Sampling, Resampling, and Warping

Adam Finkelstein
Princeton University
COS 426, Spring 2018
Digital Image Processing

- Changing pixel values
  - Linear: scale, offset, etc.
  - Nonlinear: gamma, saturation, etc.
  - Histogram equalization

- Filtering over neighborhoods
  - Blur & sharpen
  - Detect edges
  - Median
  - Bilateral filter

- Moving image locations
  - Scale
  - Rotate
  - Warp

- Combining images
  - Composite
  - Morph

- Quantization
  - Spatial / intensity tradeoff
  - Dithering
Image Warping

- Move pixels of an image

Source image  \[\text{Warp}\]  Destination image
Image Warping

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

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

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

• Forward mapping

• Reverse mapping
Mapping

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

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

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

• Shear in X by factor:
  ◦ $x = u + \text{factor} \times v$
  ◦ $y = v$

• Shear in Y by factor:
  ◦ $x = u$
  ◦ $y = v + \text{factor} \times u$
Other Parametric Mappings

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

Fish-eye

“Swirl”

“Rain”
More COS426 Examples

Sid Kapur

Michael Oranato

Eirik Bakke
Point Correspondence Mappings

- Mappings implied by correspondences:
  - $A \leftrightarrow A'$
  - $B \leftrightarrow B'$
  - $C \leftrightarrow C'$

![Warping Example](image)
Line Correspondence Mappings

- Beier & Neeley use pairs of lines to specify warp
Image Warping

• Issues:
  ◦ Specifying where every pixel goes (mapping)
  ➢ Computing colors at destination pixels (resampling)
When implementing operations that move pixels, must account for the fact that digital images are sampled versions of continuous ones.
Sampling and Reconstruction

Continuous function

Discrete samples

Sampling
Sampling and Reconstruction

Sampling

Reconstruction

Continuous function

Discrete samples

Continuous function
Sampling and Reconstruction

Figure 19.9 FvDFH
Sampling Theory

How many samples are enough?

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

What happens when we use too few samples?
What happens when we use too few samples?

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

Specifically, in graphics:

- Spatial aliasing
- Temporal aliasing

Figure 14.17 FvDFH
Spatial Aliasing

Artifacts due to limited spatial resolution
Spatial Aliasing

Artifacts due to limited spatial resolution

(Barely) adequate sampling

Inadequate sampling
Spatial Aliasing

Artifacts due to limited spatial resolution
Spatial Aliasing

Artifacts due to limited spatial resolution

“Jaggies”
Temporal Aliasing

Artifacts due to limited temporal resolution

- Strobing
- Flickering
Temporal Aliasing

Artifacts due to limited temporal resolution

- Strobing
- Flickering
Temporal Aliasing

Artifacts due to limited temporal resolution

- Strobing
- Flickering
Temporal Aliasing

Artifacts due to limited temporal resolution

- Strobing
- Flickering
Sampling Theory

How many samples are enough to avoid aliasing?

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

How many samples are enough to avoid aliasing?

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

Inadequate
Sampling Theory

How many samples are enough to avoid aliasing?

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

Adequate?
Sampling Theory

How many samples are enough to avoid aliasing?

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

Inadequate
Sampling Theory

How many samples are enough to avoid aliasing?

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

Adequate
Sampling Theory

How many samples are enough to avoid aliasing?

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

• Spatial domain:
  ◦ Function: \( f(x) \)
  ◦ Filtering: convolution

• Frequency domain:
  ◦ Function: \( F(u) \)
  ◦ Filtering: multiplication

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

\[ f(x) \]

\[ |F(u)| \]

Figure 2.6 Wolberg
Fourier Transform

- Fourier transform:
  \[ F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi xu} \, dx \]

- Inverse Fourier transform:
  \[ f(x) = \int_{-\infty}^{\infty} F(u) e^{+i2\pi ux} \, du \]
Sampling Theorem

• A signal can be reconstructed from its samples iff it has no content \( \geq \frac{1}{2} \) the sampling frequency – Shannon

• The minimum sampling rate for a bandlimited function is called the “Nyquist rate”

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

• Sample at higher rate
  ○ Not always possible
  ○ Doesn’t always solve the problem

• Pre-filter to form bandlimited signal
  ○ Use low-pass filter to limit signal to $< 1/2$ sampling rate
  ○ Trades blurring for aliasing
Image Processing

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

Original image  
1/4 resolution
Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display
Image Processing

Real world

Sample
- Discrete samples (pixels)

Reconstruct
- Reconstructed function

Transform
- Transformed function

Filter
- Bandlimited function

Sample
- Discrete samples (pixels)

Reconstruct
- Display

Continuous Function
Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display

Discrete Samples
Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display

Reconstructed Function
Image Processing

Real world

Sample
Discrete samples (pixels)

Reconstruct
Reconstructed function

Transform
Transformed function

Filter
Bandlimited function

Sample
Discrete samples (pixels)

Reconstruct
Display

Transformed Function
Image Processing

Real world

Sample

Discrete samples (pixels)

Reconstruct

Reconstructed function

Transform

Transformed function

Filter

Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display

Bandlimited Function
Image Processing

Real world → Sample → Discrete samples (pixels)

