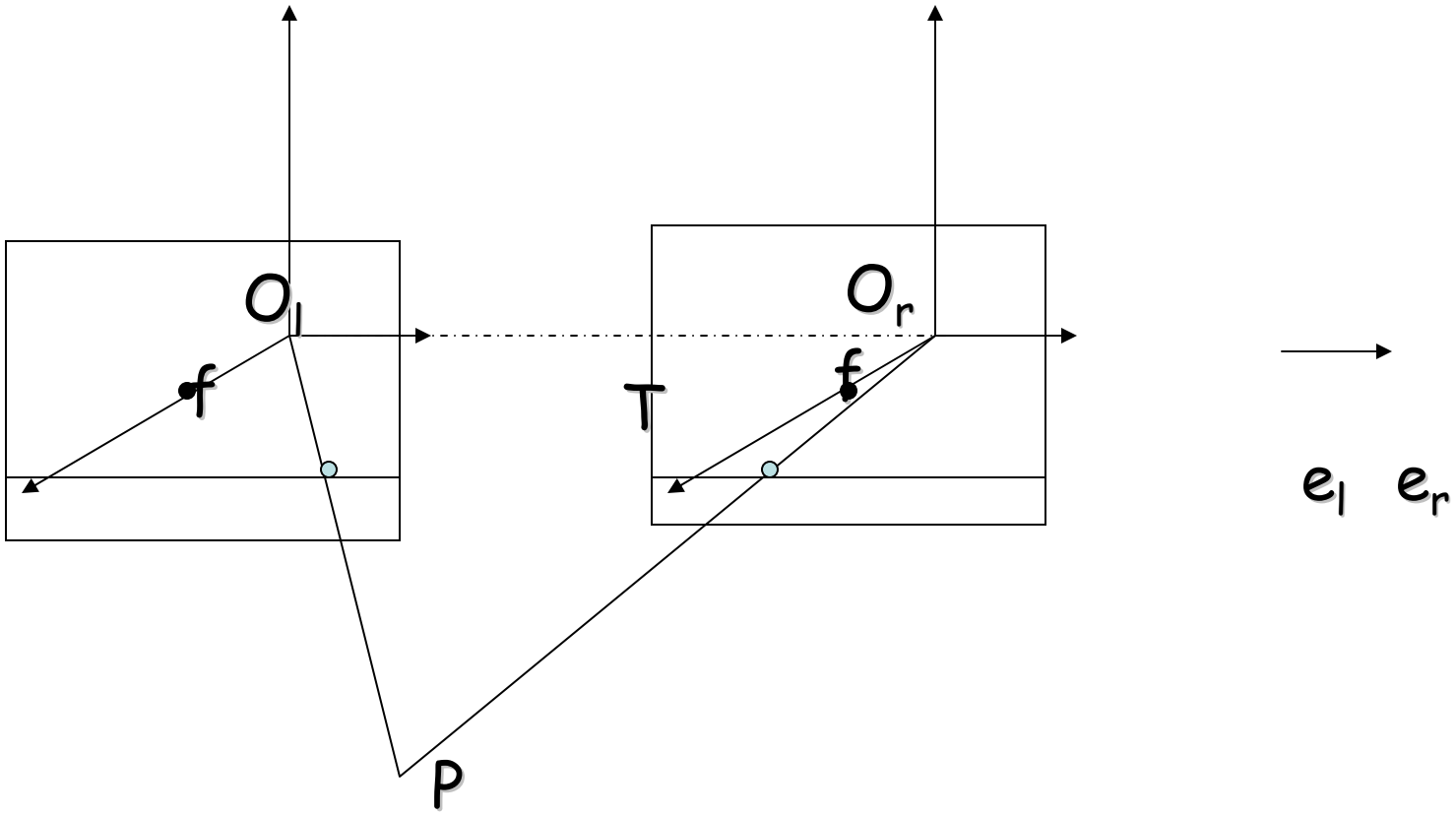
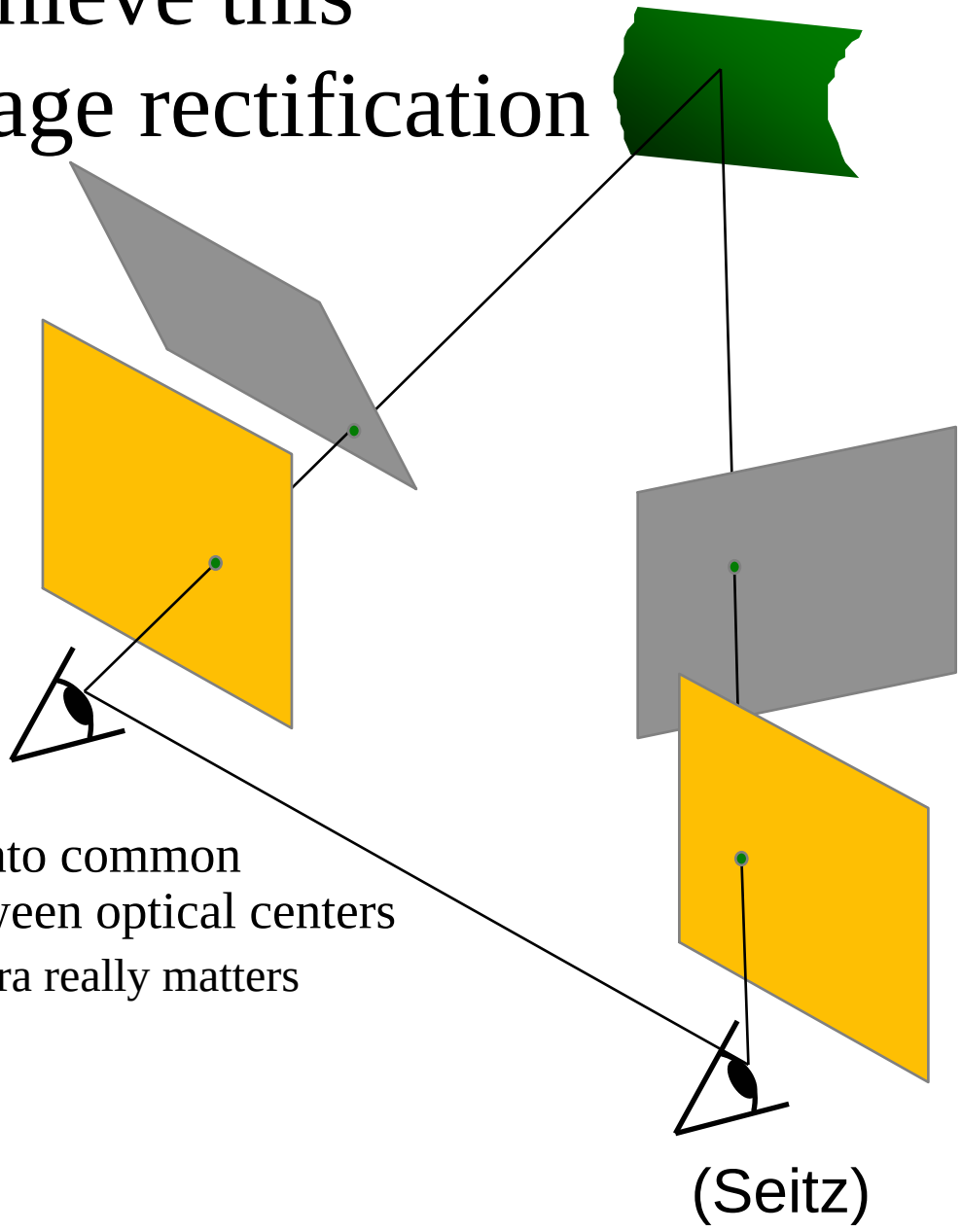
Stereopsis-III
Motion -I

