Last time: edge dection
Lat time: edge detection

• Edge detection algorithms typically proceed in three or four steps:
  – **Filtering**: cut down on noise
  – **Enhancement**: amplify the difference between edges and non-edges
  – **Detection**: use a threshold operation
  – **Localization** (optional): estimate geometry of edges, which generally pass between pixels
The Canny Edge Detector

original image (Lena)
The Canny Edge Detector

magnitude of the gradient
The Canny Edge Detector

After non-maximum suppression
Lat time: edge detection

• Edge detection algorithms typically proceed in three or four steps:
  
  – **Filtering**: cut down on noise
  
  – **Enhancement**: amplify the difference between edges and non-edges
  
  – **Detection**: use a threshold operation
  
  – **Localization** (optional): estimate geometry of edges, which generally pass between pixels
Last time: Mattes

- A *matte* is an image that shows which parts of another image are foreground objects
- Term dates from film editing and cartoon production
- How would I use a matte to insert an object into a background?
- How are mattes usually generated for television?
Basic Compositing Operation

- At each pixel, combine the pixel data from $f$ and the pixel data from $g$ with the equation:

\[
\begin{align*}
c_f &= [\alpha_f R_f, \alpha_f G_f, \alpha_f B_f, \alpha_f ] \\
c_g &= [\alpha_g R_g, \alpha_g G_g, \alpha_g B_g, \alpha_g ] \\
c_o &= 1 \cdot c_f + (1 - \alpha_f) \cdot c_g
\end{align*}
\]

"Over" Operator
Last time: 2D Transformations

- Transformation $T$ is a coordinate-changing machine: $p' = T(p)$
- What does it mean that $T$ is global?
  - can be described by just a few numbers (parameters)
  - the parameters are the same for any point $p$
- Represent $T$ as a matrix: $p' = Mp$
  \[
  \begin{bmatrix}
  x' \\
  y'
  \end{bmatrix}
  = \begin{bmatrix}
  x \\
  y
  \end{bmatrix}
  \]
Scaling

• *Non-uniform scaling*: different scalars per component:

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
2x \\
0.5y
\end{bmatrix}
\]

\[x \times 2, \quad y \times 0.5\]
2-D Rotation

\[ x' = x \cos(\theta) - y \sin(\theta) \]
\[ y' = x \sin(\theta) + y \cos(\theta) \]
2x2 Matrices

• What types of transformations can be represented with a 2x2 matrix?

2D Shear?

\[ x' = x + s h_x \cdot y \]
\[ y' = s h_y \cdot x + y \]

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
1 & s h_x \\
s h_y & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]
2x2 Matrices

• What types of transformations can be represented with a 2x2 matrix?

**2D Mirror about Y axis?**

\[
x' = -x \\
y' = y
\]

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

**2D Mirror over (0,0)?**

\[
x' = -x \\
y' = -y
\]

\[
\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]
All 2D Linear Transformations

• Linear transformations are combinations of …
  – Scale,
  – Rotation,
  – Shear, and
  – Mirror

• Any 2D transform can be decomposed into the product of a rotation, scale, and a rotation

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}
\]

\[
\begin{bmatrix}
  1 & 1 \\
  0 & 1
\end{bmatrix} = \text{rotate}(31.7^\circ) \cdot \text{scale}(1.618,0.618) \cdot \text{rotate}(-58.3^\circ)
\]
All 2D Linear Transformations

• Linear transformations are combinations of …
  – Scale,
  – Rotation,
  – Shear, and
  – Mirror

• A symmetric 2D transform can be decomposed into the product of a rotation, scale, and the inverse rotation

\[
\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}
\]

\[
\begin{bmatrix} 2 & 1 \\ 1 & 1 \end{bmatrix} = \text{rotate}(31.7^\circ) \cdot \text{scale}(2.618,0.382) \cdot \text{rotate}(-31.7^\circ)
\]
Today

• More on 2D transformation
• Use it for image warping and morphing
All 2D Linear Transformations

- Linear transformations are combinations of …
  - Scale,
  - Rotation,
  - Shear, and
  - Mirror

\[
\begin{bmatrix}
x' \\
y'
\end{bmatrix} = \begin{bmatrix}
a & b \\
c & d
\end{bmatrix} \begin{bmatrix}
x \\
y
\end{bmatrix}
\]

- Properties of linear transformations:
  - Origin maps to origin
  - Lines map to lines
  - Parallel lines remain parallel
  - Ratios are preserved
  - Closed under composition

\[
\frac{AB}{BC} = \frac{A'B'}{B'C'} \quad \text{if } A, B, C \text{ are on a line}
\]
2x2 Matrices

• What types of transformations can not be represented with a 2x2 matrix?

