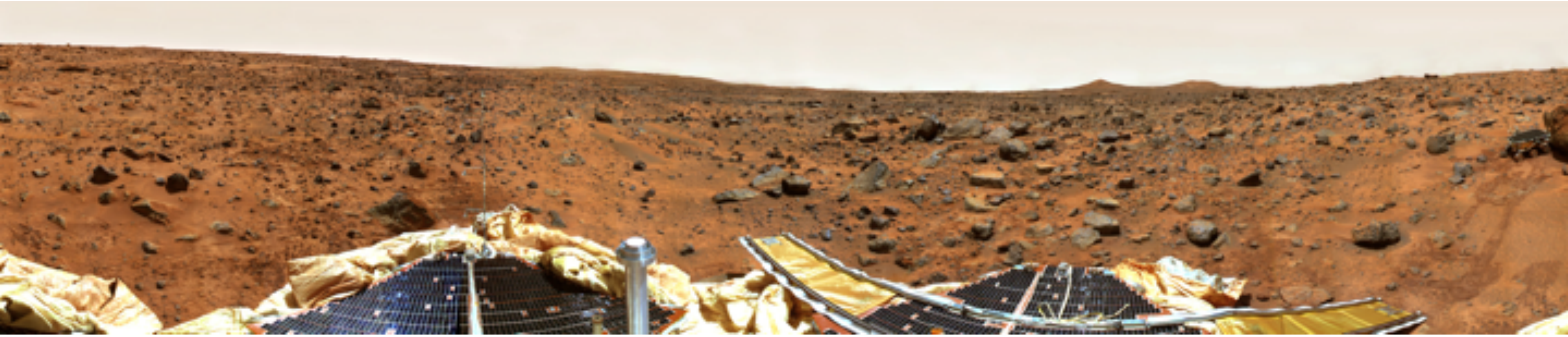
CS6670: Computer Vision

Noah Snavely

Lecture 15: Single-view modeling

Project 3 Overview



- Due Oct 18
- Teams of 2 students
- Concepts covered in lectures 12 (Ransac), 15 (Panoramas)

stitch2.txt

```
// Warp two of the half-resolution input images
// usage: project2 sphrWarp input.tga output.tga f [k1 k2]
Panorama sphrWarp pano1_0008.tga warp08.tga 595 -0.15 0.0
Panorama sphrWarp pano1_0009.tga warp09.tga 595 -0.15 0.0

// Generate features for the two images
Features computeFeatures warp08.tga warp08.f
Features computeFeatures warp09.tga warp09.f

// Match features (using ratio test)
Features matchFeatures warp08.f warp09.f 0.8 match-08-09.txt 2
// OR
Features matchSIFTFeatures warp08.sift warp09.sift 0.8 match-08-09.txt 2

// Align the pairs using feature matching:
Panorama alignPair warp08.f warp09.f match-08-09.txt 200 1
// OR
Panorama alignPair warp08.f warp09.f match-08-09.txt 200 1 sift
// ** NOTE: if using SIFT features and matches for debugging, use:
// Panorama alignPair warp08.key warp09.key match-08-09.txt 200 1 sift

// Finally, blend these two images together
// usage: project2 blendPairs pairlist.txt outfile.tga blendWidth
// assume the output from previous command was saved in pairlist2.txt
Panorama blendPairs pairlist2.txt stitch2.tga 200
```

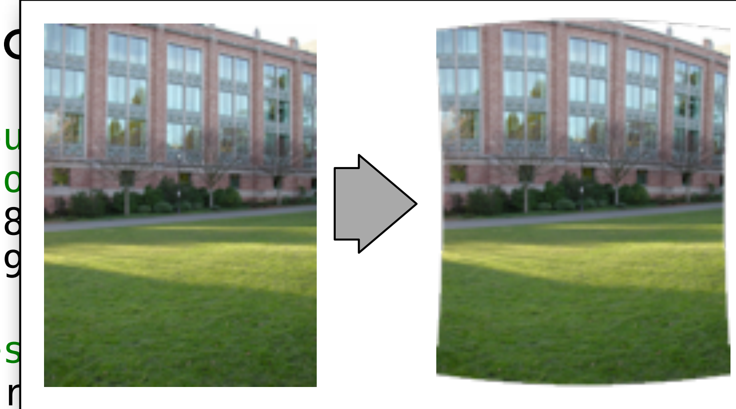
TODO 1

P2

TODO 2

TODO 3

stitc



```
// Warp two of the half-resolution inpu
// usage: project2 sphrWarp input.tga o
Panorama sphrWarp pano1_0008.tga warp08
Panorama sphrWarp pano1_0009.tga warp09

// Generate features for the two images
Features computeFeatures warp08.tga war
Features computeFeatures warp09.tga warp09.f

// Match features (using ratio test)
Features matchFeatures warp08.f warp09.f 0.8 match-08-09.txt 2
// OR
Features matchSIFTFeatures warp08.sift warp09.sift 0.8 match-08-09.txt 2

// Align the pairs using feature matching:
Panorama alignPair warp08.f warp09.f match-08-09.txt 200 1
// OR
Panorama alignPair warp08.f warp09.f match-08-09.txt 200 1 sift
// ** NOTE: if using SIFT features and matches for debugging, use:
// Panorama alignPair warp08.key warp09.key match-08-09.txt 200 1 sift

// Finally, blend these two images together
// usage: project2 blendPairs pairlist.txt outfile.tga blendWidth
// assume the output from previous command was saved in pairlist2.txt
Panorama blendPairs pairlist2.txt stitch2.tga 200
```

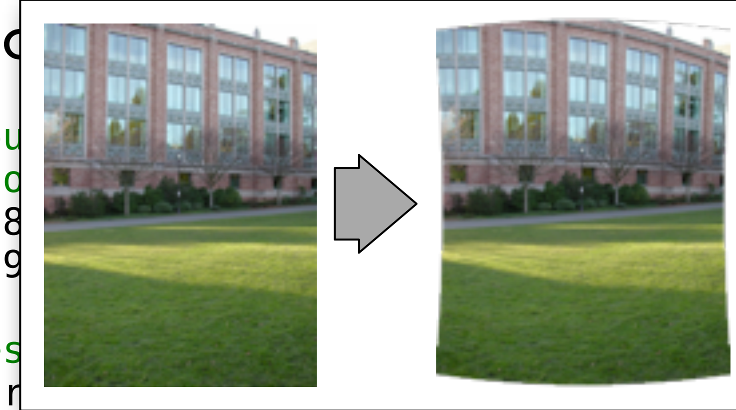
TODO 1

P2

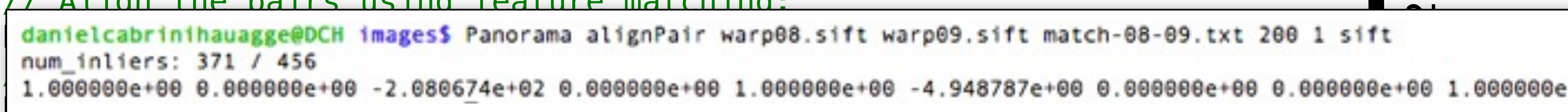
TODO 2

TODO 3

stitc



```
// Warp two of the half-resolution inpu
// usage: project2 sphrWarp input.tga c
Panorama sphrWarp pano1_0008.tga warp08
Panorama sphrWarp pano1_0009.tga warp09

// Generate features for the two images
Features computeFeatures warp08.tga war
Features computeFeatures warp09.tga warp09.f

// Match features (using ratio test)
Features matchFeatures warp08.f warp09.f 0.8 match-08-09.txt 2
// OR
Features matchSIFTFeatures warp08.sift warp09.sift 0.8 match-08-09.txt 2

// Align the pairs using feature matching:
```




```
// ** NOTE: if using SIFT features and matches for debugging, use:
// Panorama alignPair warp08.key warp09.key match-08-09.txt 200 1 sift

// Finally, blend these two images together
// usage: project2 blendPairs pairlist.txt outfile.tga blendWidth
// assume the output from previous command was saved in pairlist2.txt
Panorama blendPairs pairlist2.txt stitch2.tga 200
```

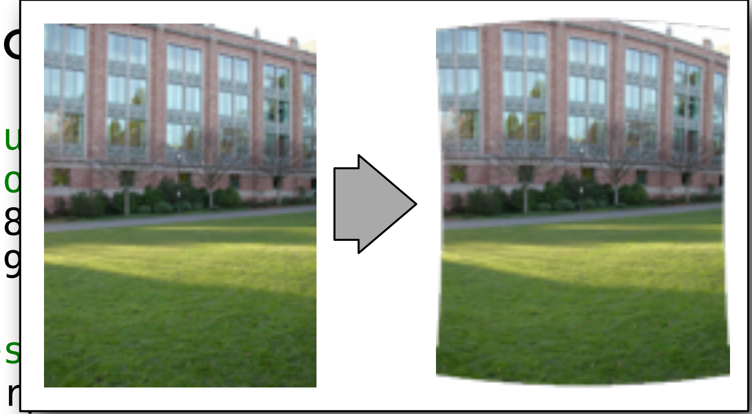
TODO 1

P2

TO

TODO 3

stitc



```
// Warp two of the half-resolution inpu
// usage: project2 sphrWarp input.tga o
Panorama sphrWarp pano1_0008.tga warp08
Panorama sphrWarp pano1_0009.tga warp09

// Generate features for the two images
Features computeFeatures warp08.tga war
Features computeFeatures warp09.tga warp09.f

// Match features (using ratio test)
Features matchFeatures warp08.f warp09.f 0.8 match-08-09.txt 2
// OR
Features matchSIFTFeatures warp08.sift warp09.sift 0.8 match-08-09.txt 2

// Align the pairs using feature matching:
```

```
danielcabrinihauagge@DCH images$ Panorama alignPair warp08.sift warp09.sift match-08-09.txt 200 1 sift
num inliers: 371 / 456
1.000000e+00 0.000000e+00 -2.080674e+02 0.000000e+00 1.000000e+00 -4.948787e+00 0.000000e+00 0.000000e+00 1.000000e
```

```
// ** NOTE: if using SIFT features and matches for debugging, use:
// Panorama alignPair warp08.key warp09.key match-08-09.txt 200 1 sift

// Finally, blend these two images together
// usage: project2 blendPairs pairlist.txt outfile.tga blendWidth
// assume the output from previous command was saved in pairlist2.txt
Panorama blendPairs pairlist2.txt stitch2.tga 200
```

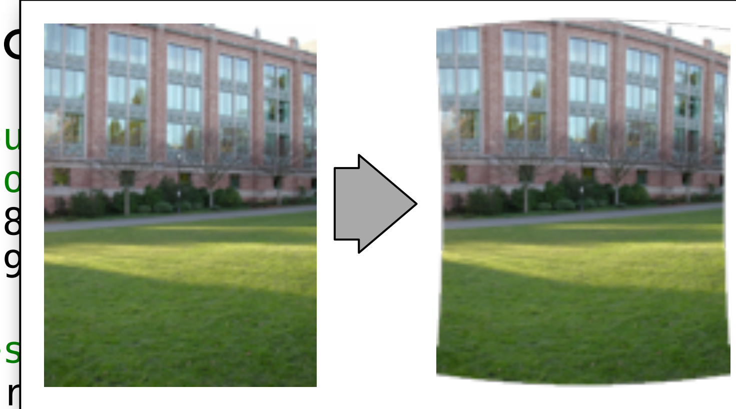
TODO 1

P2

T

TODO 3

stitc



```
// Warp two of the half-resolution inpu
// usage: project2 sphrWarp input.tga c
Panorama sphrWarp pano1_0008.tga warp08
Panorama sphrWarp pano1_0009.tga warp09

// Generate features for the two images
Features computeFeatures warp08.tga war
Features computeFeatures warp09.tga warp09.f

// Match features (using ratio test)
Features matchFeatures warp08.f warp09.f 0.8 match-08-09.txt 2
// OR
Features matchSIFTFeatures warp08.sift warp09.sift 0.8 match-08-09.txt 2

// Align the pairs using feature matching:
```

```
danielcabrinihauagge@DCH images$ Panorama alignPair warp08.sift warp09.sift match-08-09.txt 200 1 sift
num inliers: 371 / 456
1.000000e+00 0.000000e+00 -2.080674e+02 0.000000e+00 1.000000e+00 -4.948787e+00 0.000000e+00 0.000000e+00 1.000000e
```

```
// ** NOTE: if using SIFT features and matches for debugging, use:
// Panorama alignPair warp08.key warp09.key match-08-09.txt 200 1 sift

// Finally, blend these two images together
// usage: project2 blendPairs pairlist.txt outfile.tga blendWidth
// assume the output from previous command was saved in pairlist2.txt
Panorama blendPairs pairlist2.txt stitch2.tga 200
```

TODO 1

P2

TO

TODO 3

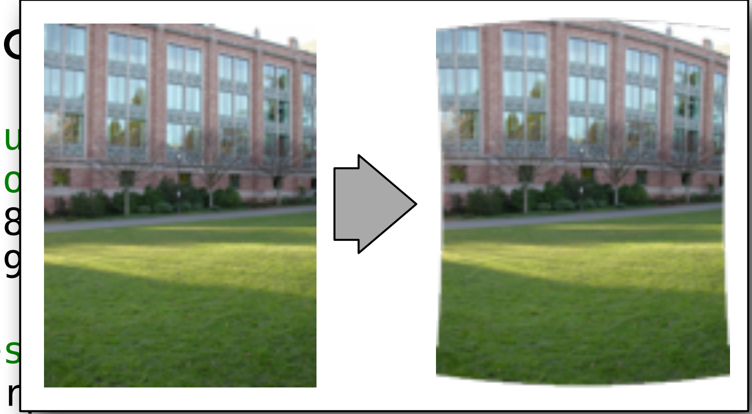

```
warp08.tga warp09.tga 1.000000e+00 0.000000e+00 -2.080708e+02 0.000000e+00 0.00000
```

stitc

```
// Warp two of the half-resolution inpu
// usage: project2 sphrWarp input.tga o
Panorama sphrWarp pano1_0008.tga warp08
Panorama sphrWarp pano1_0009.tga warp09

// Generate features for the two images
Features computeFeatures warp08.tga war
Features computeFeatures warp09.tga warp09.f

// Match features (using ratio test)
Features matchFeatures warp08.f warp09.f 0.8 match-08-09.txt 2
// OR
Features matchSIFTFeatures warp08.sift warp09.sift 0.8 match-08-09.txt 2

// Align the pairs using feature matching:
```

```
danielcabrinihauagge@DCH images$ Panorama alignPair warp08.sift warp09.sift match-08-09.txt 200 1 sift
num inliers: 371 / 456
1.000000e+00 0.000000e+00 -2.080674e+02 0.000000e+00 1.000000e+00 -4.948787e+00 0.000000e+00 0.000000e+00 1.000000e
```

```
// ** NOTE: if using SIFT features and ma
// Panorama alignPair warp08.key warp09.

// Finally, blend these two images toget
// usage: project2 blendPairs pairlist.tx
// assume the output from previous comma
Panorama blendPairs pairlist2.txt stitch2
```



TODO 1

P2

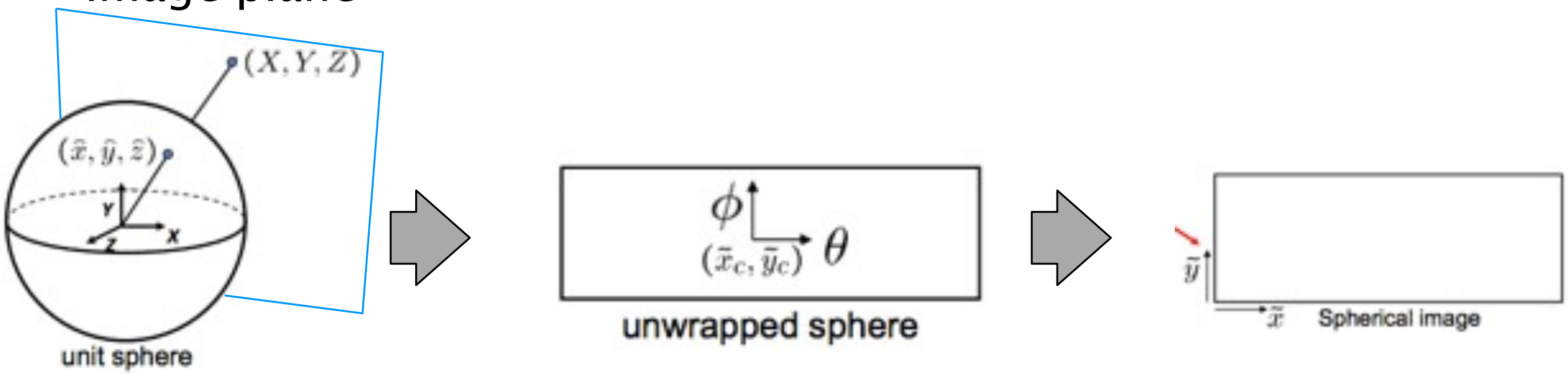
TODO 2

TODO 3

```
warp08.tga warp09.tga 1.000000e+00 0        000e+00 0.00000
```

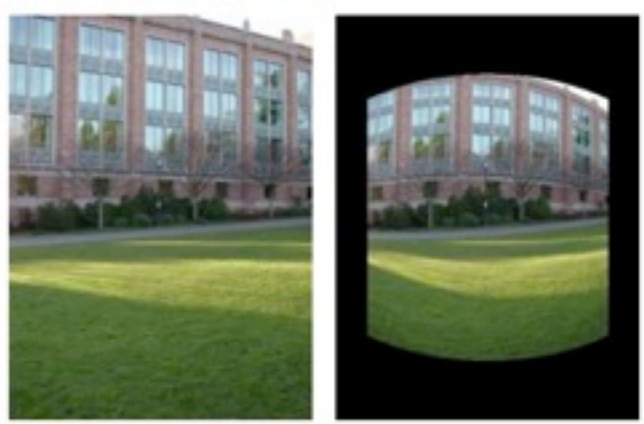
TODO1: Spherical Warping

Image plane



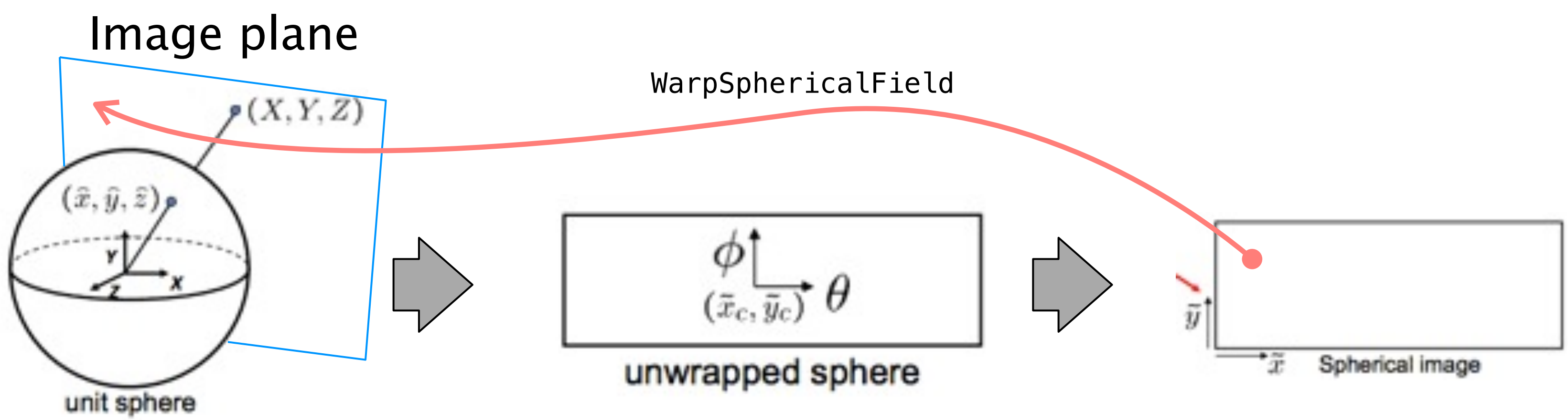
You will need the camera focal length here

Latitude vs Longitude



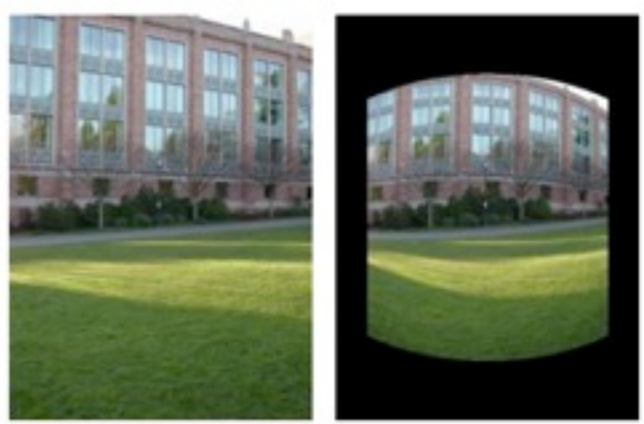
input f = 200 (pixels)

