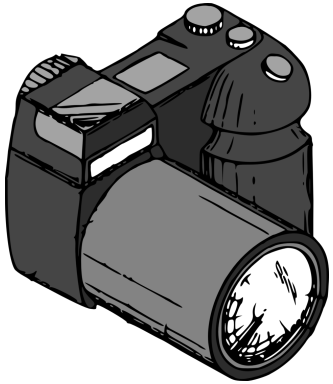
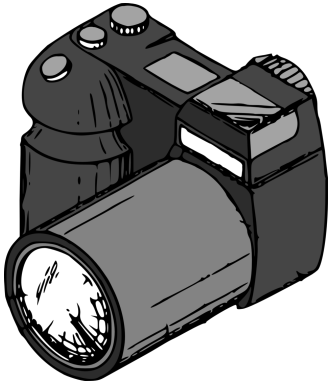
CSE 152 Section 5

HW2: Stereo Geometry

April 29, 2019

Owen Jow

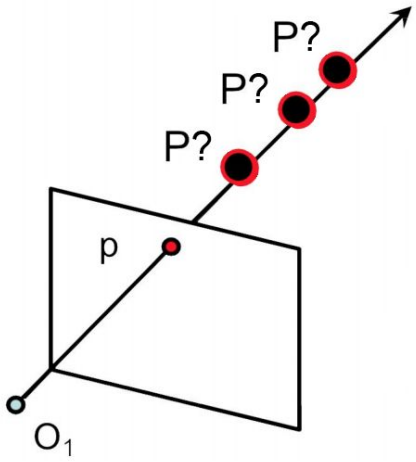




Stereo: two views.

Why is one view not sufficient?

1. Depth and Disparity

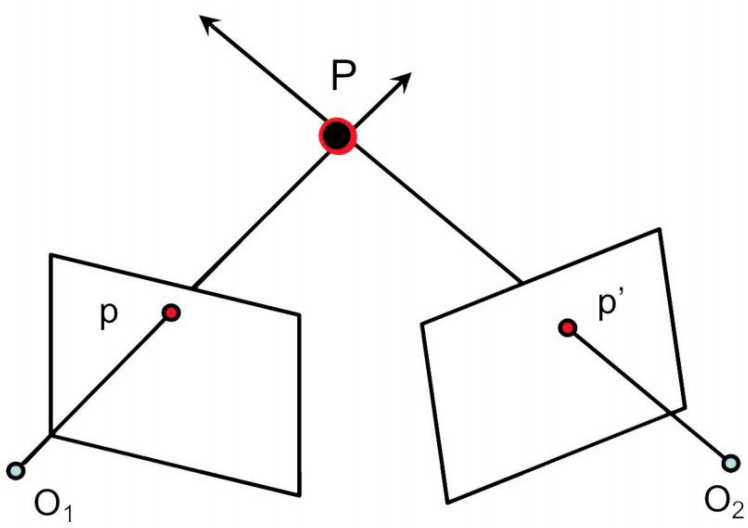


3D from a single view?

❌ **Ambiguity:** depth lost during projection.

If you multiply by the inverse intrinsic matrix, you'll get the direction of the 3D point, but you won't know exactly how far away it is.

1. Depth and Disparity



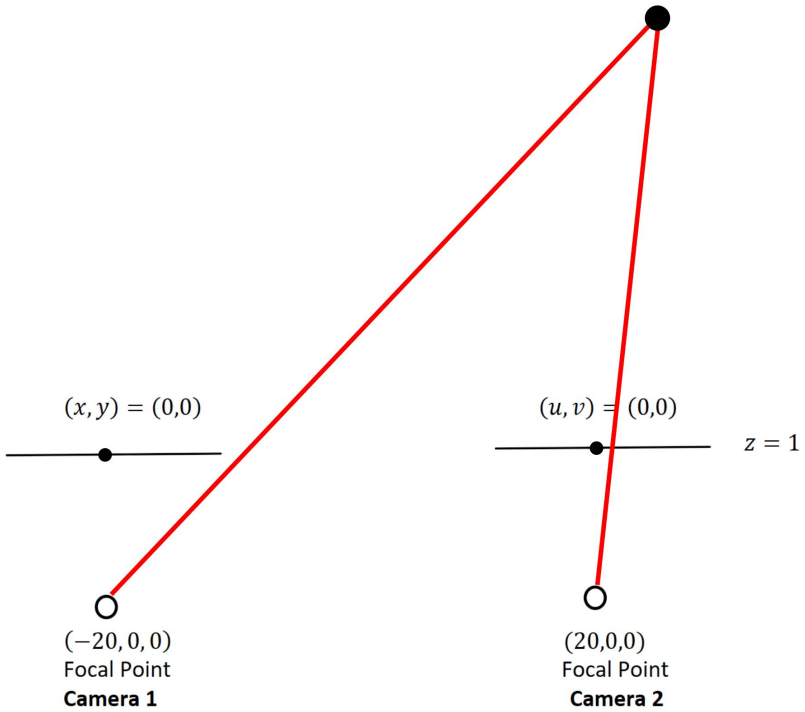
With **two** views, you can take a pair of corresponding **image points** and find the **3D point** as the intersection of rays from the centers of projection through the image points.

(...assuming perfect correspondence, which is the case in HW2 Problem 1)

1. Depth and Disparity

(**x**, **y**) = (12, 12) and (**u**, **v**) = (1, 12) are corresponding image points.

What is the associated 3D point?