2D Translation?

\[ x' = x + t_x \]
\[ y' = y + t_y \]

Only linear 2D transformations can be represented with a 2x2 matrix
Translation

• Example of translation

\[
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix} \rightarrow \begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

Homogeneous Coordinates

\[
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
1 & 0 & t_x \\
0 & 1 & t_y \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
1
\end{bmatrix} = \begin{bmatrix}
x + t_x \\
y + t_y \\
1
\end{bmatrix}
\]

t_x = 2

\( t_y = 1 \)
Homogeneous coordinates

• Why do we need it?
  – Can express all linear transformation as special cases
Homogeneous coordinates

• Why do we need it?
  – Can express all linear transformation as special cases
  – Easy to compute a composite transformation that involve several translations and linear transformation
Homogeneous coordinates

• Why do we need it?
  – Can express all linear transformation as special cases
  – Easy to compute a composite transformation that involve several translations and linear transformation
  – More to come
Affine Transformations

• Affine transformations are combinations of …
  – Linear transformations, and
  – Translations

• Properties of affine transformations:
  – Origin does not necessarily map to origin
  – Lines map to lines
  – Parallel lines remain parallel
  – Ratios are preserved
  – Closed under composition
  – Models change of basis

\[
\begin{bmatrix}
  x' \\
y' \\
1
\end{bmatrix} =
\begin{bmatrix}
a & b & c \\
d & e & f \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]
Image warping

- Given a coordinate transform $\mathbf{x}' = T(\mathbf{x})$ and a source image $I(\mathbf{x})$, how do we compute a transformed image $I'(\mathbf{x}') = I(T(\mathbf{x}))$?
Forward warping

- Send each pixel $I(x)$ to its corresponding location $x' = T(x)$ in $I'(x')$
Forward warping

\[ \text{fwarp}(I, I', T) \]
\[
\{ \\
\text{for}\ (y=0; \ y<I.\text{height}; \ y++) \\
\text{for}\ (x=0; \ x<I.\text{width}; \ x++) \ { \\
(x',y')=T(x,y); \\
I'(x',y')=I(x,y); \\
}\}
\]
Forward warping

- Send each pixel $I(x)$ to its corresponding location $x' = T(x)$ in $I'(x')$
- What if pixel lands “between” two pixels?
- Will be there holes?
- Answer: add “contribution” to several pixels, normalize later (splatting)
Forward warping

\[ \text{fwarp}(I, I', T) \]
\[
\begin{array}{c}
\{ \\
\quad \text{for } (y=0; y<I.\text{height}; y++) \\
\quad \quad \text{for } (x=0; x<I.\text{width}; x++) \{ \\
\quad \quad \quad (x',y')=T(x,y); \\
\quad \quad \quad \text{Splatting}(I',x',y',I(x,y),\text{kernel}); \\
\quad \}
\}
\end{array}
\]
Splatting

• Computed weighted sum of contributed colors using a kernel function, where weights are normalized values of filter kernel $k$, such as Gauss

May get a blurry image!

for all $q$
\[
q.color = 0;
q.weight = 0;
\]

for all $p$ from source image
for all $q$’s dist < radius
\[
d = \text{dist}(p, q);
\]
\[
w = \text{kernel}(d); \quad e^{\frac{d^2}{2\sigma^2}}
\]
\[
q.color += w*p;
q.weight += w;
\]

for all $q$
\[
q.Color /= q.weight;
\]
Inverse warping

- Get each pixel $I'(x')$ from its corresponding location $x = T^{-1}(x')$ in $I(x)$
Inverse warping

\texttt{iwarp(I, I', T)}

\begin{verbatim}
{ 
    for (y=0; y<I'.height; y++)
        for (x=0; x<I'.width; x++) {
            (x,y)=T^{-1}(x',y');
            I'(x',y')=I(x,y);
        }
}
\end{verbatim}

\begin{figure}
\centering
\includegraphics[width=\textwidth]{image}
\caption{Illustration of the inverse warping process.}
\end{figure}
Inverse warping

• Get each pixel $I'(x')$ from its corresponding location $x = T^{-1}(x')$ in $I(x)$

• What if pixel comes from “between” two pixels?
• Answer: resample color value from interpolated source image
Inverse warping

\[ \text{iwarp}(I, I', T) \]

\[
\begin{align*}
&\text{for (y=0; y<I'.height; y++)} \\
&\text{for (x=0; x<I'.width; x++)} \\
&(x,y) = T^{-1}(x', y') \\
&I'(x', y') = \text{Reconstruct}(I, x, y, \text{kernel});
\end{align*}
\]
Reconstruction (interpolation)

