UNIVERSITAT DE BARCELONA

>>> IMAGE PROCESSING AND COMPUTATIONAL PHOTOGRAPHY

SESSION 10: HOMOGRAPHIES, PANORAMAS AND PERSPECTIVE

Oriol Pujol & Simone Balocco

Inspired by D. Hoiem slides

Why Mosaic?

Are you getting the whole picture?

- Compact Camera FOV = 50 x 35°

Slide from Brown & Lowe

Why Mosaic?

Are you getting the whole picture?

- Compact Camera FOV = 50 x 35°
- Human FOV = 200 x 135°

Slide from Brown & Lowe

Why Mosaic?

Are you getting the whole picture?

- Compact Camera FOV = 50 x 35°
- Human FOV = 200 x 135°
- Panoramic Mosaic = 360 x 180°

Slide from Brown & Lowe

Mosaics: stitching images together

virtual wide-angle camera

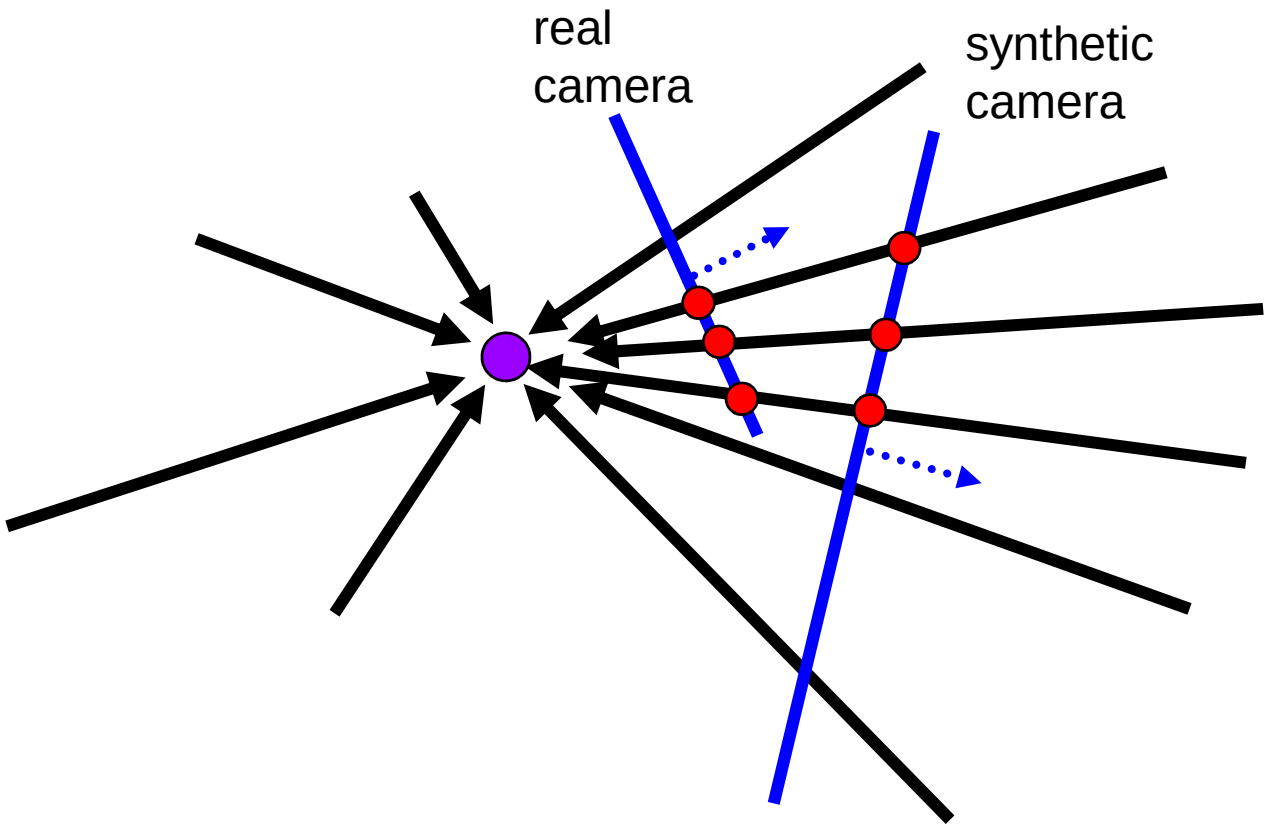
Naïve Stitching

left on top

right on top

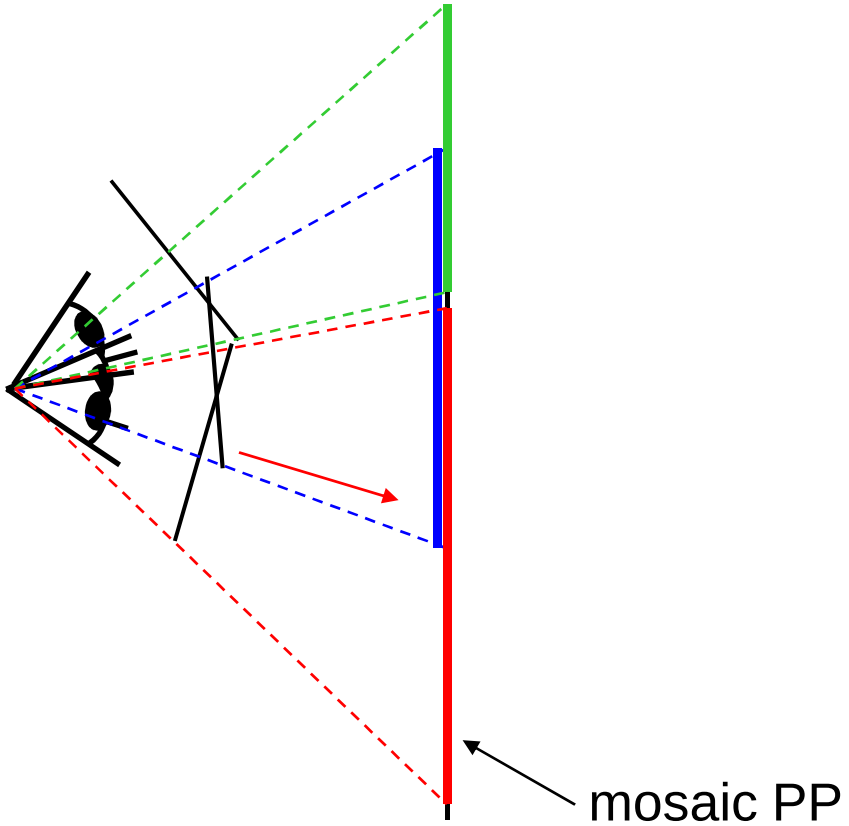
Translations are not enough to align the images

A pencil of rays contains all views



Can generate any synthetic camera view
as long as it has **the same center of projection**!

Image reprojection



The mosaic has a natural interpretation in 3D

- The images are reprojected onto a common plane
- The mosaic is formed on this plane
- Mosaic is a *synthetic wide-angle camera*

How to do it?

Basic Procedure

- Take a sequence of images from the same position
  - Rotate the camera about its optical center
- Compute transformation between second image and first
- Transform the second image to overlap with the first
- Blend the two together to create a mosaic
- If there are more images, repeat

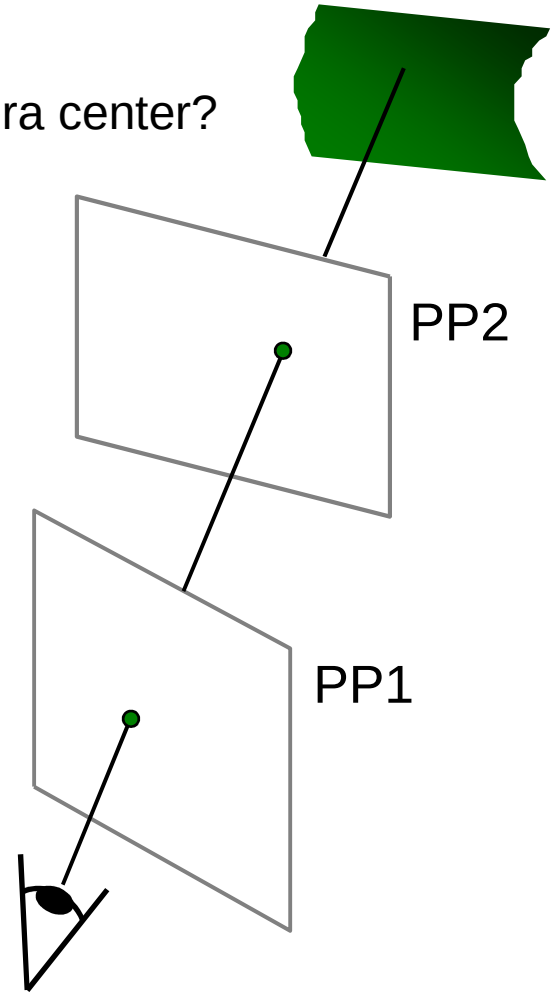
…but **wait**, why should this work at all?

- What about the 3D geometry of the scene?
- Why aren’t we using it?

Image reprojection

Basic question

- How to relate two images from the same camera center?
  - how to map a pixel from PP1 to PP2



Answer

- Cast a ray through each pixel in PP1
- Draw the pixel where that ray intersects PP2

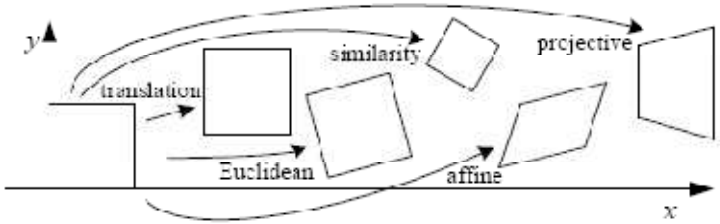
But don’t we need to know the geometry of the two planes in respect to the eye?

Observation:
Rather than thinking of this as a 3D reprojection, think of it as a 2D **image warp** from one image to another

Back to Image Warping

Which t-form is the right one for warping PP1 into PP2?

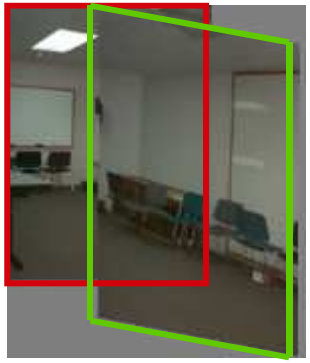
e.g. translation, Euclidean, affine, projective



Translation

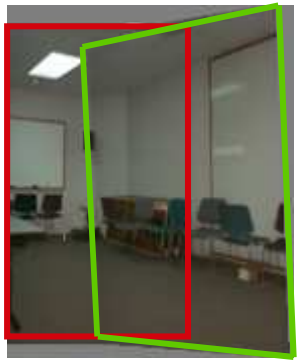
2 unknowns

Affine



6 unknowns

Perspective



8 unknowns

Homography

A: Projective – mapping between any two PPs with the same center of projection

- rectangle should map to arbitrary quadrilateral
- parallel lines aren't
- but must preserve straight lines
- same as: project, rotate, reproject

called Homography

$$\underset{p'}{\begin{pmatrix} wx' \\ wy' \\ w \end{pmatrix}} = \underset{\mathbf{H}}{\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}} \underset{p}{\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}}$$

To apply a homography **H**

- Compute **p'** = **Hp** (regular matrix multiply)
- Convert **p'** from homogeneous to image coordinates

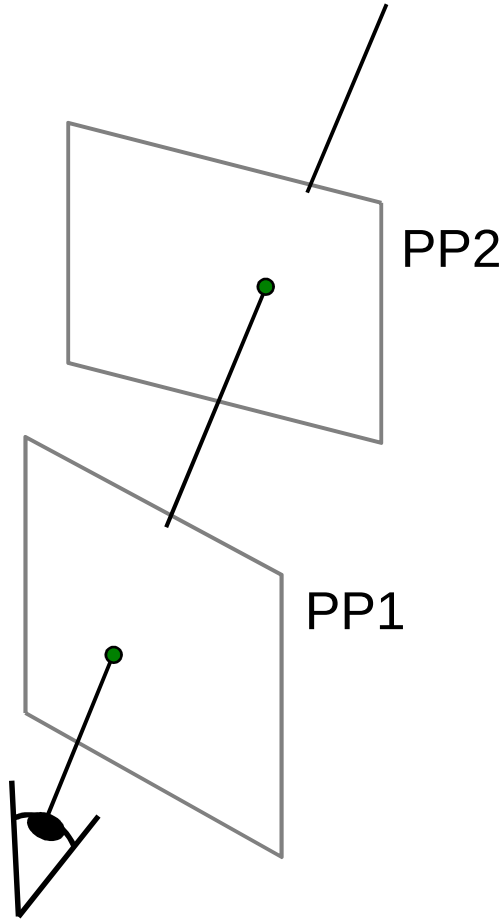


Image warping with homographies

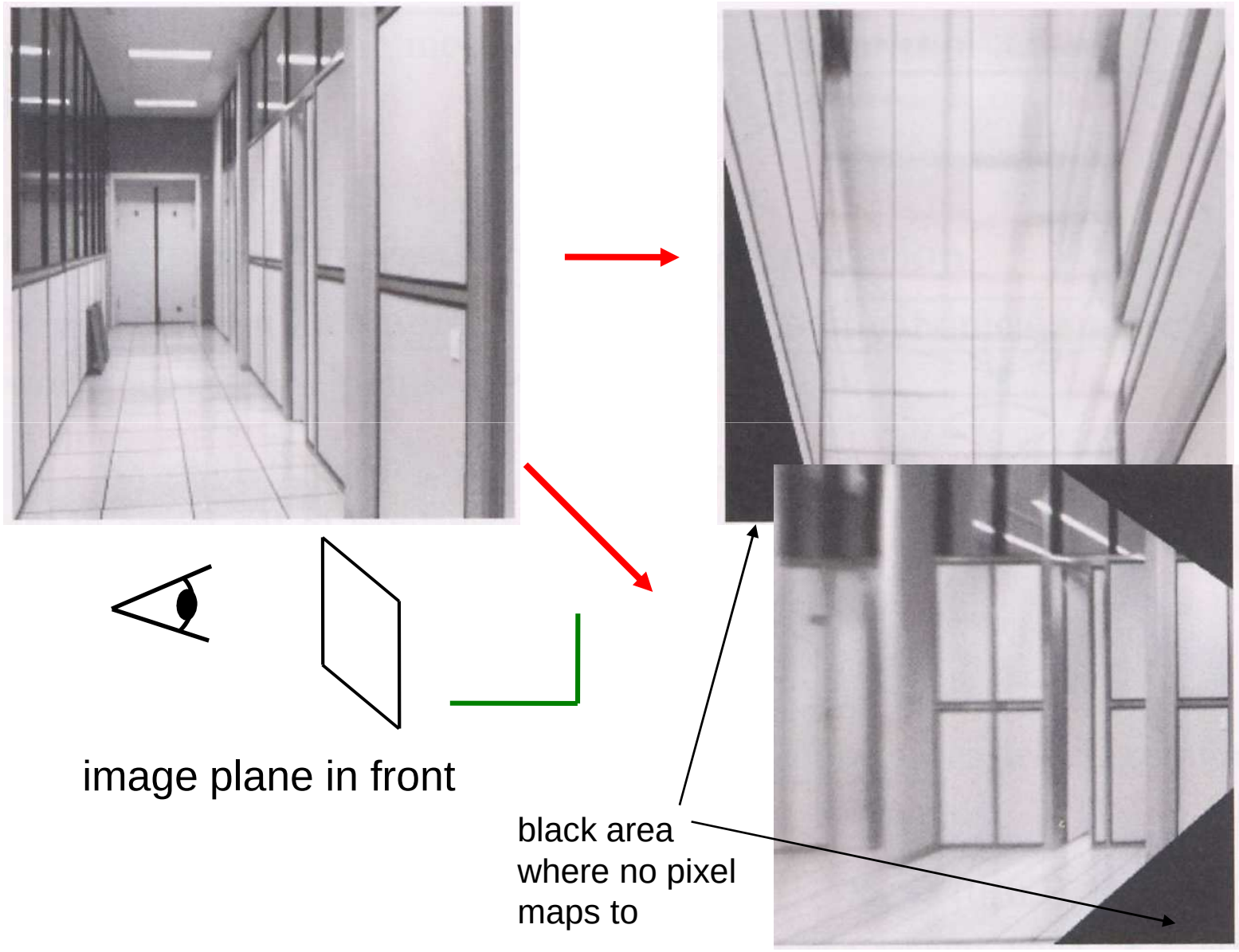
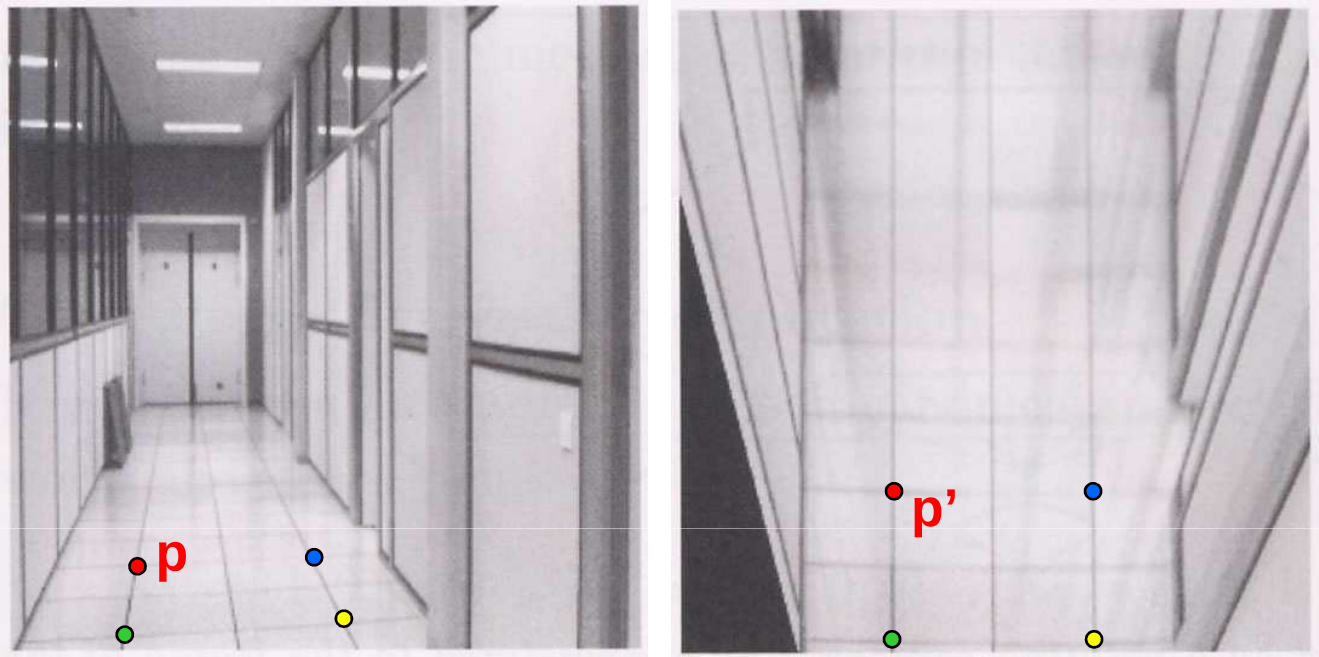


Image rectification



To unwarp (rectify) an image

- Find the homography **H** given a set of **p** and **p’** pairs
- How many correspondences are needed?
- Tricky to write H analytically, but we can solve for it!
  - Find such H that “best” transforms points p into p’
  - Use least-squares!