Epipolar Geometry for Parallel Cameras



Epipoles are at infinity
Epipolar lines are parallel to the baseline

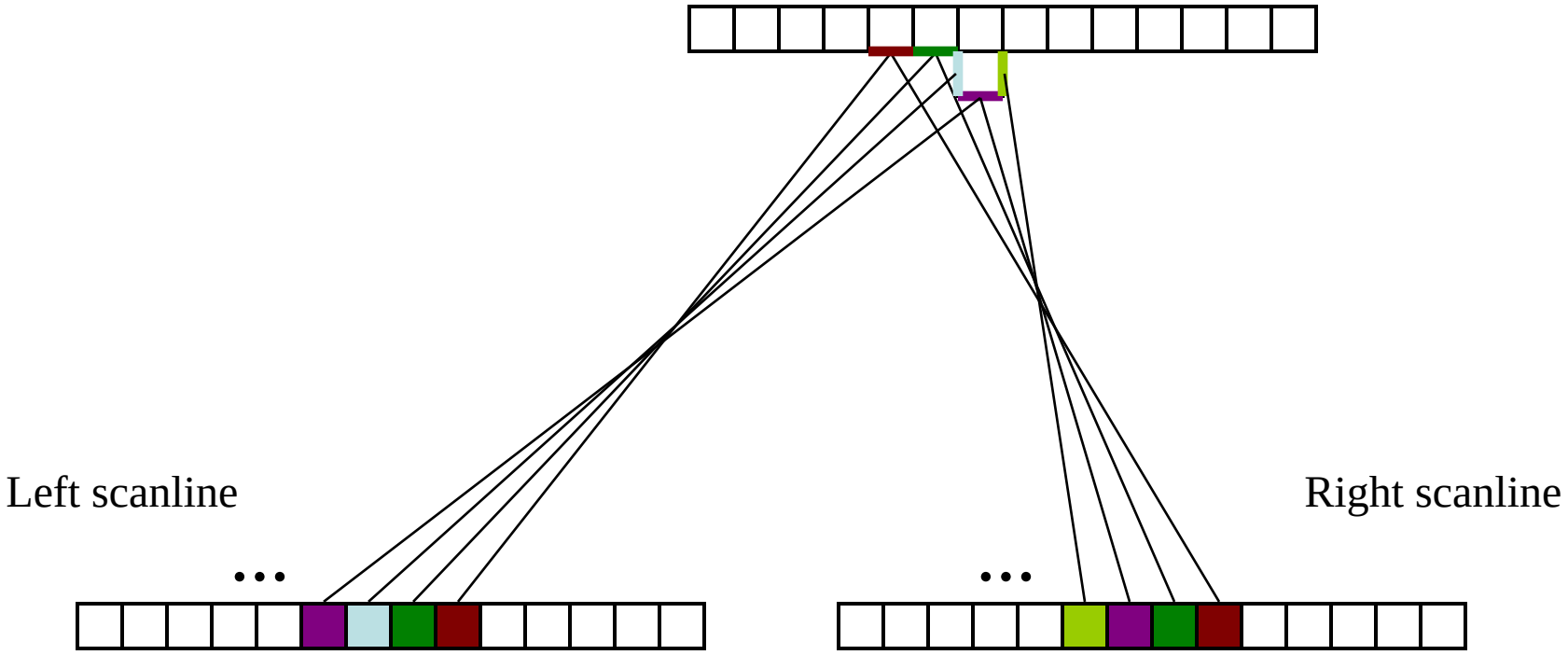
We can always achieve this geometry with image rectification



