Introduction to Computer Vision

Michael J. Black

Oct 2009

Motion estimation

Goals

- Motion estimation
  - Affine flow
  - Optimization
  - Large motions
  - Why affine?
- Monday
  - dense, smooth motion and regularization. Robust statistics
- Mon or Wed – discuss projects.

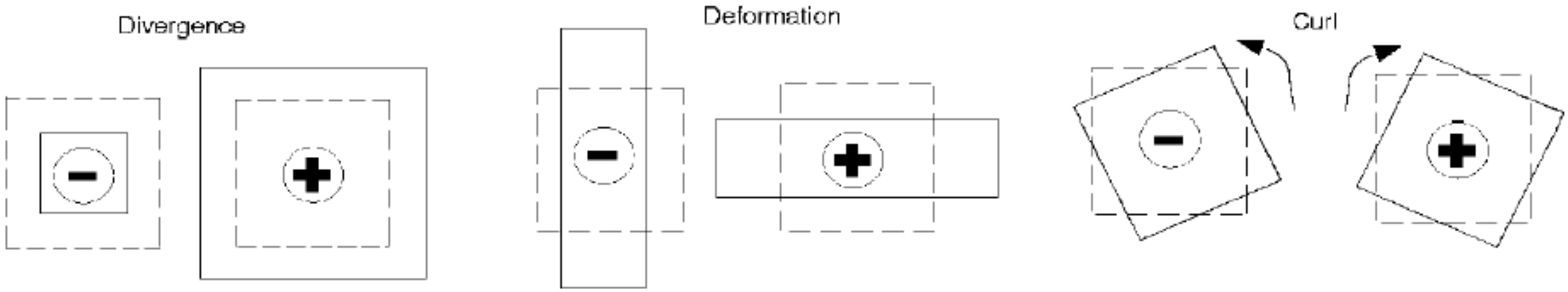
Assignment 3

- Part 1 and 2 due Nov 3 (Tuesday) 11am
- All due Nov 9 11am

Affine Flow

$$E(\mathbf{a}) = \sum_{x,y \in R} (\nabla I^T \mathbf{u}(\mathbf{x};\mathbf{a}) + I_t)^2$$

$$\mathbf{u}(\mathbf{x};\ \mathbf{a}) = \begin{bmatrix} u(\mathbf{x};\mathbf{a}) \\ v(\mathbf{x};\mathbf{a}) \end{bmatrix} = \begin{bmatrix} a_1 + a_2 x + a_3 y \\ a_4 + a_5 x + a_6 y \end{bmatrix}$$



Affine Transformation

$$\begin{bmatrix} x \\ y \end{bmatrix}^* = \begin{bmatrix} a_1 & a_2 \\ a_4 & a_5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} a_3 \\ a_6 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}^* = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Homogeneous coordinates

Linear transformation

Translation

Affine flow

Motion (flow) between frames

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Transformation of pixels

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 0 \end{bmatrix} + \begin{bmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Use when transforming pixels

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a1+1 & a2 & a3 \\ a4 & a5+1 & a6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Important Slide!

When I say x and y, I mean *relative to the center of the patch*. The patch may be the whole image.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - x_c \\ y - y_c \\ 1 \end{bmatrix}$$

[x_c y_c] defines the center of the patch.
So the affine transformation is wrt (0, 0).

What can be represented?

What does this do?

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a3 \\ 0 & 1 & a6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

What can be represented?

What does this do?

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

What can be represented?

What does this do?

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

What can be represented?

What does this do?

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Composition and inversion

- Apply two affine transforms successively.
- Represent affine transformations as matrices

