Single-view 3D Reconstruction

Computational Photography

Yuxiong Wang, University of Illinois

Slides adopted from Derek Hoiem

Some slides from Alyosha Efros, Steve Seitz

Take-home question

Suppose you have estimated finite three vanishing points corresponding to orthogonal directions:

1) How to solve for intrinsic matrix? (assume K has three parameters)
   - The transpose of the rotation matrix is its inverse
   - Use the fact that the 3D directions are orthogonal
2) How to recover the rotation matrix that is aligned with the 3D axes defined by these points?
   - In homogeneous coordinates, 3d point at infinity is (X, Y, Z, 0)

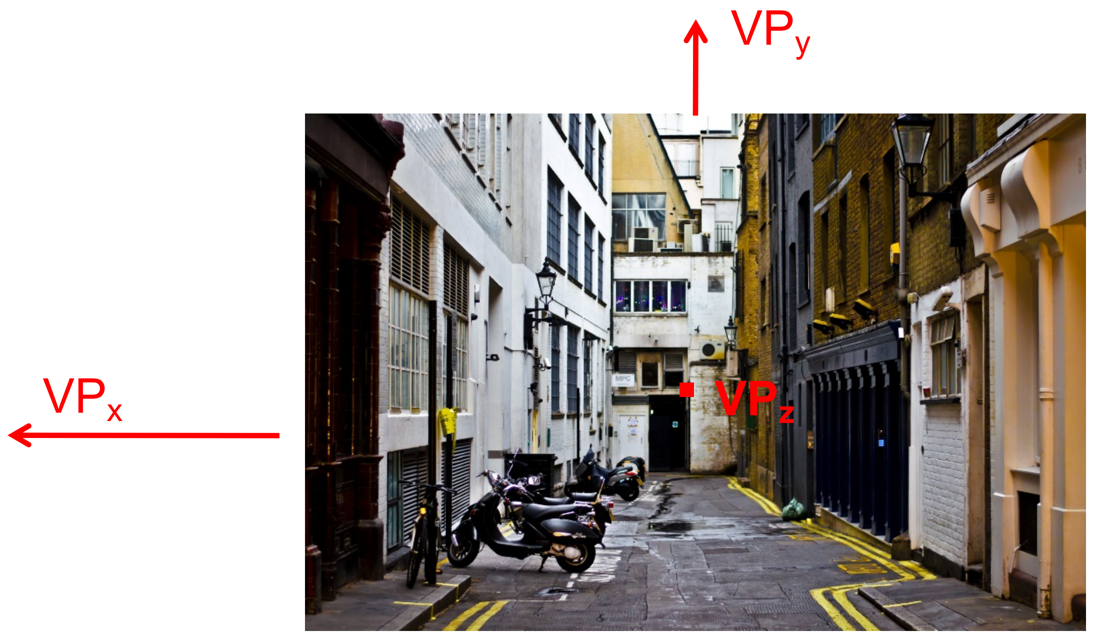
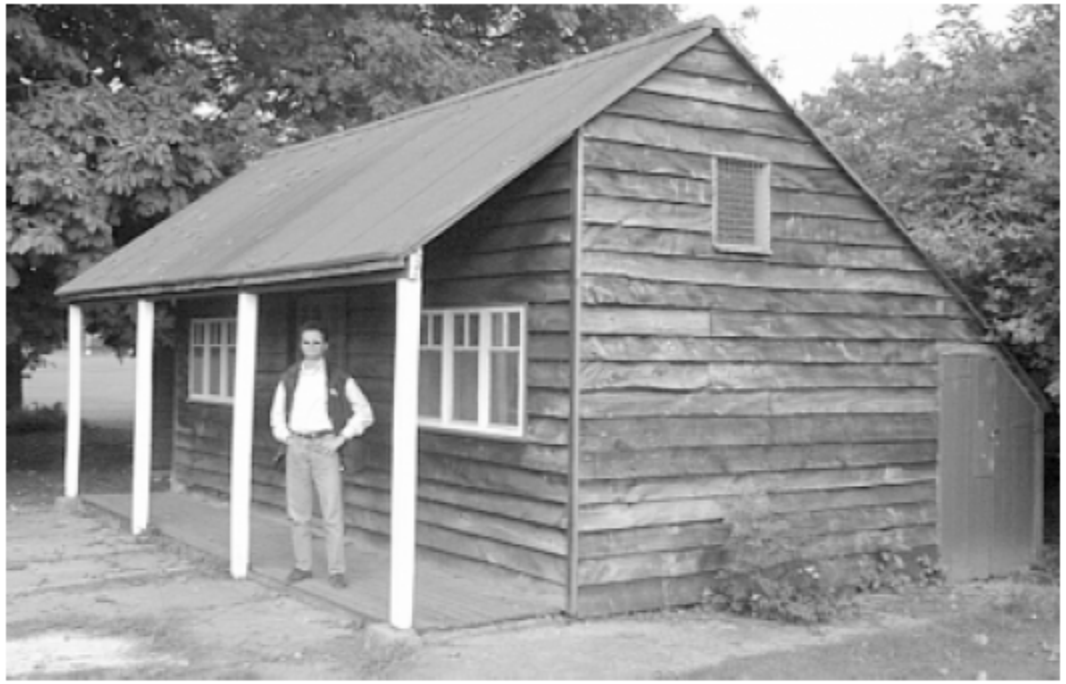


Photo from Garry Knight

Take-home question

Assume that the man is 6 ft tall.

- What is the height of the front of the building?
- What is the height of the camera?



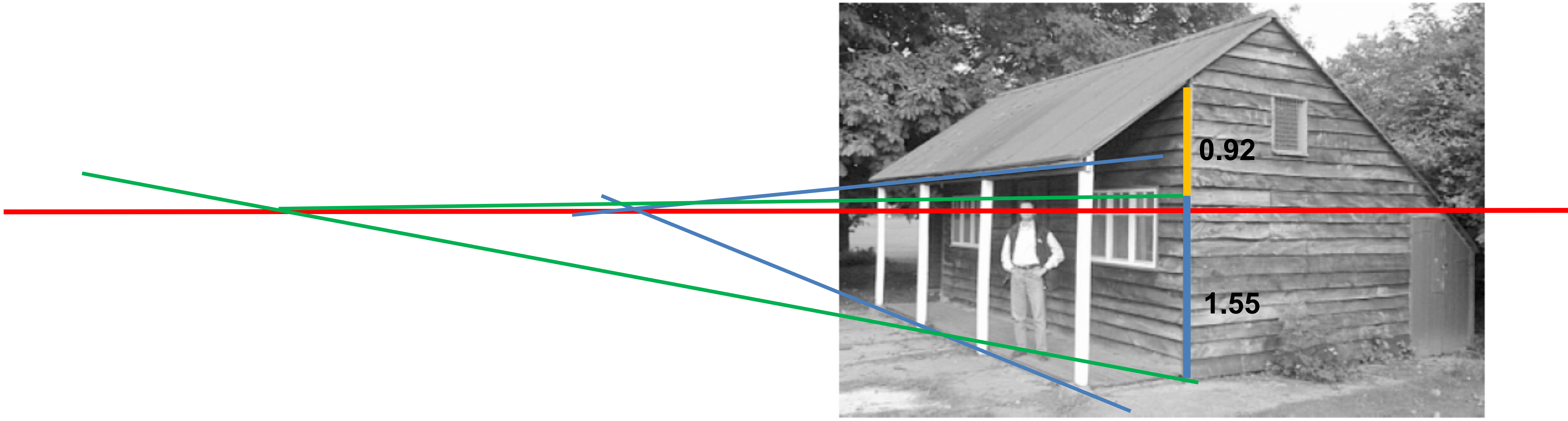
Take-home question

Assume that the man is 6 ft tall.

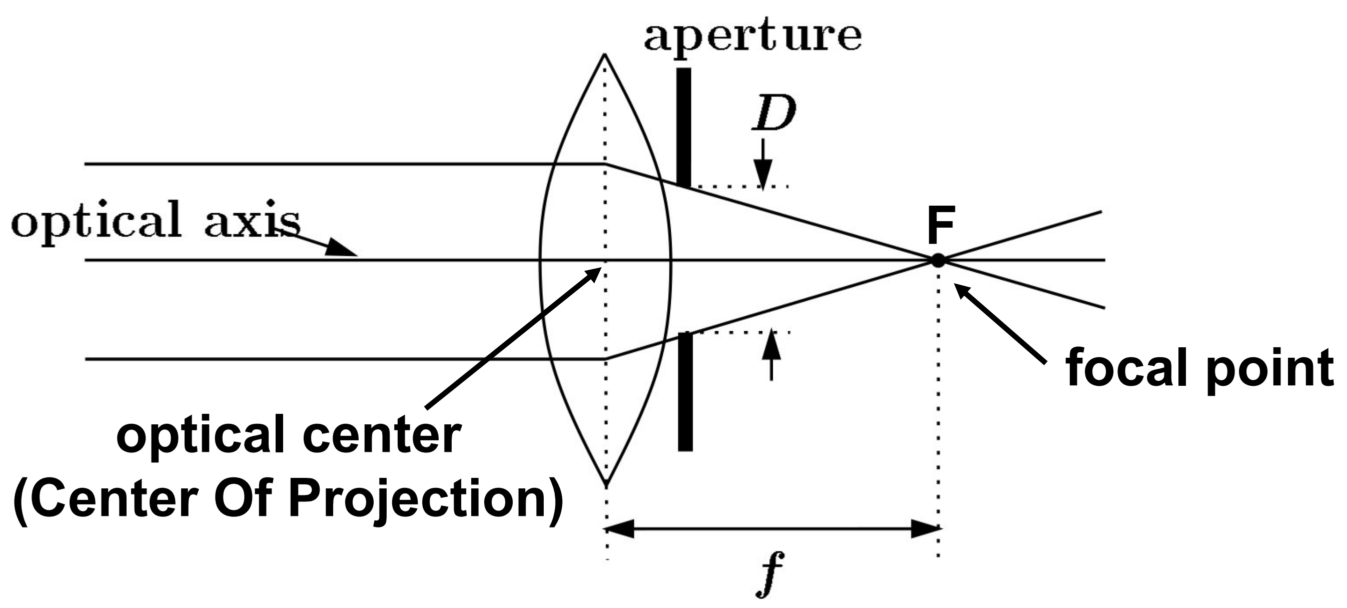
(0.92+1.55)/1.55*6=9.56

- What is the height of the front of the building?
- What is the height of the camera?

~5'7



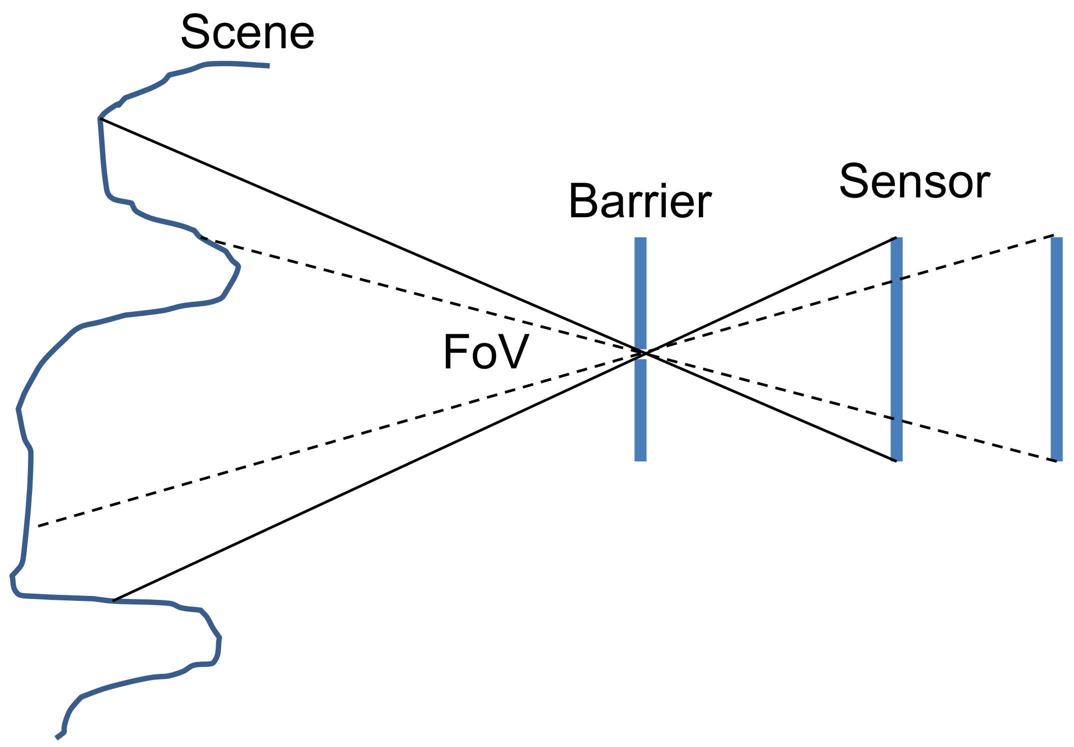
Focal length, aperture, depth of field



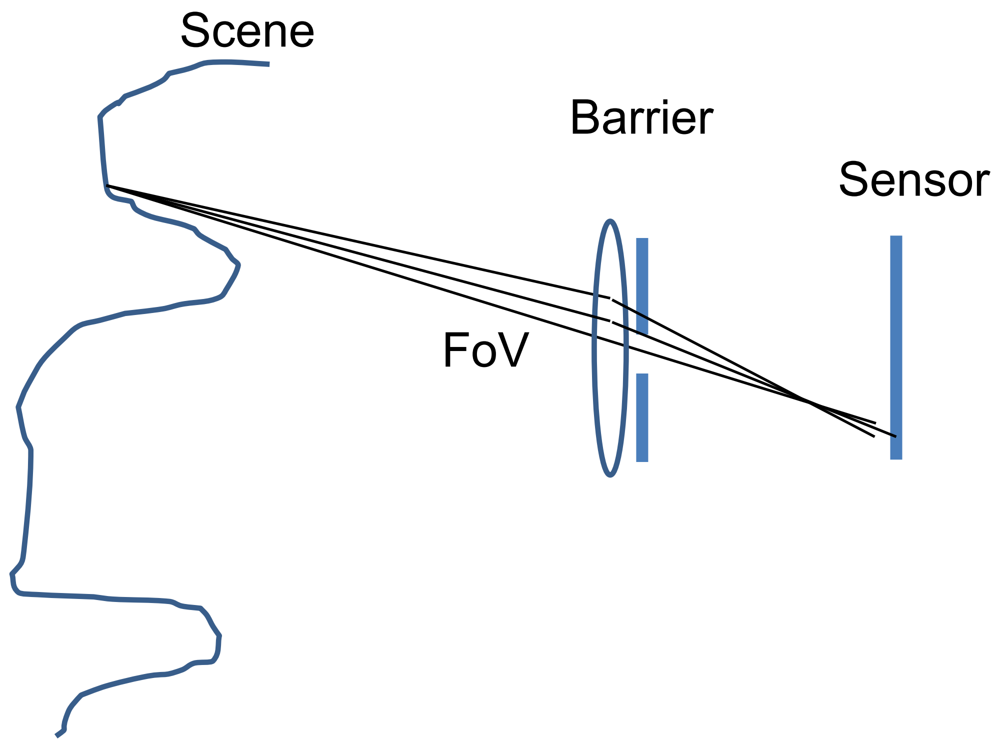
- Increase in focal length "zooms in", decreasing field of view (and light per pixel), increasing depth of field (less blur)

- Increase in aperture lets more light in but decreases depth of field

Increasing focal length decreases field of view because smaller range of rays to scene can hit sensor



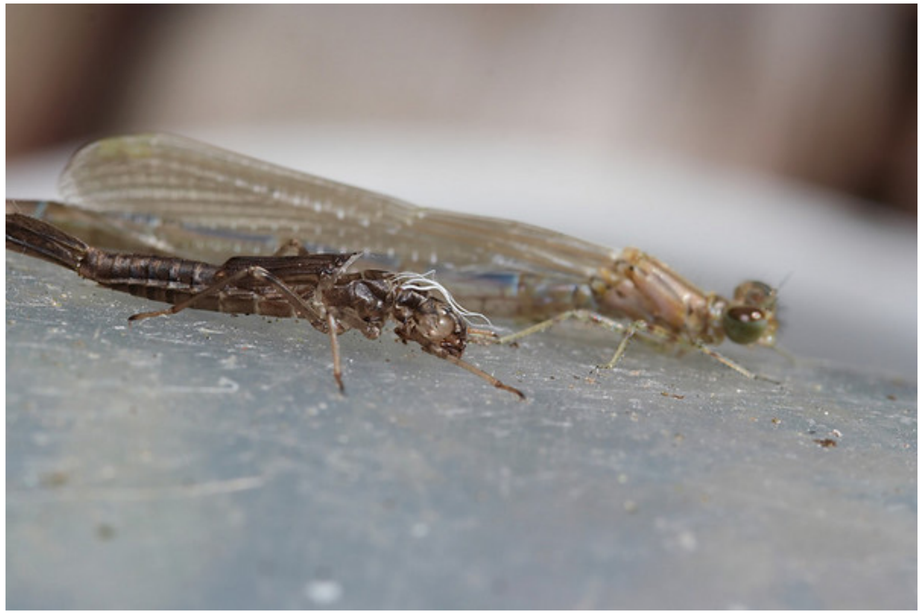
Decreasing aperture increases depth of field because lens refocuses rays from smaller range of angles

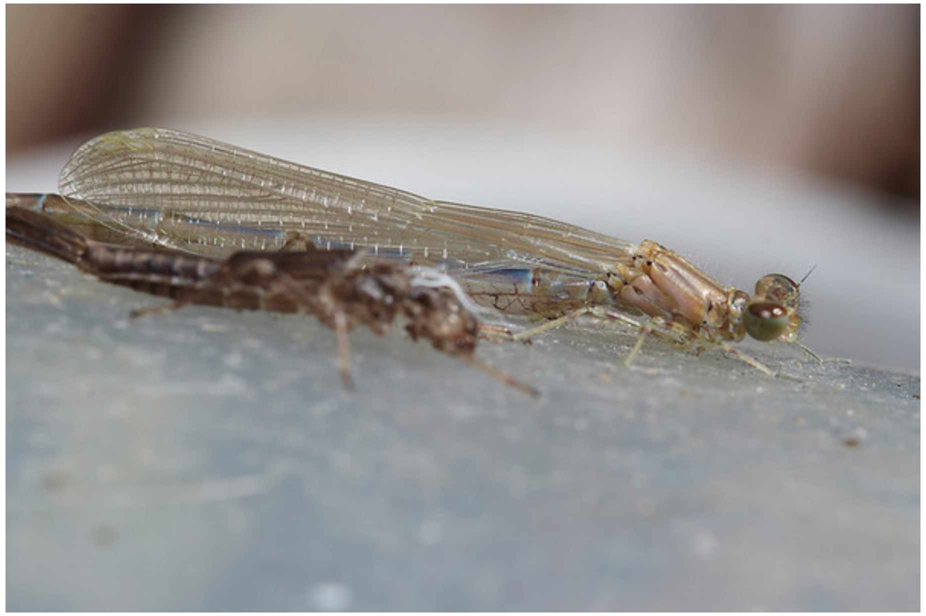


Difficulty in macro (close-up) photography

- For close objects, we have a small relative DOF
- Can only shrink aperture so far

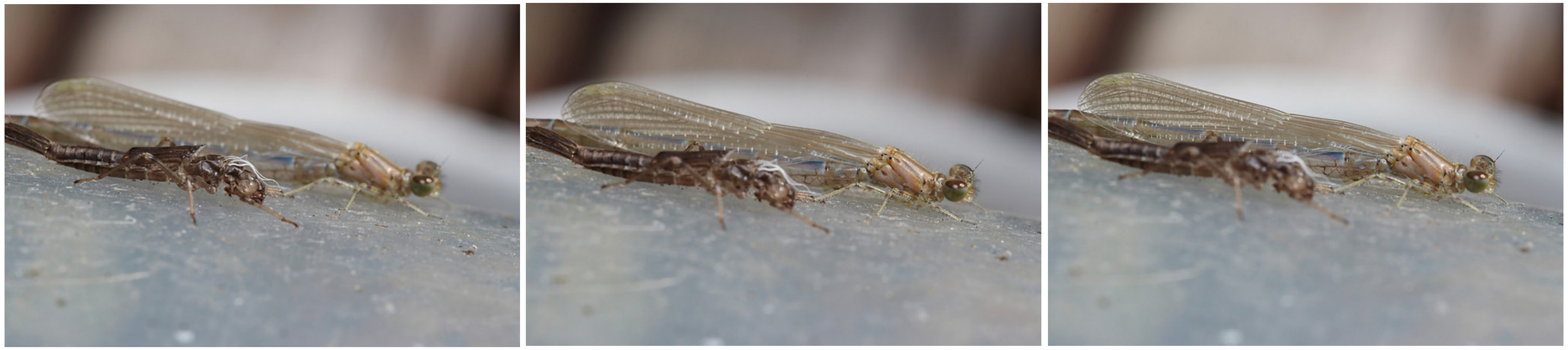
How to get both bugs in focus?