- Image Reprojection
  - reproject image planes onto common plane parallel to line between optical centers
- Notice, only focal point of camera really matters

(Seitz)

Stereo Correspondences



Dynamic Programming: Assume ordering constraint: a matched pair means no other matches can occur to their left on either scanline

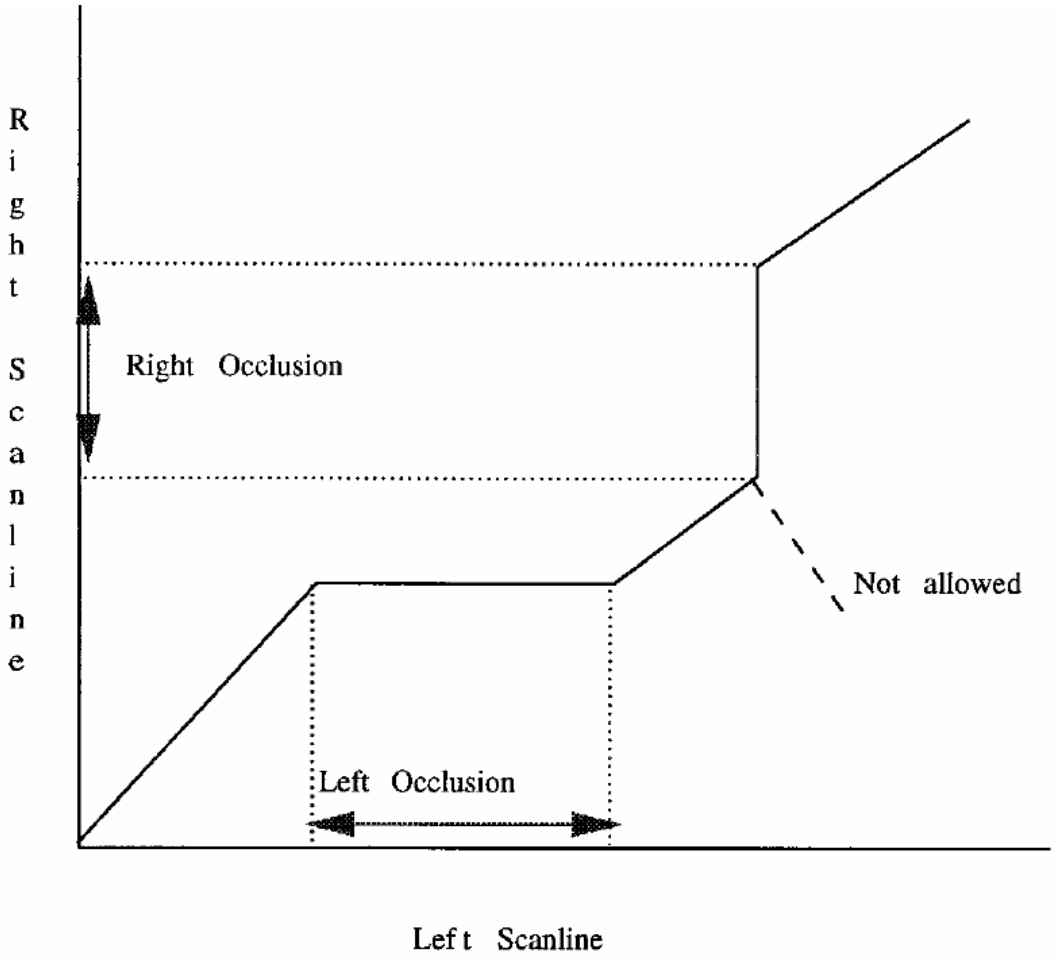


FIG. 1. A path representing a matching of points in the left and right images. The solid line represents a legal set of matches. The dashed path violates the uniqueness and ordering constraints.

Possibilities at each pixel pair

- At any pixel all possibilities are to the right … pixels on the left have already been matched
- Thus any “global” optimal match will contain a “local” optimal match
- Match pixel pair
- Goodness of match determined by some cost-function
- Or, we cannot match pixels because:
    1. Occlusion in the right image
    2. Occlusion in the left image

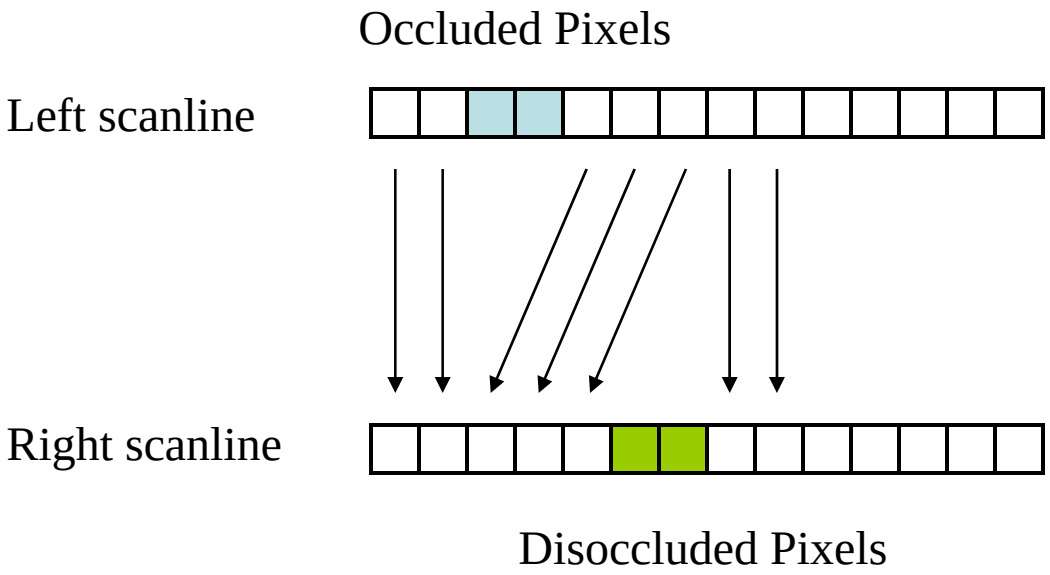
Possibilities at each match

```
Occlusion =
```

