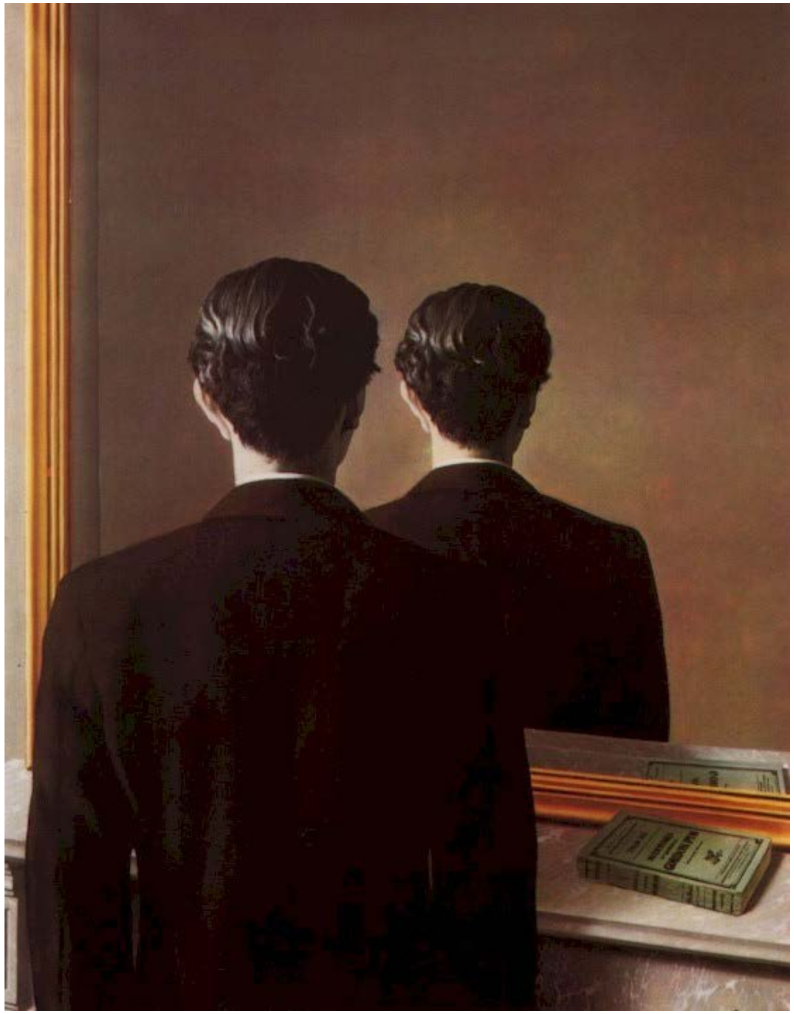
Scene Modeling for a Single View



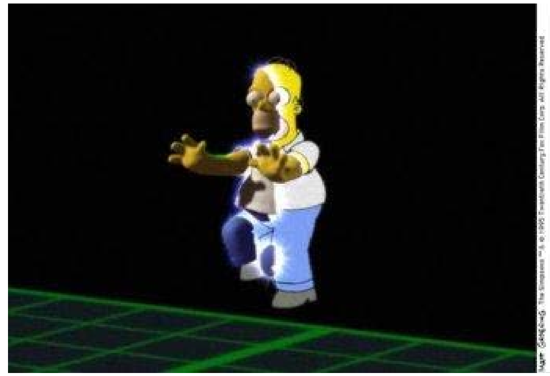
René MAGRITTE
Portrait d'Edward James

CS194: Image Manipulation & Computational Photography
Alexei Efros, UC Berkeley, Fall 2014

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

Breaking out of 2D

…now we are ready to break out of 2D

And enter the real world!

on to 3D…

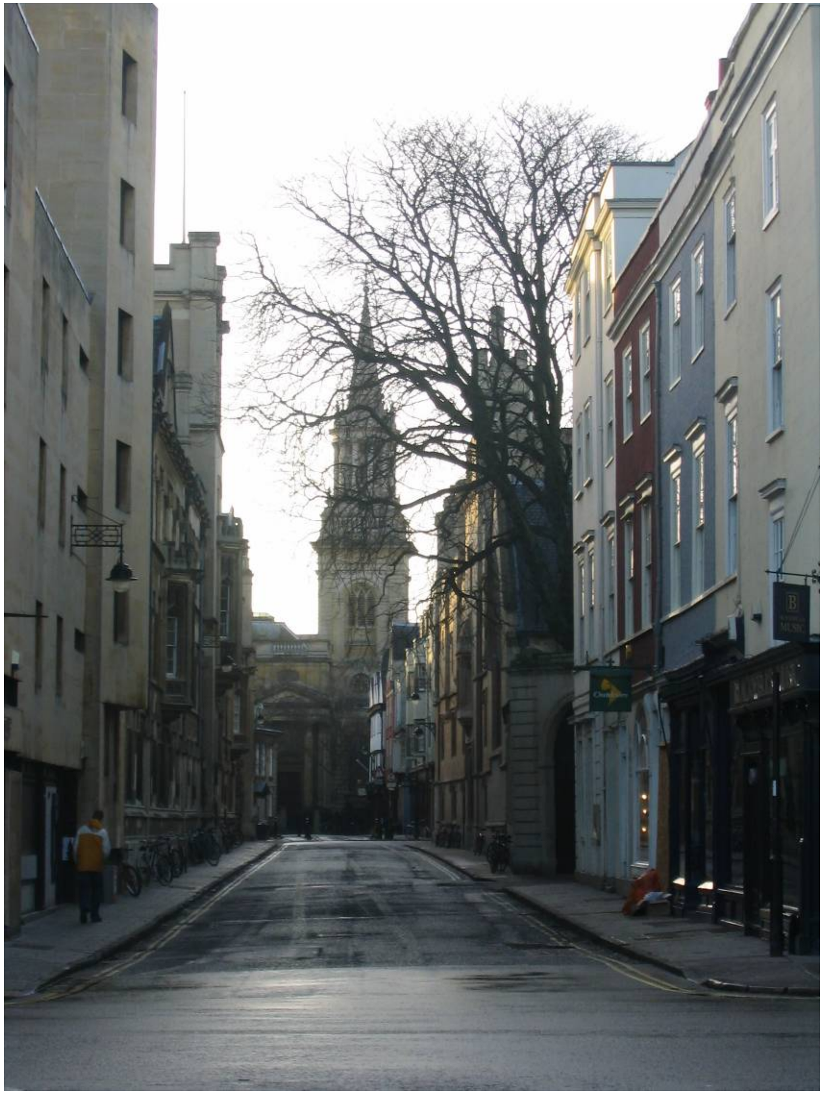
Enough of images!

