Image Warping and Sampling

Jason Lawrence

CS445: Graphics

Acknowledgement: slides by 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.

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

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

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

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

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

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)

Filter =
[1/16 2/16 1/16]
[2/16 4/16 2/16]
[1/16 2/16 1/16]

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

Original

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

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

\[
\begin{array}{c|c|c}
X - 1 & X & X + 1 \\
\hline
36 & 109 & 146 \\
32 & 36 & 109 \\
32 & 36 & 73 \\
\end{array}
\]

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

• **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.
In general, the mask can have arbitrary size:

- We can have more non-zero entries to give rise to a wider blur.

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

\[
\begin{bmatrix}
0 & 1 & 2 & 1 & 0 \\
1 & 2 & 4 & 2 & 1 \\
2 & 4 & 8 & 4 & 2 \\
1 & 2 & 4 & 2 & 1 \\
0 & 1 & 2 & 1 & 0
\end{bmatrix}/48
\]
A general way for defining the entries of an $n \times n$ mask is to use the values of a Gaussian:

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

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

Original

Detected Edges

\[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1
\end{bmatrix}
\]
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**

Original

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

<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>32</td>
<td>36</td>
<td>109</td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>36</td>
<td>73</td>
</tr>
<tr>
<td>Y + 1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 =
(36 * -1) + (109 * -1) + (146 * -1)
(32 * -1) + (36 * 8) + (109 * -1)
(32 * -1) + (36 * -1) + (73 * -1)

Filter = 
\[-1 \quad -1 \quad -1\]
\[-1\quad 8\quad -1\]
\[-1\quad -1\quad -1\]
Edge Detection

New value for $\text{Pixel}(x,y).\text{red} = -285$

Filter:

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

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

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

Original

Detected Edges

\[
\begin{bmatrix}
-1 & -1 & -1 \\
-1 & 8 & -1 \\
-1 & -1 & -1 \\
\end{bmatrix}
\]
Outline

- Image Processing
- Image Warping
- Image Sampling
Image Warping

- Move pixels of image
  - Mapping
  - Resampling

Source image  
Destination image
Overview

• Mapping
  ◦ Forward
  ◦ Reverse

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

- Define transformation
  - Describe the destination \((x,y)\) for every location \((u,v)\) in the source (or vice-versa, if invertible)
Example Mappings

- Scale by factor:
  - $x = \text{factor} \times u$
  - $y = \text{factor} \times v$

Scale 0.8
Example Mappings

- Rotate by $\theta$ degrees:
  
  $\begin{align*}
  x &= uc\cos \theta - vs\sin \theta \\
  y &= us\sin \theta + vc\cos \theta
  \end{align*}$

  Rotate 30
Example Mappings

- Shear in X by \textit{factor}:
  \[ x = u + \text{factor} \times v \]
  \[ y = v \]

- Shear in Y by \textit{factor}:
  \[ x = u \]
  \[ y = v + \text{factor} \times u \]

Shear X
\[ 1.3 \]
Shear Y
\[ 1.3 \]
Other Mappings

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

Fish-eye

“Swirl”

“Rain”
Image Warping Implementation I

- Forward mapping:

```c
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);
```

![Diagram](image.png)
Forward Mapping

- Iterate over source image
Forward Mapping – BAD!

- Iterate over source image

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

- Iterate over source image

Many source pixels can map to same destination pixel

Some destination pixels may not be covered

Rotate -30
Image Warping Implementation II

- Reverse mapping:

```c
for (int x = 0; x < xmax; x++)
    for (int y = 0; y < ymax; y++)
        float u = f^{-1}_x(x,y);
        float v = f^{-1}_y(x,y);
        dst(x,y) = src(u,v);
```

![Diagram of source and destination images with reverse mapping formula](source_image_destination_image_diagram.png)
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

Source image to Destination image
Overview

• Mapping
  ◦ Forward
  ◦ Reverse

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

- Take value at closest pixel:
  
  ```
  int iu = trunc(u+0.5);
  int iv = trunc(v+0.5);
  dst(x,y) = src(iu,iv);
  ```
Bilinear Sampling

• Bilinearly interpolate four closest pixels
  
  \[ a = \text{linear interpolation of src}(x_1,y_1) \text{ and src}(x_2,y_1) \]
  
  \[ b = \text{linear interpolation of src}(x_1,y_2) \text{ and src}(x_2,y_2) \]
  
  \[ \text{dst}(x,y) = \text{linear interpolation of “a” and “b”} \]

\[ x1 = \text{floor}(x) ; \]
\[ x2 = x1 + 1 ; \]
\[ y1 = \text{floor}(y) ; \]
\[ y2 = y1 + 1 ; \]
\[ dx = x - x1 ; \]
\[ dy = y - y1 ; \]
\[ a = \text{src}(x1,y1)*(1-dx) + \text{src}(x2,y1)*dx ; \]
\[ b = \text{src}(x1,y2)*(1-dx) + \text{src}(x2,y2)*dx ; \]
\[ \text{dst}(x,y) = a*(1-dy) + b*dy ; \]
Bilinear Sampling