Solution: Focus stacking

1. Take pictures with varying focal length

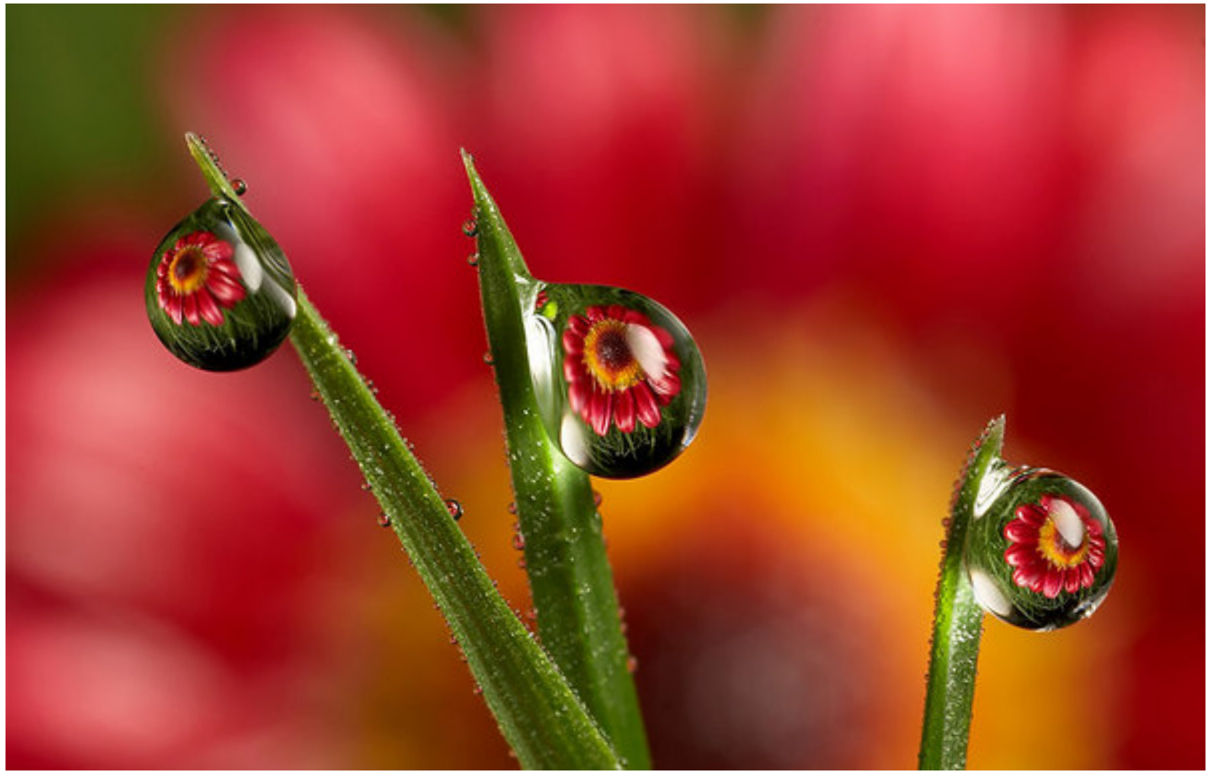


Example from
http://www.wonderfulphotos.com/articles/macro/focus_stacking/

Solution: Focus stacking

1. Take pictures with varying focal length
2. Combine

Focus stacking



http://www.wonderfulphotos.com/articles/macro/focus_stacking/

Focus stacking

How to combine?

1. Align images (e.g., using corresponding points)
2. Two ideas
   a) Mask regions by hand and combine with pyramid blend
   b) Gradient domain fusion (mixed gradient) without masking

Automatic solution would make an interesting final project

Recommended Reading:

http://www.digital-photography-school.com/an-introduction-to-focus-stacking

http://www.zen20934.zen.co.uk/photography/Workflow.htm#Focus%20Stacking

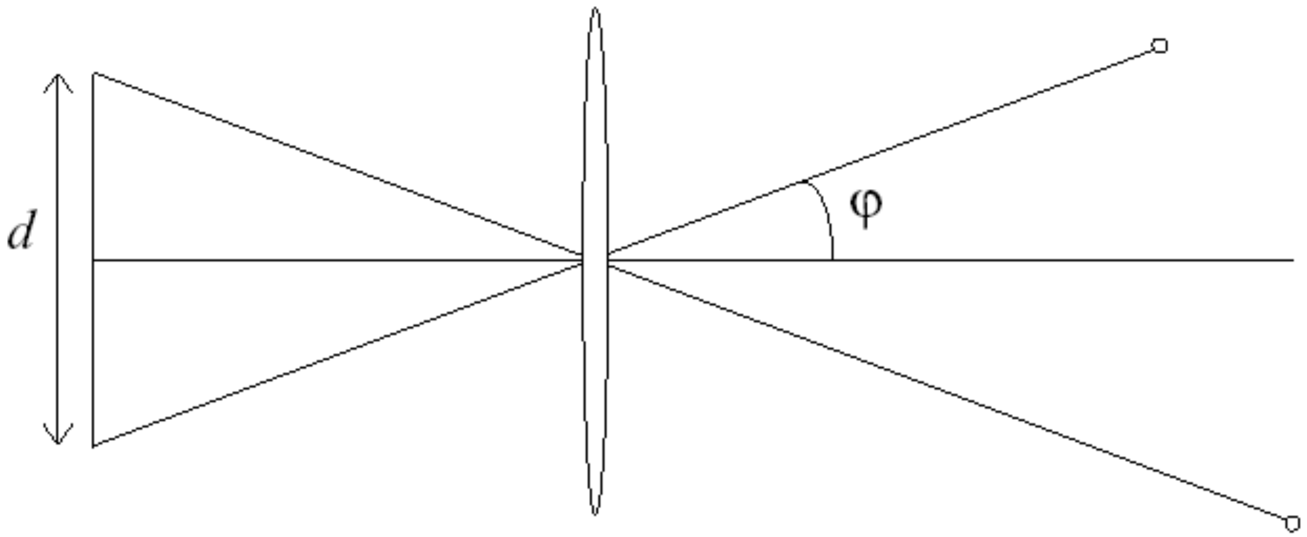
Relation between field of view and focal length

Field of view (angle width)

Film/Sensor Width

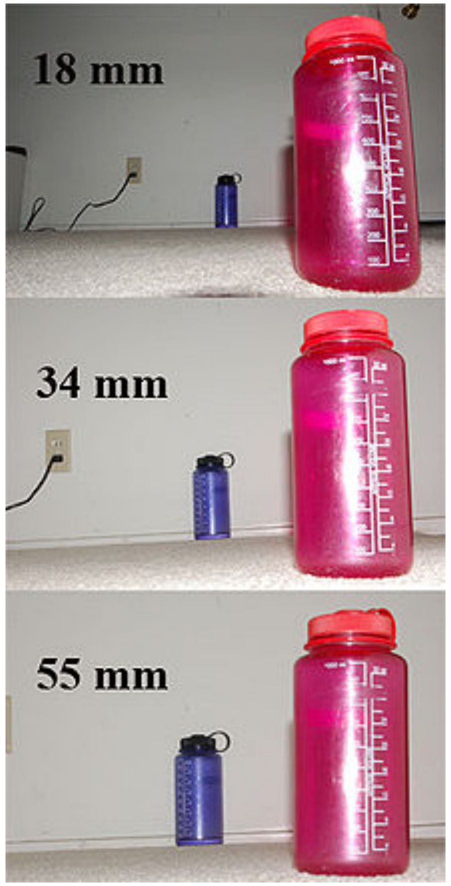
$$fov = 2\tan^{-1}\frac{d}{2f}$$

Focal length



Dolly Zoom or “Vertigo Effect”

http://www.youtube.com/watch?v=NB4bikrNzMk



How is this done?

Zoom in while moving away

http://en.wikipedia.org/wiki/Focal_length

Dolly zoom (or "Vertigo effect")

Field of view (angle width)

Film/Sensor Width

$$fov = 2\tan^{-1}\frac{d}{2f}$$

Focal length

width of object

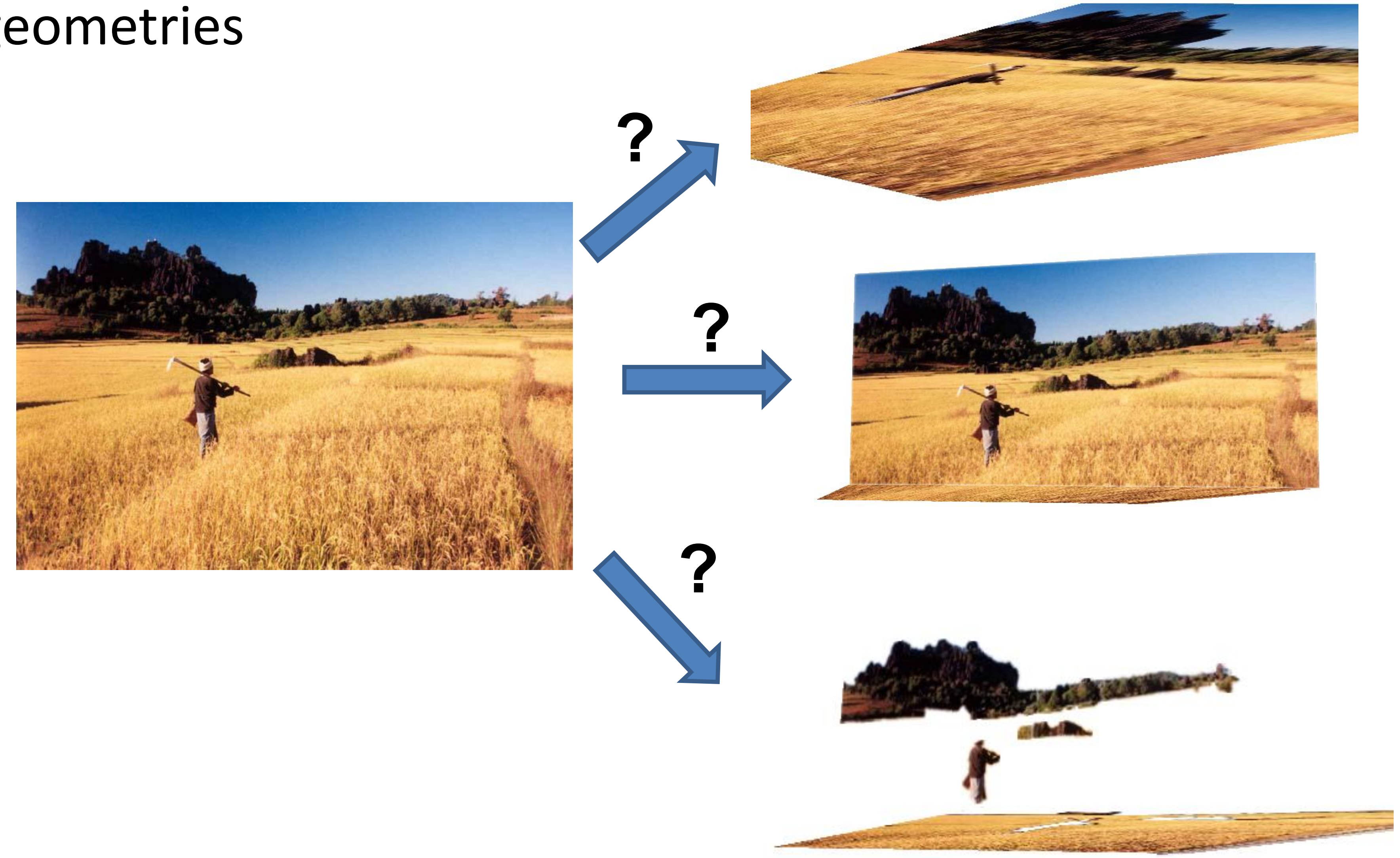
$$2\tan\frac{fov}{2} = \frac{width}{\text{distance}}$$

Distance between object and camera

Today’s class: 3D Reconstruction

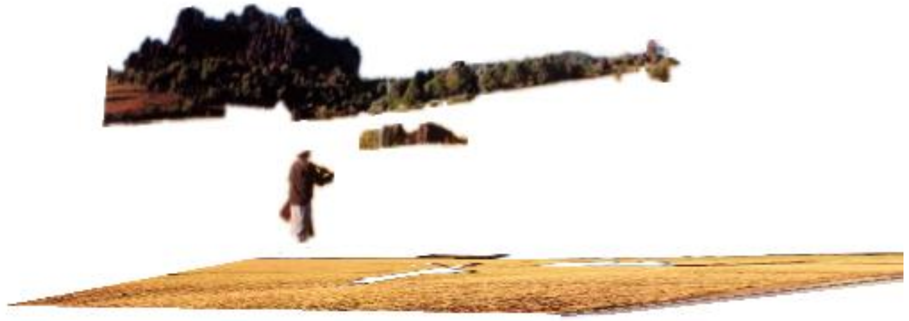
The challenge

One 2D image could be generated by an infinite number of 3D geometries

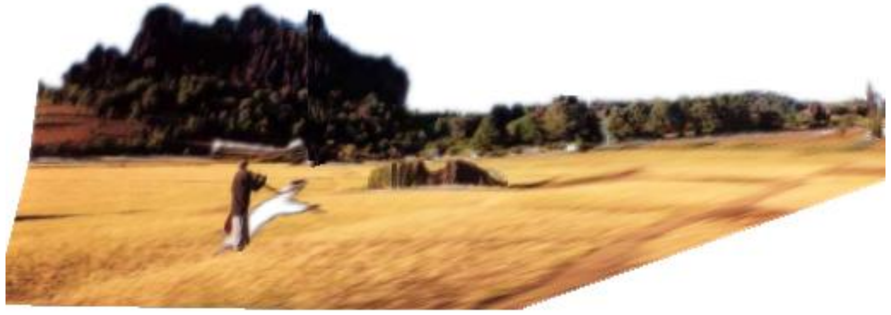


The solution

Make simplifying assumptions about 3D geometry



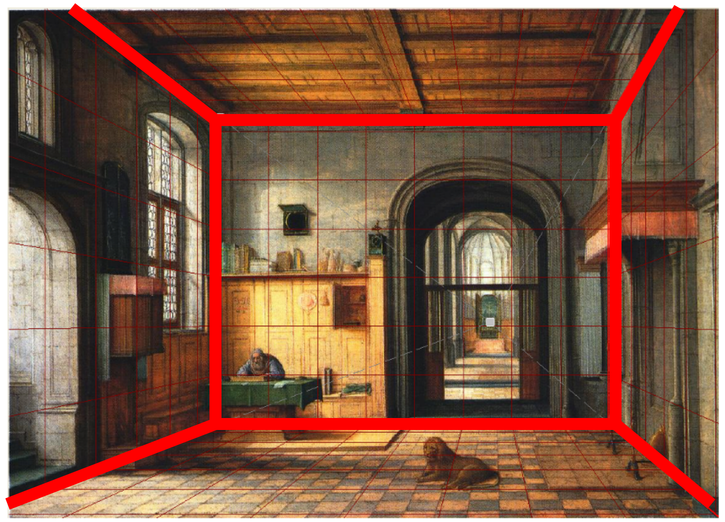
Unlikely



Likely

Today’s class: Two Models

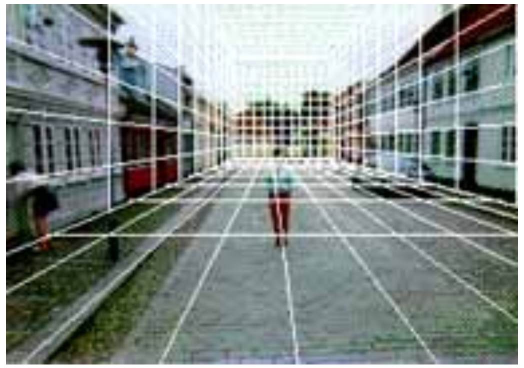
- Box + frontal billboards



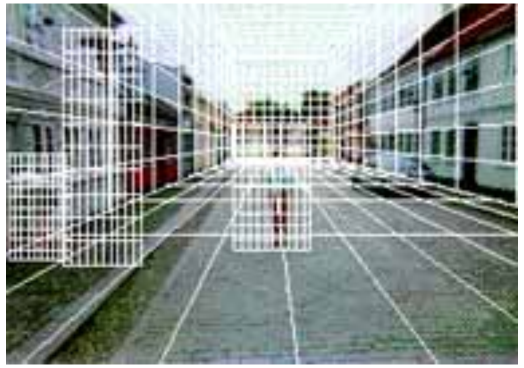
- Ground plane + non-frontal billboards

“Tour into the Picture” (Horry et al. SIGGRAPH ’97)

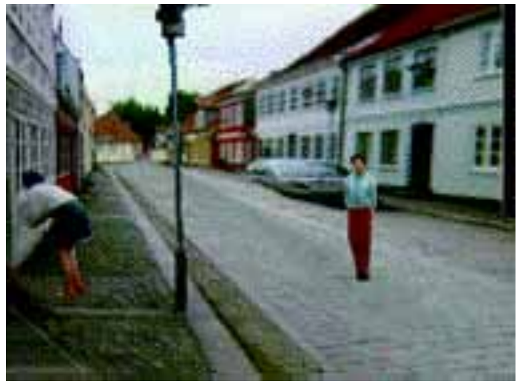
Create a 3D “theatre stage” of five planes



Specify foreground objects through bounding polygons



Use camera transformations to navigate through the scene



Following slides modified from Efros

The idea

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

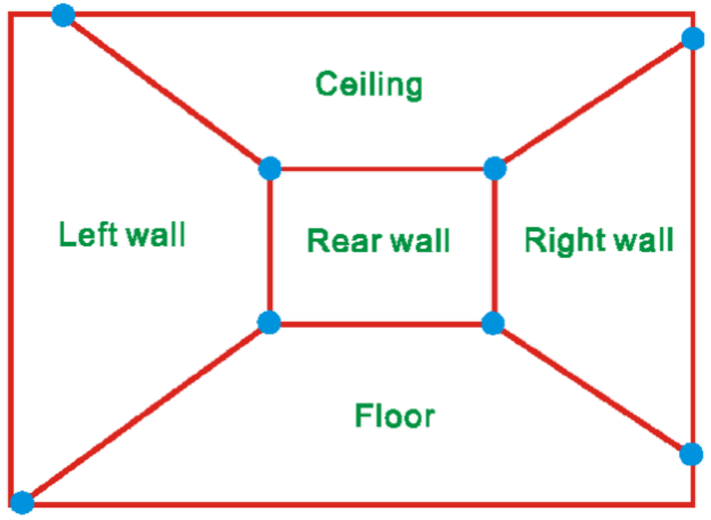
Key assumptions

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

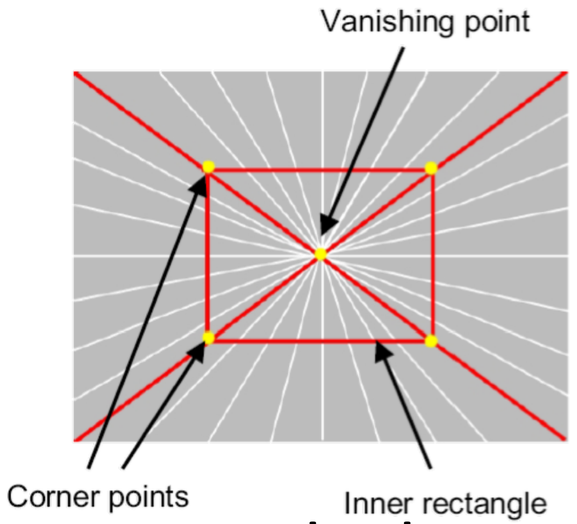
How many vanishing points does the box have?

