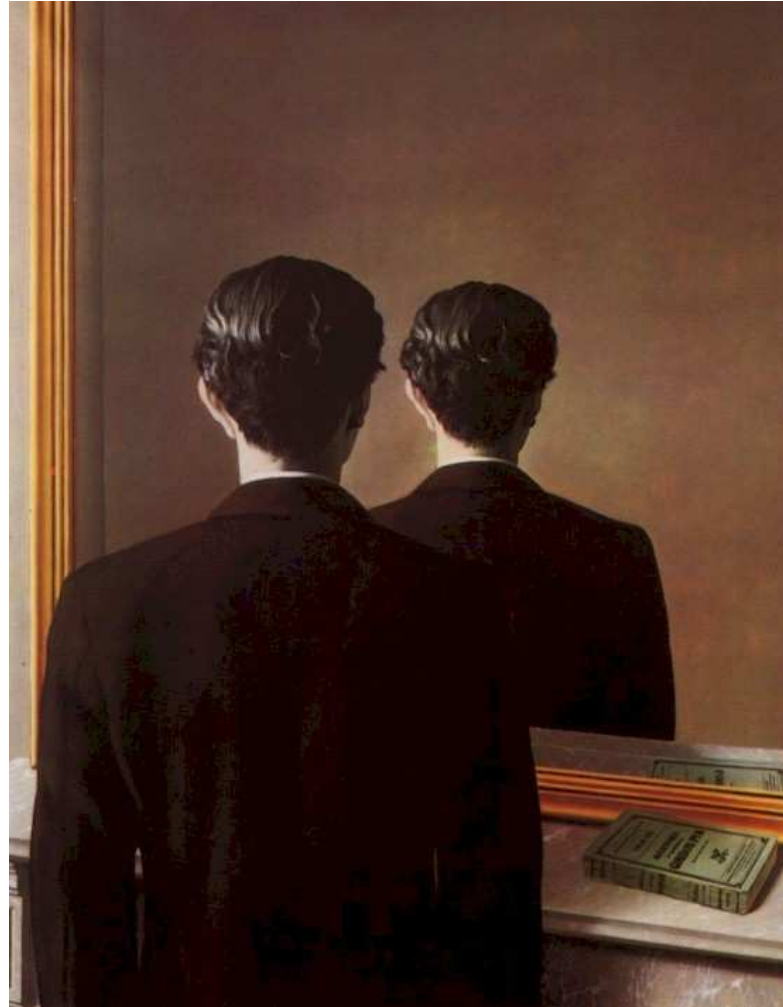


# Scene Modeling for a Single View

---



René MAGRITTE  
*Portrait d'Edward James*

*...with a lot of slides stolen from  
Steve Seitz and David Brogan,*

15-463: Computational Photography  
Alexei Efros, CMU, Fall 2011



# 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 more of the  
plenoptic function

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

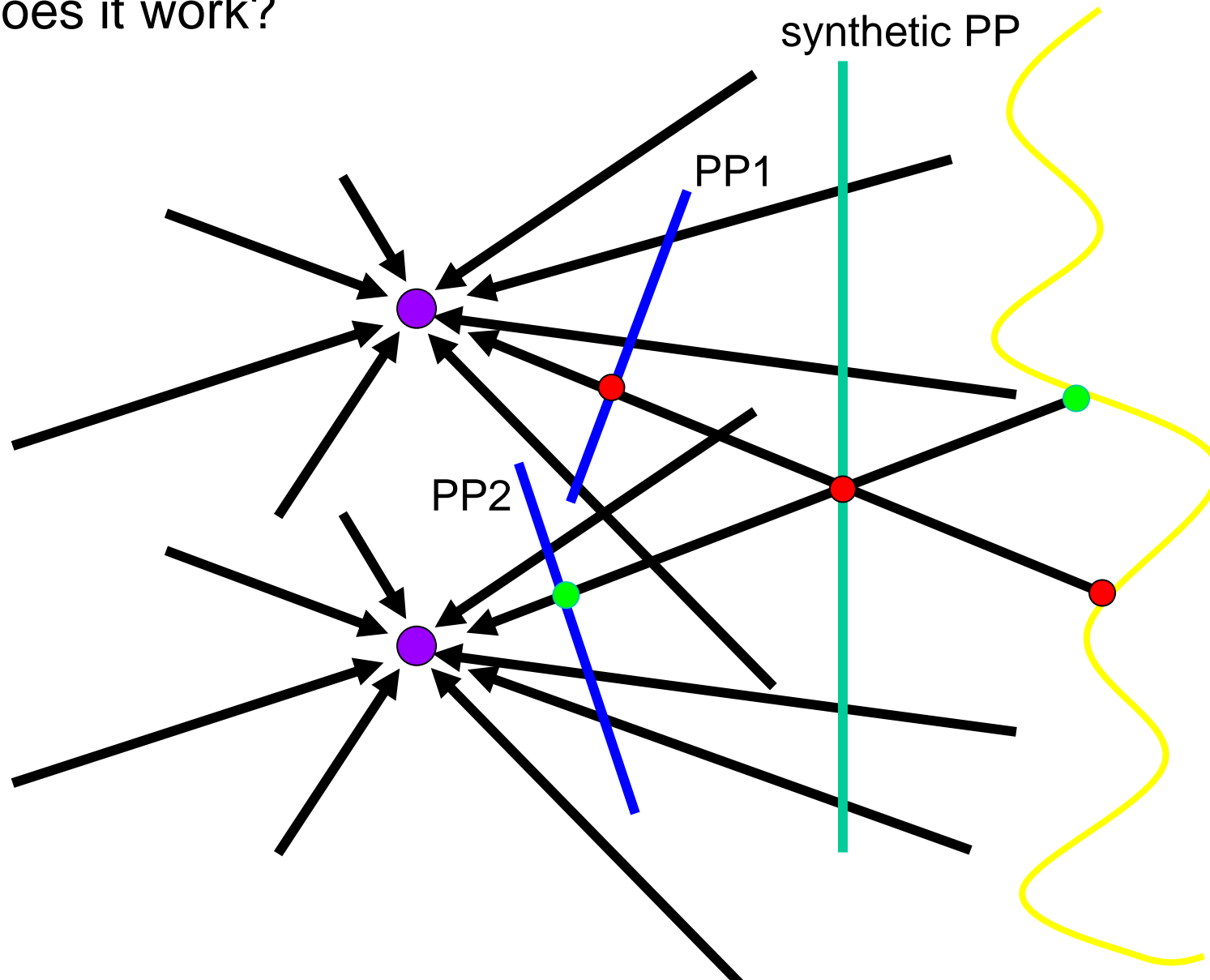




# Camera translation

---

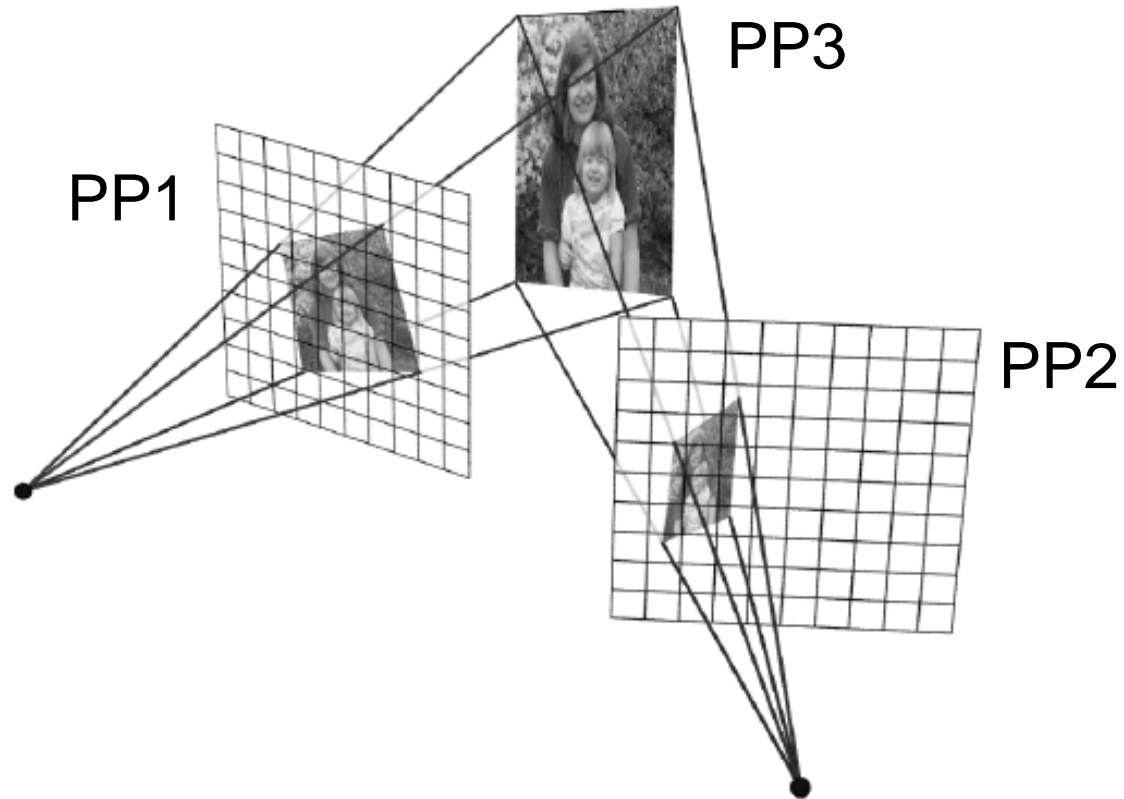
Does it work?





# Yes, with planar scene (or far away)

---



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



\_\_\_\_\_





# Some preliminaries: projective geometry

---



[Ames Room](#)



# Silly Euclid: Trix are for kids!

---



Parallel lines???