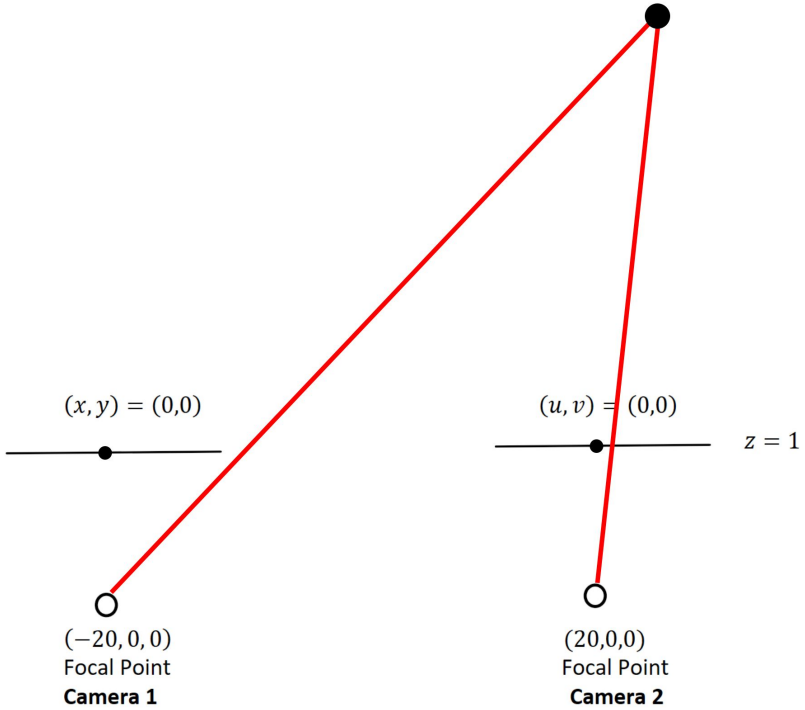
1. Depth and Disparity

(**x**, **y**) = (12, 12) and (**u**, **v**) = (1, 12) are corresponding image points.

What is the associated 3D point?

`Strategy 1.` Set up the equations as per Lecture 5 p19, solve problem in **XZ**-plane to determine **X** and **Z** in 3D, use **Z** to determine **Y** in 3D.

- Remember that camera 1's focal point is at (-20, 0, 0) not (0, 0, 0).

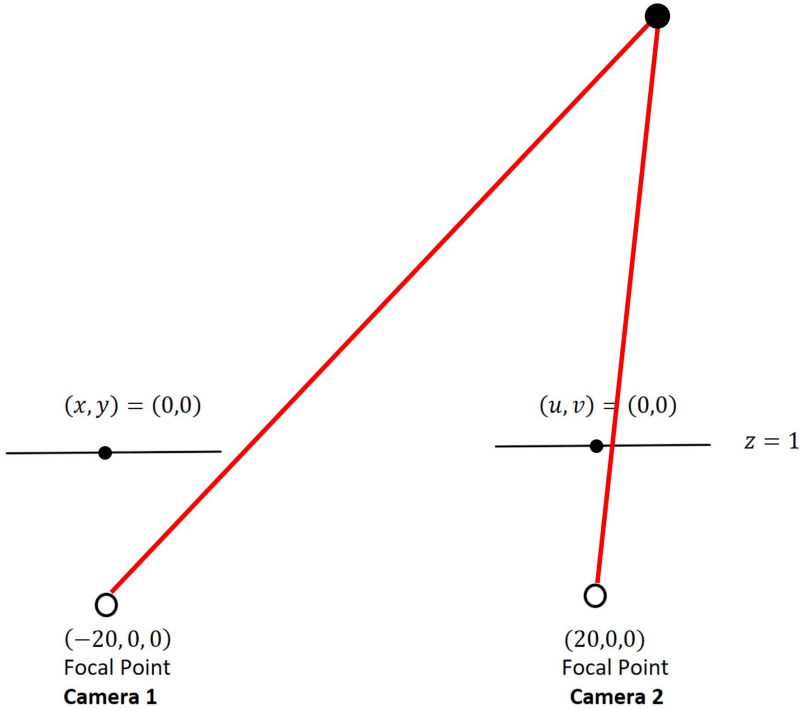


1. Depth and Disparity

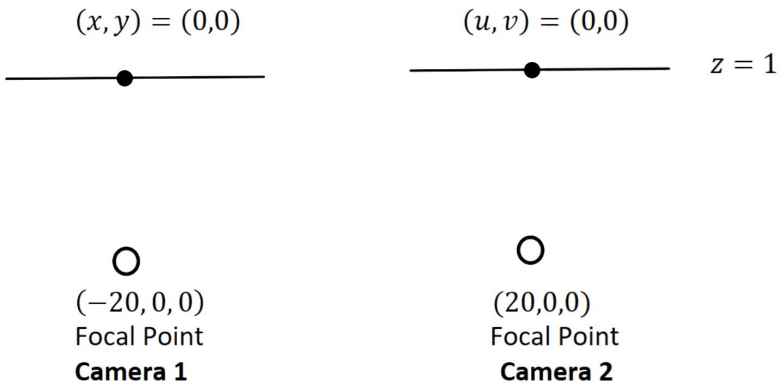
(**x**, **y**) = (12, 12) and (**u**, **v**) = (1, 12) are corresponding image points.

What is the associated 3D point?

`Strategy 2.` Set up the **o + td** equations for the two 3D rays and solve for their intersection.



1. Depth and Disparity

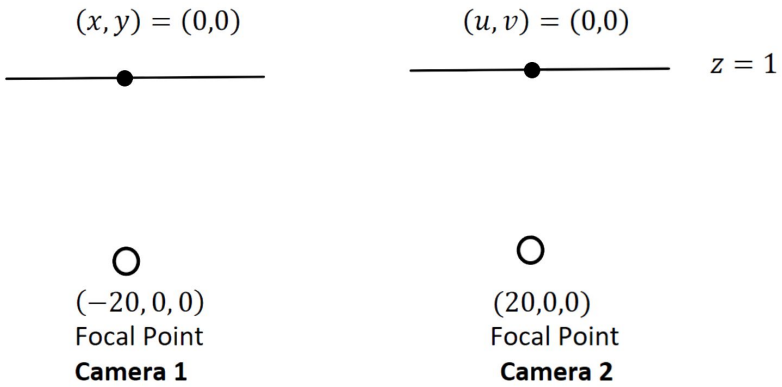


Deriving an expression for disparity:

- **x**-disparity: **x - u**
- **y**-disparity: **y - v**

(Do we need to worry about **y**-disparity? Why or why not?)