Least Squares Example

Say we have a set of data points (X1,X1'), (X2,X2'), (X3,X3'), etc. (e.g. person's height vs. weight)

We want a nice compact formula (a line) to predict X's from Xs: Xa + b = X'

We want to find a and b

How many (X,X') pairs do we need?

$$X_1 a + b = X'_1$$
$$X_2 a + b = X'_2$$

$$\begin{bmatrix} X_1 & 1 \\ X_2 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} X'_1 \\ X'_2 \end{bmatrix}$$

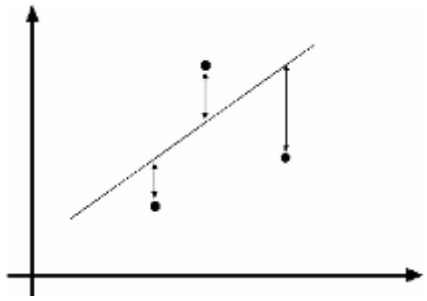
Ax=B

What if the data is noisy?

$$\begin{bmatrix} X_1 & 1 \\ X_2 & 1 \\ X_3 & 1 \\ \dots & \dots \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} X'_1 \\ X'_2 \\ X'_3 \\ \dots \end{bmatrix}$$

overconstrained

$$\min \| Ax - B \|^2$$



Solving for homographies

$$\mathbf{p'} = \mathbf{Hp}$$

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Can set scale factor *i*=1. So, there are 8 unkowns.

Set up a system of linear equations:

$$\mathbf{Ah = b}$$

where vector of unknowns h = [a,b,c,d,e,f,g,h]^T

Need at least 8 eqs, but the more the better…

Solve for h. If overconstrained, solve using least-squares:

$$\min \| Ah - b \|^2$$

Can be done in Matlab using "\" command

- see "help lmdivide"

Fun with homographies

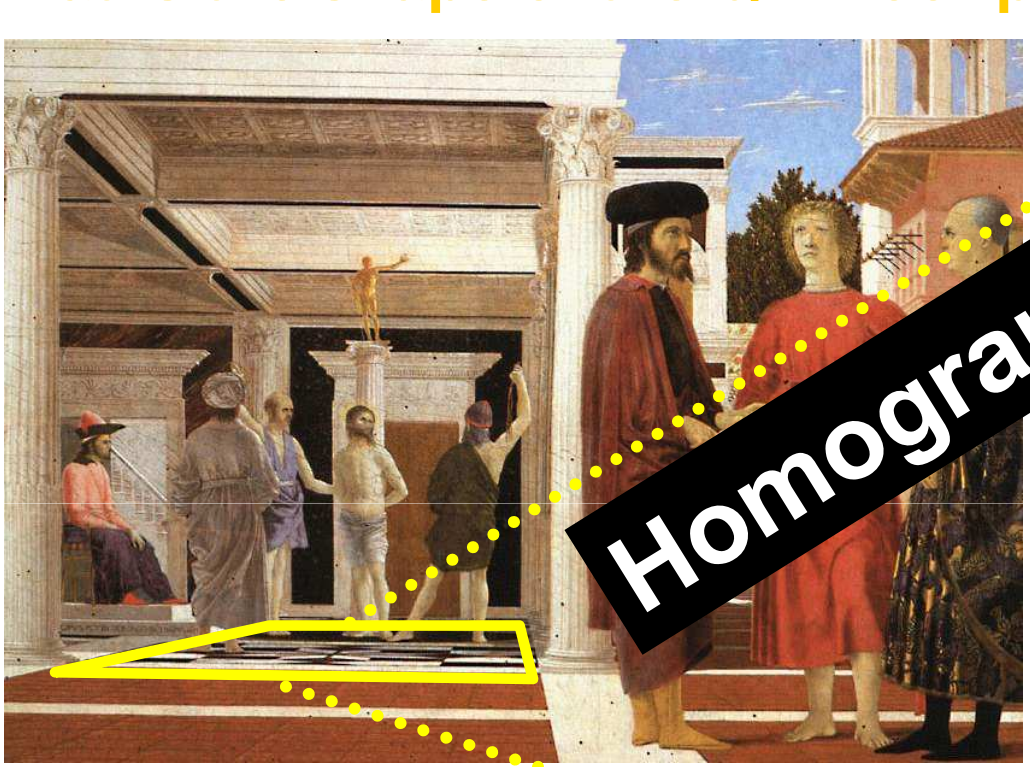
Original image

St.Petersburg
photo by A. Tikhonov

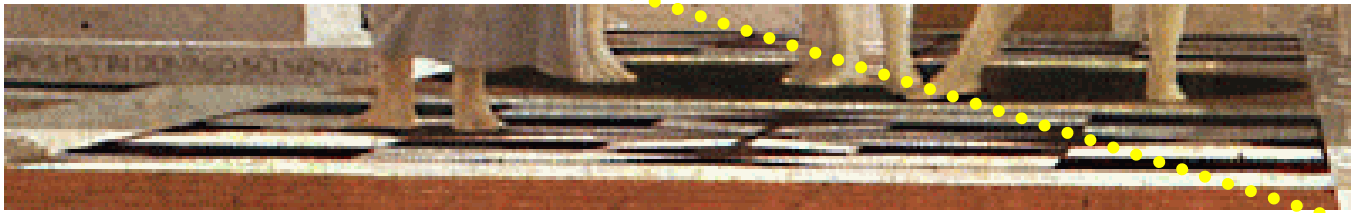
Virtual camera rotations

Analysing patterns and shapes

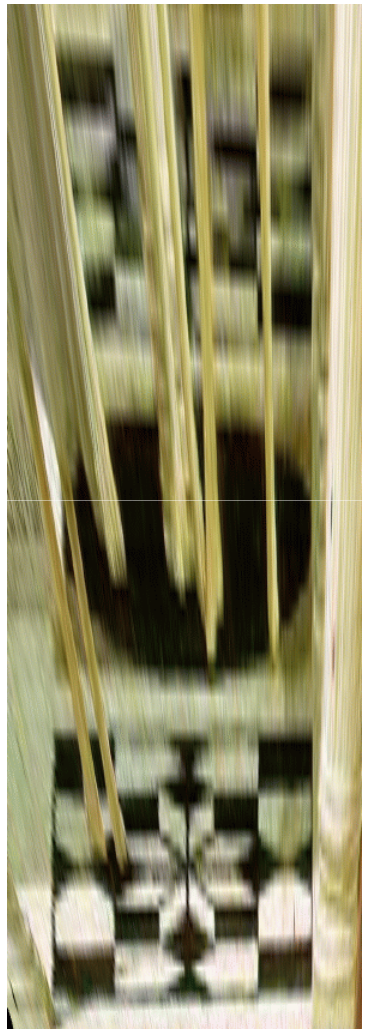
What is the shape of the b/w floor pattern?



Homography



The floor (enlarged)

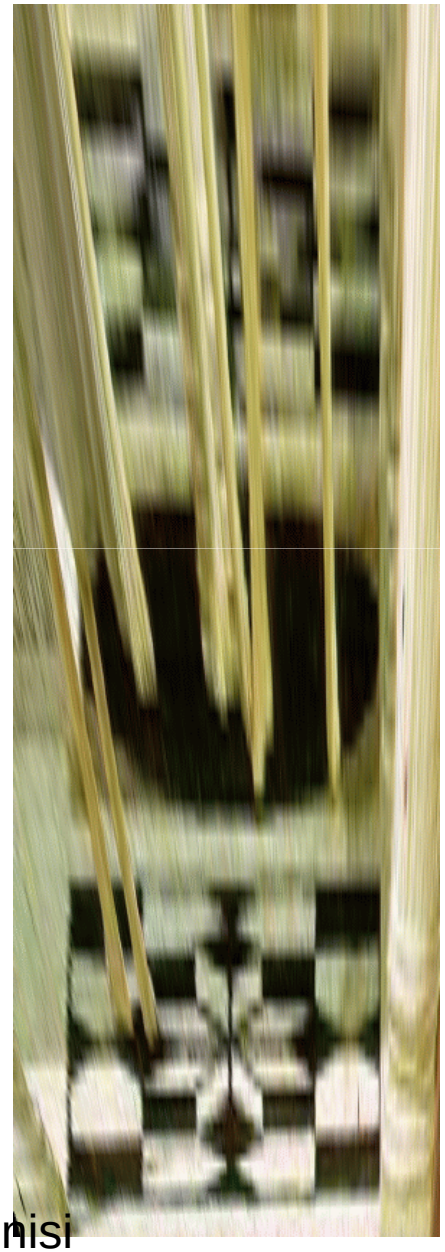


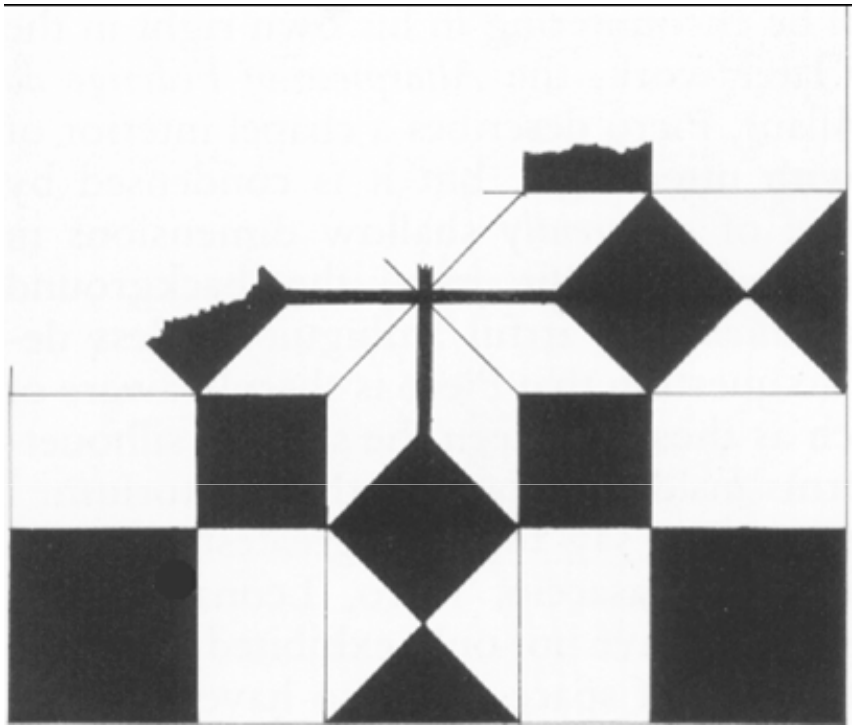
Automatically rectified floor

Slide from Criminisi

Analysing patterns and shapes

Automatic rectification



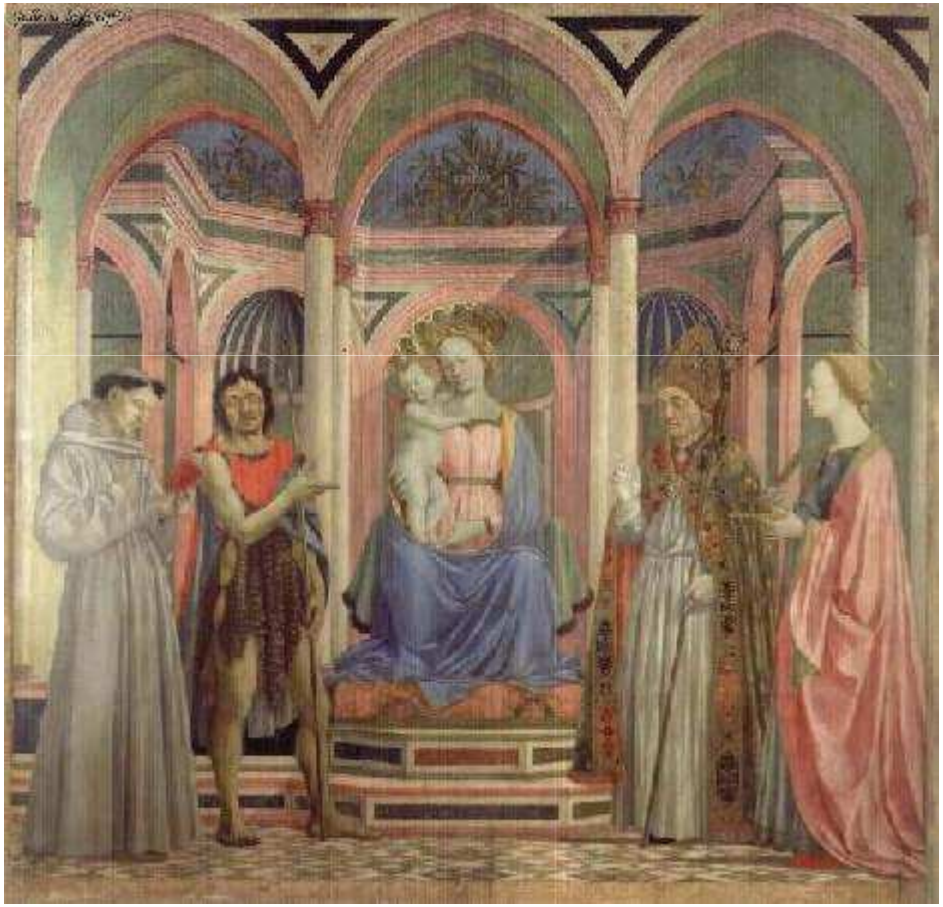


From Martin Kemp *The Science of Art*
(manual reconstruction)

2 patterns have been discovered !

Slide from Criminisi

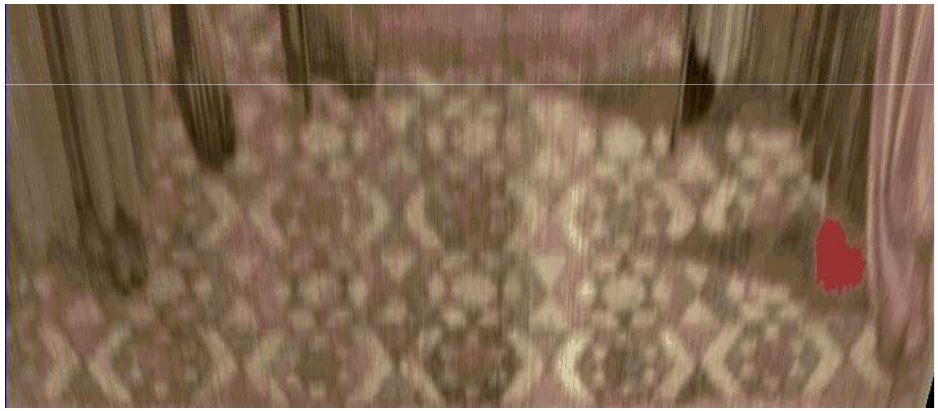
Analysing patterns and shapes



***St. Lucy Altarpiece,* D. Veneziano**

Slide from Criminisi

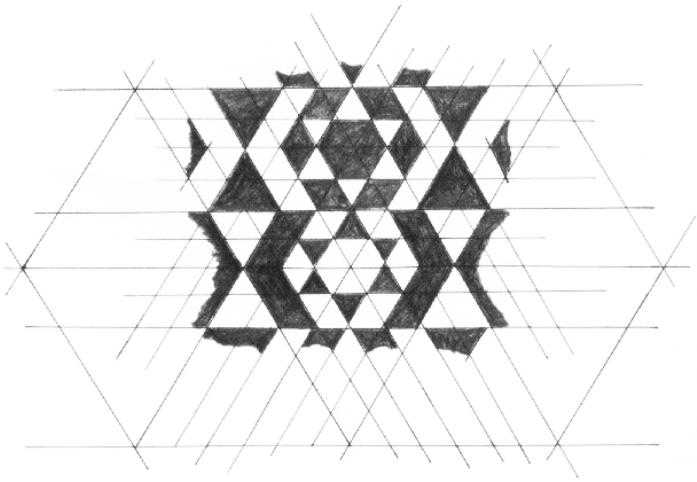
What is the (complicated) shape of the floor pattern?