# The projective plane

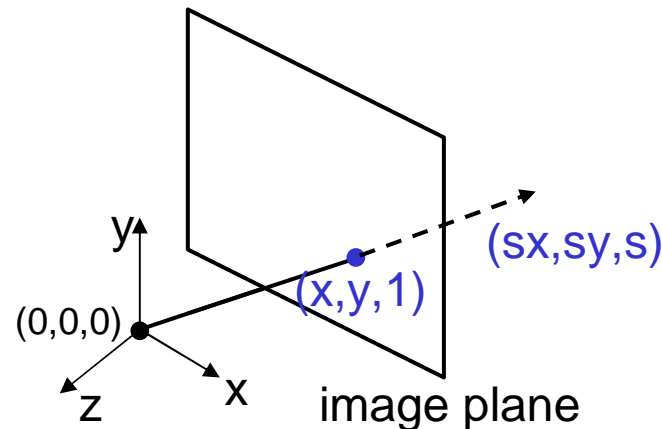
---

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



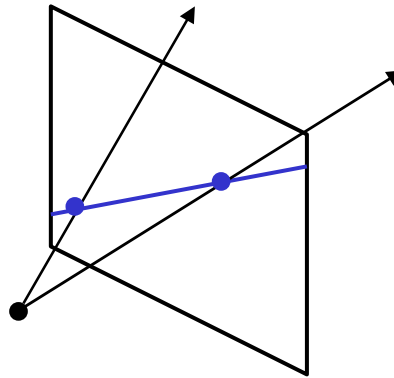
- Each *point*  $(x,y)$  on the plane is represented by a *ray*  $(sx,sy,s)$ 
  - all points on the ray are equivalent:  $(x, y, 1) \equiv (sx, sy, s)$



# 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:  $ax + by + cz = 0$

in vector notation:  $0 = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$

**l      p**

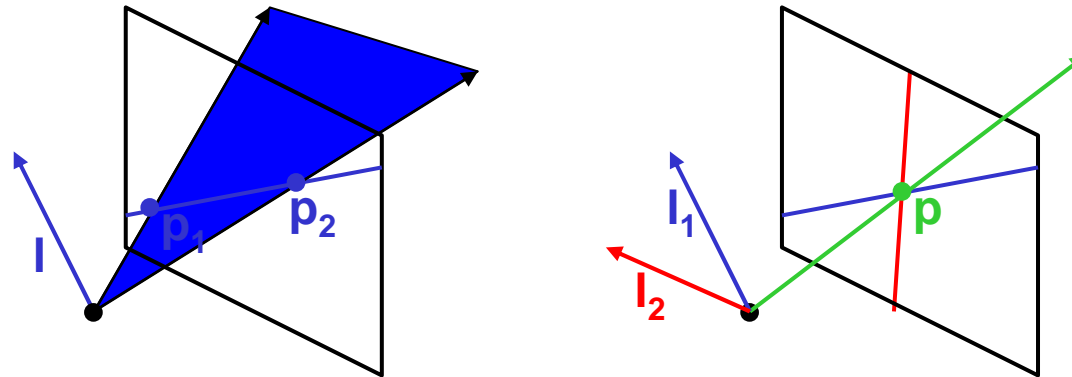
- A line is also represented as a homogeneous 3-vector **l**



# Point and line duality

---

- A line  $\mathbf{l}$  is a homogeneous 3-vector
- It is  $\perp$  to every point (ray)  $\mathbf{p}$  on the line:  $\mathbf{l} \cdot \mathbf{p} = 0$



What is the line  $\mathbf{l}$  spanned by rays  $\mathbf{p}_1$  and  $\mathbf{p}_2$  ?

- $\mathbf{l}$  is  $\perp$  to  $\mathbf{p}_1$  and  $\mathbf{p}_2 \Rightarrow \mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$
- $\mathbf{l}$  is the plane normal

What is the intersection of two lines  $\mathbf{l}_1$  and  $\mathbf{l}_2$  ?

- $\mathbf{p}$  is  $\perp$  to  $\mathbf{l}_1$  and  $\mathbf{l}_2 \Rightarrow \mathbf{p} = \mathbf{l}_1 \times \mathbf{l}_2$

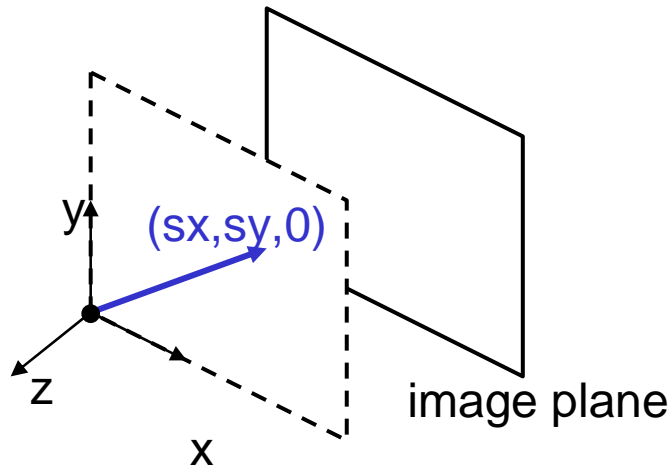
Points and lines are *dual* in projective space

- given any formula, can switch the meanings of points and lines to get another formula