1. Depth and Disparity



Deriving an expression for disparity:

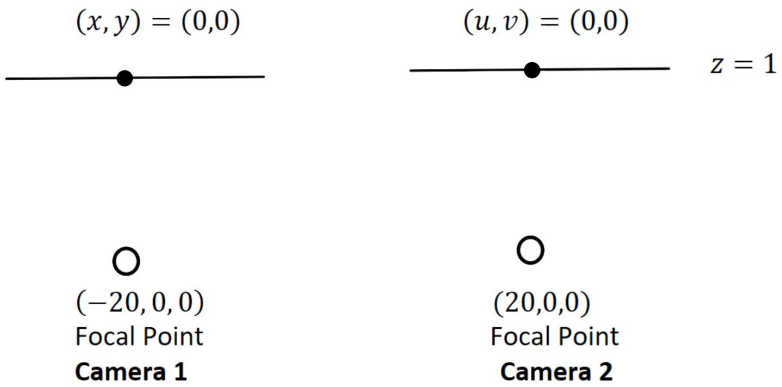
- **x**-disparity: **x - u**

We're interested in points on the line **X** + **Z** = 0, for 3D **X** and **Z**.

How does **X** relate to **x**?

Take a look at how $\mathbf{X}_R$ is computed on Lecture 5 p19!

1. Depth and Disparity



Deriving an expression for disparity:

- **x**-disparity: **x - u**

We're interested in points on the line **X** + **Z** = 0, for 3D **X** and **Z**.

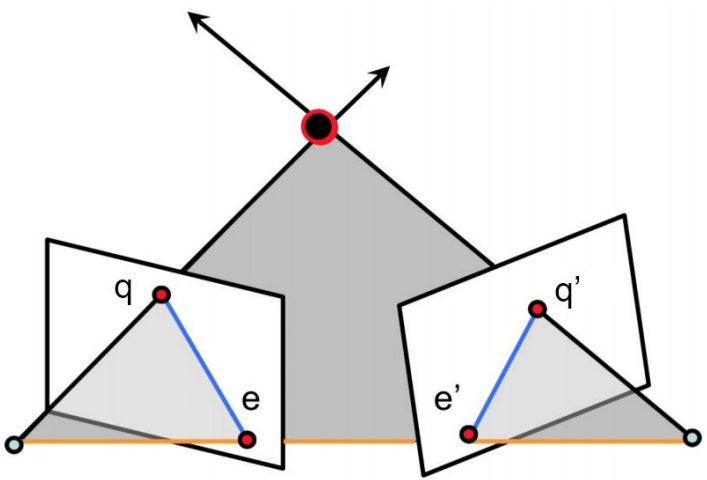
How does **u** relate to **X**?

Don't forget to put everything in terms of **u**!

We need correspondences to get 3D.

How can we efficiently establish these correspondences?

THE FUNDAMENTAL MATRIX



$$q^T F q' = 0$$

Epipolar constraint

$$\{q : q^T(Fq') = 0\} \quad \{q' : (F^T q)^T q' = 0\}$$

Epipolar line in image 1 Epipolar line in image 2

$$e^T F q' = 0 \ \forall \ q' \implies F^T e = 0$$
$$(F^T q)^T e' = 0 \ \forall \ q \implies F e' = 0$$

Epipole equations

The fundamental matrix **F** relates corresponding points in stereo images.

Given a point in one image, it'll constrain the location of the corresponding point in the other image.

2a. Computing the Fundamental Matrix

We can estimate the fundamental matrix using the **eight-point algorithm**.

Input: 8+ pairs of corresponding points $\mathbf{q_i = (x_i, y_i, 1)}$, $\mathbf{q_i' = (x_i', y_i', 1)}$
Output: fundamental matrix **F**

each pair of corresponding points yields one equation $\mathbf{q_i^T F q_i' = 0}$

$$\begin{bmatrix} x_i & y_i & 1 \end{bmatrix} \begin{bmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{bmatrix} \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = 0$$

$$x_i x_i' F_{11} + x_i y_i' F_{12} + x_i F_{13} + y_i x_i' F_{21} + y_i y_i' F_{22} + y_i F_{23} + x_i' F_{31} + y_i' F_{32} + F_{33} = 0$$

$$\begin{bmatrix} x_i x_i' & x_i y_i' & x_i & y_i x_i' & y_i y_i' & y_i & x_i' & y_i' & 1 \end{bmatrix} \begin{bmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{bmatrix} = 0$$

2a. Computing the Fundamental Matrix

Approach: find a least-squares solution to the following system of equations.
But we don't want the trivial solution **f = 0**. Since **f** is homogeneous, let's enforce that its norm be 1.

$$\begin{bmatrix} x_1x_1' & x_1y_1' & x_1 & y_1x_1' & y_1y_1' & y_1 & x_1' & y_1' & 1 \\ & & & & \vdots & & & & \\ x_nx_n' & x_ny_n' & x_n & y_nx_n' & y_ny_n' & y_n & x_n' & y_n' & 1 \end{bmatrix} \begin{bmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{bmatrix} = 0$$

$$A\mathbf{f} = 0$$

$$\min_{\|\mathbf{f}\|=1} \|A\mathbf{f}\|^2 \rightarrow \min_{\|\mathbf{f}\|=1} \mathbf{f}^T A^T A\mathbf{f}$$

We want the eigenvector **f** associated with the smallest eigenvalue of $\mathbf{A}^T\mathbf{A}$.