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

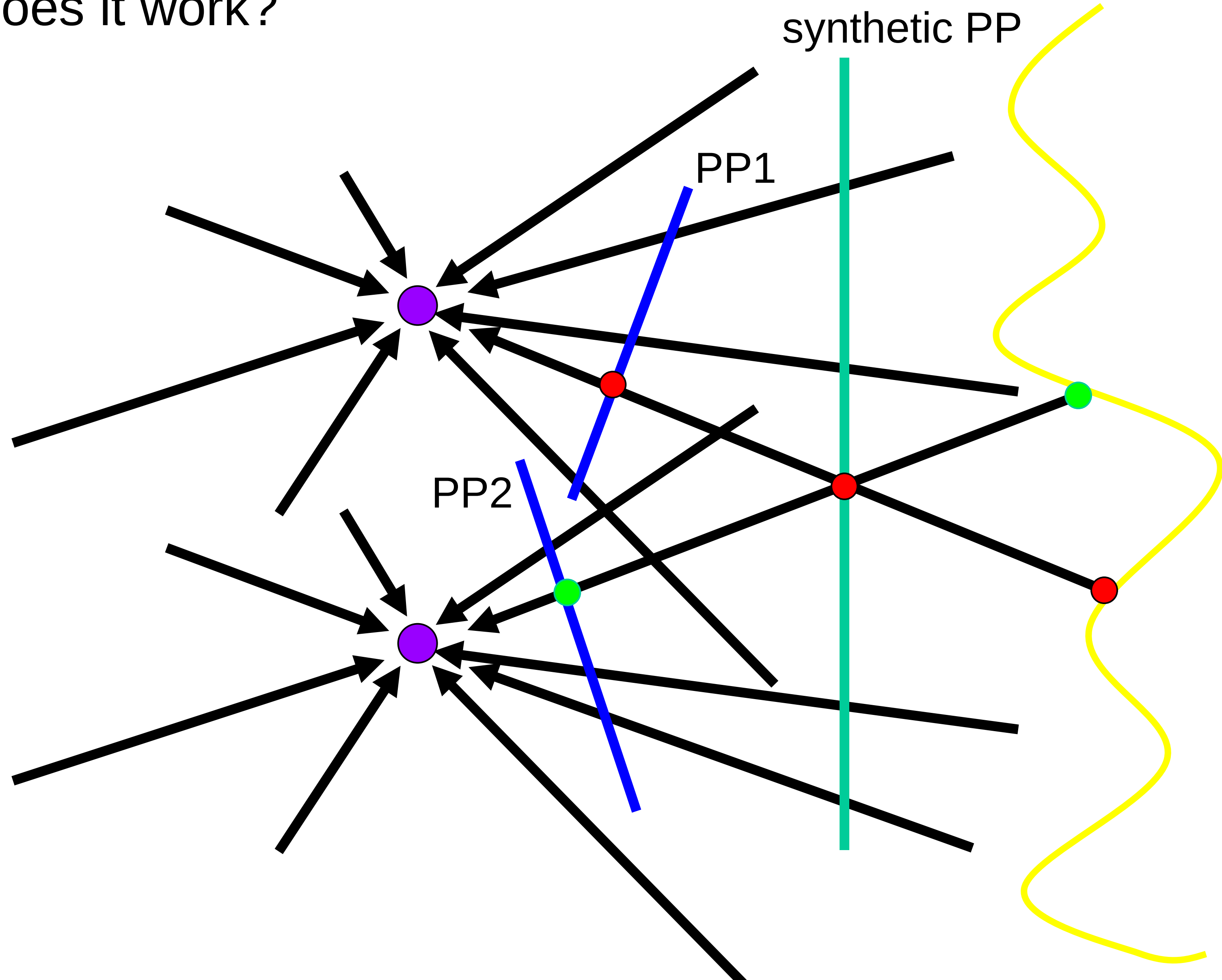
Original image

St.Petersburg
photo by A. Tikhonov

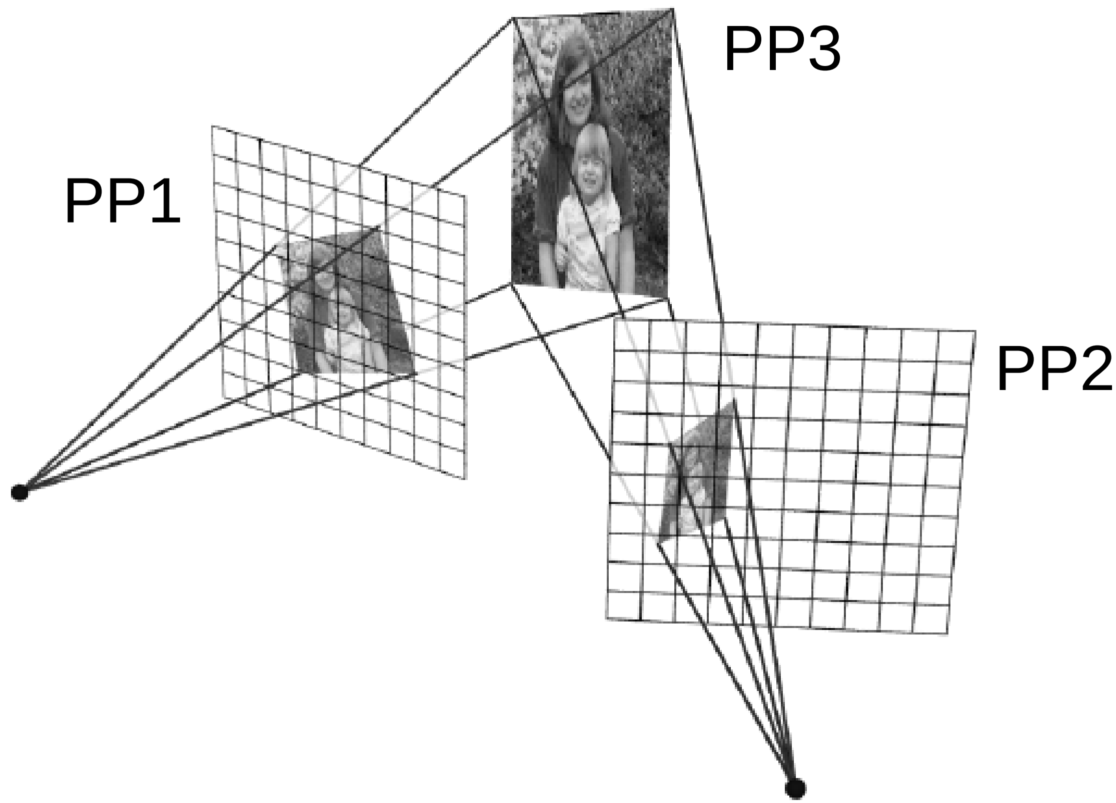
Virtual camera rotations

Camera translation

Does it work?



Yes, with planar scene (or far away)

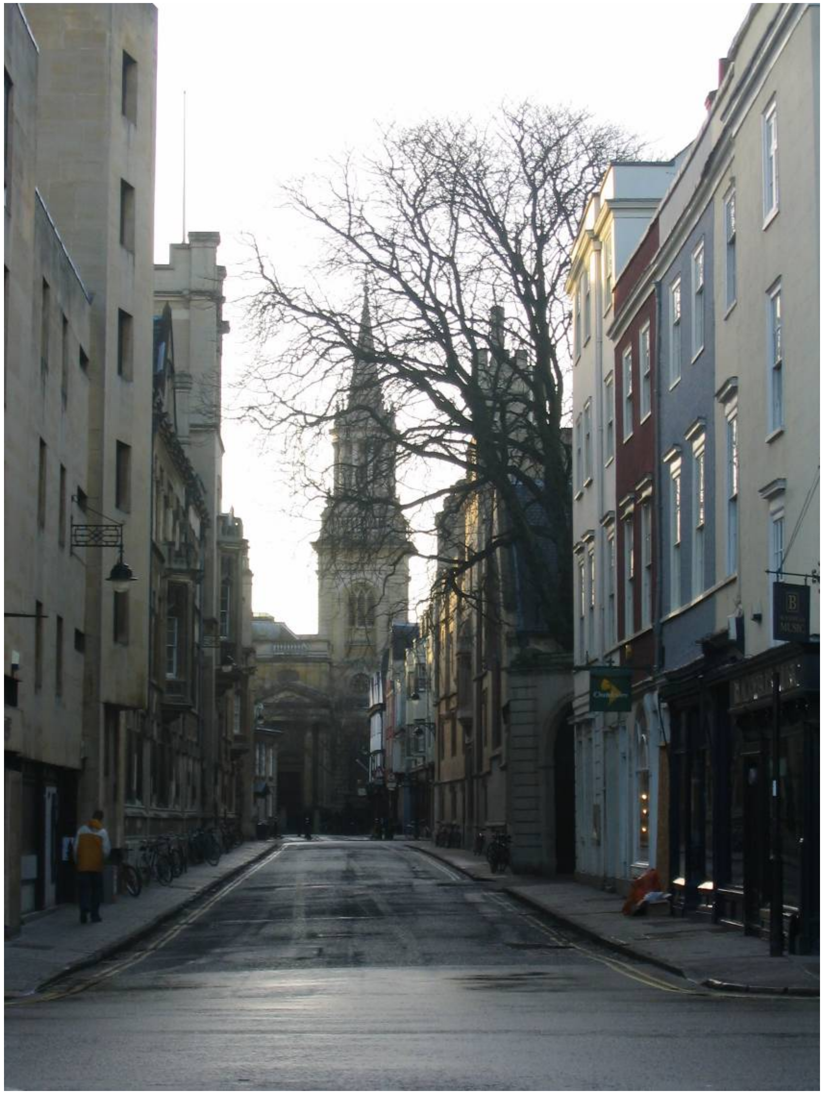


PP3 is a projection plane of both centers of projection, so we are OK!

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???

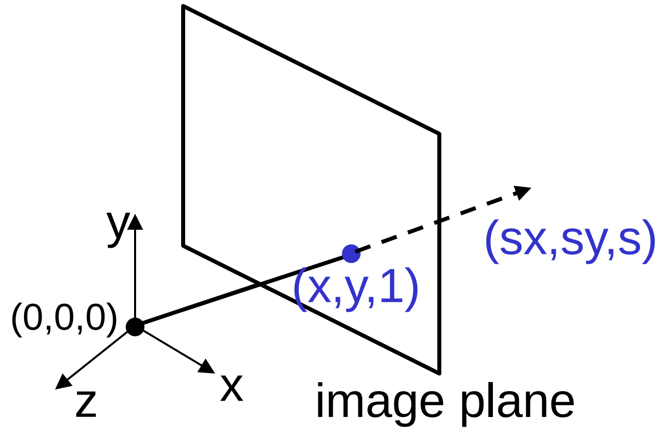
The projective plane

Why do we need homogeneous coordinates?

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

What is the geometric intuition?

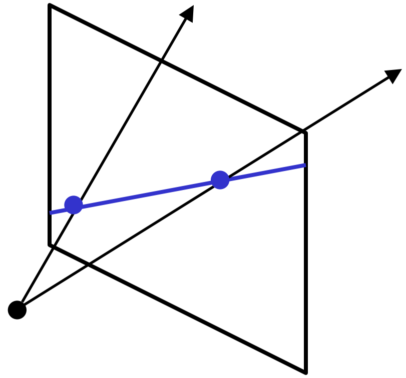
- a point in the image is a *ray* in projective space



- 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

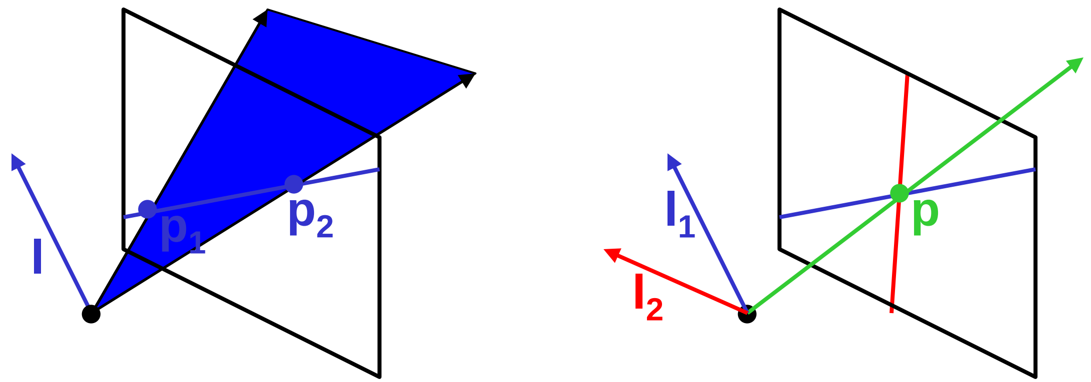
$$\text{in vector notation: } \quad 0 = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$\mathbf{l} \qquad \mathbf{p}$$

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

Point and line duality

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



What is the line **l** spanned by rays $\mathbf{p_1}$ and $\mathbf{p_2}$?

- **l** is $\perp$ to $\mathbf{p_1}$ and $\mathbf{p_2}$ $\Rightarrow$ $\mathbf{l} = \mathbf{p_1} \times \mathbf{p_2}$
- **l** is the plane normal

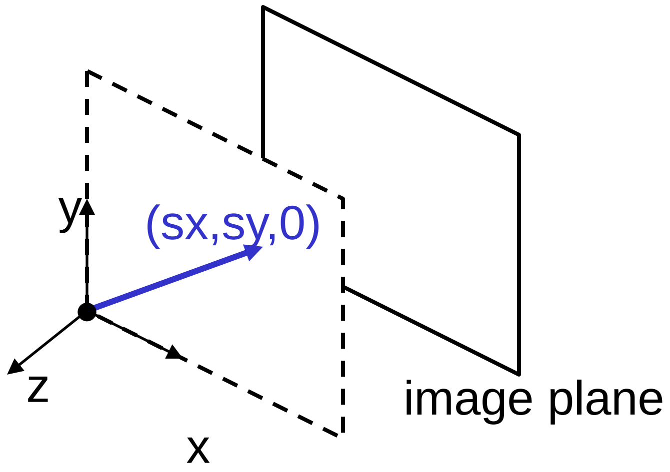
What is the intersection of two lines $\mathbf{l_1}$ and $\mathbf{l_2}$?