Const.

```
for (i=1;i≤ N;i++){ C(i,0) = i*Occlusion }
for (i=1;i≤ M;i++) { C(0,i) = i*Occlusion}
for(i=1;i≤ N;i++){
    for(j=1;j≤ M;j++){
        min1 = C(i-1,j-1)+c(z_{1,i},z_{2,j});
        min2 = C(i-1,j)+Occlusion;
        min3 = C(i,j-1)+Occlusion;
        C(i,j) = cmin = min(min1,min2,min3);
        if(min1==cmin) M(i,j) = 1;
        if(min2==cmin) M(i,j) = 2;
        if(min3==cmin) M(i,j) = 3;
    }}
```

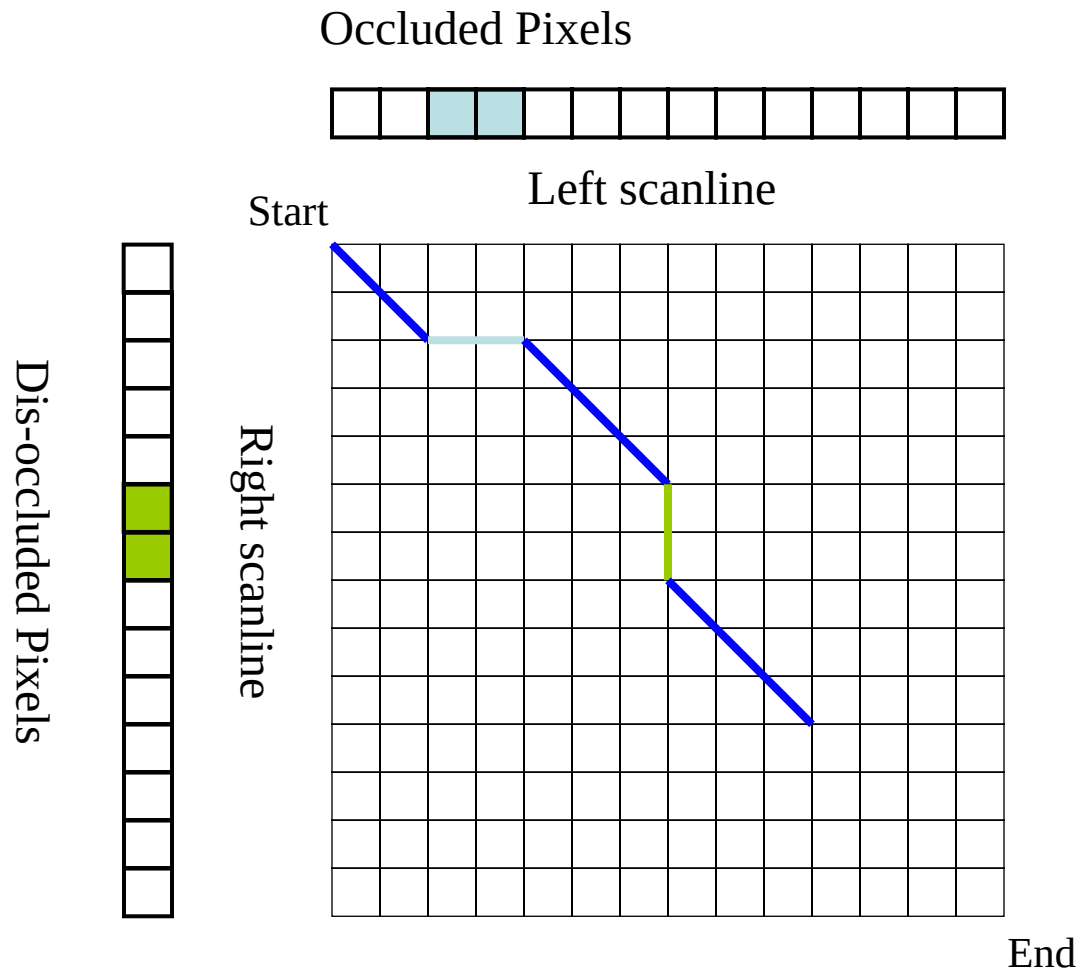
Search Over Correspondences



Three cases:

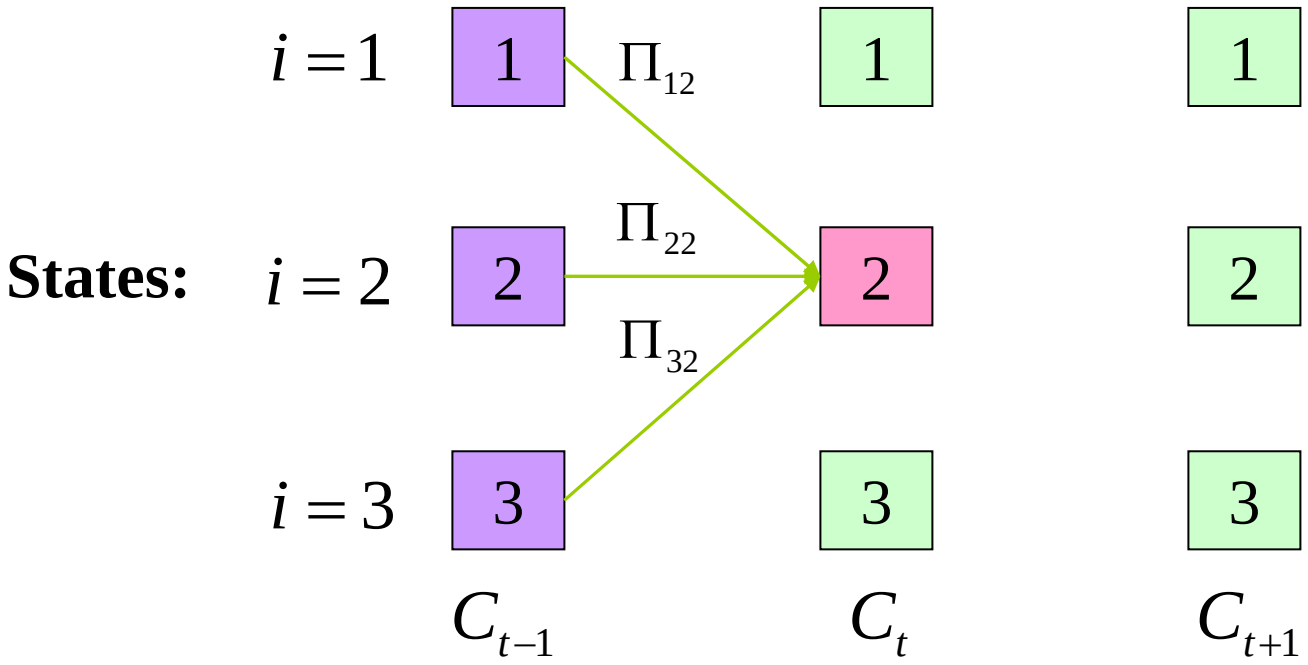
- Sequential – add cost of match (small if intensities agree)
- Occluded – add cost of no match (large cost)
- Disoccluded – add cost of no match (large cost)

Stereo Matching with Dynamic Programming



Dynamic programming yields the optimal path through grid. This is the best set of matches that satisfy the ordering constraint

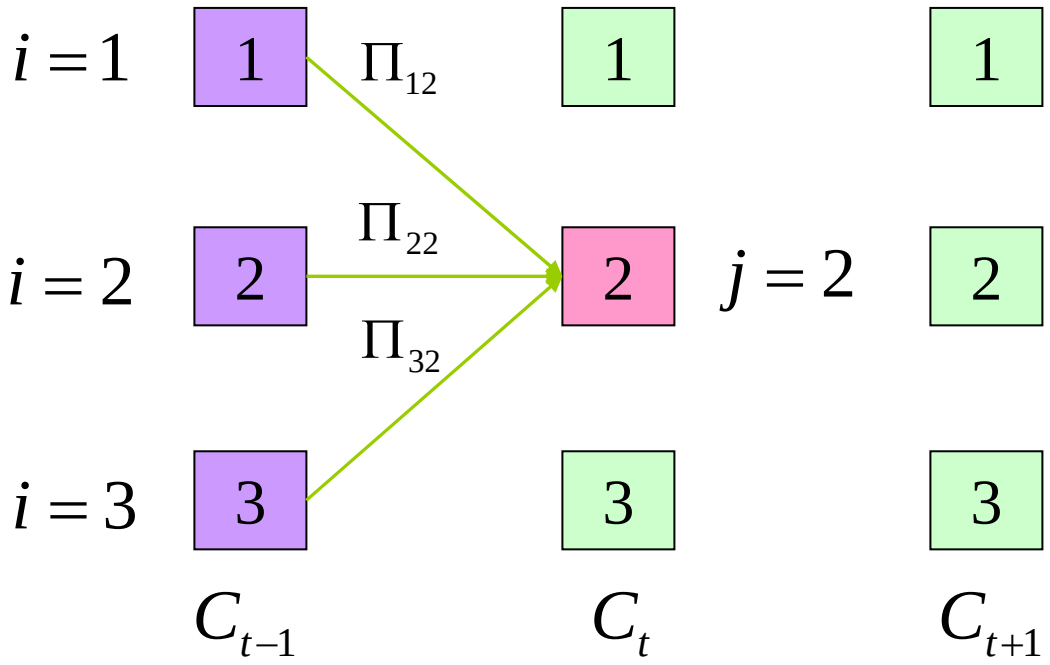
Dynamic Programming



Suppose cost can be decomposed into stages:

$$\Pi_{ij} = \text{Cost of going from state } i \text{ to state } j$$

Dynamic Programming



Principle of Optimality for an n-stage assignment problem:

$$C_t(j) = \min_i (\Pi_{ij} + C_{t-1}(i))$$

Algorithm

- Decide which cost is minimum
- Start from end

```
p=N;
q=M;
while(p!=0 && q!=0){
    switch(M(p,q)){
        case 1:
            p matches q
            p--;q--;
            break;
        case 2:
            p is unmatched
            p--;
            break;
        case 3:
            q is unmatched
            q--;
            break;
    }}
```