→ i.e. the right singular vector corresponding to the smallest singular vector of A

2a. Computing the Fundamental Matrix

The **rank** of the fundamental matrix is **2**.
(It represents a non-invertible mapping from points to lines.)
To enforce this, we take another SVD and zero out the last singular value in the decomposition.

$$F = U \begin{bmatrix} s_1 & 0 & 0 \\ 0 & s_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

2a. Computing the Fundamental Matrix

Also, $\mathbf{A}^T\mathbf{A}$ is typically extremely ill-conditioned.
It might contain values all over the place from, say, 1 to $1{,}000^4$ (= 1,000,000,000,000).
To rectify this, we will normalize the image coordinates before constructing the **A** matrix.
Then our computed **F** will be meant for normalized points, so we'll have to de-normalize it.

$$(T_1 q)^T F (T_2 q') = 0$$
$$\rightarrow q^T (T_1^T F T_2) q' = 0$$

Note that almost all of this is already implemented.
For 2a, all you need to do is construct the A matrix.

2b. Plotting Epipolar Lines

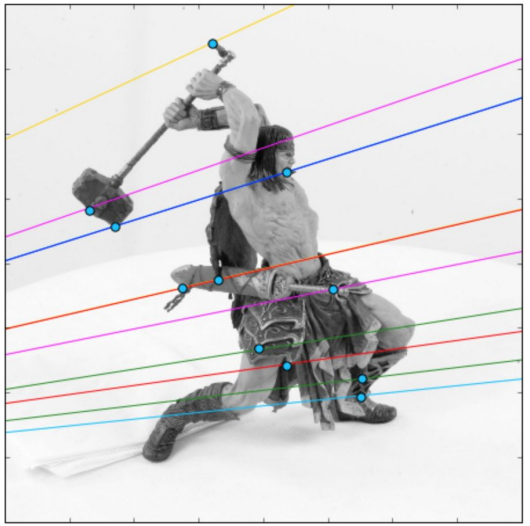
The epipolar line associated with **q'** is $\ell' = \mathbf{Fq'}$.

- If $\ell = \mathbf{[a, b, c]}^T$ and $\mathbf{q = [x, y, 1]}^T$,
  then the equation of the line is $\ell^T\mathbf{q = ax + by + c = 0}$.

The epipolar line associated with **q** is $\ell' = \mathbf{F}^T\mathbf{q}$.

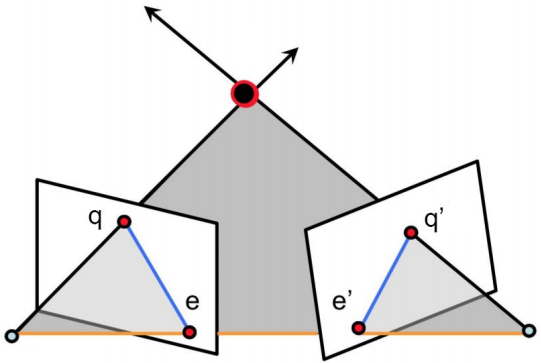
- If $\ell' = \mathbf{[a', b', c']}^T$ and $\mathbf{q' = [x', y', 1]}^T$,
  then the equation of the line is $\ell'^T\mathbf{q' = a'x' + b'y' + c' = 0}$.

(Useful function: `matplotlib.pyplot.plot`.)



2c. Computing the Epipoles

- The epipole in image 1 is the solution to $\mathbf{F}^T\mathbf{e} = \mathbf{0}$.
- The epipole in image 2 is the solution to $\mathbf{F}\mathbf{e} = \mathbf{0}$.



You can use SVD (`np.linalg.svd`) to solve this too.
To compute the right nullspace of **M**, take the SVD:

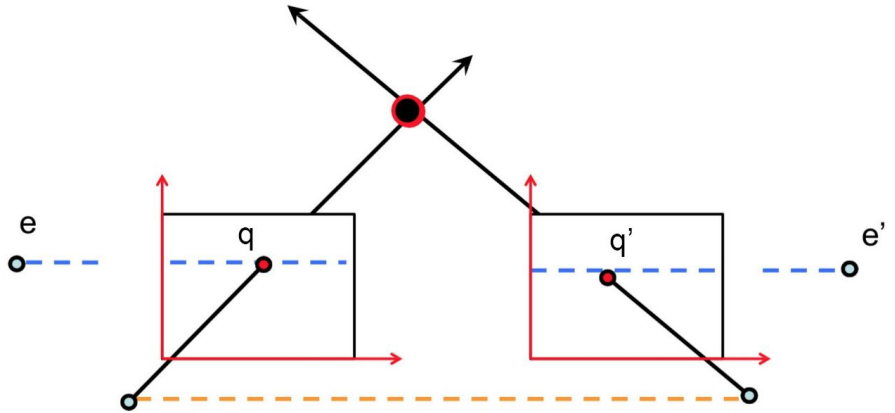
$$\mathbf{M} = \mathbf{U}\mathbf{S}\mathbf{V}^T$$

The right nullspace of **M** will be the rightmost column of **V**
(assuming the columns are listed in descending order of singular value).