- **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



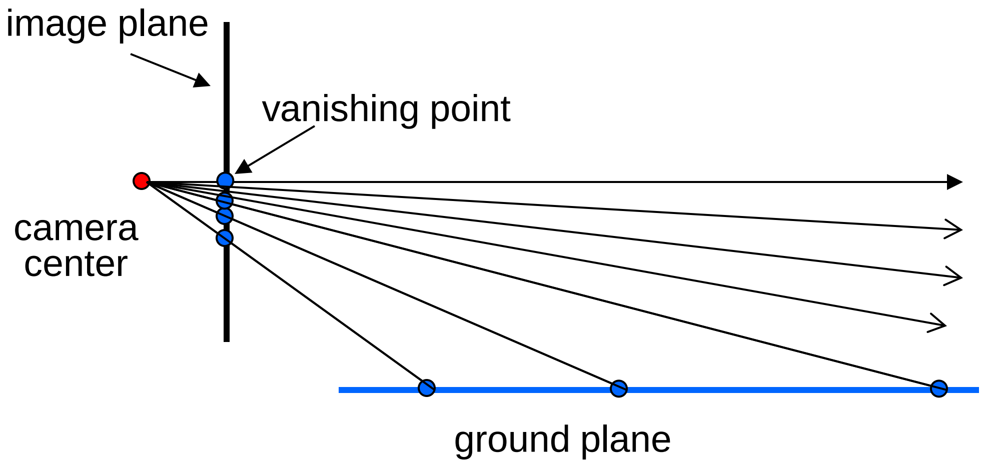
Ideal point ("point at infinity")

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

Ideal line

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

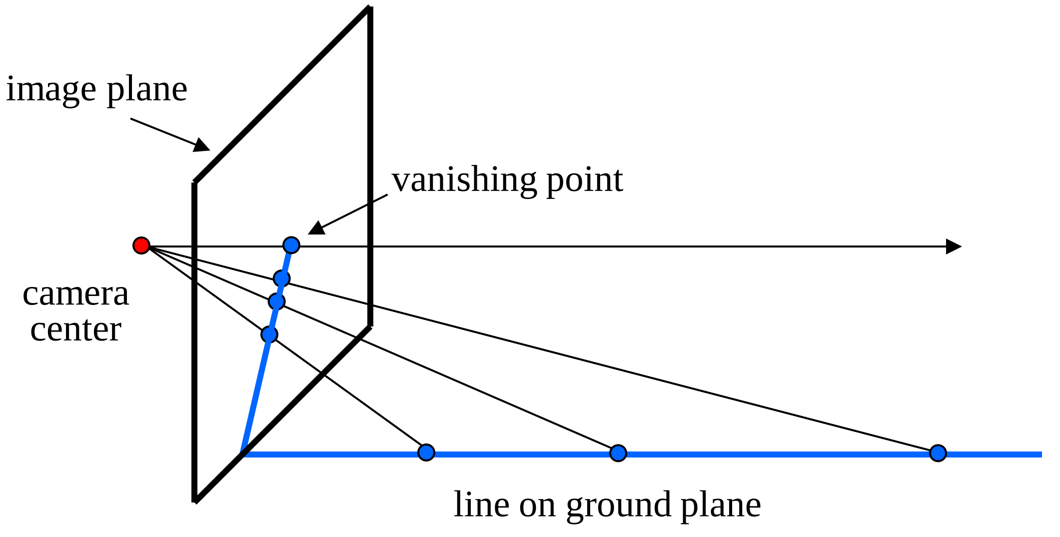
Vanishing points



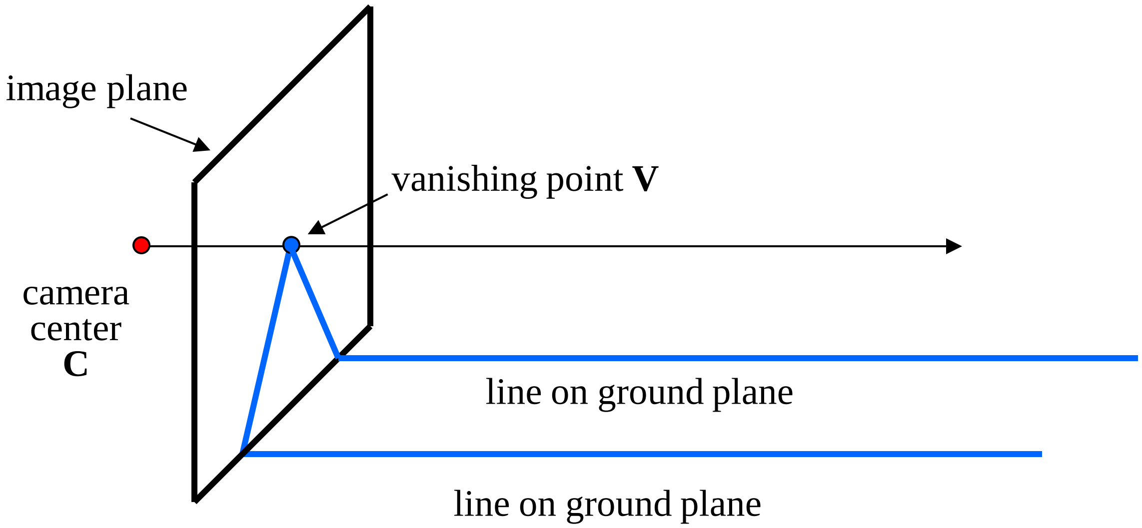
Vanishing point

- projection of a point at infinity

Vanishing points (2D)



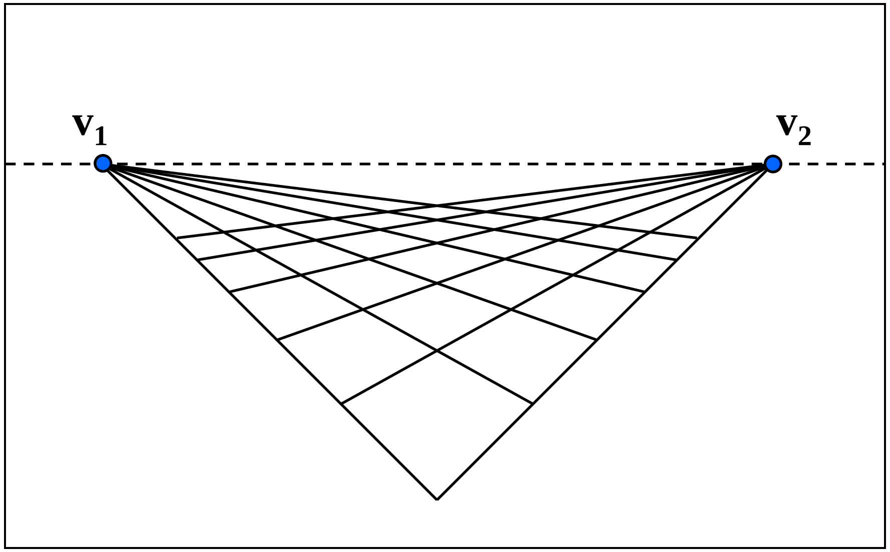
Vanishing points



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

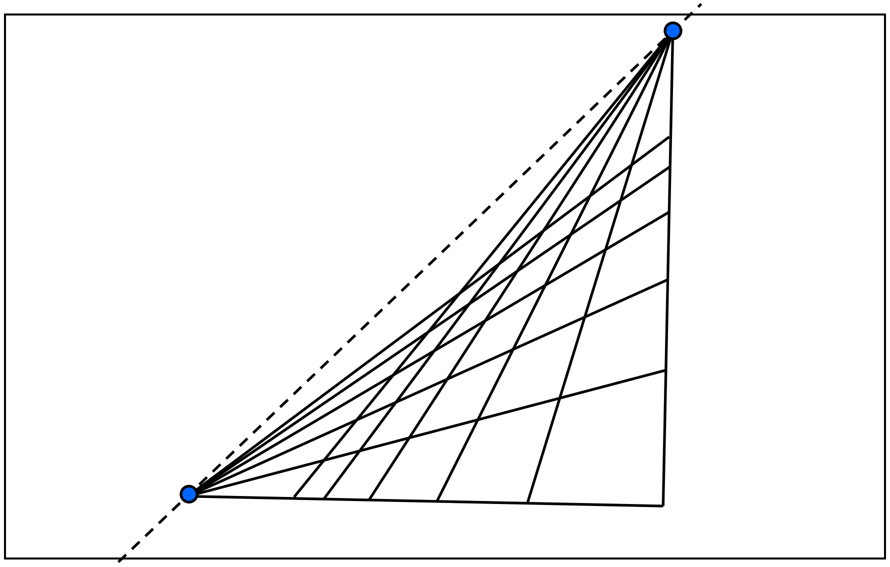
Vanishing lines



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

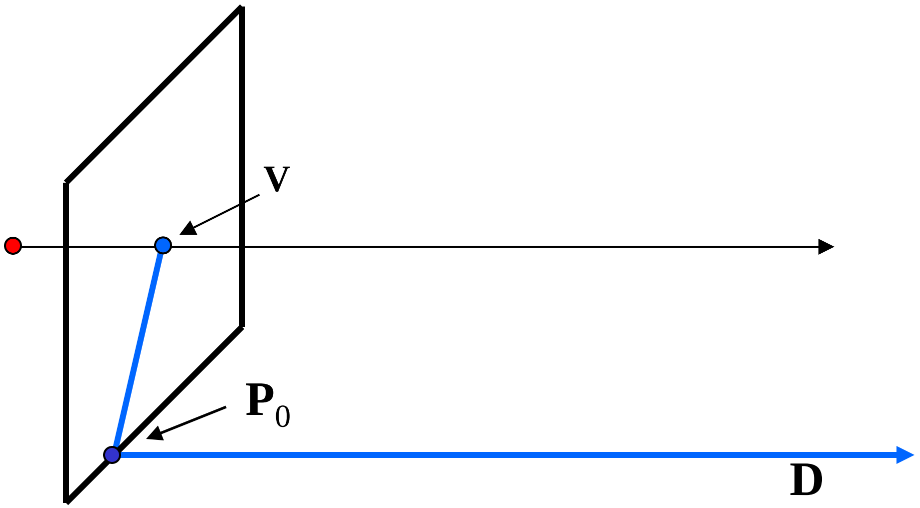
Vanishing lines



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



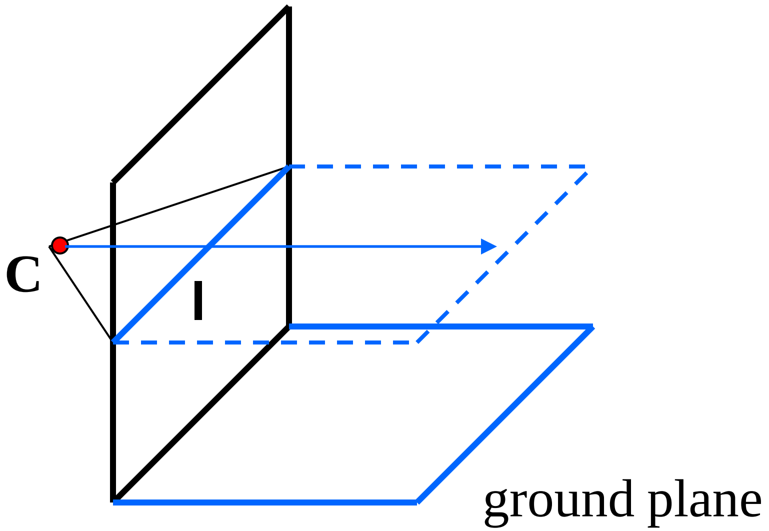
$$\mathbf{P} = \mathbf{P}_0 + t\mathbf{D}$$