Structure-from-Motion

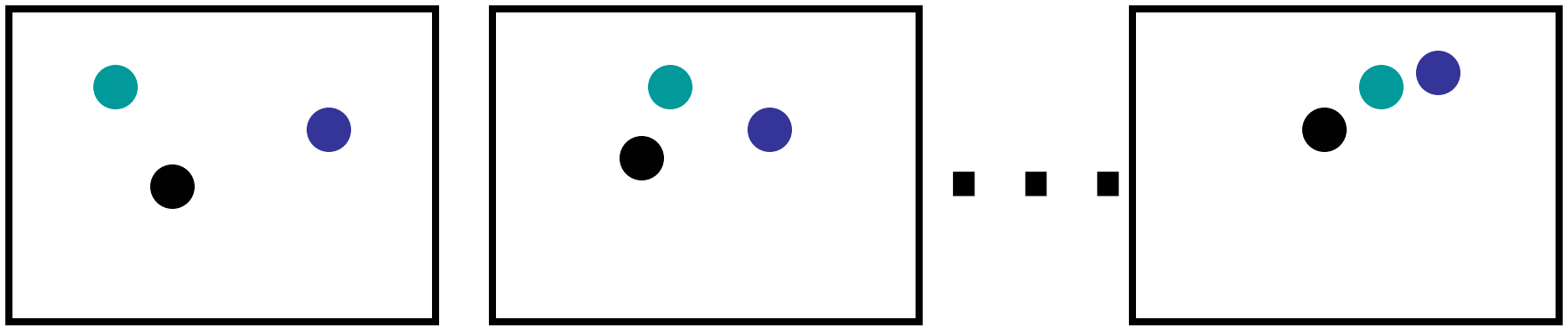
- Determining the 3-D structure of the world, and/or the motion of a camera using a sequence of images taken by a moving camera.
  - Equivalently, we can think of the world as moving and the camera as fixed.
- Like stereo, but the position of the camera isn't known (and it's more natural to use many images with little motion between them, not just two with a lot of motion).
  - We may or may not assume we know the parameters of the camera, such as its focal length.

Structure-from-Motion

- As with stereo, we can divide problem:
  - Correspondence.
  - Reconstruction.
- Again, we'll talk about reconstruction first.
  - Assume that each image contains some points,
  - we know which points match which

- Cases to consider
  - Moving camera (moving human)
  - Moving objects in the scene
  - General case: Both are moving
- We must relate motion in the real world with motion in images

Structure-from-Motion



Reconstruction

- A lot harder than with stereo.
- Start with simpler case: scaled orthographic projection (weak perspective).
  - Recall, in this we remove the z coordinate and scale all x and y coordinates the same amount.
  - We will consider a fixed camera and a moving scene

First: Represent motion

- We'll talk about a fixed camera, and moving object.
- Key point:

Some matrix

$$S = \begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} & t_x \\ r_{2,1} & r_{2,2} & r_{2,3} & t_y \\ r_{3,1} & r_{3,2} & r_{3,3} & t_z \end{pmatrix}$$

Points

$$P = \begin{pmatrix} x_1 & x_2 & & & x_n \\ y_1 & y_2 & . & . & y_n \\ z_1 & z_2 & & & z_n \\ 1 & 1 & & & 1 \end{pmatrix}$$

The image (ignoring *z)*

$$I = \begin{pmatrix} X_1 & X_2 & . & . & X_n \\ Y_1 & Y_2 & & & Y_n \end{pmatrix}$$

Then: $I = SP$

- Objects represented as a moving set of points
- These are projected into the image
- Projection is represented with Matrix multiplication
    - take inner product between each row of S and each point.
    - First row of S produces X coordinates, while second row produces Y.
- Projection model is “scaled orthographic” here
    - from now on we ignore third row of S.
- Translation occurs with t_x and t_y; t_z does not matter
- Scaling encoded with a scale factor in S.
- The rest of S accounts for rotation

Examples:

- S = [s, 0, 0, 0;
  0, s, 0, 0];

  This is just projection, with scaling by s.

- S = [s, 0, 0, s*tx;
  0, s, 0, s*ty];

This is translation by (tx,ty,something), projection, and scaling.

Structure-from-Motion

- S encodes:
  - Projection: only two lines
  - Scaling, since S can have a scale factor.
  - Translation, by tx and ty.
  - Rotation:

$$I = SP$$

Rotation

$$\begin{pmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{pmatrix} P$$

Represents a 3D rotation of the points in P.

First, look at 2D rotation (easier)

Matrix R acts on points by rotating them.

$$R = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \end{pmatrix}$$

• Also, RR^T = Identity. R^T is also a rotation matrix, in the opposite direction to R.

Why does multiplying points by R rotate them?

- rows of R are basis vectors of a new coordinate system
- Taking inner products of each point with these expresses that point in that coordinate system.
    - This means rows of R must be orthonormal vectors (orthogonal unit vectors).
    - points (1,0) and (0,1) go to (cos θ, -sin θ), and (sin θ, cos θ). They remain orthonormal, and rotate clockwise by theta.
    - Any other point, (a,b) can be written as a(1,0) + b(0,1).
    - So R(a(1,0)+b(0,1) = Ra(1,0) + Ra(0,1) = aR(1,0) + bR(0,1).
    - So it's in the same position relative to the rotated coordinates that it was in before rotation relative to the x, y coordinates. That is, it's rotated.