# Ideal points and lines

---



## Ideal point (“point at infinity”)

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

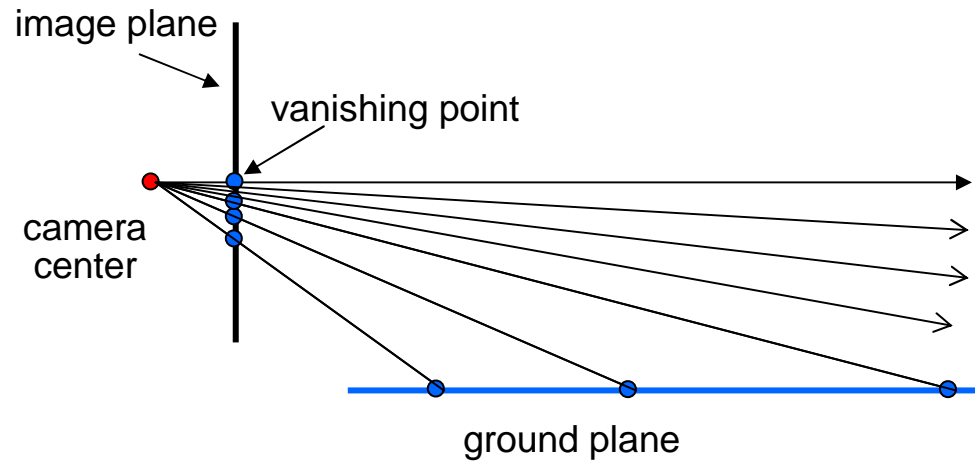
## Ideal line

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



# Vanishing points

---



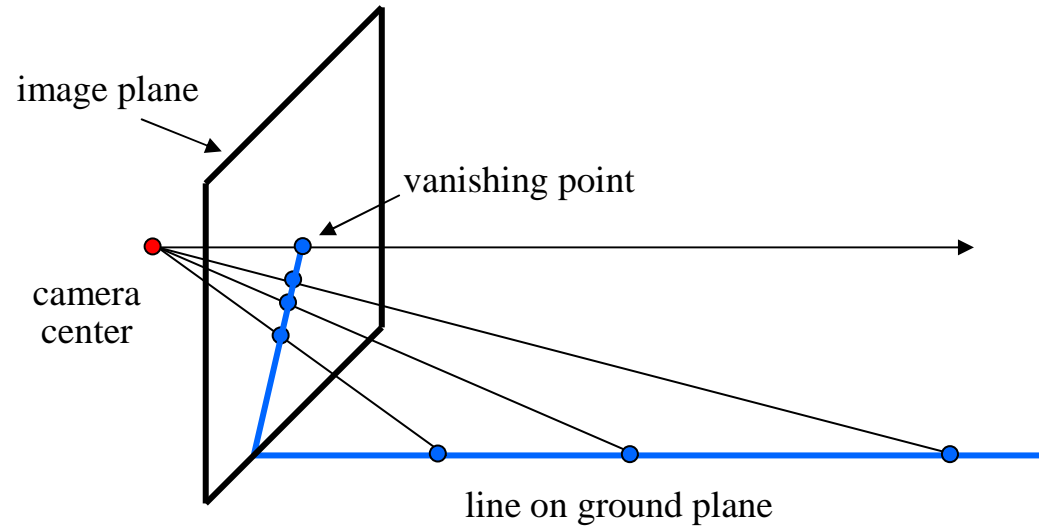
## Vanishing point

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



# Vanishing points (2D)

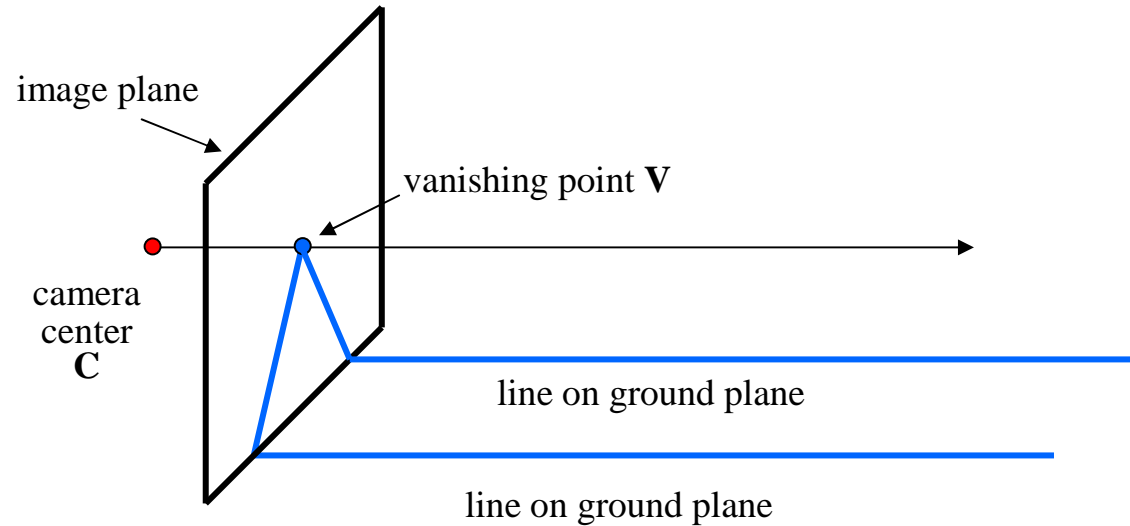
---





# Vanishing points

---



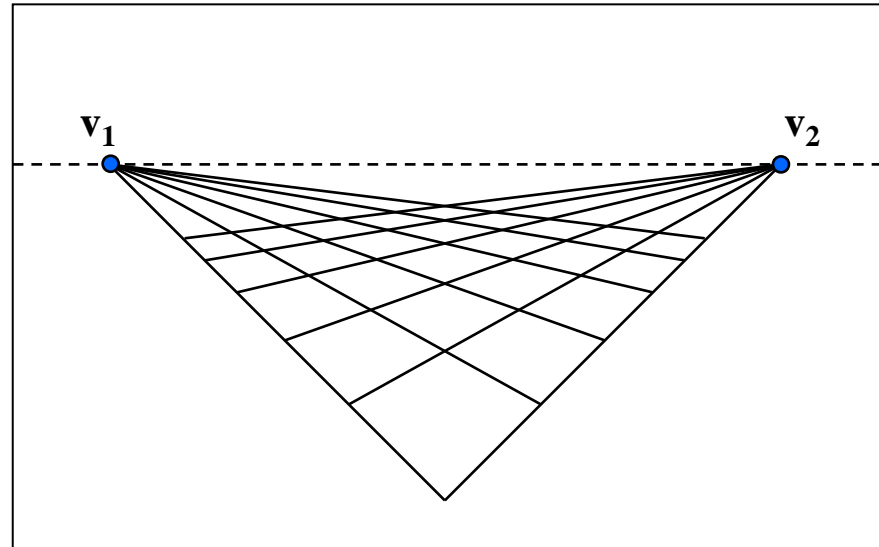
## Properties

- 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

---



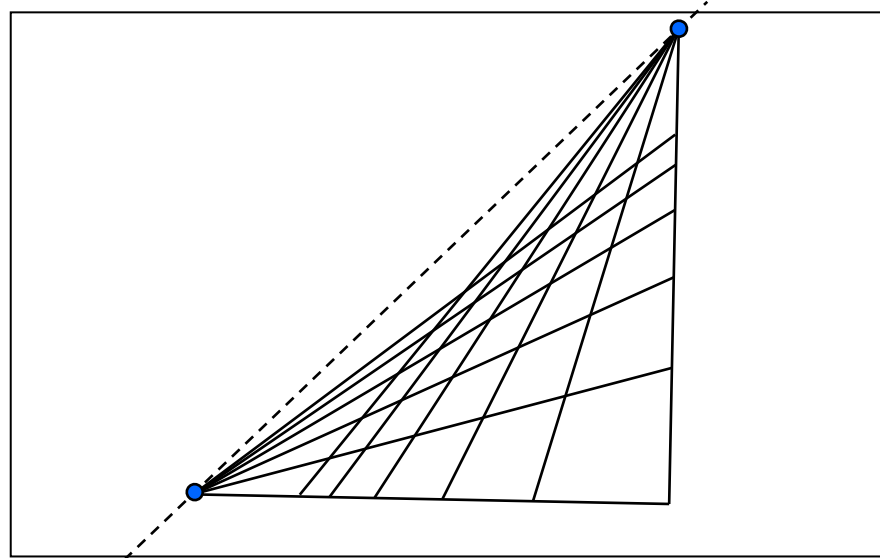
## Multiple Vanishing Points

- 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

---



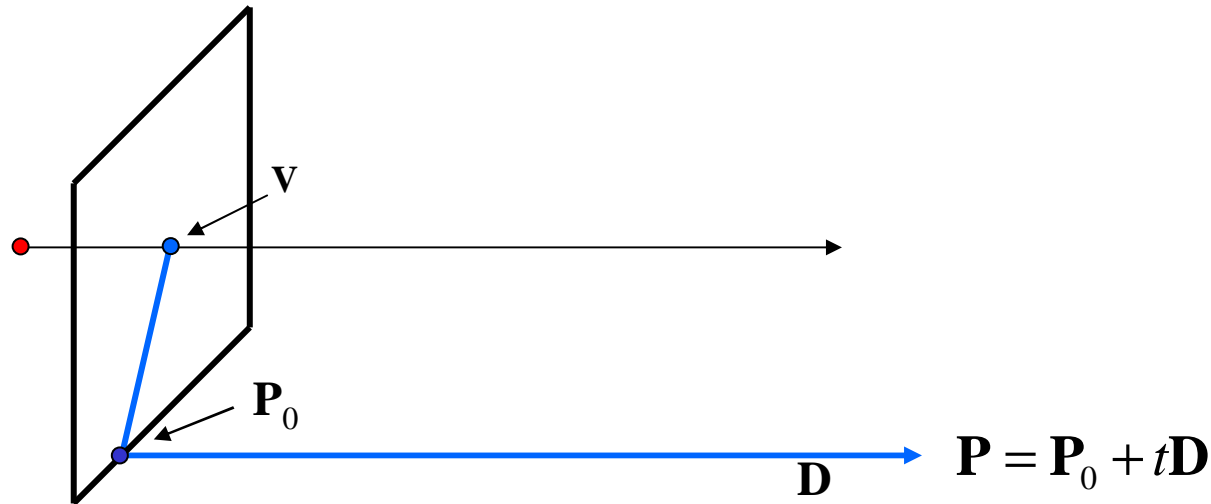
## Multiple Vanishing Points

- 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

---



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

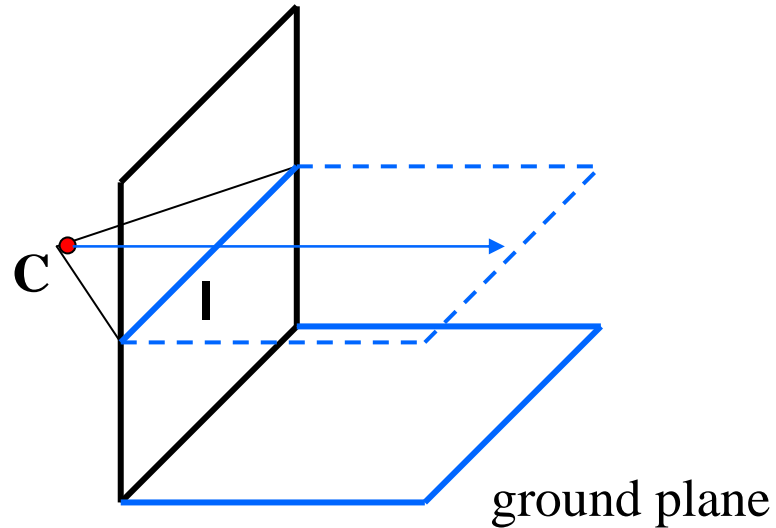
Properties  $\mathbf{v} = \Pi \mathbf{P}_\infty$

- $\mathbf{P}_\infty$  is a point at *infinity*,  $\mathbf{v}$  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

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

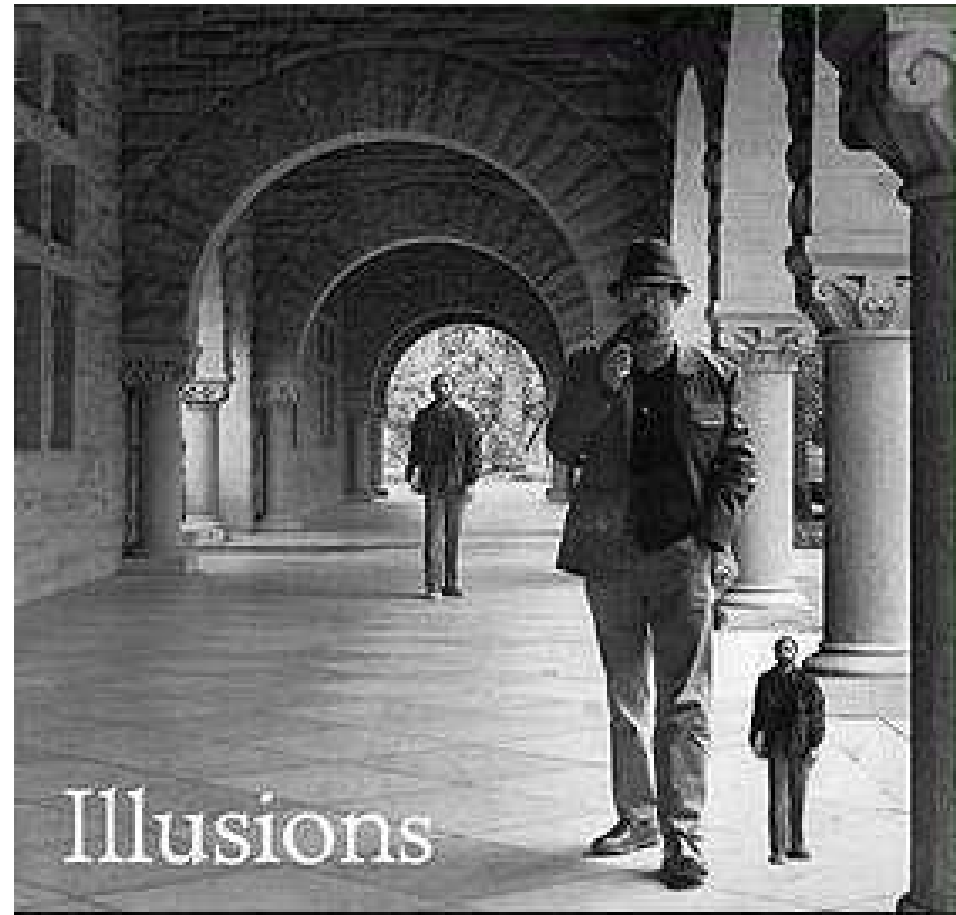
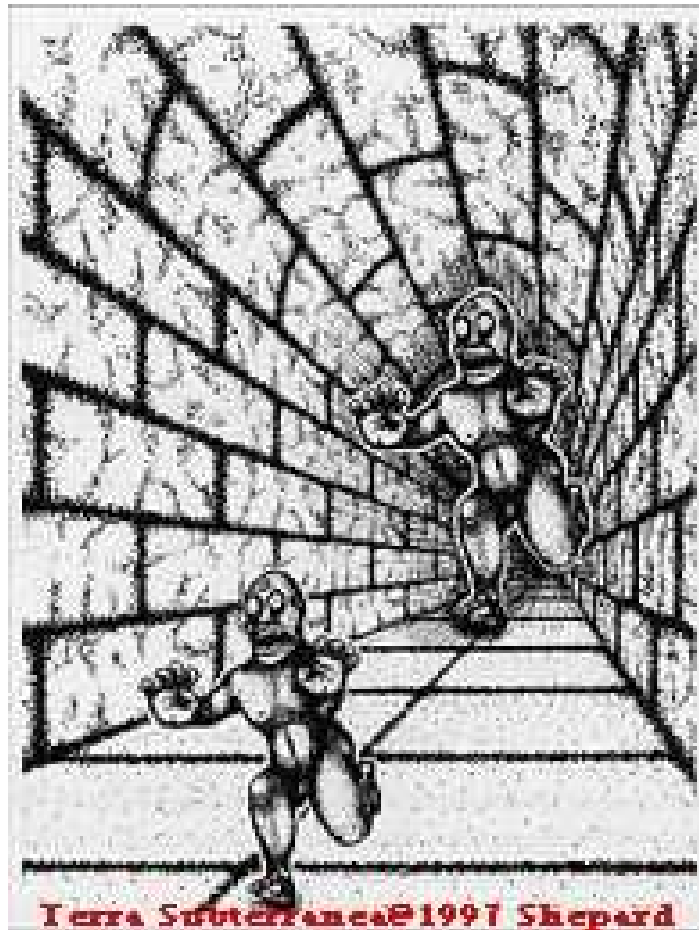