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

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

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

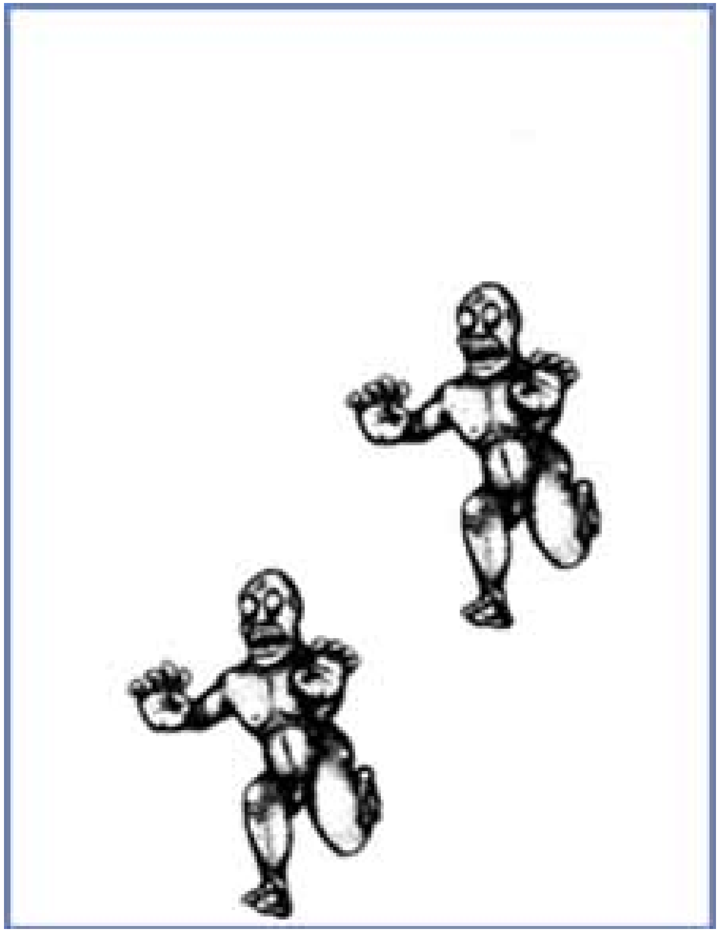
Computing vanishing lines

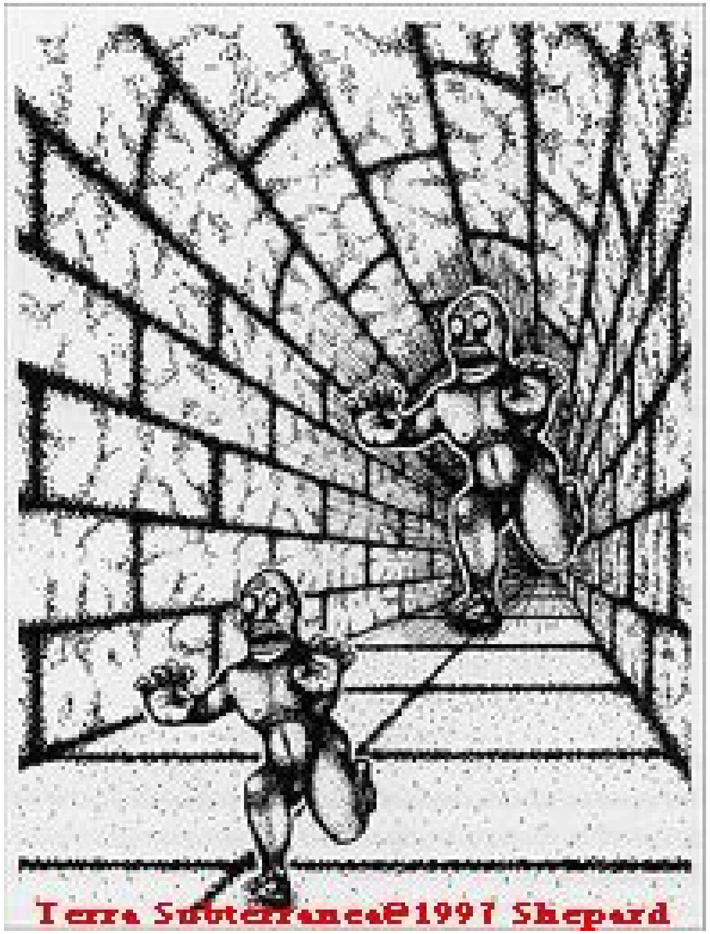


Properties

- **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

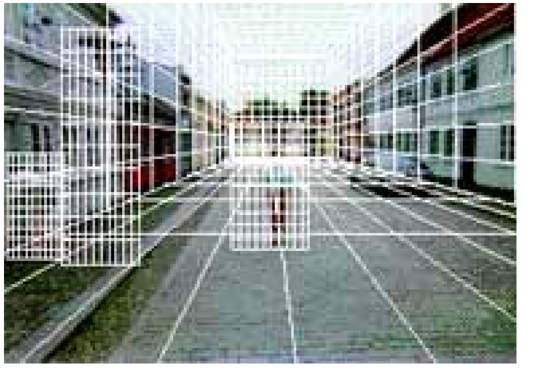




“Tour into the Picture” (SIGGRAPH ’97)

Create a 3D “theatre stage” of five billboards

Specify foreground objects through bounding polygons



Use camera transformations to navigate through the scene

The idea

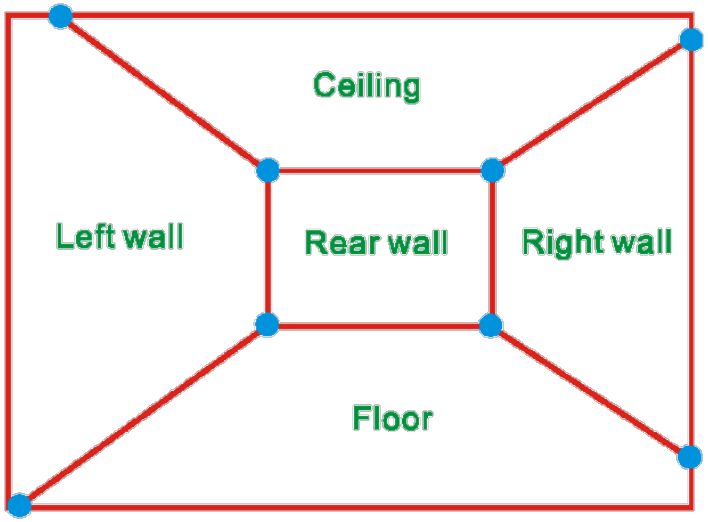
Many scenes (especially paintings), can be represented as an axis-aligned box volume (i.e. a stage)

Key assumptions:

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

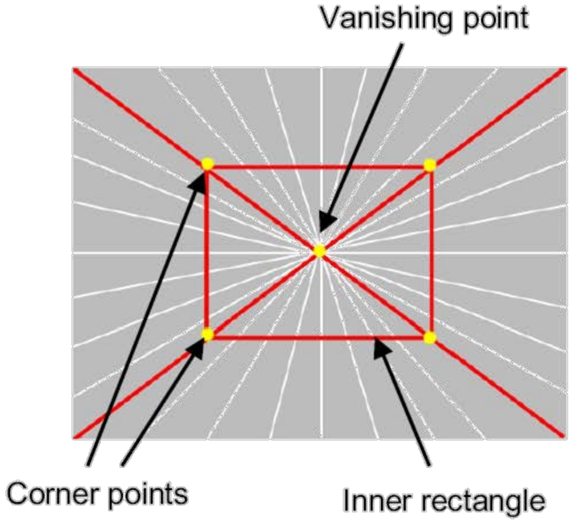
How many vanishing points does the box have?

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



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

Fitting the box volume

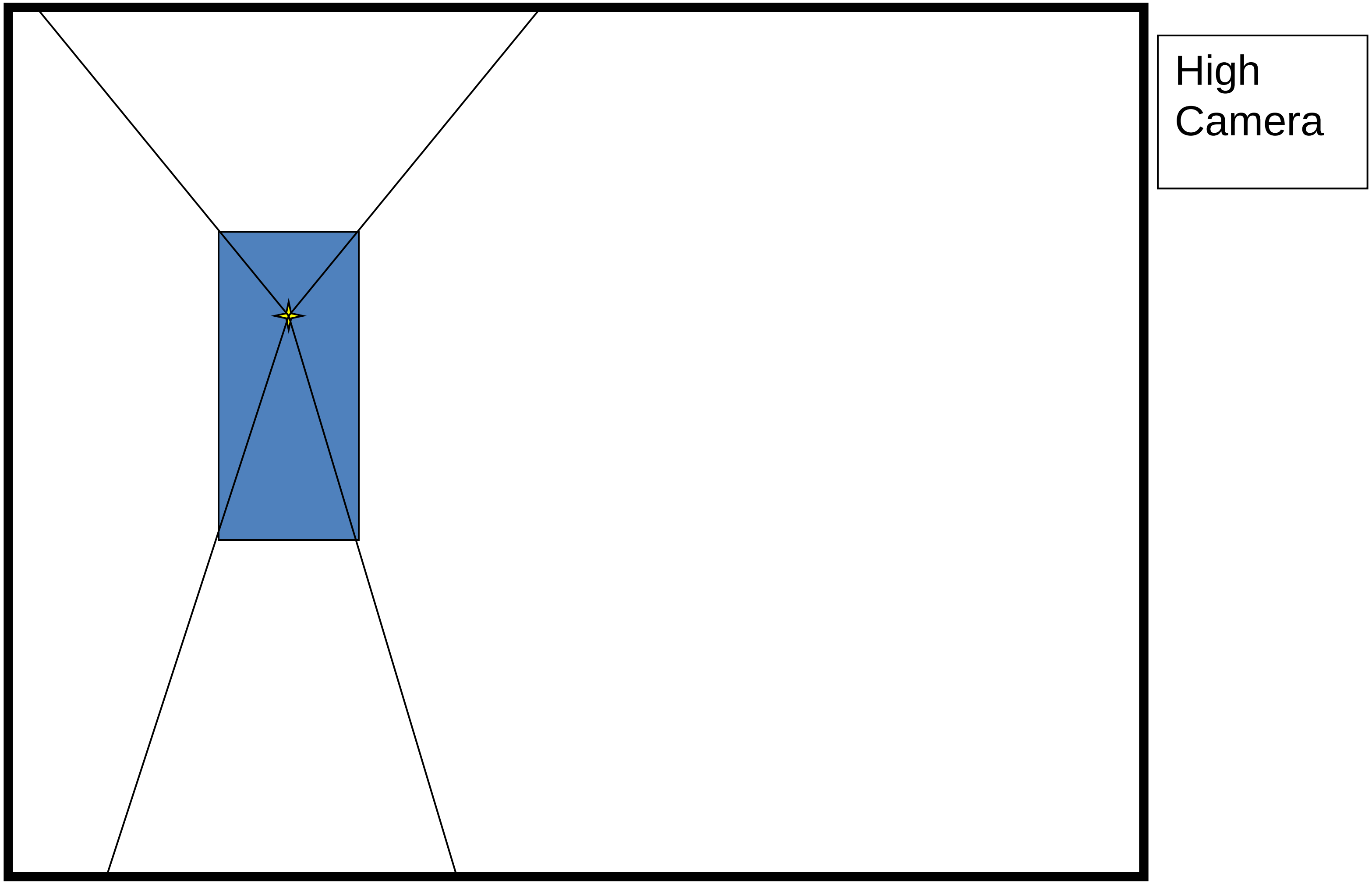


User controls the inner box and the vanishing point placement (# of DOF???)