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

Can use the vanishing point to fit the box to the particular scene



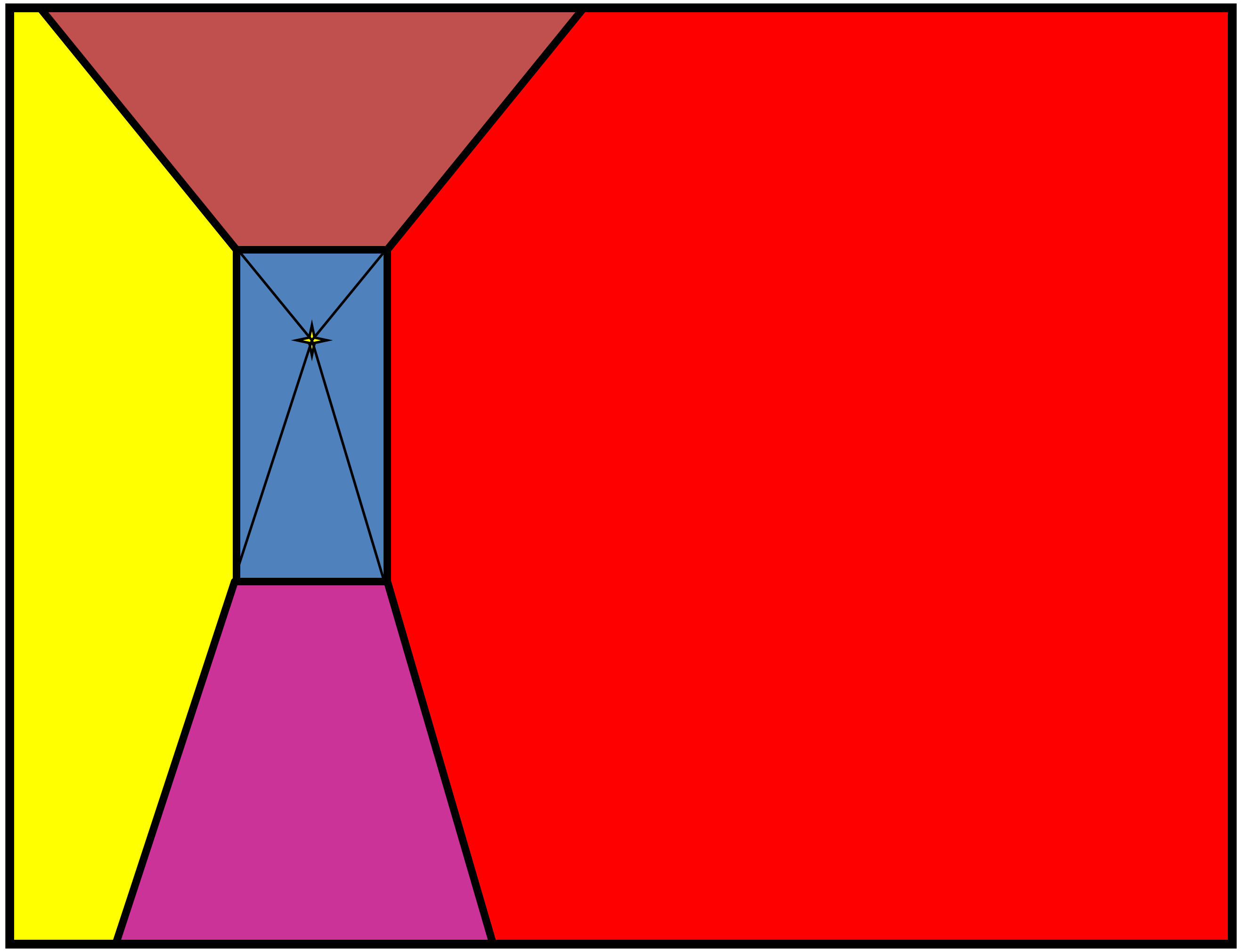
Step 1: specify scene geometry



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

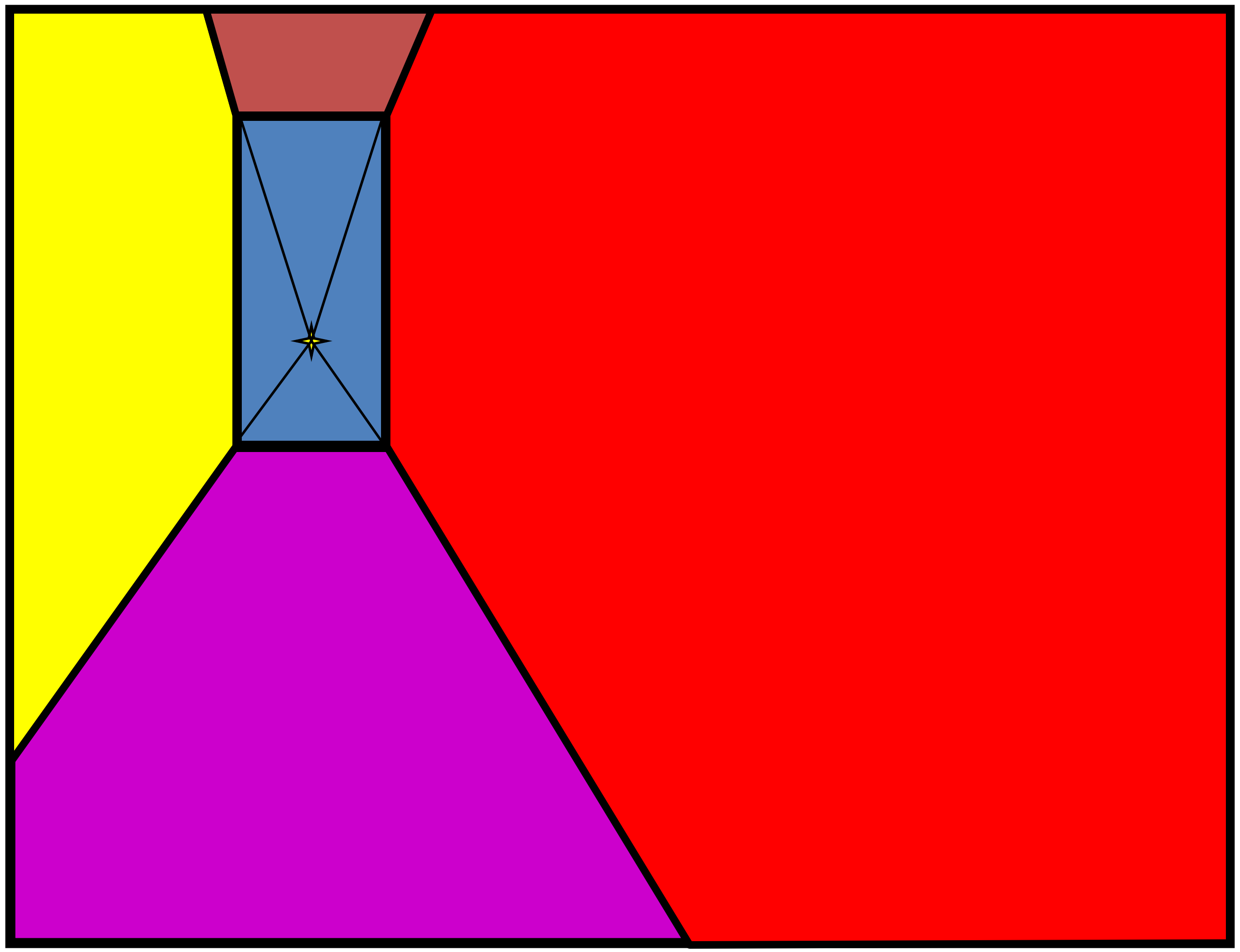
- Q: If we assume camera is looking straight at back wall, what camera parameter(s) does the vanishing point position provide?
- A: Vanishing point direction is perpendicular to image plane, so the vp is the principal point

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



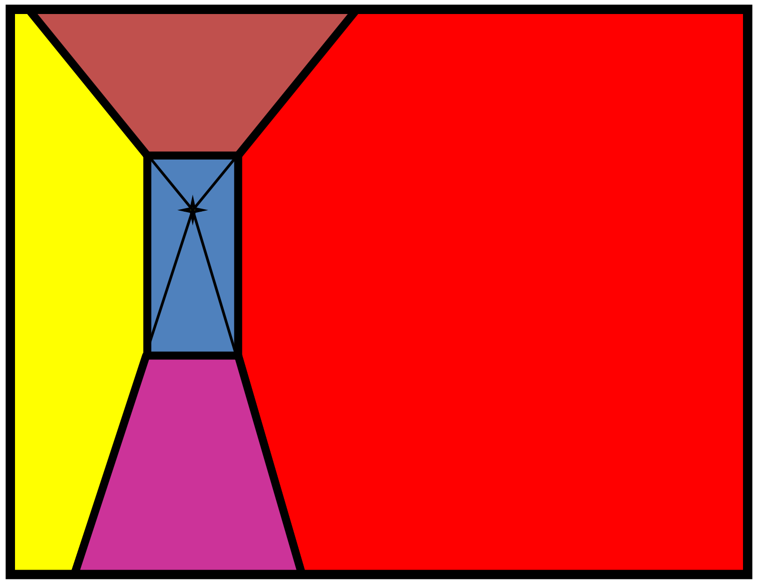
High Camera

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

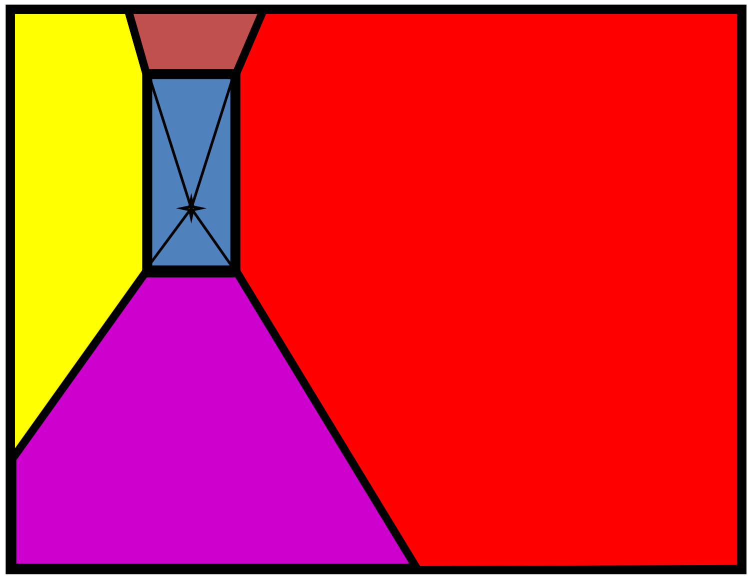


Low Camera

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.