Automatically rectified floor

Analysing patterns and shapes

Automatic rectification



From Martin Kemp, *The Science of Art* *(manual reconstruction)*

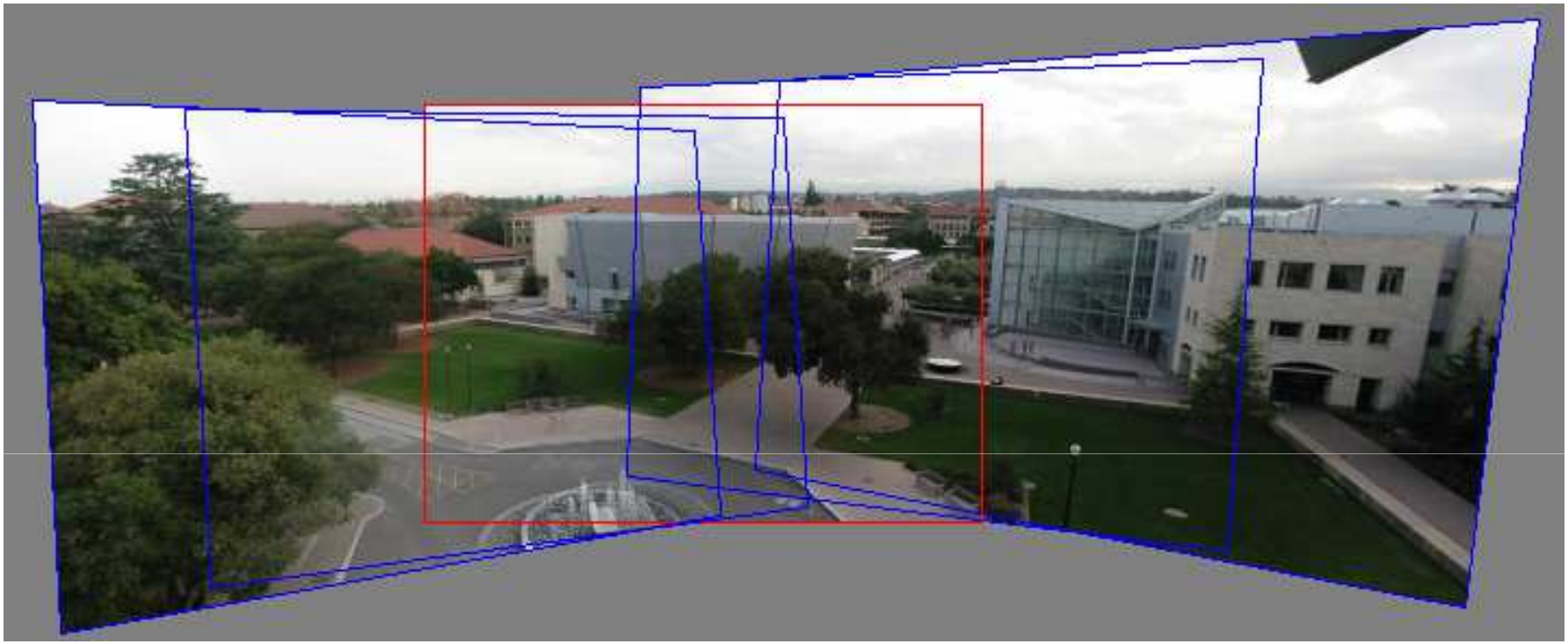
Slide from Criminisi

Julian Beever: Manual Homographies

http://users.skynet.be/J.Beever/pave.htm

Holbein, *The Ambassadors*

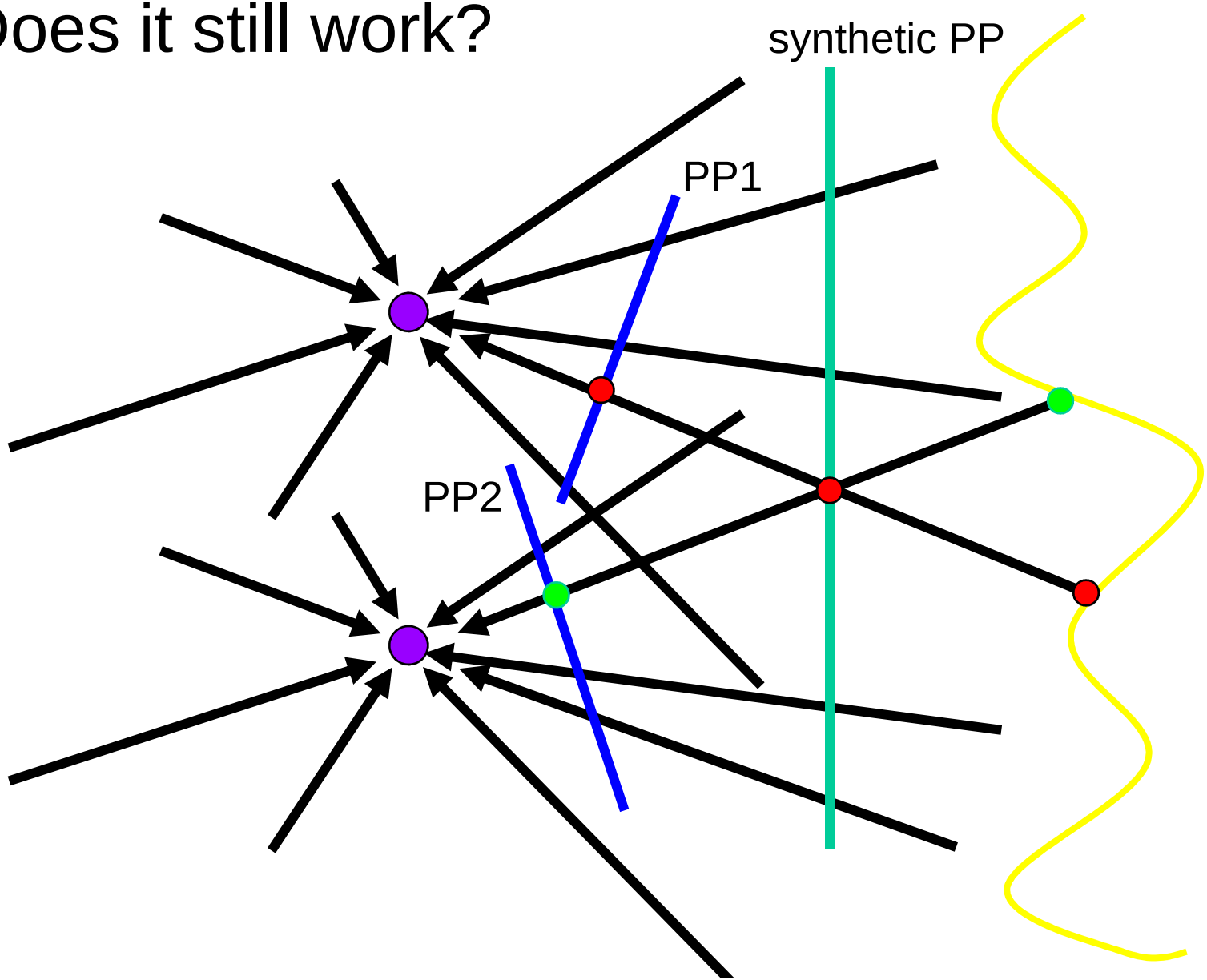
Panoramas



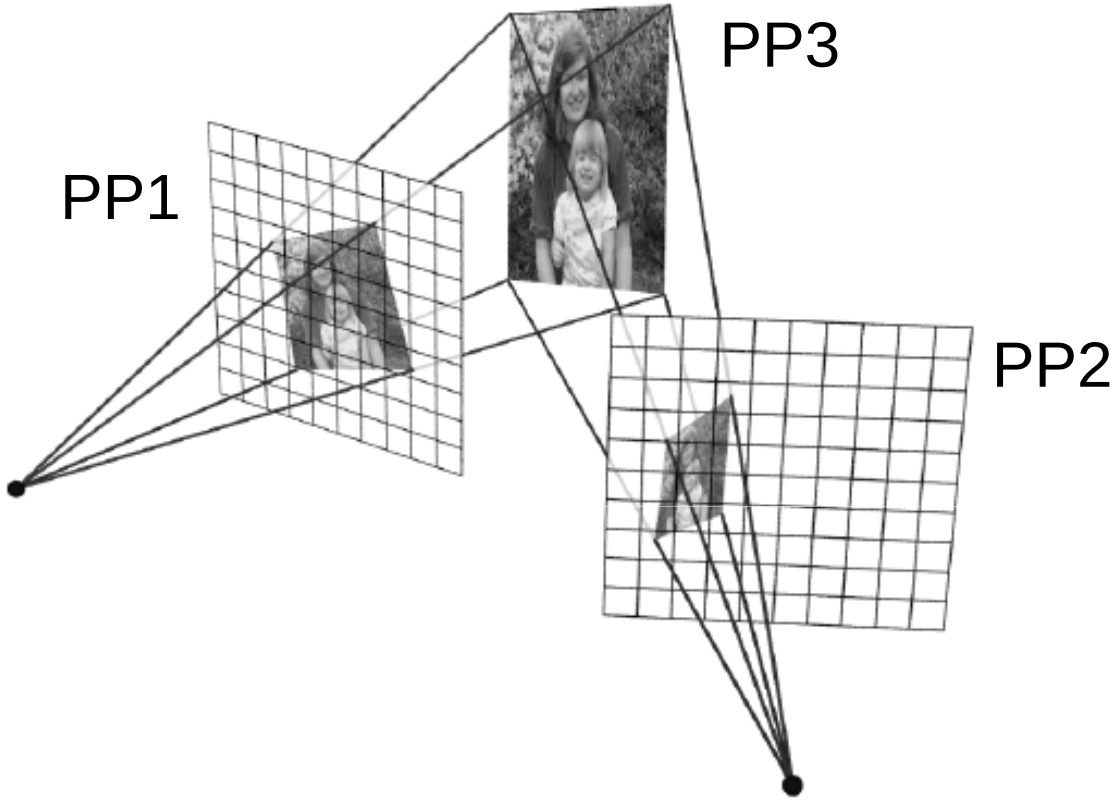
1. Pick one image (red)
2. Warp the other images towards it (usually, one by one)
3. Blend

changing camera center

Does it still work?



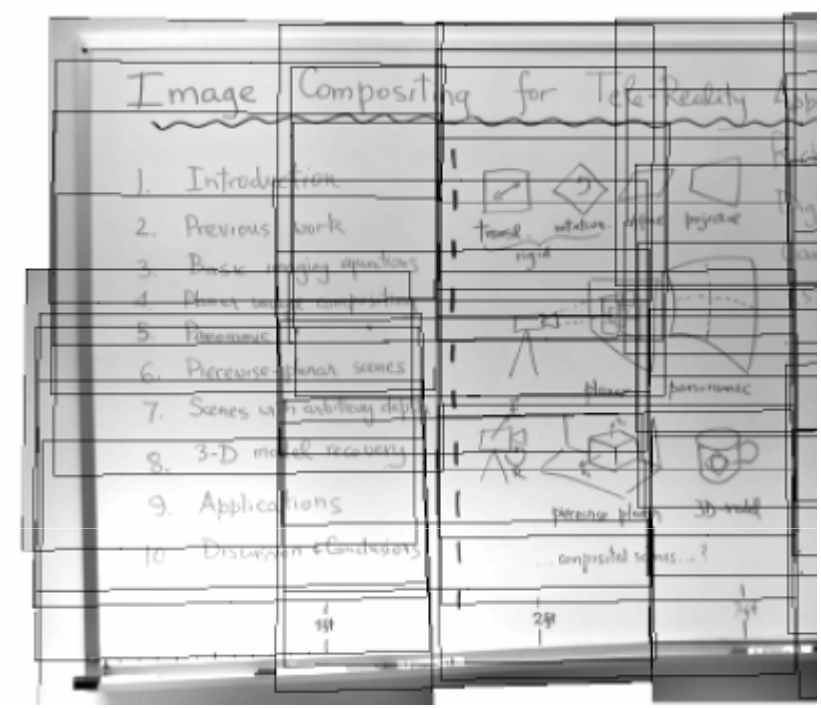
Planar scene (or far away)

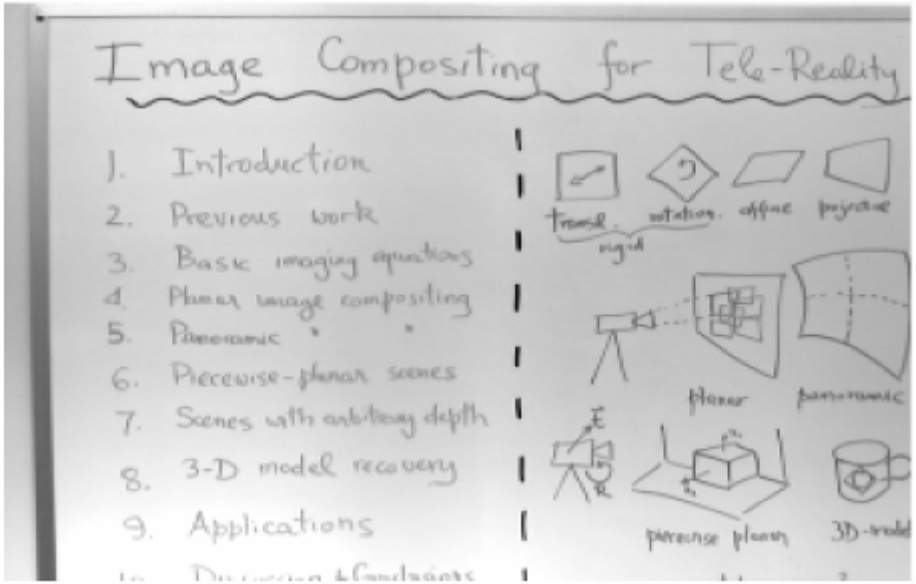


PP3 is a projection plane of both centers of projection, so we are OK!

This is how big aerial photographs are made

Planar mosaic





Some examples at CMU

Ben Hollis, 2004

Ben Hollis, 2004

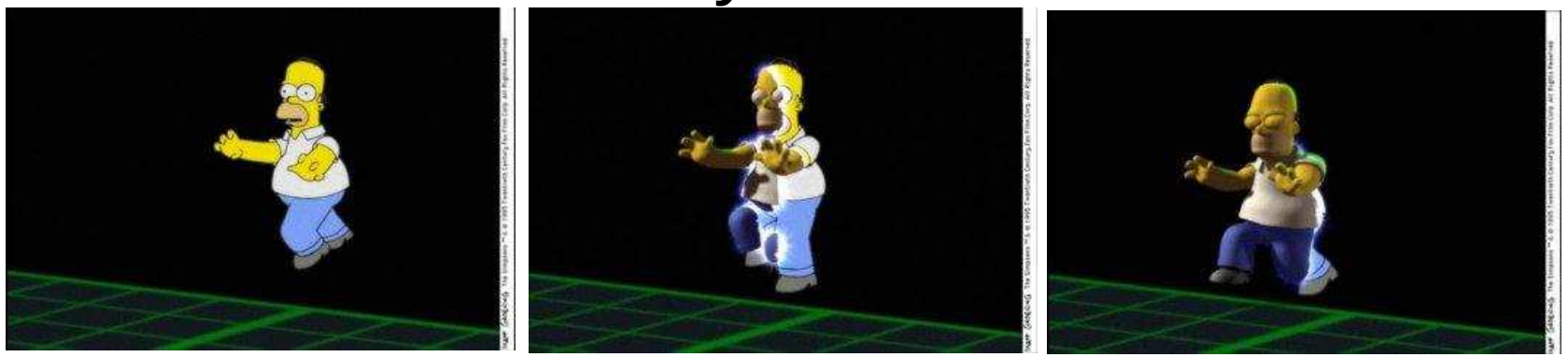
Eunjeong Ryu (E.J), 2004

Matt Pucevich , 2004

Go Explore!

Breaking out of 2D

…now we are ready to break out of 2D



And enter the real world!

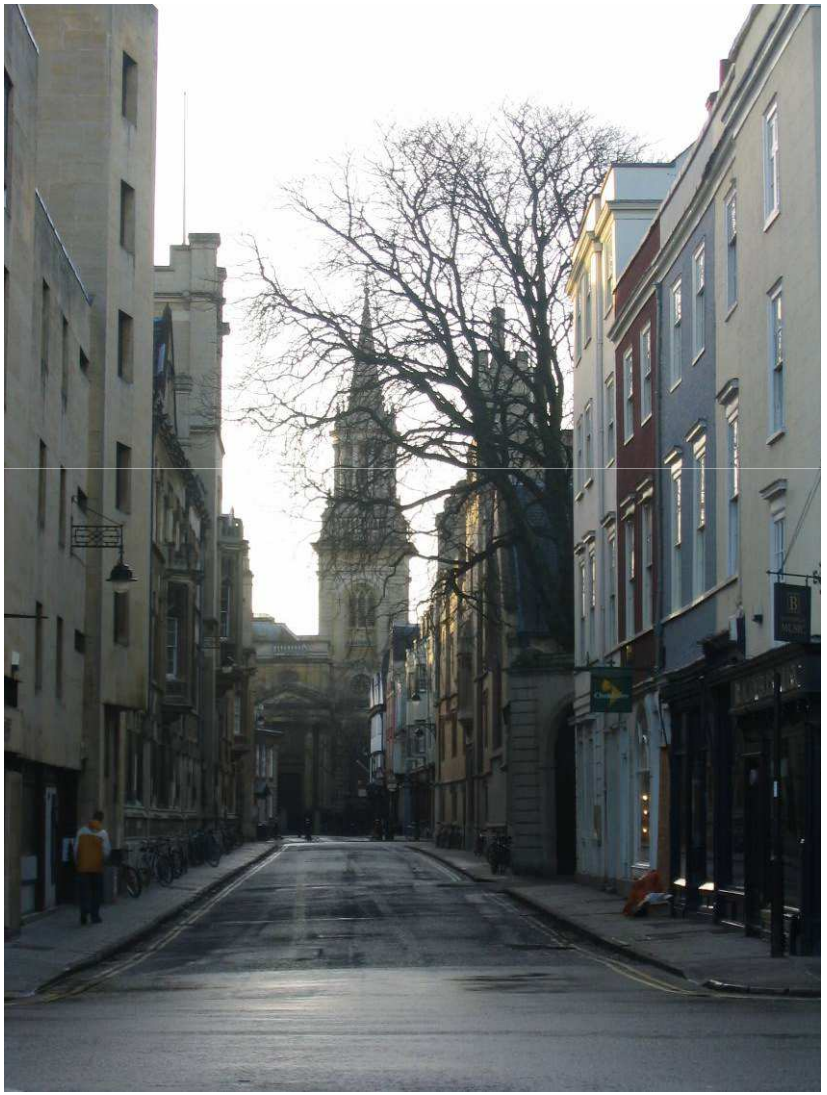
on to 3D…

Enough of images!

We want real 3D scene walk-throughs:

- Camera rotation
- Camera translation

Can we do it from a single photograph?



