Sampling, Resampling, and Warping

COS 426
Digital Image Processing

- Changing intensity/color
  - Linear: scale, offset, etc.
  - Nonlinear: gamma, saturation, etc.
  - Add random noise

- Filtering over neighborhoods
  - Blur
  - Detect edges
  - Sharpen
  - Emboss
  - Median

- Moving image locations
  - Scale
  - Rotate
  - Warp

- Combining images
  - Composite
  - Morph

- Quantization

- Spatial / intensity tradeoff
  - Dithering
Digital Image Processing

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

Continuous function

Discrete samples

Sampling
Sampling and Reconstruction

Sampling

Continuous function

Discrete samples

Reconstruction

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?
Sampling Theory

What happens when we use too few samples?

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

Figure 14.17 FvDFH
What happens when we use too few samples?

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

What happens when we use too few samples?

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

(Barely) adequate sampling

Inadequate sampling
Sampling Theory

How many samples are enough to avoid aliasing?

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

How many samples are enough to avoid aliasing?

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

How many samples are enough to avoid aliasing?

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

How many samples are enough to avoid aliasing?

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

How many samples are enough to avoid aliasing?

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

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

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

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

\[ f(x) \]

\[ |F(u)| \]

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 the original signal has no content $\geq 1/2$ the sampling frequency - Shannon

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

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

- Consider reducing the image resolution

Original image

1/4 resolution
Image Processing

- Image processing is a resampling problem
Sampling Theorem

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

**Aliasing** will occur if the signal is under-sampled

Under-sampling

Figure 14.17 FvDFH
Aliasing

• In general:
  ○ Artifacts due to under-sampling or poor reconstruction

• Specifically, in graphics:
  ○ Spatial aliasing
  ○ Temporal aliasing

Under-sampling

Figure 14.17 FvDFH
Spatial Aliasing

Artifacts due to limited spatial resolution
Spatial Aliasing

Artifacts due to limited spatial resolution

“Jaggies”
Temporal Aliasing

Artifacts due to limited temporal resolution
- Strobing
- Flickering
Temporal Aliasing

Artifacts due to limited temporal resolution

- Strobing
- Flickering
Temporal Aliasing

Artifacts due to limited temporal resolution
- Strobing
- Flickering
Temporal Aliasing

Artifacts due to limited temporal resolution
- Strobing
- Flickering
Antialiasing

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

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

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

Transform

Transformed function

Filter

Bandlimited function

Sample

Discrete samples (pixels)

Reconstruct

Display
Image Processing

1. Real world
2. Sample → Discrete samples (pixels)
3. Reconstruct
4. Transform → Reconstructed function
5. Transform → Transformed function
6. Filter → Bandlimited function
7. Sample → Discrete samples (pixels)
8. Reconstruct
9. 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)

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
Ideal Bandlimiting Filter

- Frequency domain
- Spatial domain

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

Figure 4.5 Wolberg
Practical Image Processing

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

![Diagram showing the process of practical image processing]

1. Real world
2. Sample
3. Discrete samples (pixels)
4. Reconstruct
5. Reconstructed function
6. Transform
7. Transformed function
8. Filter
9. Bandlimited function
10. Sample
11. Discrete samples (pixels)
12. Reconstruct
13. Display
Example: Scaling

- Resample with triangle or Gaussian filter
General Image Warping

- Move pixels of an image

Source image → Warp → Destination image
General Image Warping

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

 ![Source image](image1.png) ![Destination image](image2.png)
General Image Warping

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

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

Source image  Warp  Destination image
Two Options

• Forward mapping

• Reverse mapping
Mapping

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

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

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

Rotate 30
Parametric Mappings

• Shear in X by \textit{factor}:
  \begin{itemize}
  \item $x = u + \text{factor} \times v$
  \item $y = v$
  \end{itemize}

• Shear in Y by \textit{factor}:
  \begin{itemize}
  \item $x = u$
  \item $y = v + \text{factor} \times u$
  \end{itemize}
Other Parametric Mappings