Sample → Reconstruct → Reconstructed function

Reconstruct → Transform → Transformed function

Transform → Filter → Bandlimited function

Filter → Sample → Discrete samples (pixels)

Sample → Reconstruct → Display

Discrete samples
Image Processing

Real world

Sample → Discrete samples (pixels)

Reconstruct

Transform → Reconstructed function

Transform → Transformed function

Filter → Bandlimited function

Sample → Discrete samples (pixels)

Reconstruct

Display
Image Processing

Real world

Sample
Discrete samples (pixels)

Reconstruct
Reconstructed function

Transform
Transformed function

Filter
Bandlimited function

Sample
Discrete samples (pixels)

Reconstruct
Display

• Ideal resampling requires correct filtering to avoid artifacts

• Reconstruction filter especially important when magnifying

• Bandlimiting filter especially important when minifying
Ideal Image Processing Filter

- Frequency domain

- Spatial domain

\[
\text{Sinc}(x) = \frac{\sin \pi x}{\pi x}
\]

Figure 4.5 Wolberg
Practical Image Processing

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

- Possible (poor) resampling implementation:

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

- Use nearest sample
Point Sampling

Point Sampled: Aliasing!  Correctly Bandlimited
Resampling with Filter

• Output is weighted average of inputs:

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

Source image  Destination image
Image Resampling

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

$k(ix,iy)$ represented by gray value
Image Resampling

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

\[ k(i,j) = \max(1 - \frac{d}{w}, 0) \]

Filter Width = 2
Image Resampling

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

Filter Width = 1

Width of filter affects blurriness

Triangle filter

(u,v)
Gaussian Filtering

- Kernel is Gaussian function

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

- Drops off quickly, but never gets to exactly 0
- In practice: compute out to \( w \approx 2.5\sigma \) or \( 3\sigma \)
Image Resampling

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

Filter Width $< 1$
Image Resampling (with width < 1)

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

\[ a \]  \[ b \]

\[
\begin{align*}
(u_1,v_2) & \quad a & \quad (u_2,v_2) \\
(u_1,v_1) & \quad b & \quad (u_2,v_1)
\end{align*}
\]

Filter Width < 1
Image Resampling (with width < 1)

- Alternative: force width to be at least 1

Filter Width < 1
Putting it All Together

• Possible implementation of image blur:

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

Increasing sigma
Putting it All Together

• Possible implementation of image scale:

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

Source image

Destination image

$$f((u,v)) \rightarrow (ix,iy)$$
Putting it All Together

- Possible implementation of image rotation:

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

![Image rotation diagram]
Sampling Method Comparison

- Trade-offs
  - Aliasing versus blurring
  - Computation speed
Forward vs. Reverse Mapping

• Reverse mapping:

\[
\text{Warp}(\text{src}, \text{dst}) \{
\text{for (int } ix = 0; ix < xmax; ix++) \{
\text{for (int } iy = 0; iy < ymax; iy++) \{
\text{float } w \approx 1 / \text{scale}(ix, iy);
\text{float } u = f^{-1}_{x}(ix, iy);
\text{float } v = f^{-1}_{y}(ix, iy);
\text{dst}(ix, iy) = \text{Resample}(\text{src}, u, v, w);
\}
\}
\}
\]
Forward vs. Reverse Mapping

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

• Forward mapping:

\[
\text{Warp}(\text{src}, \text{dst}) \{
\text{for (int } \text{iu} = 0; \text{iu} < \text{umax}; \text{iu}++) \{
\text{for (int } \text{iv} = 0; \text{iv} < \text{vmax}; \text{iv}++) \{
\text{float } x = f_x(\text{iu}, \text{iv});
\text{float } y = f_y(\text{iu}, \text{iv});
\text{float } w \approx 1 / \text{scale}(x, y);
\text{Splat}(\text{src}(\text{iu}, \text{iv}), x, y, k, w);
\}
\}
\}
\]

Source image \hspace{2cm} Destination image

(iu, iv) \hspace{2cm} (x, y)
Forward vs. Reverse Mapping

- Forward mapping:

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

Problem?
Forward vs. Reverse Mapping

- Forward mapping:
  
  ```c
  for (int iu = 0; iu < umax; iu++) {
    for (int iv = 0; iv < vmax; iv++) {
      float x = f_x(iu,iv);
      float y = f_y(iu,iv);
      float w = 1 / scale(x, y);
      for (int ix = xlo; ix <= xhi; ix++) {
        for (int iy = ylo; iy <= yhi; iy++) {
          dst(ix,iy) += k(x,y,ix,iy,w) * src(iu,iv);
          ksum(ix,iy) += k(x,y,ix,iy,w);
        }
      }
    }
  }
  
  for (ix = 0; ix < xmax; ix++)
    for (iy = 0; iy < ymax; iy++)
      dst(ix,iy) /= ksum(ix,iy)
  ```

Destination image (x,y)
Forward vs. Reverse Mapping

• Tradeoffs?
Forward vs. Reverse Mapping

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

• Mapping
  ◦ Forward vs. reverse
  ◦ Parametric vs. correspondences

• Sampling, reconstruction, resampling
  ◦ Frequency analysis of signal content
  ◦ Filter to avoid undersampling: point, triangle, Gaussian
  ◦ Reduce visual artifacts due to aliasing
    » Blurring is better than aliasing
Next Time…

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