Global Flow

- Dominant Motion in the scene
  - Motion of all points in the scene
  - Motion of most of the points in the scene
  - A Component of motion of all points in the scene

- Global Motion is caused by
  - Motion of sensor (Ego Motion)
  - Motion of a rigid scene

- Estimation of Global Motion can be used to
  - Video Mosaics
  - Image Alignment (Registration)
  - Removing Camera Jitter
  - Tracking (By neglecting camera motion)
  - Video Segmentation etc.
Global Flow

Application: Image Alignment

Estimation of Global Flow

Assume Affine Flow: \[
\begin{bmatrix}
u \\ v
\end{bmatrix} = A \begin{bmatrix} x \\ y \end{bmatrix} + B
\]
\[
u = v_x = a_1 x + a_2 y + b_1
\]
\[
v = v_y = a_3 x + a_4 y + b_2
\]
\[
\begin{bmatrix}
u \\ v
\end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & b_1 \\ 0 & 0 & 0 & x & y & 1 & a_1 \\ a_2 \\ a_3 \\ a_4 \\ b_2
\end{bmatrix}
\]
\[
U = Xa
\]

Estimation of Global Flow

\[
(\nabla I)^T [\begin{bmatrix} u \\ v \end{bmatrix}] + I_t = 0 \quad \text{(Optical Flow Constraint Equation)}
\]

\[
(\nabla I)^T X a + I_t = 0
\]

\[
\sum_{(x,y) \in T} [(\nabla I)^T X a + I_t]^2 = 0
\]

\[
\min \sum_{(x,y) \in T} [(\nabla I)^T X a + I_t]^2 = \left[ \sum_{(x,y) \in T} X^T (\nabla I)^T X \right] a = - \sum_{(x,y) \in T} I_t X^T \nabla I
\]

\[
A a = B
\]

---

Estimation of Global Flow

**Single Iteration**

- Compute A and B
- Solve Aa = B
- Warp by a

Image ‘t’

Image ‘t+1’
Estimation of Global Flow

Initial Estimate \( a = [a_1, a_2, b_1, a_3, a_4, b_2]^T \)

Image ‘t’
Warp by \( a \)

Compute \( A \) and \( B \)
Solve \( A\tilde{a} = B \)
Warp by \( \delta a \)

Image ‘t+1’

Estimation of Global Flow

Parameters Update

Initial Estimate \( a = [a_1, a_2, b_1, a_3, a_4, b_2]^T \)

Computed Parameters \( \delta a = [\delta a_1, \delta a_2, \delta a_3, \delta a_4, \delta a_5]^T \)

Update Equations

\[
\begin{align*}
a_1 &= a_1\delta a_1 + a_2\delta a_2 + a_3 + \delta a_1 \\
a_2 &= a_1\delta a_1 + a_2\delta a_2 + a_3 + \delta a_2 \\
b_1 &= b_1\delta a_1 + b_2\delta a_2 + b_3 + \delta b_1 \\
a_3 &= a_1\delta a_1 + a_2\delta a_2 + a_3 + \delta a_3 \\
a_4 &= a_1\delta a_1 + a_2\delta a_2 + a_3 + \delta a_4 \\
b_2 &= b_1\delta a_1 + b_2\delta a_2 + b_3 + \delta b_2
\end{align*}
\]
Estimation of Global Flow

Initial Estimate $\mathbf{a} = [a_1, a_2, b_1, a_3, a_4, b_2]^T$

Iterative Algorithm

1. Given $\text{Im}_1$, $\text{Im}_2$ and Initial Estimate $\mathbf{a}$.
2. Warp $\text{Im}_1$ towards $\text{Im}_2$ using the Estimate $\mathbf{a}$ and let $\text{Im}_1$ be the warped image
   - Use image warping techniques (to be covered later)
3. Compute $\mathbf{A}$ and $\mathbf{B}$ by using $\text{Im}_1$ and $\text{Im}_2$
4. Estimate global flow by solving linear system $\mathbf{A}\hat{\mathbf{a}} = \mathbf{B}$
5. Update Parameters and let the updated parameters be $\mathbf{a}$.
6. Repeat until convergence or a fixed number of iterations
Coarse-to-fine global flow estimation

image H
Gaussian pyramid of image H

u=10 pixels

u=5 pixels

u=2.5 pixels

u=1.25 pixels

image I
Gaussian pyramid of image I

Coarse-to-fine global flow estimation

Compute Flow Iteratively

warp & upsample

Compute Flow Iteratively

...
Coarse-to-fine global motion estimation

Basic Components

- Pyramid Construction
- Motion Estimation
- Image Warping
- Coarse to Fine Refinement
Result of Global Motion Estimation

Image ‘t’  