TODO1: Spherical Warping



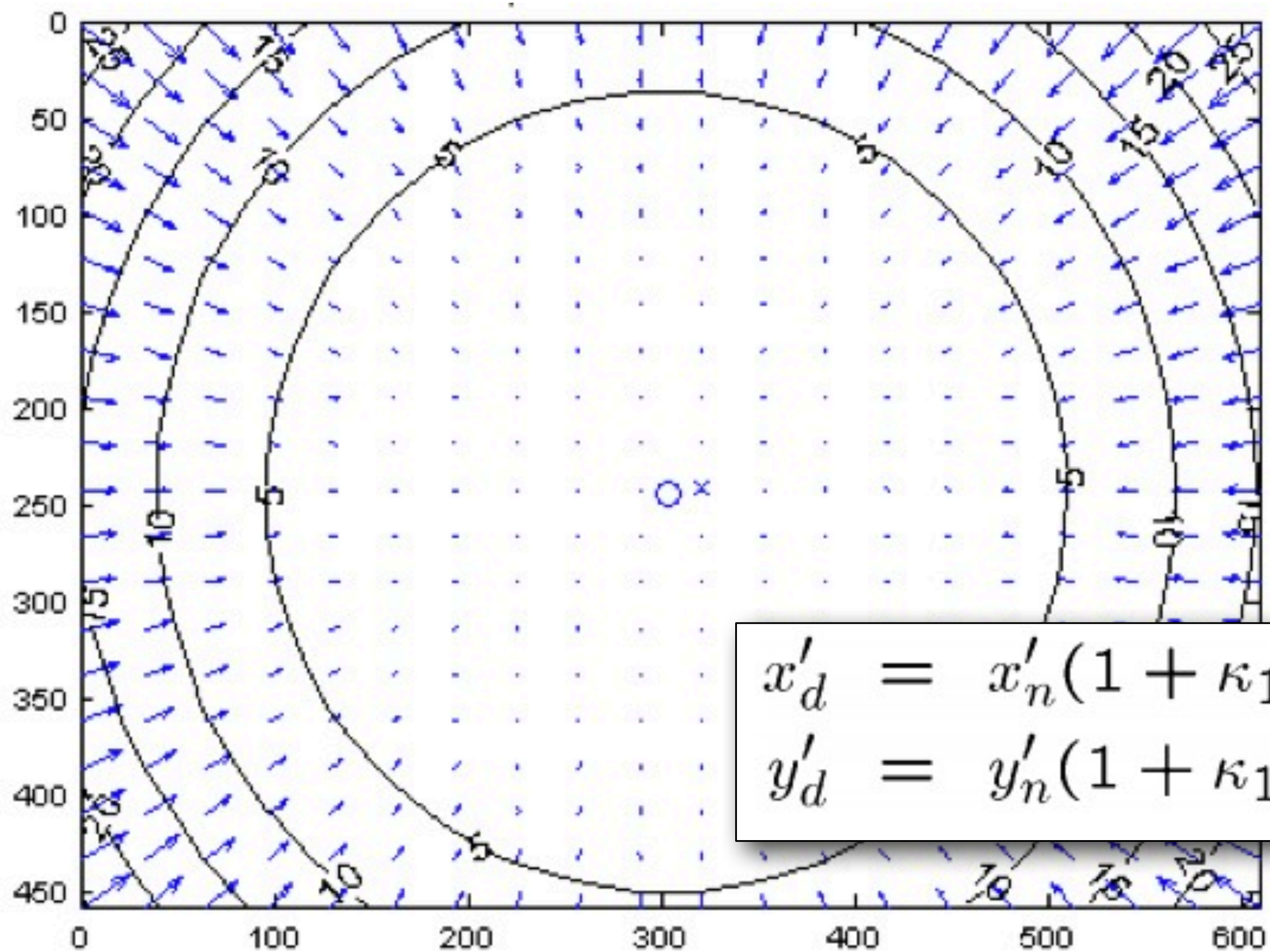
You will need the camera focal length here

Latitude vs Longitude



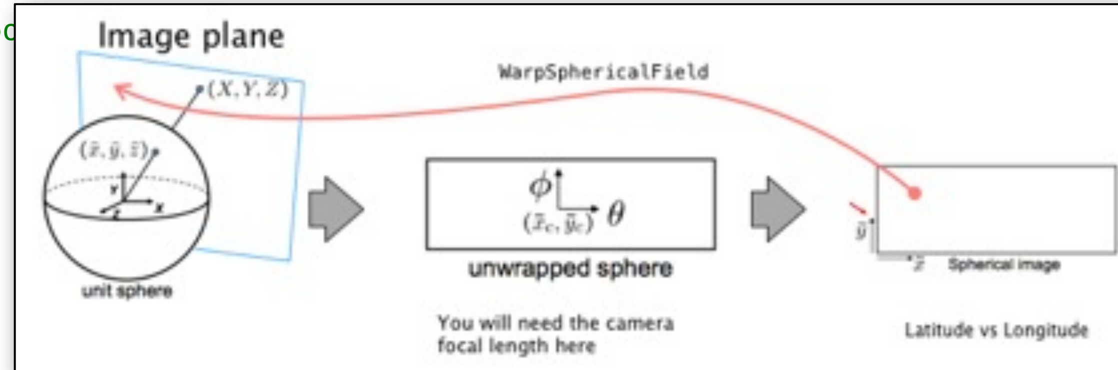
input f = 200 (pixels)

TODO1: Radial Distortion



$$
\begin{aligned}
x'_d &= x'_n(1+\kappa_1 r^2 + \kappa_2 r^4) \\
y'_d &= y'_n(1+\kappa_1 r^2 + \kappa_2 r^4)
\end{aligned}
$$

```
CFloatImage WarpSphericalField(CShape srcSh, CShape dstSh, float f,
                               float k1, float k2, const CTransform3x3 &r)
{
  // Set up the pixel coordinate image
  dstSh.nBands = 2;
  CFloatImage uvImg(dstSh);   // (u,v) coo

  CVector3 p;

  p[0] = sin(0.0) * cos(0.0);
  p[1] = sin(0.0);
  p[2] = cos(0.0) * cos(0.0);
  p = r * p;
  double min_y = p[1];

  // Fill in the values
  for (int y = 0; y < dstSh.height; y++) {
      float *uv = &uvImg.Pixel(0, y, 0);
      for (int x = 0; x < dstSh.width; x++, uv += 2) {
           // (x,y) is the spherical image coordinates.
           // (xf,yf) is the spherical coordinates, e.g., xf is the angle theta
           // and yf is the angle phi

           float xf = (float) ((x - 0.5f*dstSh.width ) / f);
           float yf = (float) ((y - 0.5f*dstSh.height) / f - min_y);

           // (xt,yt,zt) are intermediate coordinates to which you can
           // apply the spherical correction and radial distortion
           float xt, yt;
           CVector3 p;

           // BEGIN TODO

           // END TODO

           // Convert back to regular pixel coordinates and store
           float xn = 0.5f*srcSh.width  + xt*f;
           float yn = 0.5f*srcSh.height + yt*f;
           uv[0] = xn;
           uv[1] = yn;
```



TODO2: RANSAC

```
procedure RANSAC

n_inliers_best := 0
for nRANSAC rounds do
{
    p := random subset of points
    m := fit model using points p
    n_inliers := count inliers given model m

    if n_inliers > n_inliers_best
    {
        n_inliers_best := n_inliers
        m_best := m
    }
}

m_final := least squares fit of m with all
          inliers to m_best
```

TODO2: RANSAC

```
procedure RANSAC

n_inliers_best := 0
for nRANSAC rounds do
{
    p := random subset of points
    m := fit model using points p
    n_inliers := count inliers given model m

    if n_inliers > n_inliers_best
    {
        n_inliers_best := n_inliers
        m_best := m
    }
}

m_final := least squares fit of m with all
           inliers to m_best
```

```
int alignPair(const FeatureSet &f1, cons
              const vector<FeatureMatch>
              int nRANSAC, double RANSAC
```

TODO2: RANSAC

```
procedure RANSAC

n_inliers_best := 0
for nRANSAC rounds do
{
    p := random subset of points
    m := fit model using points p
    n_inliers := count inliers given model m

    if n_inliers > n_inliers_best
    {
        n_inliers_best := n_inliers
        m_best := m
    }
}

m_final := least squares fit of m with all
          inliers to m_best
```

```
int alignPair(const FeatureSet &f1, cons
              const vector<FeatureMatch>
              int nRANSAC, double RANSAC
```

```
ComputeHomography(const FeatureSet &f1,
                  const vector<FeatureMa
```

TODO2: RANSAC

```
procedure RANSAC

n_inliers_best := 0
for nRANSAC rounds do
{
    p := random subset of points
    m := fit model using points p
    n_inliers := count inliers given model m

    if n_inliers > n_inliers_best
    {
        n_inliers_best := n_inliers
        m_best := m
    }
}

m_final := least squares fit of m with all
          inliers to m_best
```

```
int alignPair(const FeatureSet &f1, cons
              const vector<FeatureMatch>
              int nRANSAC, double RANSA

ComputeHomography(const FeatureSet &f1,
                  const vector<FeatureMa

int countInliers(const FeatureSet &f1, c
                 const vector<FeatureMat
                 CTransform3x3 M, double
```

TODO2: RANSAC

```
procedure RANSAC

n_inliers_best := 0
for nRANSAC rounds do
{
    p := random subset of points
    m := fit model using points p
    n_inliers := count inliers given model m

    if n_inliers > n_inliers_best
    {
        n_inliers_best := n_inliers
        m_best := m
    }
}

m_final := least squares fit of m with all
          inliers to m_best
```

```
int alignPair(const FeatureSet &f1, cons
              const vector<FeatureMatch>
              int nRANSAC, double RANSA
```

```
ComputeHomography(const FeatureSet &f1,
                  const vector<FeatureMa
```

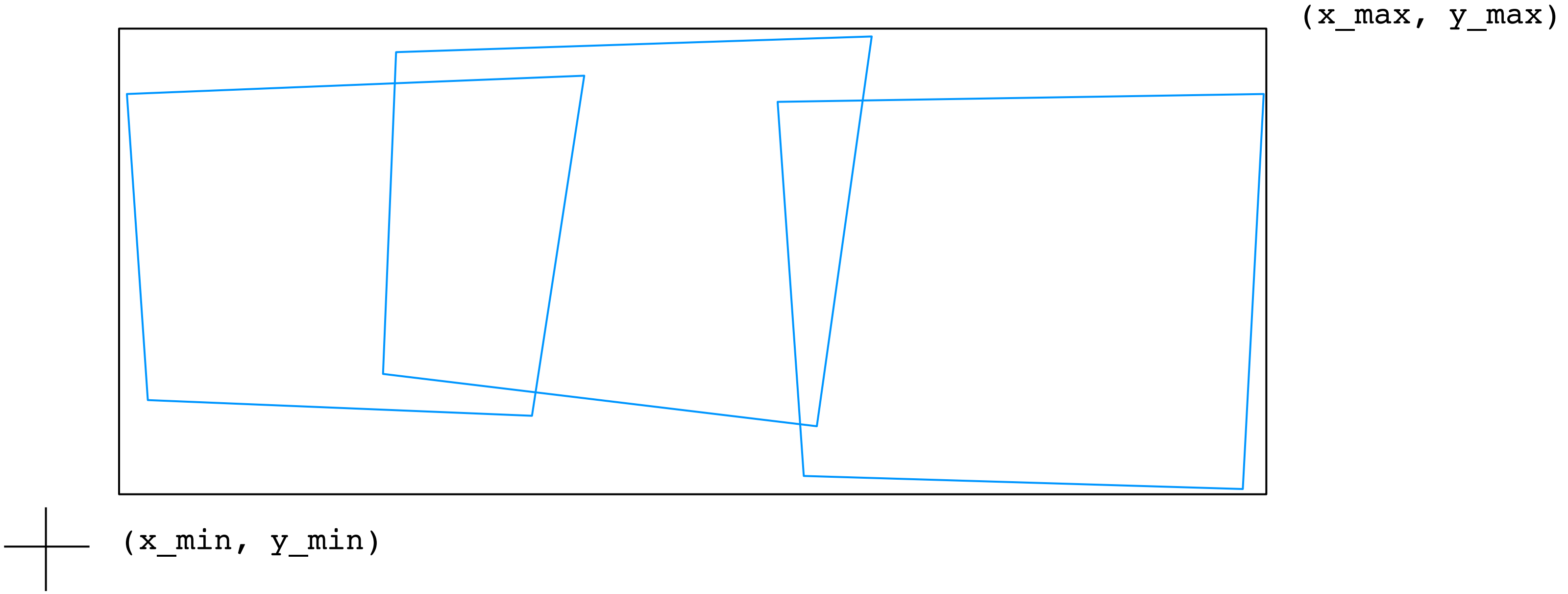
```
int countInliers(const FeatureSet &f1, c
                 const vector<FeatureMat
                CTransform3x3 M, double
```

```
int leastSquaresFit(const FeatureSet &f
                    const vector<Featur
                    const vector<int> &
```

TODO3: Image Blending

```
CByteImage BlendImages(CImagePositionV& ipv, float blendWidth)
```

Part1: Figure out the bounding box of the composite
Will have to reproject image corners using transforms

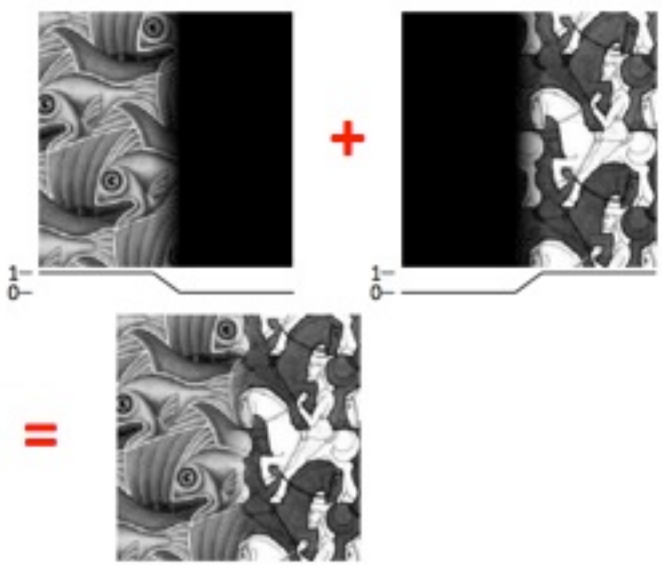


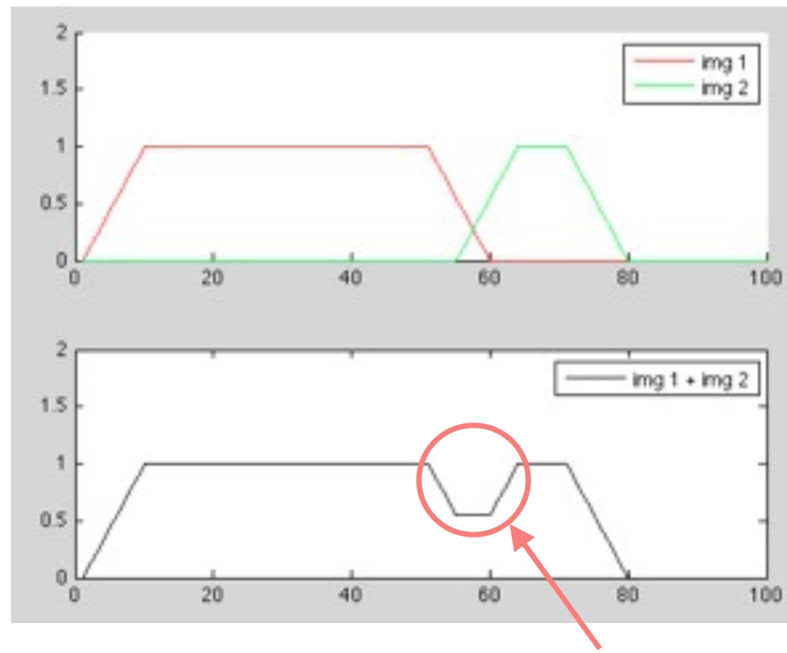
TODO3: Image Blending

```
static void AccumulateBlend(CByteImage& img, CFloatImage& acc, CTransform3x3 M, float blendWidth)
```

For linear interpolation of pixel values you can use the method

```
double CImageOf<T>::PixelLerp(double x, double y, int band)
```





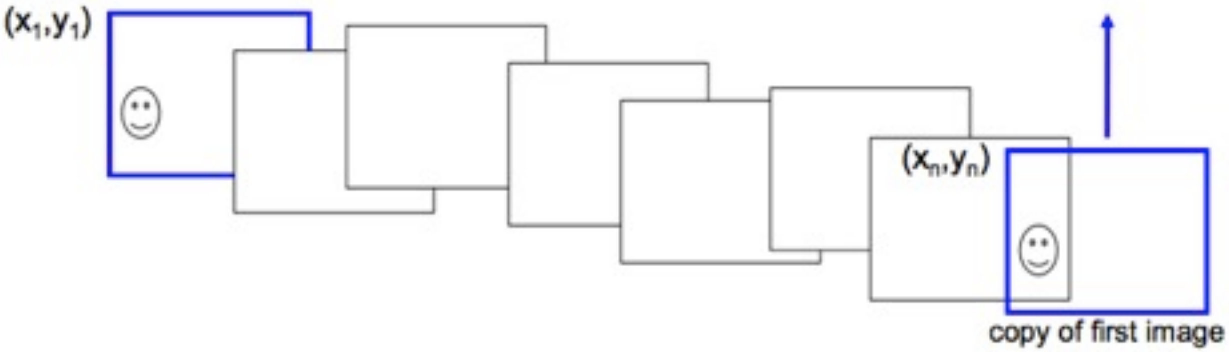
Dark strip in image

```
static void NormalizeBlend(CFloatImage& acc, CByteImage& img)
```

divides composite by total weight to get range values back to [0,1]

TODO3: Image Blending

- Final step: drift correction



- Solution
  - add another copy of first image at the end
  - this gives a constraint: $y_n = y_1$
  - there are a bunch of ways to solve this problem
    - add displacement of $(y_1 - y_n)/(n - 1)$ to each image after the first
    - **apply an affine warp: $y' = y + ax$ [you will implement this for P3]**
    - run a big optimization problem, incorporating this constraint
      - best solution, but more complicated
      - known as "bundle adjustment"

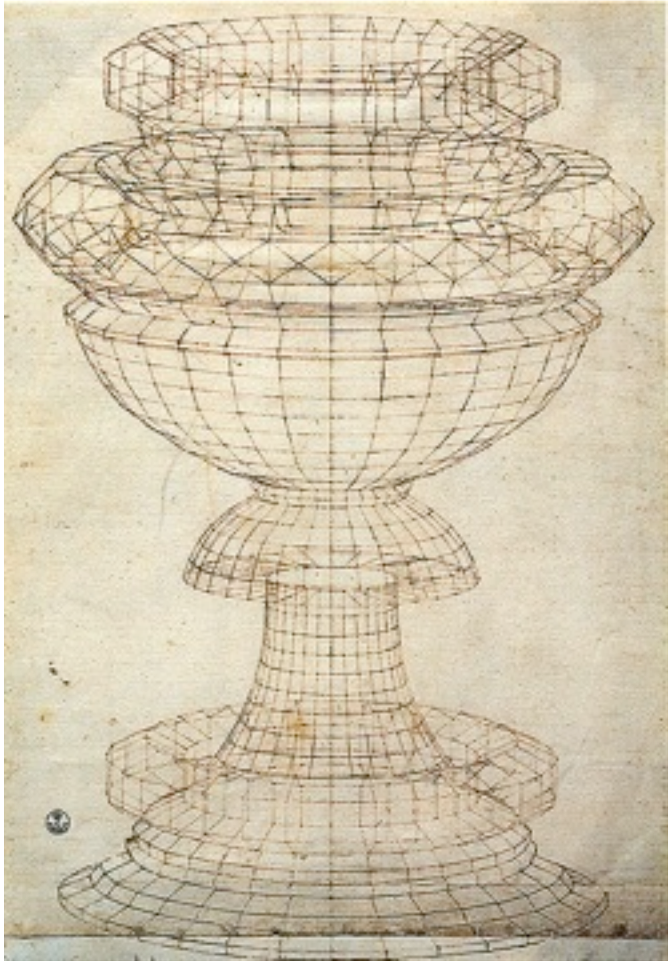
Projective geometry

Ames Room

- Readings
  - Mundy, J.L. and Zisserman, A., Geometric Invariance in Computer Vision, Appendix: Projective Geometry for Machine Vision, MIT Press, Cambridge, MA, 1992,
    (read 23.1 - 23.5, 23.10)
    - available online: http://www.cs.cmu.edu/~ph/869/papers/zisser-mundy.pdf

Projective geometry—what's it good for?