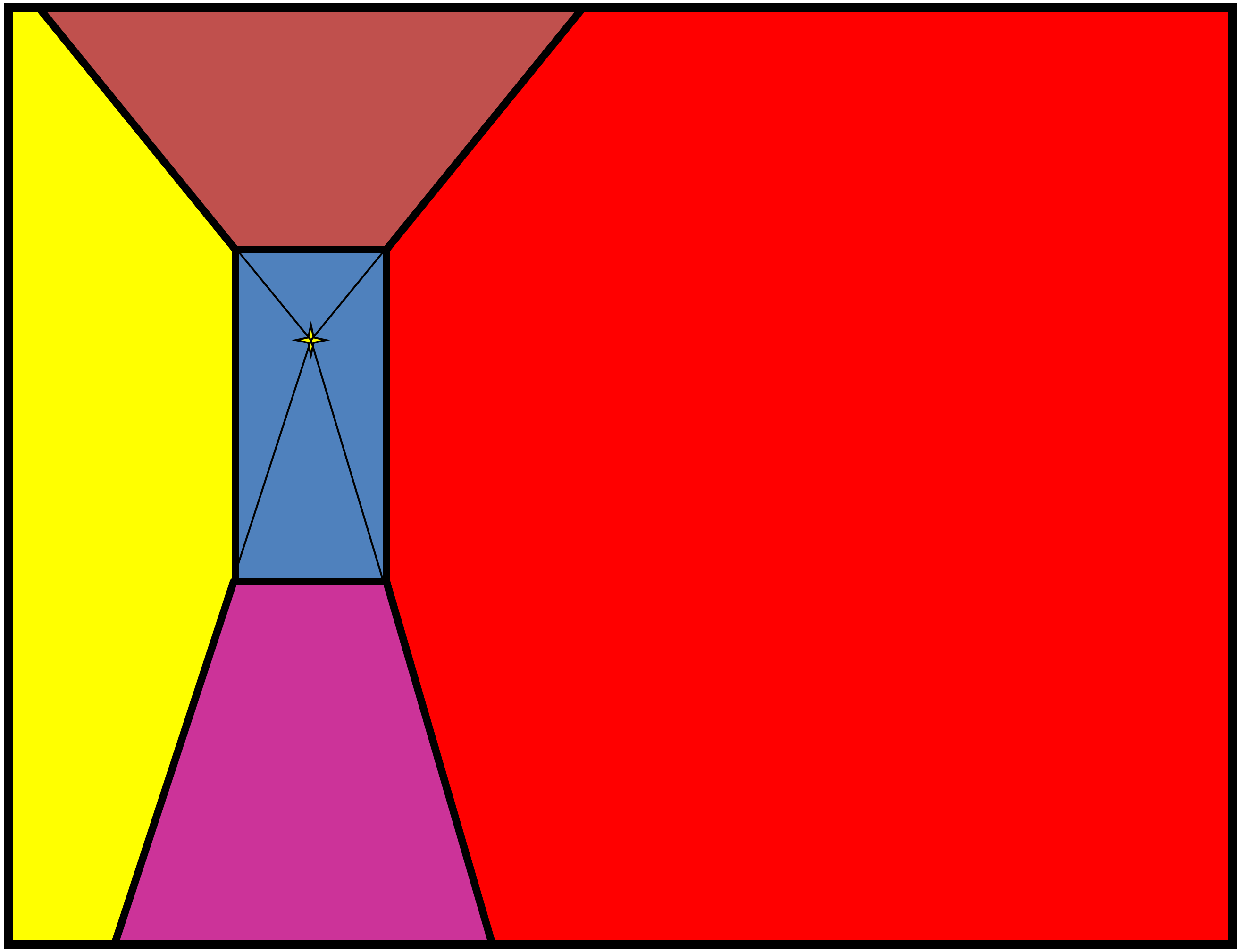
Q: What's the significance of the vanishing point location?