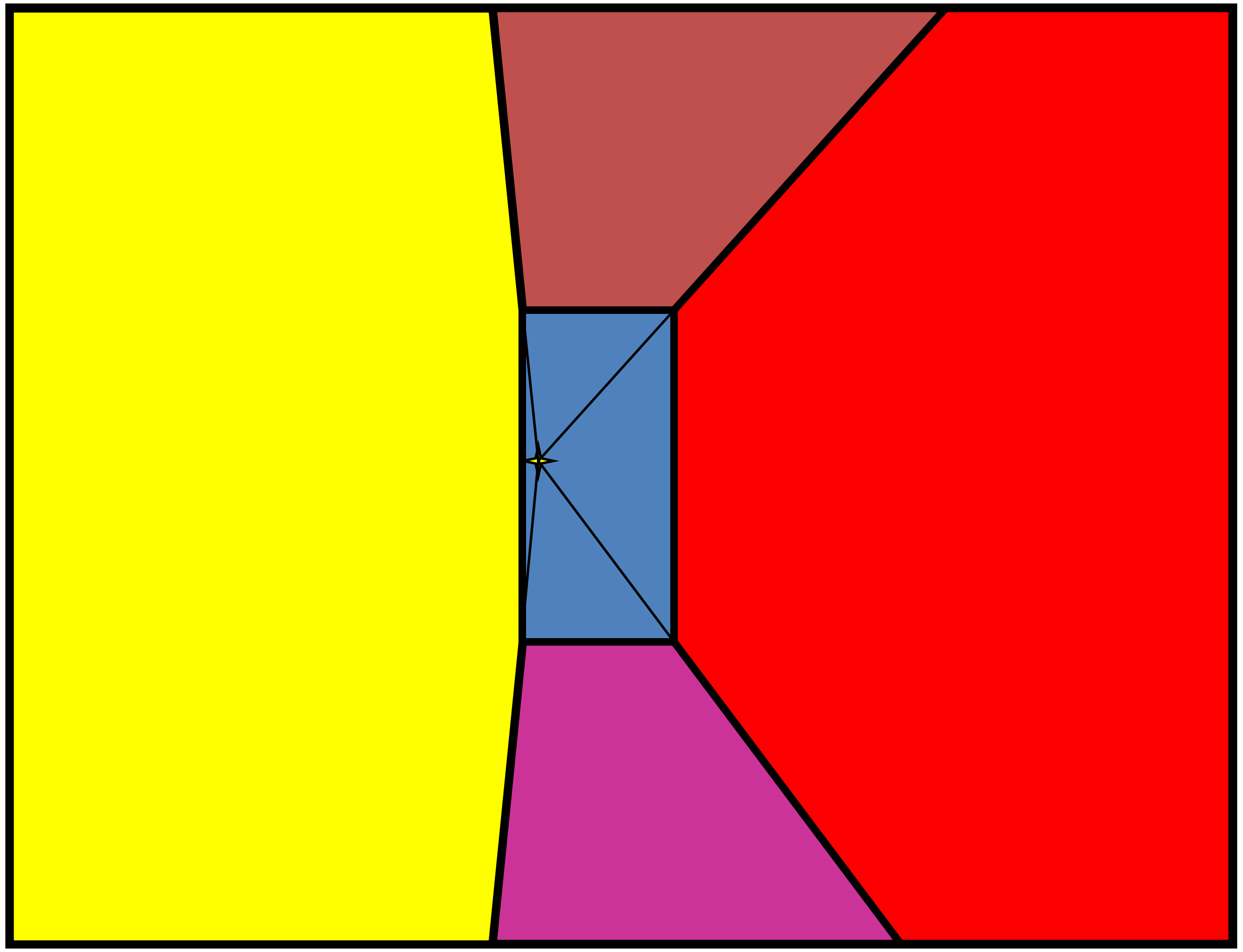


High Camera



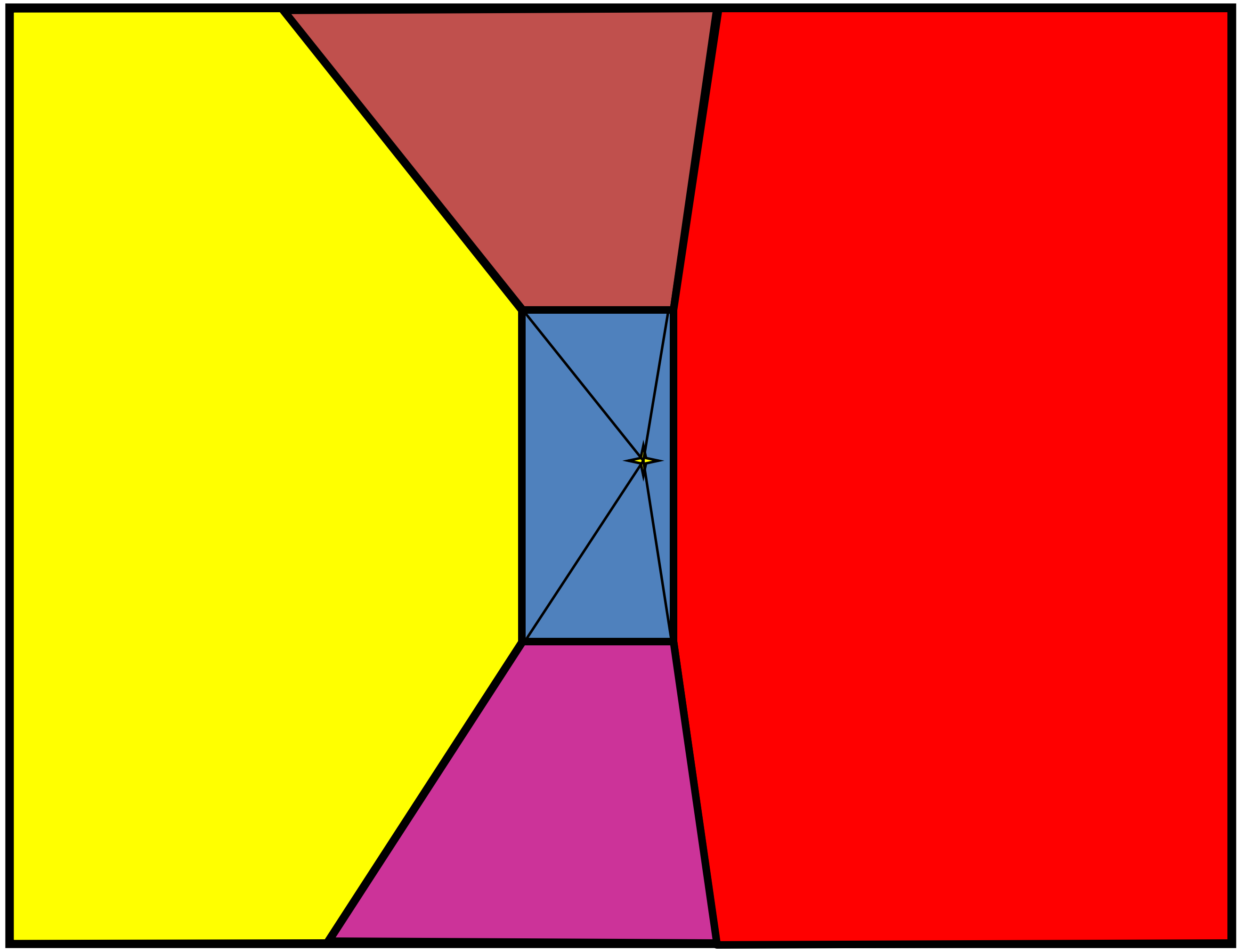
Low Camera

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



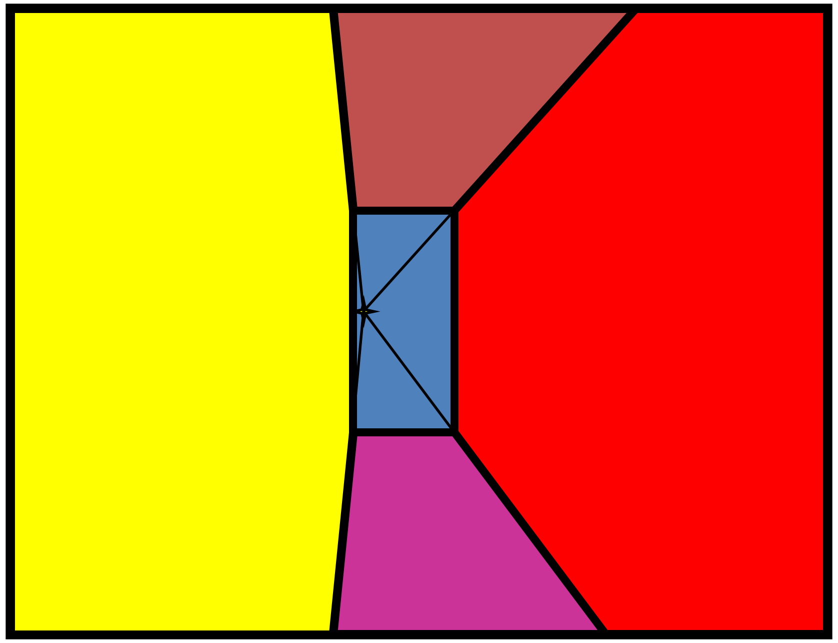
Left Camera

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

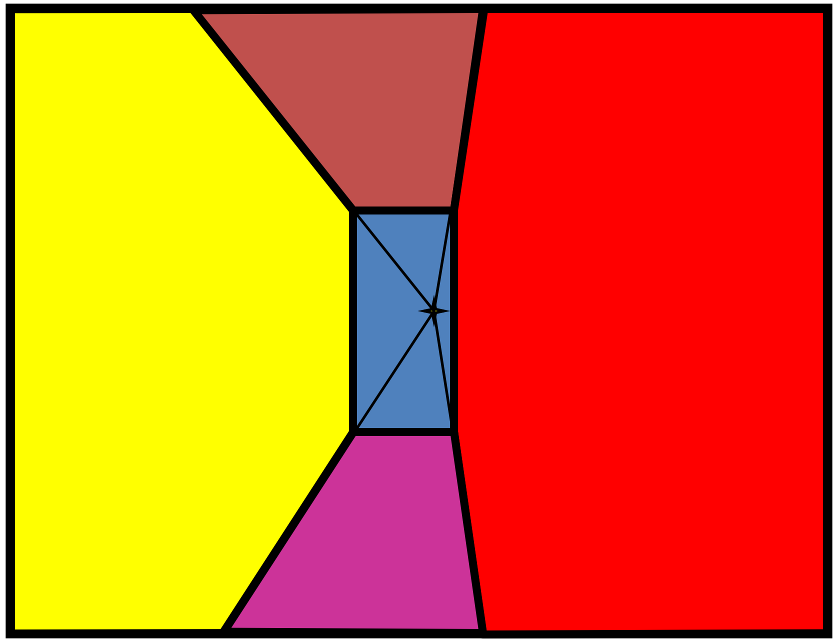


Right Camera

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



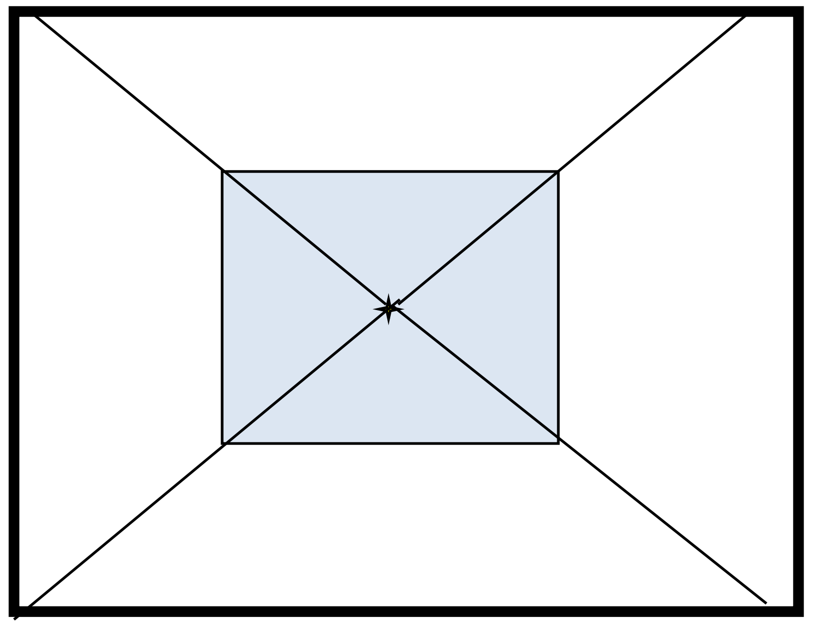
Left Camera



Right Camera

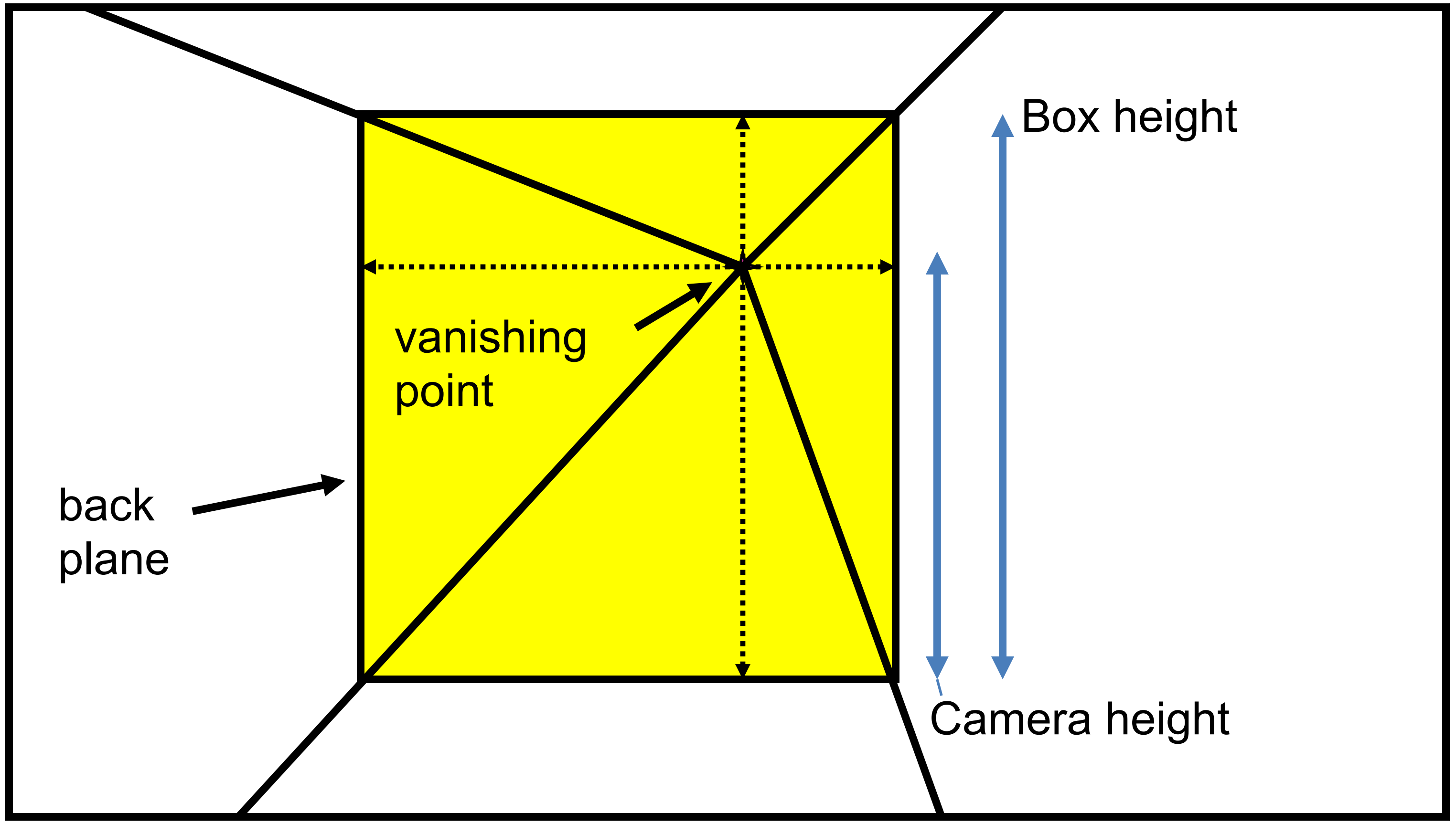
Question

- Think about the camera center and image plane…
  - What happens when we move the box?
  - What happens when we move the vanishing point?



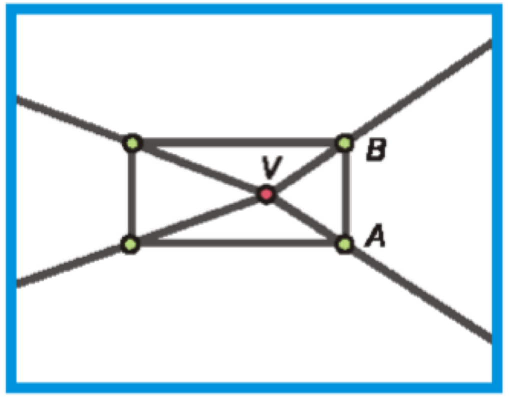
2D to 3D conversion

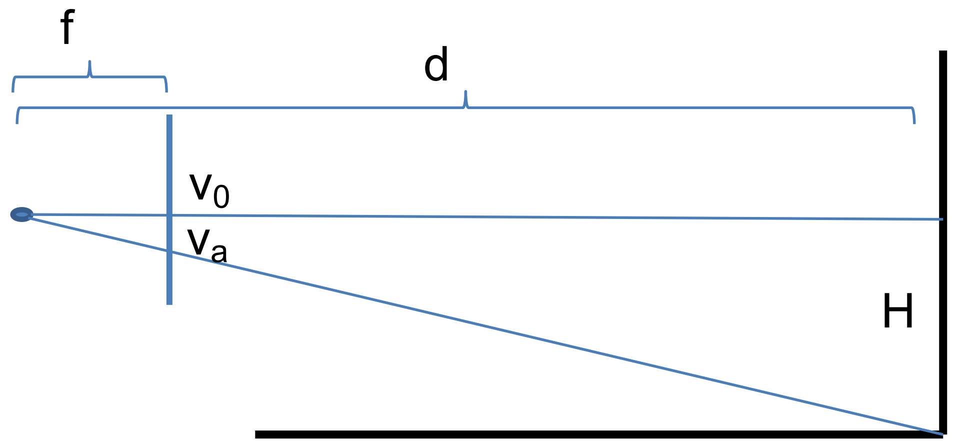
- Use ratios



Box width / height in 3D is proportional to width over height in the image because back plane is parallel to image plane

Get depth using similar triangles



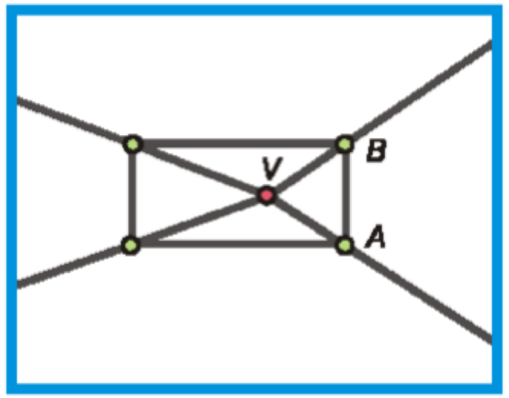


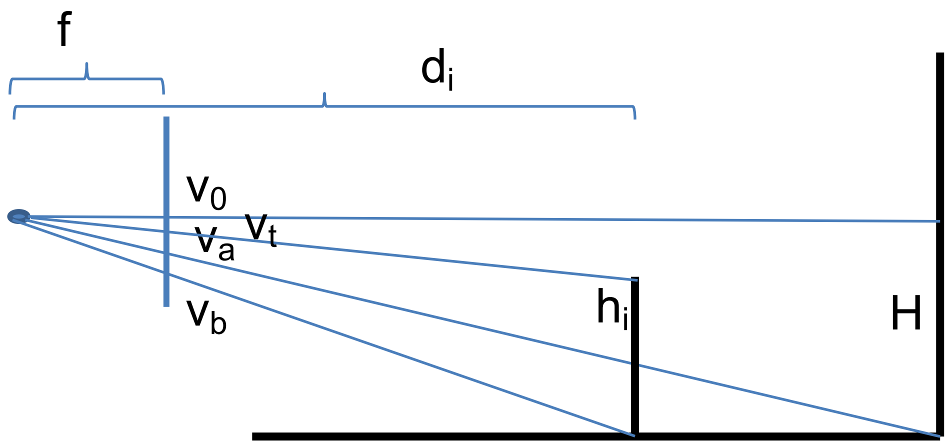
$f / (v_a - v_0) = d / H$

- Can compute by similar triangles (CVA vs. CV’A’)
- Need to know focal length f (or FoV)

- Note: can compute position of any object on the ground
  - Simple unprojection
  - What about things off the ground?

Get depth using similar triangles





$f / (v_a - v_0) = d / h$

$h_i = (v_t - v_b) / (v_0 - v_a) * H$

$f / (v_b - v_t) = d_i / h_i$

- Can compute by similar triangles (CVA vs. CV'A')
- Need to know focal length f (or FoV)

- Can compute 3D position of any object on the ground w/ unprojection
  – What about things off the ground?

Step 2: map image textures into frontal view

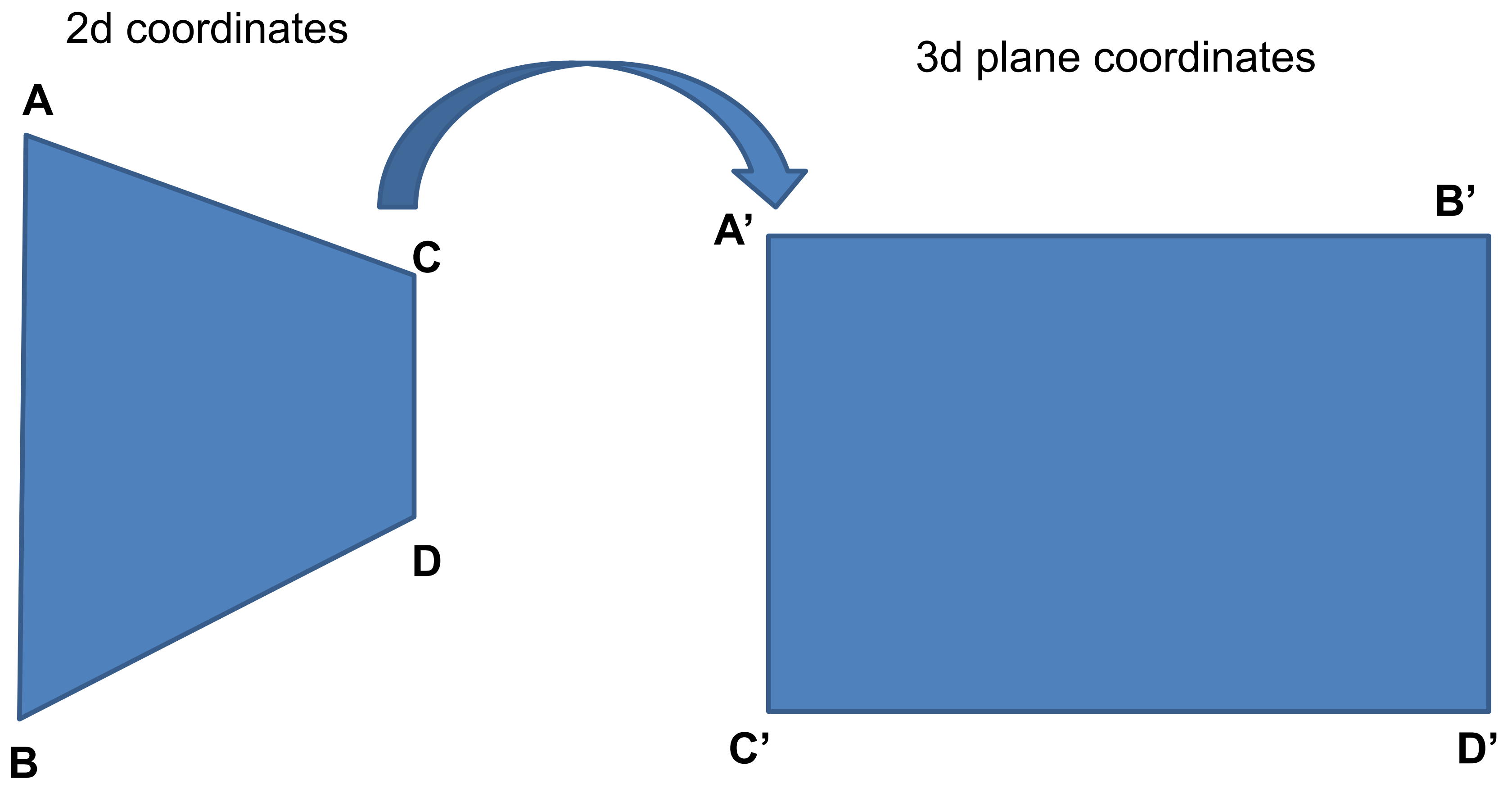
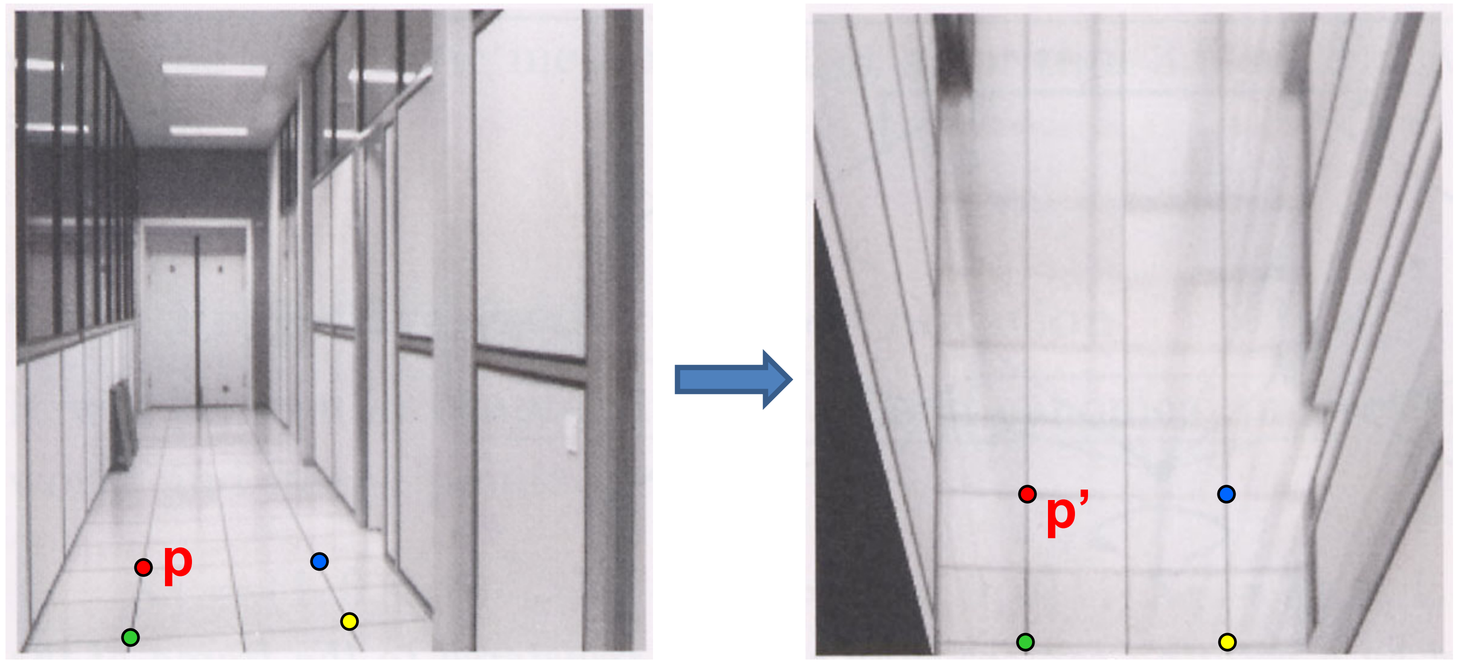


Image rectification by homography



To unwarp (rectify) an image solve for homography **H** given **p** and **p'**: w**p'**=**Hp**

Computing homography

Assume we have four matched points: How do we compute homography **H**?

