Global Flow

- Estimate motion using all pixels in the image.
- Parametric flow gives an equation, which describes optical flow for each pixel.
  - Affine
  - Projective
- Global motion can be used to
  - generate mosaics
  - Object-based segmentation

Global Motion
Affine

\[
\begin{align*}
    u(x, y) &= a_1 x + a_2 y + b_1 \\
    v(x, y) &= a_3 x + a_4 y + b_2 \\
    U &= X - X'
\end{align*}
\]
Affine

\[ u(x, y) = a_1 x + a_2 y + b_1 \]
\[ v(x, y) = a_3 x + a_4 y + b_2 \]

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

Spatial Transformations

- translation
- rotation
- shear
- Rigid (rotation and translation)
- affine
Bergan et al

\[ u(x, y) = a_1 x + a_2 y + b_1 \]

\[ v(x, y) = a_3 x + a_4 y + b_2 \]

• Affine

\[
\begin{bmatrix}
  u(x, y) \\
  v(x, y)
\end{bmatrix}
= 
\begin{bmatrix}
  x & y & 1 & 0 & 0 & 0 \\
  0 & 0 & 0 & x & y & 1
\end{bmatrix}
\begin{bmatrix}
  a_1 \\
  a_2 \\
  b_1 \\
  a_3 \\
  a_4 \\
  b_2
\end{bmatrix}
\]

\[ u(x) = X(x)a \]

Bergan et al

Optical flow constraint eq

\[ f_x u + f_y v = -f_t \]

\[ E(u) = \sum_{\forall x \in f(x, y)} (f_t + f_x^T u)^2 \]

\[ E(a) = \sum_{\forall x \in f(x, y)} (f_t + f_x^T X(x)a)^2 \]

\[ E(\delta a) = \sum_{\forall x \in f(x, y)} (f_t + f_x^T X\delta a)^2 \]
Bergan et al

\[ E(\delta a) = \sum_{\forall x \in f(x,y)} (f_i + f_x^T X \delta a)^2 \]

\[ \min \sum X^T (f_x (f_x)^T X) \delta a = -\sum X^T f_x f_i \]

\[ Aa = B \]

Linear system

Basic Components

• Pyramid construction
• Motion estimation
• Image warping
• Coarse-to-fine refinement
Coarse-to-fine global flow estimation

Gaussian pyramid of image H

Gaussian pyramid of image I

Coarse-to-fine global flow estimation

Compute Flow Iteratively

warp & upsample

Compute Flow Iteratively

image H

image I

Gaussian pyramid of image H

Gaussian pyramid of image I
Estimation of Global Flow

Single Iteration

Compute A and B

Solve $Aa = B$

Image ‘t’

Warp by a

Image ‘t+1’
Image Warping

Warping an image $f$ into image $h$ using some transformation $g$, involves mapping intensity at each pixel $(x, y)$ in image $f$ to a pixel $(g(x), g(y))$ in image $h$ such that

$$(x', y') = (g(x), g(y))$$

In case of affine transformation, is transformed to as:

$$x' = Ax + b$$

Displacement model

$$U = x - x' = Ax + b$$

Instantaneous model

Image Warping (Bergan et al)

$X' = X - U = X - (AX + b)$

$X' = (I - A)X - b$

$X' = A'X - b$

$X' + b = A'X$

$(A')^{-1}(X' + b) = X$

$$\downarrow \text{warp}$$

$(A')^{-1}(X' + b) = X''$

$X'' = (I - A'X) - b$

$X'' = A''X - b$

$X'' + b = A''X$

$(A'')^{-1}(X'' + b) = X''$
Image Warping

• How about values in $X'' = (x'', y'')$ are not integer.
• But image is sampled only at integer rows and columns.

• Instead of converting $x''$ to $x'$ and copying $f(X', t-1)$ at $f(X'', t-1)$ we can convert integer values $x''$ to $x'$ and copy $f(X', t-1)$ at $f(X'', t-1)$.

\[ X' = X - U \]
\[ X' = X - (AX + b) \]
\[ X' = X'' - (AX'' + b) \]
Image Warping

• But how about the values in $X'$ are not integer.