Camera rotations with homographies

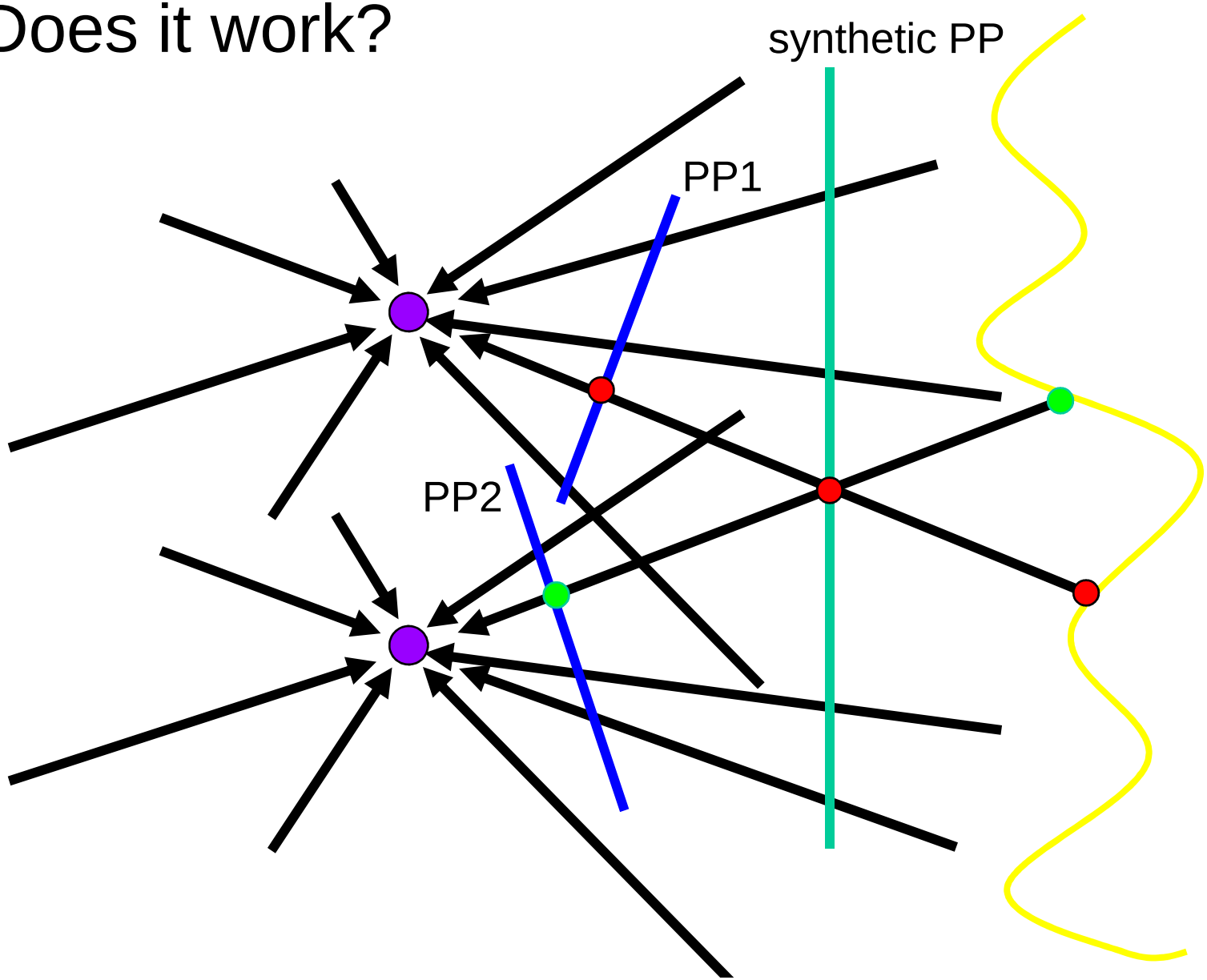
Original image

St.Petersburg
photo by A. Tikhonov

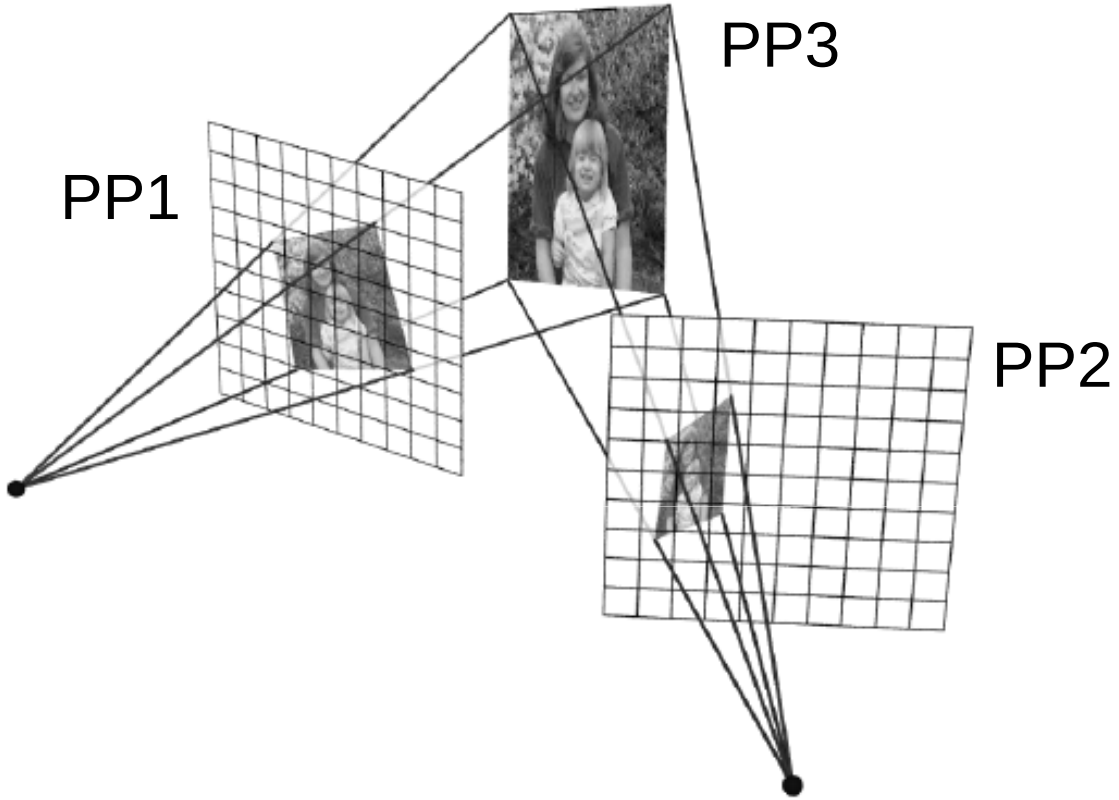
Virtual camera rotations

Camera translation

Does it work?



Yes, with planar scene (or far away)

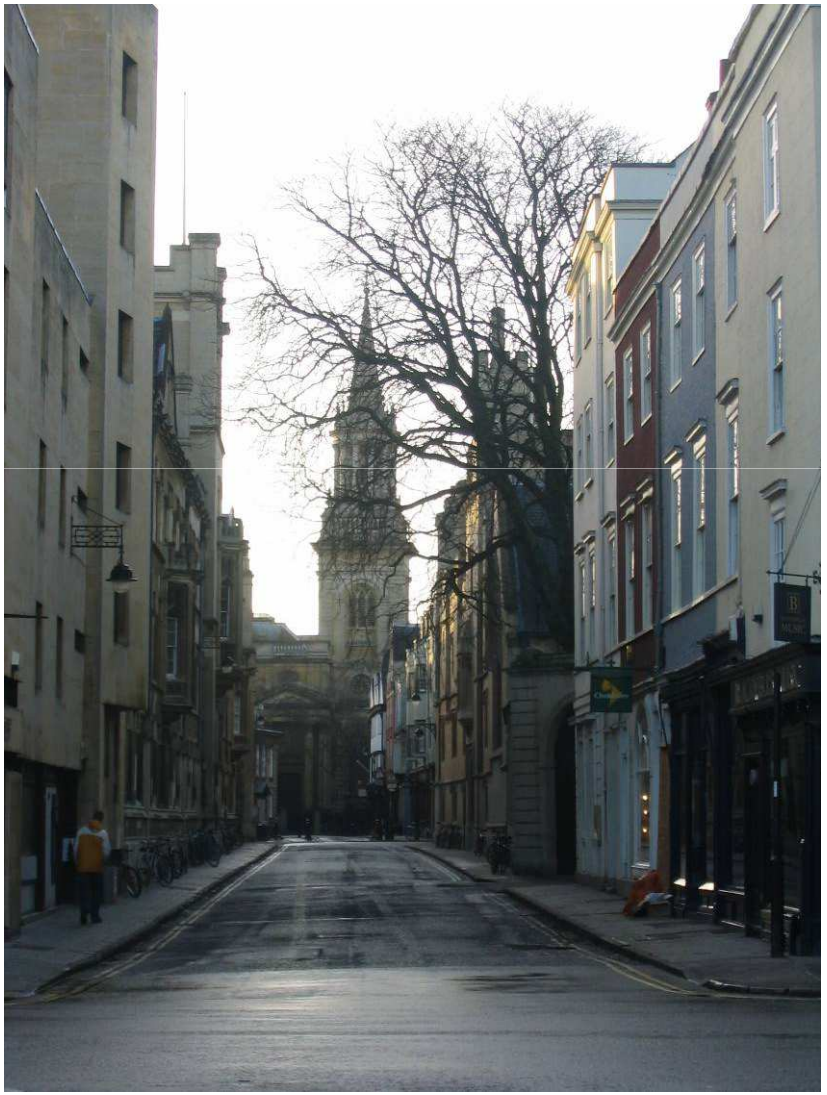


PP3 is a projection plane of both centers of projection, so we are OK!

So, what can we do here?

Model the scene as a set of planes!

Now, just need to find the orientations of these planes.



Some preliminaries: projective geometry

Ames Room

Silly Euclid: Trix are for kids!

Parallel lines???

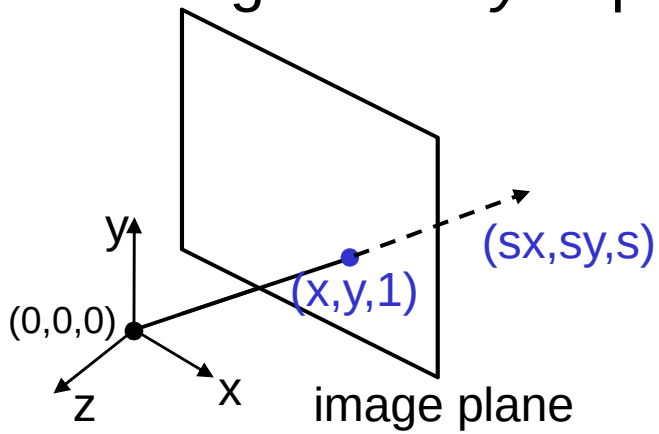
The projective plane

Why do we need homogeneous coordinates?

- represent points at infinity, homographies, perspective projection, multi-view relationships

What is the geometric intuition?

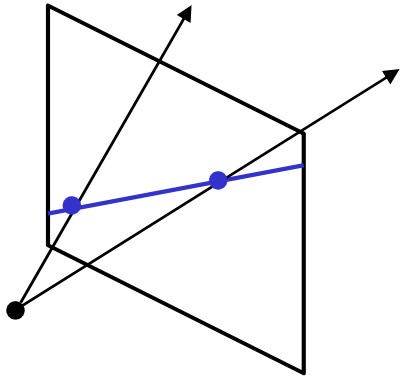
- a point in the image is a *ray* in projective space



- Each *point* (x,y) on the plane is represented by a *ray* (sx,sy,s)
  - all points on the ray are equivalent: (x, y, 1) ≅ (sx, sy, s)

Projective lines

What does a line in the image correspond to in projective space?



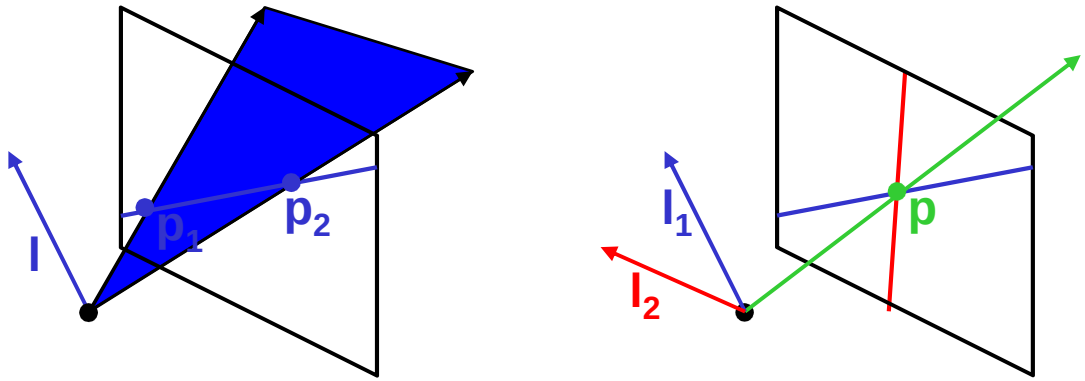
- A line is a *plane* of rays through origin
  - all rays (x,y,z) satisfying: ax + by + cz = 0

$$0 = \underset{l}{\begin{pmatrix} a & b & c \end{pmatrix}} \underset{p}{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}$$

- A line is also represented as a homogeneous 3-vector **l**

Point and line duality

- A line is **l** is an homogeneous 3-vector
- It is ⊥ to every point (ray) on the line **lp = 0**



What is the line l spanned by the rays p1 and p2?

- **l** is ⊥ to **p1** and **p2** ⇒ **l = p1 x p2**
- **l** is the plane normal vector

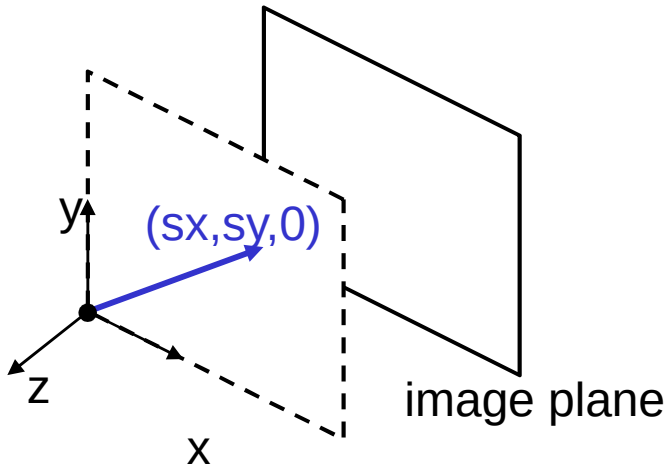
What is the intersection of two lines l1 and l2?

- **p** is ⊥ to **p1** and **p2** ⇒ **l = p1 x p2**

Points and lines are dual in the projective space

- Given any formula, we can switch the meaning of points and lines to get another one.

Ideal points and lines



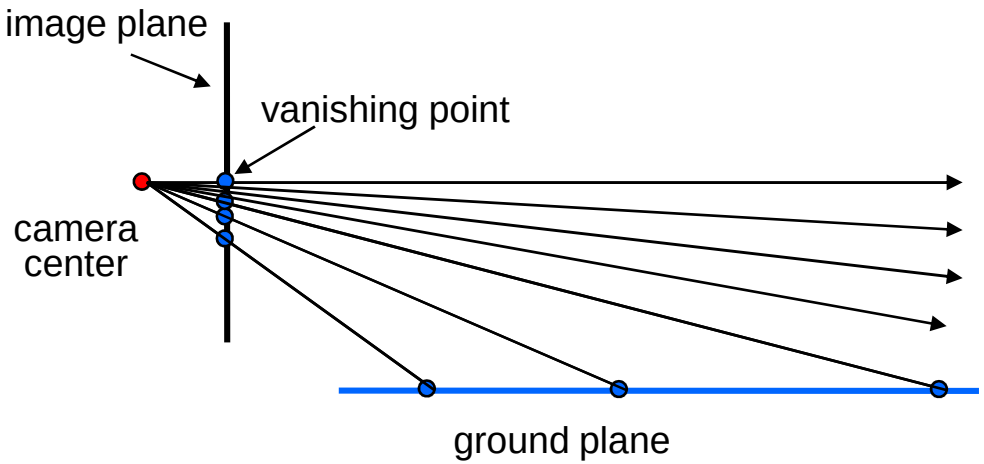
Ideal point (“point at infinity”)

- $p \cong (x, y, 0)$ – parallel to image plane
- It has infinite image coordinates

Ideal line

$l \cong (0, 0, 1)$ – parallel to image plane

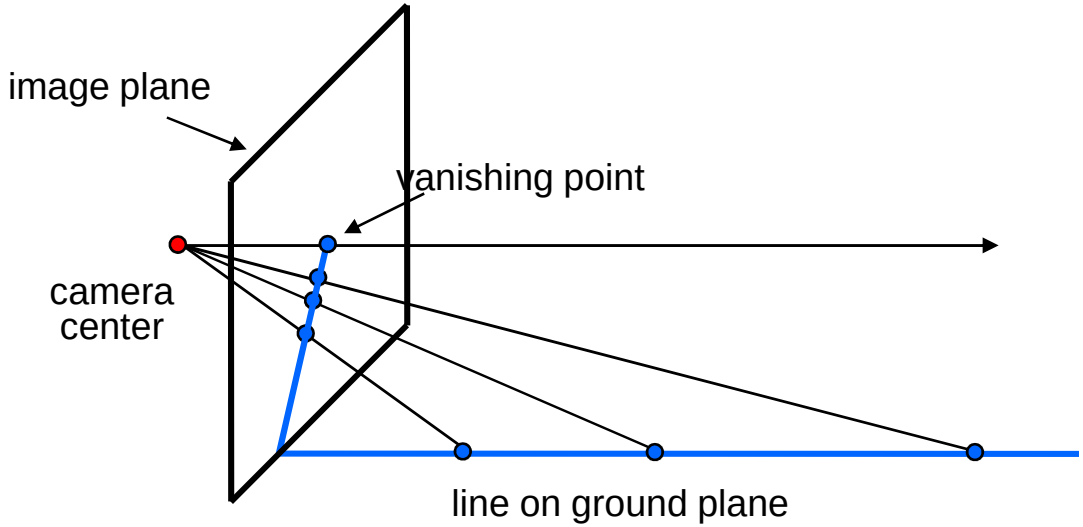
Vanishing points



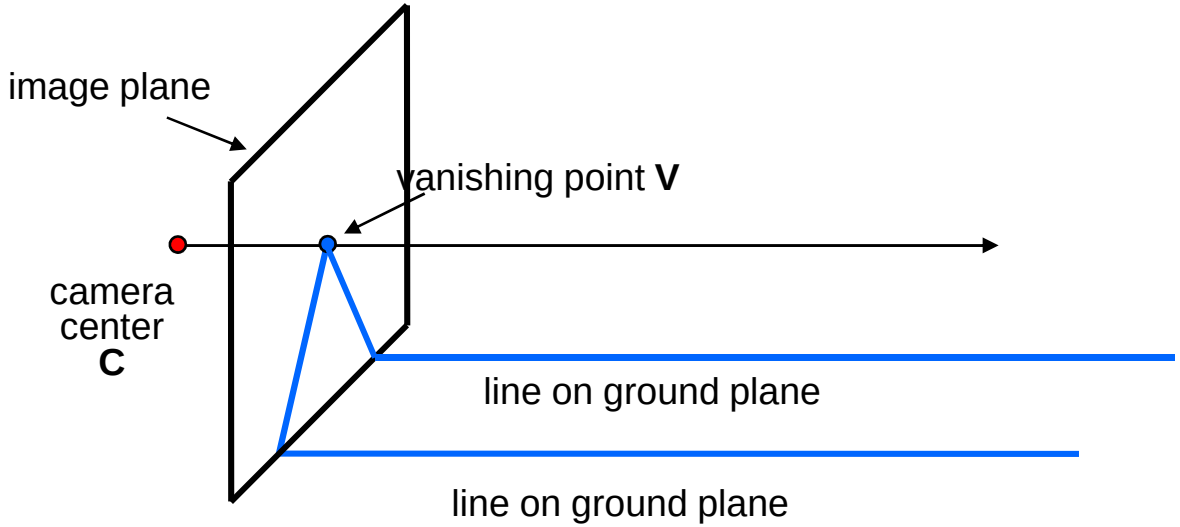
Vanishing point

- projection of a point at infinity
- Caused by ideal line

Vanishing points (2D)