$$x^* = A(Bx) = (AB)x$$

- Inversion

$$x = (AB)^{-1}x^*$$

Transforming images

- Affine transformation is applied to image coordinates x, y

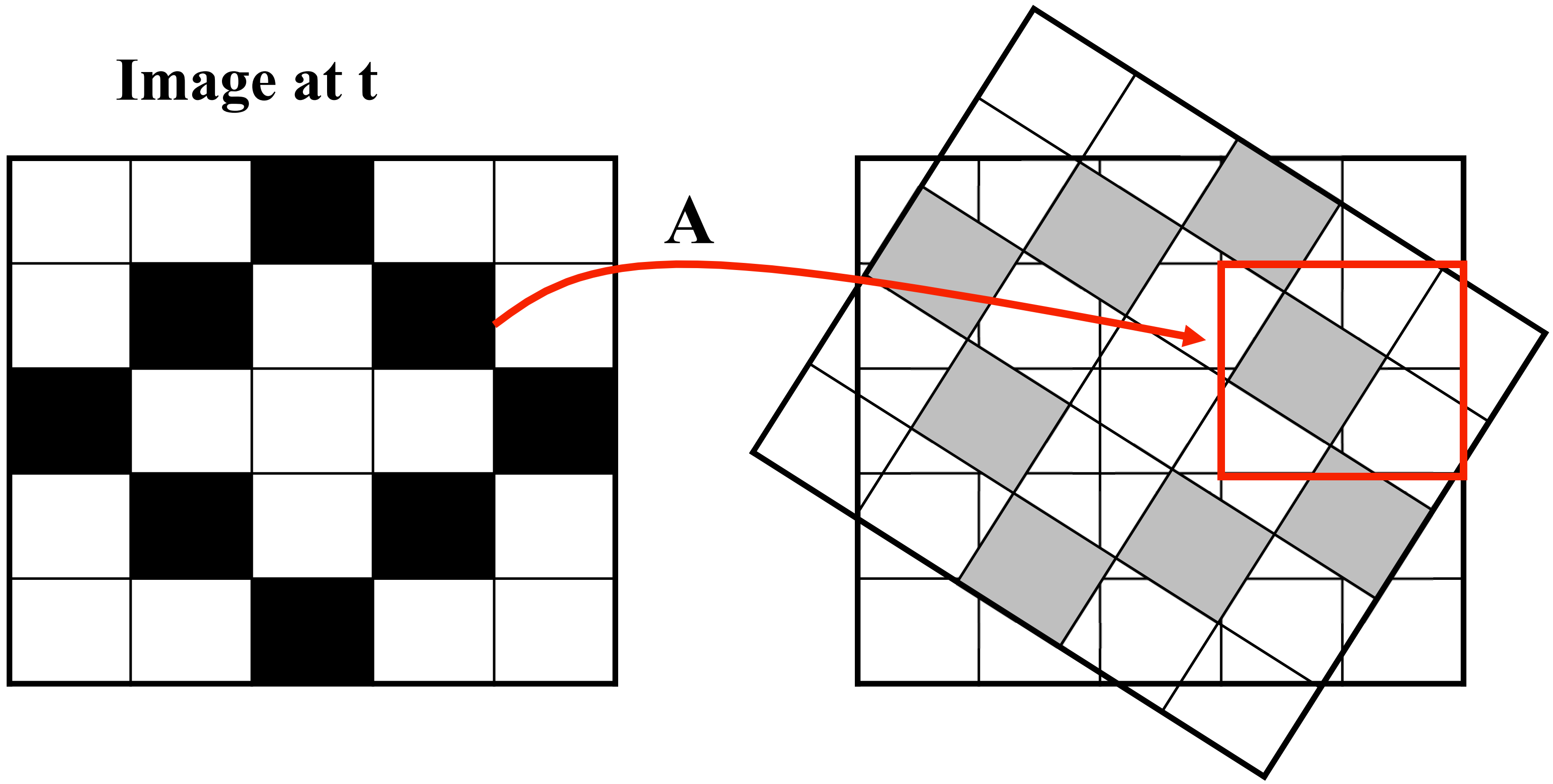
$$I' = I(A[x, y, 1]^T)$$

How do we do this in Matlab? What are the issues?