• Perform bilinear interpolation to compute $f(X', t-1)$ at non-integer values.
Video Mosaic

Sprite
Mann & Picard

Projective

Projective Flow (weighted)

\[ u f_x + v f_y + f_t = 0 \]
\[ u^T f_x + f_t = 0 \]

Optical Flow const. equation

\[ x' = \frac{A x + b}{C^T x + 1} \]

Projective transform

\[ u = x' - x = \frac{A x + b}{C^T x + 1} - x \]
Projective Flow (weighted)

\[
\mathcal{E}_{\text{flow}} = \sum (u^T f_x + f_t)^2 \\
= \sum ((\frac{Ax + b}{C} - x)^T (Ax + f) + f_t)^2 \\
= \sum ((Ax + b - (C^T + 1)x)^T f_x + (C^T + 1)f_t)^2
\]

\[\text{minimize}\]

Projective Flow (weighted)

\[
(\sum \phi \phi^T) a = \sum (x^T f_x - f_t) \phi \\
a = [a_1, a_2, b_1, a_3, a_4, b_2, c_1, c_2]^T \\
\phi' = [f_x, f_y, f_z, f_x, f_y, f_z, a_1 - x^2 - y^2 - x y f_y, y f_x - x y f_y - y^2 f_y]
\]
Projective Flow (unweighted)

\[ x' = \frac{Ax + b}{C^T x + 1} \]

Pseudo-Perspective

\[ x + u = a_1 + a_2 x + a_3 y + a_4 x^2 + a_5 xy \]
\[ y + v = a_6 + a_7 x + a_8 y + a_4 xy + a_5 y^2 \]
Bilinear

\[ x' = \frac{A x + b}{C^T x + 1} \]

Taylor Series & remove
Square terms

\[ u + x = a_1 + a_2 x + a_3 y + a_4 xy \]
\[ v + y = a_5 + a_6 x + a_7 y + a_8 xy \]

Projective Flow (unweighted)

\[ \mathcal{E}_{flow} = \sum (u^T f_X + f_t)^2 \]

Minimize
Bilinear and Pseudo-Perspective

\[
(\sum \Phi \Phi^T)q = -\sum f_i \Phi
\]

\[
\Phi^T = [f_x(xy, x, y, 1), f_y(xy, x, y, 1)] \text{ bilinear}
\]

\[
\Phi^T = [f_x(x, y, 1), f_y(x, y, 1), c_1, c_2]
\]

\[
c_1 = x^2 f_x + xy f_x
\]

\[
c_2 = xy f_x + y^2 f_y
\]

Algorithm-1

- Estimate “q” (using approximate model, e.g. bilinear model).
- Relate “q” to “p”
  - select four points S1, S2, S3, S4
  - apply approximate model using “q” to compute \((x'_k, y'_k)\)
  - estimate exact “p”: 
True Projective

\[ x' = \frac{a_1 x + a_2 y + b_1}{c_1 x + c_2 y + 1} \]
\[ y' = \frac{a_3 x + a_4 y + b_1}{c_1 x + c_2 y + 1} \]

\[
\begin{bmatrix} x'_k \\ y'_k \end{bmatrix} = \begin{bmatrix} x_k & y_k & 1 & 0 & 0 & 0 & -x_k x'_k & -y_k x'_k \\ 0 & 0 & 0 & x_k & y_k & 1 & -x_k y'_k & -y_k y'_k \end{bmatrix} \begin{bmatrix} a_1 & a_2 & b_1 & a_3 & a_4 & b_2 & c_1 & c_1 \end{bmatrix}'
\]
Perform least squares fit to compute $a$.

\[
\begin{bmatrix}
  x'_1 \\
y'_1
\end{bmatrix} = \begin{bmatrix}
  x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x'_1 & -y_1x'_1 \\
  0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y'_1 & -y_1y'_1
\end{bmatrix} \begin{bmatrix}
  a
\end{bmatrix}
\]

\[
\begin{bmatrix}
  x'_k \\
y'_k
\end{bmatrix} = \begin{bmatrix}
  x_k & y_k & 1 & 0 & 0 & 0 & -x_kx'_k & -y_kx'_k \\
  0 & 0 & 0 & x_k & y_k & 1 & -x_ky'_k & -y_ky'_k
\end{bmatrix} \begin{bmatrix}
  a
\end{bmatrix}
\]

$P = Aa$

Perform least squares fit to compute $a$.

**Final Algorithm**

- A Gaussian pyramid of three or four levels is constructed for each frame in the sequence.
- The parameters “p” are estimated at the top level of the pyramid, between the two lowest resolution images, “g” and “h”, using algorithm-1.
Final Algorithm

• The estimated "p" is applied to the next higher resolution image in the pyramid, to make images at that level nearly congruent.
• The process continues down the pyramid until the highest resolution image in the pyramid is reached.