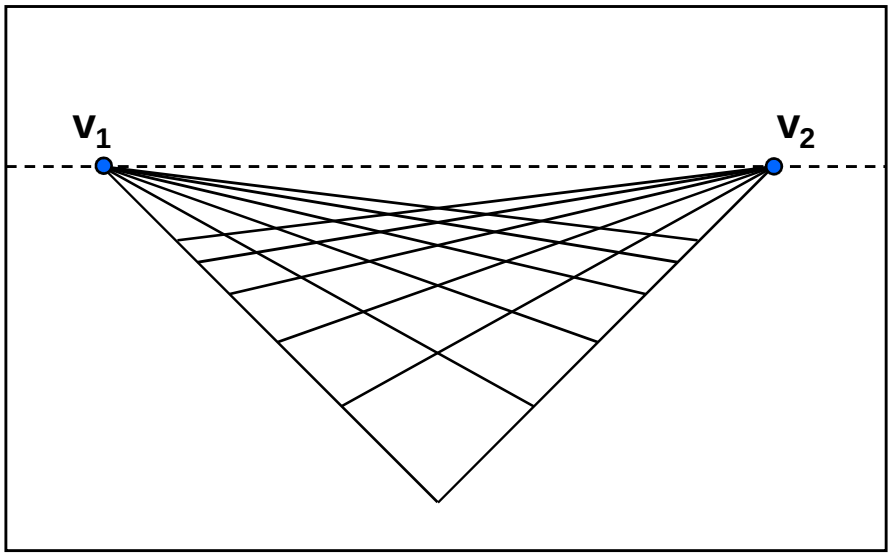
Vanishing points



Properties

- Any two parallel lines have the same vanishing point **v**
- The ray from **C** through **v** is parallel to the lines
- An image may have more than one vanishing point

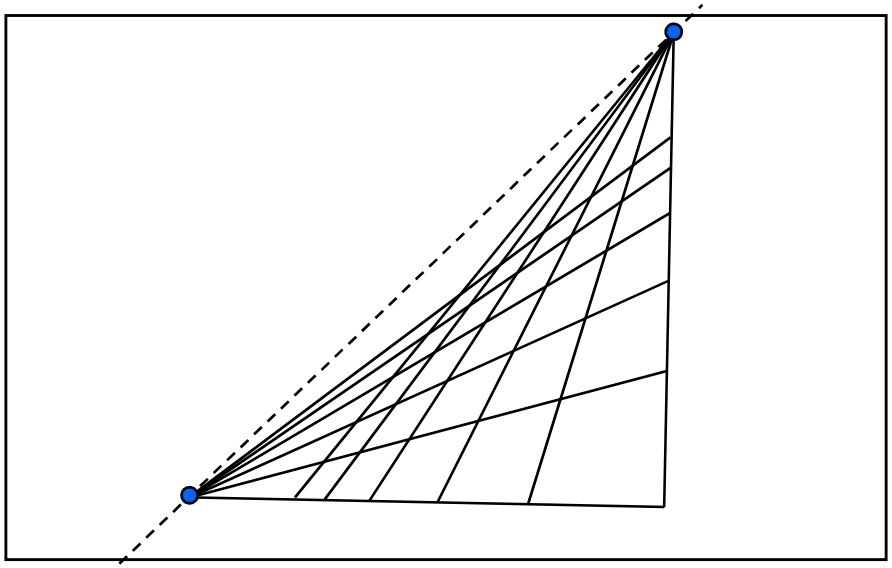
Vanishing lines



Multiple Vanishing Points

- Any set of parallel lines on the plane define a vanishing point
- The union of all of these vanishing points is the *horizon line*
  - also called *vanishing line*
- Note that different planes define different vanishing lines

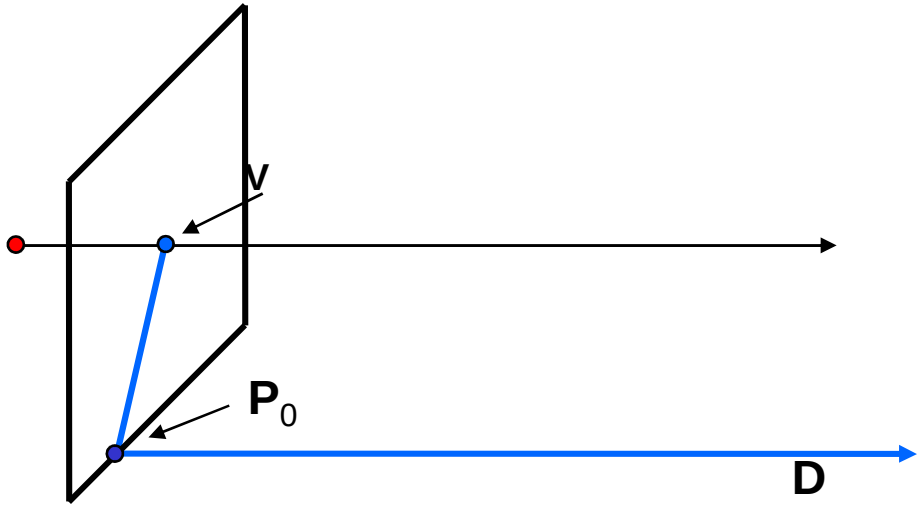
Vanishing lines



Multiple Vanishing Points

- Any set of parallel lines on the plane define a vanishing point
- The union of all of these vanishing points is the *horizon line*
  - also called *vanishing line*
- Note that different planes define different vanishing lines

Computing vanishing points



$$\mathbf{P} = \mathbf{P}_0 + t\mathbf{D}$$

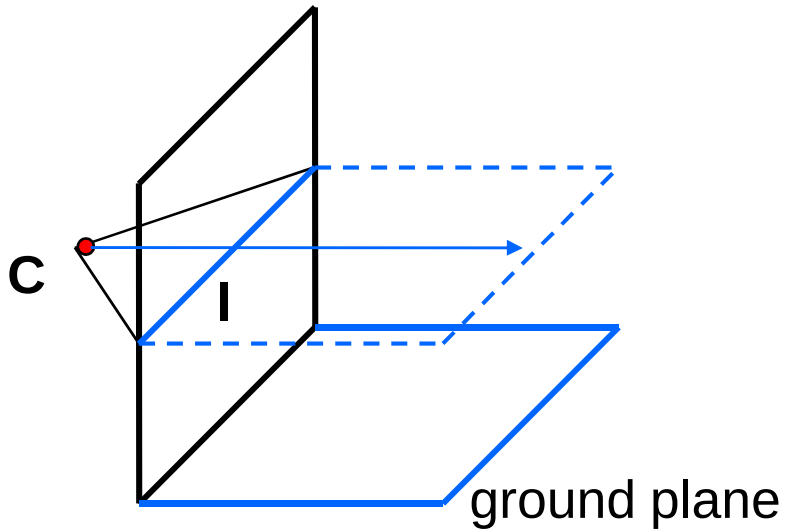
$$\mathbf{P}_t = \begin{bmatrix} P_X + tD_X \\ P_Y + tD_Y \\ P_Z + tD_Z \\ 1 \end{bmatrix} \cong \begin{bmatrix} P_X/t + D_X \\ P_Y/t + D_Y \\ P_Z/t + D_Z \\ 1/t \end{bmatrix} \qquad t \to \infty \qquad \mathbf{P}_\infty \cong \begin{bmatrix} D_X \\ D_Y \\ D_Z \\ 0 \end{bmatrix}$$

$$\mathbf{v} = \Pi\mathbf{P}_\infty$$

Properties

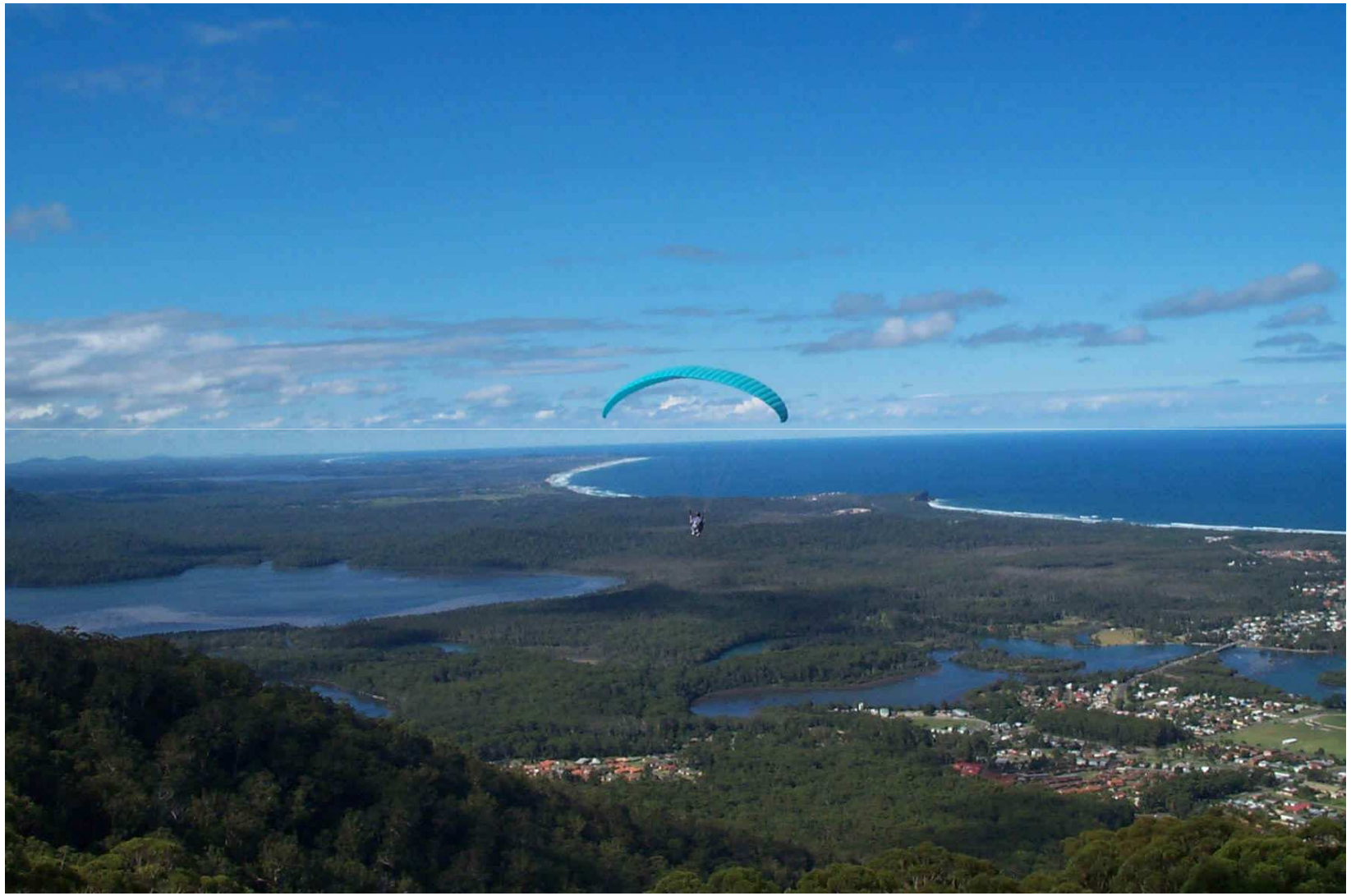
- $\mathbf{P}_\infty$ is a point at *infinity*, **v** is its projection
- They depend only on line *direction*
- Parallel lines $\mathbf{P}_0 + t\mathbf{D}$, $\mathbf{P}_1 + t\mathbf{D}$ intersect at $\mathbf{P}_\infty$

Computing vanishing lines

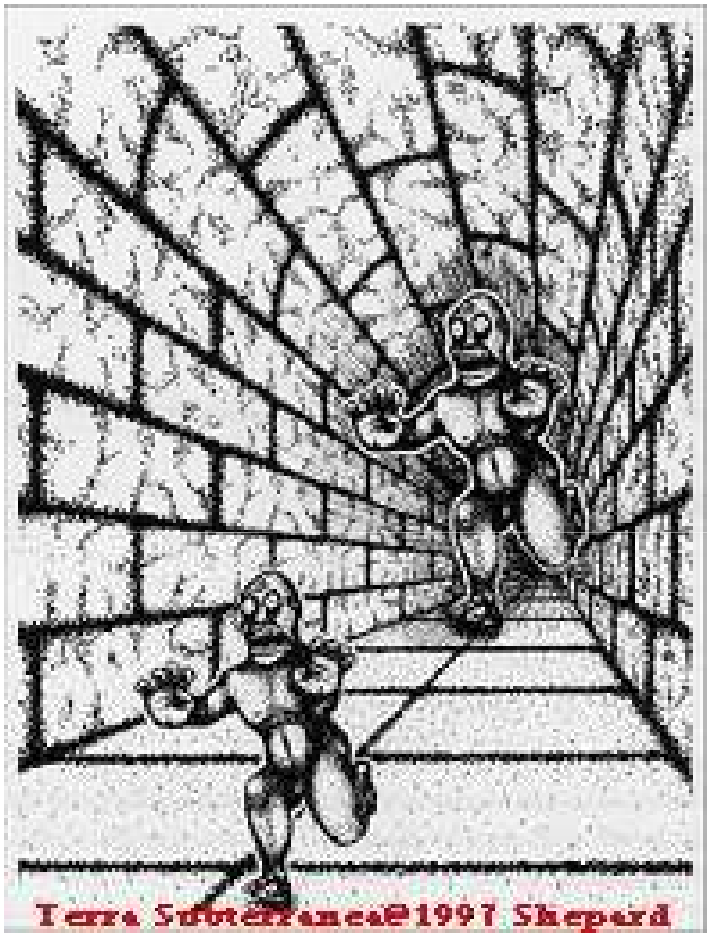


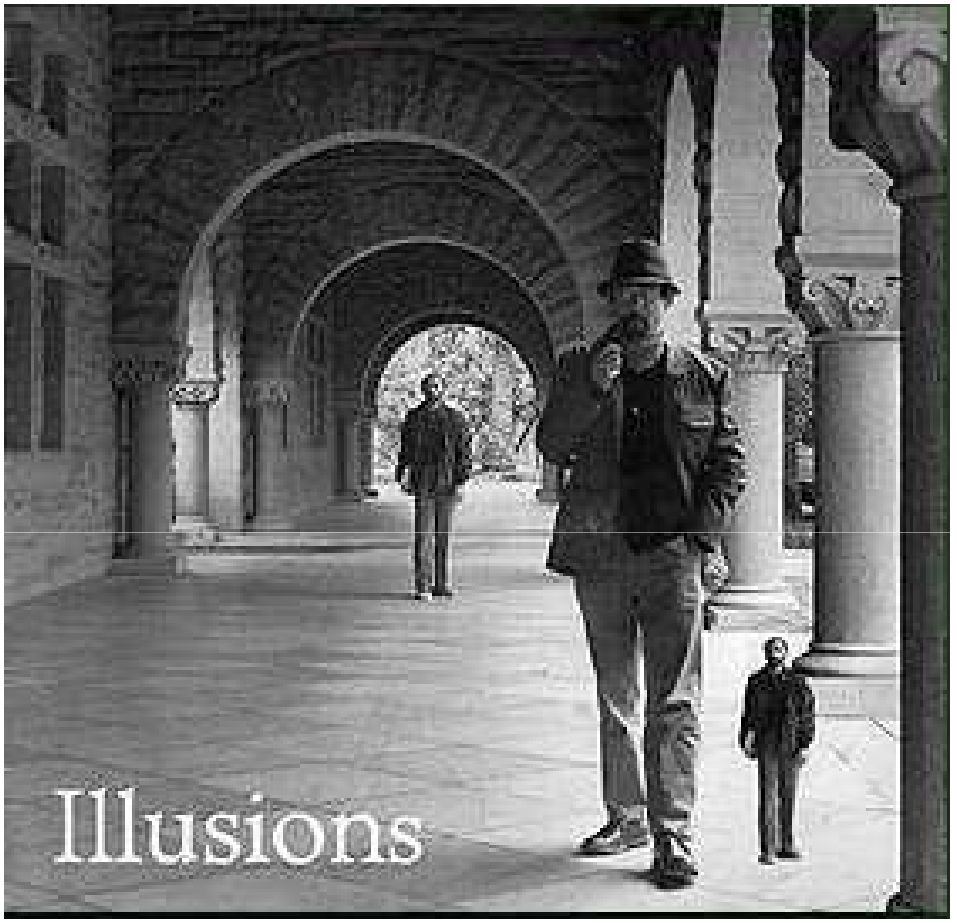
Properties

- **l** is intersection of horizontal plane through **C** with image plane
- Compute **l** from two sets of parallel lines on ground plane
- All points at same height as **C** project to **l**
  - points higher than C project above l
- Provides way of comparing height of objects in the scene



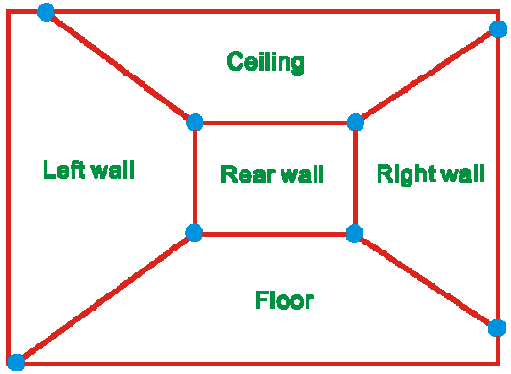
Fun with vanishing points





“Tour into the Picture” (SIGGRAPH ’97)

Create a 3D “theatre stage” of five billboards



Specify foreground objects through bounding polygons

Use camera transformations to navigate through the scene

The idea

Many scenes (especially paintings), can be represented as an axis-aligned box volume (i.e. a stage)

Key assumptions:

- All walls of volume are orthogonal
- Camera view plane is parallel to back of volume
- Camera up is normal to volume bottom

How many vanishing points does the box have?