# Fun with vanishing points

---





# “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)

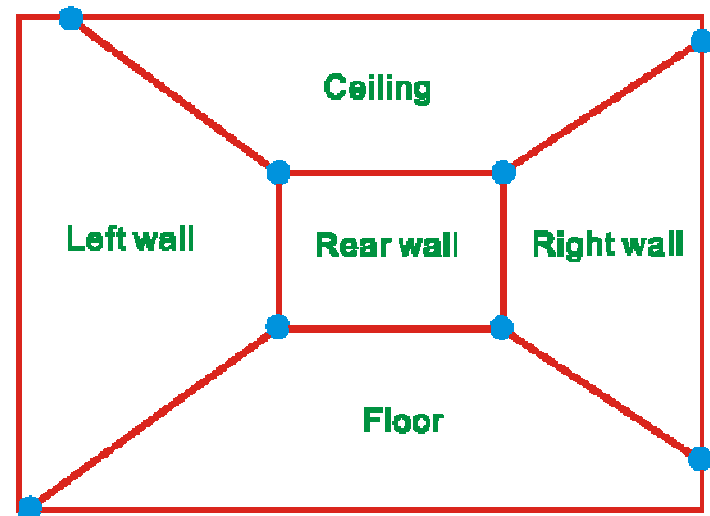
Key assumptions:

- 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

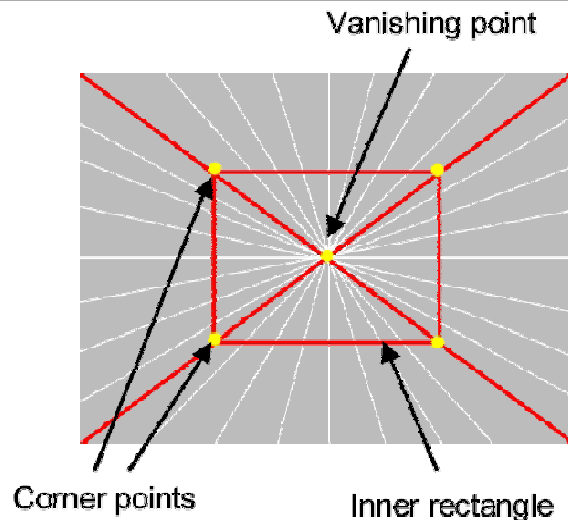
Can use the vanishing point to fit the box to the particular Scene!





# Fitting the box volume

---



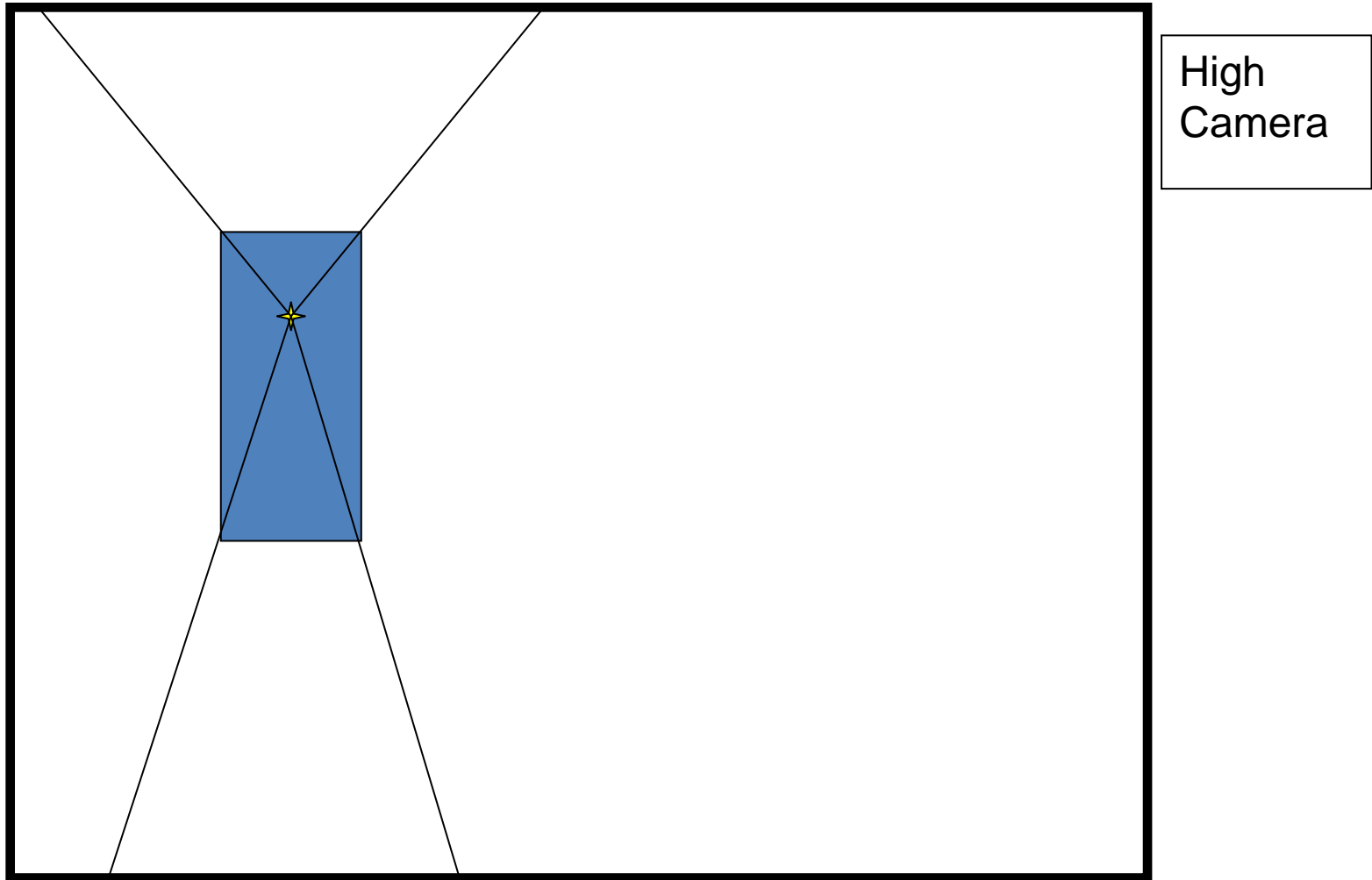
User controls the inner box and the vanishing point placement (# of DOF???)

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.