Video Mosaics

• Mosaic aligns different pieces of a scene into a larger piece, and seamlessly blend them.
  – High resolution image from low resolution images
  – Increased filed of view
Steps in Generating A Mosaic

- Take pictures
- Pick reference image
- Determine transformation between frames
- Warp all images to the same reference view

Applications of Mosaics

- Virtual Environments
- Computer Games
- Movie Special Effects
- Video Compression
Steve Mann

Author's 'wearable computer/personal imaging' system

1980  Mid 1980s  Early 1990s  Mid 1990s  Late 1990s

Sequence of Images
Projective Mosaic

Affine Mosaic
Building

Wal-Mart
Scientific American Frontiers
Head-mounted Camera at Restaurant

MIT Media Lab
References

• http://wearcam.org/pencigraphy (C code for generating mosaics)
• The Laplacian Pyramid as a compact code, Burt and Adelson, IEEE Trans on Communication, 1983.

References

Szeliski

Projective

\[ x' = \frac{a_1 x + a_2 y + b_1}{c_1 x + c_2 y + 1} \]
\[ y' = \frac{a_3 x + a_4 y + b_2}{c_1 x + c_2 y + 1} \]
Szeliski

\[
x' = \frac{a_1 x + a_2 y + b_1}{c_1 x + c_2 y + 1} \quad \text{Projective}
\]

\[
y' = \frac{a_3 x + a_4 y + b_2}{c_1 x + c_2 y + 1}
\]

\[
E = \sum [f(x', y') - f(x, y)]^2 = \sum e^2
\]

\[\text{min}\]

Szeliski

**Motion Vector:**

\[
m = [a_1 \quad a_2 \quad a_3 \quad a_4 \quad b_1 \quad b_2 \quad c_1 \quad c_2]^T
\]
Szeliski (Levenberg-Marquardet)

\[
\alpha_{kl} = \sum_n \frac{\partial e_n}{\partial m_k} \frac{\partial e_n}{\partial m_l}
\]

\[
\beta_k = -\sum e \frac{\partial e_n}{\partial m_k}
\]

\[
\Delta m = (A + \lambda I)^{-1} b
\]

Approximation of Hessian (J^TJ, Jacobian)

\[
\frac{\partial e}{\partial a_1} = \frac{\partial e}{\partial x'} \frac{\partial x'}{\partial a_1} + \frac{\partial e}{\partial y'} \frac{\partial y'}{\partial a_1}
\]

\[
E = \sum [f(x', y') - f(x, y)]^2 = \sum e^2
\]

\[
x' = \frac{a_1x + a_2y + b_1}{c_1x + c_2y + 1}
\]

\[
y' = \frac{a_3x + a_4y + b_2}{c_1x + c_2y + 1}
\]
Approximation of Hessian

\[ f^T = \begin{bmatrix} \frac{\partial f}{\partial m_1} & \frac{\partial f}{\partial m_2} & \cdots & \frac{\partial f}{\partial m_t} \\ \frac{\partial f}{\partial m_1} & \frac{\partial f}{\partial m_2} & \cdots & \frac{\partial f}{\partial m_t} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f}{\partial m_1} & \frac{\partial f}{\partial m_2} & \cdots & \frac{\partial f}{\partial m_t} \end{bmatrix} \]

\[ J = \begin{bmatrix} \frac{\partial e_n}{\partial m_1} & \frac{\partial e_n}{\partial m_2} & \cdots & \frac{\partial e_n}{\partial m_t} \\ \frac{\partial e_n}{\partial m_1} & \frac{\partial e_n}{\partial m_2} & \cdots & \frac{\partial e_n}{\partial m_t} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial e_n}{\partial m_1} & \frac{\partial e_n}{\partial m_2} & \cdots & \frac{\partial e_n}{\partial m_t} \end{bmatrix} \]

\[ A = J^T J \]

\[ \alpha_{kl} = \sum_n \frac{\partial e_n}{\partial m_k} \frac{\partial e_n}{\partial m_l} \quad \text{A Matrix} \]

Gradient Vector

\[ b = \begin{bmatrix} -\sum_n e_n \frac{\partial e_n}{\partial a_1} \\ -\sum_n e_n \frac{\partial e_n}{\partial a_2} \\ -\sum_n e_n \frac{\partial e_n}{\partial a_3} \\ -\sum_n e_n \frac{\partial e_n}{\partial a_4} \\ -\sum_n e_n \frac{\partial e_n}{\partial b_1} \\ -\sum_n e_n \frac{\partial e_n}{\partial b_2} \\ -\sum_n e_n \frac{\partial e_n}{\partial c_1} \\ -\sum_n e_n \frac{\partial e_n}{\partial c_2} \end{bmatrix} \]
Partial Derivatives wrt motion parameters

\[
\frac{\partial e}{\partial a_1} = \frac{\partial e}{\partial x'} \frac{\partial x'}{\partial a_1} + \frac{\partial e}{\partial y'} \frac{\partial y'}{\partial a_1}, \quad \frac{\partial x'}{\partial a_1} = \frac{x}{c_1 x + c_2 y + 1}, \quad \frac{\partial y'}{\partial a_1} = 0
\]

