Image Processing

5th~6th Weeks, 2008

Contents

- Quantization
  - Classical halftoning
  - Random dither
  - Ordered dither
  - Error diffusion dither
- Pixel operations
  - Adding random noise
  - Adding luminance
  - Adding contrast
  - Adding saturation
- Filtering
  - Blurring
  - Edge-detection
- Warping
  - Scaling
  - Rotation
  - Warping
- Combining
  - Compositing
  - Morphing
What is an Image?

- An image is a 2D rectilinear array of pixels

![Continuous image](image1)
![Digital image](image2)

Quantization

- Artifact due to limited intensity resolution
  - Frame buffers have limited number of bits per pixel
  - Physical devices have limited dynamic range

---

Computer Graphics

3

Computer Graphics

4
Uniform Quantization

\[ P(x, y) = \text{trunc}(I(x, y) + 0.5) \]

Image with decreasing bits per pixel:

- 8 bits
- 4 bits
- 2 bits
- 1 bit

Notice contouring
Reducing Effects of Quantization

- Halftoning and Dithering
  - Trading spatial resolution for intensity resolution
  - Reducing visual artifacts due to quantization

- Halftoning
  - Classical halftoning

- Dithering
  - Random dither
  - Ordered dither
  - Error diffusion dither

Classical Halftoning (1)

- Using dots of varying size to represent intensities
  - Area of dots proportional to intensity in image

\[ I(x, y) \] 

\[ P(x, y) \]
Classical Halftoning (2)

Newspaper image

From New York Times 9/21/99

Halftone Patterns (1)

- Using cluster of pixels to represent intensity
- Trade spatial resolution for intensity resolution

0 ≤ I < 0.2
0.2 ≤ I < 0.4
0.4 ≤ I < 0.6
0.6 ≤ I < 0.8
0.8 ≤ I ≤ 1.0
Halftone Patterns (2)

- How many intensities in a $n \times n$ cluster?

![Intensities in a $n \times n$ cluster](image)

Dithering

- Distributing errors among pixels
  - Exploiting spatial integration in our eye
  - Displaying greater range of perceptible intensities

![Comparative images](image)
Random Dither (1)

- Randomizing quantization errors
- Errors appear as noise

\[ P(x, y) = \text{trunc}(I(x, y) + \text{noise}(x, y) + 0.5) \]

Random Dither (2)

Original (8 bits)  | Uniform Quantization (1 bit)  | Random Dither (1 bit)
Ordered Dither (1)

- Pseudo-random quantization errors
  - Matrix stores pattern of thresholds

\[ D_2 = \begin{bmatrix} 3 & 1 \\ 0 & 2 \end{bmatrix} \]

\[
\begin{align*}
  j &= (x \mod n) + 1 \\
  i &= (y \mod n) + 1 \\
  e &= I(x, y) - D_n(i, j) \\
  \text{if}( e > 0 ) & \Rightarrow P(x, y) = \text{ceil}(I(x, y)) \rightarrow \text{turn on the pixel} \\
  \text{else} & \Rightarrow P(x, y) = \text{floor}(I(x, y)) \rightarrow \text{turn off the pixel}
\end{align*}
\]

Ordered Dither (2)

Original (8 bits)

Uniform Quantization (1 bit)

Ordered Dither (1 bit)
Error Diffusion Dither (1)

- Spread quantization error over neighbor pixels
- Error dispersed to pixels right and below

\[ \alpha + \beta + \gamma + \delta = 1.0 \]
Contents

- Quantization
  - Classical halftoning
  - Random dither
  - Ordered dither
  - Error diffusion dither

- Pixel operations
  - Adding random noise
  - Adding luminance
  - Adding contrast
  - Adding saturation

- Filtering
  - Blurring
  - Edge-detection

- Warping
  - Scaling
  - Rotation
  - Warping

- Combining
  - Compositing
  - Morphing

Adjusting Brightness

- Simplifying scale pixel components
  - Must clamp to range (e.g., 0 to 255)

Original

Brighter
Adjusting Contrast

- Computing mean luminance $L$ for all pixels
  - Luminance = $0.30*r + 0.59*g + 0.11*b$
- Scaling deviation from $L$ for each pixel component
  - Must clamp to range (e.g. 0 to 255)