• Possible reconstruction filters (kernels):
  – nearest neighbor
  – bilinear
  – bicubic
  – sinc
Bilinear interpolation (tent filter)

- A simple method for resampling images

\[ f(x, y) = (1 - a)(1 - b) f[i, j] + a(1 - b) f[i + 1, j] + ab f[i + 1, j + 1] + (1 - a)b f[i, j + 1] \]

What might be the problem of bilinear interpolation?
Non-parametric image warping
Non-parametric image warping

• Specify a more detailed warp function
Non-parametric image warping

• Specify a more detailed warp function
• Tabulate pixel motion (lookup table)
Non-parametric image warping

- Mappings implied by correspondences
- Inverse warping
Non-parametric image warping
Warping between two triangles

- Idea: find an affine that transforms ABC to A’B’C’

\[
\begin{bmatrix}
a & b & e \\
c & d & f \\
0 & 0 & 1 \\
\end{bmatrix}
\]
Warping between two triangles

• Idea: find an affine that transforms ABC to A’B’C’
• 6 unknowns, 6 equations

\[
\begin{bmatrix}
a & b & e \\
c & d & f \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_A \\
y_A \\
1
\end{bmatrix}
=
\begin{bmatrix}
x_{A'} \\
y_{A'} \\
1
\end{bmatrix}
\]

\[
\begin{bmatrix}
a & b & e \\
c & d & f \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_B \\
y_B \\
1
\end{bmatrix}
=
\begin{bmatrix}
x_{B'} \\
y_{B'} \\
1
\end{bmatrix}
\]

\[
\begin{bmatrix}
a & b & e \\
c & d & f \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_C \\
y_C \\
1
\end{bmatrix}
=
\begin{bmatrix}
x_{C'} \\
y_{C'} \\
1
\end{bmatrix}
\]
Warping between two triangles

- Idea: find an affine that transforms ABC to A’B’C’
- 6 unknowns, 6 equations
- A more direct way
Barycentric coordinates

- Idea: represent $P$ using $A_1, A_2, A_3$

$$P - A_1 = \beta \cdot (A_2 - A_1) + \gamma \cdot (A_3 - A_1)$$

$$P = (1 - \beta - \gamma) \cdot A_1 + \beta \cdot A_2 + \gamma \cdot A_3$$

$$P = t_1 \cdot A_1 + t_2 \cdot A_2 + t_3 \cdot A_3$$

$$t_1 + t_2 + t_3 = 1$$
Barycentric coordinates

- Idea: represent $P$ using $A_1, A_2, A_3$

$$P - A_1 = \beta \cdot (A_2 - A_1) + \gamma \cdot (A_3 - A_1)$$

$$P = (1 - \beta - \gamma) \cdot A_1 + \beta \cdot A_2 + \gamma \cdot A_3$$

$$\begin{bmatrix} x_p \\ y_p \end{bmatrix} = t_1 \cdot \begin{bmatrix} x_{A_1} \\ y_{A_1} \end{bmatrix} + t_2 \cdot \begin{bmatrix} x_{A_2} \\ y_{A_2} \end{bmatrix} + t_3 \cdot \begin{bmatrix} x_{A_3} \\ y_{A_3} \end{bmatrix}$$

$$t_1 + t_2 + t_3 = 1$$

$$t_1 = \frac{\text{area}(PA_2A_3)}{\text{area}(A_1A_2A_3)}$$

$$t_2 = \frac{\text{area}(PA_3A_1)}{\text{area}(A_1A_2A_3)}$$

$$t_3 = \frac{\text{area}(PA_1A_2)}{\text{area}(A_1A_2A_3)}$$
Barycentric coordinates

- Idea: represent $P$ using $A_1, A_2, A_3$

\[
P - A_1 = \beta \cdot (A_2 - A_1) + \gamma \cdot (A_3 - A_1)
\]

\[
P = (1 - \beta - \gamma) \cdot A_1 + \beta \cdot A_2 + \gamma \cdot A_3
\]

\[
P = t_1 \cdot A_1 + t_2 \cdot A_2 + t_3 \cdot A_3
\]

\[t_1 + t_2 + t_3 = 1\]

\[
t_1 = \frac{\text{area}(PA_2A_3)}{\text{area}(A_1A_2A_3)}
\]

\[
t_2 = \frac{\text{area}(PA_3A_1)}{\text{area}(A_1A_2A_3)}
\]

\[
t_3 = \frac{\text{area}(PA_1A_2)}{\text{area}(A_1A_2A_3)}
\]
Non-parametric image warping

\[ P = w_A A + w_B B + w_C C \]

\[ P' = w_A A' + w_B B' + w_C C' \]

Barycentric coordinate

Turns out to be equivalent to affine transform