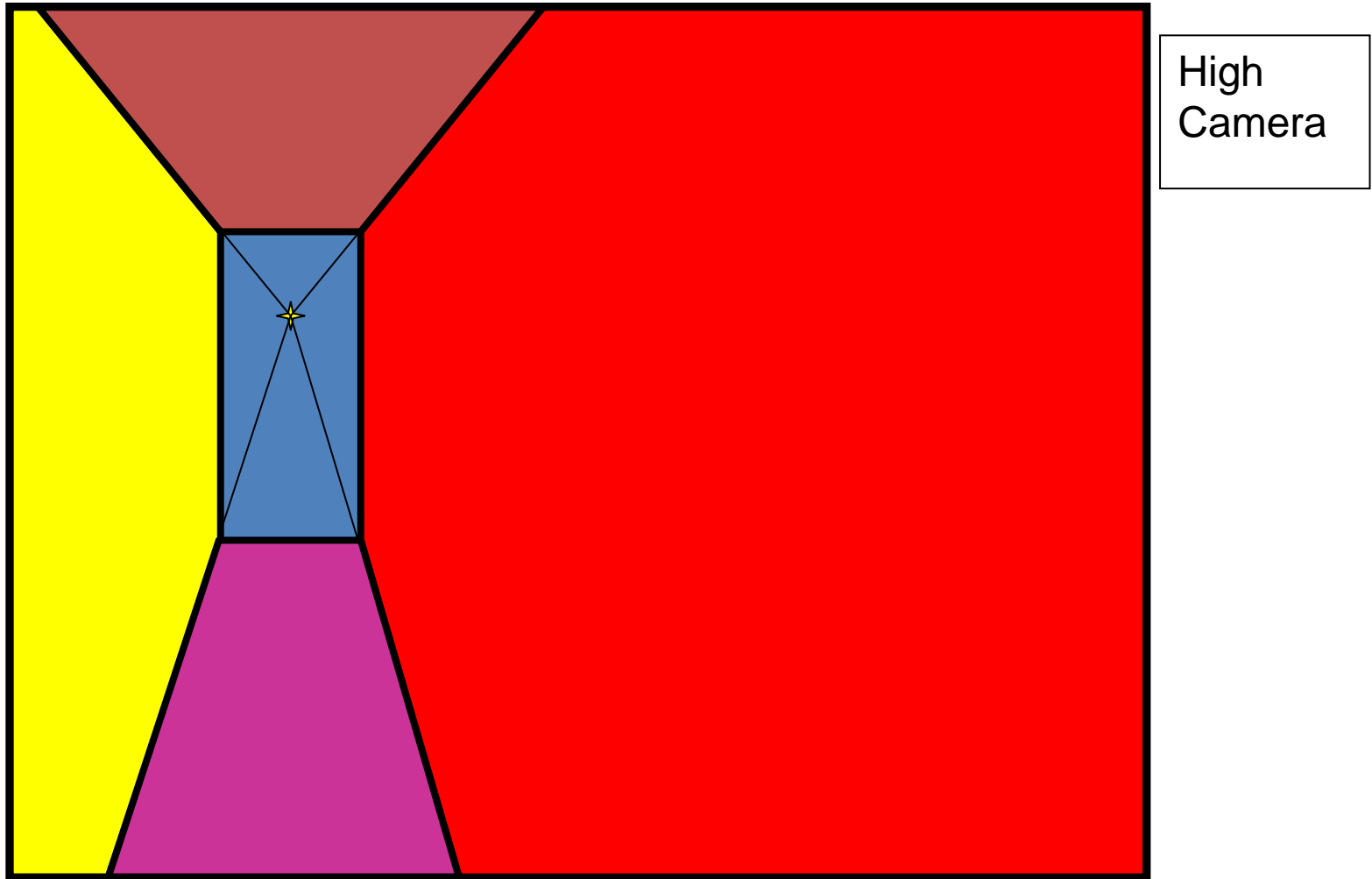


Example of user input: vanishing point and back face of view volume are defined



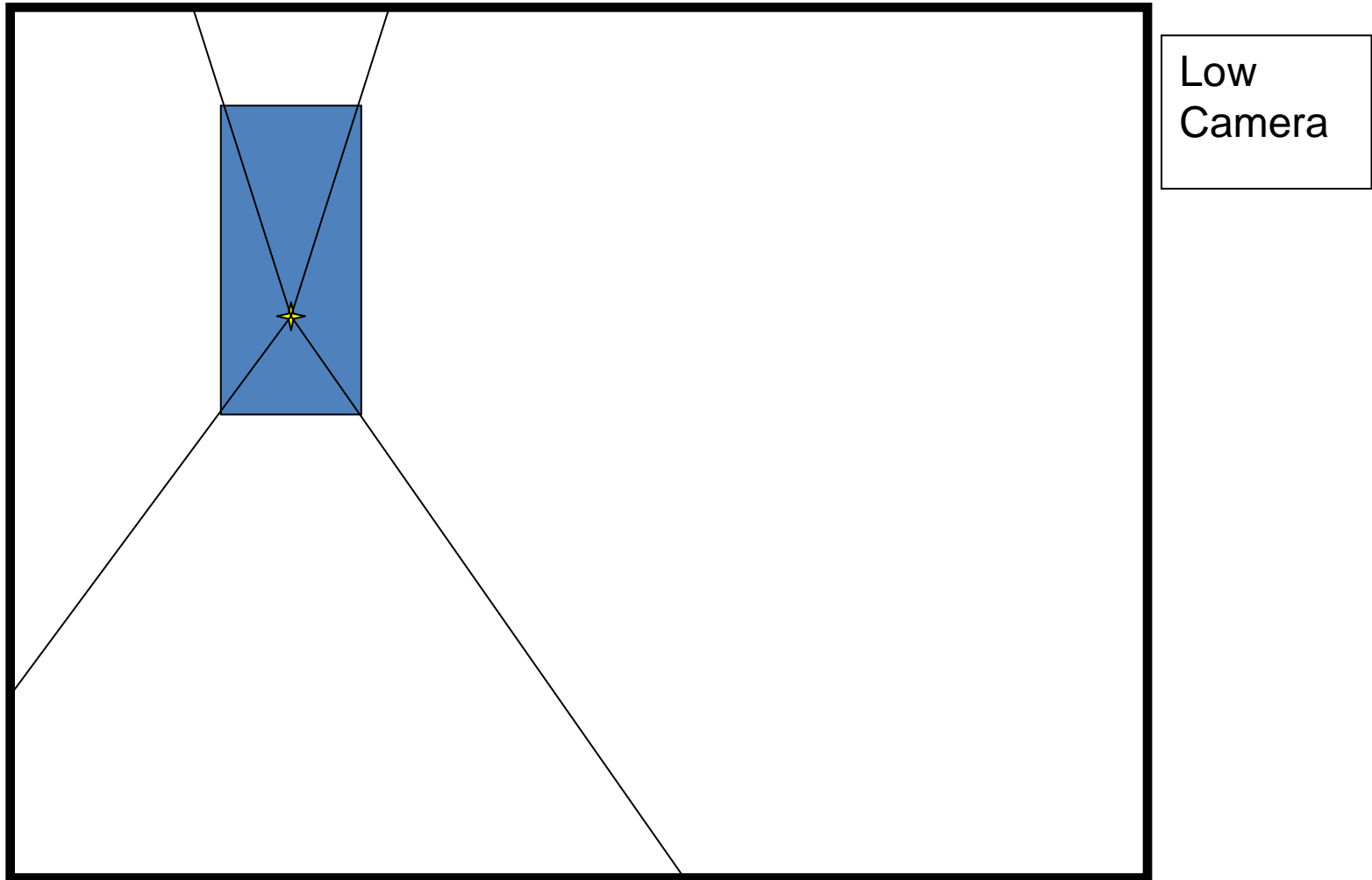


Example of user input: vanishing point and back face of view volume are defined



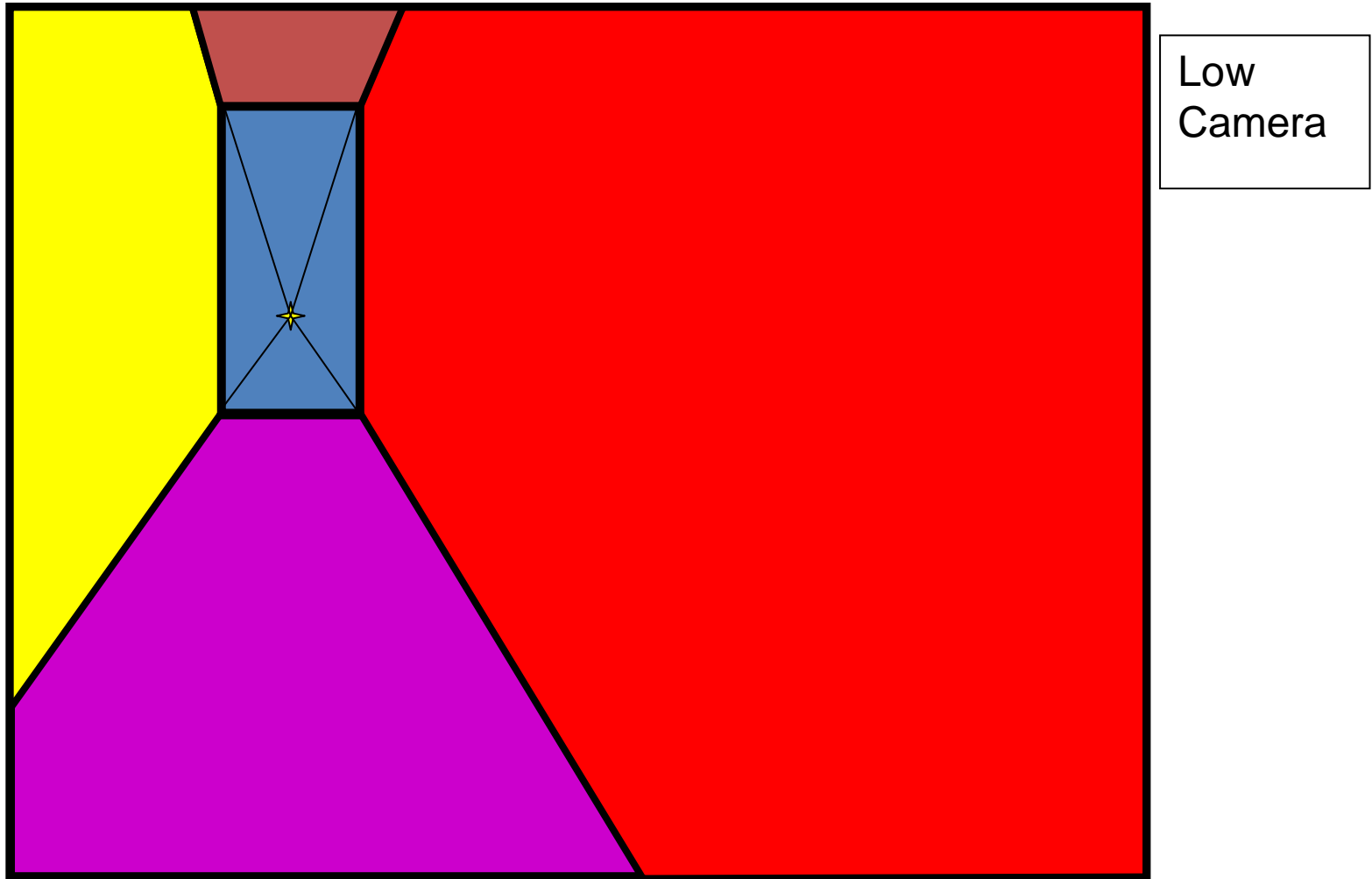


Example of user input: vanishing point and back face of view volume are defined



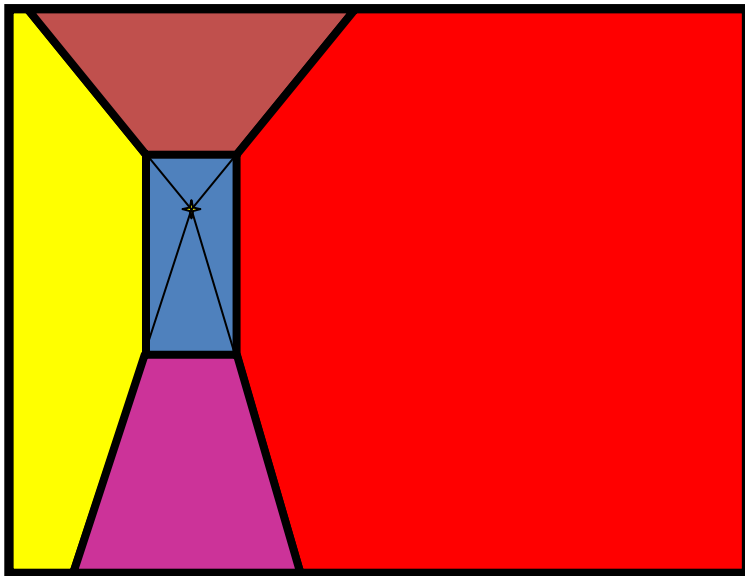


Example of user input: vanishing point and back face of view volume are defined

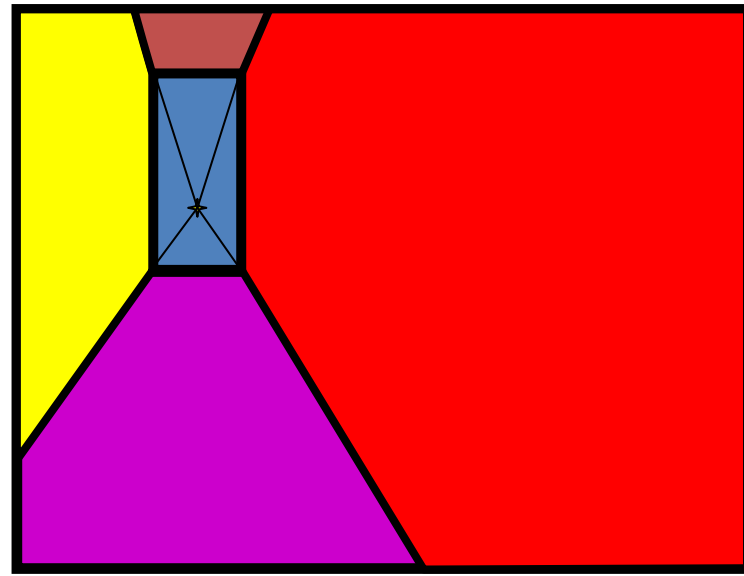




Comparison of how image is subdivided based on two different camera positions. You should see how moving the box corresponds to moving the eyepoint in the 3D world.