Image ‘t+1’  

output  

4 Pyramids Level  
5 Iterations/Pyramid Level

Video Mosaic
Image Warping

\[
\begin{bmatrix}
  u \\
  v
\end{bmatrix} = \begin{bmatrix}
  x' - x \\
  y' - y
\end{bmatrix} = \begin{bmatrix}
  a_1 & a_2 \\
  a_3 & a_4
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix} + \begin{bmatrix}
  b_1 \\
  b_2
\end{bmatrix}
\]

\[
\Rightarrow \begin{bmatrix}
  x' \\
  y'
\end{bmatrix} = \begin{bmatrix}
  1 + a_1 & a_2 \\
  a_3 & 1 + a_4
\end{bmatrix} \begin{bmatrix}
  x \\
  y
\end{bmatrix} + \begin{bmatrix}
  b_1 \\
  b_2
\end{bmatrix}
\]
Interpolation

1-D Interpolation

\[ y = mx + c \]
\[ f(x) = mx + c \]

Interpolation

2-D Interpolation

\[ f(x, y) = a_0 + a_1 x + a_2 y + a_3 xy \]  \textbf{Bilinear}

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Interpolation

Bi-linear Interpolation

Four nearest points of (x,y) are:

\((x, y), (x, y), (x, y), (x, y)\)
\((3,5), (4,5), (3,6), (4,6)\)

\(x = \text{int}(x)\)
\(y = \text{int}(y)\)
\(x = x + 1\)
\(y = y + 1\)

Interpolation

\[ f'(x, y) = \varepsilon_x \varepsilon_y f(x, y) + \varepsilon_x \varepsilon_y f(x, y) + \varepsilon_x \varepsilon_y f(x, y) + \varepsilon_x \varepsilon_y f(x, y) \]

\(\varepsilon_x = \bar{x} - x\)
\(\varepsilon_y = \bar{y} - y\)
\(\varepsilon_x = x - \bar{x}\)
\(\varepsilon_y = y - \bar{y}\)
**Image Warping**

\[
\begin{bmatrix}
    x' \\
    y'
\end{bmatrix} = \begin{bmatrix}
    1 + a_1 & a_2 \\
    a_3 & 1 + a_4
\end{bmatrix} \begin{bmatrix}
    x \\
    y
\end{bmatrix} + \begin{bmatrix}
    b_1 \\
    b_2
\end{bmatrix}
\]

\[
\Rightarrow \begin{bmatrix}
    x \\
    y
\end{bmatrix} = \begin{bmatrix}
    1 + a_1 & a_2 \\
    a_3 & 1 + a_4
\end{bmatrix}^{-1} \left( \begin{bmatrix}
    x' \\
    y'
\end{bmatrix} - \begin{bmatrix}
    b_1 \\
    b_2
\end{bmatrix} \right)
\]

---

**Motion Tracking**
Motion tracking

Suppose we have more than two images

- How to track a point through all of the images?
  - In principle, we could estimate motion between each pair of consecutive frames
  - Given point in first frame, follow arrows to trace out its path
  - Problem: DRIFT
    - Small errors will tend to grow and grow over time—the point will drift way off course

Feature Tracking

- Choose only the points (“features”) that are easily tracked
- How to find these features?
  - Windows where \( \sum \nabla I(\nabla I)^T \) has two large eigenvalues
- Called the Harris Corner Detector

Feature Detection
Tracking features

Feature tracking
• Compute optical flow for that feature for each consecutive H, I

When will this go wrong?
• Occlusions—feature may disappear
  – need mechanism for deleting, adding new features
• Changes in shape, orientation
  – allow the feature to deform
• Changes in color
• Large motions
  – will pyramid techniques work for feature tracking?

Handling large motions

L-K requires small motion
• If the motion is much more than a pixel, use discrete search instead

<table>
<thead>
<tr>
<th>H(x, y)</th>
<th>I(x, y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>W</td>
<td></td>
</tr>
</tbody>
</table>

• Given feature window W in H, find best matching window in I
• Minimize sum squared difference (SSD) of pixels in window

\[
\min_{(u,v)} \left\{ \sum_{(x,y) \in W} |I(x + u, y + v) - H(x, y)|^2 \right\}
\]

• Solve by doing a search over a specified range of (u,v) values
  – this (u,v) range defines the search window
Tracking Over Many Frames

Feature tracking with m frames
1. Select features in first frame
2. Given feature in frame i, compute position in i+1
3. Select more features if needed
4. \( i = i + 1 \)
5. If \( i < m \), go to step 2

Issues
• Discrete search vs. Lucas Kanade?
  – depends on expected magnitude of motion
  – discrete search is more flexible
• Compare feature in frame i to i+1 or frame 1 to i+1?
  – affects tendency to drift...
• How big should search window be?
  – too small: lost features. Too large: slow

Incorporating Dynamics

Idea
• Can get better performance if we know something about the way points move
• Most approaches assume constant velocity
  \[
  \begin{align*}
  \dot{x}_{i+1} &= \dot{x}_i \\
  x_{i+1} &= 2x_i - x_{i-1}
  \end{align*}
  \]
  or constant acceleration
  \[
  \begin{align*}
  \ddot{x}_{i+1} &= \ddot{x}_i \\
  x_{i+1} &= 3x_i - 3x_{i-1} + x_{i-2}
  \end{align*}
  \]
• Use above to predict position in next frame, initialize search