Original

More contrast

Contents

- Quantization
  - Classical halftoning
  - Random dither
  - Ordered dither
  - Error diffusion dither
- Pixel operations
  - Adding random noise
  - Adding luminance
  - Adding contrast
  - Adding saturation
- Filtering
  - Blurring
  - Edge-detection
- Warping
  - Scaling
  - Rotation
  - Warping
- Combining
  - Compositing
  - Morphing
Adjusting Blurriness

- Convolving with a filter whose entries sum to one
  - Each pixel becomes a weighted average of its neighbors

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

Edge Detection

- Convolving with a filter that finds differences between neighbor pixels

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

- Quantization
  - Classical halftoning
  - Random dither
  - Ordered dither
  - Error diffusion dither

- Pixel operations
  - Adding random noise
  - Adding luminance
  - Adding contrast
  - Adding saturation

- Filtering
  - Blurring
  - Edge-detection

- Warping
  - Scaling
  - Rotation
  - Warping

- Combining
  - Compositing
  - Morphing

Image Warping

- Moving pixels of image
  - Step1: mapping
  - Step2: resampling

Source Image

Destination Image
Mapping

- Defining transformation
  - Describing the destination \((x, y)\) for every location \((u, v)\) in the source (or vice-versa, if invertible)

![Mapping Diagram]

Example Mappings (1)

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

![Example Mapping Diagram]
Example Mappings (2)

Rotating by $\theta$ degrees:

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

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

Example Mappings (3)

Shearing in X by factor $\text{factor}$:

$x = u + \text{factor} \times v$

$y = v$

Shearing in Y by factor $\text{factor}$:

$x = u$

$y = v + \text{factor} \times u$
Other Mappings

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

![Fish-eye](image1.png)  ![Swirl](image2.png)  ![Rain](image3.png)

Image Warping Implementation (1)

- Forward mapping:

```cpp
for(int u=0; u<umax; u++) {
    for(int v=0; v<vmax; v++) {
        float x = f_x(u,v);
        float y = f_y(u,v);
        dst(x,y) = src(u,v);
    }
}
```

![Source Image](source_image.png)  ![Destination Image](destination_image.png)
Forwarding Mapping

- Iterating over source image

![Diagram showing source and destination pixels before and after rotation]

Forwarding Mapping – NOT

- Iterating over source image

Many source pixels can map to same destination pixel

Some destination pixels may not be covered

![Diagram showing source and destination pixels before and after rotation, highlighting the difference in coverage]
Image Warping Implementation (2)

Reverse mapping:

```c
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);
    }
}
```

![Source Image](image-source)

![Destination Image](image-destination)

Reverse Mapping

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

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

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

\[\text{(u, v) \rightarrow (x, y)}\]

Source Image \hspace{1cm} Destination Image

Point Sampling

- Taking value at closest pixel

\[
\begin{align*}
\text{int } \text{iu} &= \text{trunc}(u+0.5); \\
\text{int } \text{iv} &= \text{trunc}(v+0.5); \\
\text{dst}(x, y) &= \text{src}(\text{iu}, \text{iv});
\end{align*}
\]

This method is simple, but it causes aliasing

\[\text{Rotate } -30 \text{ Scale } 0.5\]
Triangle Filtering

- Bilinearly interpolating four closest pixels
  - \( a \) = linear interpolation of \( \text{src}(u_1, v_2) \) and \( \text{src}(u_2, v_2) \)
  - \( b \) = linear interpolation of \( \text{src}(u_1, v_1) \) and \( \text{src}(u_2, v_1) \)
  - \( \text{dst}(x, y) \) = linear interpolation of “\( a \)” and “\( b \)”

Gaussian Filtering

- Computing weighted sum of pixel neighborhood
  - Weights are normalized values of Gaussian function
Filtering Methods Comparison

- Trade-offs
  - Aliasing versus blurring
  - Computation speed

Image Warping Implementation (3)

- Reverse mapping

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

Source Image → Destination Image
Example: Scaling

Scale (src, dst, sx, sy):

float w = \max(1/sx, 1/sy)
for(int x=0; x<xmax; x++) {
    for(int y=0; y<ymax; y++) {
        float u = x/sx;
        float v = y/sy;
        dst(x,y) = resample_src(u,v,w);
    }
}

Example: Rotation

Rotate (src, dst, theta):