- Uses of projective geometry
  - Drawing
  - Measurements
  - Mathematics for projection
  - Undistorting images
  - Camera pose estimation
  - **Object recognition**



[Paolo Uccello]

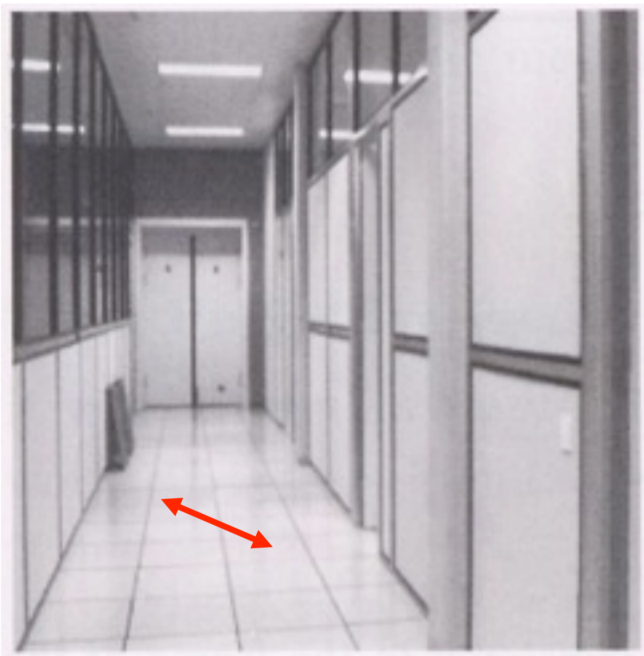
Applications of projective geometry

Vermeer's Music Lesson

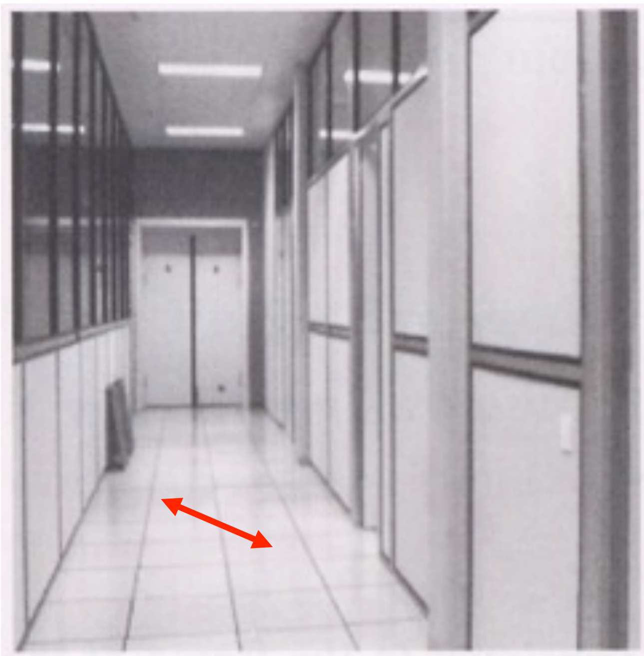
Reconstructions by Criminisi et al.

Measurements on planes

Measurements on planes

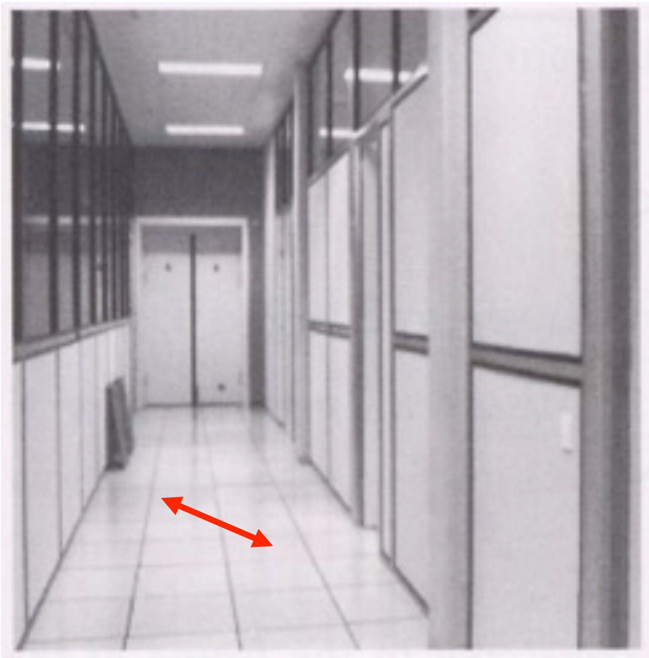


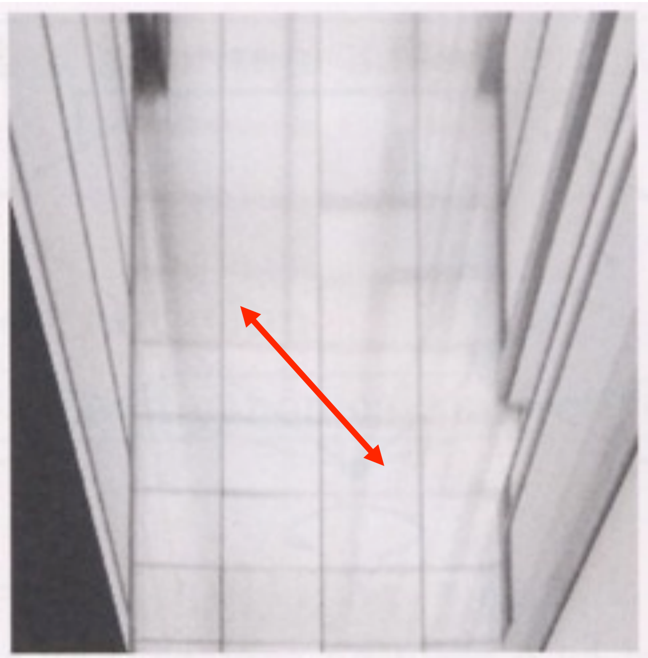
Measurements on planes



Approach: unwarp then measure

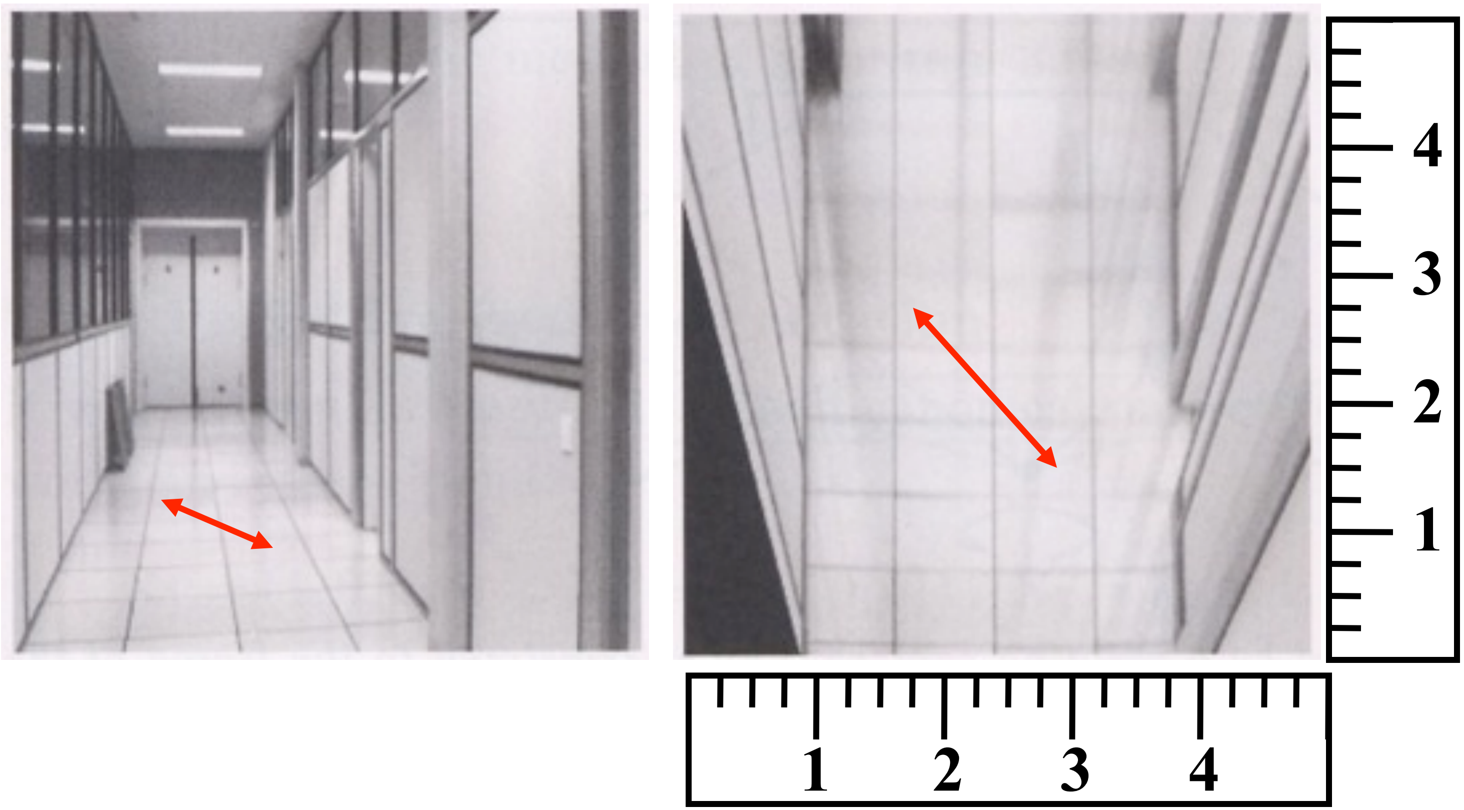
Measurements on planes





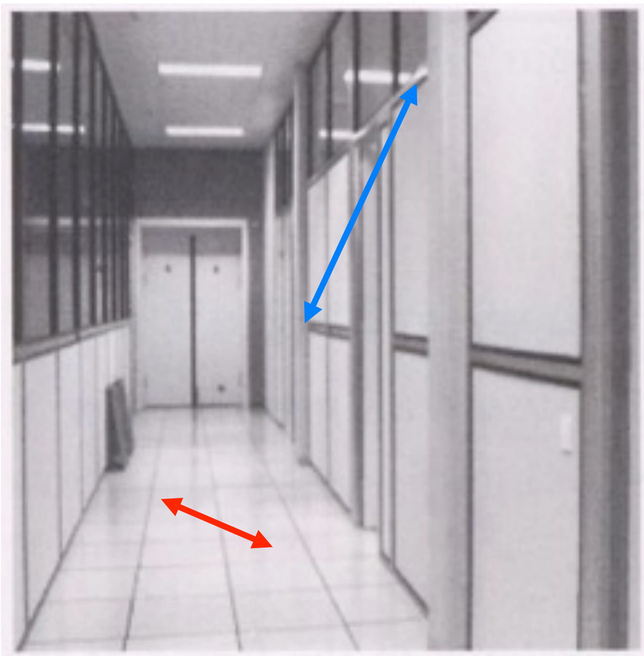
Approach: unwarp then measure

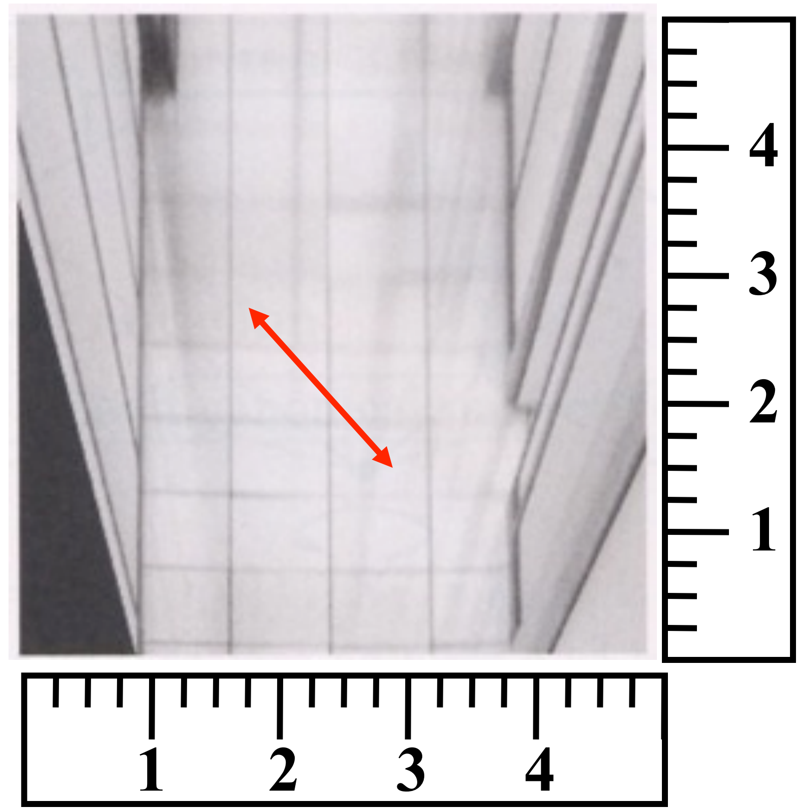
Measurements on planes



Approach: unwarp then measure

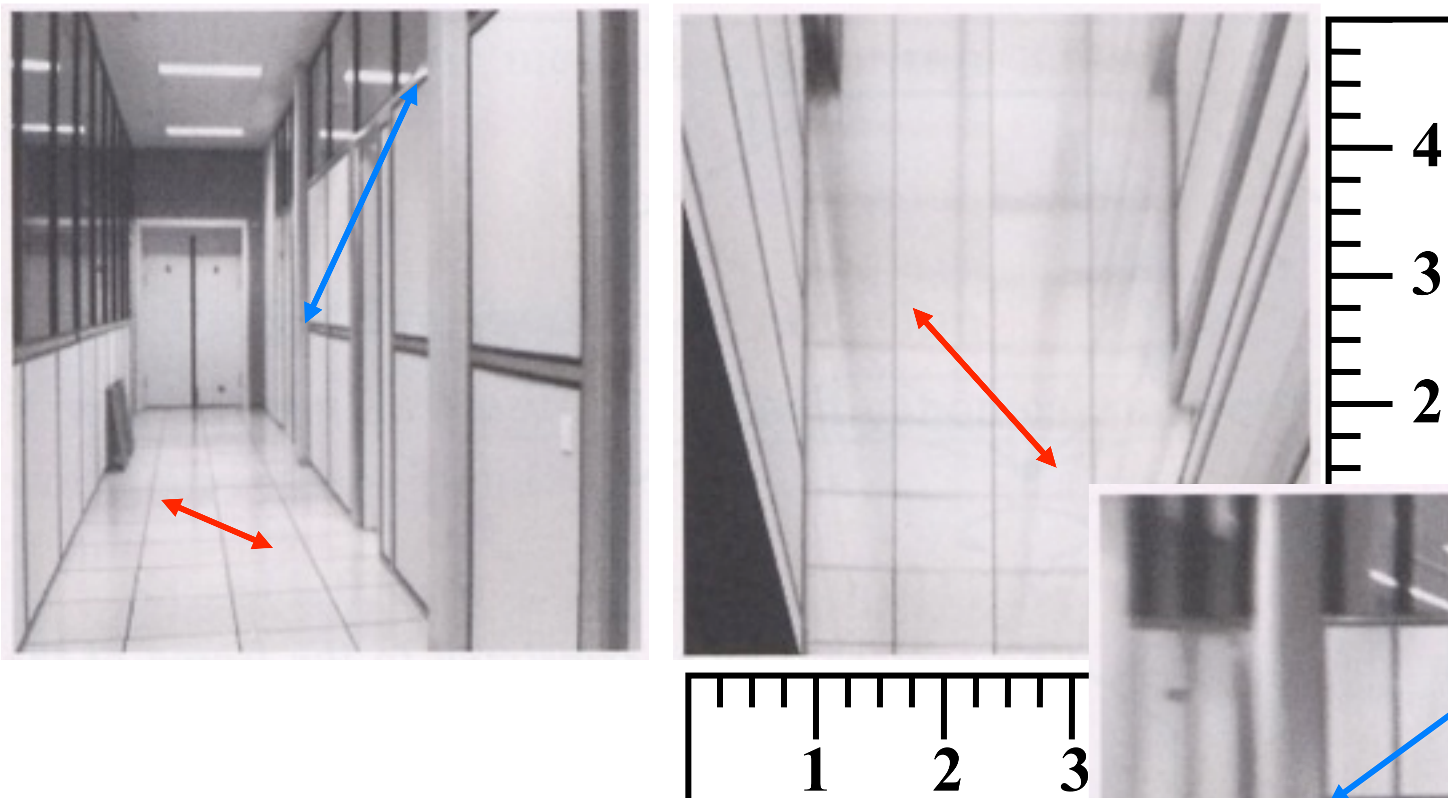
Measurements on planes

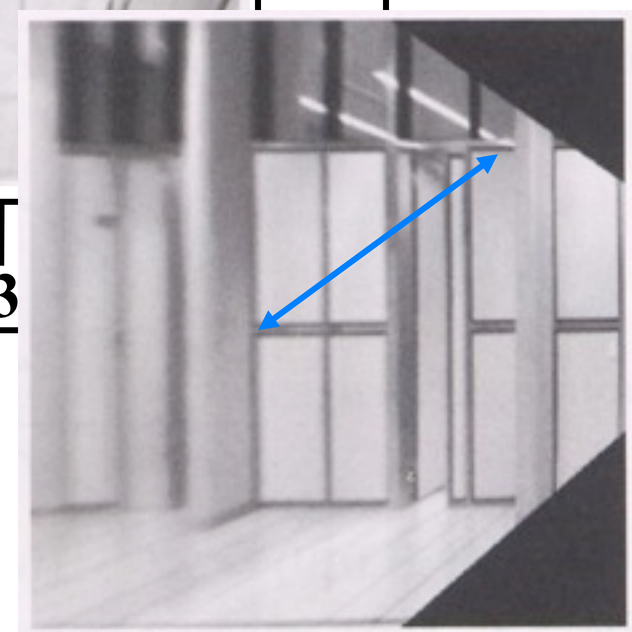




Approach: unwarp then measure

Measurements on planes

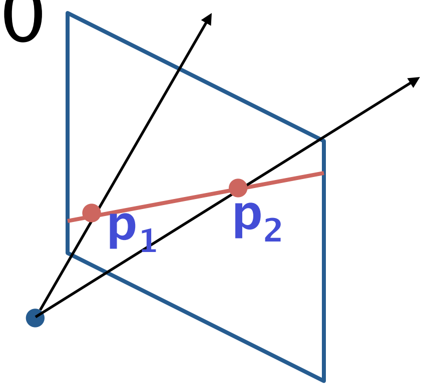




Approach: unwarp then measure

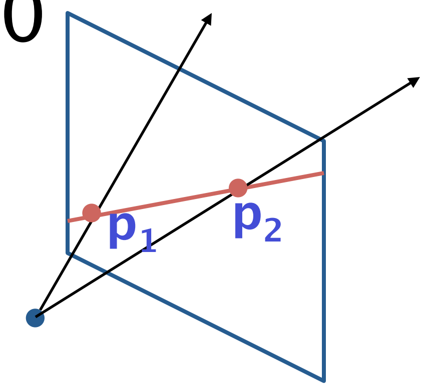
Point and line duality

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



Point and line duality

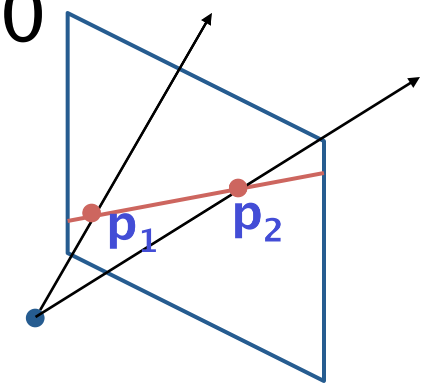
- A line **l** is a homogeneous 3-vector
- It is ⊥ to every point (ray) **p** on the line: **l p**=0



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

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: $\mathbf{l}\,\mathbf{p}=0$



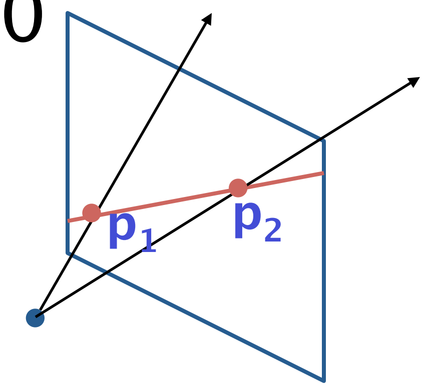
What is the line $\mathbf{l}$ spanned by rays $\mathbf{p}_1$ and $\mathbf{p}_2$?

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

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: $\mathbf{l}\,\mathbf{p}=0$

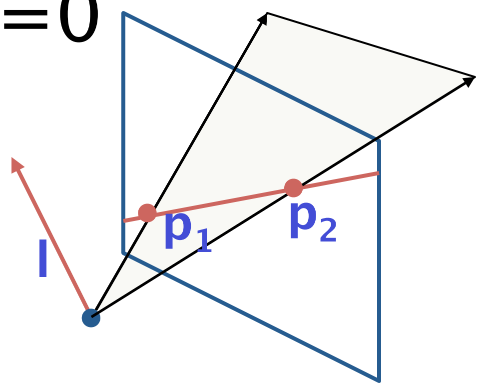


What is the line $\mathbf{l}$ spanned by rays $\mathbf{p}_1$ and $\mathbf{p}_2$?

- $\mathbf{l}$ is $\perp$ to $\mathbf{p}_1$ and $\mathbf{p}_2$ $\Rightarrow$ $\mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$
- $\mathbf{l}$ can be interpreted as a plane normal

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: $\mathbf{l}\,\mathbf{p}=0$

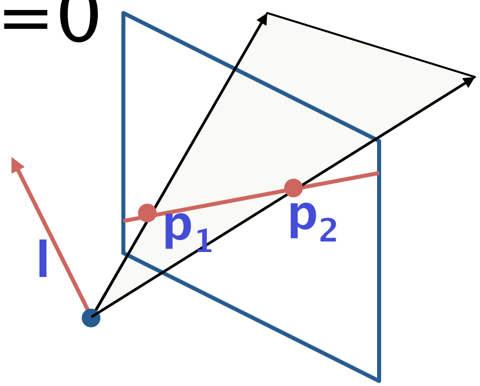


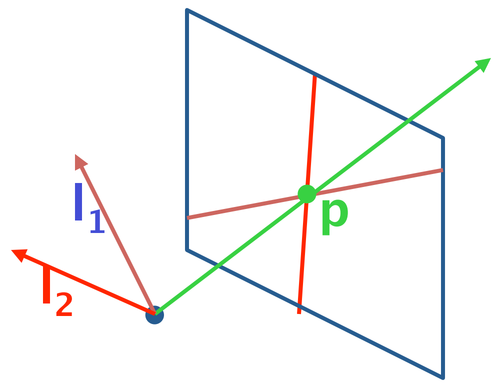
What is the line $\mathbf{l}$ spanned by rays $\mathbf{p}_1$ and $\mathbf{p}_2$?