2d. Image Rectification

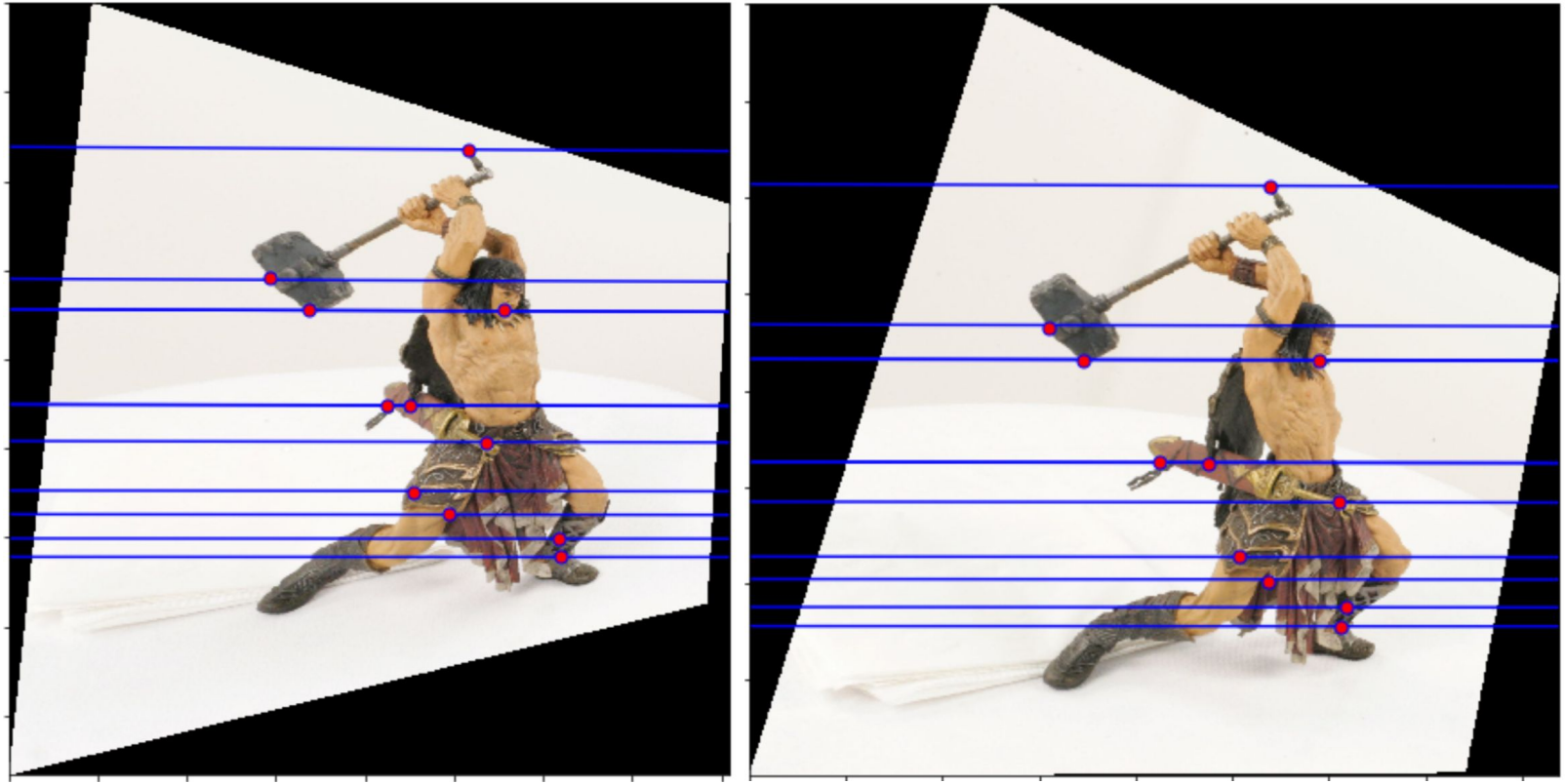
- To rectify, map the epipoles to horizontal infinity (1, 0, 0).

epipolar lines
become scan lines →



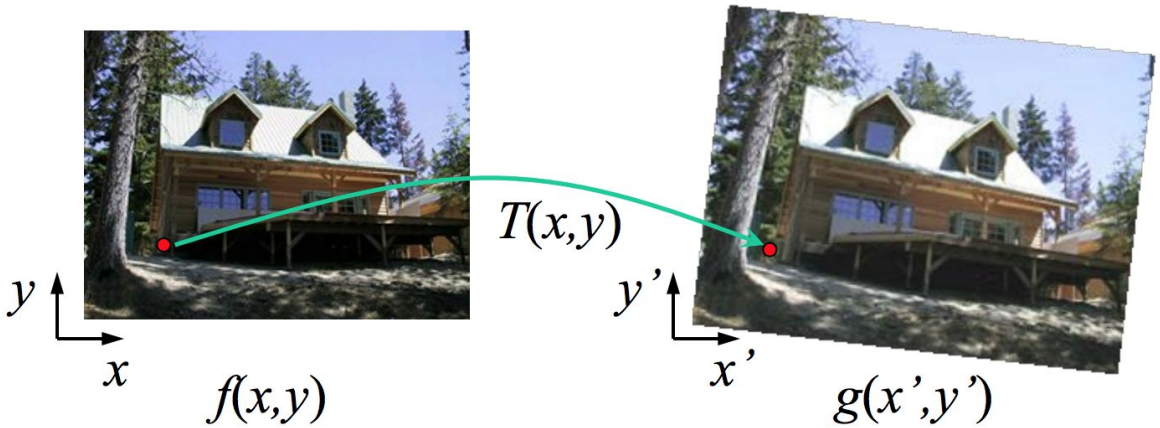
We've already provided the code to compute the homographies for both images. All you have to do is apply them.

2d. Image Rectification



Warping

- We want to apply a transformation on the coordinates (**warping**)
  ...as opposed to the values at the coordinates (**filtering**).
- The naive approach is to apply the forward transform to all of the input coordinates, figure out where they go, and copy the values accordingly.

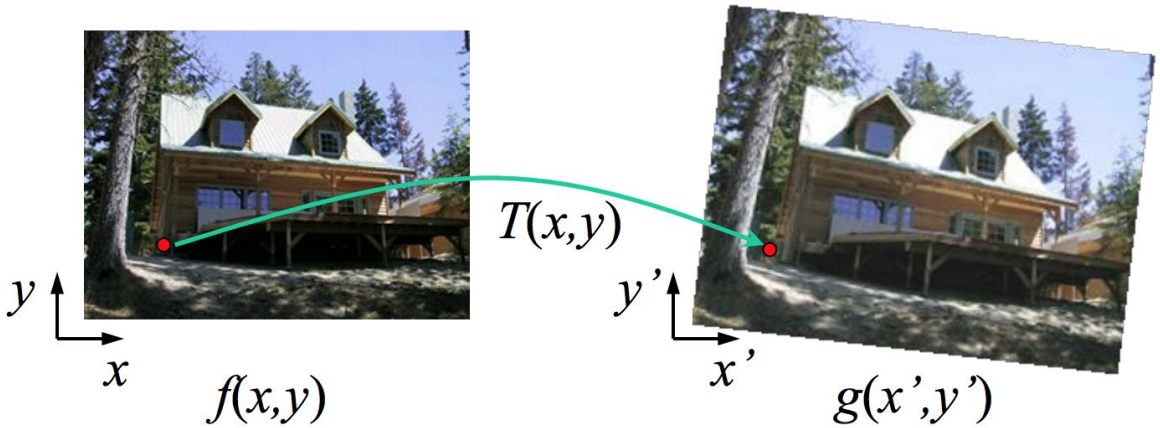


What could go wrong?