- Bilinearly interpolate four closest pixels
  
  \[ a = \text{linear interpolation of } \text{src}(x_1, y_1) \text{ and } \text{src}(x_2, y_1) \]
  
  \[ b = \text{linear interpolation of } \text{src}(x_1, y_2) \text{ and } \text{src}(x_2, y_2) \]
  
  \[ \text{dst}(x, y) = \text{linear interpolation of } "a" \text{ and } "b" \]

\[
x_1 = \text{floor}(x)
\]
\[
x_2 = x_1 + 1
\]
\[
y_1 = \text{floor}(y)
\]
\[
y_2 = y_1 + 1
\]
\[
dx = x - x_1
\]
\[
dy = y - y_1;
\]
\[
a = \text{src}(x_1, y_2)*(1-dx) + \text{src}(x_2, y_2)*dx;
\]
\[
b = \text{src}(x_1, y_1)*(1-dx) + \text{src}(x_2, y_1)*dx;
\]
\[
\text{dst}(x, y) = a*(1-dy) + b*dy;
\]

Make sure to test that the pixels \((x_1, y_1), (x_2, y_2), (x_1, y_2), \text{ and } (x_2, y_1)\) are within the image.
Gaussian Sampling

• Compute weighted sum of pixel neighborhood:
  ◦ The blending weights are the normalized values of a Gaussian function.
Filtering Methods Comparison

- Trade-offs
  - Jagged edges versus blurring
  - Computation speed
Image Warping Implementation

- Reverse mapping:

  for (int x = 0; x < xmax; x++)
    for (int y = 0; y < ymax; y++)
      float u = f^{-1}_x(x,y);
      float v = f^{-1}_y(x,y);
      dst(x,y) = resample_src(u,v,w);
Image Warping Implementation

- Reverse mapping:
  
  ```
  for (int x = 0; x < xmax; x++)
    for (int y = 0; y < ymax; y++)
      float u = f^{-1}_x(x,y);
      float v = f^{-1}_y(x,y);
      dst(x,y) = resample_src(u,v,w);
  ```

![Diagram of image warping process](image)
Example: Scale

- Scale \((\text{src}, \text{dst}, s)\):

  
  
  ```
  float w \equiv \, ?;
  for (int x = 0; x < x_{\text{max}}; x++)
    for (int y = 0; y < y_{\text{max}}; y++)
      float u = x / s;
      float v = y / s;
      \text{dst}(x,y) = \text{resample}_\text{src}(u,v,w);
  ```

  ![Diagram showing scale operation on an image](image.png)
Example: Scale

- 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);
  ```

  ![Diagram of Scale Transformation]

  \[w=1.0/s\]
Example: Rotate

- Rotate \((src, \text{dst}, \theta)\):
  
  ```plaintext
  float w \equiv ?;
  for (int x = 0; x < xmax; x++)
    for (int y = 0; y < ymax; y++)
      float u = x*\cos(-\theta) - y*\sin(-\theta);
      float v = x*\sin(-\theta) + y*\cos(-\theta);
      \text{dst}(x,y) = \text{resample}_src(u,v,w);
  