- $\mathbf{l}$ is $\perp$ to $\mathbf{p}_1$ and $\mathbf{p}_2$ $\Rightarrow$ $\mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$
- $\mathbf{l}$ can be interpreted as a plane normal

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: $\mathbf{l}\ \mathbf{p}=0$



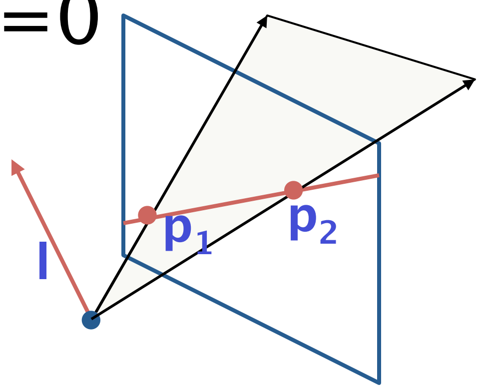


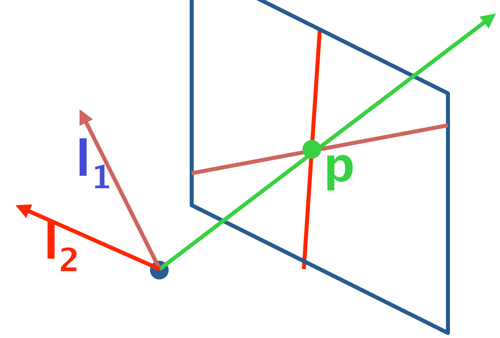
What is the line $\mathbf{l}$ spanned by rays $\mathbf{p_1}$ and $\mathbf{p_2}$?

- $\mathbf{l}$ is $\perp$ to $\mathbf{p_1}$ and $\mathbf{p_2}$ $\Rightarrow$ $\mathbf{l} = \mathbf{p_1} \times \mathbf{p_2}$
- $\mathbf{l}$ can be interpreted as a plane normal

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: $\mathbf{l}\ \mathbf{p}=0$





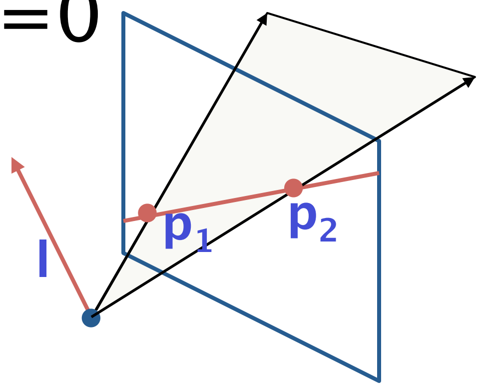
What is the line $\mathbf{l}$ spanned by rays $\mathbf{p}_1$ and $\mathbf{p}_2$?

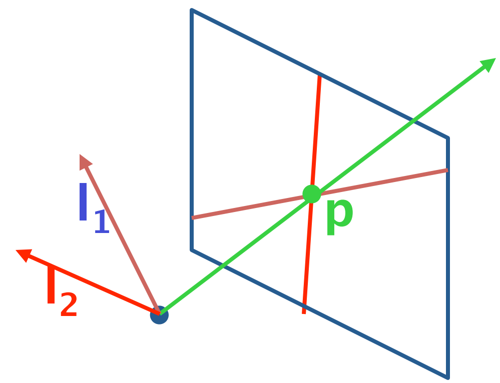
- $\mathbf{l}$ is $\perp$ to $\mathbf{p}_1$ and $\mathbf{p}_2$ $\Rightarrow$ $\mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$
- $\mathbf{l}$ can be interpreted as a plane normal

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

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: $\mathbf{l}\ \mathbf{p}=0$





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

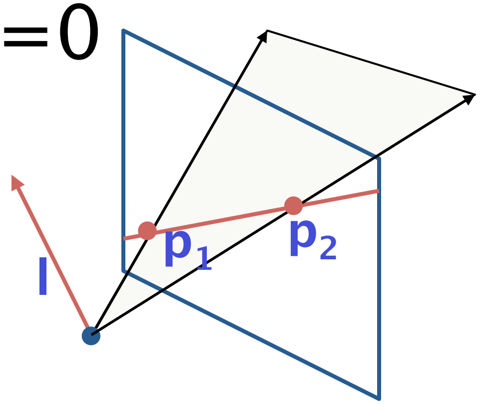
- $\mathbf{l}$ is $\perp$ to $\mathbf{p}_1$ and $\mathbf{p}_2$ $\Rightarrow$ $\mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$
- $\mathbf{l}$ can be interpreted as a plane normal

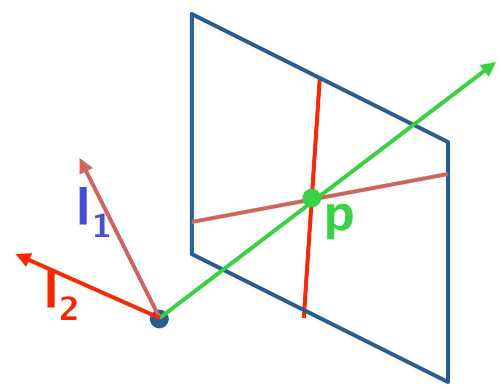
What is the intersection of two lines $\mathbf{l}_1$ and $\mathbf{l}_2$?

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

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: $\mathbf{l}\ \mathbf{p}=0$





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

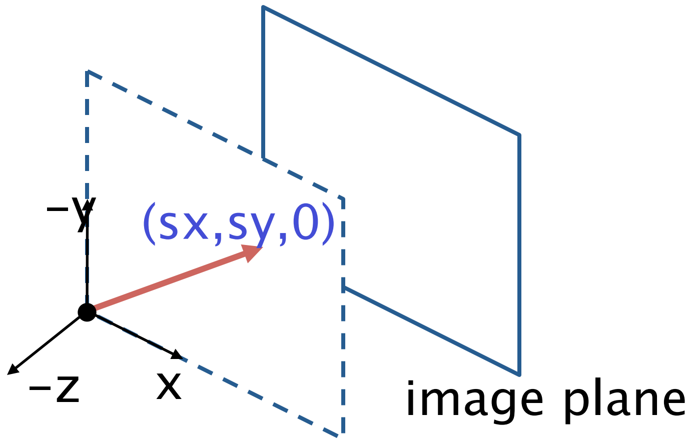
- $\mathbf{l}$ is $\perp$ to $\mathbf{p}_1$ and $\mathbf{p}_2$ $\Rightarrow$ $\mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$
- $\mathbf{l}$ can be interpreted as a plane normal

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

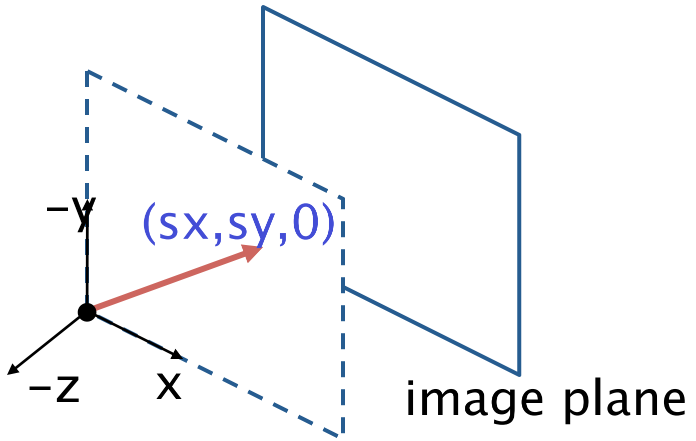
- $\mathbf{p}$ is $\perp$ to $\mathbf{l}_1$ and $\mathbf{l}_2$ $\Rightarrow$ $\mathbf{p} = \mathbf{l}_1 \times \mathbf{l}_2$

Points and lines are dual in projective space

Ideal points and lines

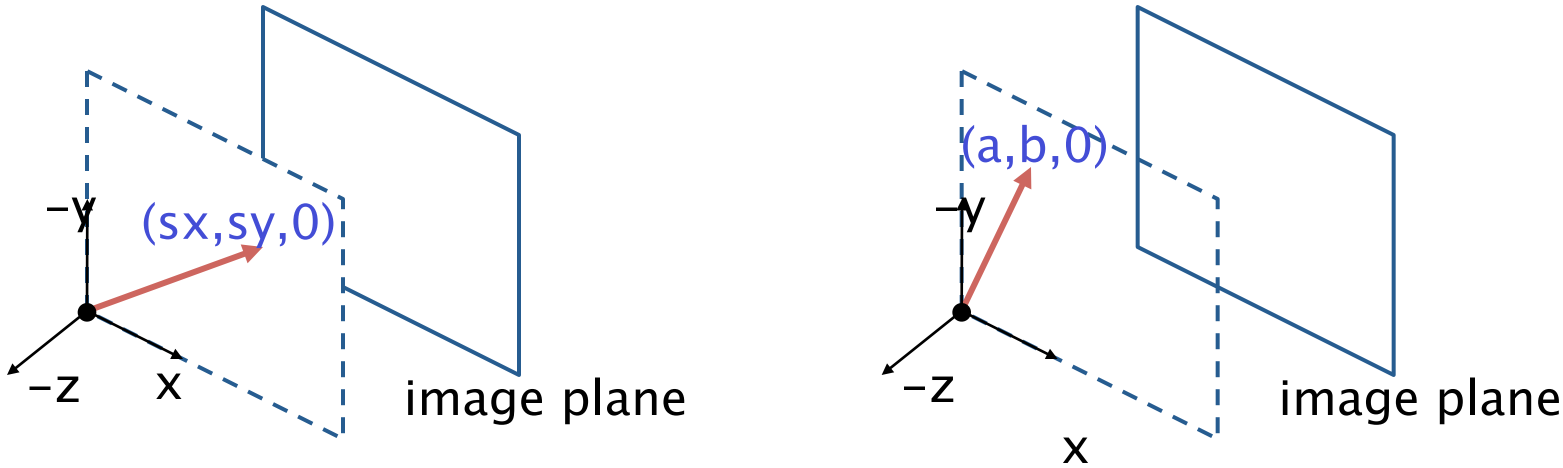


Ideal points and lines



- Ideal point ("point at infinity")
  - p ≅ (x, y, 0) – parallel to image plane
  - It has infinite image coordinates

Ideal points and lines

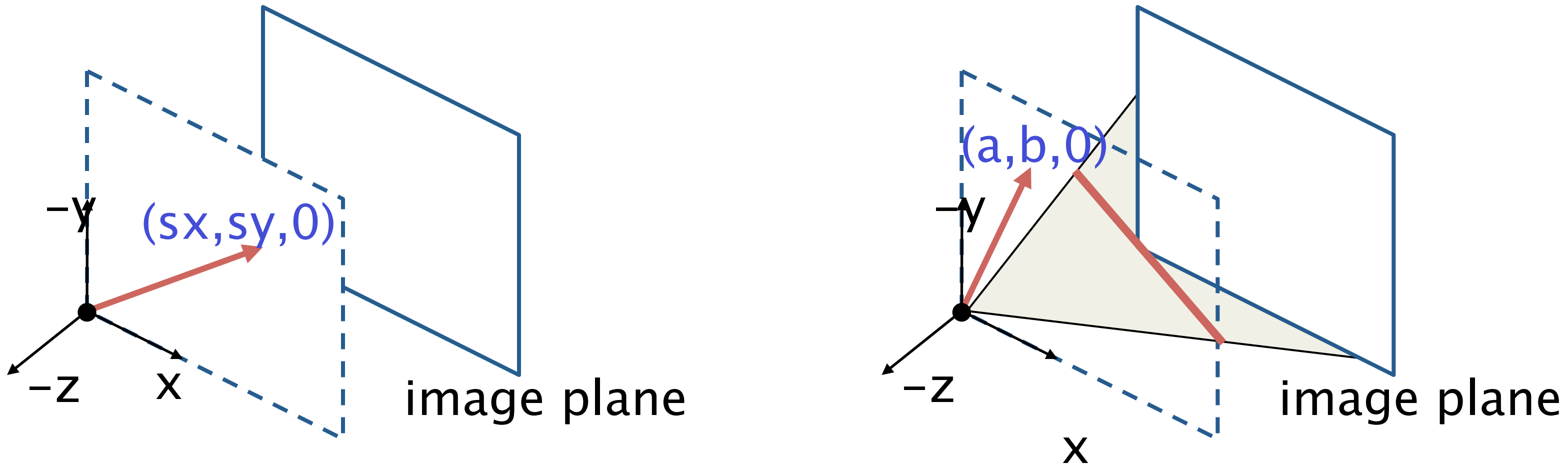


- Ideal point ("point at infinity")
  - $p \cong (x, y, 0)$ – parallel to image plane
  - It has infinite image coordinates

Ideal line

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

Ideal points and lines

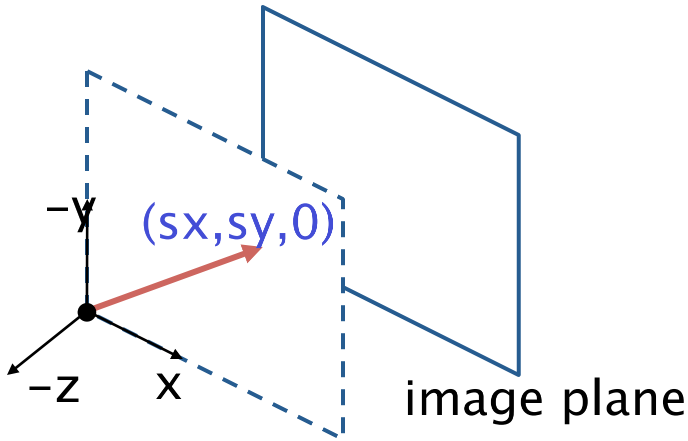


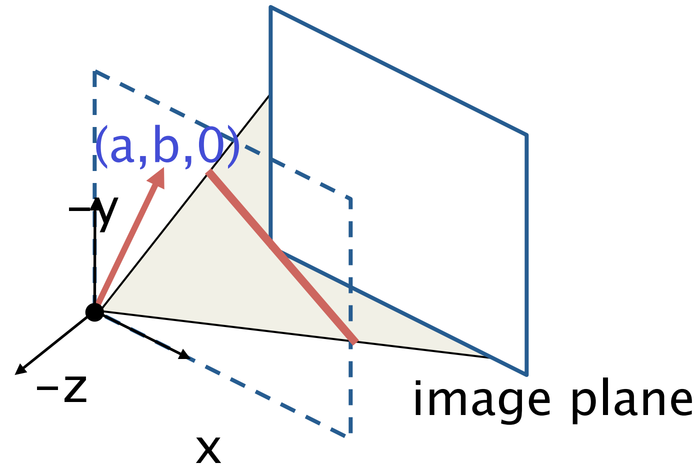
- Ideal point (“point at infinity”)
  - p ≅ (x, y, 0) – parallel to image plane
  - It has infinite image coordinates

Ideal line

- l ≅ (a, b, 0) – parallel to image plane

Ideal points and lines





- Ideal point ("point at infinity")
  - p ≅ (x, y, 0) – parallel to image plane
  - It has infinite image coordinates

Ideal line

- l ≅ (a, b, 0) – parallel to image plane
- Corresponds to a line in the image (finite coordinates)
  - goes through image origin (principle point)

3D projective geometry

3D projective geometry

- These concepts generalize naturally to 3D
  - Homogeneous coordinates
    - Projective 3D points have four coords: **P** = (X,Y,Z,W)

3D projective geometry

- These concepts generalize naturally to 3D
  - Homogeneous coordinates
    - Projective 3D points have four coords: **P** = (X,Y,Z,W)
  - Duality

3D projective geometry

- These concepts generalize naturally to 3D
  - Homogeneous coordinates
    - Projective 3D points have four coords: **P** = (X,Y,Z,W)
  - Duality
    - A plane **N** is also represented by a 4-vector

3D projective geometry

- These concepts generalize naturally to 3D
  - Homogeneous coordinates
    - Projective 3D points have four coords: **P** = (X,Y,Z,W)
  - Duality
    - A plane **N** is also represented by a 4-vector
    - Points and planes are dual in 3D: **N** **P**=0

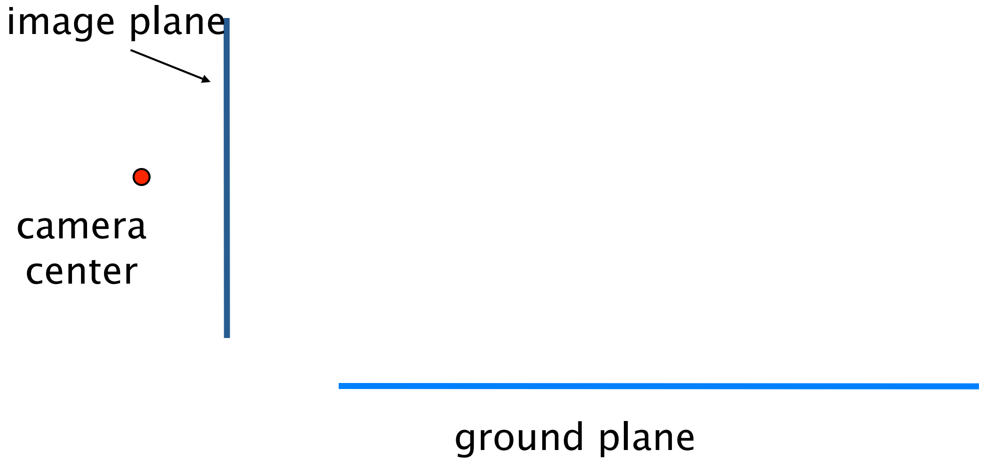
3D projective geometry

- These concepts generalize naturally to 3D
  - Homogeneous coordinates
    - Projective 3D points have four coords: **P** = (X,Y,Z,W)
  - Duality
    - A plane **N** is also represented by a 4-vector
    - Points and planes are dual in 3D: **N** **P**=0
    - Three points define a plane, three planes define a point

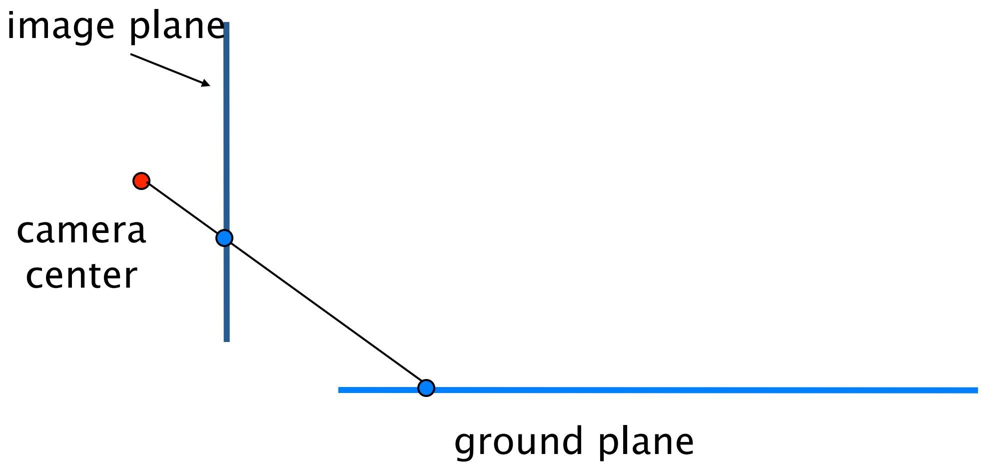
3D to 2D: perspective projection

Projection: $$\mathbf{p} = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = \begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \text{Đ}\mathbf{P}$$

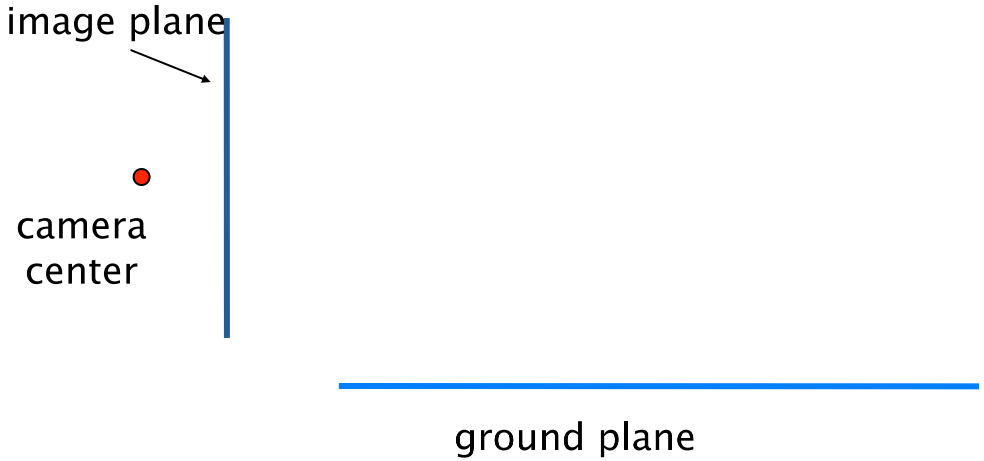
Vanishing points (1D)