Direct Linear Transformation (DLT)

$$\mathbf{p}' = \mathbf{H}\mathbf{p} \qquad \mathbf{p}' = \begin{bmatrix} w'u' \\ w'v' \\ w' \end{bmatrix} \qquad \mathbf{p} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \qquad \mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

$$\begin{bmatrix} -u & -v & -1 & 0 & 0 & 0 & uu' & vu' & u' \\ 0 & 0 & 0 & -u & -v & -1 & uv' & vv' & v' \end{bmatrix} \mathbf{h} = \mathbf{0} \qquad \mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix}$$

Computing homography

Direct Linear Transform

$$\begin{bmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & u_1u_1' & v_1u_1' & u_1' \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & u_1v_1' & v_1v_1' & v_1' \\ & & & & \vdots & & & & \\ 0 & 0 & 0 & -u_n & -v_n & -1 & u_nv_n' & v_nv_n' & v_n' \end{bmatrix}\mathbf{h} = \mathbf{0} \Rightarrow \mathbf{Ah} = \mathbf{0}$$

- Apply SVD: ***USV***^{T} **=** ***A***
- ***h* = *V***$_{\text{smallest}}$ (column of ***V***^{T} corresponds to smallest singular value)

$$\mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_9 \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix}$$

Python

```
U, S, Vt = scipy.linalg.svd(A)
# last column corr. to smallest singular value
h = Vt[:,-1];
```

Explanation of SVD, solving systems of linear equations, derivation of solution here

Solving for homography (another formulation)

$$\begin{array}{c}\begin{bmatrix} -u_1 & -v_1 & -1 & 0 & 0 & 0 & u_1u_1' & v_1u_1' & u_1' \\ 0 & 0 & 0 & -u_1 & -v_1 & -1 & u_1v_1' & v_1v_1' & v_1' \\ & & & & \vdots & & & & \\ 0 & 0 & 0 & -u_n & -v_n & -1 & u_nv_n' & v_nv_n' & v_n' \end{bmatrix} \mathbf{h} = \mathbf{0} \Rightarrow \mathbf{Ah} = \mathbf{0}\end{array}$$

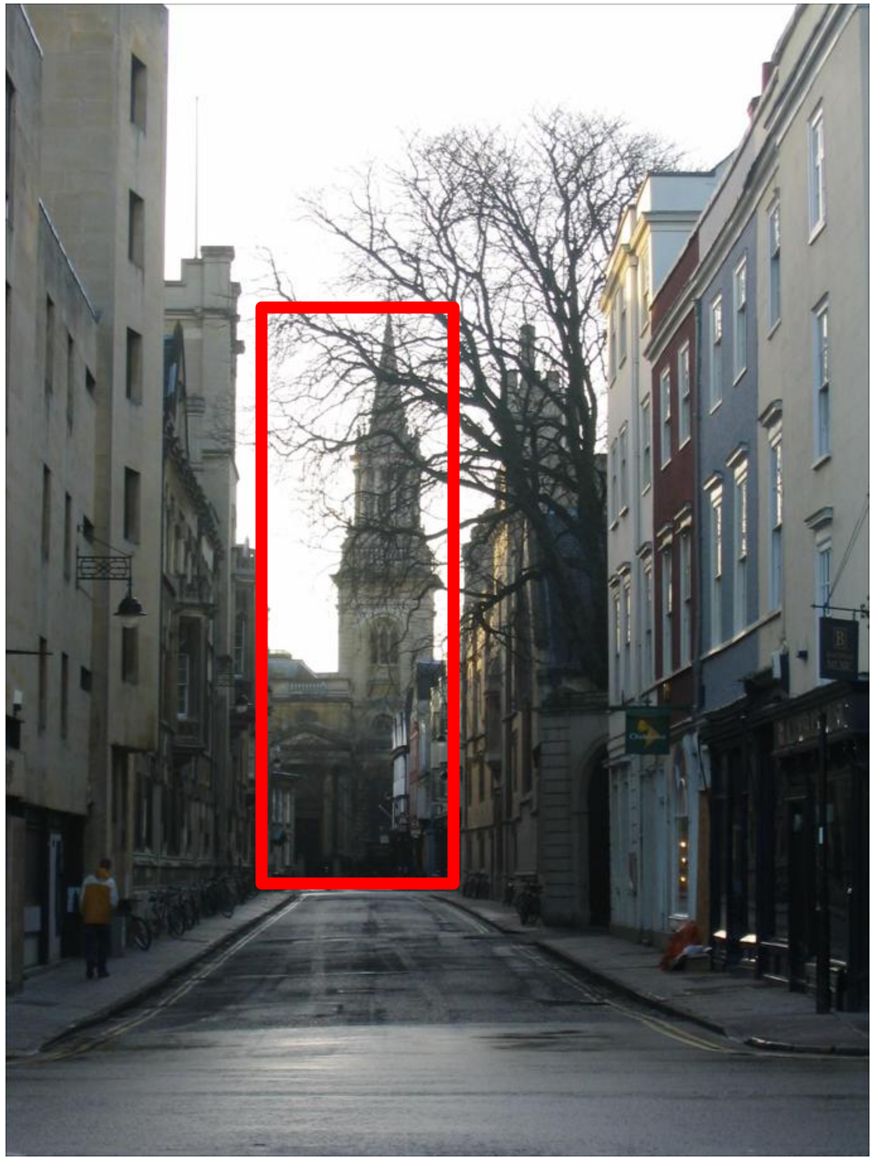
$$\underset{2n \times 9}{\mathbf{A}} \qquad \underset{9}{\mathbf{h}} \qquad \underset{2n}{\mathbf{0}}$$

Defines a least squares problem: $\text{minimize } \|\mathbf{Ah} - \mathbf{0}\|^2$

- Since **h** is only defined up to scale, solve for unit vector $\hat{\mathbf{h}}$
- Solution: $\hat{\mathbf{h}}$ = eigenvector of $\mathbf{A}^T\mathbf{A}$ with smallest eigenvalue
  - Can derive using Lagrange multipliers method
- Works with 4 or more points

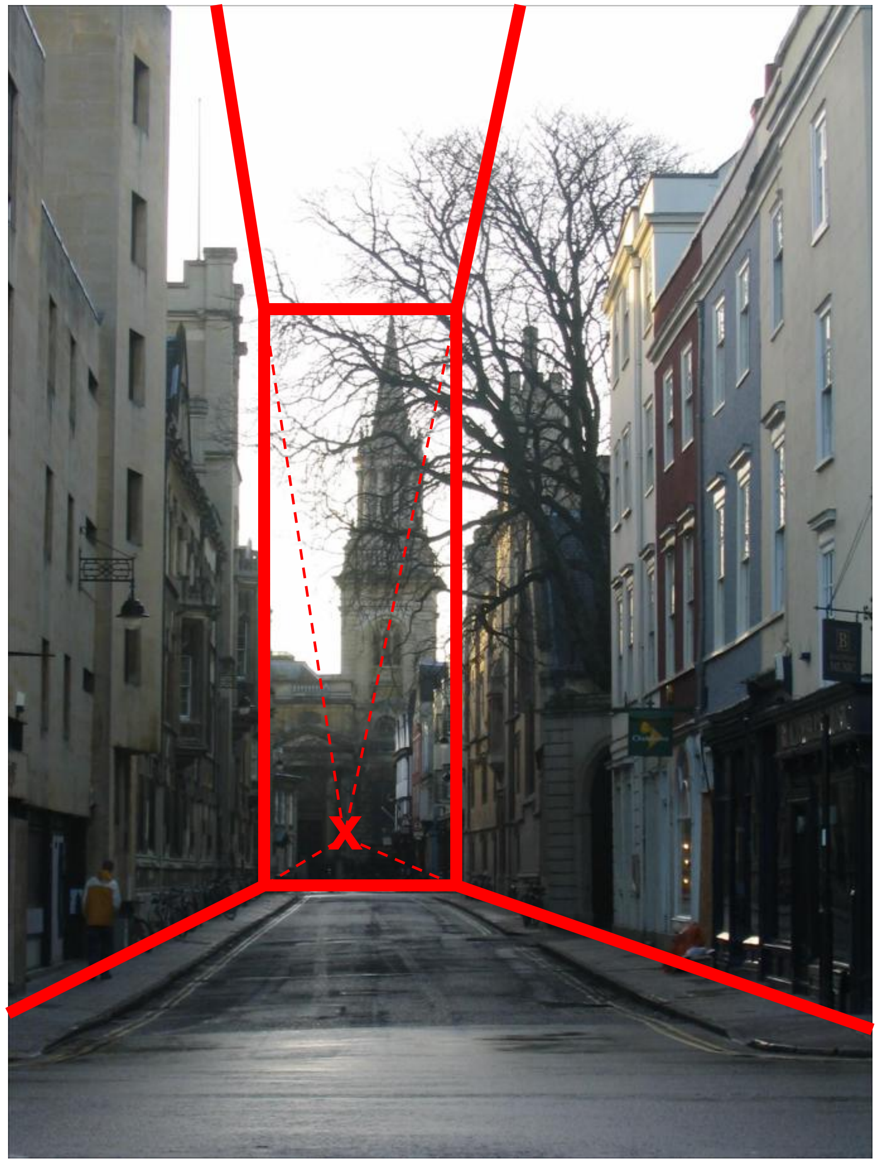
Tour into the picture algorithm

1. Set the box corners



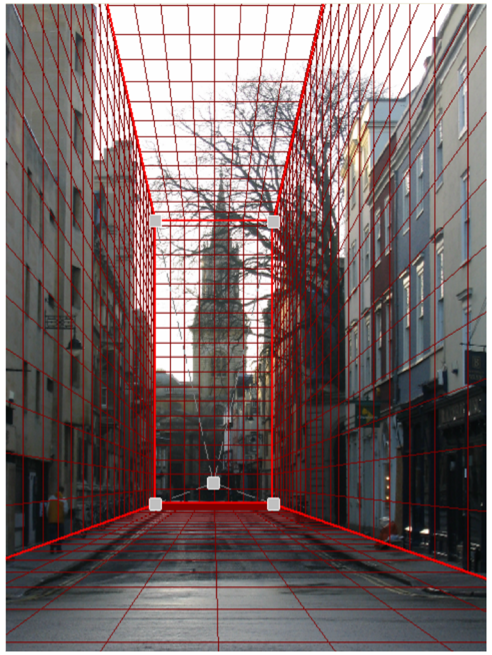
Tour into the picture algorithm

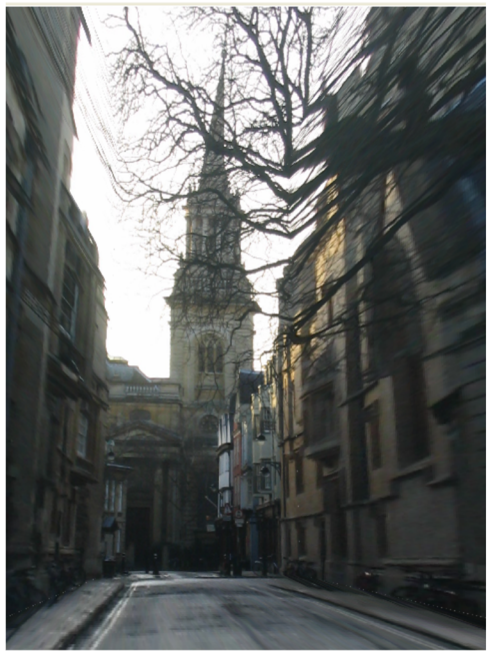
1. Set the box corners
2. Set the VP
3. Get 3D coordinates
   – Compute height, width, and depth of box
4. Get texture maps
   – homographies for each face
5. Create file to store plane coordinates and texture maps

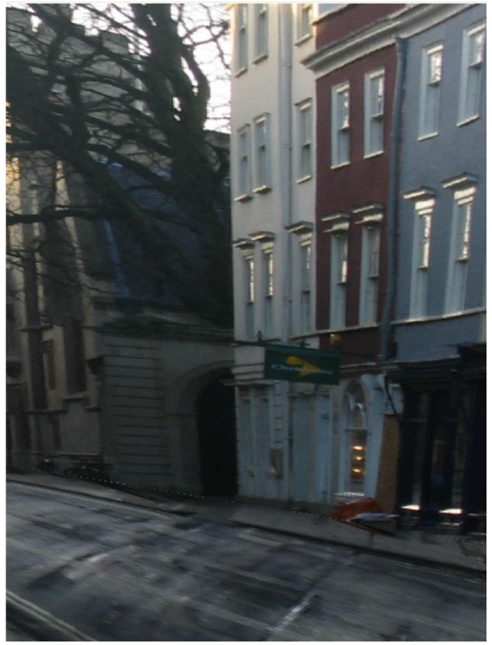


Result

Render from new views







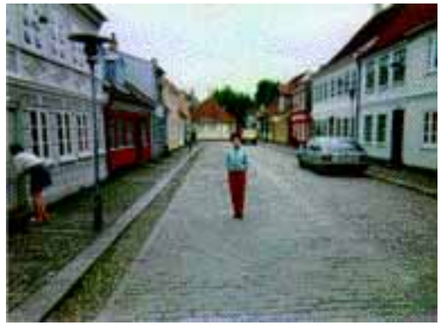
http://www.cs.cmu.edu/afs/cs.cmu.edu/academic/class/15463-f08/www/proj5/www/dmillett/

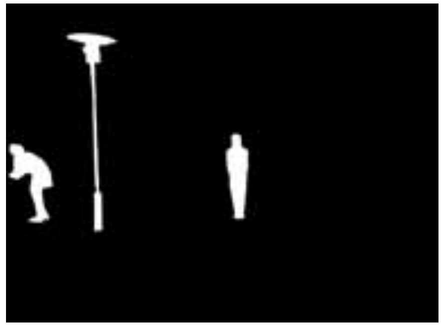
Foreground Objects

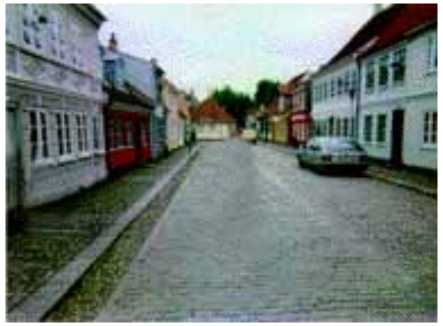
Use separate billboard for each

For this to work, three separate images used:

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





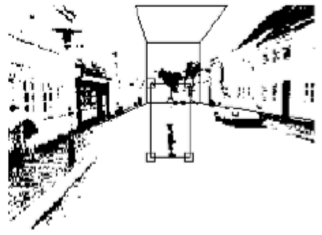


Foreground Objects

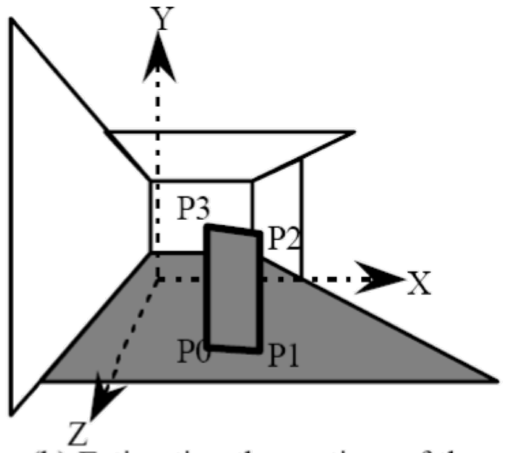
Add vertical rectangles for each foreground object

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

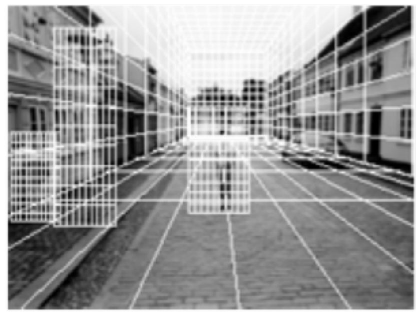
P2, P3 can be computed as before (similar triangles)



(a) Specifying of a foreground object

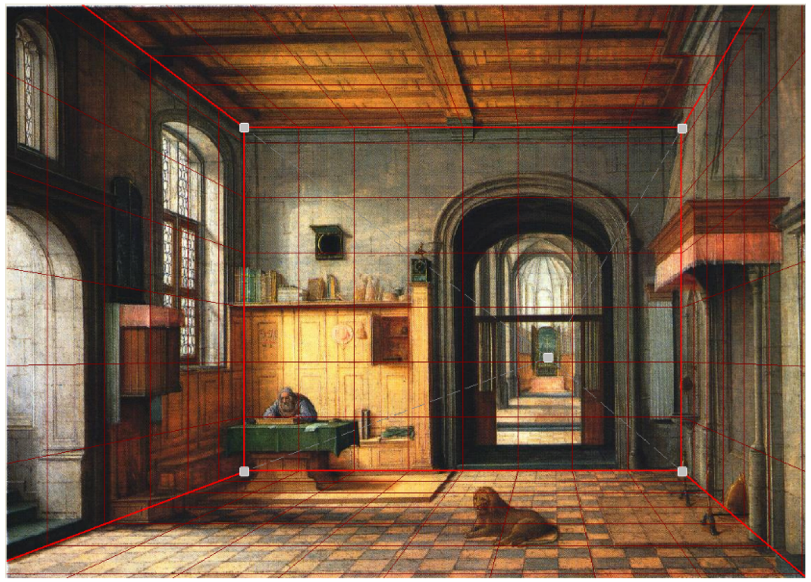


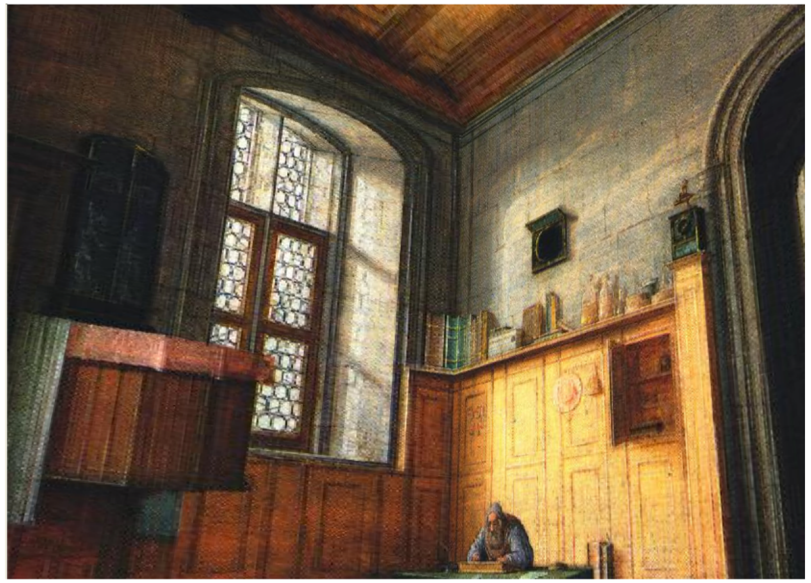
(b) Estimating the vertices of the foreground object model