  x = uc\cos\theta - vs\sin\theta
  y = us\sin\theta + vc\cos\theta
  ```
Example: Rotate

- Rotate (src, dst, theta):
  
  \[
  \text{float } w \approx \?; \\
  \text{for (int } x = 0; x < \text{xmax}; x++) \\
  \quad \text{for (int } y = 0; y < \text{ymax}; y++) \\
  \quad \text{float } u = x \cdot \cos(-\theta) - y \cdot \sin(-\theta); \\
  \quad \text{float } v = x \cdot \sin(-\theta) + y \cdot \cos(-\theta); \\
  \quad \text{dst}(x,y) = \text{resample}\_\text{src}(u,v,w); \\
  \]

\[w=1.0\]

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

- 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?
What is an image?

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

How do we define the in-between values?
Nearest Point Sampling

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

Reconstructed Function  Discrete Samples
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

- 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

<table>
<thead>
<tr>
<th>Initial Function</th>
<th>0th Order Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(\theta)$</td>
<td></td>
</tr>
</tbody>
</table>

$0^{th}$ Order Component

\[ f_0(\theta) = a_0 \cos(\theta + \phi_0) \]
Fourier Analysis

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

Initial Function

$\begin{align*}
\text{Initial Function} & \quad | \quad 1^{\text{st}} \text{ Order Approximation} \\
\hline
f(\theta) & \quad | \quad 1^{\text{st}} \text{ Order Approximation} \\
\hline
\end{align*}$

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

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

\[
\begin{align*}
\text{Initial Function} & \quad f(\theta) \\
\text{1st Order Approximation} & \quad + \\
\text{2nd Order Component} & \quad f_2(\theta) = a_2(\cos 2(\theta + \phi_2)) \\
\text{2nd Order Approximation} & 
\end{align*}
\]
Fourier Analysis

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

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

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

Initial Function

<table>
<thead>
<tr>
<th>4th Order Approximation</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f(\theta) )</td>
</tr>
</tbody>
</table>

3rd Order Approximation

\[ f_4(\theta) = a_4 \cos(4(\theta + \phi_4)) \]

4th Order Component
Fourier Analysis

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

$\theta$

Initial Function

$\theta$

5th Order Approximation

$\theta$

4th Order Approximation

$\theta$

5th Order Component

$f_5(\theta) = a_5\cos(5(\theta + \phi_5))$
Fourier Analysis

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

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

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

\[ f_7(\theta) = a_7 \cos(7(\theta + \phi_7)) \]

6th Order Approximation

7th Order Approximation

Initial Function

\[ f(\theta) \]
Fourier Analysis

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

\[
f_8(\theta) = a_8 \cos(8(\theta + \phi_8))
\]

\[
f(\theta) = f_7(\theta) + f_8(\theta)
\]
Fourier Analysis

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

\[ f_9(\theta) = a_9 \cos(9(\theta + \phi_9)) \]

9th Order Component
Fourier Analysis

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

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

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

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

**Initial Function**

**11th Order Approximation**

**10th Order Approximation**

**11th Order Component**
Fourier Analysis

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

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

12\(^{th}\) Order Component
Fourier Analysis

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

Initial Function

13\textsuperscript{th} Order Approximation

12\textsuperscript{th} Order Approximation

13\textsuperscript{th} Order Component

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

\[ f(\theta) \]
Fourier Analysis

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

\[ f_{14}(\theta) = a_{14}\cos(14(\theta + \phi_{14})) \]

Initial Function | 14\textsuperscript{th} Order Approximation

13\textsuperscript{th} Order Approximation + 14\textsuperscript{th} Order Component
Fourier Analysis

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

Initial Function

15th Order Approximation

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

14th Order Approximation

\[ f(\theta) \]
Fourier Analysis

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

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

16th Order Component
Fourier Analysis

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

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

- \( a_k \): 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

• 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
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:

\[
\begin{align*}
\text{Initial Power Spectrum} & \quad \times \quad \text{Frequency Filter} \\
\begin{array}{c}
\sum_{k=0}^{\infty} a_k \cos(k(\theta + \phi_k)) \\
\sum_{k=0}^{n/2} a_k \cos(k(\theta + \phi_k))
\end{array} & \quad = \\
\text{Band-Limited Spectrum}
\end{align*}
\]
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 $g$. 

$$f(\theta)$$

$$g(\theta)$$
Convolution

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

$$ (f * g)(\theta) $$
Convolution

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

\[ (f \ast g)(\theta) \]
Convolution

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

\[ f(\theta) \]

\[ g(\theta) \]

\[ (f * g)(\theta) \]
Convolution

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

$$f(\theta) * g(\theta)$$
Convolution

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

$$ (f * g)(\theta) $$
Convolution

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

$$f(\theta) \ast g(\theta)$$ 

$$(f \ast g)(\theta)$$
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 $g$. 

$$(f * g)(\theta)$$
Convolution

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

$$f(\theta) \ast g(\theta)$$
Convolution

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

$$f(\theta) \ast g(\theta)$$
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

Discrete Samples $\ast$ Reconstruction Filter = Reconstructed Function

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:

\[
sinc(\theta) = \begin{cases} 
\frac{\sin(\theta)}{\theta} & \text{if } \theta \neq 0 \\
1 & \text{if } \theta = 0 
\end{cases}
\]
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:
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.
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.

Discrete Samples * Reconstruction Filter = Reconstructed Function
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.

Discrete Samples * Reconstruction Filter = Reconstructed Function
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.

Discrete Samples  ✖  Reconstruction Filter  =  Reconstructed Function
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.

Discrete Samples * Reconstruction Filter = Reconstructed Function
Summary

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