsuperscript{th} Order Approximation} + \]
\[\text{6\textsuperscript{th} Order Component}\]
Fourier Analysis

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

$$f_7(\theta) = a_7 \cdot \cos(7 \cdot (\theta + \phi_8))$$

6th Order Approximation

7th Order Approximation

Initial Function

7th Order Component
Fourier Analysis

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

\[
f(\theta) = a_8 \cdot \cos(8 \cdot (\theta + \phi_8))
\]

8th Order Component
Fourier Analysis

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

Initial Function

$\text{9}^{\text{th}}$ Order Approximation

$\text{8}^{\text{th}}$ Order Approximation

$\text{9}^{\text{th}}$ Order Component

$f_9(\theta) = a_9 \cdot \cos(9 \cdot (\theta + \phi_9))$
Fourier Analysis

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

\[ f_{10}(\theta) = a_{10} \cdot \cos(10 \cdot (\theta + \phi_{10})) \]
Fourier Analysis

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

\[
f_{11}(\theta) = a_{11} \cdot \cos(11 \cdot (\theta + \phi_{11}))
\]

11th Order Component
Fourier Analysis

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

\[ f_{12}(\theta) = a_{12} \cdot \cos(12 \cdot (\theta + \phi_{12})) \]

Initial Function

12\textsuperscript{th} Order Component

12\textsuperscript{th} Order Approximation

11\textsuperscript{th} Order Approximation

\[ f(\theta) \]
Fourier Analysis

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

\[
f_{13}(\theta) = a_{13} \cdot \cos(13 \cdot (\theta + \phi_{13}))
\]

13th Order Component
Fourier Analysis

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

Initial Function

$\mathbf{f(\theta)}$

14th Order Approximation

$\mathbf{f_{14}(\theta)} = a_{14} \cdot \cos(14 \cdot (\theta + \phi_{14}))$

13th Order Approximation
Fourier Analysis

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

Initial Function

\[ f_{15}(\theta) = a_{15} \cdot \cos(15 \cdot (\theta + \phi_{15})) \]

15th Order Approximation
Fourier Analysis

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

\[
f_{16}(\theta) = a_{16} \cdot \cos(16 \cdot (\theta + \phi_{16}))
\]

16th Order Component
Fourier Analysis

• Combining all of the frequency components together, we get the initial function:

\[ f(\theta) = \sum_{k=0}^{\infty} f_k(\theta) = \sum_{k=0}^{\infty} a_k \cdot \cos(k(\theta + \phi_k)) \]

- \( a_k \): amplitude of the \( k^{th} \) frequency component
- \( \phi_k \): shift of the \( k^{th} \) 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

Shannon’s Theorem:

A signal can be reconstructed from its samples, if the original signal has no frequencies above $1/2$ the sampling frequency.

Definition:

- A signal is *band-limited* if its highest non-zero frequency is bounded.
- The frequency is called the *bandwidth*.
- The minimum sampling rate for band-limited function is called the *Nyquist frequency* (twice the bandwidth).
Image Sampling

1. To reconstruct the continuous function from $m$ samples, we can find the unique function of frequency $m/2$ that interpolates the values.

2. Why don’t we just evaluate this function at the $n$ sample positions?

If $n < m$ we sample below the Nyquist frequency!
Aliasing

- When a high-frequency signal is sampled with insufficiently many samples, it can 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 can 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 can 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 can be perceived as a lower-frequency signal. This masking of higher frequencies as lower ones is referred to as \textit{aliasing}. 
Temporal Aliasing

- Artifacts due to limited temporal resolution

10 fps
Temporal Aliasing

- Artifacts due to limited temporal resolution

![Diagram showing the effect of different frame rates on temporal aliasing](image)
Temporal Aliasing

- Artifacts due to limited temporal resolution
Temporal Aliasing

- Artifacts due to limited temporal resolution
Temporal Aliasing

• Artifacts due to limited temporal resolution
Temporal Aliasing

• Artifacts due to limited temporal resolution
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.