(c) Three foreground object models

Foreground Result





Video from CMU class:
http://www.youtube.com/watch?v=dUAtdmGwcuM

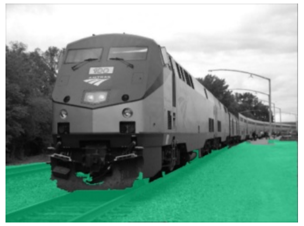
Automatic Photo Pop-up

Input　　Geometric Labels　　Cut'n'Fold　　3D Model

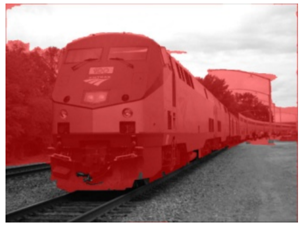
Image

Learned Models

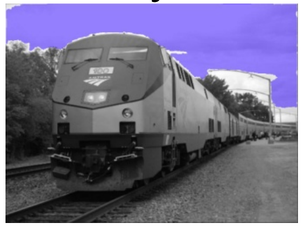
Ground

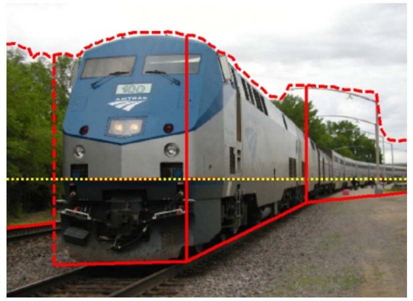


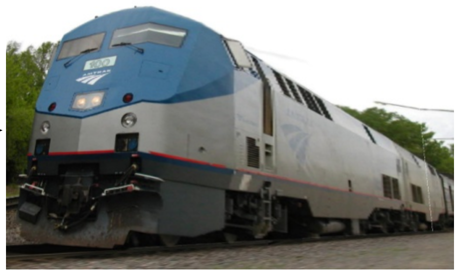
Vertical



Sky

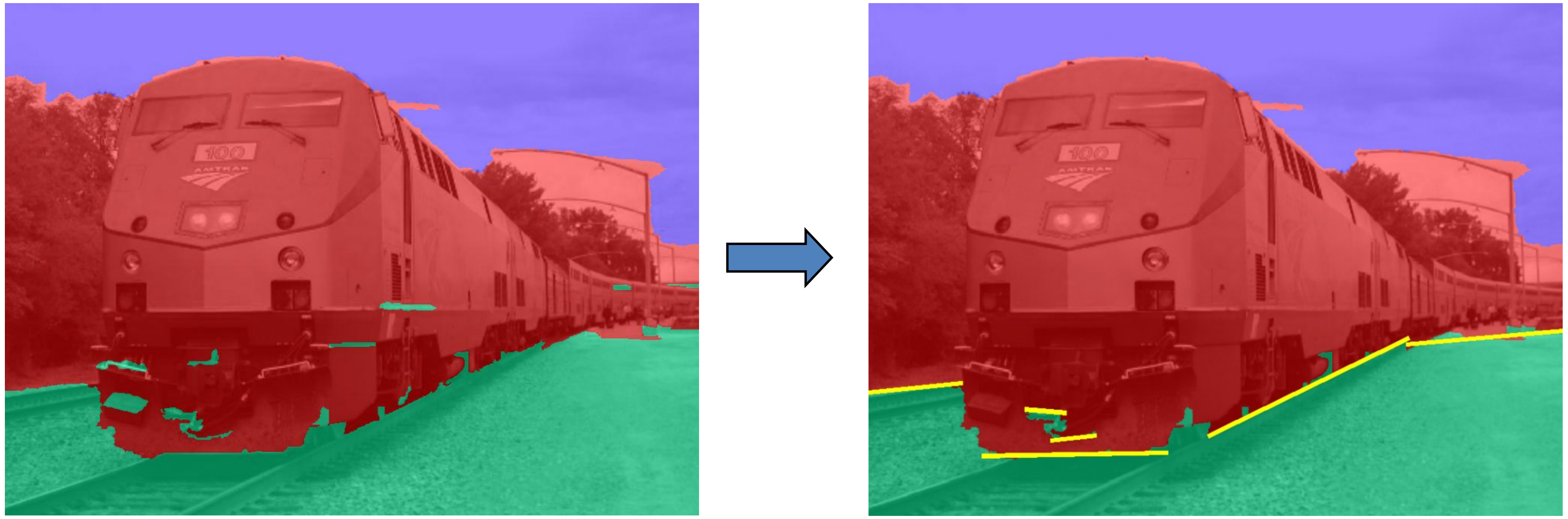






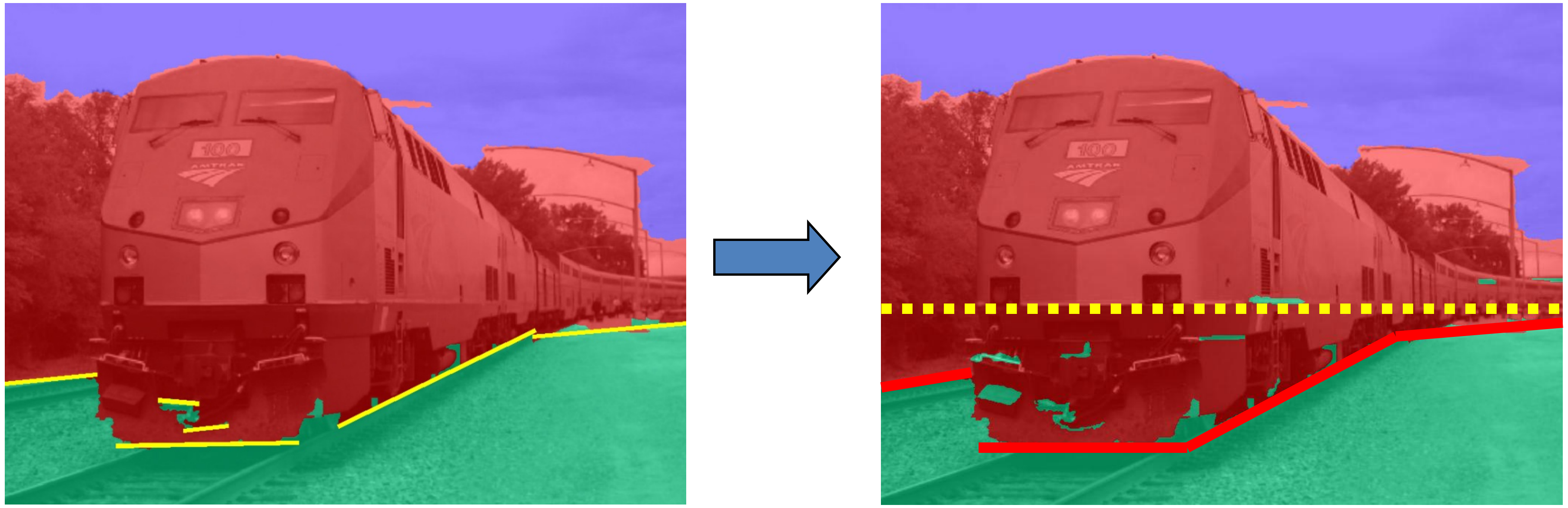
Hoiem et al. 2005

Cutting and Folding



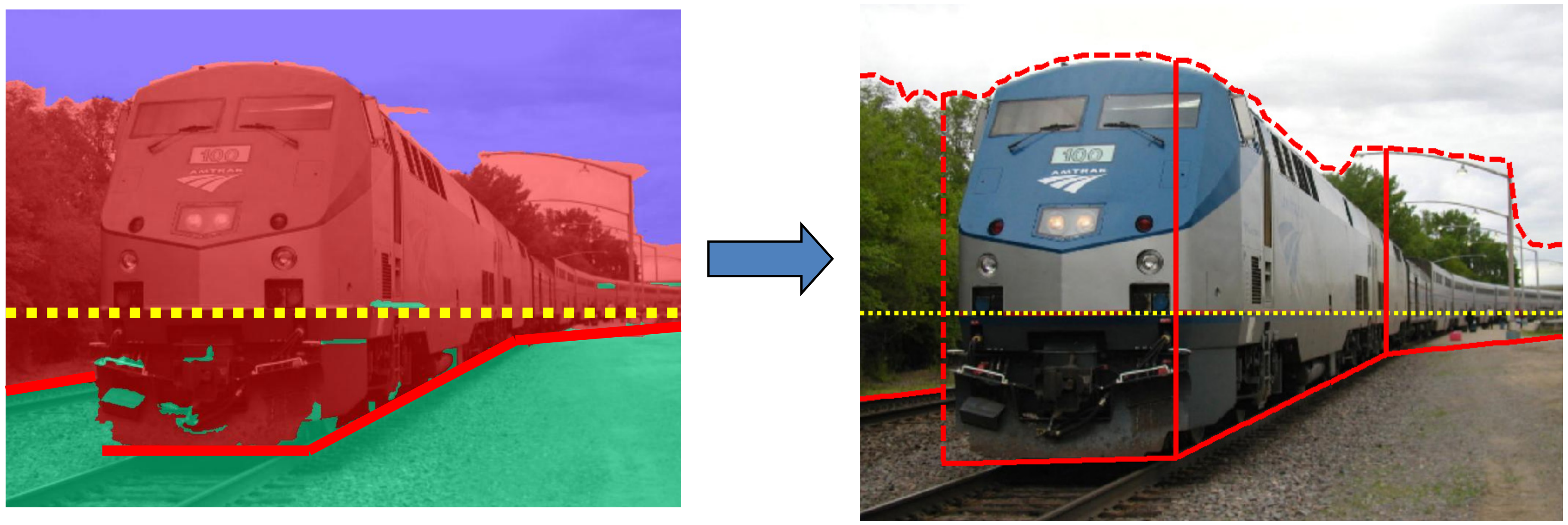
- Fit ground-vertical boundary
  - Iterative Hough transform

Cutting and Folding



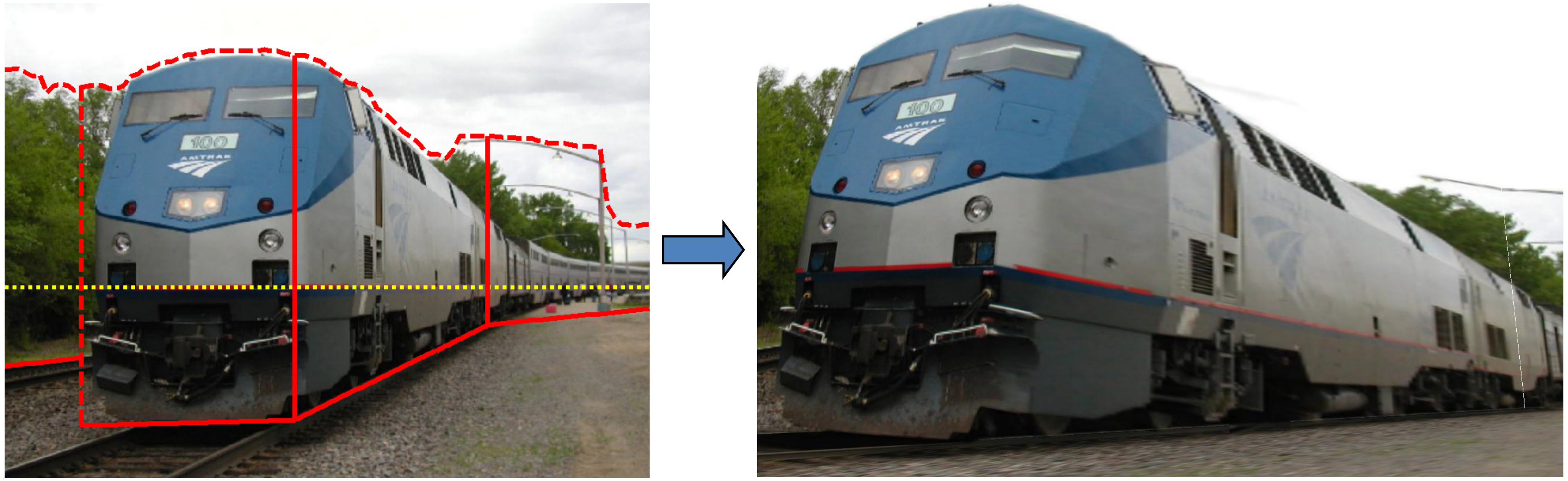
- Form polylines from boundary segments
  - Join segments that intersect at slight angles
  - Remove small overlapping polylines
- Estimate horizon position from perspective cues

Cutting and Folding



- ``Fold'' along polylines and at corners
- ``Cut'' at ends of polylines and along vertical-sky boundary