image credit: CS 194-26

Warping

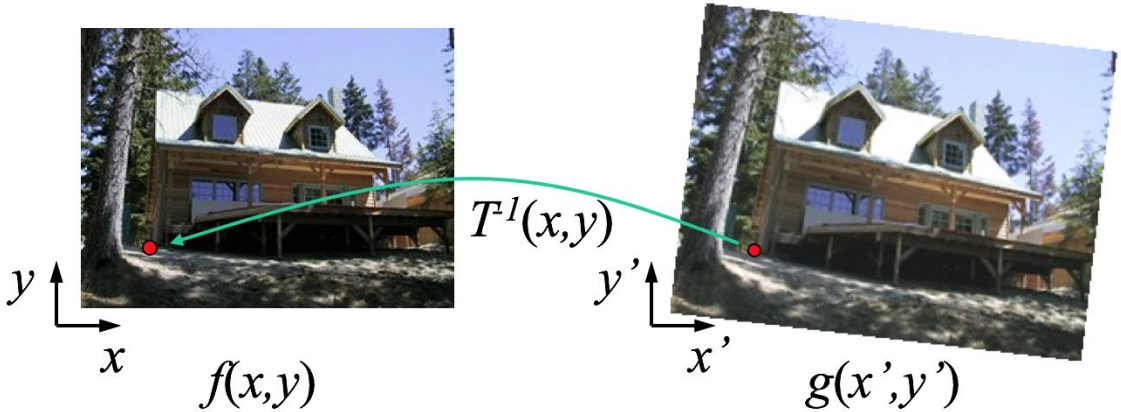
- We want to apply a transformation on the coordinates (**warping**)
  ...as opposed to the values at the coordinates (**filtering**).
- The naive approach is to apply the forward transform to all of the input coordinates, figure out where they go, and copy the values accordingly.



What could go wrong?
Might not hit every location in the output image (holes).

image credit: CS 194-26

Inverse Warping

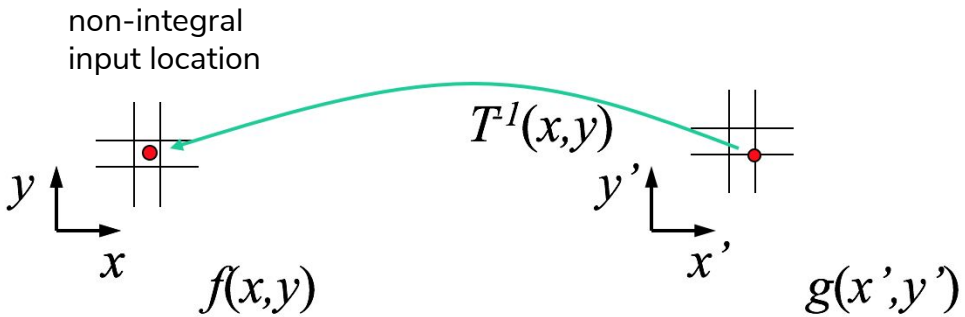


Better: explicitly determine a value for every output location.

(For every output location, apply the **inverse** coordinate transform to identify the corresponding input location. Then fill the output location with the associated input value.)

image credit: CS 194-26

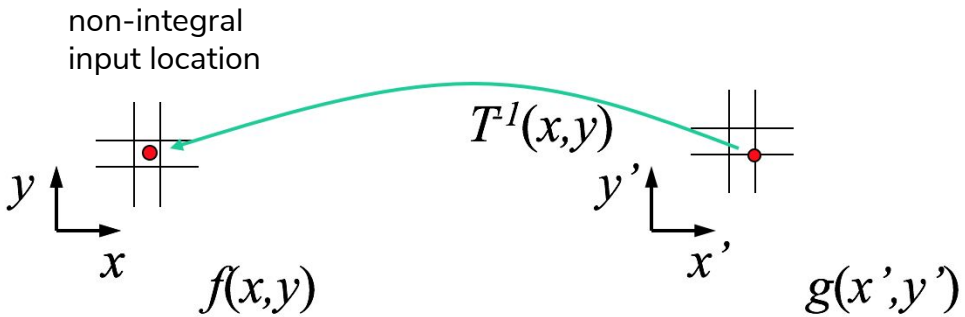
Inverse Warping



What if the pixel comes from “between” two pixels?