Simple 3D Rotation

$$\begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \\ z_1 & z_2 & & & & z_n \end{pmatrix}$$

Rotation about z axis.

Rotates x,y coordinates. Leaves z coordinates fixed.

Full 3D Rotation

$$R = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{pmatrix}$$

• Any rotation can be expressed as combination of three rotations about three axes.

$$RR^T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

• Rows (and columns) of *R* are orthonormal vectors.

• R has determinant 1 (not -1).

•3D rotations can be expressed as 3 separate rotations about fixed axes. Rotations have 3 degrees of freedom; two describe an axis of rotation, and one the amount.

•Order of rotations matter

•Typically rotations described using “Euler angles”

• Rotations preserve the length of a vector, and the angle between two vectors.

•Axes, (1,0,0), (0,1,0), (0,0,1) must remain orthonormal after rotation.

• After rotation, they are the three columns of R. So these columns must be orthonormal vectors for R to be a rotation.

• Same reasoning as 2D tells us all other points rotate too. If R has determinant -1, then R is a rotation plus a reflection.

Putting it Together

Scale

3D Translation

$$s\begin{pmatrix}1 & 0 & 0\\ 0 & 1 & 0\end{pmatrix}\begin{pmatrix}1 & 0 & 0 & t_x\\ 0 & 1 & 0 & t_y\\ 0 & 0 & 1 & t_z\end{pmatrix}\begin{pmatrix}r_{1,1} & r_{1,2} & r_{1,3} & 0\\ r_{2,1} & r_{2,2} & r_{2,3} & 0\\ r_{3,1} & r_{3,2} & r_{3,3} & 0\\ 0 & 0 & 0 & 1\end{pmatrix}P$$

Projection

3D Rotation

$$\equiv\begin{pmatrix}s_{1,1} & s_{1,2} & s_{1,3} & st_x\\ s_{2,1} & s_{2,2} & s_{2,3} & st_y\end{pmatrix}P$$

where

$$(s_{1,1}, s_{1,2}, s_{1,3}) \bullet (s_{2,1}, s_{2,2}, s_{2,3}) = 0$$

$$\left\|(s_{1,1}, s_{1,2}, s_{1,3})\right\| = \left\|(s_{2,1}, s_{2,2}, s_{2,3})\right\|$$

We can just write st_x as t_x and st_y as t_y.

Affine Structure from Motion

$$\begin{pmatrix} s_{1,1} & s_{1,2} & s_{1,3} & t_x \\ s_{2,1} & s_{2,2} & s_{2,3} & t_y \end{pmatrix} P$$

where

$$(s_{1,1}, s_{1,2}, s_{1,3}) \bullet (s_{2,1}, s_{2,2}, s_{2,3}) = 0$$

$$\left\| (s_{1,1}, s_{1,2}, s_{1,3}) \right\| = \left\| (s_{2,1}, s_{2,2}, s_{2,3}) \right\|$$

Affine Structure-from-Motion: Two Frames (1)

$$\begin{pmatrix} u_1^1 & u_2^1 & . & . & . & u_n^1 \\ v_1^1 & v_2^1 & & & & v_n^1 \\ u_1^2 & u_2^2 & & & & u_n^2 \\ v_1^2 & v_2^2 & & & & v_n^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \\ z_1 & z_2 & & & & z_n \\ 1 & 1 & & & & 1 \end{pmatrix}$$

Affine Structure-from-Motion: Two Frames (2)

To make things easy, suppose:

$$\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Affine Structure-from-Motion: Two Frames (3)

$$\begin{pmatrix} u_1^1 & u_2^1 & . & . & . & u_n^1 \\ v_1^1 & v_2^1 & & & & v_n^1 \\ u_1^2 & u_2^2 & & & & u_n^2 \\ v_1^2 & v_2^2 & & & & v_n^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \\ z_1 & z_2 & & & & z_n \\ 1 & 1 & & & & 1 \end{pmatrix}$$

Looking at the first four points, we get:

$$\begin{pmatrix} u_1^1 & u_2^1 & u_3^1 & u_4^1 \\ v_1^1 & v_2^1 & v_3^1 & v_4^1 \\ u_1^2 & u_2^2 & u_3^2 & u_4^2 \\ v_1^2 & v_2^2 & v_3^2 & v_4^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Affine Structure-from-Motion: Two Frames (4)

$$\begin{pmatrix} u_1^1 & u_2^1 & u_3^1 & u_4^1 \\ v_1^1 & v_2^1 & v_3^1 & v_4^1 \\ u_1^2 & u_2^2 & u_3^2 & u_4^2 \\ v_1^2 & v_2^2 & v_3^2 & v_4^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

We can solve for motion by inverting matrix of points.

Or, explicitly, we see that first column on left (images of first point) give the translations. After solving for these, we can solve for the each column of the *s* components of the motion using the images of each point, in turn.

Affine Structure-from-Motion: Two Frames

(5)

$$\begin{pmatrix} u_k^1 \\ v_k^1 \\ u_k^2 \\ v_k^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} \begin{pmatrix} a_k \\ b_k \\ c_k \\ 1 \end{pmatrix}$$

Once we know the motion, we can use the images of another point to solve for the structure. We have four linear equations, with three unknowns.

