TWO VIEW GEOMETRY

\[ p_l^T F p_r = 0 \]

- \( p_l, p_r \) are 2D homogenous co-ordinates of left and right points.
- Co-ordinates in "image space": \( F \) is called the fundamental matrix.
- So given a specific point \( p_r \), says \( p_l^T (F p_r) = 0 \).
  - This is the equation of a line!
- Same for the other way round.
- Has rank 2. Why?
- Vector \( p \) such that \( F p = [0, 0, 0]^T \).
- Means that this vector \( p \) will satisfy \( p_l^T F p \) for every \( p_l \).
- \( p \) is the homogeneous co-ordinate for the epipole in the right image.

Fundamental matrix has seven free parameters.
- One free parameter from scale.
Fundamental matrix has seven free parameters.
- One free parameter from scale.
- Require that $\det(F) = 0$
- Estimate using correspondences.
  - (see "eight point algorithm" in Szeliski 7.2 / Wikipedia)
If both cameras are calibrated, then only five unknowns
- Three for rotation
- Only two for translation!
- Only direction of translation matters. Epipolar lines stay the same irrespective of magnitude (how far you move the second camera in the same direction).
Epipolar Lines stay the same: relationship between depth and location changes.

If you compute fundamental matrix from image correspondences, you can use matching points to get depth only up to this unknown scale corresponding to translation magnitude.

Having arbitrary lines is a little annoying.
TWO VIEW GEOMETRY

Rectification

If the cameras were related only by translation, and viewing direction was orthogonal to the translation vector.

TWO VIEW GEOMETRY

Rectification

If the cameras were related only by translation, and viewing direction was orthogonal to the translation vector.

TWO VIEW GEOMETRY

Rectification

If the cameras were related only by translation, and viewing direction was orthogonal to the translation vector.

TWO VIEW GEOMETRY

Rectification

If the cameras were related only by translation, and viewing direction was orthogonal to the translation vector.
TWO VIEW GEOMETRY

Almost always, not ideal if camera moved in viewing direction. Too much distortion / non-overlap.

RECTIFIED BINOCULAR STEREO

Epipoles at infinity. Epipolar lines all parallel to the X axis.
RECTIFIED BINOCULAR STEREO

**Left**

L[x,y] matches to R[x',y]

**Right**

Epipolar Lines are Horizontal

Why are we doing this? Two equations tell us 3D position of point

**Left**

L[x,y] matches to R[x',y]

**Right**

Epipolar Lines are Horizontal

Visibility Constraint: (object in front of camera)

**Left**

L[x,y] matches to R[x',y]

**Right**

Epipolar Lines are Horizontal

Visibility Constraint: x' <= x (object in front of camera)
RECTIFIED BINOCULAR STEREO

Left
L[x,y] matches to R[x',y]

Right
Epipolar Lines are Horizontal

Visibility Constraint: x' ≤ x
(object in front of camera)

Left
L[x,y] matches to R[x-d[x,y],y]

Right
Epipolar Lines are Horizontal

d[x,y] ≥ 0 is called the "disparity map"

d[x,y] is inversely proportional to depth

Left
L[x,y] matches to R[x-d[x,y],y]

Right
Epipolar Lines are Horizontal

d[x,y] ≥ 0 is called the "disparity map"

d[x,y] is inversely proportional to depth

When is d[x,y] = 0?
**Left**

$L[x,y]$ matches to $R[x-d[x,y],y]$  

- $d[x,y] \geq 0$ is called the "disparity map"  
- $d[x,y]$ is inversely proportional to depth

**Right**

Epipolar Lines are Horizontal

- When is $d[x,y] = 0$?  
- When the point is at infinity.

---

**How do you find the correct match?**

- Smooth regions are ambiguous. Too many pixels on the right will look like a pixel on the left.
- Non-lambertian regions are ambiguous. The correct pixel on the right will not look like the pixel on the left.
How do you find the correct match?
- Smooth regions are ambiguous. Too many pixels on the right will look like a pixel on the left.
- Non-lambertian regions are ambiguous. The correct pixel on the right will not look like the pixel on the left.
- Occlusions: pixel on the left is NOT visible in the image on the right and vice-versa.

What is the right answer?
\[ d(x,y) = \text{Value that the pixel } (x,y) \text{ in the left image has moved by, even if it is not visible.} \]
In other words, \( x-d(x,y) \) should be the co-ordinate of the projection of that 3D surface point in the right image.

Useful because we want to eventually use it to estimate depth.

Step 1: At each location in left and right image, compute some representation of the appearance of (neighborhood around) that location.

Step 2: Define a "distance" function.

Step 3: Determine disparity as the "best match" according to this distance.
Option 1: Encoding = Intensity, Distance = Absolute Value

Noisy, unstable, susceptible to specular highlights

Option 2: Encoding = Gradients, Distance = Absolute Value

A little better.