Cutting and Folding

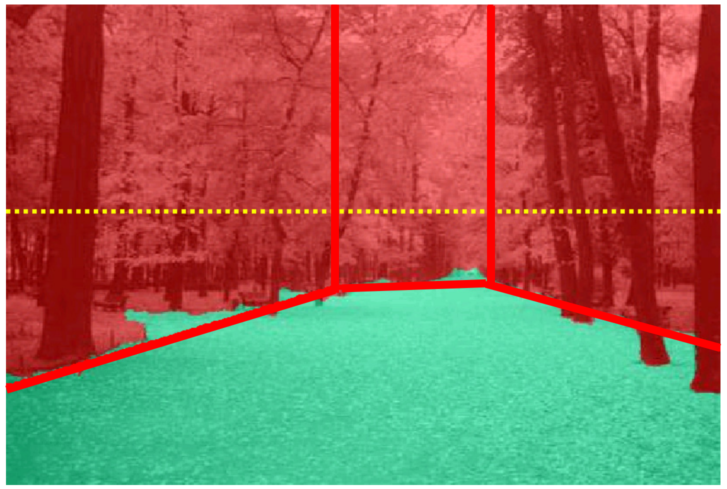


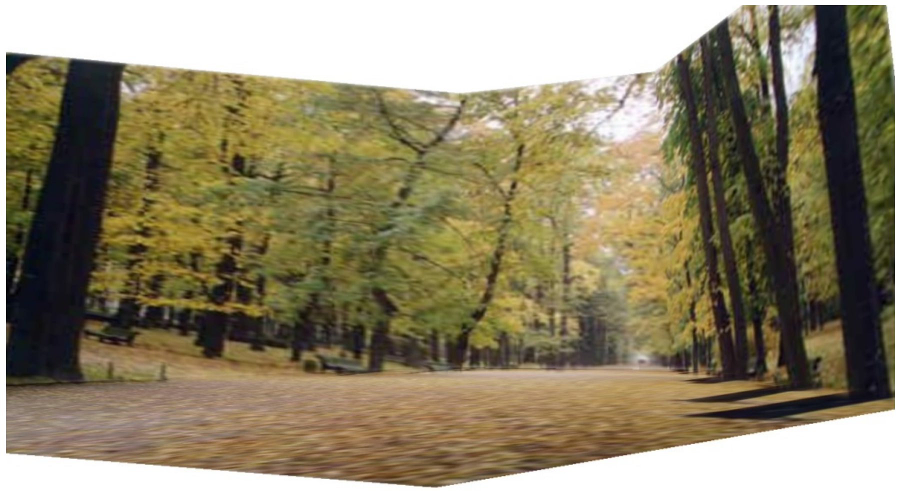
- Construct 3D model
- Texture map

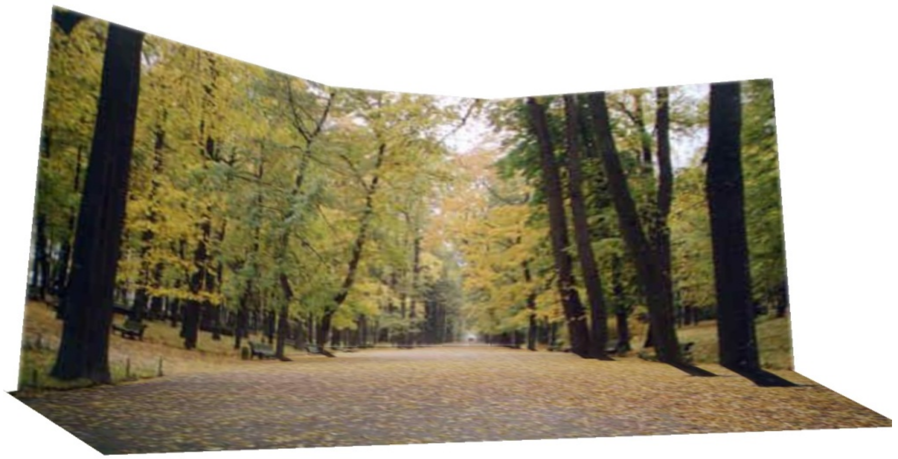
Results

http://www.cs.illinois.edu/homes/dhoiem/projects/popup/

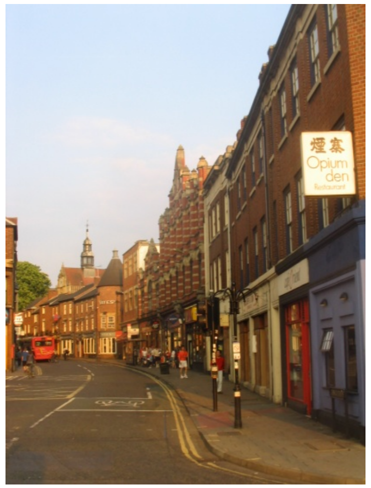
Input Image

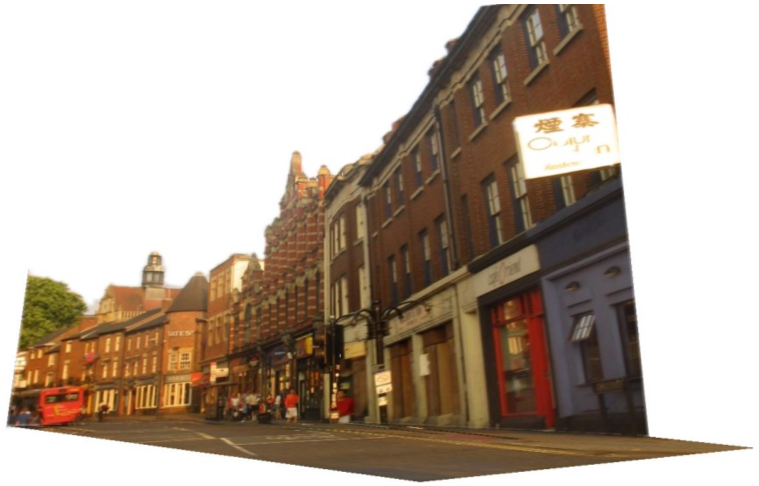

Cut and Fold

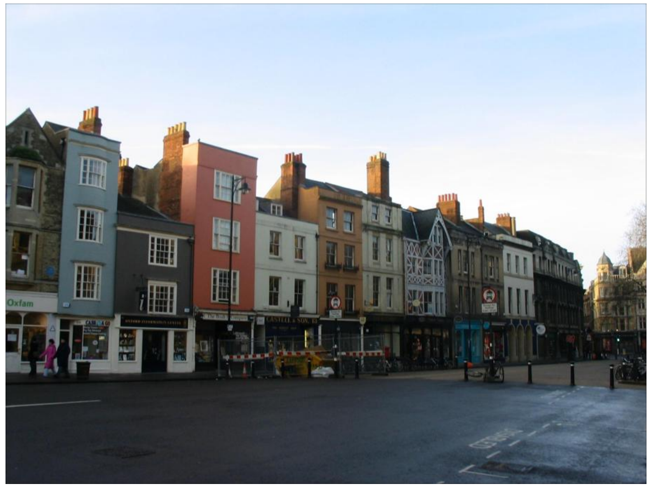


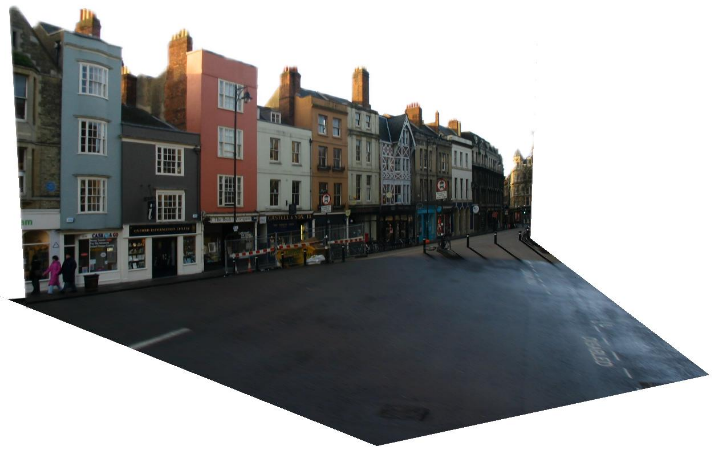

Automatic Photo Pop-up

Results





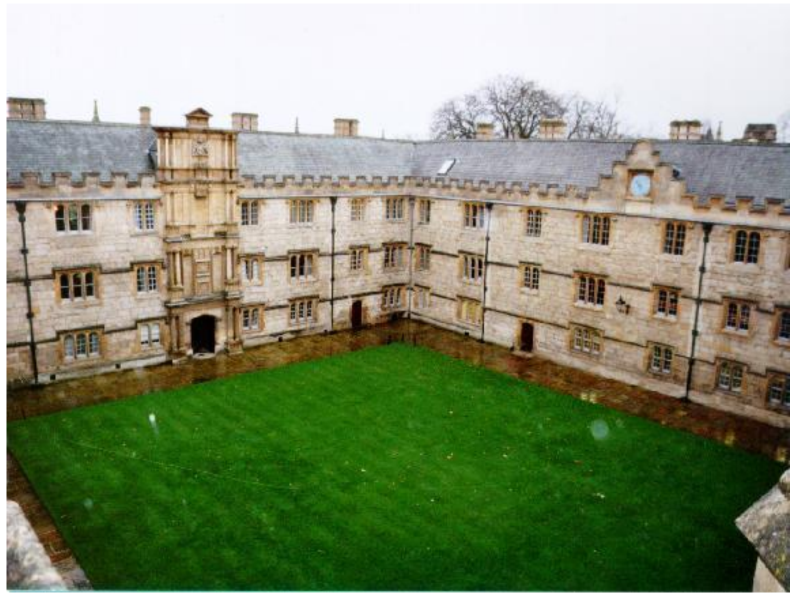


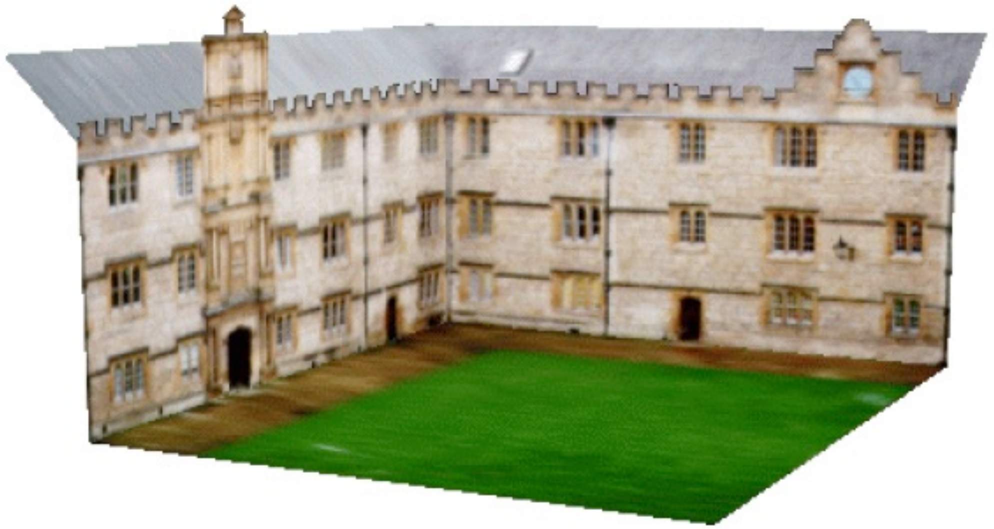


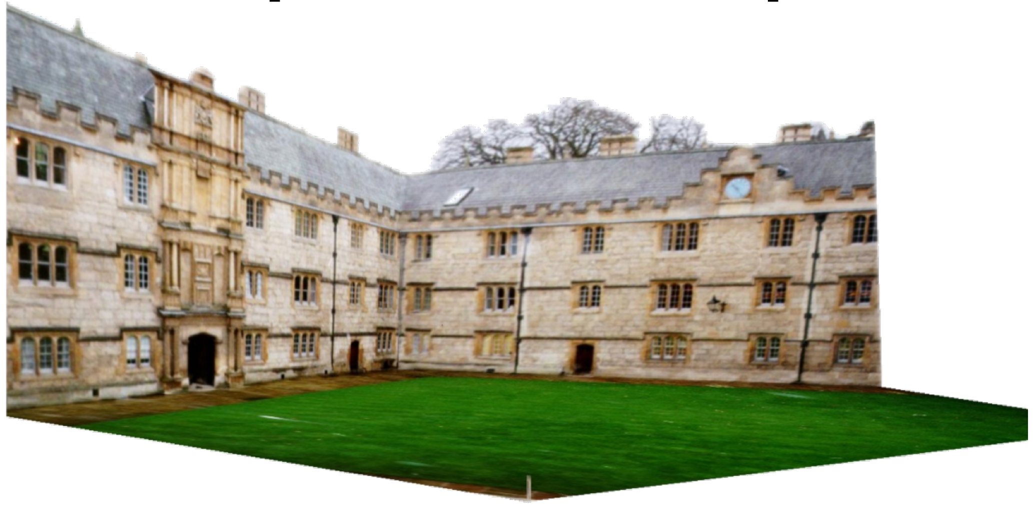
Input Image

Automatic Photo Pop-up

Comparison with Manual Method

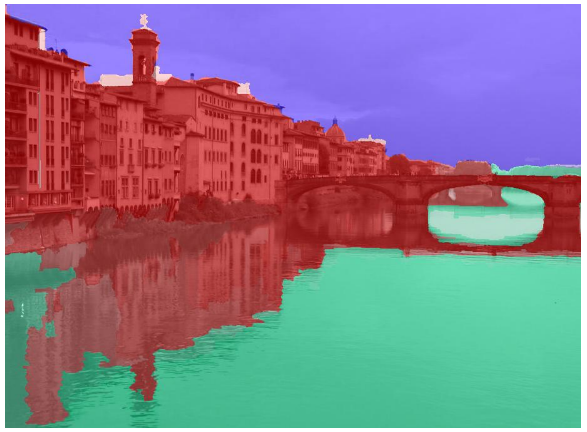

Input Image

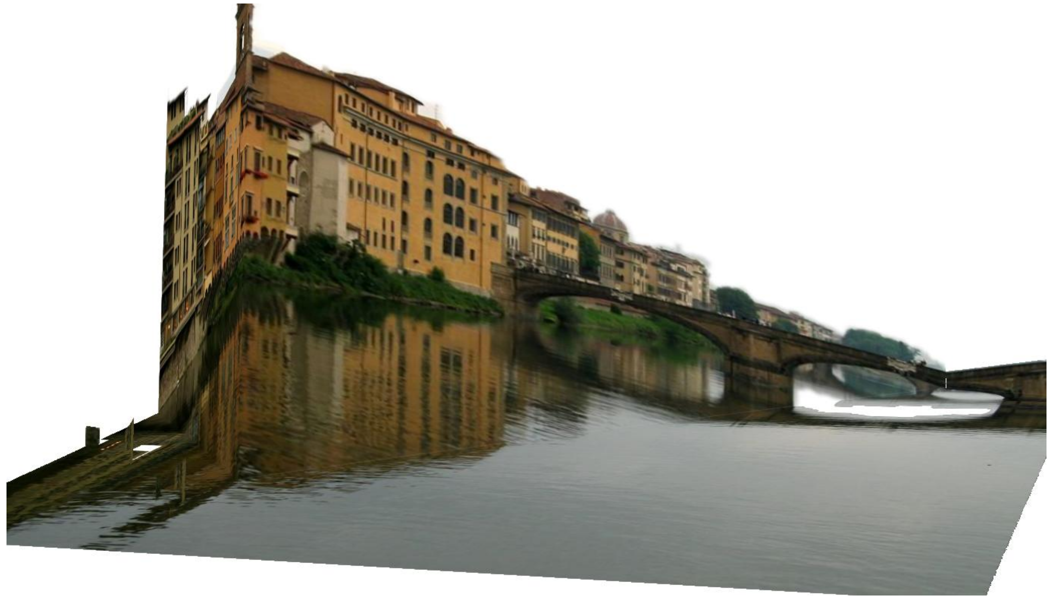

[Liebowitz et al. 1999]


Automatic Photo Pop-up (15 sec)!

Failures

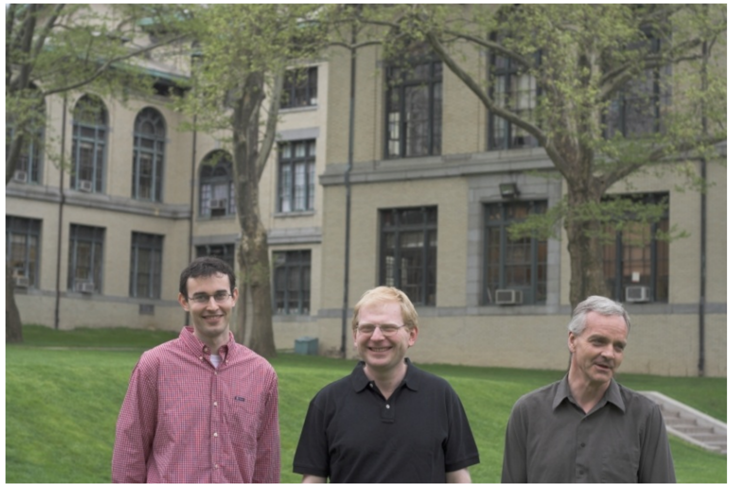
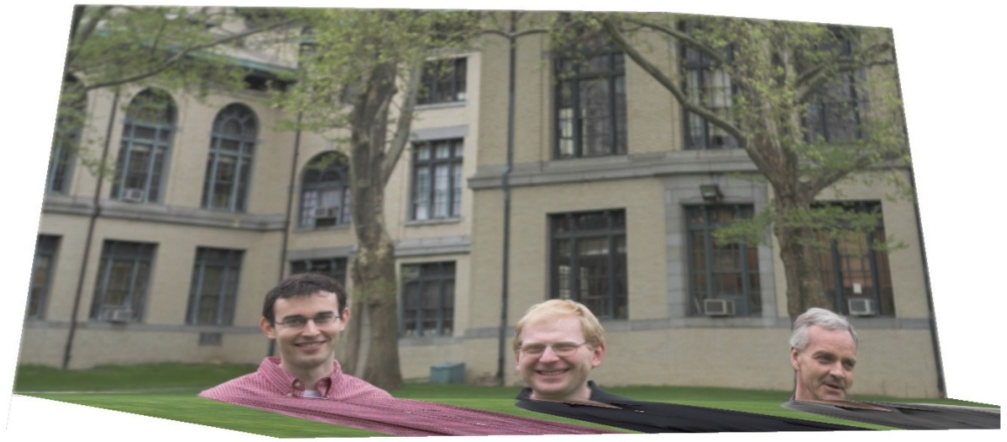
Labeling Errors

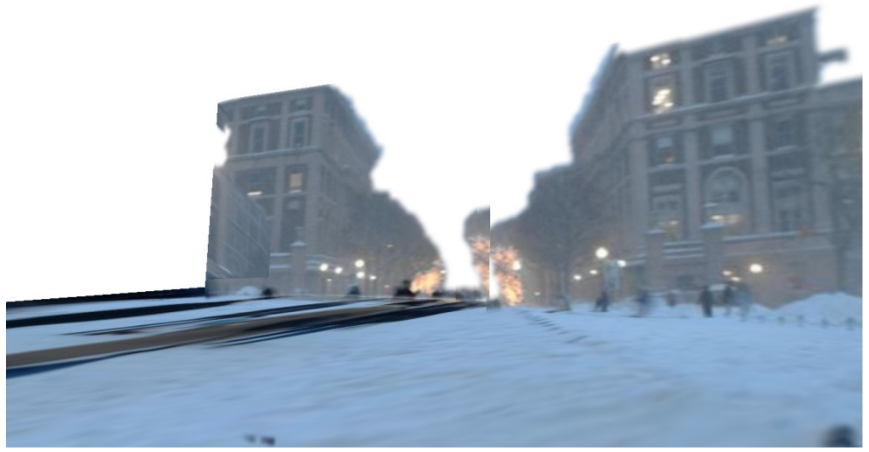




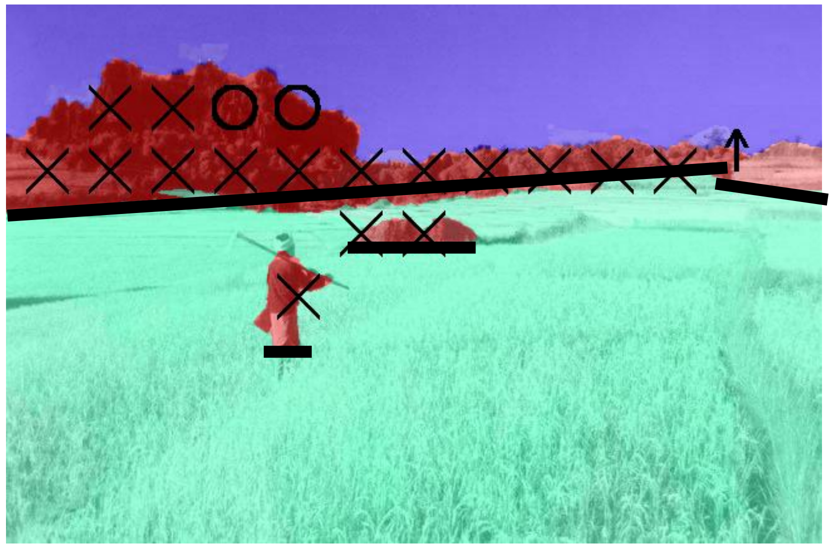
Failures

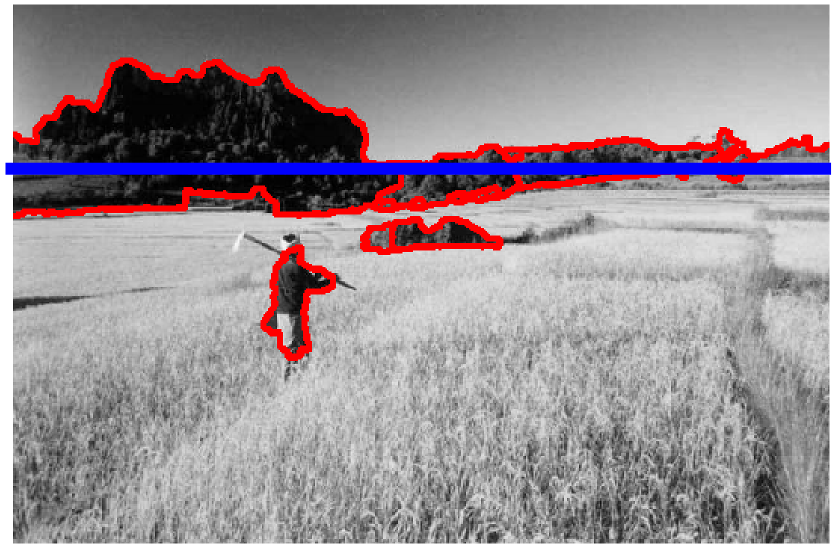
Foreground Objects

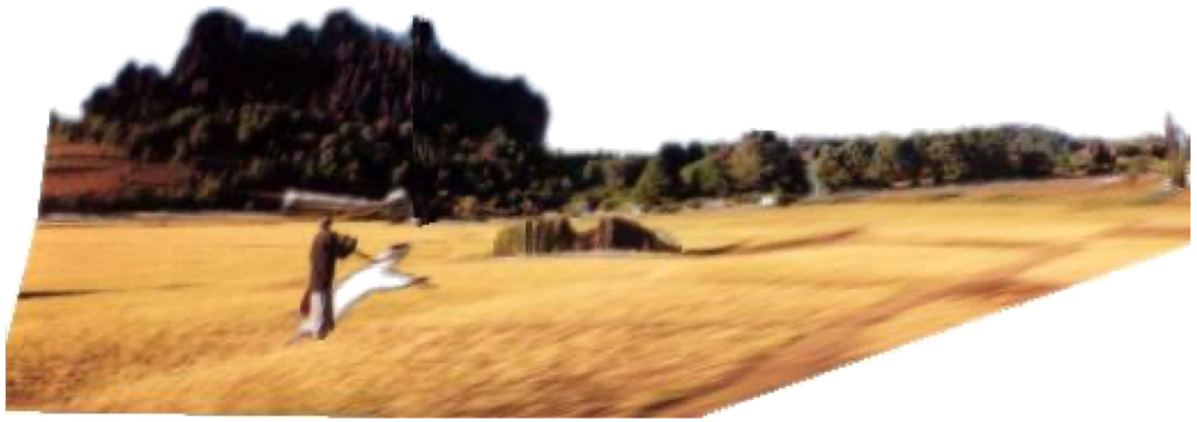





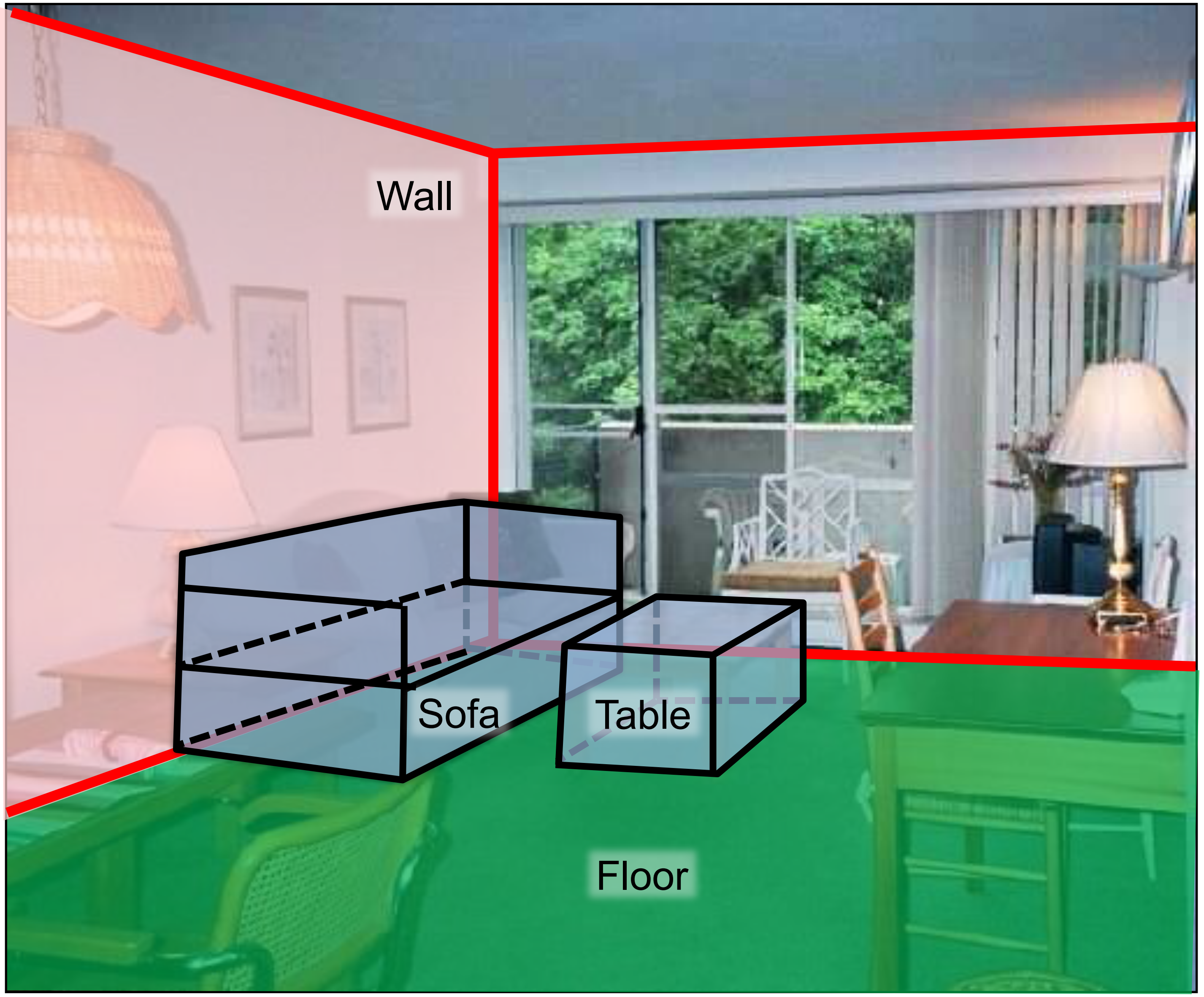
Adding Foreground Labels


Recovered Surface Labels +
Ground-Vertical Boundary Fit


Object Boundaries + Horizon

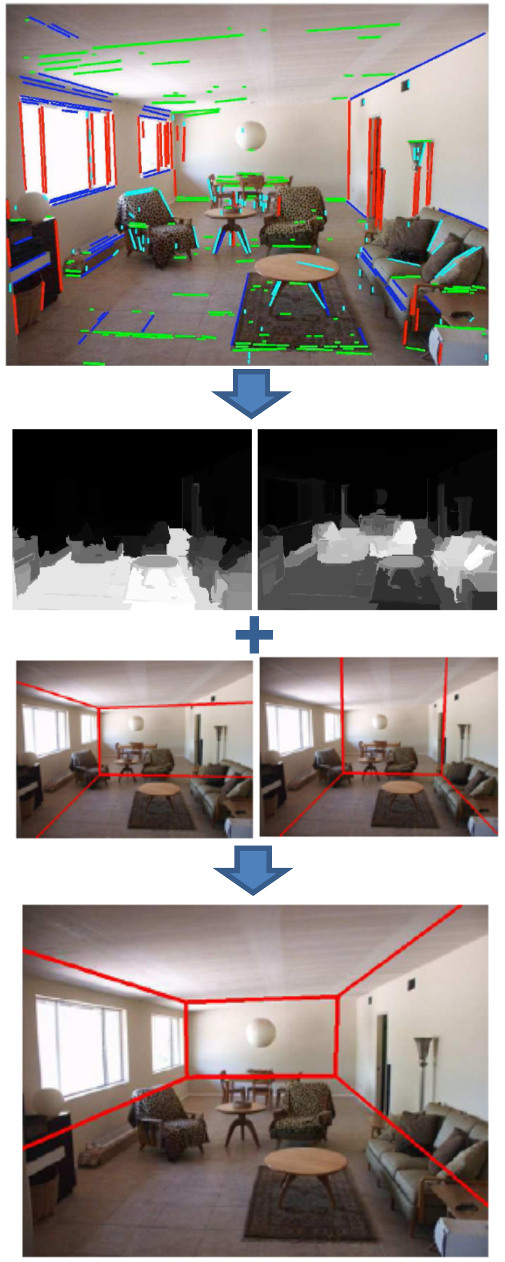


Fitting boxes to indoor scenes



Hedau et al. 2010

Box Layout Algorithm



1. Detect edges

2. Estimate 3 orthogonal vanishing points

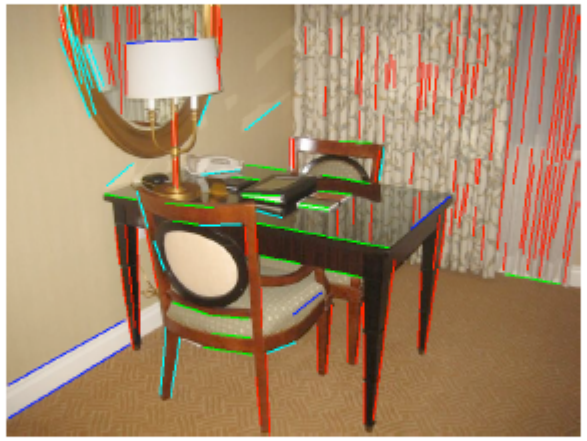
3. Apply region classifier to label pixels with visible surfaces
   - Boosted decision trees on region based on color, texture, edges, position