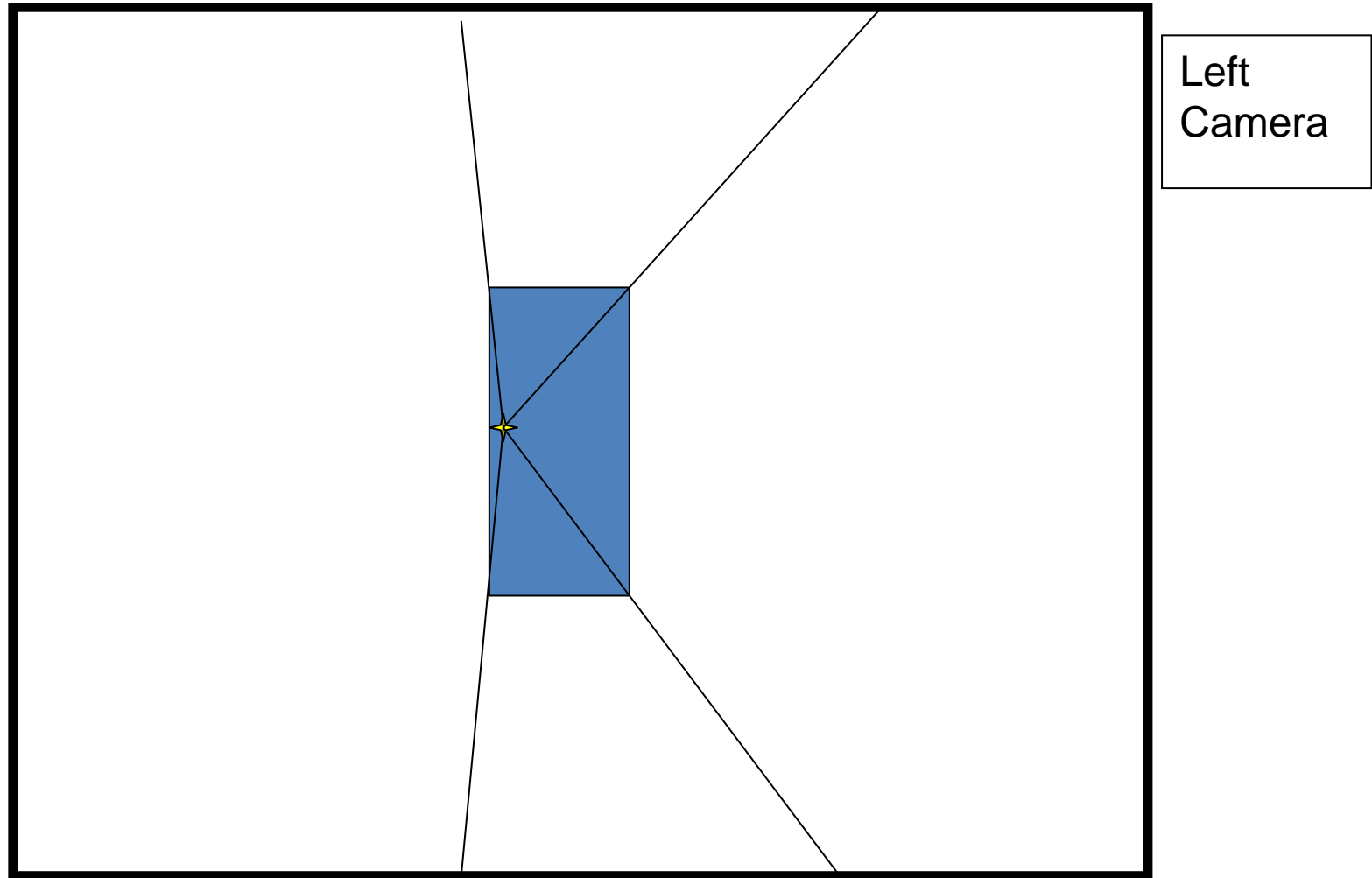
High Camera



Low Camera

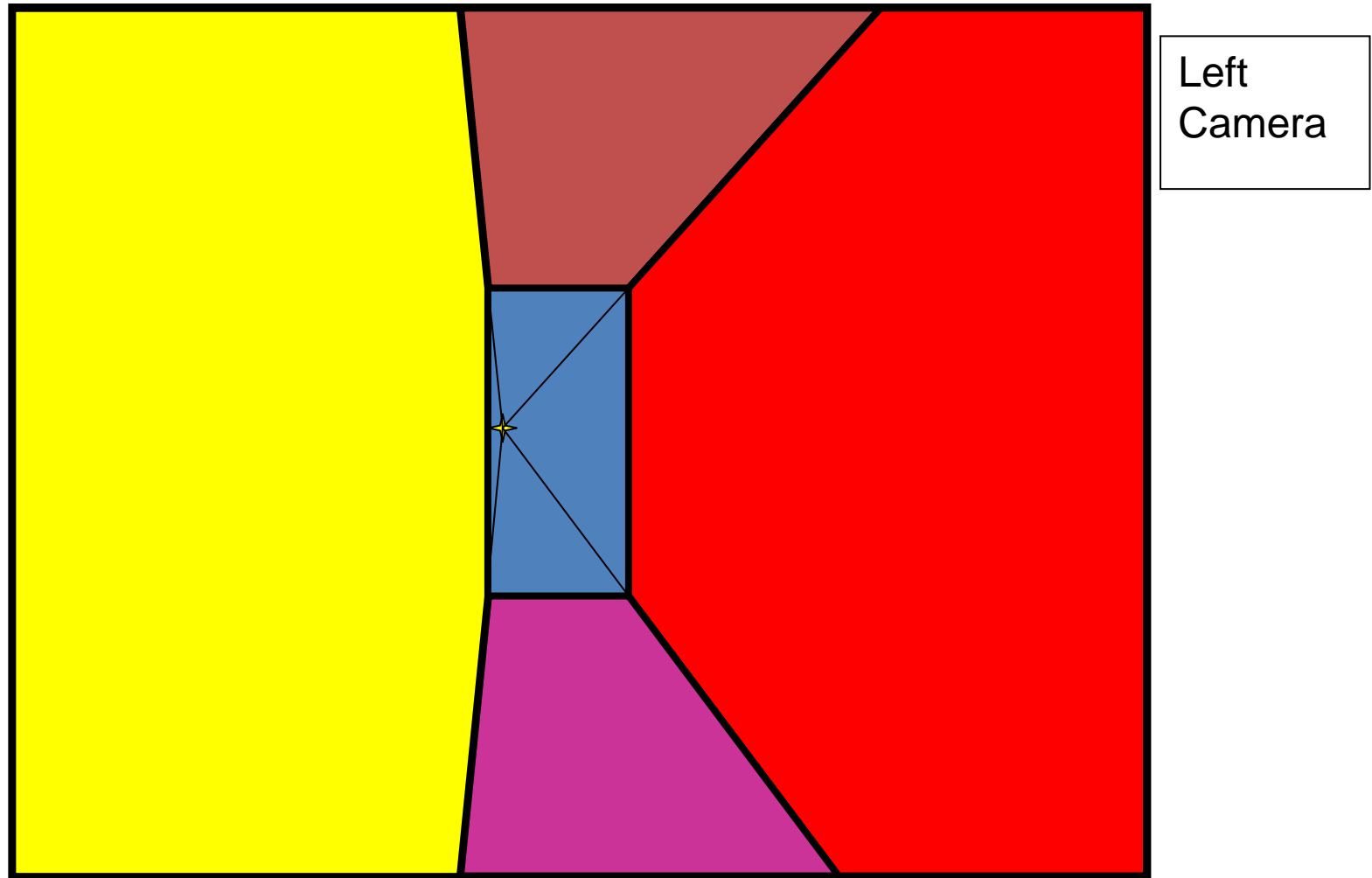


Another example of user input: vanishing point and back face of view volume are defined



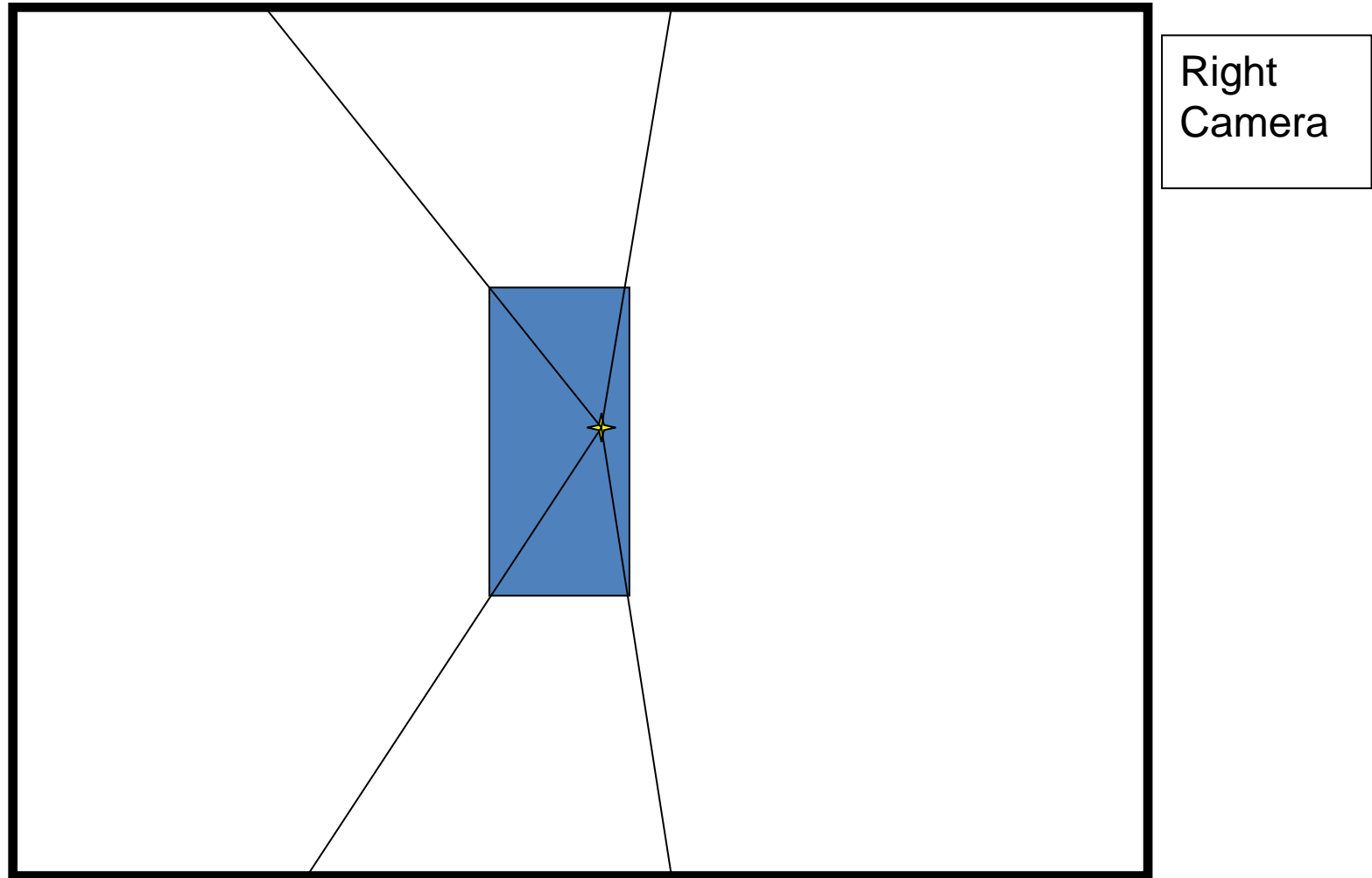


Another example of user input: vanishing point and back face of view volume are defined