Get the image coordinates:

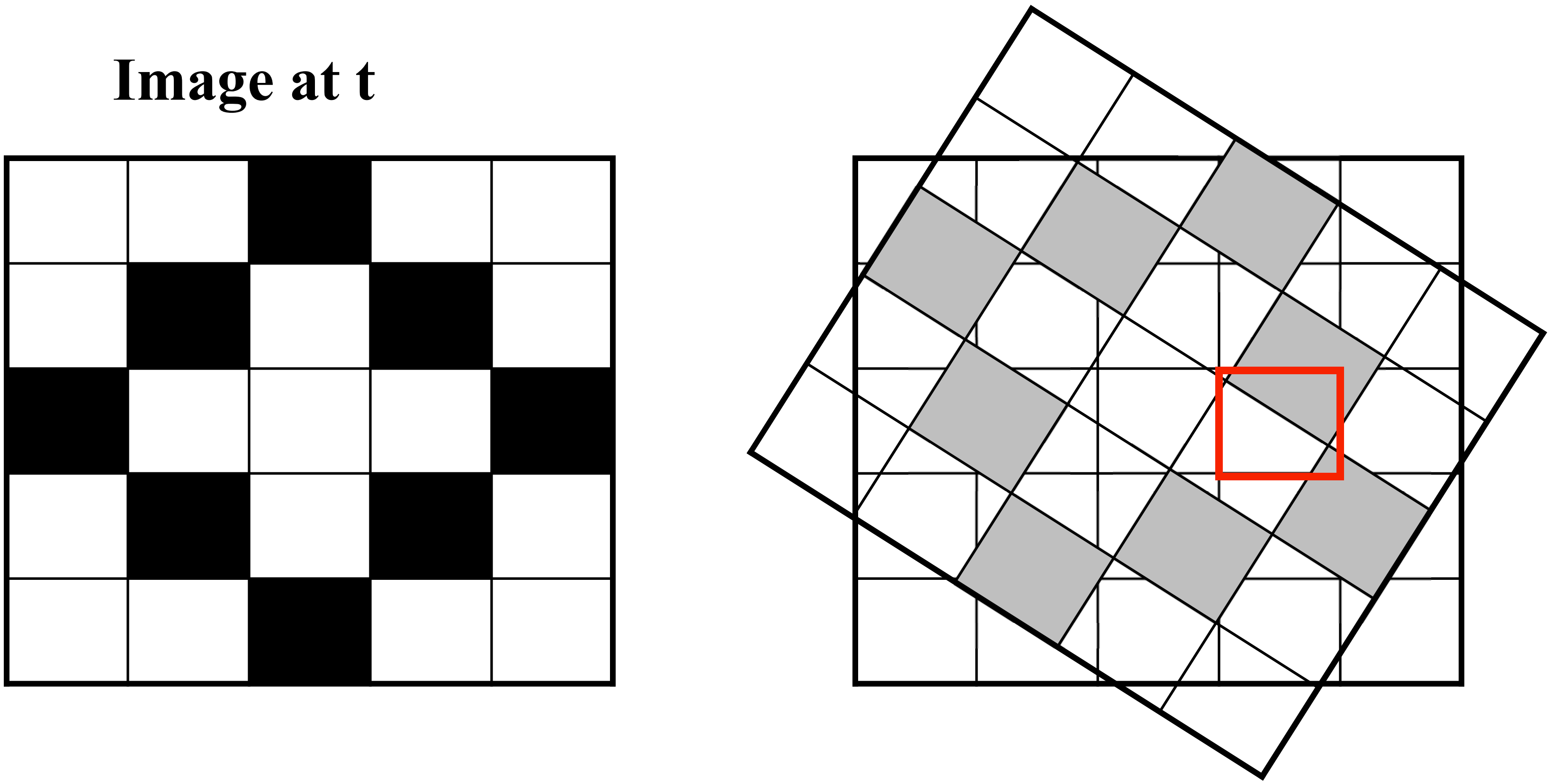
```
>> [y,x]=meshgrid(1:10, 1:10)

y =
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10
     1     2     3     4     5     6     7     8     9    10

x =
     1     1     1     1     1     1     1     1     1     1
     2     2     2     2     2     2     2     2     2     2
     3     3     3     3     3     3     3     3     3     3
     4     4     4     4     4     4     4     4     4     4
     5     5     5     5     5     5     5     5     5     5
     6     6     6     6     6     6     6     6     6     6
     7     7     7     7     7     7     7     7     7     7
     8     8     8     8     8     8     8     8     8     8
     9     9     9     9     9     9     9     9     9     9
    10    10    10    10    10    10    10    10    10    10
```