- Three, but two at infinity
- Single-point perspective

Can use the vanishing point

to fit the box to the particular

Scene!

The idea

Many scenes (especially paintings), can be represented as an axis-aligned box volume (i.e. a stage)

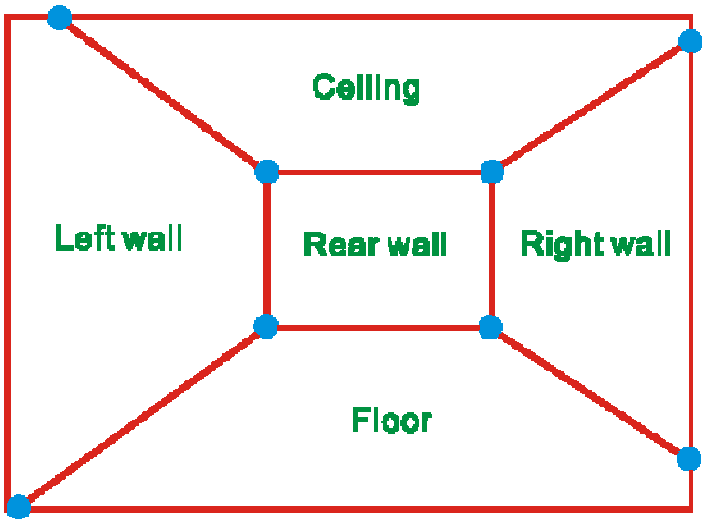
Key assumptions:

- All walls of volume are orthogonal
- Camera view plane is parallel to back of volume
- Camera up is normal to volume bottom

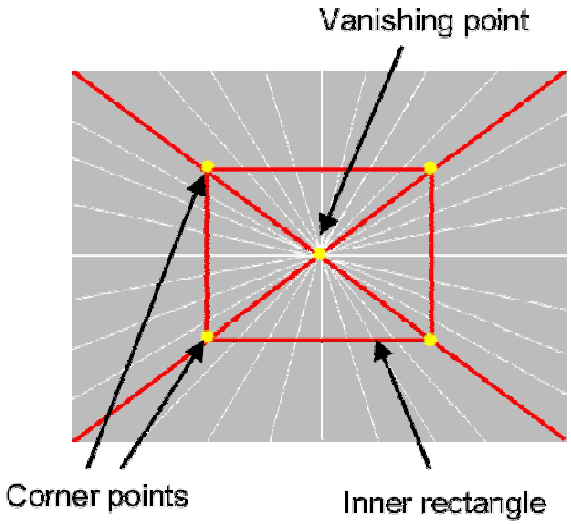
How many vanishing points does the box have?

- Three, but two at infinity
- Single-point perspective

Can use the vanishing point to fit the box to the particular Scene!



Fitting the box volume

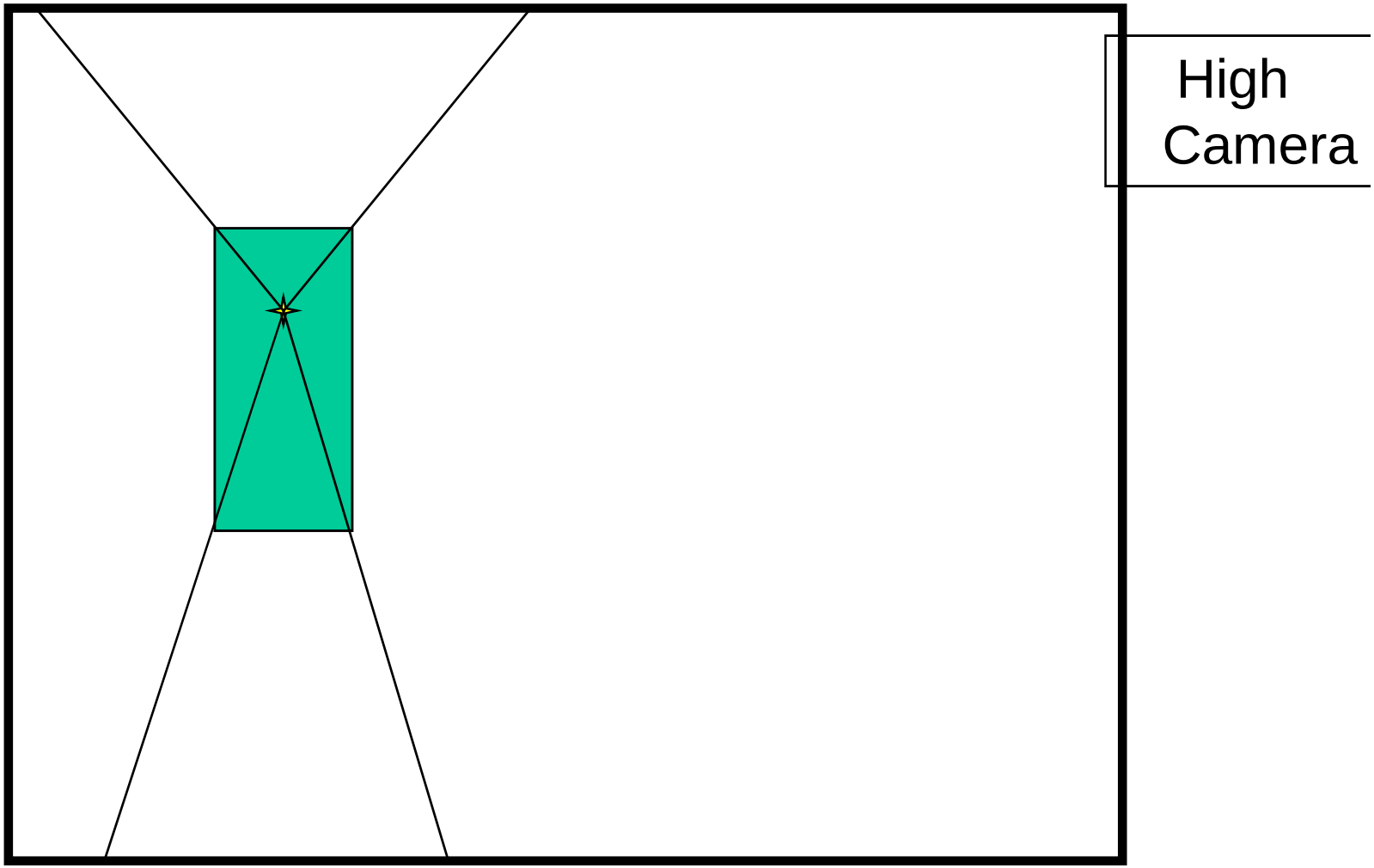


User controls the inner box and the vanishing point placement (# of DOF???)

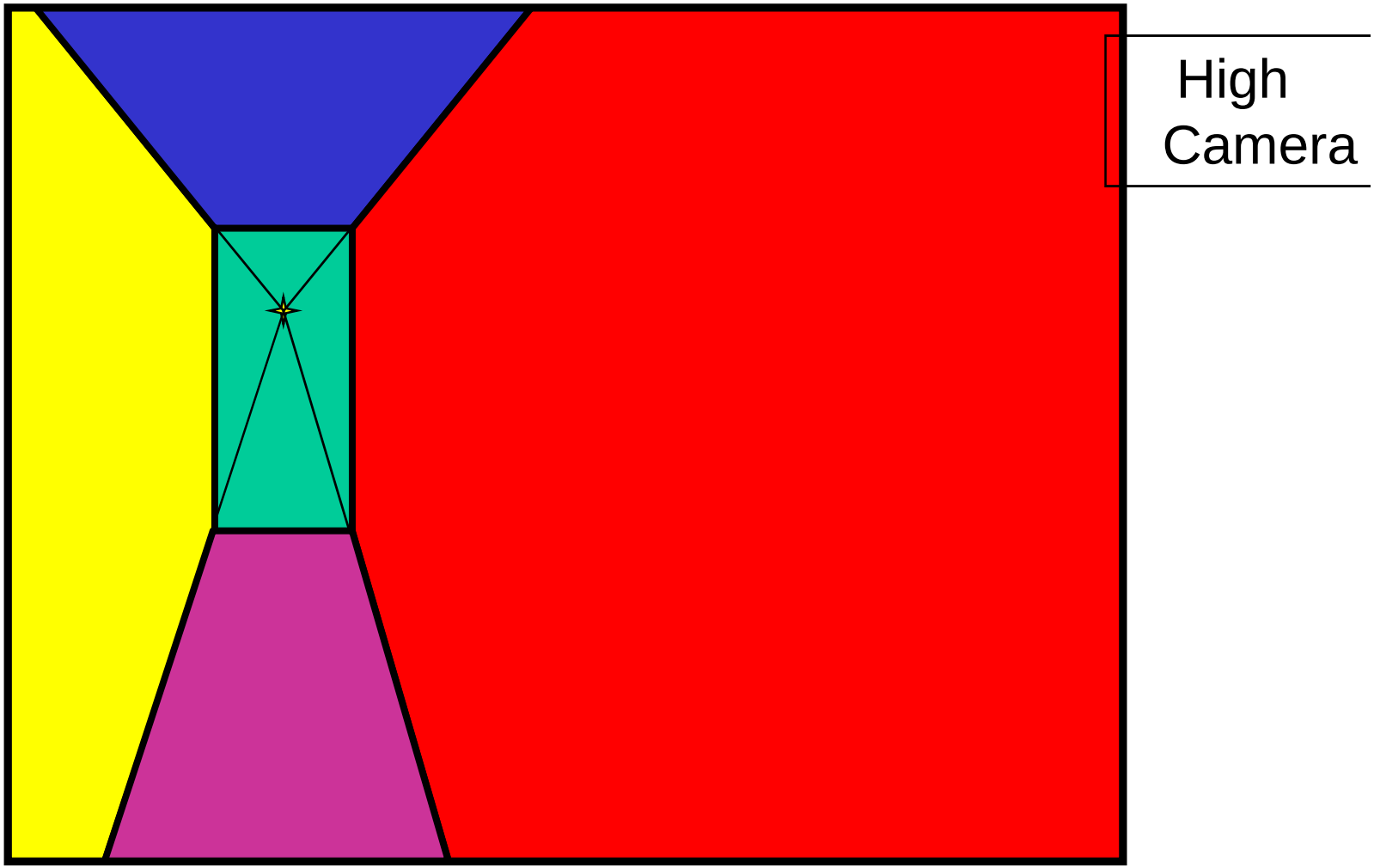
Q: What's the significance of the vanishing point location?

A: It's at eye level: ray from COP to VP is perpendicular to image plane.

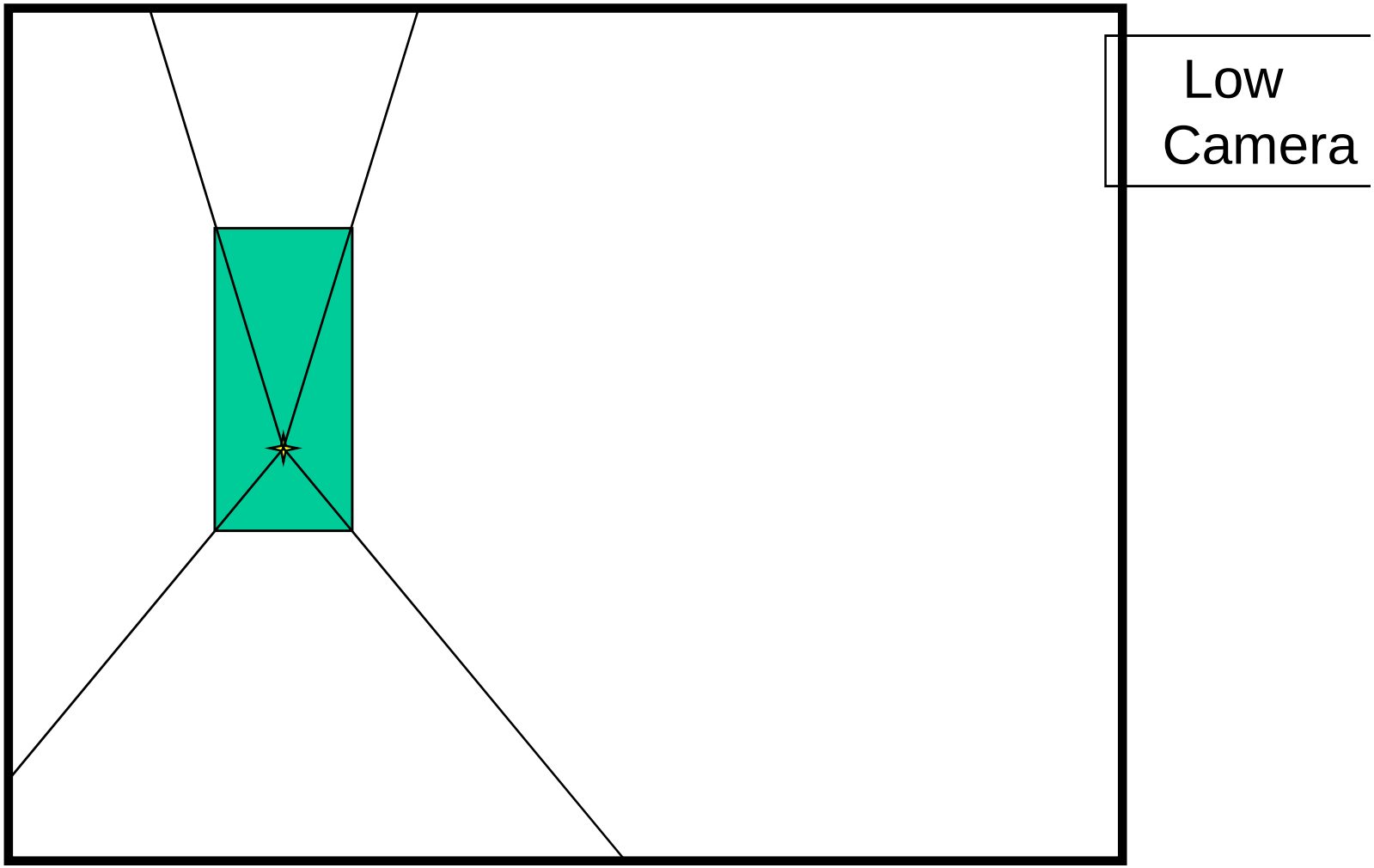
Example of user input: vanishing point and back face of view volume are defined



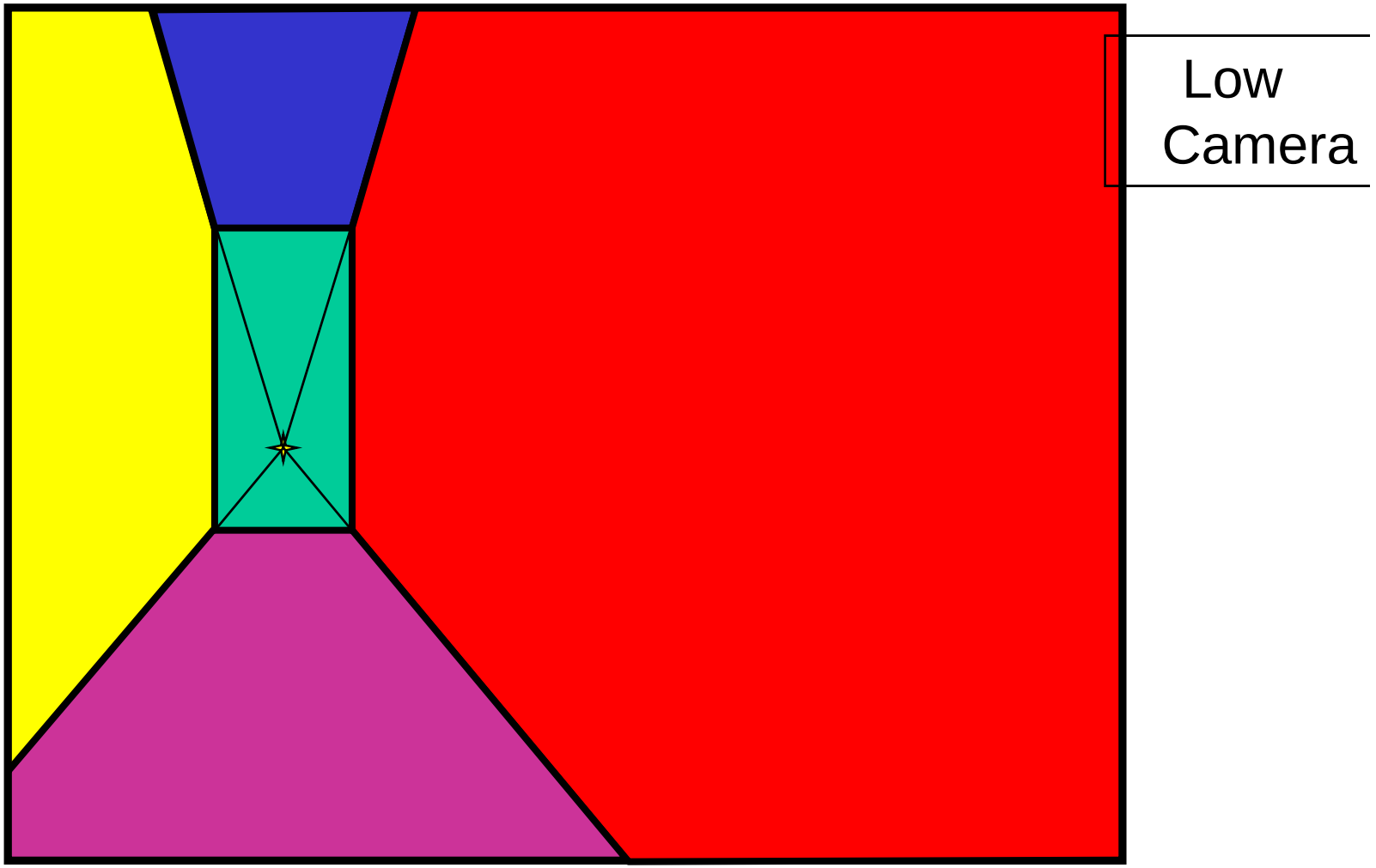
Example of user input: vanishing point and back face of view volume are defined