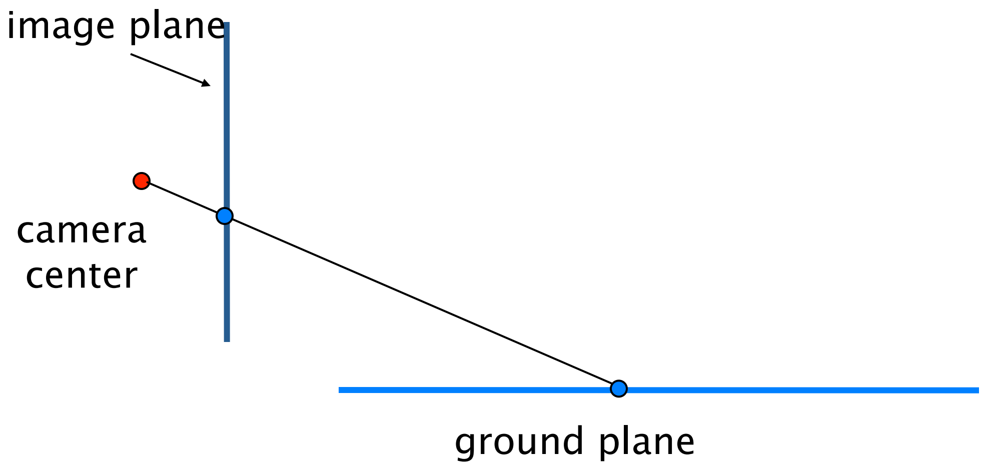
Vanishing points (1D)



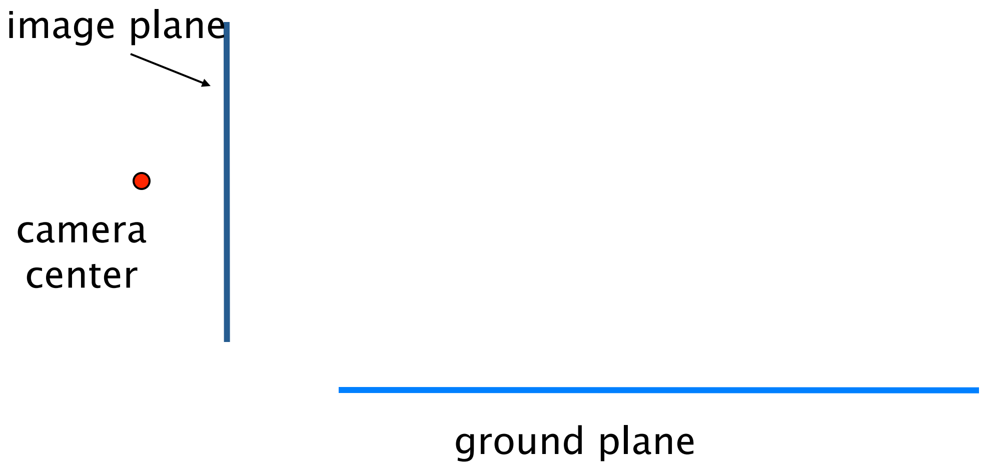
Vanishing points (1D)



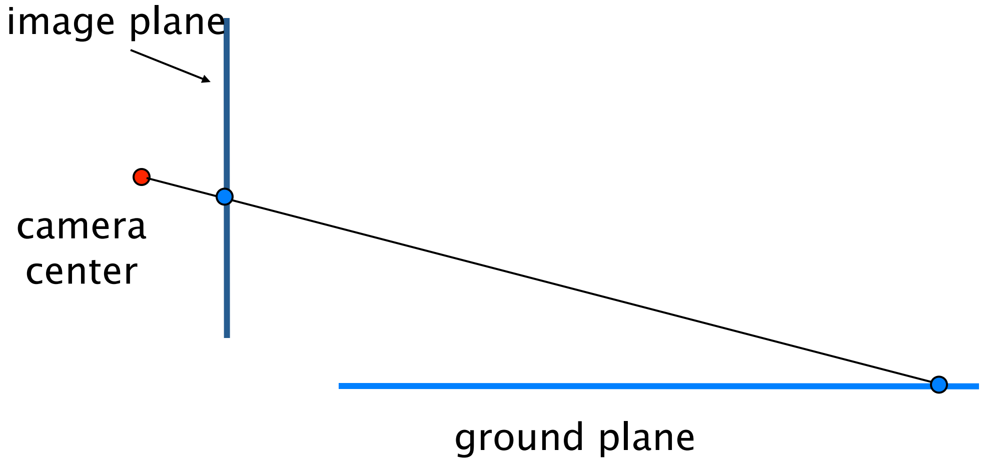
Vanishing points (1D)



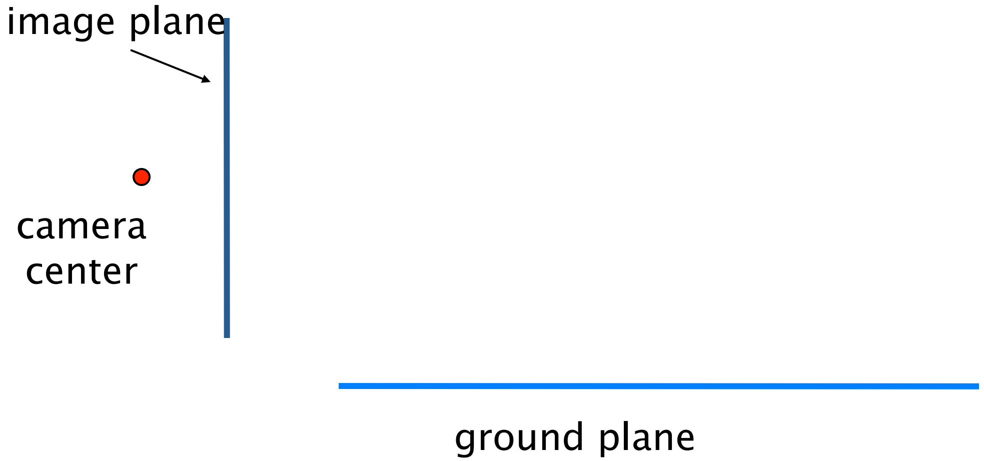
Vanishing points (1D)



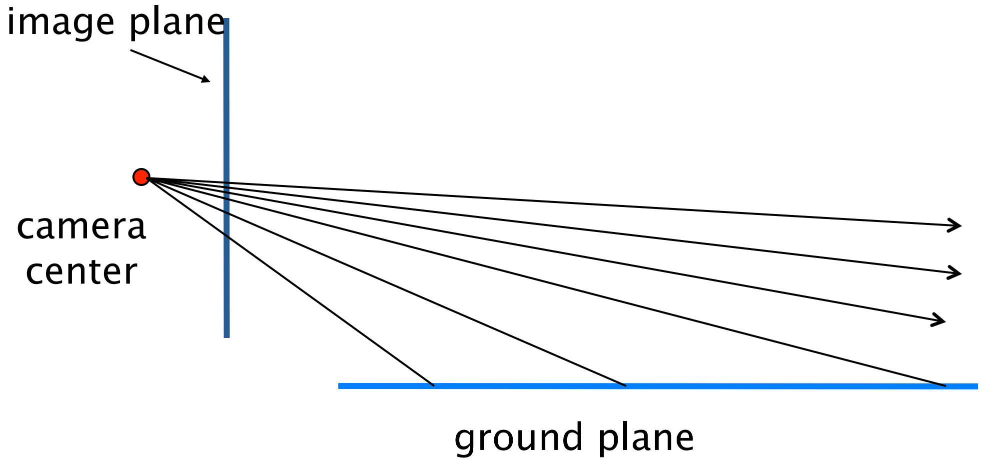
Vanishing points (1D)



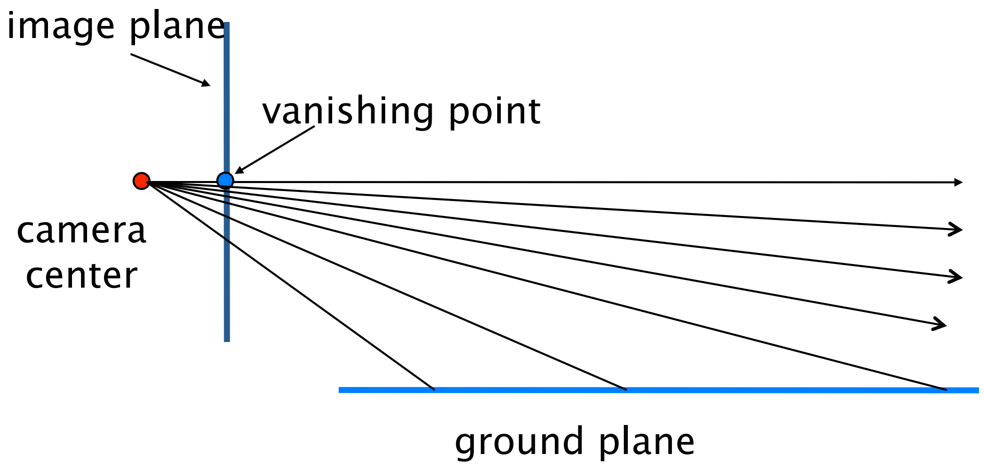
Vanishing points (1D)



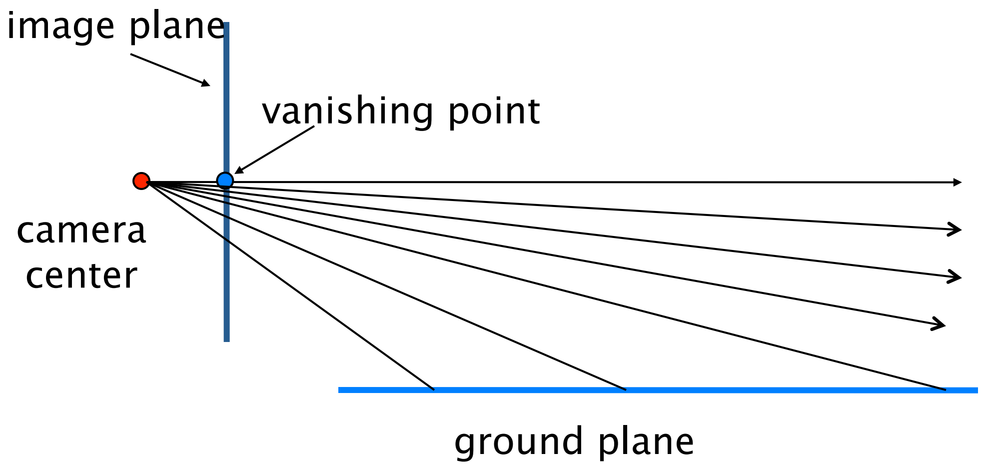
Vanishing points (1D)



Vanishing points (1D)

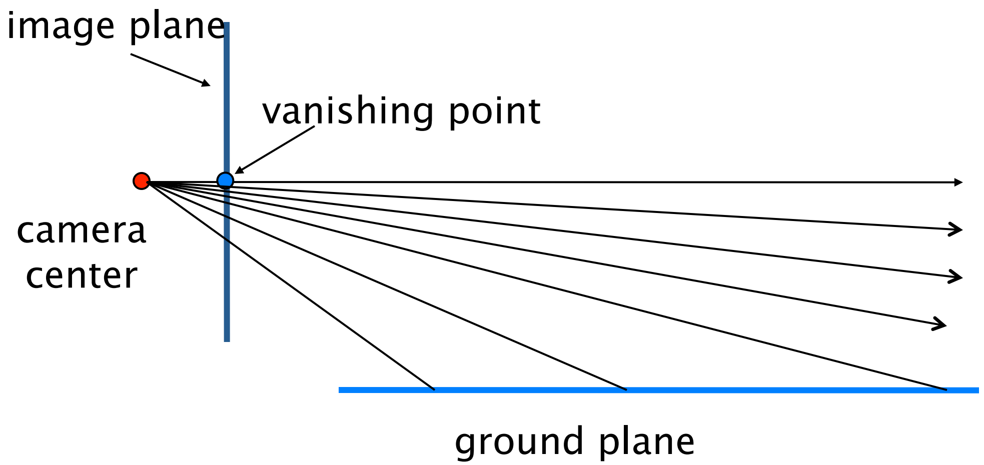


Vanishing points (1D)

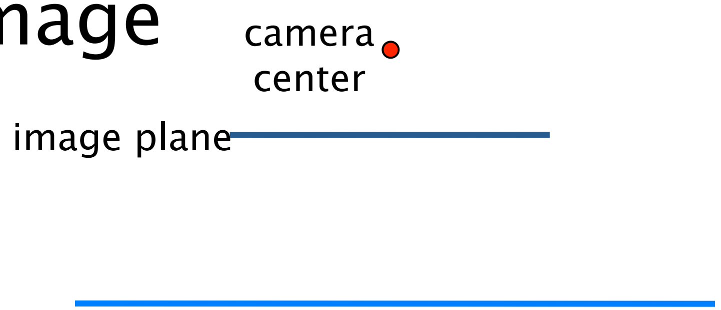


- Vanishing point
  - projection of a point at infinity
  - can often (but not always) project to a finite point in the image

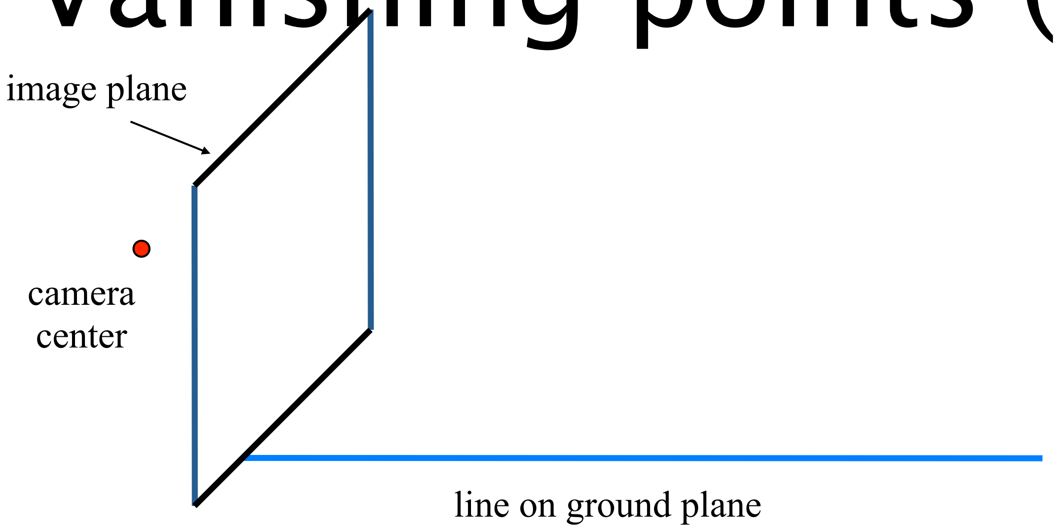
Vanishing points (1D)



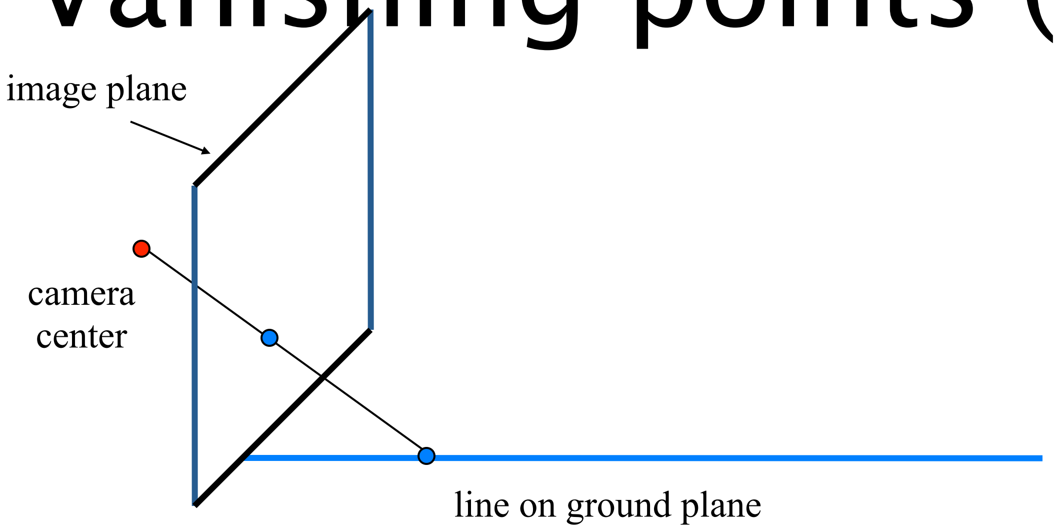
- Vanishing point
  - projection of a point at infinity
  - can often (but not always) project to a finite point in the image



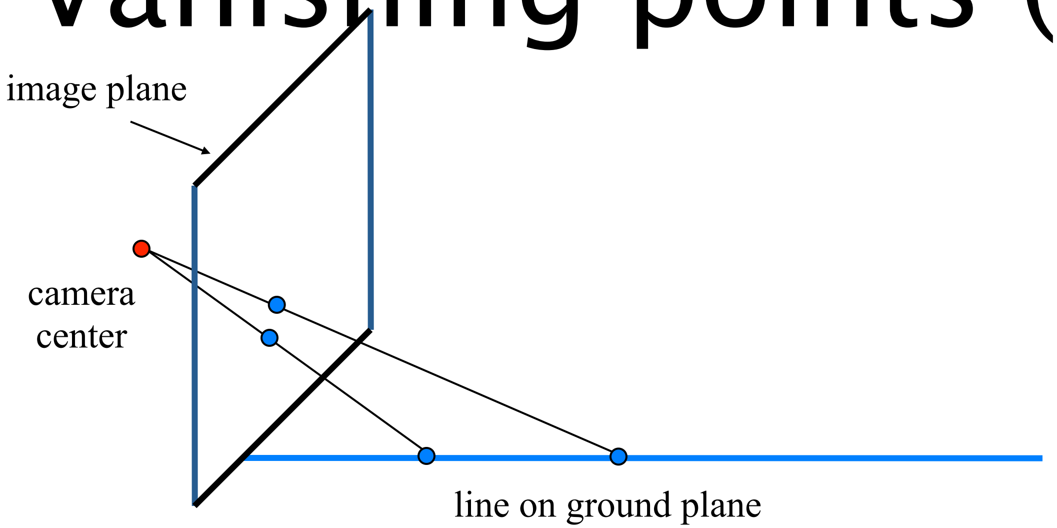
Vanishing points (2D)



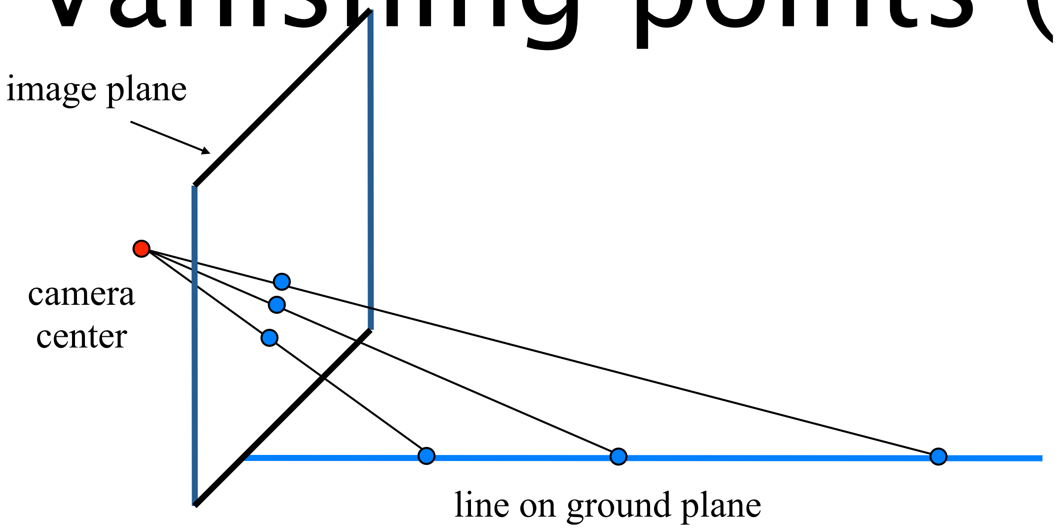
Vanishing points (2D)



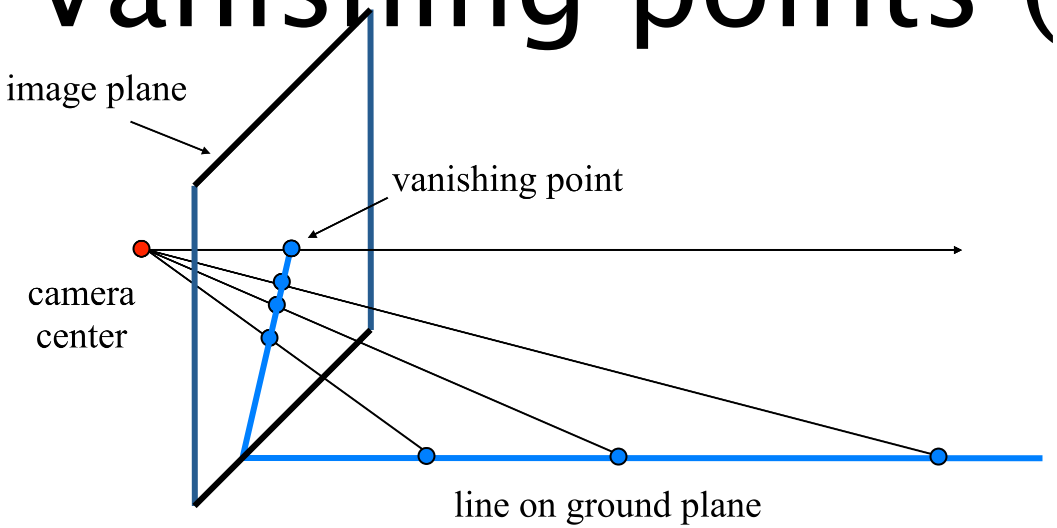
Vanishing points (2D)