slide credit: CS 194-26

Inverse Warping

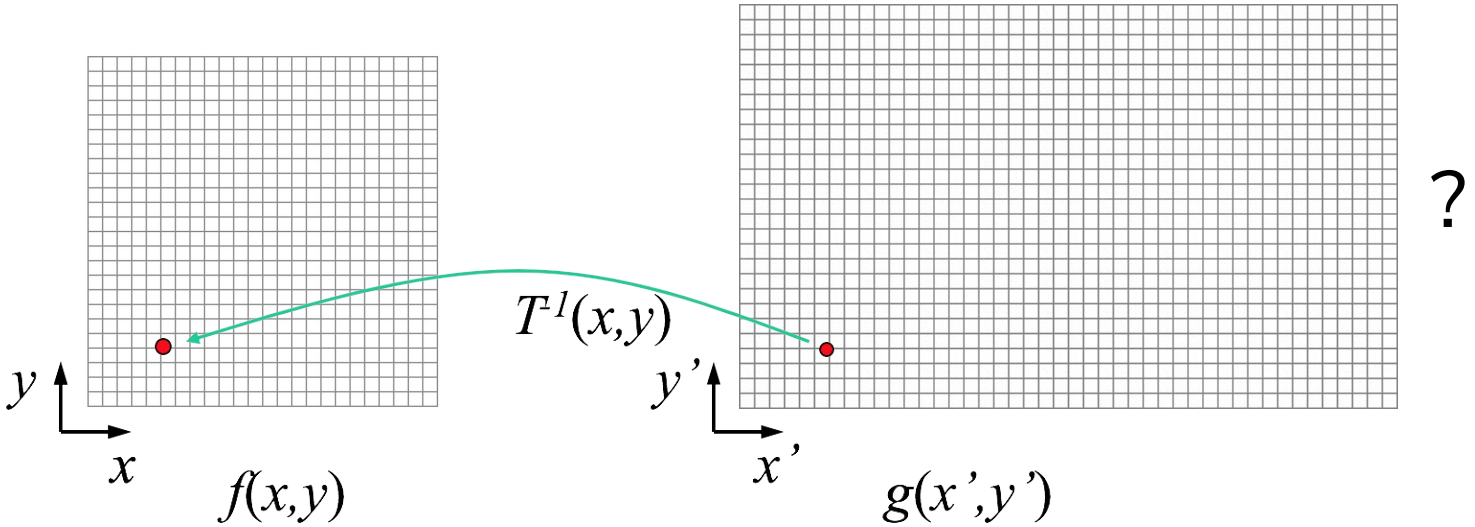


What if the pixel comes from “between” two pixels?

Take the nearest neighbor value (simplest), or bilinearly interpolate.

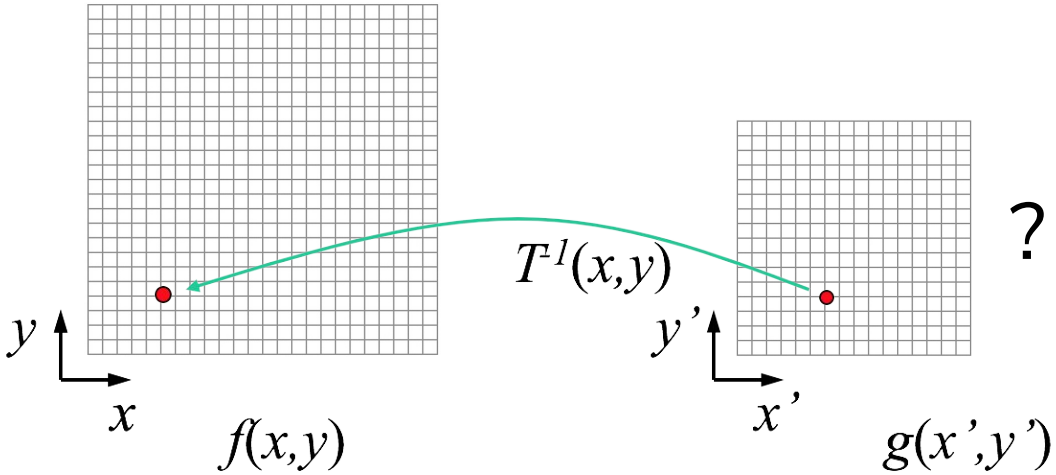
slide credit: CS 194-26

Inverse Warping



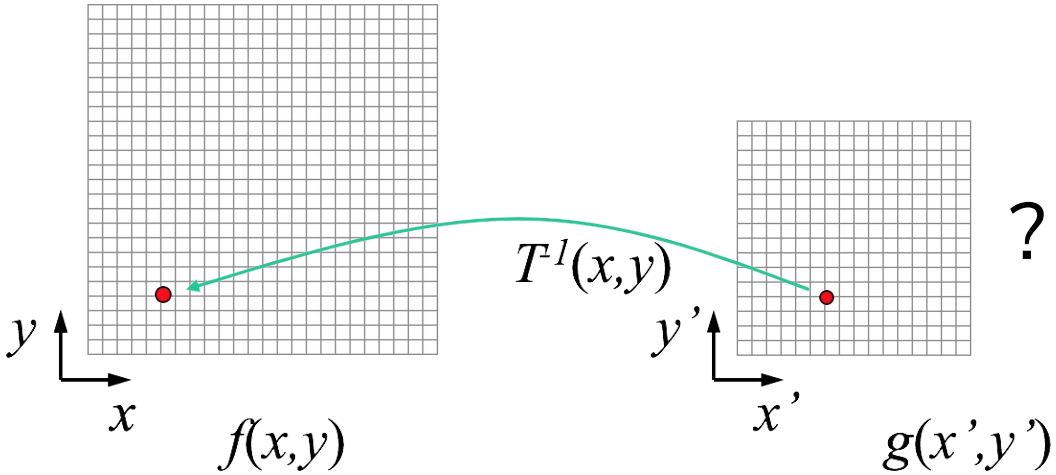
What range of coordinates to use for the output image?

Inverse Warping



What range of coordinates to use for the output image?

Inverse Warping



What range of coordinates to use for the output image?

Pipe the corner coordinates of the input image through the forward transform to determine the bounds for the output image.