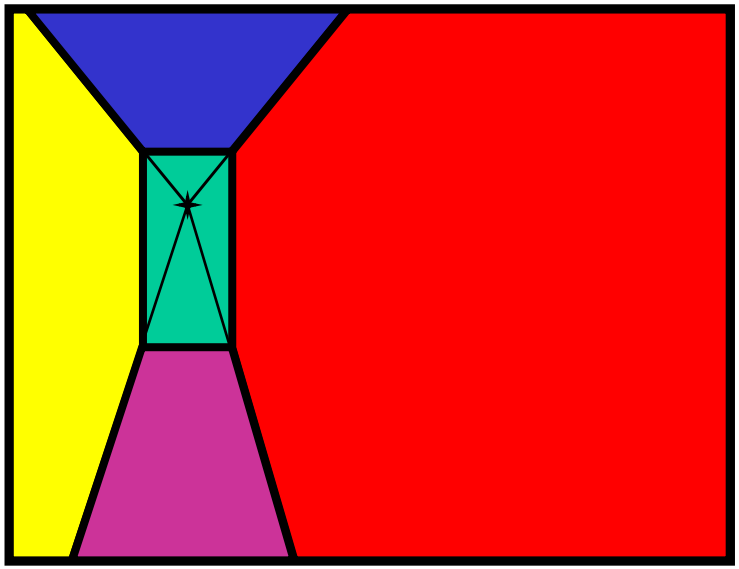
Example of user input: vanishing point and back face of view volume are defined



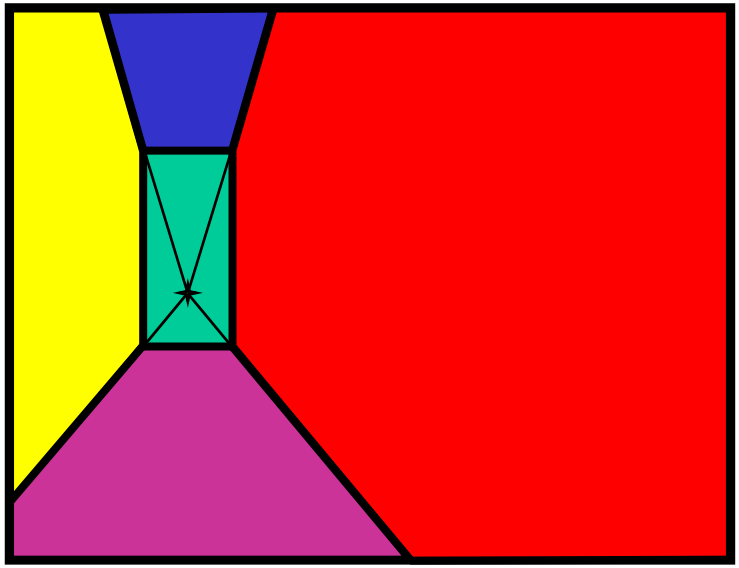
Example of user input: vanishing point and back face of view volume are defined



Comparison of how image is subdivided based on two different camera positions. You should see how moving the vanishing point corresponds to moving the eyepoint in the 3D world.

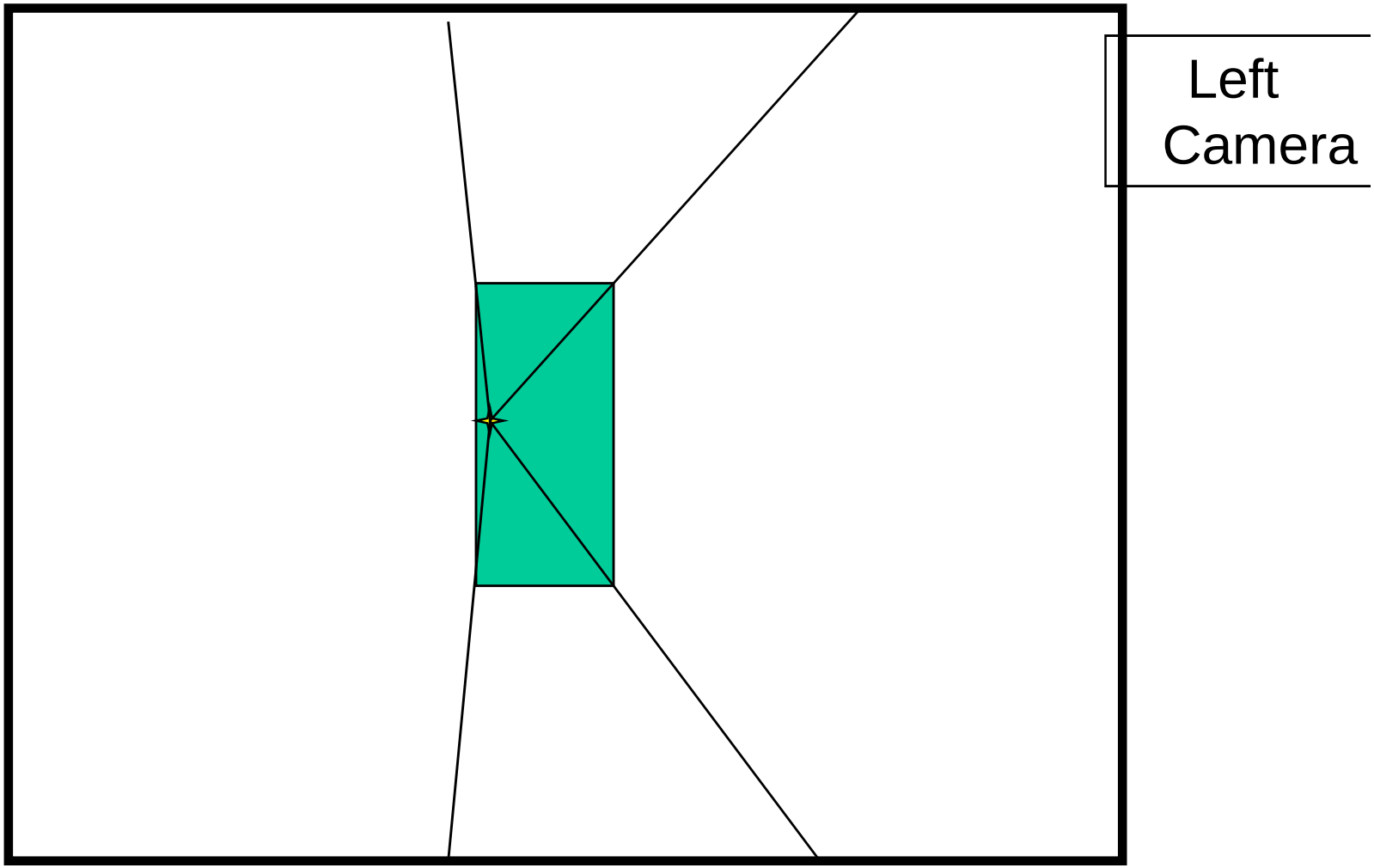


High Camera

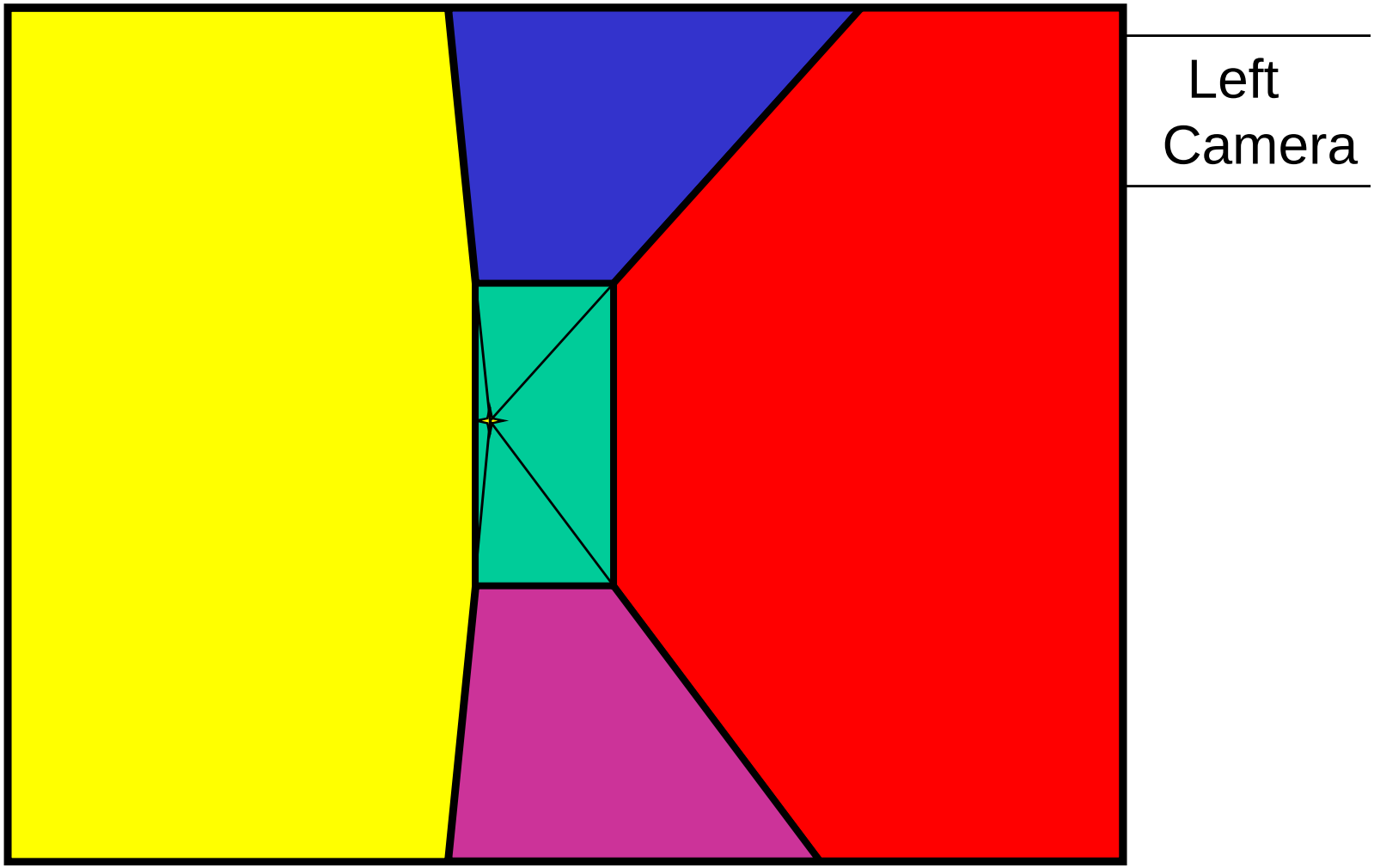


Low Camera

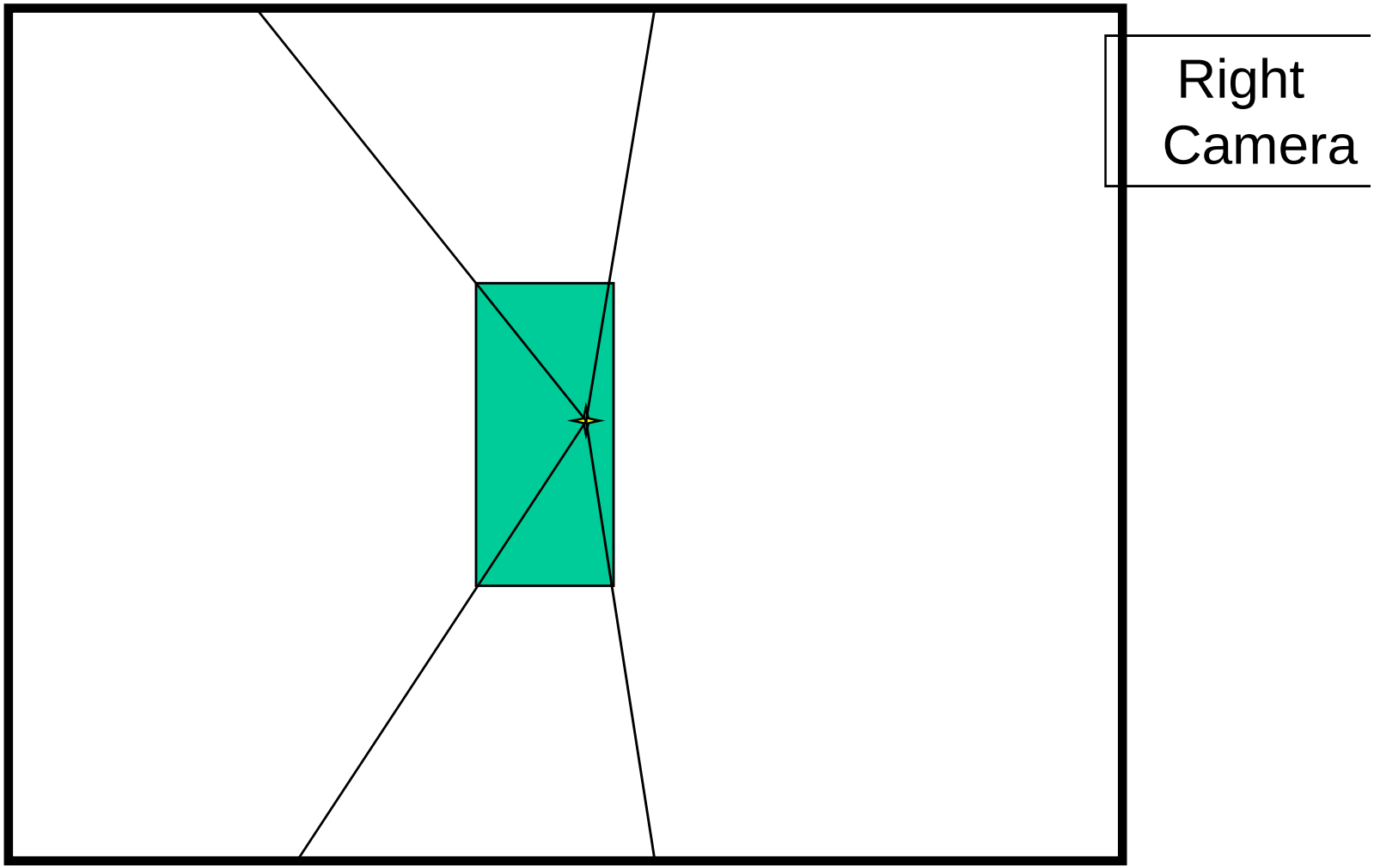
Another example of user input: vanishing point and back face of view volume are defined



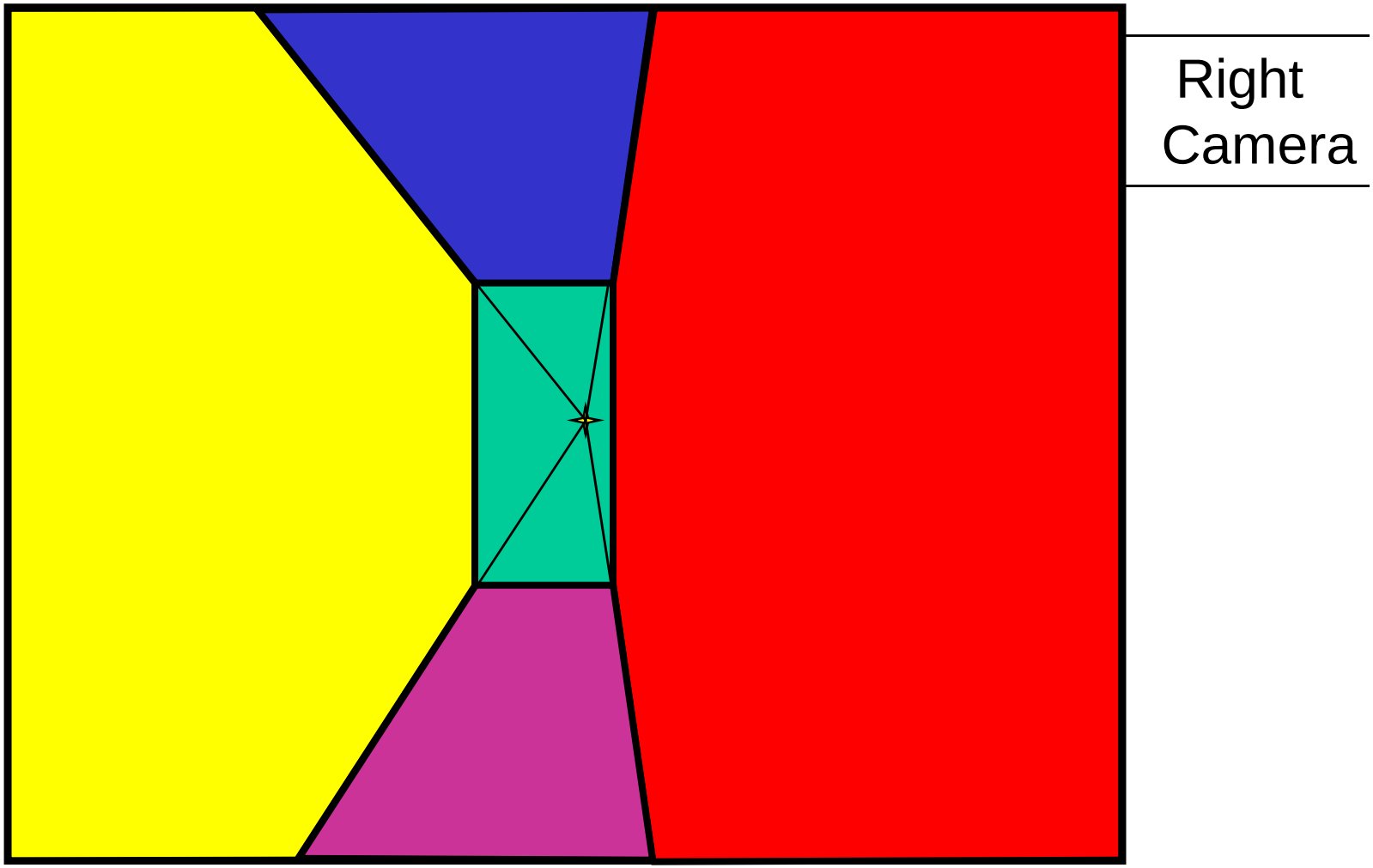
Another example of user input: vanishing point and back face of view volume are defined