A little better. But still susceptible to scaling.
Option 3: Encoding = Clipped Gradients, Distance = Absolute Value

(between -X,X)

Better. Essentially becomes a test between “signs of gradients”.

Census Transform

Encode a neighborhood as an integer, where each bit encodes if pixel at a different location had intensity higher than the center pixel.

\[ C[n] = \sum_j 2^j \delta(I[n] > I[n - n_j]) \]

Step 1: At each location in left and right image, compute some representation of the appearance of (neighborhood around) that location.

Step 2: Define a “distance” function.

Step 3: Determine disparity as the “best match” according to this distance.
**RECTIFIED BINOCULAR STereo**

Step 1: At each location in left and right image, compute some representation of the appearance of (neighborhood around) that location.

Step 2: Define a "distance" function. Hamming Distance

Step 3: Determine disparity as the "best match" according to this distance.

**COST VOLUMES**

Let us say we believe max value of disparity is D=1.
So by considering all possible matches, we are building a Walsh "cost volume"
Let us say we believe max value of disparity is D-1.
So by considering all possible matches, we are building a VolS of "cost volume"
Let us say we believe max value of disparity is D=1.
So by considering all possible matches, we are building a WristD "cost volume"
COST VOLUMES

\( C(x, y, d) \) measures the quality of the match between \( L(x, y) \) and \( R(x-d, y) \).

In problem set 3, you simply compute the best match independently at each \( (x, y) \):

\[
    d[x, y] = \arg \min_d C[x, y, d]
\]

This gives us pretty noisy disparity maps.

We've seen noise before. Smoothing helps. We could just smooth the disparity map?
**COST VOLUMES**

**Why?**

\[ d(x, y) = \arg \min_d C[x, y, d] \]

But that still gives us pretty noisy disparity maps.

We’ve seen noise before. Smoothing helps. We could just smooth the disparity map.

---

**COST VOLUMES**

**The errors in the disparity map can often be high magnitude. In a smooth region or with repeated texture, there may be a second seemingly good match very far away, and that’s what arg min chooses.**

\[ d(x, y) = \arg \min_d C[x, y, d] \]

But that still gives us pretty noisy disparity maps.

We’ve seen noise before. Smoothing helps. We could just smooth the disparity map.

---

**COST VOLUMES**

We want to express the fact that we expect our disparity map to be smooth.

\[ d(x, y) = \arg \min_d C[x, y, d] \]

But that still gives us pretty noisy disparity maps.

We’ve seen noise before. Smoothing helps. We could just smooth the disparity map.

---

**COST VOLUMES**

We want to express the fact that we expect our disparity map to be smooth.

But do it before we compute the arg min below.

\[ d(x, y) = \arg \min_d C[x, y, d] \]

But that still gives us pretty noisy disparity maps.

We’ve seen noise before. Smoothing helps. We could just smooth the disparity map.
Possible Solution: Smooth the cost volume!

d[x, y] = \arg \min_d C[x, y, d]

But that still gives us pretty noisy disparity maps

We've seen noise before. Smoothing helps. We could just smooth the disparity map?
Possible Solution: Smooth the cost volume!

Take each slice of the cost volume, and smooth it. Expresses the fact that if \((x, y)\) and \((x-d, y)\) match, then so should \((x+1, y)\) with \((x+1, y+1)\) with \((x+1, y+1)\) with \((x+1, y+1)\) with \((x+1, y+1)\)

Take \(\text{arg min} \, \overline{C}[x, y, d]\)

\[ d[x, y] = \arg \min_d \overline{C}[x, y, d] \]

But that still gives us pretty noisy disparity maps

We’ve seen noise before. Smoothing helps. We could just smooth the disparity map?

Possible Solution: Smooth the cost volume!

Take each slice of the cost volume, and smooth it. Expresses the fact that if \((x, y)\) and \((x-d, y)\) match, then so should \((x+1, y)\) with \((x+1, y+1)\) with \((x+1, y+1)\) with \((x+1, y+1)\) with \((x+1, y+1)\)

Take \(\text{arg min} \, \overline{C}[x, y, d]\)

\[ d[x, y] = \arg \min_d \overline{C}[x, y, d] \]
Here, "blurrier" means the location of disparity discontinuities, i.e. the contours, get spread out. It does not cause a more gradual change in the disparities themselves.
**Summary**

- We define a cost volume $C$ of size $W \times H \times D$
  - $C[x, y, d]$ measures dissimilarity between $(x, y)$ in left image and $(x - d, y)$ in right image
- Simplest Approach: $d[x, y] = \arg \min_d C[x, y, d]$
- Too noisy: want to express that disparity (and therefore depth) of nearby pixels is similar
- Ad-hoc Method: Cost Volume Filtering
- Still making independent decisions at each pixel.
- Averaging each disparity level promotes disparity maps where values are “equal” not close.
  - If $C[x, y, d]$ is a good match, then $C[x + 1, y, d \pm 1]$ gets no benefit from filtering.
  - Not good for slanted surfaces.