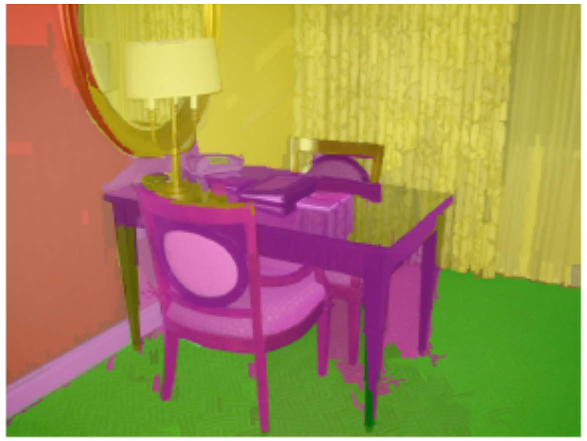
4. Generate box candidates by sampling pairs of rays from VPs

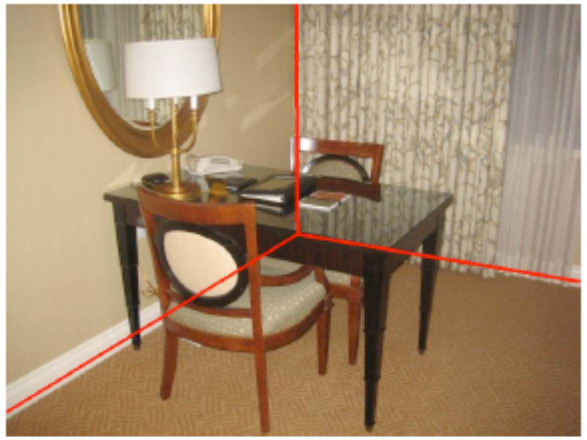
5. Score each box based on edges and pixel labels
   - Learn score via structured learning

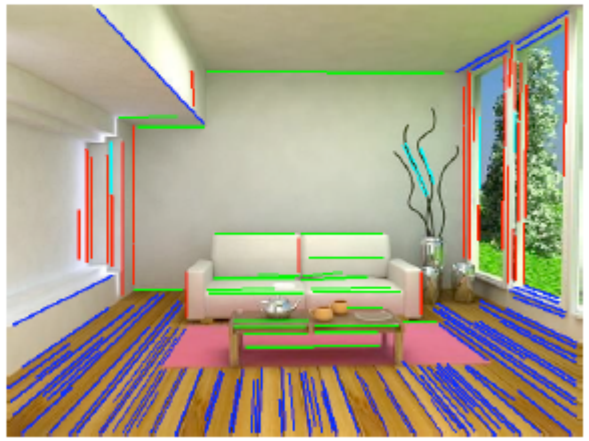
6. Jointly refine box layout and pixel labels to get final estimate

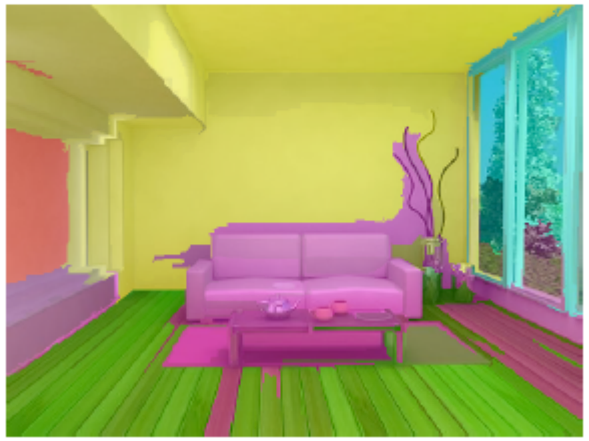
Experimental results

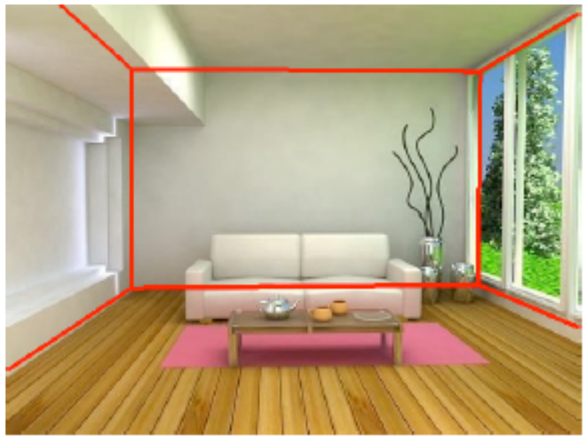

Detected Edges


Surface Labels

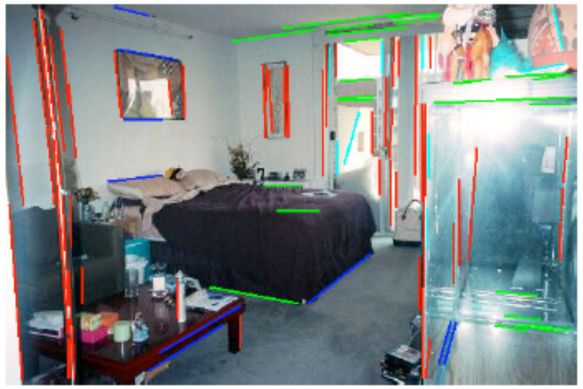

Box Layout

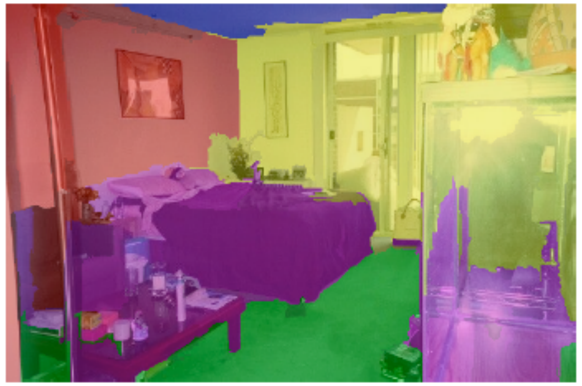

Detected Edges

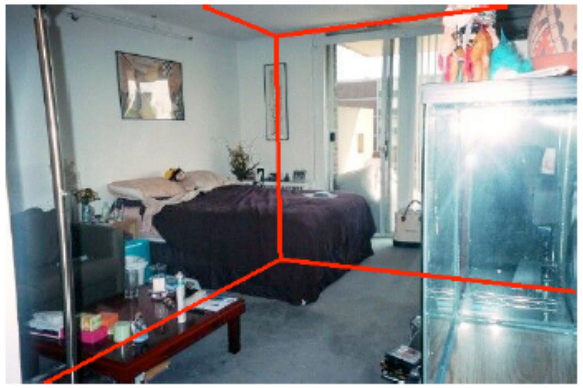

Surface Labels

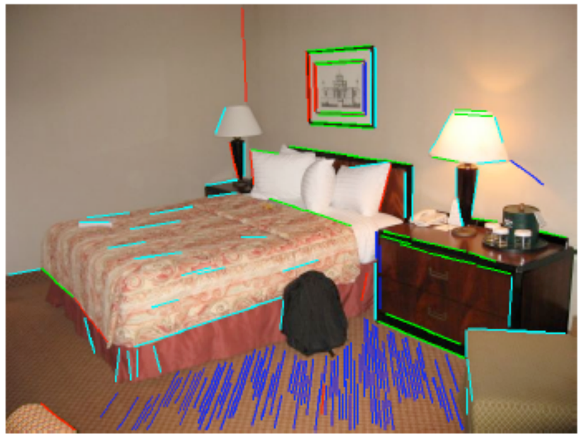

Box Layout

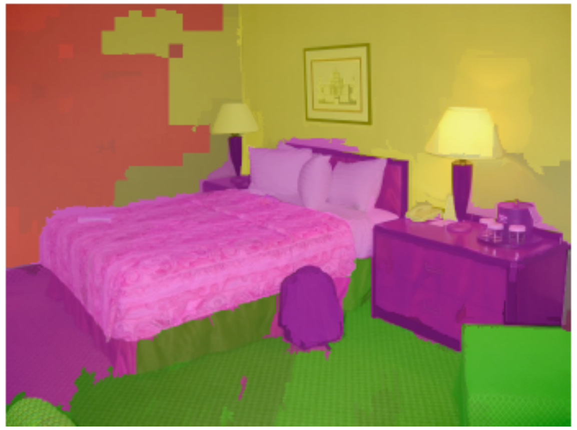
Experimental results

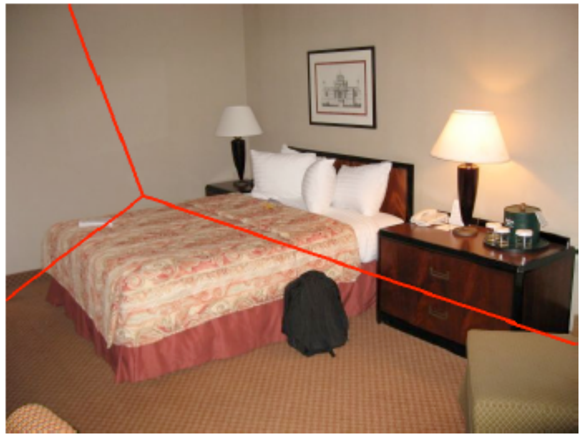

Detected Edges


Surface Labels

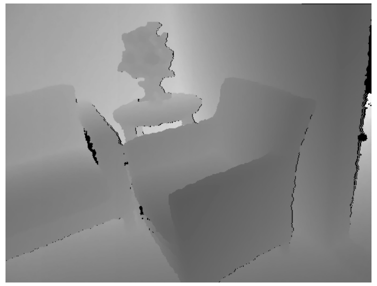

Box Layout

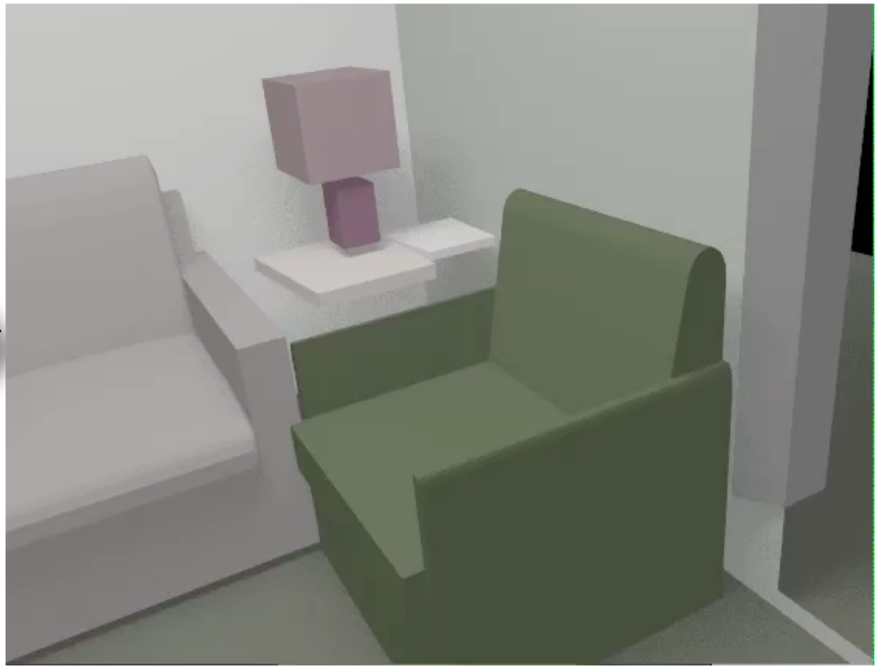

Detected Edges


Surface Labels


Box Layout

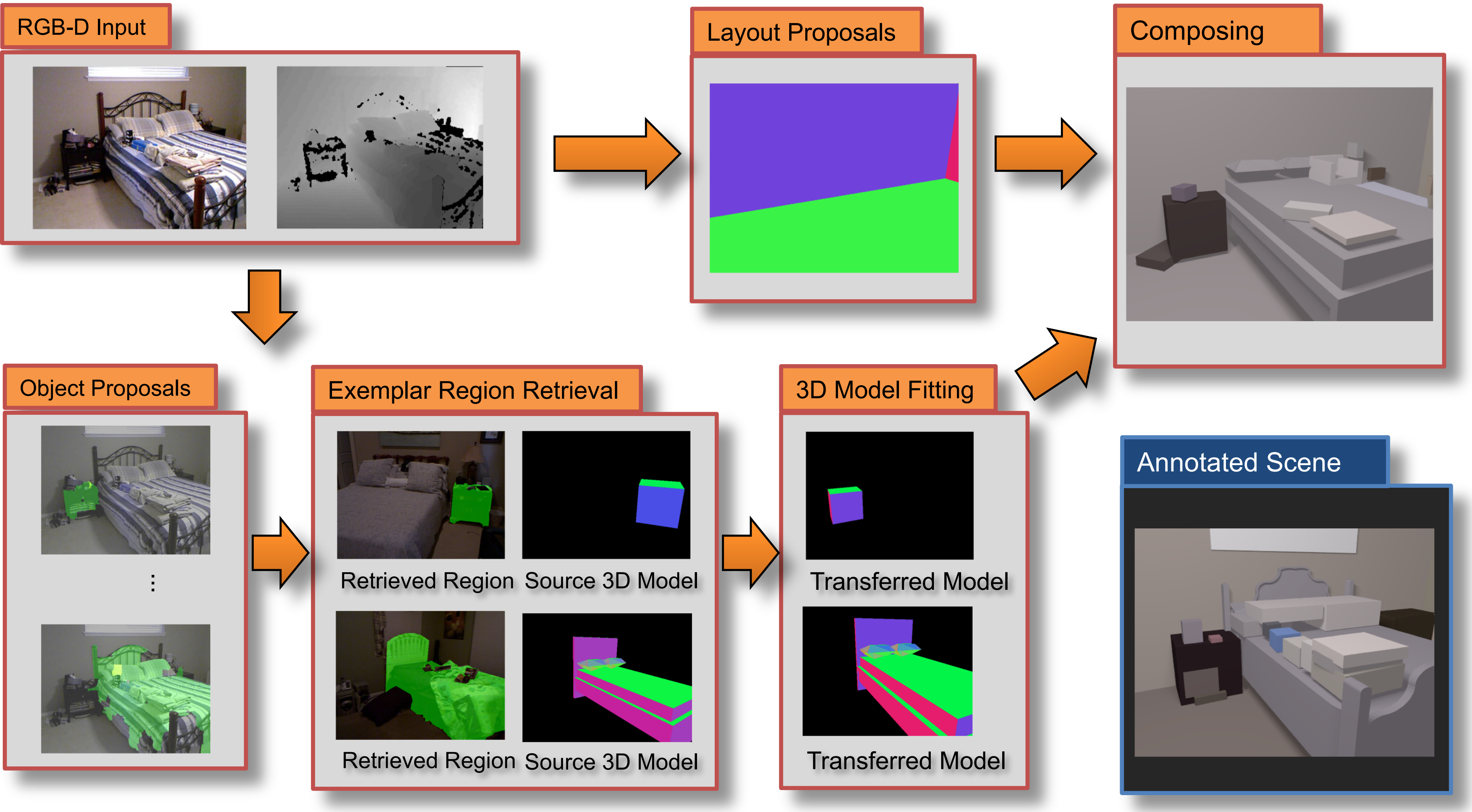
Complete 3D from RGBD



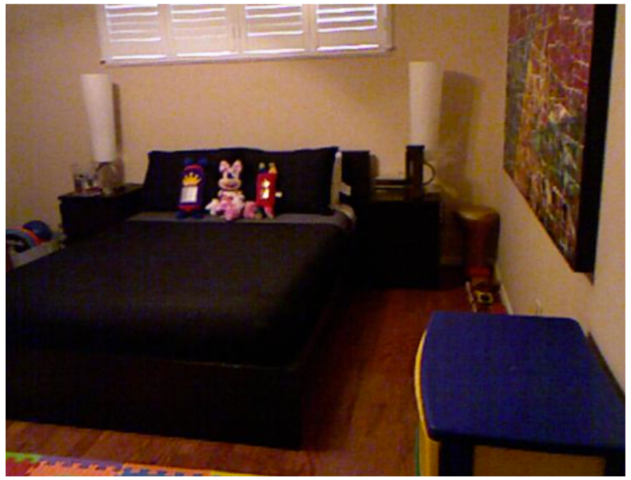


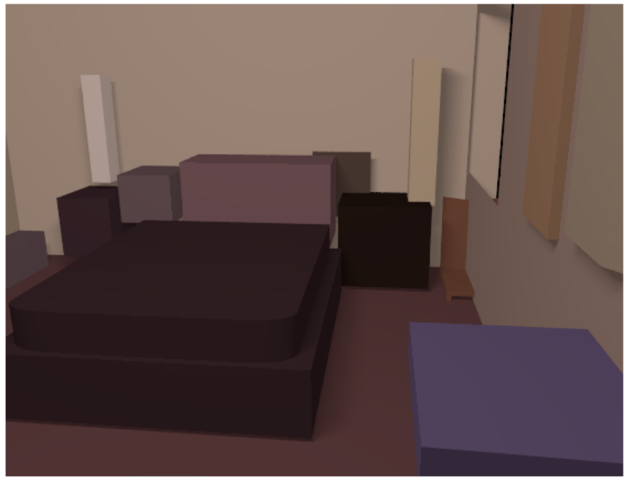
Zou et al. IJCV 2018

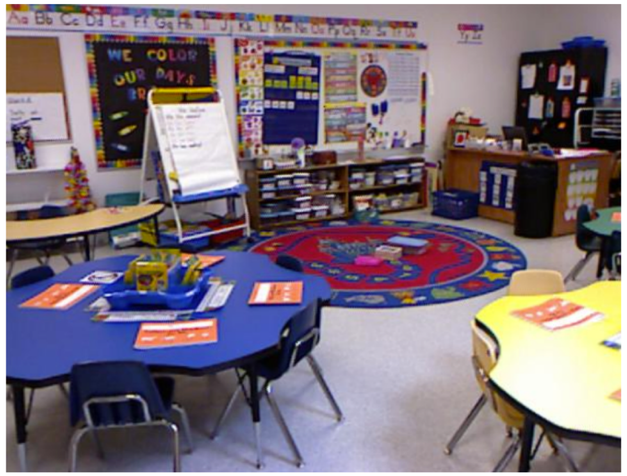
Complete 3D from RGBD

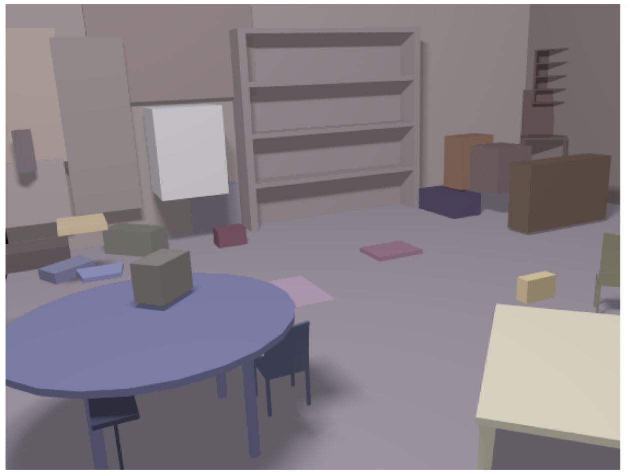


Complete 3D from RGBD









Final project ideas

- Interactive program to make 3D model from an image (e.g., output in VRML, or draw path for animation)
  - Add tools for cutting out foreground objects and automatic hole-filling

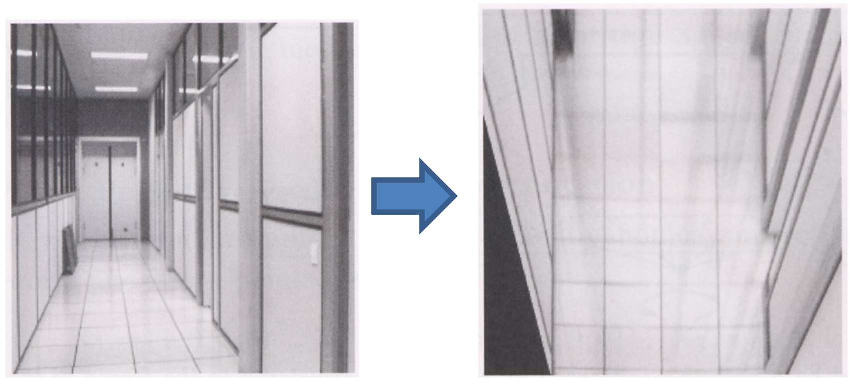
Summary

- 2D→3D is mathematically impossible

  (but we do it without even thinking)

- Need right assumptions about the world geometry

- Important tools
  - Vanishing points
  - Camera matrix
  - Homography



Next Week

- Good luck with the midterm

- Next three classes: image-based lighting
  - How to model light
  - Recover HDR image from multiple LDR images
  - Recover lighting model from an image
  - Render object into a scene with correct lighting and geometry