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

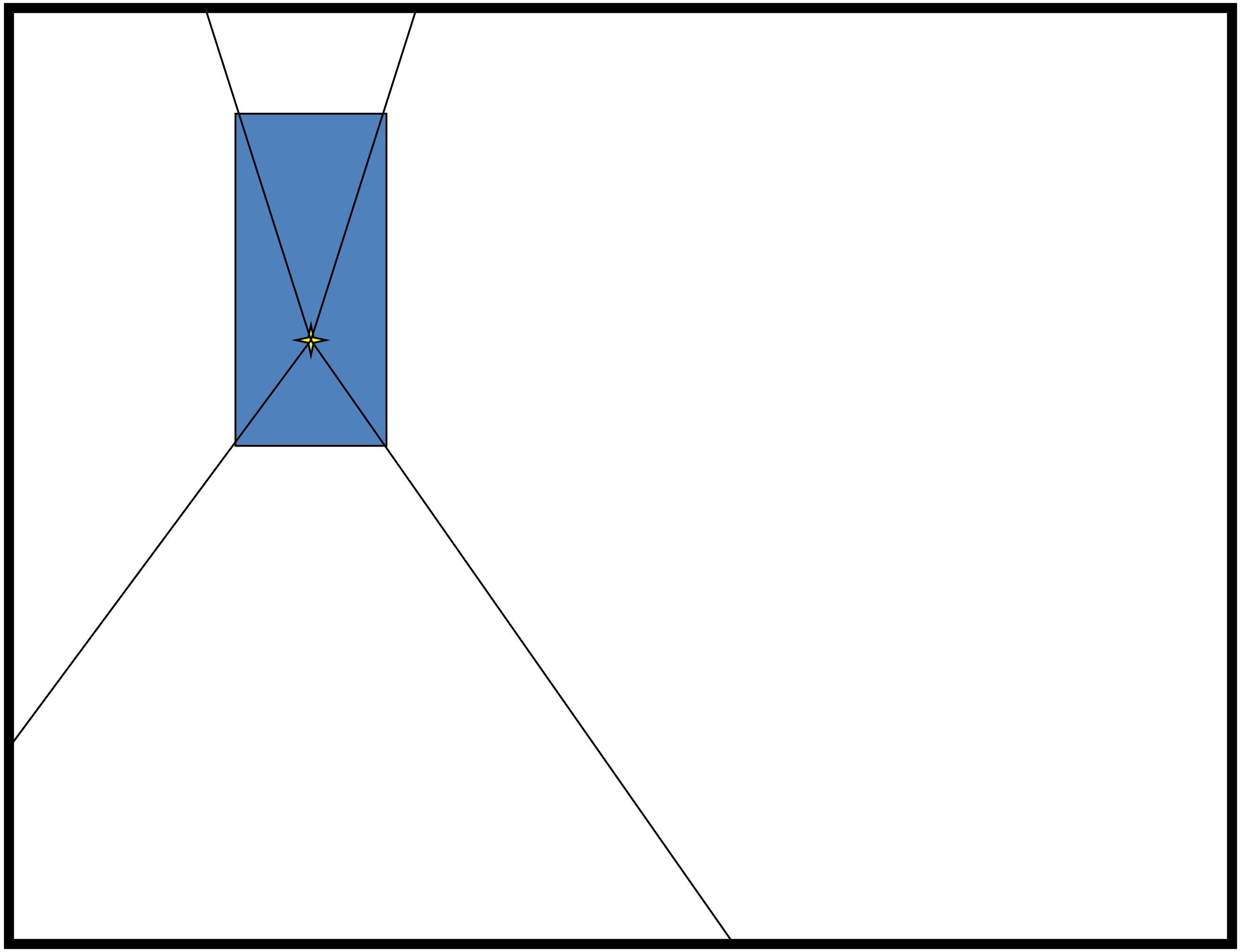


Example of user input: vanishing point and back face of view volume are defined



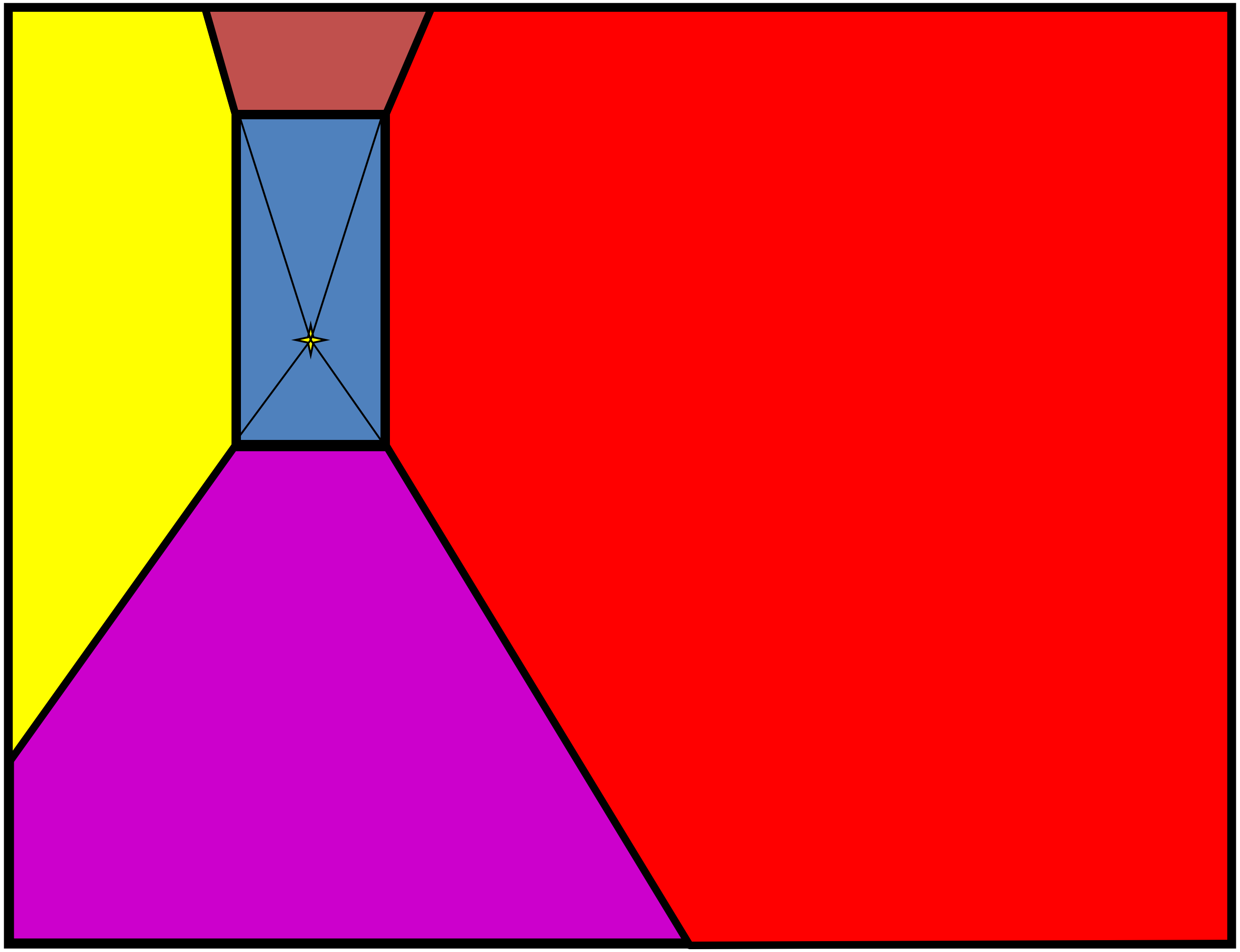
High Camera

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



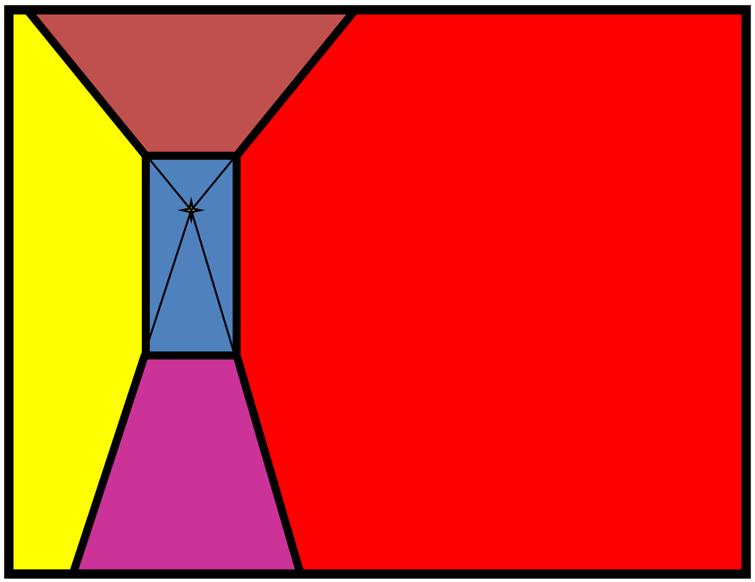
Low Camera

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

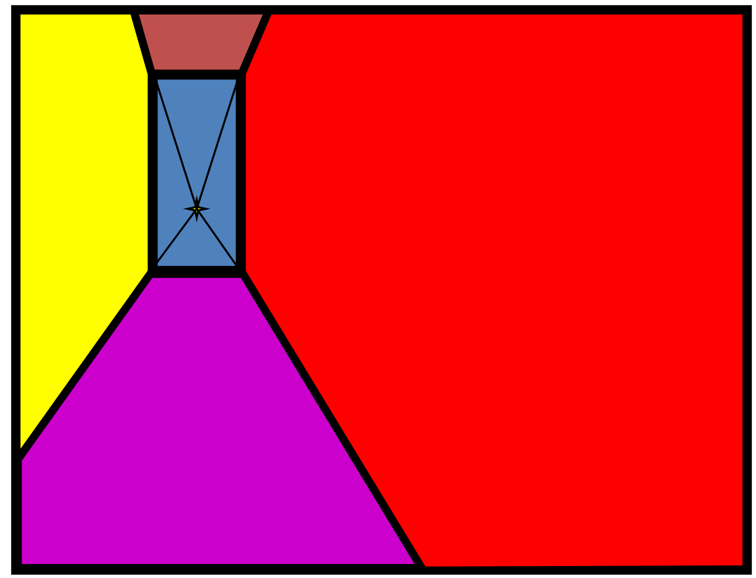


Low Camera

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.