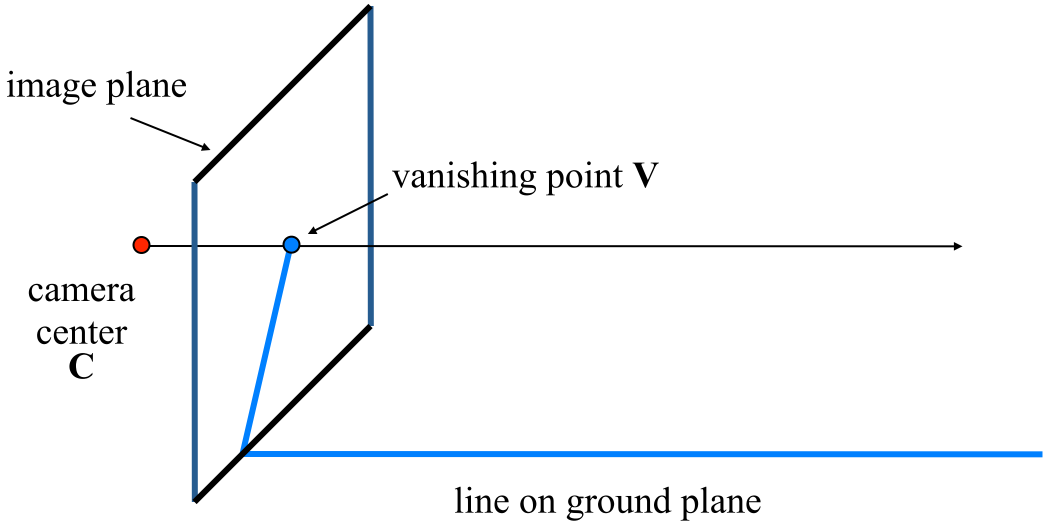
Vanishing points (2D)



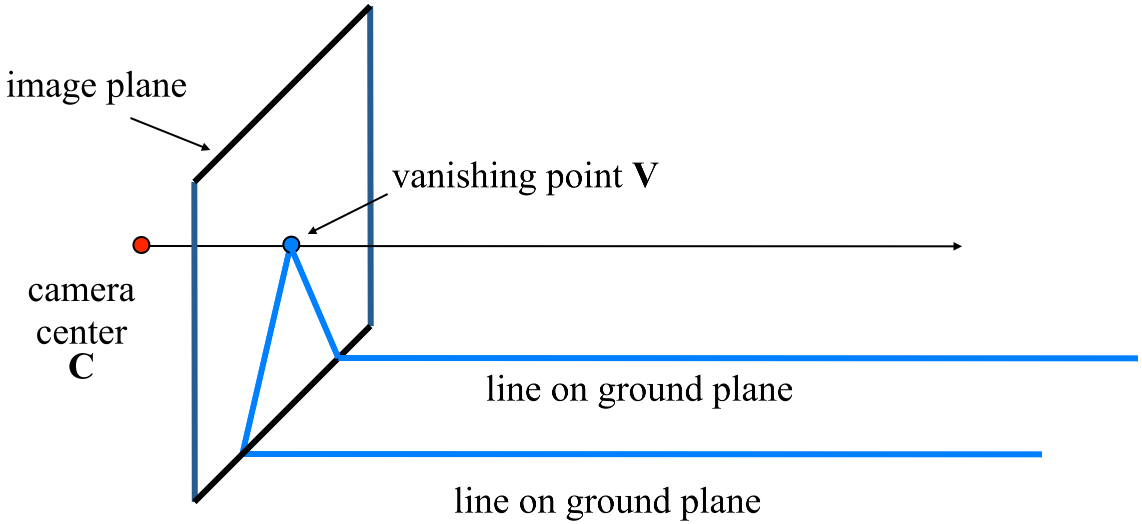
Vanishing points (2D)



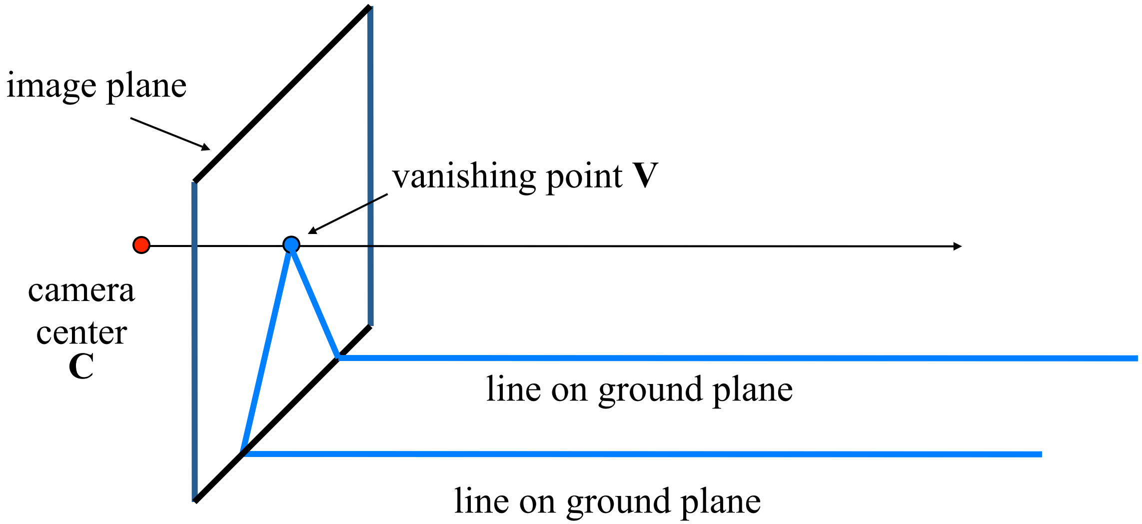
Vanishing points



Vanishing points

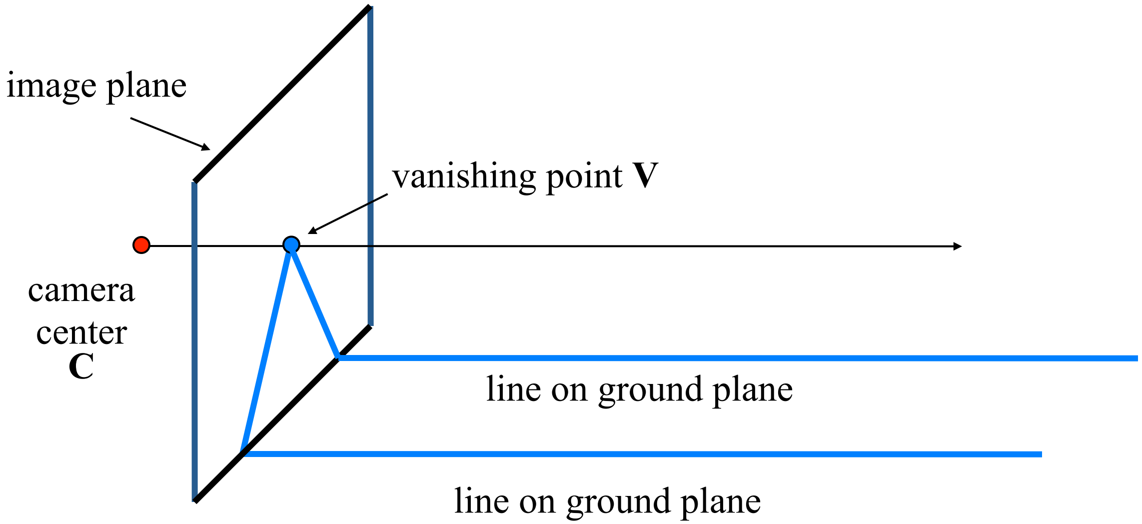


Vanishing points



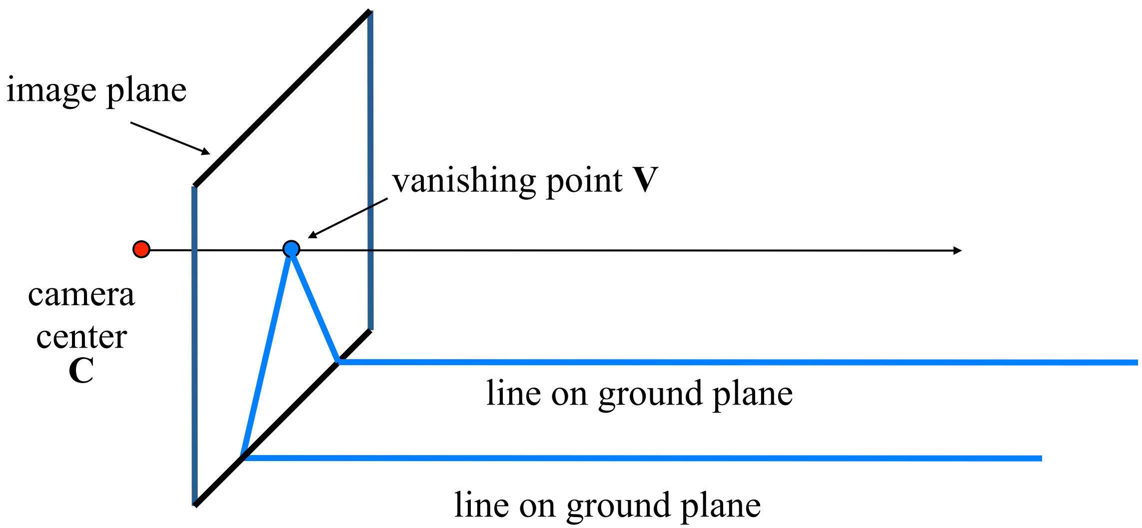
- Properties
  - Any two parallel lines (in 3D) have the same vanishing point **v**

Vanishing points



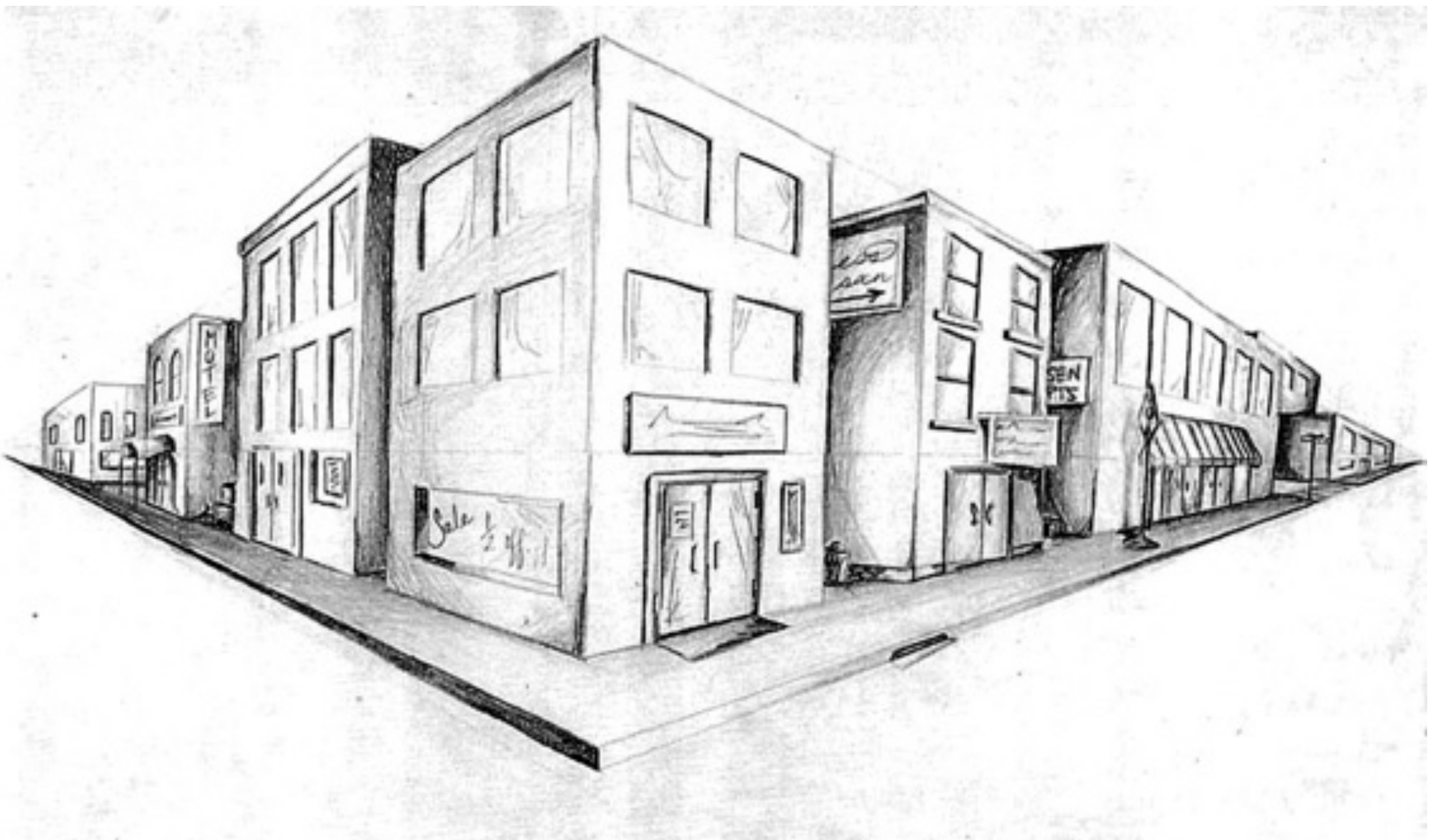
- Properties
  - Any two parallel lines (in 3D) have the same vanishing point **v**
  - The ray from **C** through **v** is parallel to the lines

Vanishing points

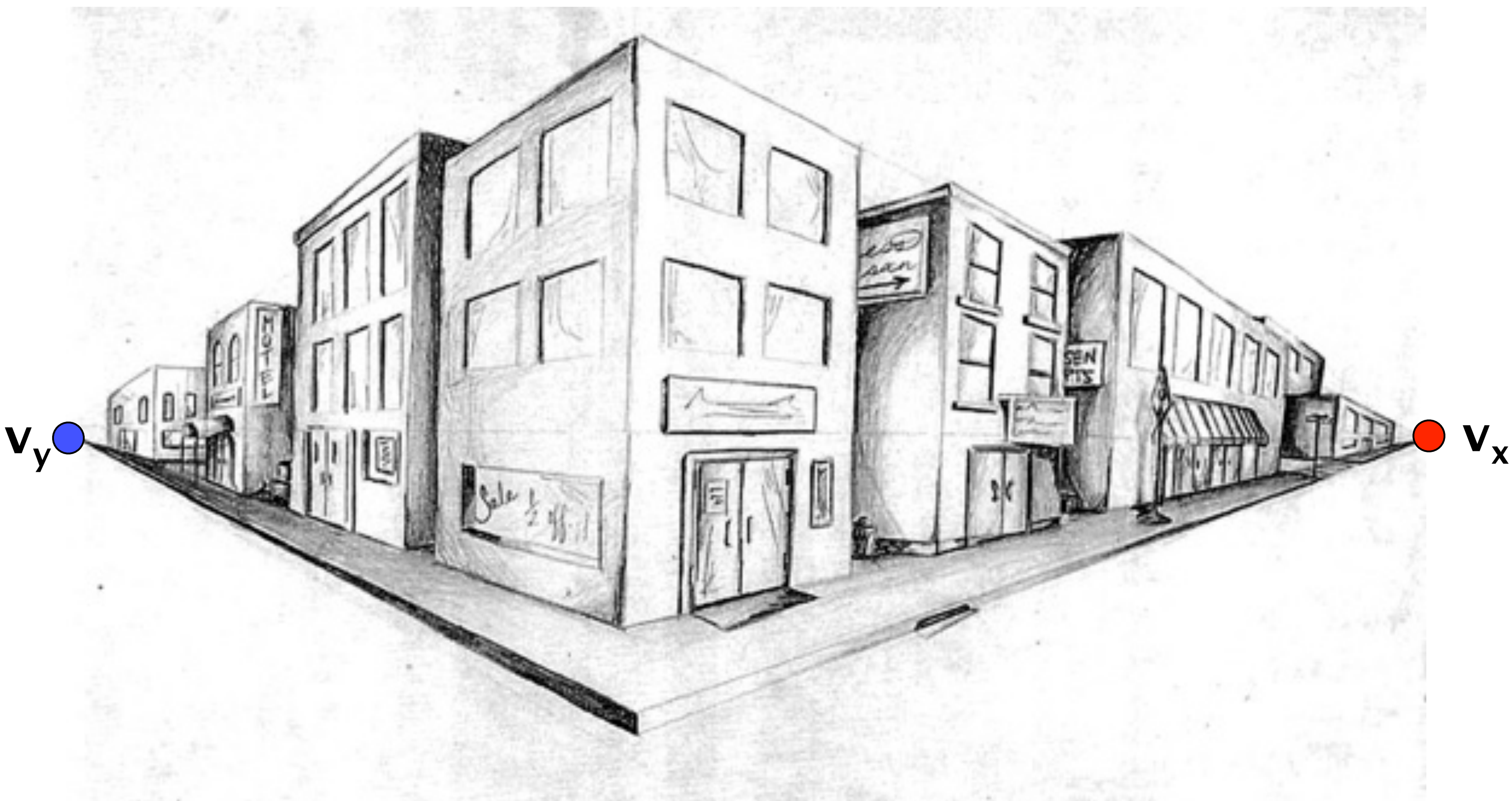


- Properties
  - Any two parallel lines (in 3D) have the same vanishing point **v**
  - The ray from **C** through **v** is parallel to the lines
  - An image may have more than one vanishing point
    - in fact, every image point is a potential vanishing point