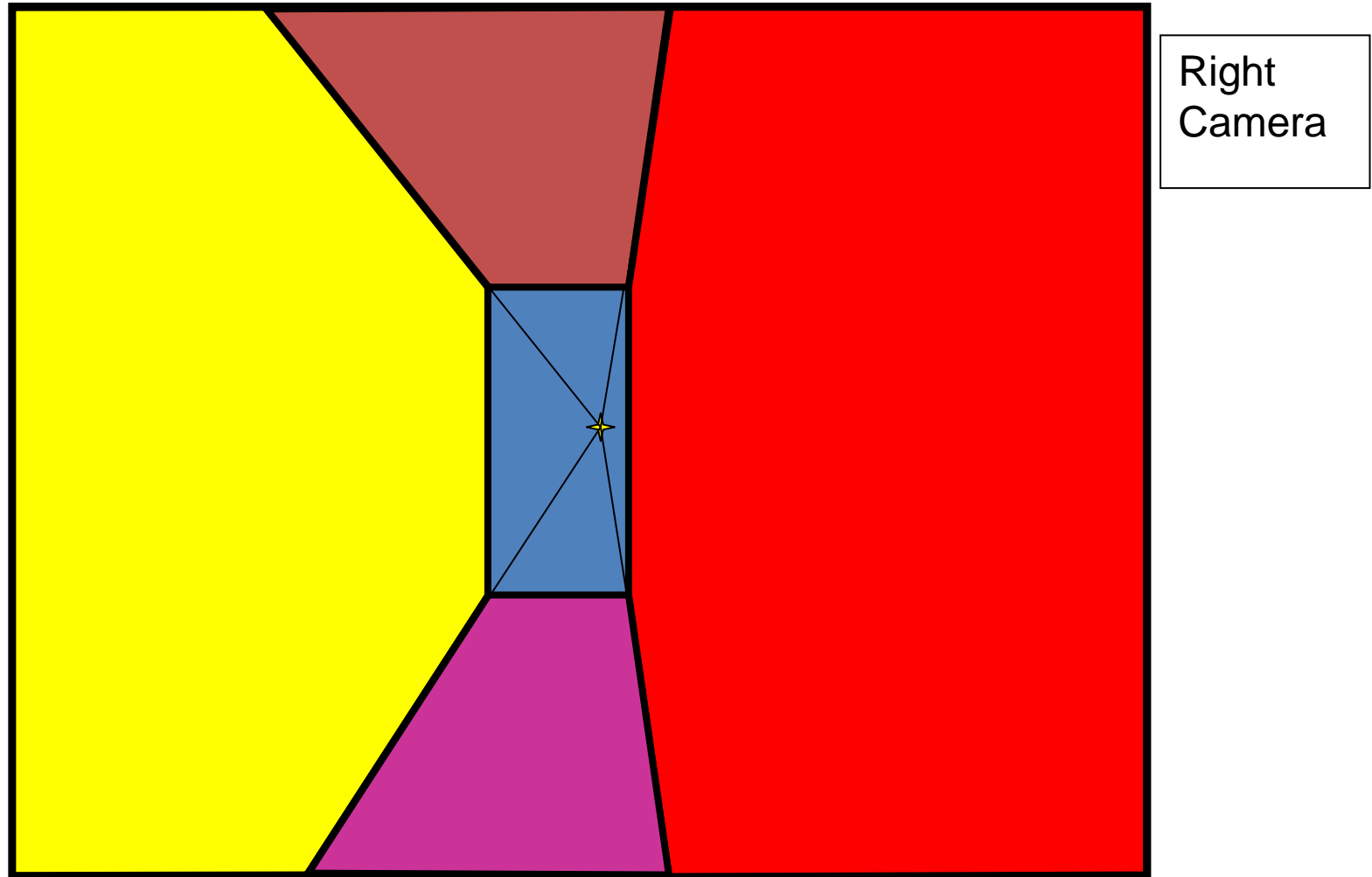


Another example of user input: vanishing point and back face of view volume are defined



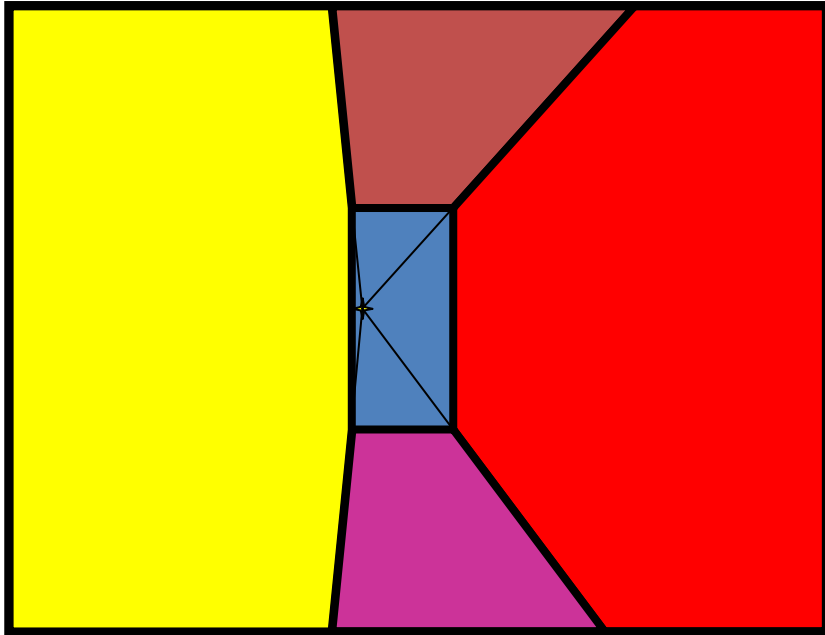


Another example of user input: vanishing point and back face of view volume are defined

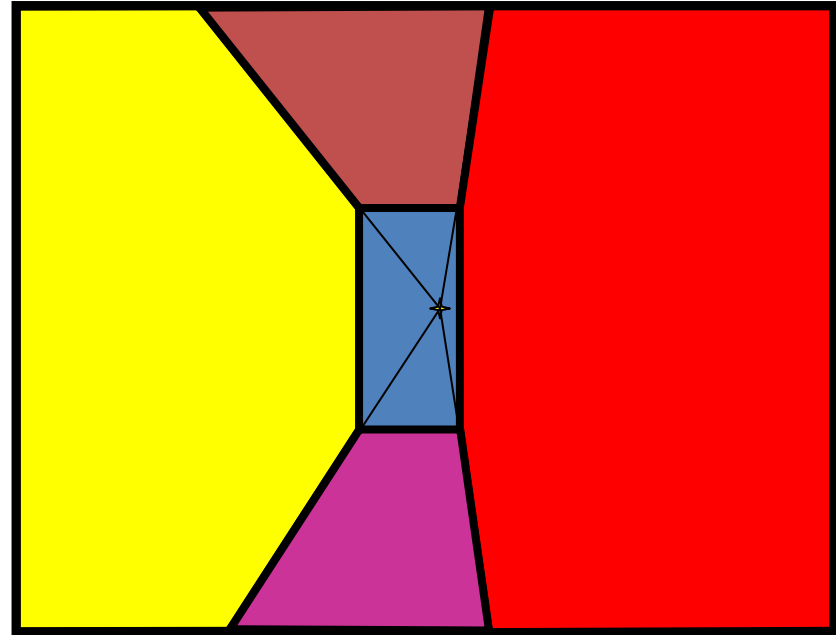




Comparison of two camera placements – left and right.  
Corresponding subdivisions match view you would see if  
you looked down a hallway.



