Targil 8:

Motion Transformations and Image Warping (cont.)

Automatic Image Alignment: Lucas Kanade (cont.)

Many slides from Alexei Efros and Steve Seitz

Reminder: Homogeneous Coordinates

Add a 3rd coordinate to every 2D point
• \((x, y, w)\) represents a point at location \((x/w, y/w)\)
• \((x, y, 0)\) represents a point at infinity
• \((0, 0, 0)\) is not allowed

Convenient coordinate system to represent many useful transformations

\[
\begin{pmatrix}
\lambda  \\
\mu  \\
\zeta
\end{pmatrix}
= \begin{pmatrix}
2,1,1  \\
4,2,2  \\
6,3,3
\end{pmatrix}
\]

Matrix Composition

Transformations can be combined by matrix multiplication

\[
\begin{pmatrix}
x' \\
y' \\
w'
\end{pmatrix}
= \begin{pmatrix}
1 & 0 & tx \\
0 & 1 & ty \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
\cos \Theta & -\sin \Theta & 0 \\
\sin \Theta & \cos \Theta & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
w
\end{pmatrix}
\]

\[
p' = T(t_x,t_y) \cdot R(\Theta) \cdot S(s_x,s_y) \cdot p
\]

Matrix Composition

Matrices are a convenient and efficient way to represent a sequence of transformations
• General purpose representation
• Hardware matrix multiply

\[
p' = (T \cdot (R \cdot (S \cdot p)))
\]

\[
p' = (T^R \cdot S^T) \cdot p
\]
Matrix Composition

Be aware: order of transformations matters
– Matrix multiplication is not commutative

\[ p' = T \ast R \ast S \ast p \]

Rotating About An Arbitrary Point

What happens when you apply a rotation transformation to an object that is not at the origin?

Rotating About An Arbitrary Point

What happens when you apply a rotation transformation to an object that is not at the origin?

• It translates as well

How Do We Fix it?

How do we rotate an about an arbitrary point?

• Hint: we know how to rotate about the origin of a coordinate system

If we want to apply scaling: where should we add the scale?

2D image transformations

These transformations are a nested set of groups
• Closed under composition and inverse is a member
Distortions under central projection

Motion models
- Translation: 2 unknowns
- Affine: 6 unknowns
- Perspective: 8 unknowns

Image warping
Given a coordinate transform $(x',y') = h(x,y)$ and a source image $f(x,y)$, how do we compute a transformed image $g(x',y') = h(T(x,y))$?
Forward warping

Send each pixel \( f(x,y) \) to its corresponding location \( (x',y') = T(x,y) \) in the second image.

Q: what if pixel lands “between” two pixels?

Inverse warping

Get each pixel \( g(x',y') \) from its corresponding location \( (x,y) = T'(x',y') \) in the first image.

Q: what if pixel comes from “between” two pixels?

Bilinear interpolation

Sampling at \( f(x,y) \):

\[
\begin{bmatrix}
   (i,j) & (i,j+1) \\
   (i+1,j) & (i+1,j+1)
\end{bmatrix}
\]

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

Forward vs. inverse warping

Q: which is better?

A: usually inverse—eliminates holes
   - however, it requires an invertible warp function—not always possible...
Why do we want to estimate motion?

Lots of uses
- Correct for camera jitter (stabilization)
- Align images (mosaics)

Image Alignment

How do we align two images automatically?

Two broad approaches:
- Feature-based alignment
  - Find a few matching features in both images
  - Compute alignment
- Direct (pixel-based) alignment
  - Search for alignment where most pixels agree

Direct Alignment

The simplest approach is a brute force search
- Need to define image matching function
  - SSD, Normalized Correlation, edge matching, etc.
- Search over all parameters within a reasonable range:

E.g. for translation:
for tx=x0:step:x1,  
for ty=y0:step:y1,  
compare image1(x,y) to image2(x+tx,y+ty)  
end;
end;

Need to pick correct x0,x1 and step
- What happens if step is too large?