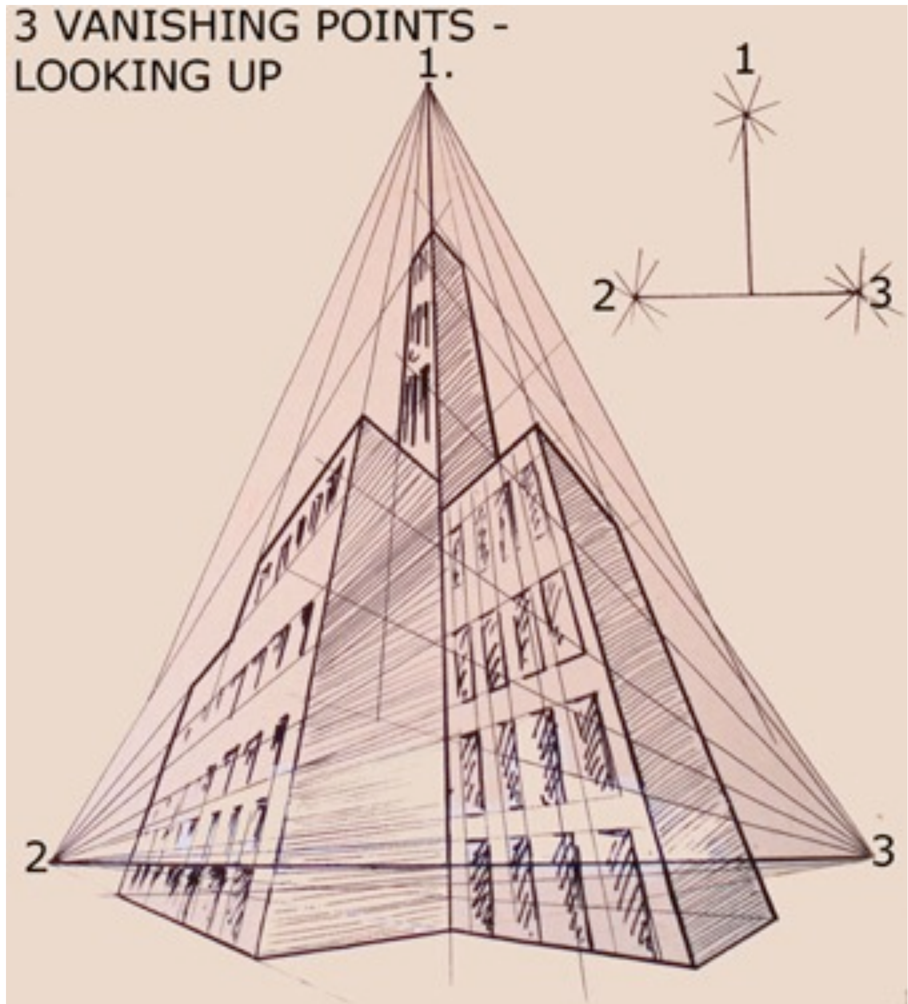
Two point perspective



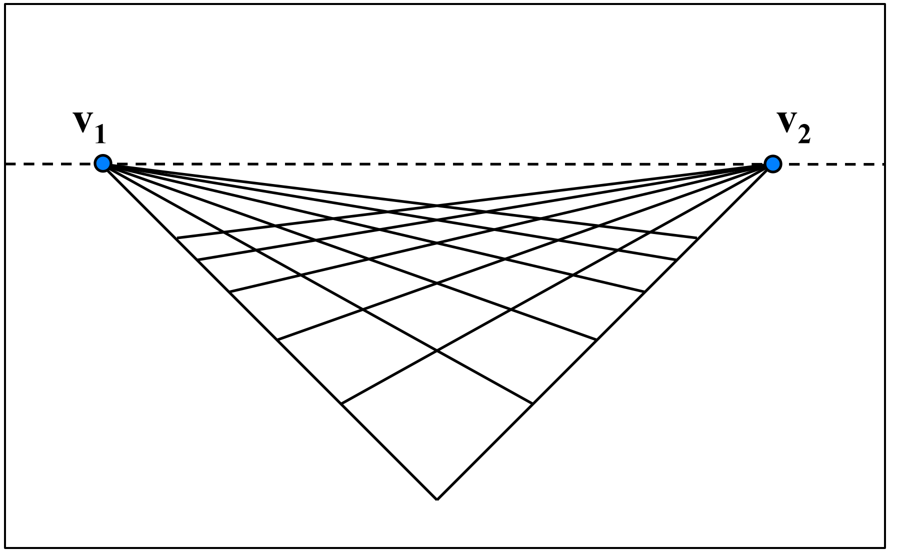
Two point perspective



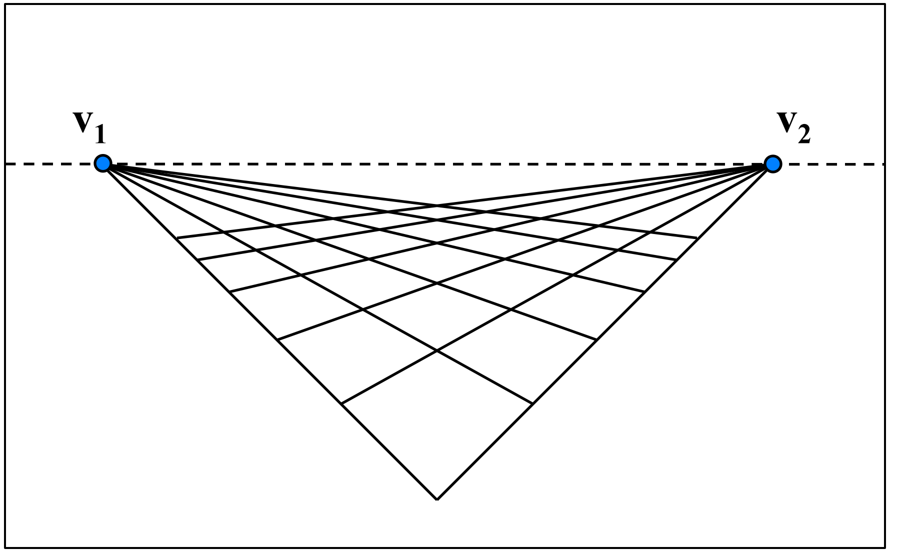
Three point perspective



Vanishing lines

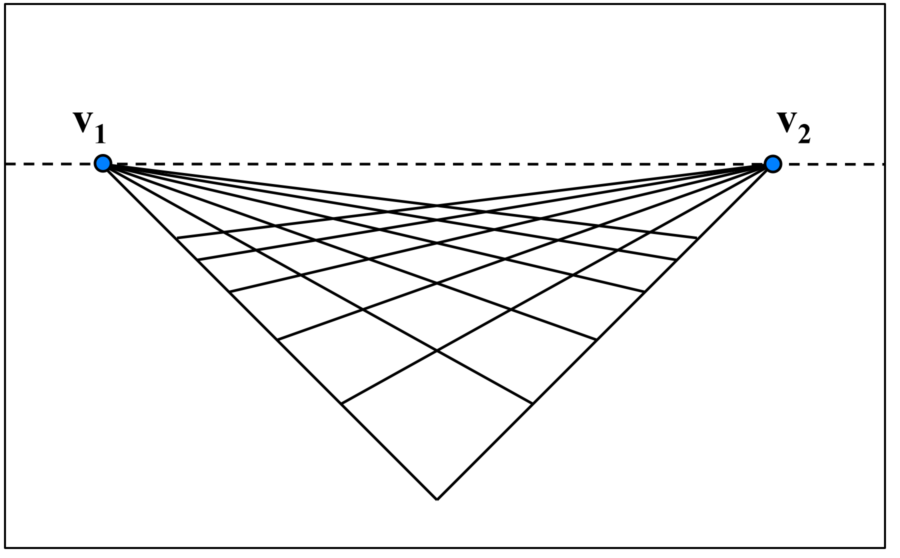


Vanishing lines



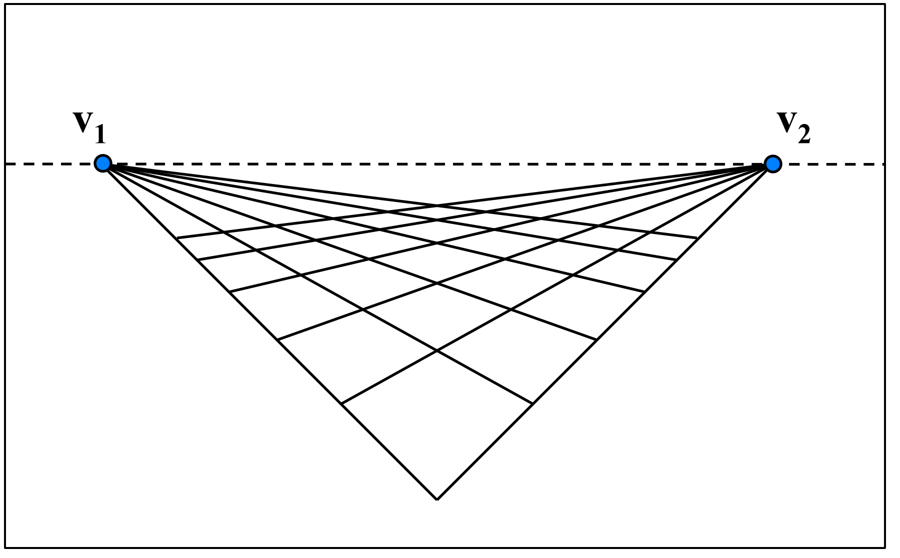
- Multiple Vanishing Points
  - Any set of parallel lines on the plane define a vanishing point

Vanishing lines



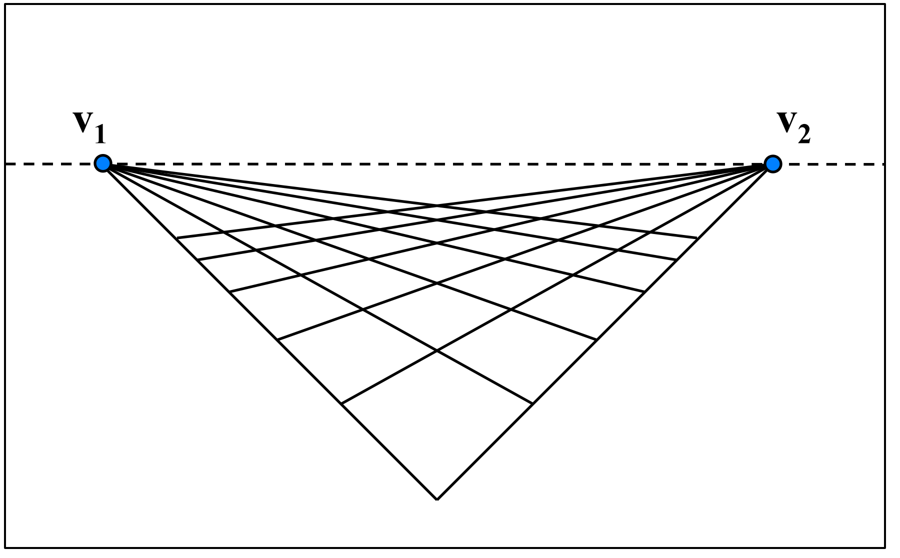
- Multiple Vanishing Points
  - Any set of parallel lines on the plane define a vanishing point
  - The union of all of these vanishing points is the horizon line

Vanishing lines



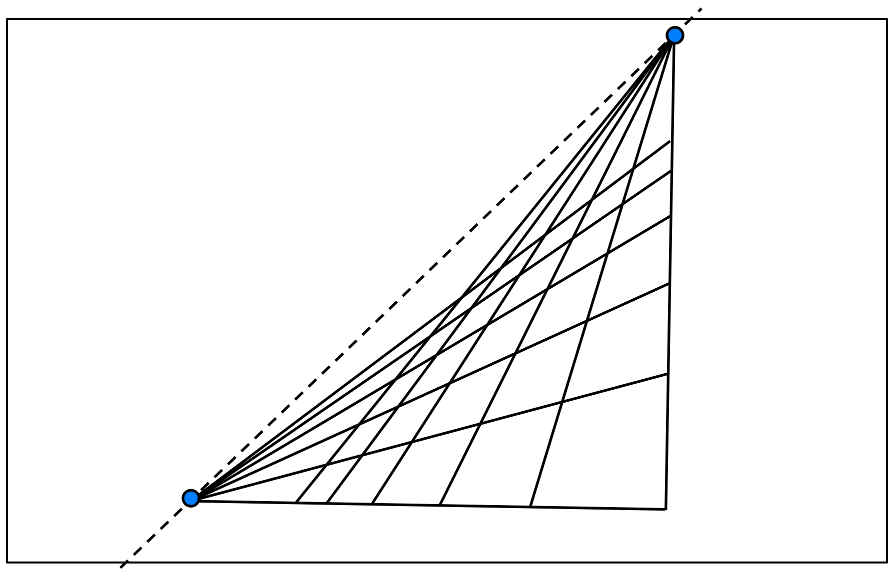
- Multiple Vanishing Points
  - Any set of parallel lines on the plane define a vanishing point
  - The union of all of these vanishing points is the horizon line
    - also called vanishing line

Vanishing lines



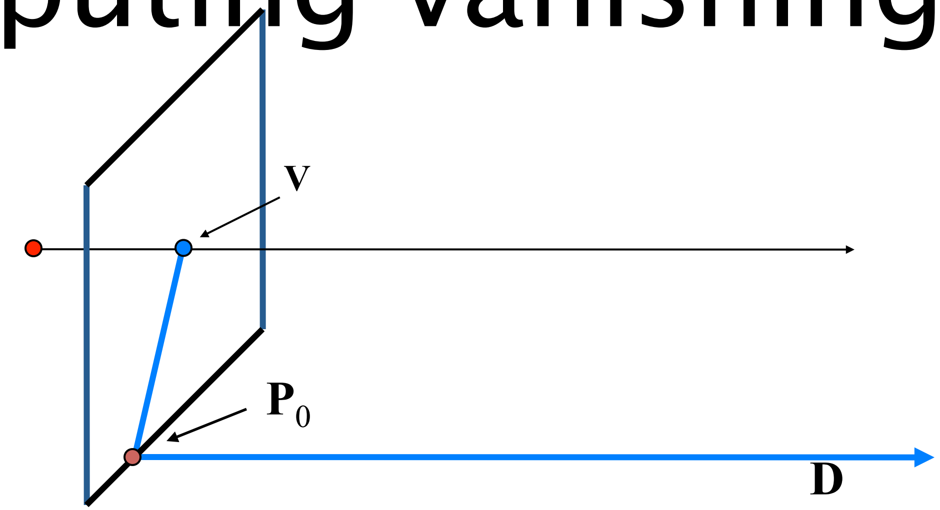
- Multiple Vanishing Points
  - Any set of parallel lines on the plane define a vanishing point
  - The union of all of these vanishing points is the horizon line
    - also called vanishing line
  - Note that different planes (can) define different vanishing lines

Vanishing lines



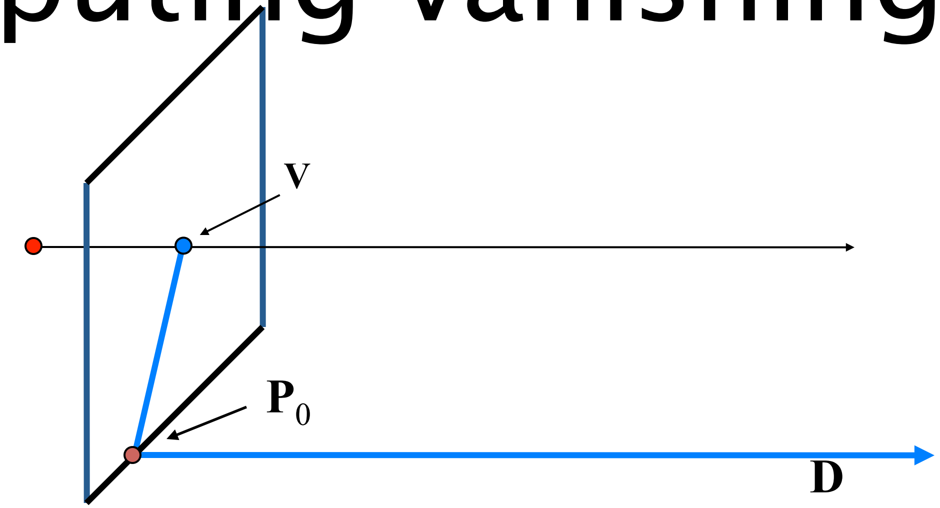
- Multiple Vanishing Points
  - Any set of parallel lines on the plane define a vanishing point
  - The union of all of these vanishing points is the horizon line
    - also called vanishing line
  - Note that different planes (can) define different vanishing lines

Computing vanishing points



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

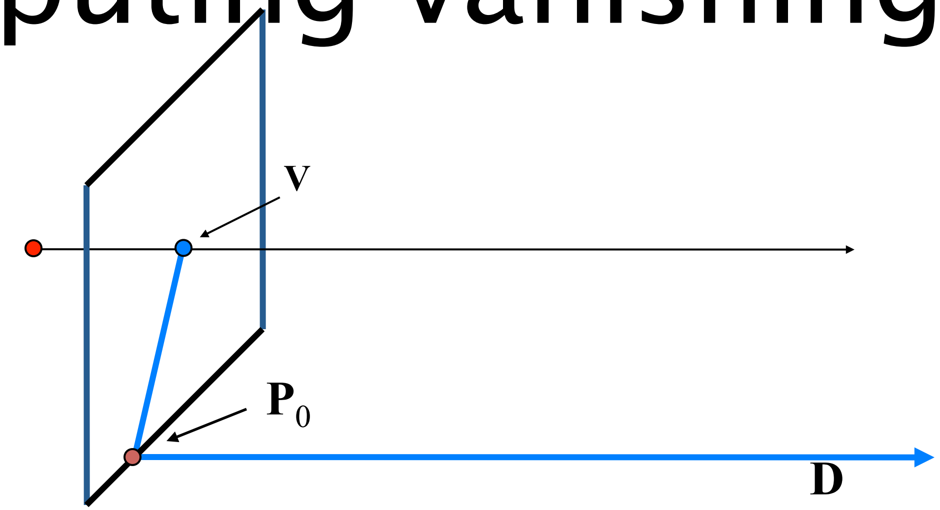
Computing vanishing points



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

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

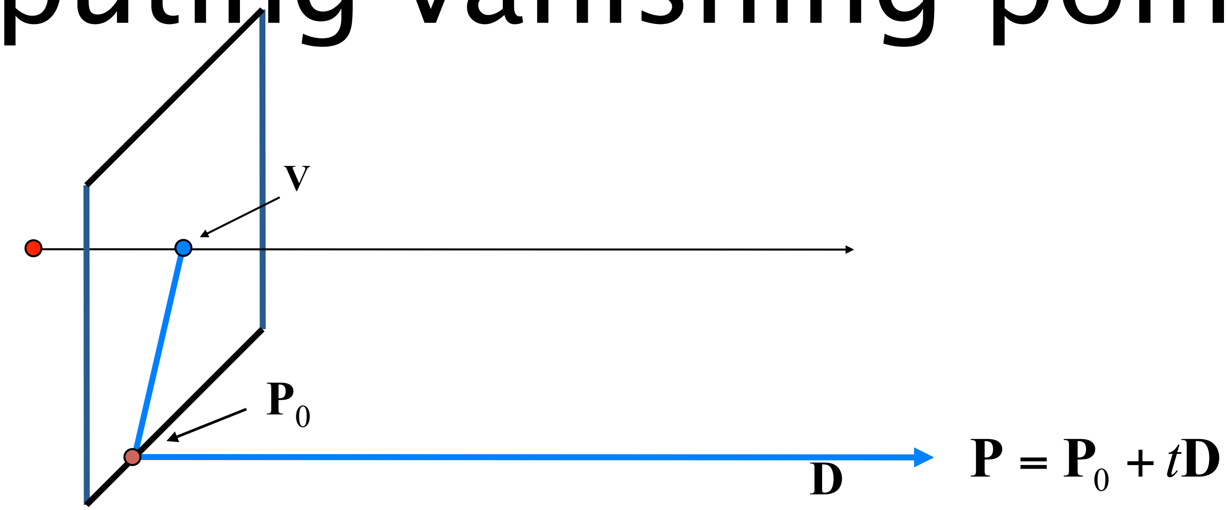
Computing vanishing points



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

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

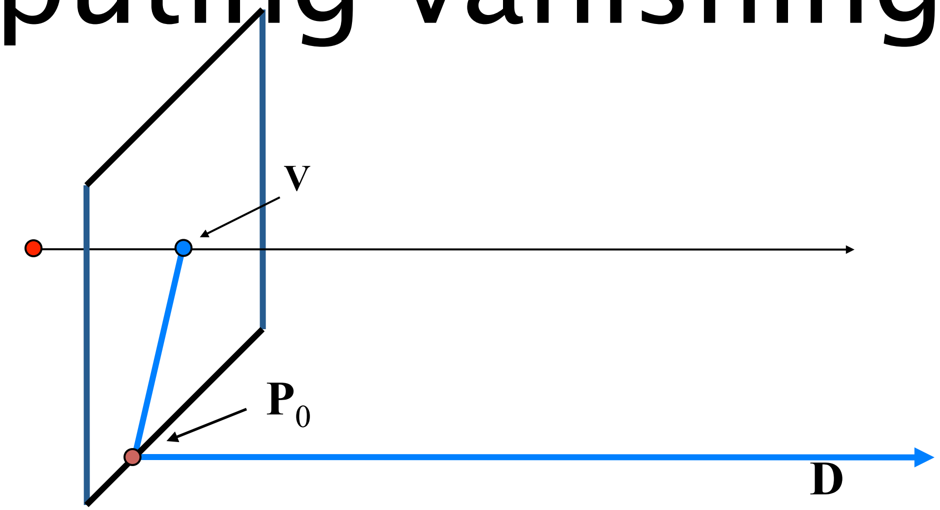
Computing vanishing points



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

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

Computing vanishing points

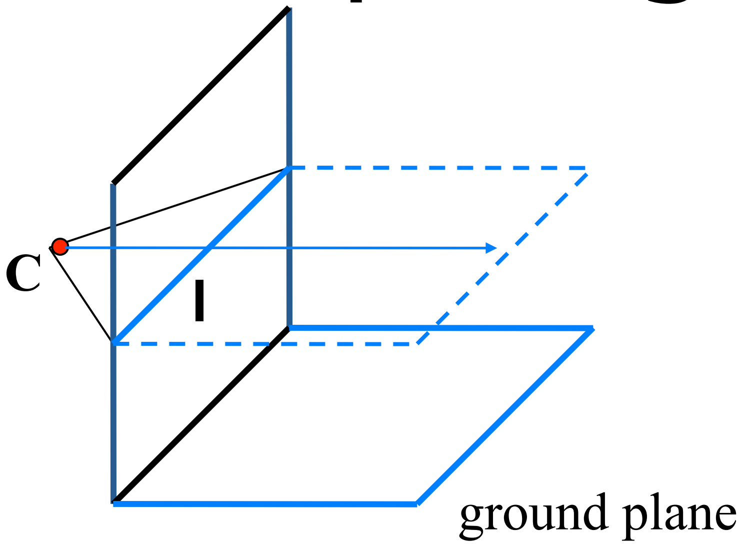


$$\mathbf{P} = \mathbf{P}_0 + t\mathbf{D}$$

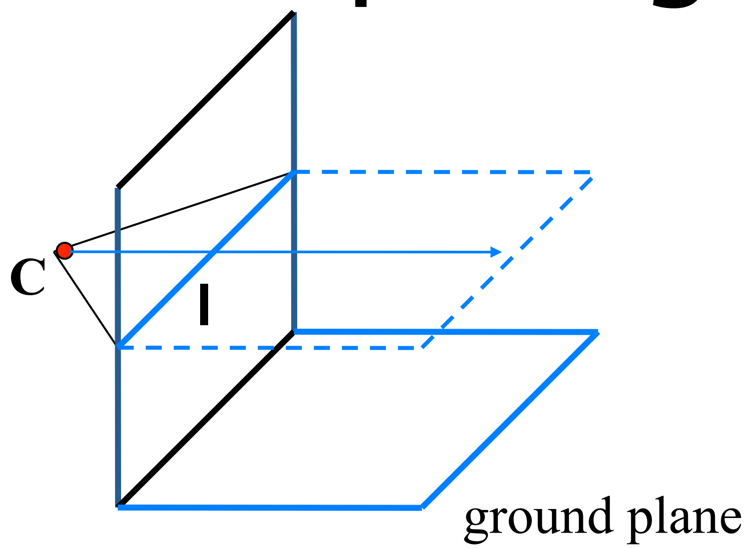
$$\mathbf{P}_t = \begin{bmatrix} P_X + tD_X \\ P_Y + tD_Y \\ P_Z + tD_Z \\ 1 \end{bmatrix} \cong \begin{bmatrix} P_X / t + D_X \\ P_Y / t + D_Y \\ P_Z / t + D_Z \\ 1/t \end{bmatrix} \qquad t \rightarrow \infty \qquad \mathbf{P}_\infty \cong \begin{bmatrix} D_X \\ D_Y \\ D_Z \\ 0 \end{bmatrix}$$