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

Fish-eye

“Swirl”

“Rain”
COS426 Examples

Aditya Bhaskara

Weixiang
More COS426 Examples

Sid Kapur

Eirik Bakke

Michael Oranato
Point Correspondence Mappings

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

Warp
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)
Image Warping

- Image warping requires resampling of image
Point Sampling

- Possible (poor) resampling implementation:

```c
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
Image Resampling Pipeline

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

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

- Output is weighted average of inputs:

```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;
}
```
Image Resampling

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

\[
\text{dst}(ix,iy) = 0; \\
\text{for } (ix = u-w; ix <= u+w; ix++) \\
\quad \text{for } (iy = v-w; iy <= v+w; iy++) \\
\quad \quad d = \text{dist}(ix,iy) \leftrightarrow (u,v) \\
\quad \quad \text{dst}(ix,iy) += k(ix,iy) \times \text{src}(ix,iy);
\]

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

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

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

Filter Width = 2
Image Resampling

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

Width of filter affects blurriness

Filter Width = 1
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 \( w \sim 2.5\sigma \) or \( 3\sigma \)
Image Resampling

- What if width (w) is smaller than sample spacing?
Image Resampling (with width < 1)

- Reconstruction filter: Bilinearly interpolate four closest pixels
  - $a = \text{linear interpolation of src}(u_1,v_2) \text{ and src}(u_2,v_2)$
  - $b = \text{linear interpolation of src}(u_1,v_1) \text{ and src}(u_2,v_1)$
  - $\text{dst}(x,y) = \text{linear interpolation of “}a\text{” and “}b\text{”}$
Image Resampling (with width < 1)

- Alternative: force width to be at least 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 → (u,v) → f → (ix,iy) → Destination image
Putting it All Together

- Possible implementation of image rotation:

  \[
  \text{Rotate}(\text{src}, \text{dst}, \Theta) \{
  \quad w \approx 1
  \quad \text{for (int } ix = 0; \ ix < \text{xmax}; \ ix++) \{ \text{ }
  \quad \quad \text{for (int } iy = 0; \ iy < \text{ymax}; \ iy++) \{ \text{ }
  \quad \quad \quad \text{float } u = ix \times \cos(-\Theta) - iy \times \sin(-\Theta); \text{ }
  \quad \quad \quad \text{float } v = ix \times \sin(-\Theta) + iy \times \cos(-\Theta); \text{ }
  \quad \quad \quad \text{dst}(ix,iy) = \text{Resample}(\text{src},u,v,k,w); \text{ }
  \quad \quad \} \text{ }
  \quad \} \}
  \]
Sampling Method Comparison

• Trade-offs
  ◦ Aliasing versus blurring
  ◦ Computation speed

Point  Triangle  Gaussian
Forward vs. Reverse Mapping

• Reverse mapping:

\[
\text{Warp}(\text{src}, \text{dst}) \{
\text{for } (\text{int } ix = 0; ix < \text{xmax}; ix++) \{
\text{for } (\text{int } iy = 0; iy < \text{ymax}; iy++) \{
\text{float } w \approx 1 / \text{scale}(ix, iy);
\text{float } u = f_x^{-1}(ix, iy);
\text{float } v = f_y^{-1}(ix, iy);
\text{dst}(ix,iy) = \text{Resample}(\text{src},u,v,w);
\}
\}
\}
\]

(u,v)

\(f\)

(ix,iy)

Source image

Destination image

Source image

Destination image

(u,v)

\(f\)

(ix,iy)
Forward vs. Reverse Mapping

- Forward mapping:

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

• Forward mapping:

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

Source image \quad \text{(iu,iv)} \quad \text{Destination image} \quad \text{(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 = fx(iu,iv);
        float y = fy(iu,iv);
        float w = 1.0 / 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);
            }
        }
    }
}
```
Forward vs. Reverse Mapping

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

Destination image
Forward vs. Reverse Mapping

- Tradeoffs?
Forward vs. Reverse Mapping

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

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

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

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