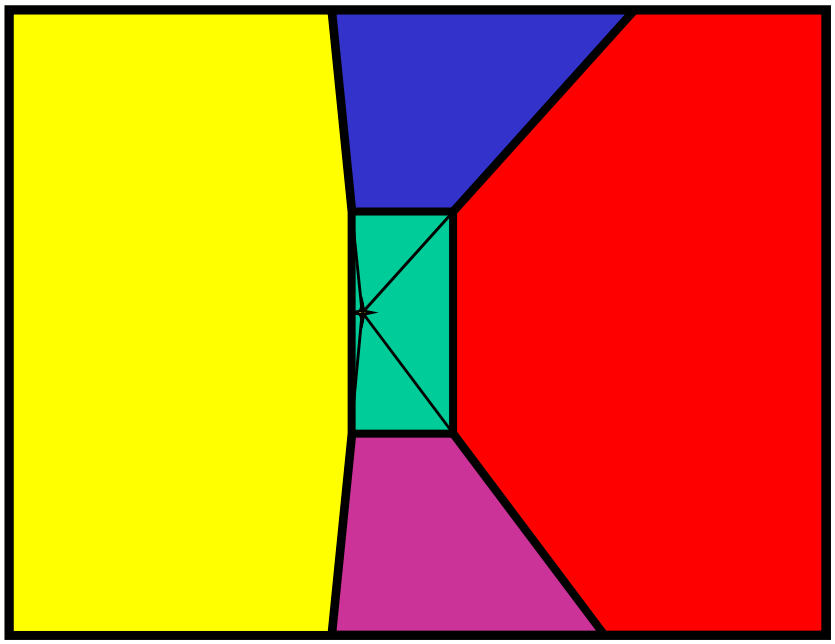
Another example of user input: vanishing point and back face of view volume are defined



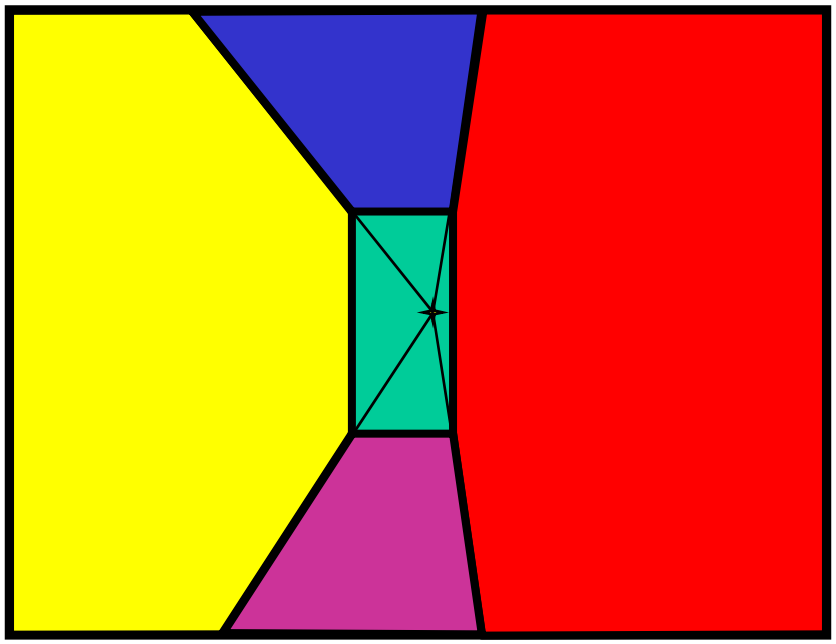
Another example of user input: vanishing point and back face of view volume are defined



Comparison of two camera placements – left and right. Corresponding subdivisions match view you would see if you looked down a hallway.



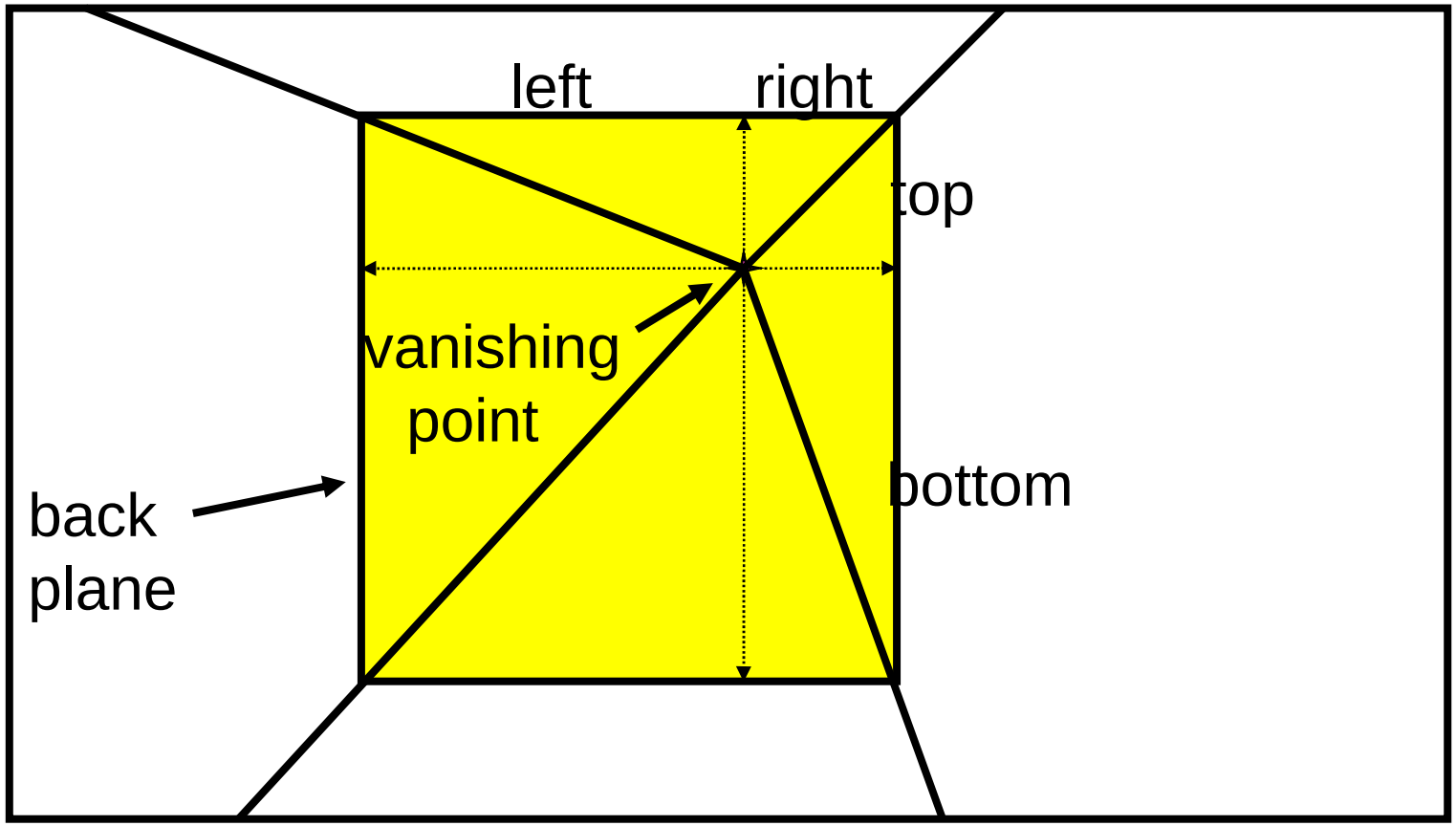
Left Camera



Right Camera

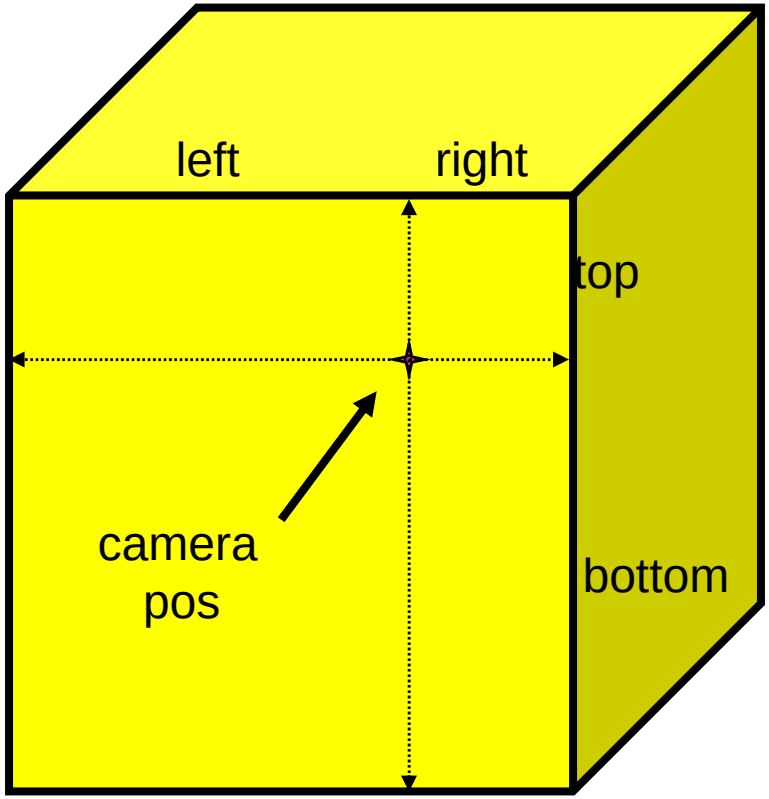
2D to 3D conversion

First, we can get ratios

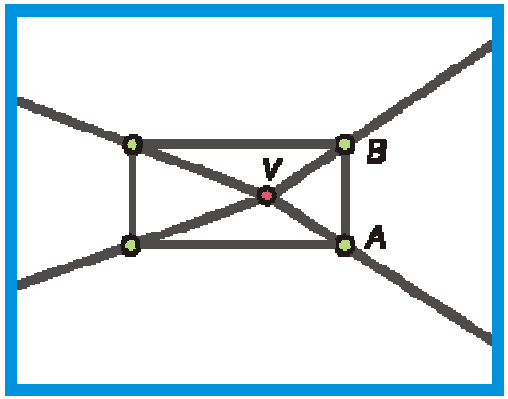


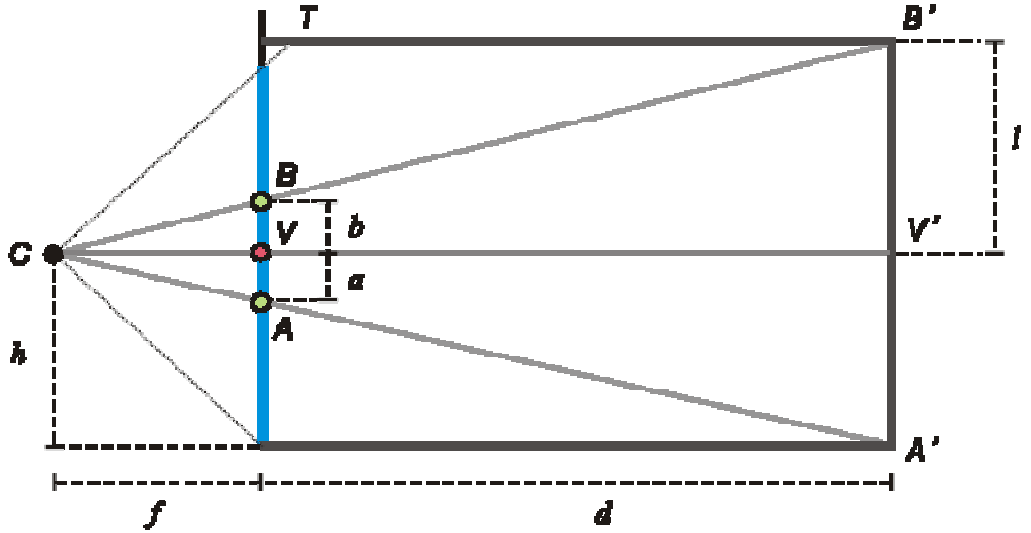
2D to 3D conversion

- Size of user-defined back plane must equal size of camera plane (orthogonal sides)
- Use top versus side ratio to determine relative height and width dimensions of box
- Left/right and top/bot ratios determine part of 3D camera placement



Depth of the box





Can compute by similar triangles (CVA vs. CV'A')

Need to know focal length f (or FOV)

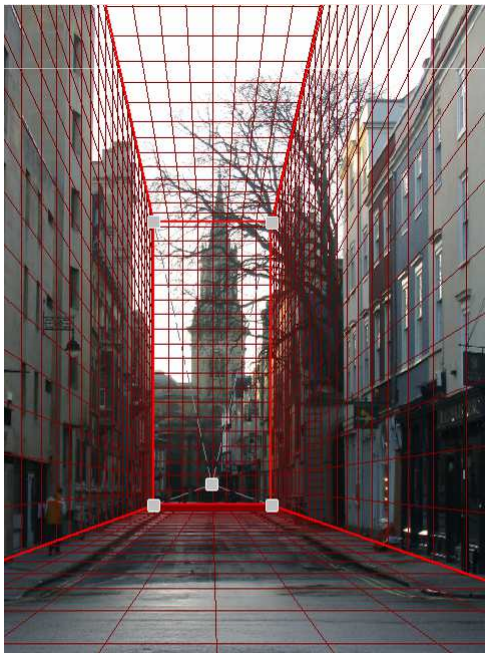
Note: can compute position on any object on the ground

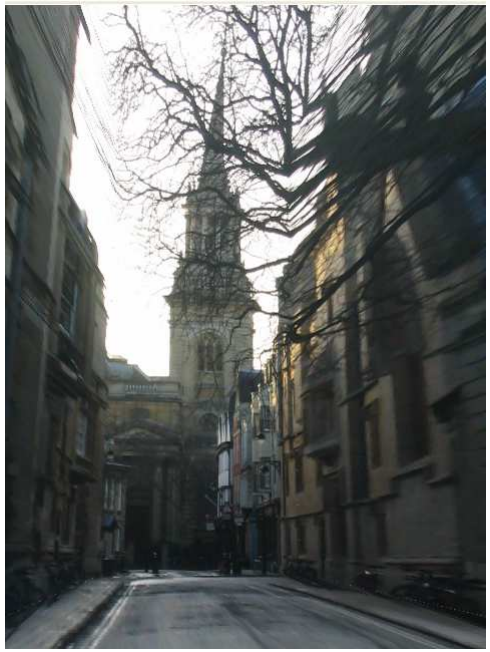
- Simple unprojection
- What about things off the ground?

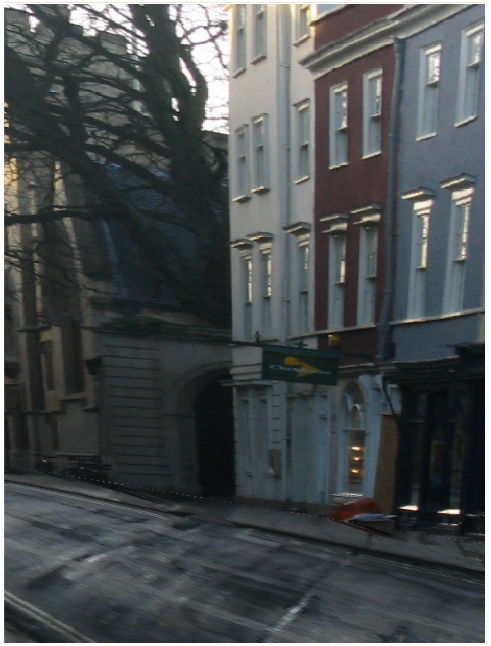
DEMO

Now, we know the 3D geometry of the box

We can texture-map the box walls with texture from the image





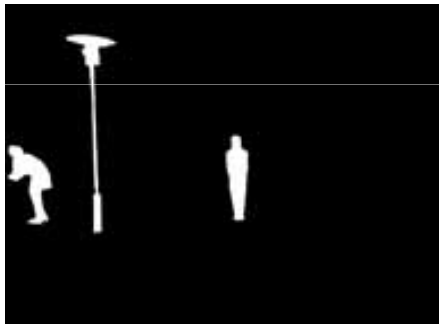


Foreground Objects

Use separate billboard for each

For this to work, three separate images used:

- Original image.
- Mask to isolate desired foreground images.
- Background with objects removed

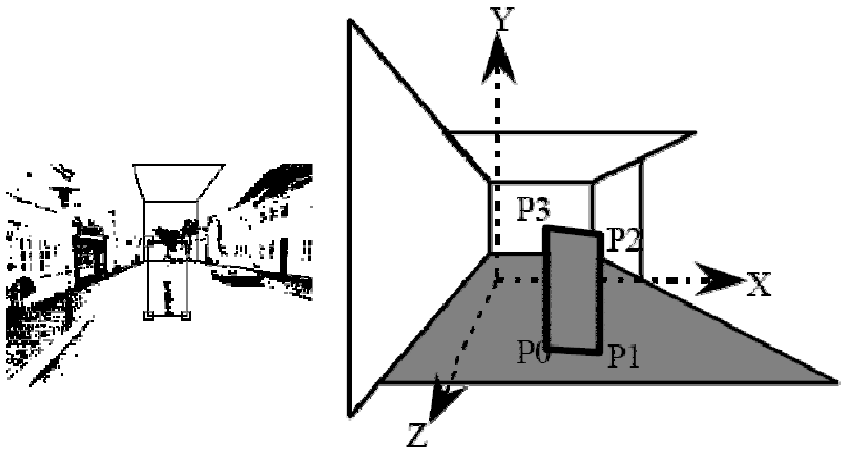


Foreground Objects

Add vertical rectangles for each foreground object

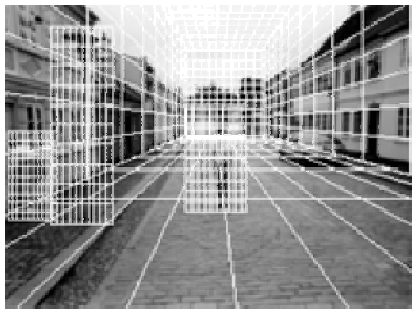
Can compute 3D coordinates P0, P1 since they are on known plane.

P2, P3 can be computed as before (similar triangles)



(a) Specifying of a foreground object

(b) Estimating the vertices of the foreground object model



(c) Three foreground object models

Foreground DEMO