Direct Alignment (brute force)

What if we want to search for more complicated transformation, e.g. homography?

\[
\begin{bmatrix}
x' \\
y' \\
w
\end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]

for a=a0:astep:a1,  
for b=b0:bstep:b1,  
for c=c0:cstep:c1,  
for d=d0:dstep:d1,  
for e=e0:estep:e1,  
for f=f0:fstep:f1,  
for g=g0:gestep:g1,  
for h=h0:hstep:h1,  
compare image1 to H(image2)  
end; end; end; end; end; end; end; end; end;

Warping Example in Matlab

function warpimg=warpImage(image, a)
    warpimg=zeros(size(image));
    [x,y]=meshgrid(1:size(image,2),1:size(image,1));
    % find the center of the image
    % compute the new pixel locations x2 and y2
    warpimg=interp2(y,x,image, y2, x2, 'linear');
    % fix NaNs
    ind=find(~(warpimg==0 & warpimg==256));
    warpimg(ind)=0.0;
Problems with brute force

Not realistic
- Search in O(N) is problematic
- Not clear how to set starting/stopping value and step

What can we do?
- Use pyramid search to limit starting/stopping/step values

Alternative: gradient decent on the error function
- i.e. how do I tweak my current estimate to make the SSD error go down?
- Can do sub-pixel accuracy
- BiG assumption?
  - Images are already almost aligned (<1 pixels difference?)
  - Can improve with pyramid

Translation motion

The constant brightness constraint

We want to find \( u(x,y), v(x,y) \) that minimize the motion error

\[
E(u,v) = \sum_{x,y} \left[ I(x+u,y+v) - I(x,y) \right]^2
\]

\[
E(u,v) = \sum_{x,y} \left[ I(x,y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v - I(x,y) \right]^2
\]

Using Taylor approximation

The final LK equation

Writing it in simple form

\[
\sum_{x,y} \left[ I(x,y) + \frac{\partial I}{\partial x} u + \frac{\partial I}{\partial y} v - I(x,y) \right]^2 = \sum_{x,y} \left[ I_x u + I_y v + I \right]^2
\]

- \( I_x \): The x derivative of image \( I \)
- \( I_y \): The y derivative of image \( I \)
- \( I \): The difference between the images \( I_i-I_{i-1} \)

Remarks: To save computations, we compute \( I_i-I_{i-1} \)

The function to minimize over the image

\[
\sum_{x,y} \left[ I_x u + I_y v + I \right]^2
\]

Motion Constraints (grayscale images)

Let's look at these constraints more closely

- Brightness constancy: \( c \): what's the equation?
  \( I(x,y) = I(x+u,y+v) \)

- Small motion: \( (u,v) \) are less than 1 pixel

Assuming small motion \( (\leq 1 \text{ pixel}) \) between images, we can do the following approximation

\[
\cos \theta = 1, \quad x = u(x,y) = x - y \theta + x
\]

\[
\sin \theta = \theta, \quad y = v(x,y) = y \theta + y
\]
Substituting $u$ and $v$ we get
\[ E(u, v) = \sum_{x,y} (I_t * \Delta u * I_u * \Delta v * I_v) \]
Beyond Translation: A simple modification to the same idea
Instead of minimizing for constant $u,v$
\[ E(u, v) = \sum_{x,y} (I_t * \Delta u * I_u * \Delta v * I_v) \]
We substitute $u,v$ as a function of their location in the picture and the motion parameters $u(x,y), v(x,y)$, taking in consideration the motion model.

For example, to account for scale changes:
\[\begin{align*}
x_2 &= s^n x_1 + x_1 \\
y_2 &= s^n y_1 + y_1
\end{align*}\]
Then it should not be hard to compute the equations:
\[ E(x_2, y_2, s) = \sum_{x,y} (I_t * (x + x_1)^n + x_1 * (y + y_1)^n) \]

Revisiting the small motion assumption
Is this motion small enough?
- Probably not—it’s much larger than one pixel (2nd order terms dominate)
- How might we solve this problem?

LK Algorithm: using pyramids
Gaussian pyramid of image $I_1$
Gaussian pyramid of image $I_2$

Coarse-to-fine optical flow estimation
Gaussian pyramid of image $I_1$
Gaussian pyramid of image $I_2$
LK Algorithm for image translation

Create pyramids $P_1, P_2$ for the images $I_1, I_2$.

Initialize $u, v$ (e.g. $u=0, v=0$)

For pyramid levels $k$
- Update $u, v$ (multiply by 2)
- Compute the derivatives $I_x, I_y$ and matrix $A$
- Iterate until convergence
  - $E(u+v) = E(x, y)$
  - Compute the vector $b$
  - Solve the equations
  - Update $u = u + du, v = v + dv$. 

\[
A = \begin{bmatrix} \sum_{x} I_x^* I_1 & \sum_{x} I_y^* I_1 \\ \sum_{x} I_x^* I_2 & \sum_{x} I_y^* I_2 \end{bmatrix}, \quad b = \begin{bmatrix} \sum_{x} I_x^* I_1 \\ \sum_{x} I_y^* I_2 \end{bmatrix}, \quad A \begin{bmatrix} du \\ dv \end{bmatrix} = -b.
\]