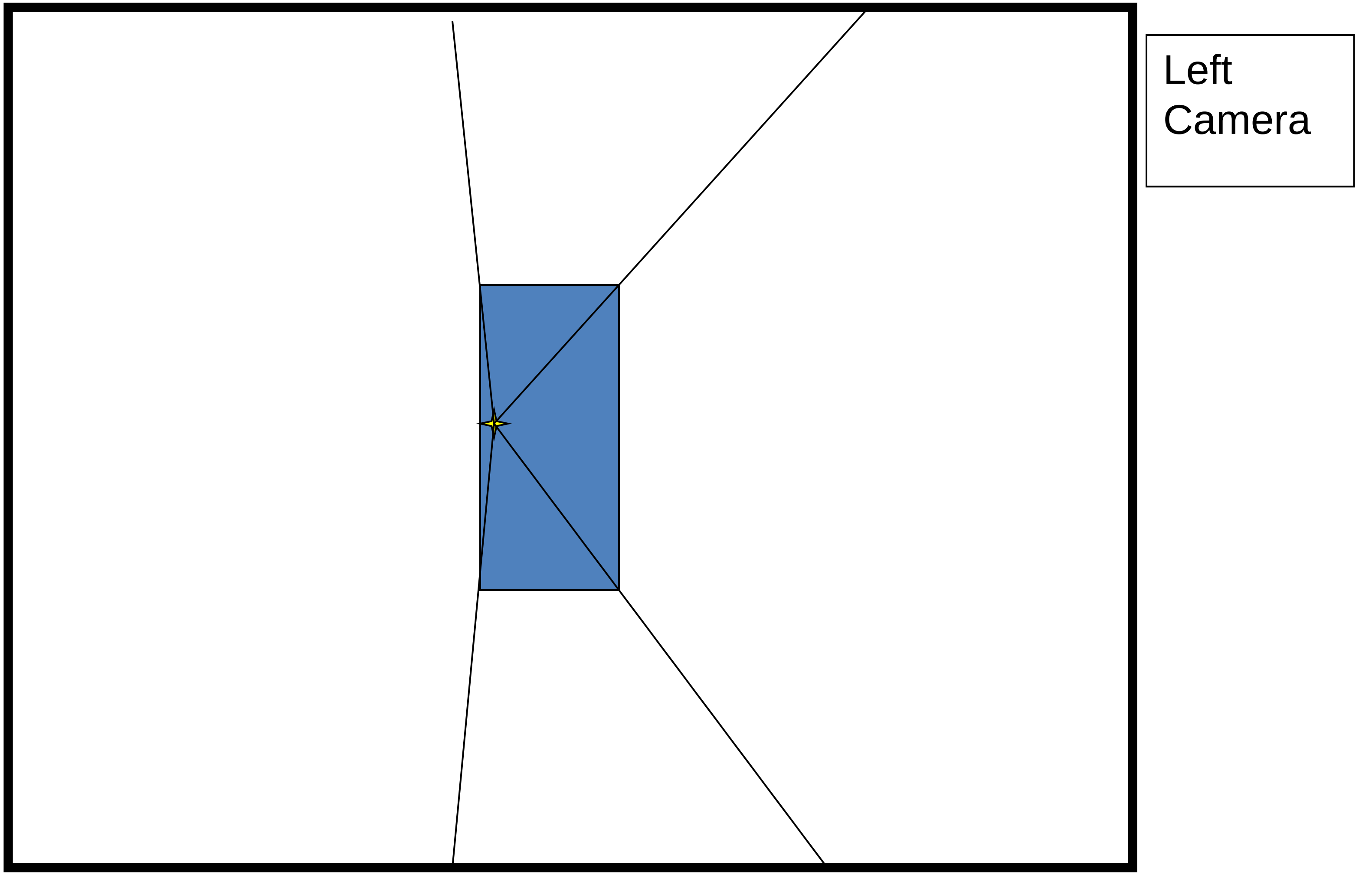


High Camera

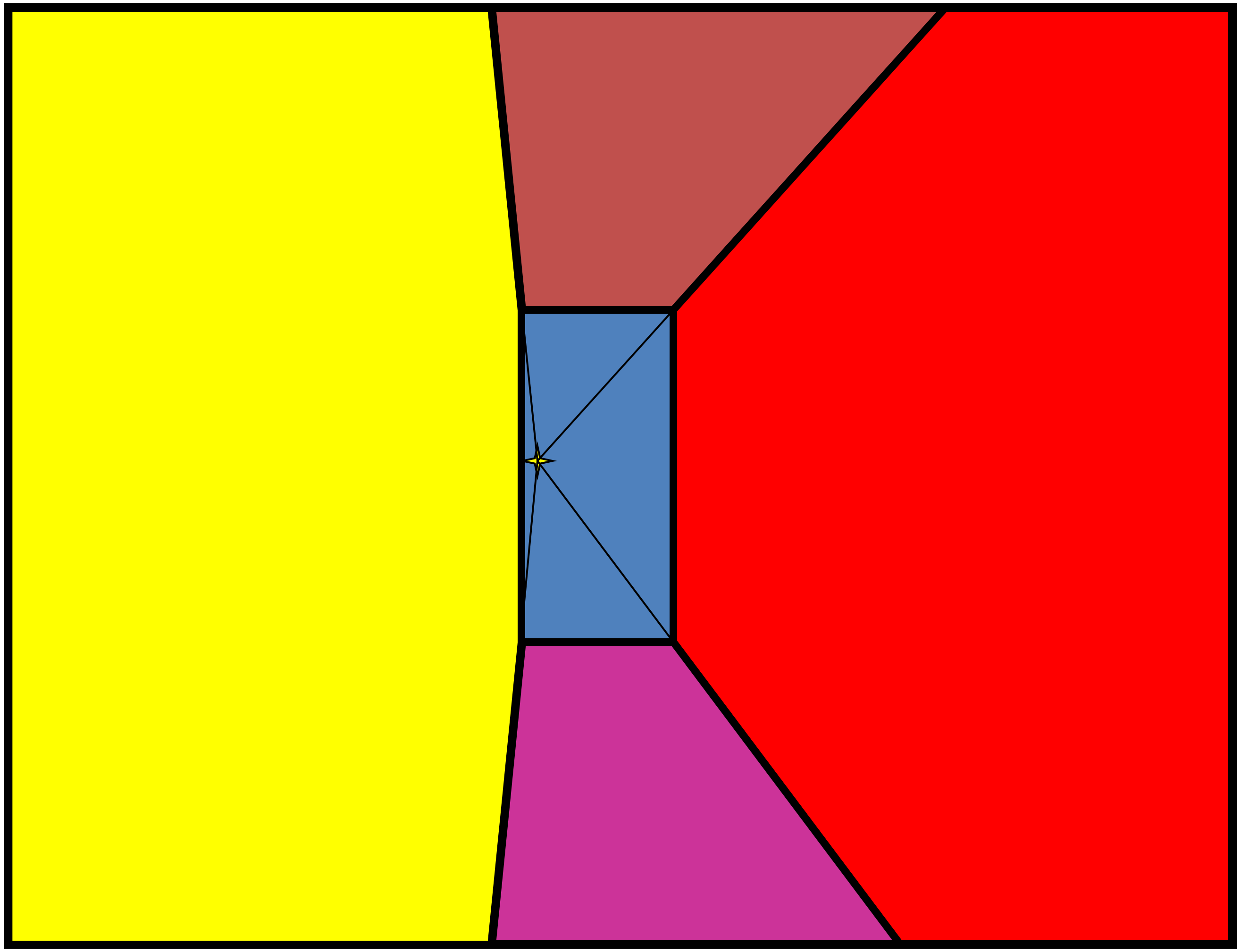


Low Camera

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

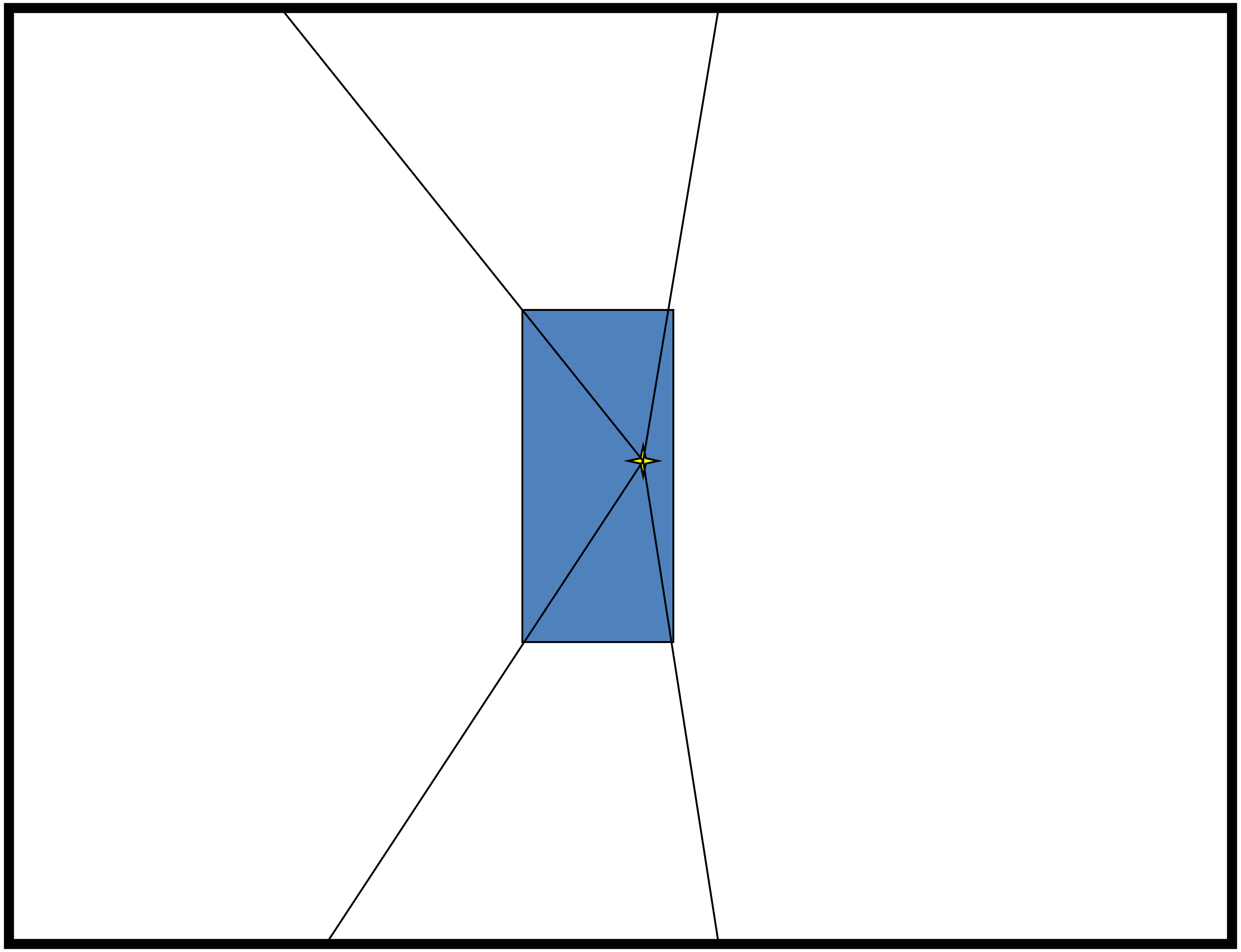


Another example of user input: vanishing point and back face of view volume are defined