Forwards Warp



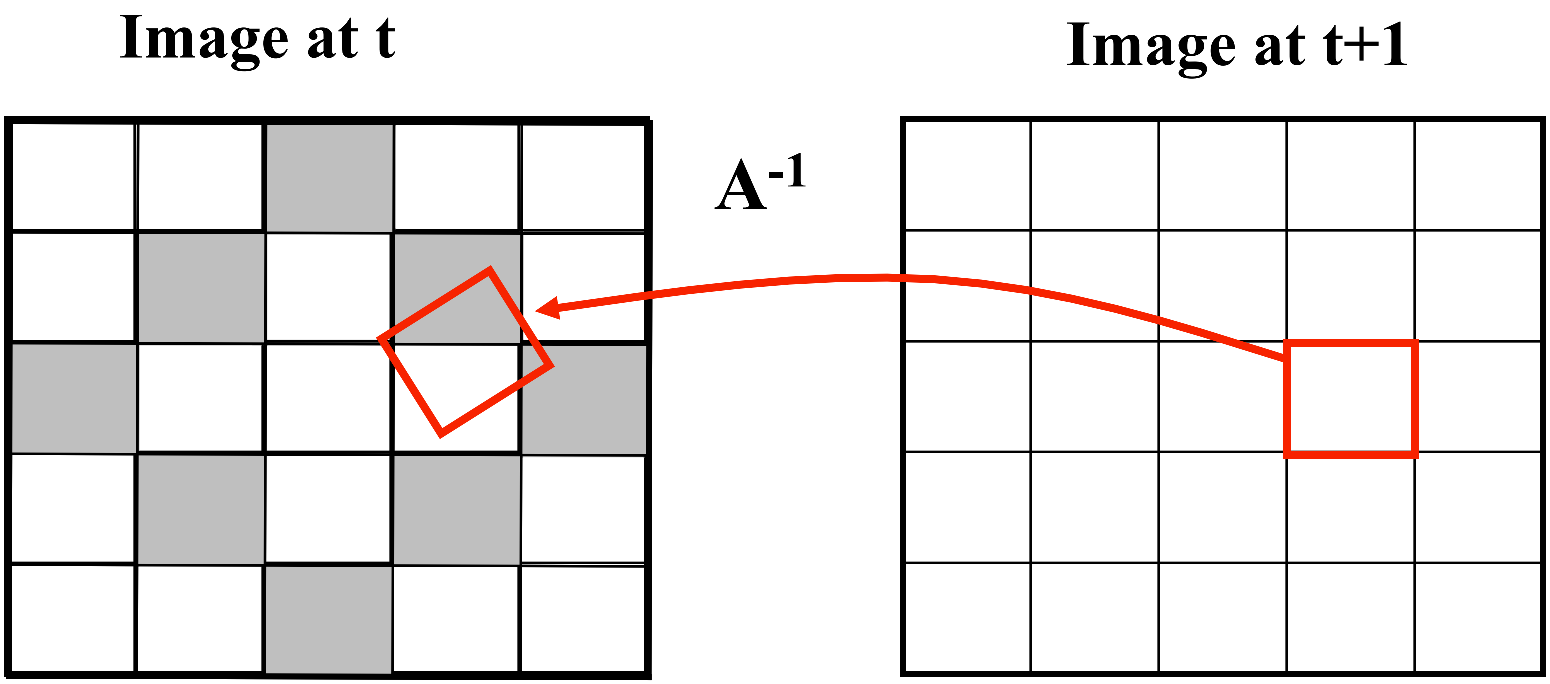
Contributes to 4 pixels.

Backwards Warp



Contributions *from* 4 pixels.

Backwards Warp



Contributions *from* 4 pixels – bi-linear interpolation

Every pixel at time t+1 defined.

Interpolation

- Possible interpolation filters:
  - nearest neighbor
  - **bilinear**
  - bicubic (interpolating)

- Needed to prevent "jaggies"
- **When *iteratively* warping, always *compose* the warps and warp the *original* image**

Szeliski and Fleet

Warping images

Example warps:

translation

rotation

aspect

affine

perspective

cylindrical

Szeliski and Fleet

interp2

INTERP2 2-D interpolation (table lookup).