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)
        dst(x,y) = resample_src(u,v,w);
    }
}
Example: Fun

- Swirl (src, dst, theta)

```c
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);
    }
}
```

Contents

- Quantization
  - Classical halftoning
  - Random dither
  - Ordered dither
  - Error diffusion dither

- Pixel operations
  - Adding random noise
  - Adding luminance
  - Adding contrast
  - Adding saturation

- Filtering
  - Blurring
  - Edge-detection

- Warping
  - Scaling
  - Rotation
  - Warping

- Combining
  - Compositing
  - Morphing
Image Compositing

- Separating an image into “elements”
  - Rendering independently
  - Compositing together

- Applications
  - Cel animation
  - Chroma-keying
  - Blue-screen matting

Dobkin meets Elvis

Blue-Screen Matting

- Compositing foreground and background images
  - Creating background image
  - Creating foreground image with blue background
  - Inserting non-blue foreground pixels into background
Alpha Channel

- Encoding pixel coverage information
  - $\alpha = 0$ : no coverage (or transparent)
  - $\alpha = 1$ : full coverage (or opaque)
  - $0 < \alpha < 1$ : partial coverage (or semi-transparent)

- Example: $\alpha = 0.3$

Compositing with Alpha

- Controlling the linear interpolation of foreground and background pixels when elements are composited
Image Composition Example

Jurassic Park

Image Morphing

- Animating transition between two images

Figure 16-9
Transformation of an STP oil can into an engine block. (Courtesy of Silicon Graphics, Inc.)
Cross-Dissolving

- Blending image with “over” operator
  - Alpha of bottom image is 1.0
  - Alpha of top image varies from 0.0 to 1.0

\[
\text{blend}(i, j) = (1 - t) \text{src}(i, j) + t \text{dst}(i, j) \quad (0 \leq t \leq 1)
\]

Image Morphing

- Combining warping and cross-dissolving
Image Morphing

- The warping step is the hard one
  - Aiming to align feature in images

How specify mapping for the warp?

Feature Based Warping

- Beier & Neeley use pairs of lines to specify warp
  - Given p in destination image, where is p' in source image?

Source image

Destination image

u is a fraction
v is a length (in pixels)
Warping with One Line Pair (1)

- What happens to the “F”?

Warping with One Line Pair (2)

- What happens to the “F”?

Translation

Scale
Warping with One Line Pair (3)

- What happens to the “F”?

Warping with One Line Pair (4)

- What happens to the “F”?

In general, similarity transformations
Warping with Multiple Line Pairs (1)

- Using weighted combination of points defined by each pair of corresponding lines

Warping with Multiple Line Pairs (2)

- Using weighted combination of points defined by each pair of corresponding lines

\[ P' \text{ is a weighted average} \]
Weighting Effect of Each Line Pair

To weight the contribution of each line pair:

\[
weight[i] = \left( \frac{(length[i])^p}{a + dist[i]} \right)^b
\]

Where

- \( length[i] \) is the length of the \( i \)-th line
- \( dist[i] \) is the distance from a point \( P \) to the \( i \)-th line
- \( a, b, p \) are constants that control the wrap

Warping Pseudocode

```plaintext
WarpImage(Image, L'[...], L[...])
Begin
    for each destination pixel \( p \) do
        psum = (0, 0)
        wsum = 0
        for each line \( L[i] \) in destination do
            \( p'[i] = p \) transformed by \( (L[i], L'[i]) \)
            psum += \( p'[i] \) * weight[i]
            wsum += weight[i]
        end
        p' = psum / wsum
        Result(p) = Image(p')
    end
end
```
Morphing Pseudocode

GenerateAnimation(Image₀, L₀[...], Image₁, L₁[...])
Begin
    for each intermediate frame time t do
        for i=1 to number of line pairs do
            L[i] = line t-th of the way from L₀[i] to L₁[i]
        end
        Warp₀ = WarpImage(Image₀, L₀, L)
        Warp₁ = WarpImage(Image₁, L₁, L)
        for each pixel p in FinalImage do
            Result(p) = (1-t)*Warp₀ + t*Warp₁
        end
    end
end

Beier & Neeley Example (1)

Image 0
Warp 0
Image 1
Warp 1
Result
Beier & Neeley Example (2)