\[
x' = \frac{a_x + a_y + b_1}{c_1 x + c_2 y + 1}, \quad y' = \frac{a_x + a_y + b_2}{c_1 x + c_2 y + 1}
\]

\[
\frac{\partial x'}{\partial a_1} = 0, \quad \frac{\partial x'}{\partial a_4} = \frac{1}{c_1 x + c_2 y + 1}, \quad \frac{\partial x'}{\partial b_2} = 0
\]

\[
\frac{\partial y'}{\partial a_1} = \frac{x}{c_1 x + c_2 y + 1}, \quad \frac{\partial y'}{\partial a_4} = \frac{y}{c_1 x + c_2 y + 1}, \quad \frac{\partial y'}{\partial b_2} = 0
\]

\[
\frac{\partial x'}{\partial c_1} = -\frac{x(a_x + a_y + b_1)}{(c_1 x + c_2 y + 1)^2}, \quad \frac{\partial y'}{\partial c_1} = -\frac{y(a_x + a_y + b_1)}{(c_1 x + c_2 y + 1)^2}
\]

\[
\frac{\partial x'}{\partial c_2} = -\frac{x(a_x + a_y + b_1)}{(c_1 x + c_2 y + 1)^2}, \quad \frac{\partial y'}{\partial c_2} = -\frac{y(a_x + a_y + b_1)}{(c_1 x + c_2 y + 1)^2}
\]

Partial derivatives wrt image coordinates

\[
E = \sum [f(x', y') - f(x, y)]^2 = \sum e^2
\]

\[
\frac{\partial e}{\partial x'} = f_x, \quad \frac{\partial e}{\partial y'} = f_y
\]
Partial derivatives

$$\frac{\partial e}{\partial x} = \frac{\partial e}{\partial x'} + \frac{\partial e}{\partial y} = f_x' \frac{x}{c_1 x + c_2 y + 1}$$
$$\frac{\partial e}{\partial y} = \frac{\partial e}{\partial x} + \frac{\partial e}{\partial y} = f_y' \frac{y}{c_1 x + c_2 y + 1}$$
$$\frac{\partial e}{\partial z} = \frac{\partial e}{\partial x} + \frac{\partial e}{\partial y} = f_z' \frac{x}{c_1 x + c_2 y + 1}$$
$$\frac{\partial e}{\partial w} = \frac{\partial e}{\partial x} + \frac{\partial e}{\partial y} = f_w' \frac{y}{c_1 x + c_2 y + 1}$$
$$\frac{\partial e}{\partial h} = \frac{\partial e}{\partial x} + \frac{\partial e}{\partial y} = f_h' \frac{1}{c_1 x + c_2 y + 1}$$

Gradient Vector

$$\mathbf{b} = \begin{bmatrix} -\sum f_x' \frac{x}{c_1 x + c_2 y + 1} \\ -\sum f_y' \frac{y}{c_1 x + c_2 y + 1} \\ -\sum f_z' \frac{x}{c_1 x + c_2 y + 1} \\ -\sum f_w' \frac{y}{c_1 x + c_2 y + 1} \\ -\sum f_h' \frac{1}{c_1 x + c_2 y + 1} \\ \sum f_x' \frac{f_x'(a_1 x + a_2 y + b)}{(c_1 x + c_2 y + 1)^2} \\ \sum f_y' \frac{f_y'(a_1 x + a_2 y + b)}{(c_1 x + c_2 y + 1)^2} \end{bmatrix}$$
Szeliski (Levenberg-Marquardt)

- Start with some initial value of \(m\), and \(\lambda=.001\)
  - For each pixel \(I\) at \((x_i, y_i)\)

- Compute \((x', y')\) using projective transform.

- Compute \(e = f(x', y') - f(x, y)\)

- Compute \[
  \frac{\partial e}{\partial m_k} = \frac{\partial e}{\partial x'} \frac{\partial x'}{\partial m_k} + \frac{\partial e}{\partial y'} \frac{\partial y'}{\partial m_k}
\]

Szeliski (Levenberg-Marquardt)

- Compute \(A\) and \(b\)

- Solve system

\[
(A - \lambda I) \Delta m = b
\]

- Update

\[
m^{t+1} = m^t + \Delta m
\]
Szeliski (Levenberg-Marquadet)

- check if error has decreased, if not increase $\lambda$ by a factor of 10 and compute a new $\Delta m$
- If error has decreased, decrease $\lambda$ by a factor of 10 and compute a new $\Delta m$
- Continue iteration until error is below threshold.