Affine Structure-from-Motion: Two Frames (6)

Suppose we just know where the *k*'th point is in image 1.

$$\begin{pmatrix} u_k^1 \\ v_k^1 \\ ? \\ ? \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} \begin{pmatrix} a_k \\ b_k \\ c_k \\ 1 \end{pmatrix}$$

Then, we can use the first two equations to write a_k and b_k as linear in c_k. The final two equations lead to two linear equations in the missing values and c_k. If we eliminate c_k we get one linear equation in the missing values. This means the unknown point lies on a known line. That is, we recover the epipolar constraint. Furthermore, these lines are all parallel.

Affine Structure-from-Motion: Two Frames (7)

But, what if the first four points aren't so simple?

Then we define *A* so that:

$$A\begin{pmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \\ 1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

This is always possible as long as the points aren't coplanar.

Affine Structure-from-Motion: Two Frames (8)

Then, given:

$$\begin{pmatrix} u_1^1 & u_2^1 & . & . & . & u_n^1 \\ v_1^1 & v_2^1 & & & & v_n^1 \\ u_1^2 & u_2^2 & & & & u_n^2 \\ v_1^2 & v_2^2 & & & & v_n^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \\ z_1 & z_2 & & & & z_n \\ 1 & 1 & & & & 1 \end{pmatrix}$$

We have:

$$\begin{pmatrix} u_1^1 & u_2^1 & . & . & . & u_n^1 \\ v_1^1 & v_2^1 & & & & v_n^1 \\ u_1^2 & u_2^2 & & & & u_n^2 \\ v_1^2 & v_2^2 & & & & v_n^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} A^{-1} A \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \\ z_1 & z_2 & & & & z_n \\ 1 & 1 & & & & 1 \end{pmatrix}$$

And:

$$\begin{pmatrix} u_1^1 & u_2^1 & . & . & . & u_n^1 \\ v_1^1 & v_2^1 & & & & v_n^1 \\ u_1^2 & u_2^2 & & & & u_n^2 \\ v_1^2 & v_2^2 & & & & v_n^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} A^{-1} \begin{pmatrix} 0 & 1 & 0 & 0 & . & . & . & x_n \\ 0 & 0 & 1 & 0 & & & & y_n \\ 0 & 0 & 0 & 1 & & & & z_n \\ 1 & 1 & 1 & 1 & & & & 1 \end{pmatrix}$$

Affine Structure-from-Motion: Two Frames (9)

Given:

$$\begin{pmatrix} u_1^1 & u_2^1 & . & . & . & u_n^1 \\ v_1^1 & v_2^1 & & & & v_n^1 \\ u_1^2 & u_2^2 & & & & u_n^2 \\ v_1^2 & v_2^2 & & & & v_n^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} A^{-1} \begin{pmatrix} 0 & 1 & 0 & 0 & . & . & . & x_n \\ 0 & 0 & 1 & 0 & & & & y_n \\ 0 & 0 & 0 & 1 & & & & z_n \\ 1 & 1 & 1 & 1 & & & & 1 \end{pmatrix}$$

Then we just pretend that:

$$\begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} A^{-1}$$

is our motion, and solve as before.

Affine Structure-from-Motion: Two Frames (10)

This means that we can never determine the exact 3D structure of the scene. We can only determine it up to some transformation, *A*. Since if a structure and motion explains the points:

$$\begin{pmatrix} u_1^1 & u_2^1 & . & . & . & u_n^1 \\ v_1^1 & v_2^1 & & & & v_n^1 \\ u_1^2 & u_2^2 & & & & u_n^2 \\ v_1^2 & v_2^2 & & & & v_n^2 \end{pmatrix} = \begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \\ z_1 & z_2 & & & & z_n \\ 1 & 1 & & & & 1 \end{pmatrix}$$

So does another image of the form:

$$\begin{pmatrix} u_1^1 & u_2^1 & . & . & . & u_n^1 \\ v_1^1 & v_2^1 & & & & v_n^1 \\ u_1^2 & u_2^2 & & & & u_n^2 \\ v_1^2 & v_2^2 & & & & v_n^2 \end{pmatrix} = \left(\begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{1,1}^2 & s_{1,2}^2 & s_{1,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} A^{-1} \right) \left(A \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \\ z_1 & z_2 & & & & z_n \\ 1 & 1 & & & & 1 \end{pmatrix} \right)$$

Affine Structure-from-Motion: Two Frames

(11)

$$\begin{pmatrix} u_1^1 & u_2^1 & . & . & . & u_n^1 \\ v_1^1 & v_2^1 & & & & v_n^1 \\ u_1^2 & u_2^2 & & & & u_n^2 \\ v_1^2 & v_2^2 & & & & v_n^2 \end{pmatrix} = \left(\begin{pmatrix} s_{1,1}^1 & s_{1,2}^1 & s_{1,3}^1 & t_x^1 \\ s_{2,1}^1 & s_{2,2}^1 & s_{2,3}^1 & t_y^1 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_x^2 \\ s_{2,1}^2 & s_{2,2}^2 & s_{2,3}^2 & t_y^2 \end{pmatrix} A^{-1} \right) \left(A \begin{pmatrix} x_1 & x_2 & . & . & . & x_n \\ y_1 & y_2 & & & & y_n \\ z_1 & z_2 & & & & z_n \\ 1 & 1 & & & & 1 \end{pmatrix} \right)$$

Note that *A* has the form:

A corresponds to translation of the points, plus a linear transformation.

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

For example, there is clearly a translational ambiguity in recovering the points. We can't tell the difference between two point sets that are identical up to a translation when we only see them after they undergo an unknown translation. Similarly, there's clearly a rotational ambiguity. The rest of the ambiguity is a stretching in an unknown direction.