ZI = interp2(X,Y,Z,XI,YI)

interpolates to find ZI, the values of the underlying 2-D function Z at the points in matrices XI and YI.

Matrices X and Y specify the points at which the data Z is given.

Out of range values are returned as NaN.

Image warping

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

Algorithm (i.e. homework #3)

Incremental optimization

* Given the images, construct the structure tensor invert it, and solve for the motion parameters.

* Warp image 1 towards image 2

* repeat until convergence

Optimization

$$E(\mathbf{a}) = \sum_{x,y \in R} (I_x u + I_y v + I_t)^2$$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} a1 & a2 & a3 \\ a4 & a5 & a6 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$E(\mathbf{a}) = \sum_{x,y \in R} (I_x a_1 x + I_x a_2 y + I_x a_3 + I_y a_4 x + I_y a_5 y + I_y a_6 + I_t)^2$$

Optimization

$$E(\mathbf{a}) = \sum_{x,y \in R} (I_x a_1 x + I_x a_2 y + I_x a_3 + I_y a_4 x + I_y a_5 y + I_y a_6 + I_t)^2$$

Differentiate wrt the a_i and set equal to zero.

$$\begin{bmatrix} \Sigma I_x^2 x^2 & \Sigma I_x^2 xy & \Sigma I_x^2 x & \Sigma I_x I_y x^2 & \Sigma I_x I_y xy & \Sigma I_x I_y x \\ \Sigma I_x^2 xy & \Sigma I_x^2 y^2 & \Sigma I_x^2 y & \Sigma I_x I_y xy & \Sigma I_x I_y y^2 & \Sigma I_x I_y y \\ & & & \vdots & & \\ & & & & & \\ & & & & & \\ & & & & & \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \\ a_6 \end{bmatrix} = \begin{bmatrix} -\Sigma I_x I_t x \\ -\Sigma I_x I_t y \\ -\Sigma I_x I_t \\ -\Sigma I_y I_t x \\ -\Sigma I_y I_t y \\ -\Sigma I_y I_t \end{bmatrix}$$

We have a problem

- Taylor approximation assumed small motions.
- Real motions may be larger than a pixel.
- Temporal derivative won’t make sense.
- Need a solution.

Compute Flow

function a=basicFlow(im1, im2, ainit, iters)

% warp im1 by current flow parameters (start with ainit)

% compute image derivatives

% build structure tensor

– % solve for motion parameters (exclude boundary pixels and non-overlapping pixels from the analysis – mark them with nan's)

% update current flow parameters (*compose* affine transformations)

% repeat

Testing your motion code

- Take an image and warp it by some known affine motion.
- Solve for the motion
- You should be able to fairly accurately recover the parameters.
- Start with only *translation*.

Coarse to Fine (Translation)

function a=pyramidFlow(im1, im2, ainit, iters, levels)

% build image pyramids (dividing a3 and a6 by 2 for each level as you go)

% starting with coarse level

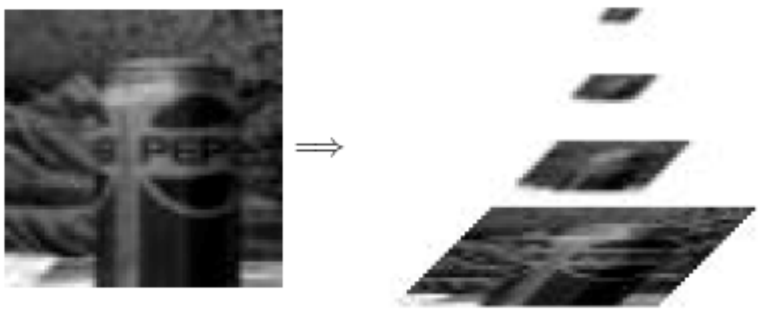
%   warp im1 by current flow

%   estimate flow

%   project flow to next level

%   (ie multiply a3 & a6 by 2)

% repeat to finest level



Why Affine?

- Where does this affine approximation come from?
- All our models are approximations to the world. What are the assumptions in the affine approximation?
- For this we need some geometry.

Pinhole cameras

- Abstract camera model - box with a small hole in it.
- Easy to build but needs a lot of light.