Inverse Warping

1. Determine bounds of output image.
2. Apply inverse coordinate transform to all output coordinates.
   a. “for each output location, find out which input location corresponds to it”
3. Assign values to output locations according to their corresponding input locations.
   a. nearest-neighbor interpolation should suffice (round to nearest integer)

Useful Functions:

- `np.indices`, `np.meshgrid`
  - gives you all the x- and y-coordinates in a grid

```
x = np.arange(-1, 5)
y = np.arange(-3, 3)
xx, yy = np.meshgrid(x, y)
```

```
xx
[[-1  0  1  2  3  4]
 [-1  0  1  2  3  4]
 [-1  0  1  2  3  4]
 [-1  0  1  2  3  4]
 [-1  0  1  2  3  4]
 [-1  0  1  2  3  4]]
```

```
yy
[[-3 -3 -3 -3 -3 -3]
 [-2 -2 -2 -2 -2 -2]
 [-1 -1 -1 -1 -1 -1]
 [ 0  0  0  0  0  0]
 [ 1  1  1  1  1  1]
 [ 2  2  2  2  2  2]]
```

np.meshgrid example continued

```
x = np.arange(-1, 5)
y = np.arange(-3, 3)
xx, yy = np.meshgrid(x, y)
```

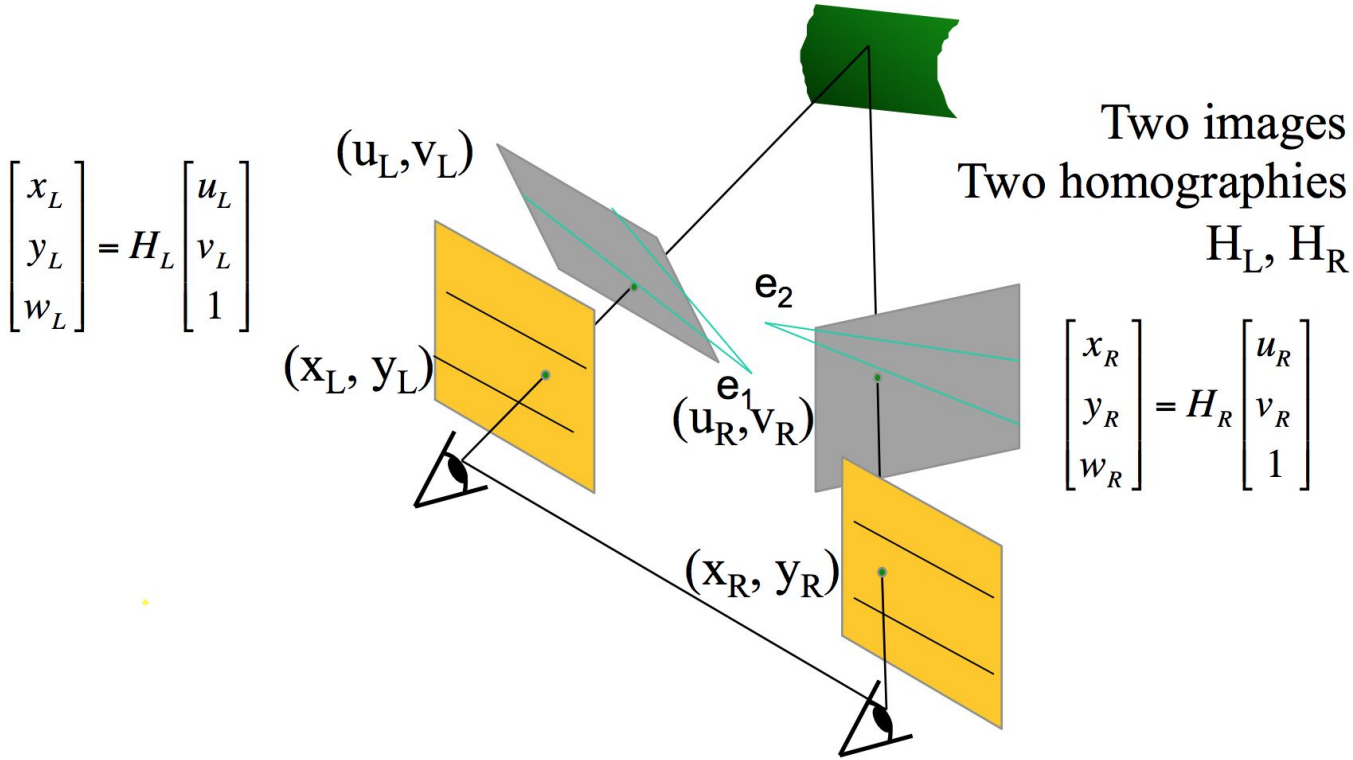
```
xx
[[-1  0  1  2  3  4]
 [-1  0  1  2  3  4]
 [-1  0  1  2  3  4]
 [-1  0  1  2  3  4]
 [-1  0  1  2  3  4]
 [-1  0  1  2  3  4]]
```

```
yy
[[-3 -3 -3 -3 -3 -3]
 [-2 -2 -2 -2 -2 -2]
 [-1 -1 -1 -1 -1 -1]
 [ 0  0  0  0  0  0]
 [ 1  1  1  1  1  1]
 [ 2  2  2  2  2  2]]
```

In [3]:
```
xy_coords = np.stack((xx, yy), axis=-1).reshape(-1, 2)
print('xy_coords.shape:', xy_coords.shape)
xy_coords[:7]
```

```
xy_coords.shape: (36, 2)
```

Out[3]:
```
array([[-1, -3],
       [ 0, -3],
       [ 1, -3],
       [ 2, -3],
       [ 3, -3],
       [ 4, -3],
       [-1, -2]])
```



A 3x3 homography maps **2D homogeneous coordinates** to **2D homogeneous coordinates**. You'll need to convert between Euclidean and homogeneous coordinates.

What if we have some not-so-good correspondences?

If we use any of them to estimate the fundamental matrix, it probably won’t end well.

3c. RANSAC for Fundamental Matrix Estimation

Robust model-fitting (+inlier detection) in the presence of outliers.

1. For `nSample` iterations:
   a. Pick eight points at random (useful function: `np.random.choice`).
   b. Use them to estimate **F** according to the eight-point algorithm.
   c. Count the total number of points that agree with **F** up to some threshold ("inliers").
      i. e.g. for each point $\mathbf{q}_i$, check how close $\mathbf{q}_i^T \mathbf{F} \mathbf{q}_i$' is to 0
   d. If the number of inliers is the highest seen so far, save all of the inliers.
2. Recompute **F** with the max-size set of inliers.
3. Recompute the set of inliers using the final **F**.

the greatest musical composition of all time

courtesy Daniel Wedge