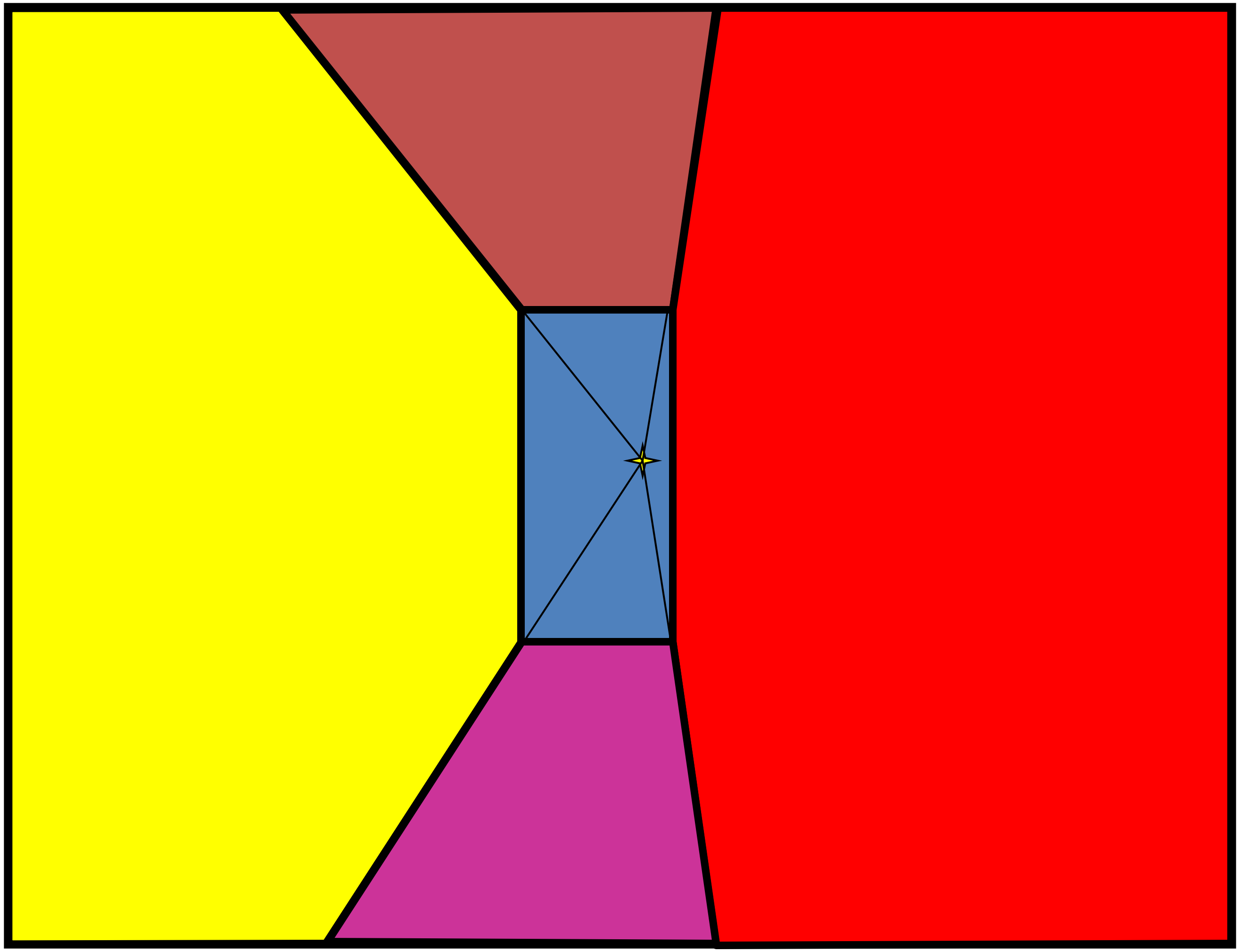
Left Camera

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



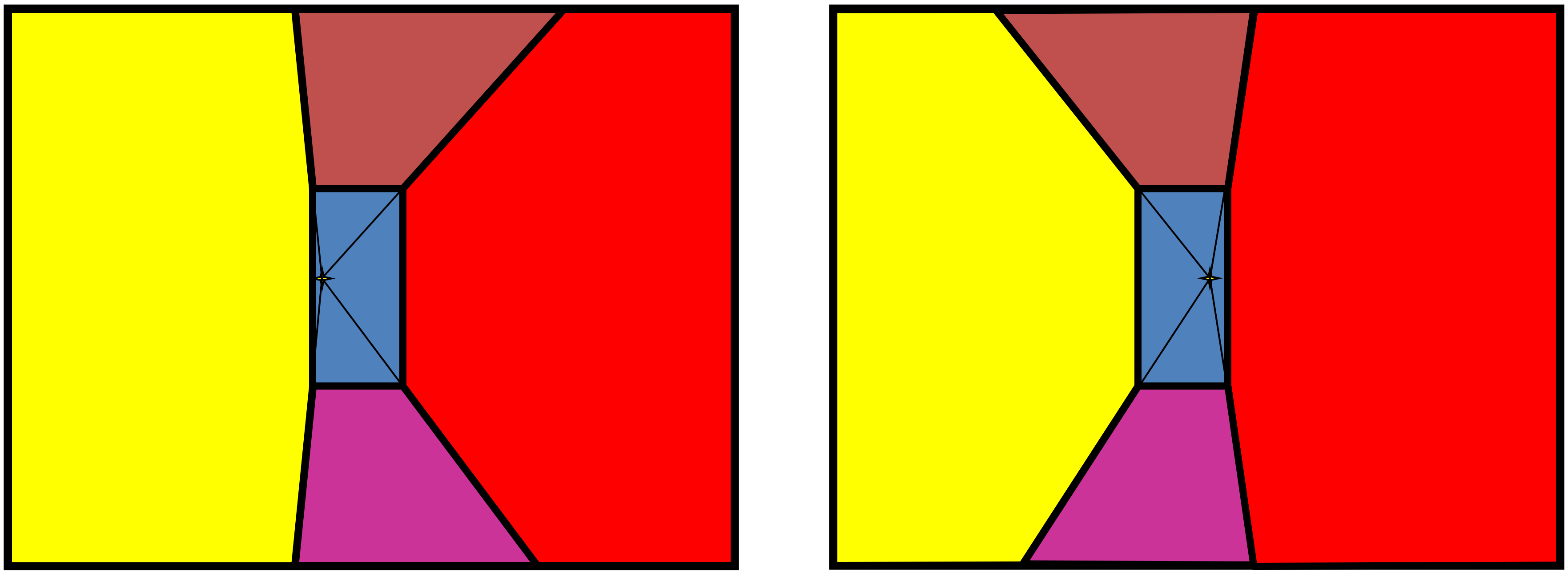
Right Camera

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



Right Camera

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

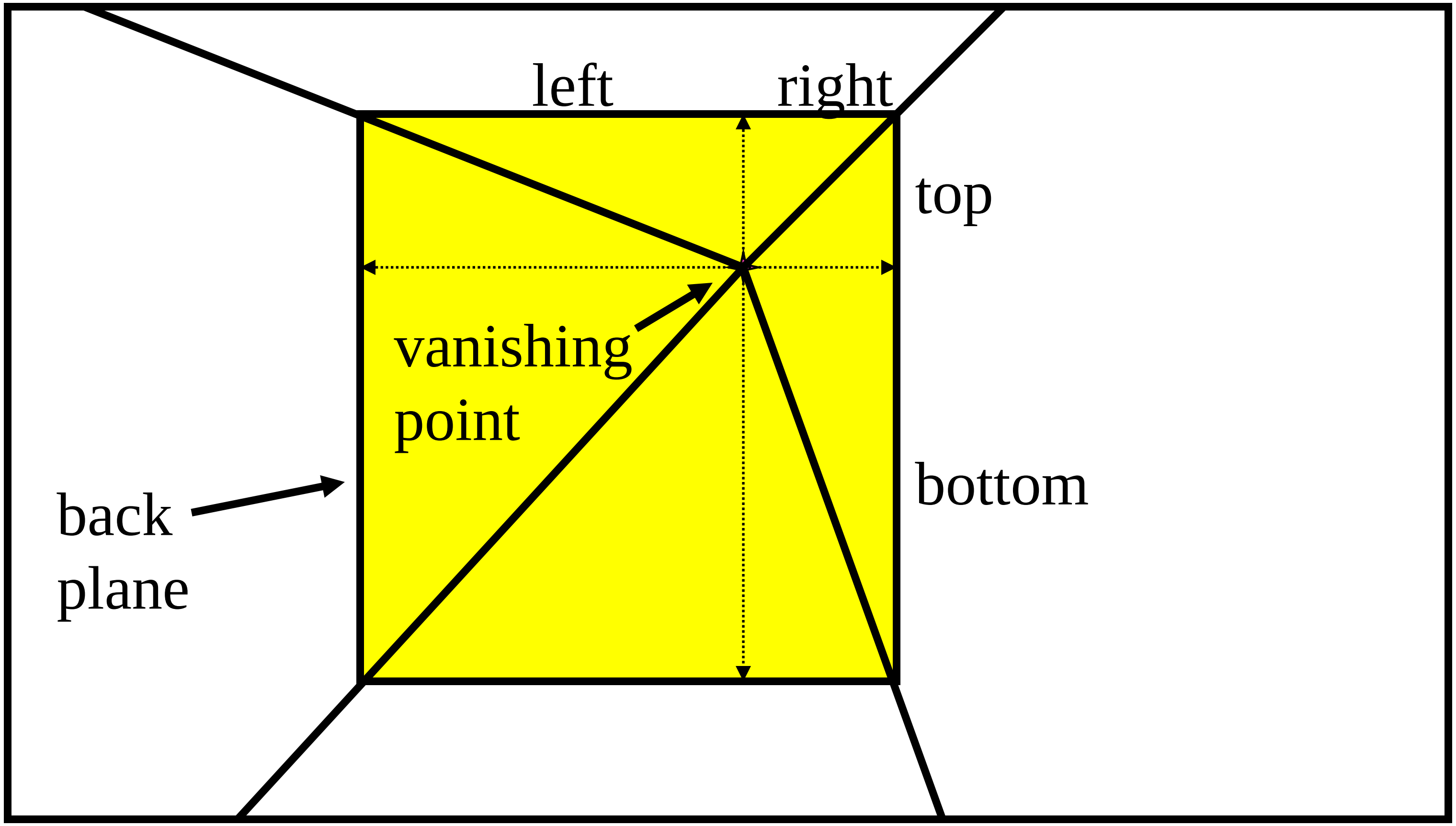


Left Camera

Right Camera

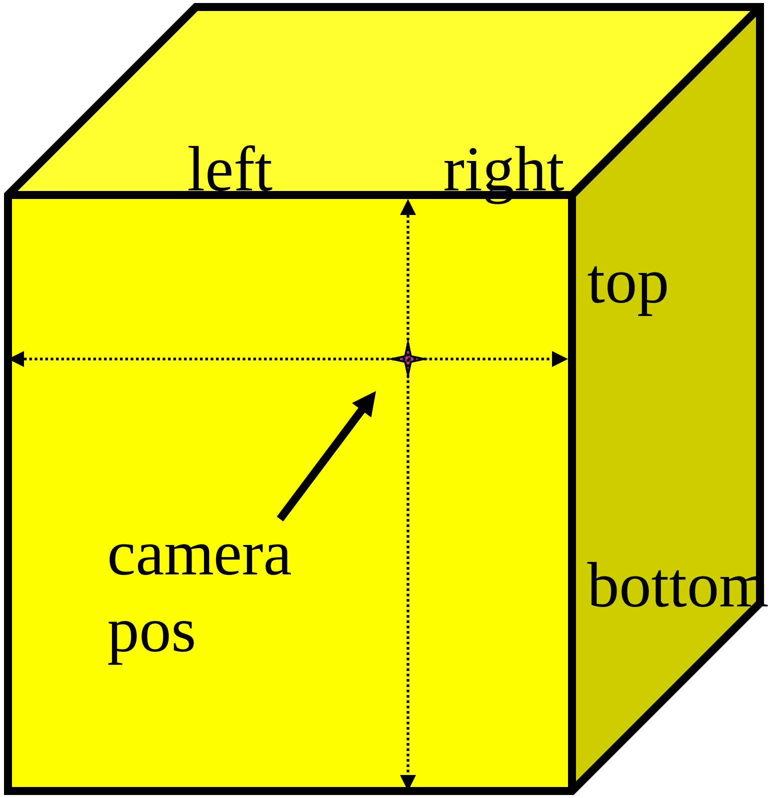
2D to 3D conversion

First, we can get ratios

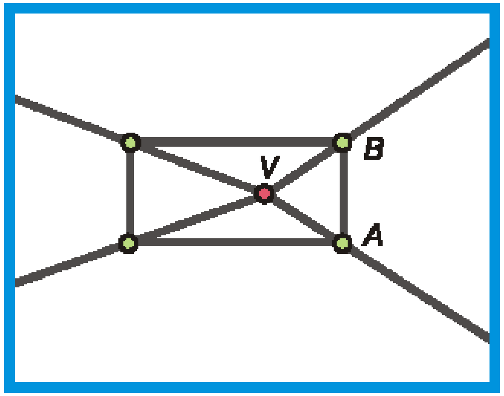


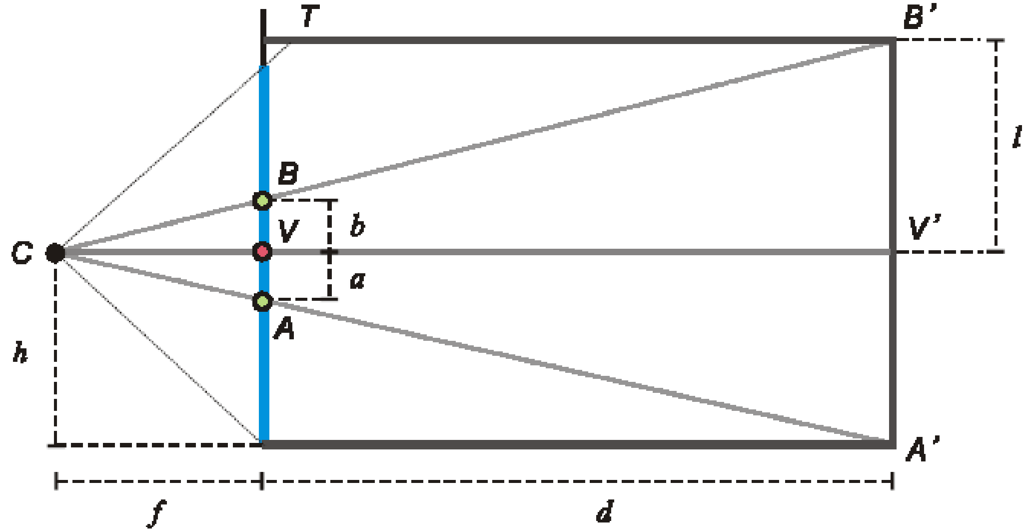
2D to 3D conversion

- 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)

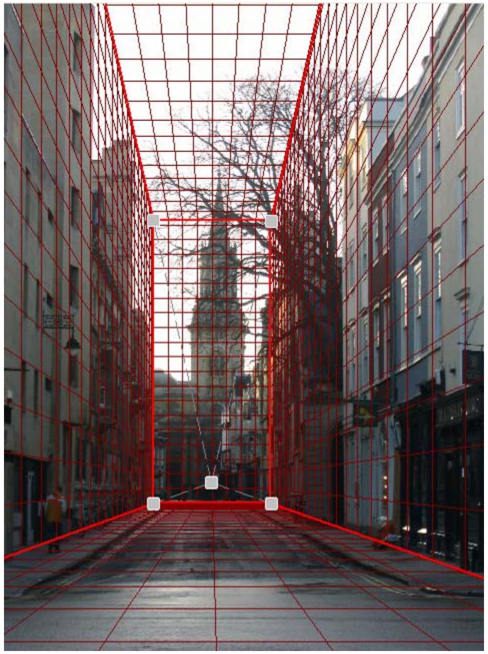
Note: can compute position on any object on the ground

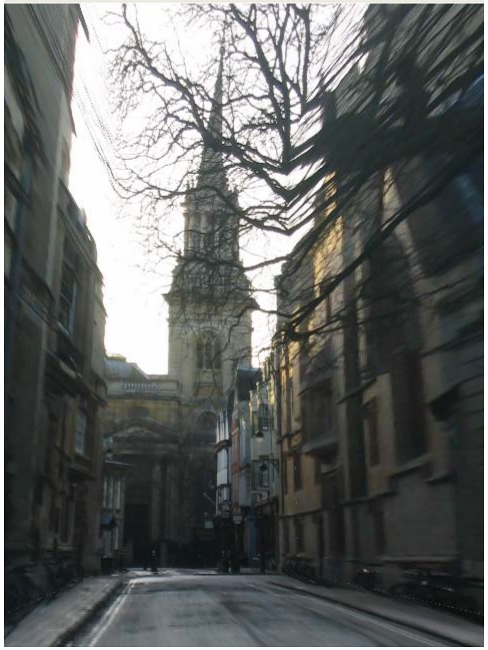
- Simple unprojection
- What about things off the ground?

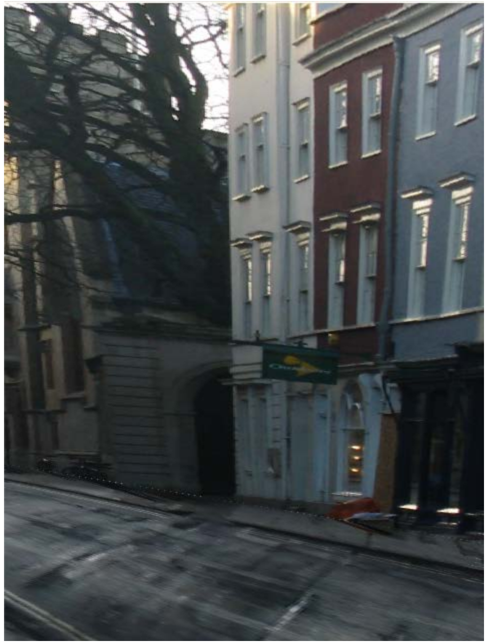
DEMO

Now, we know the 3D geometry of the box

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





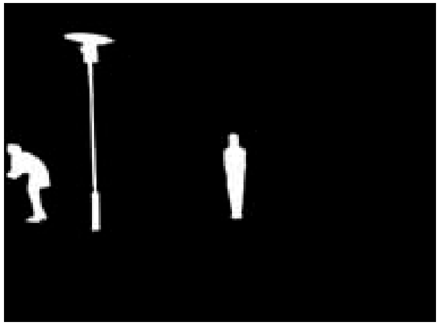


Foreground Objects

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



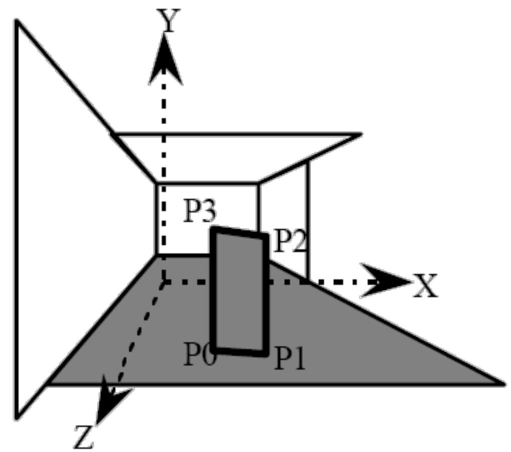
Foreground Objects

Add vertical rectangles for each foreground object

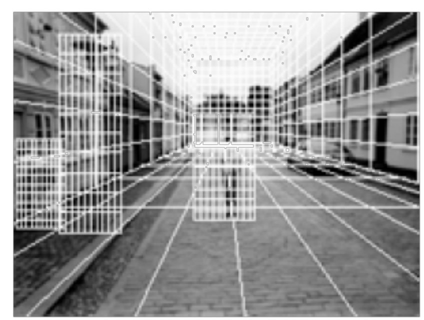
Can compute 3D coordinates P0, P1 since they are on known plane.

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