Left Camera



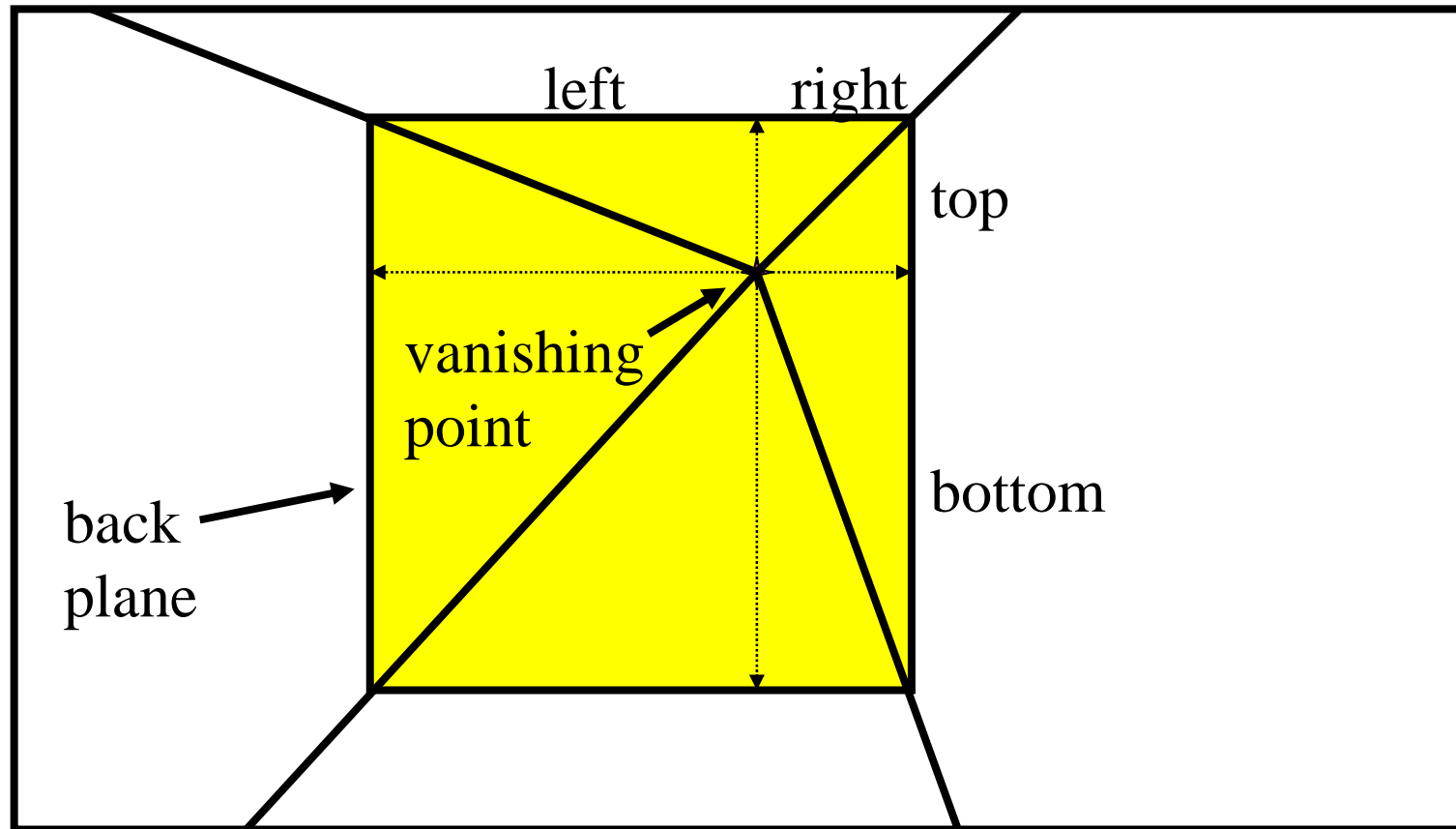
Right Camera



# 2D to 3D conversion

---

First, we can get ratios

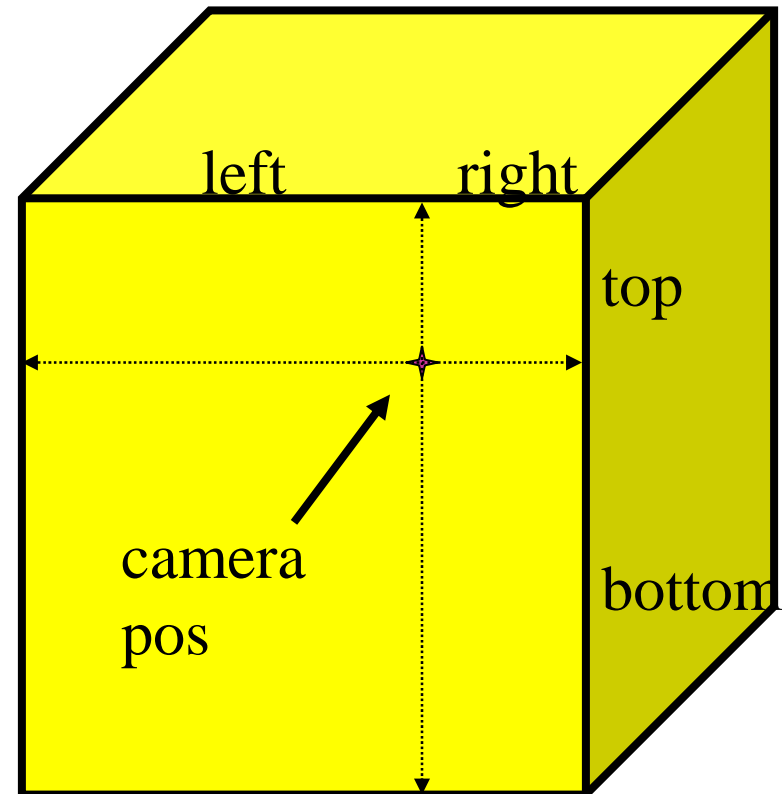




# 2D to 3D conversion

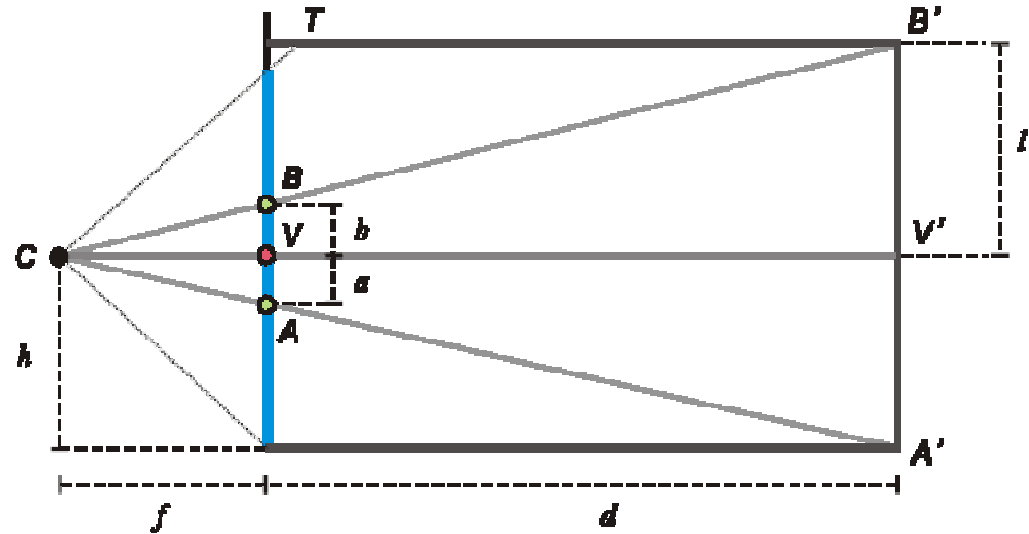
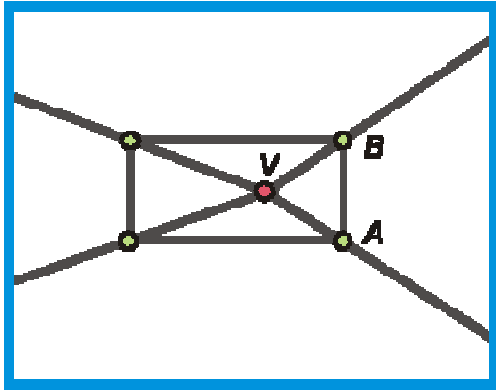
---

- 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?

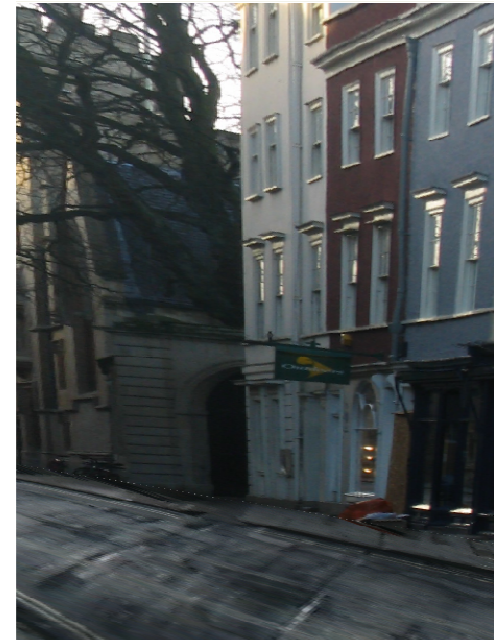
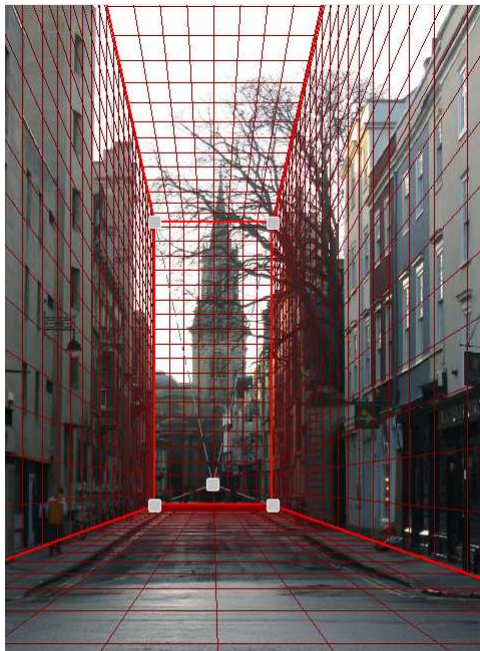


# DEMO

---

Now, we know the 3D geometry of the box

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





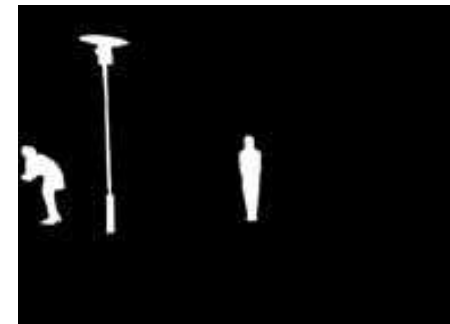
# Foreground Objects

---

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





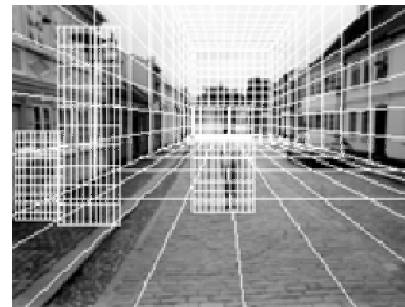
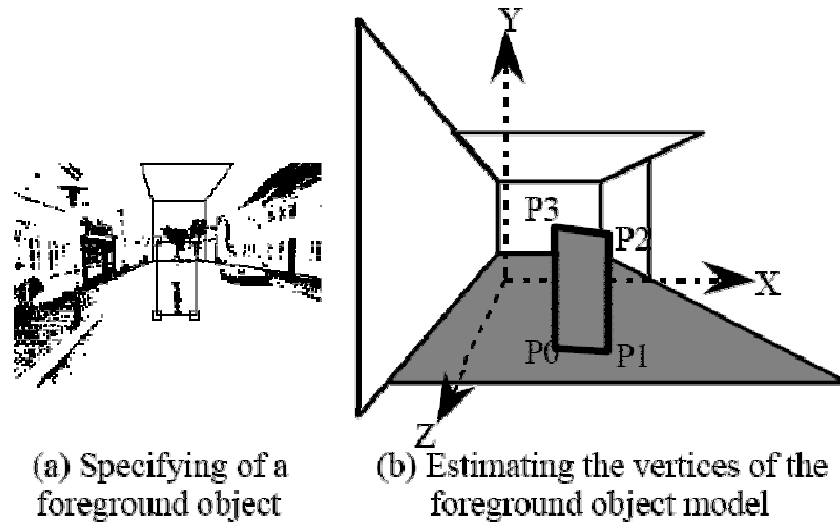
# Foreground Objects

---

Add vertical rectangles for each foreground object

Can compute 3D coordinates  $P_0$ ,  $P_1$  since they are on known plane.

$P_2$ ,  $P_3$  can be computed as before (similar triangles)



(c) Three foreground object models



# Foreground DEMO

---