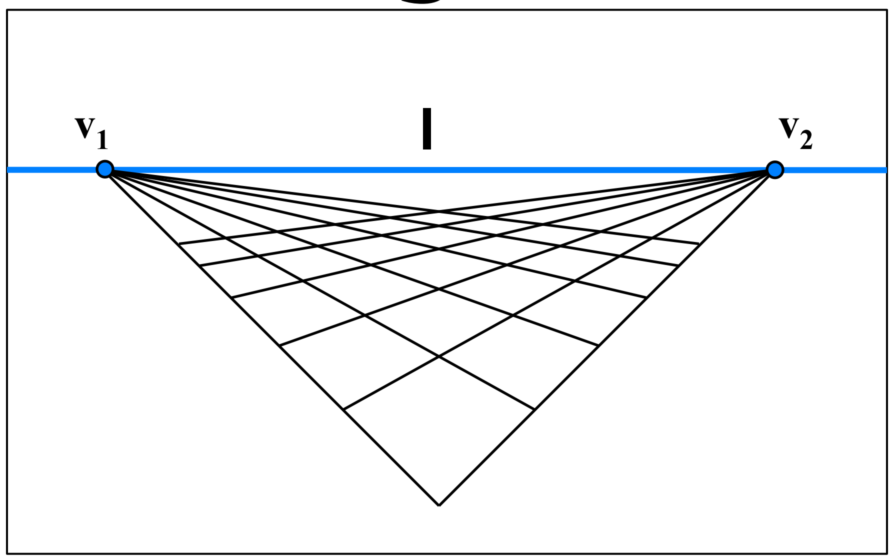
- Properties $\mathbf{v} = \Pi \mathbf{P}_\infty$
  - $\mathbf{P}_\infty$ is a point at infinity, **v** is its projection
  - Depends only on line direction
  - Parallel lines $\mathbf{P}_0$ + t**D**, $\mathbf{P}_1$ + t**D** intersect at $\mathbf{P}_\infty$

Computing vanishing lines

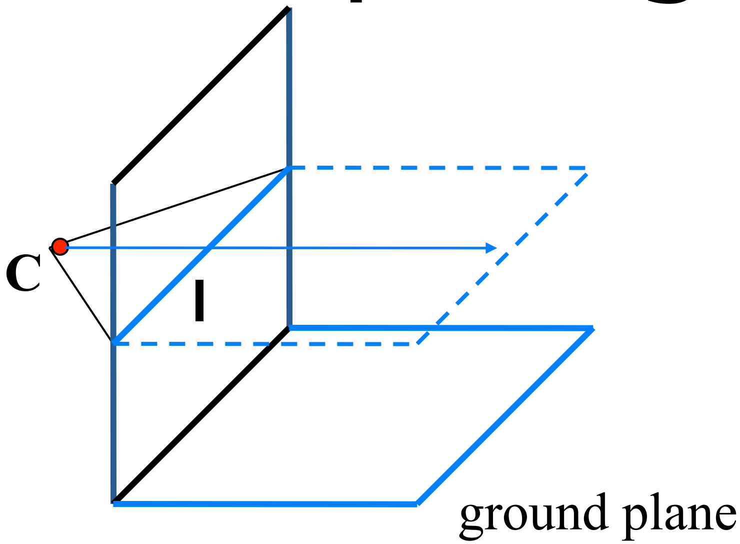


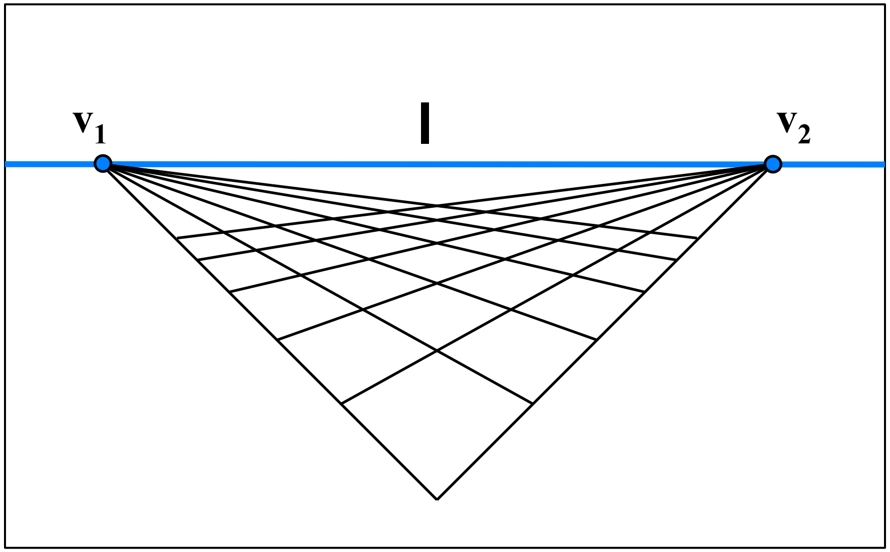
Computing vanishing lines





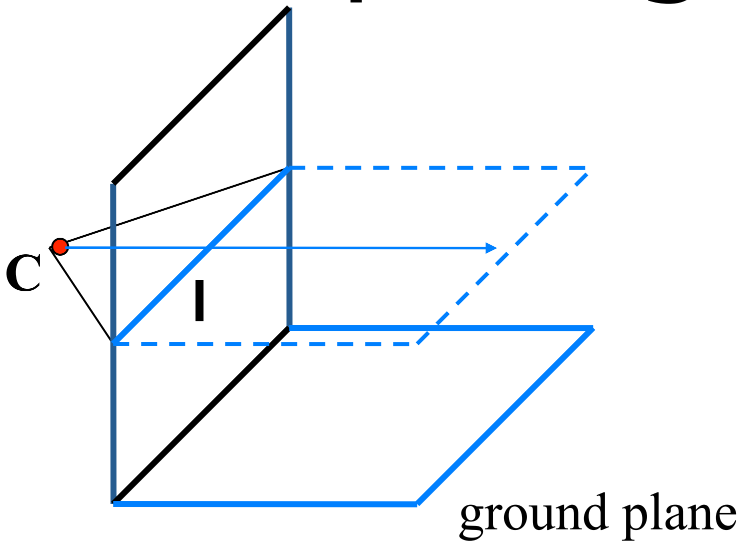
Computing vanishing lines

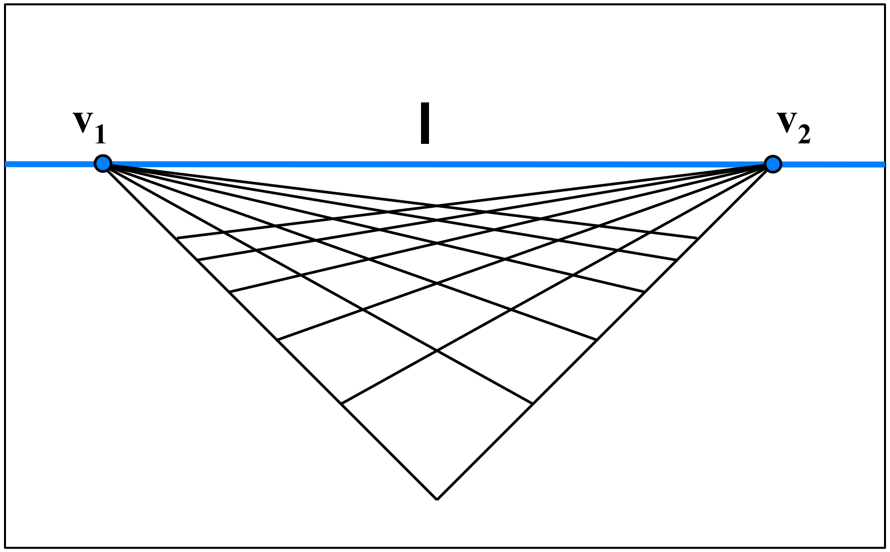




- Properties
  - **l** is intersection of horizontal plane through **C** with image plane

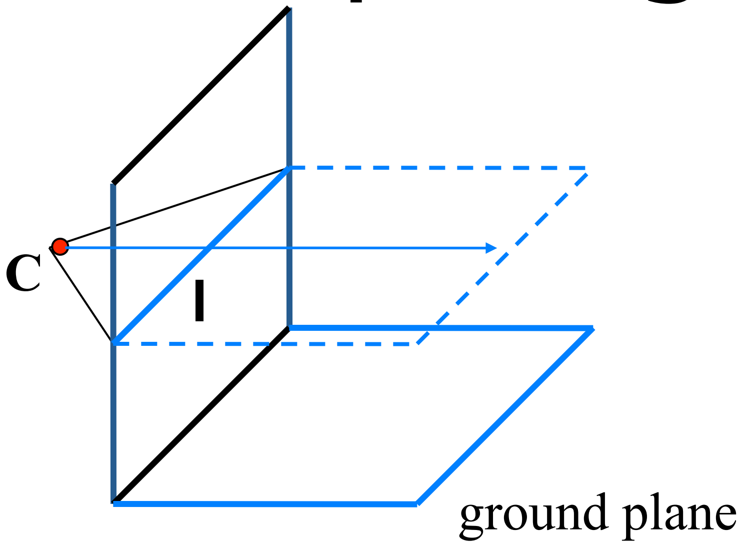
Computing vanishing lines

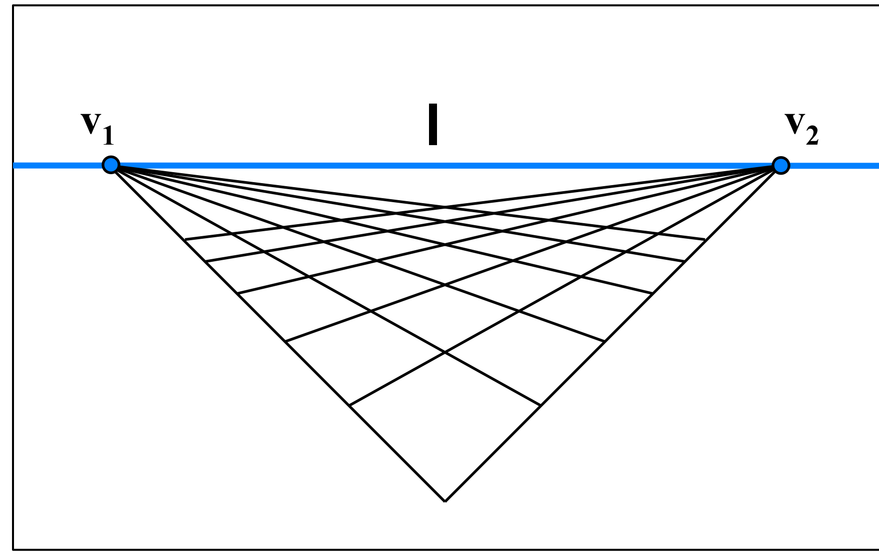




- Properties
  - **l** is intersection of horizontal plane through **C** with image plane
  - Compute **l** from two sets of parallel lines on ground plane

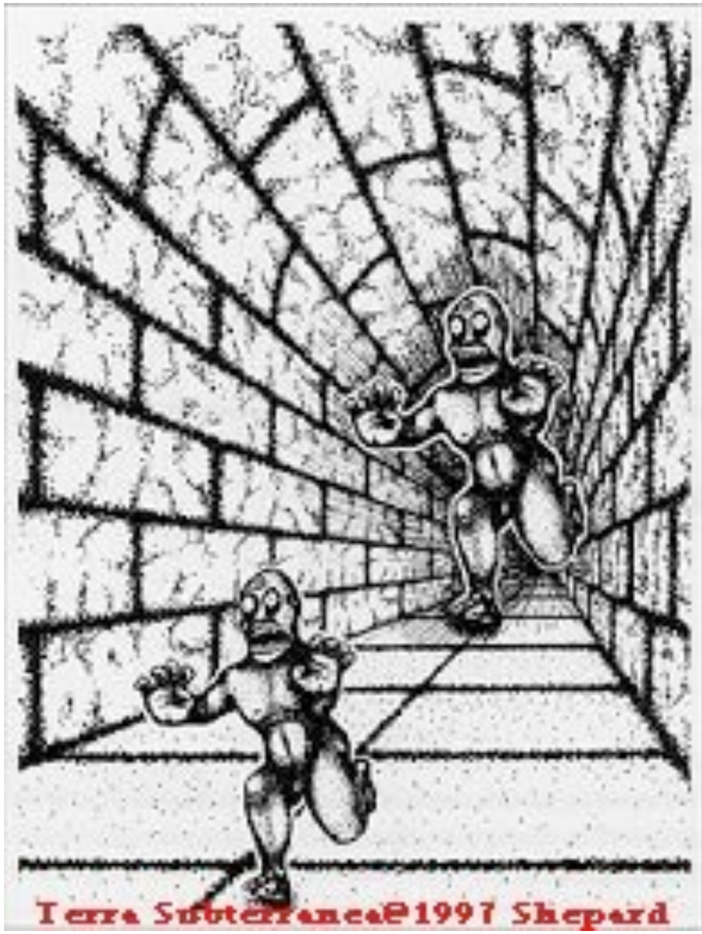
Computing vanishing lines



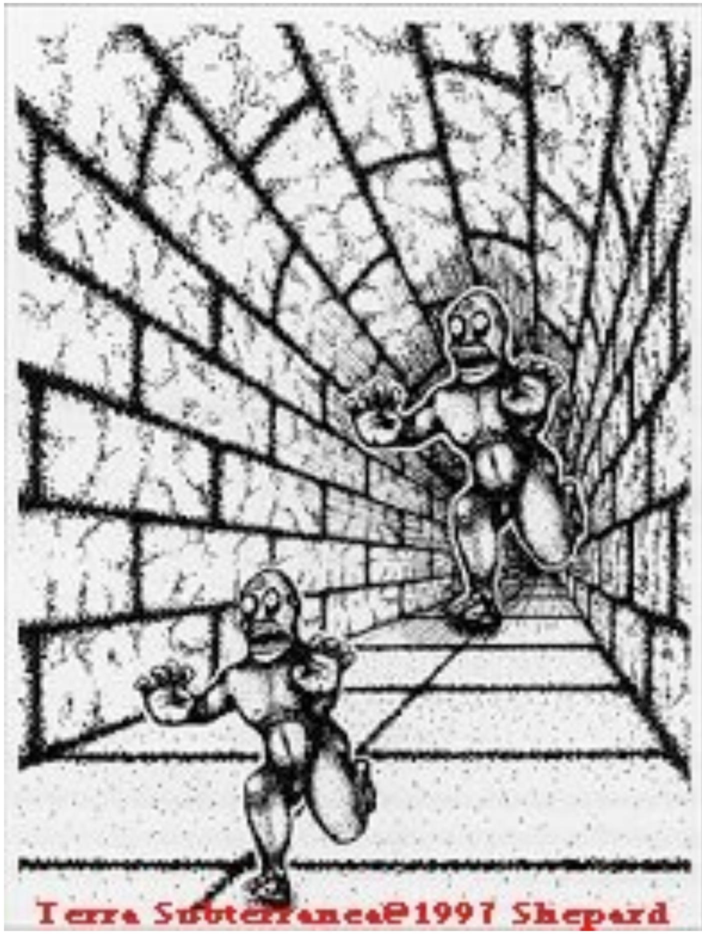


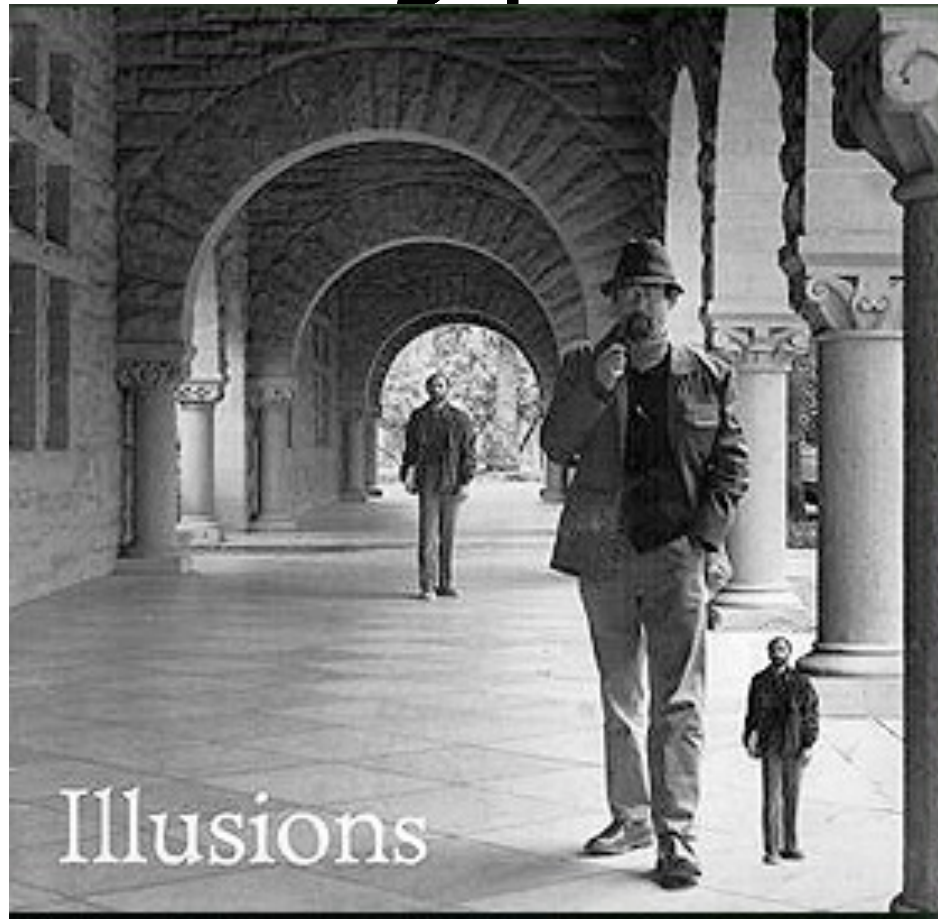
- Properties
  - **l** is intersection of horizontal plane through **C** with image plane
  - Compute **l** from two sets of parallel lines on ground plane
  - All points at same height as **C** project to **l**
    - points higher than C project above l
  - Provides way of comparing height of objects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

Fun with vanishing points

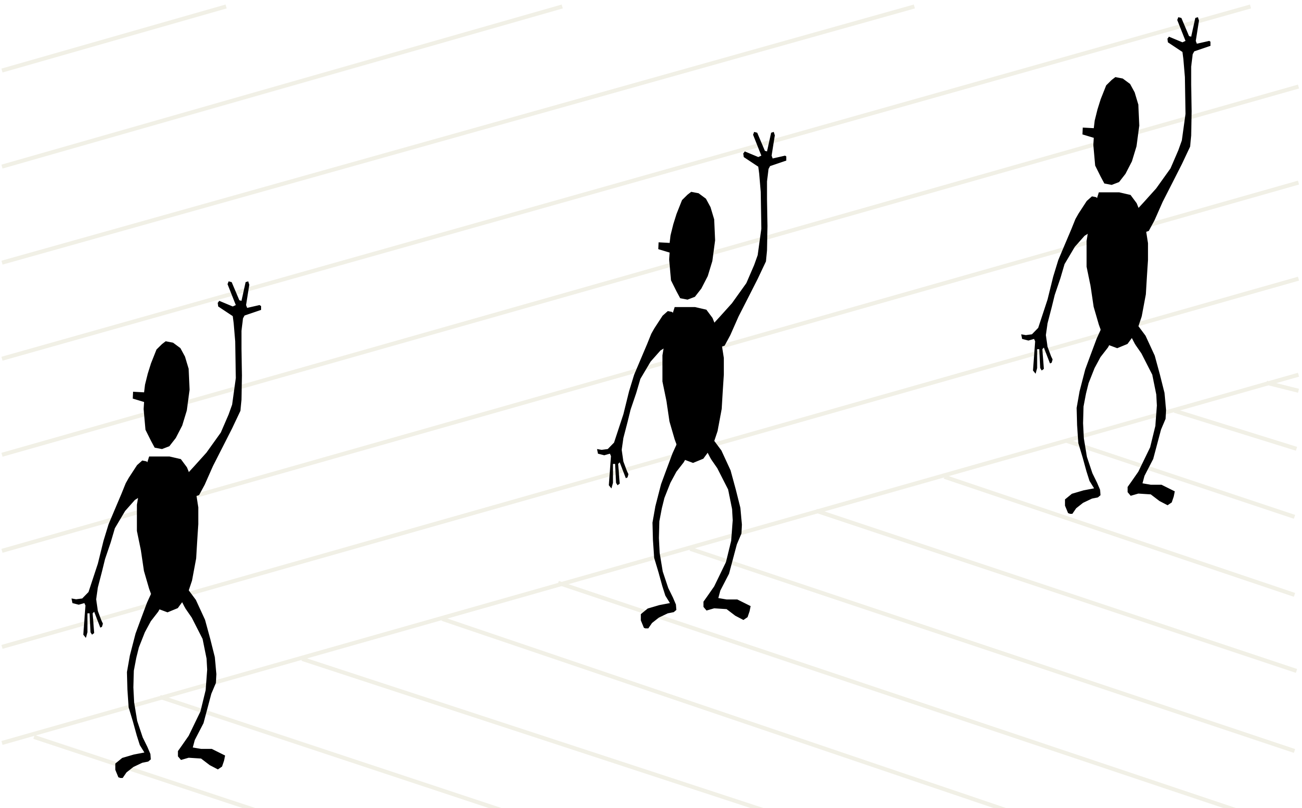


Fun with vanishing points

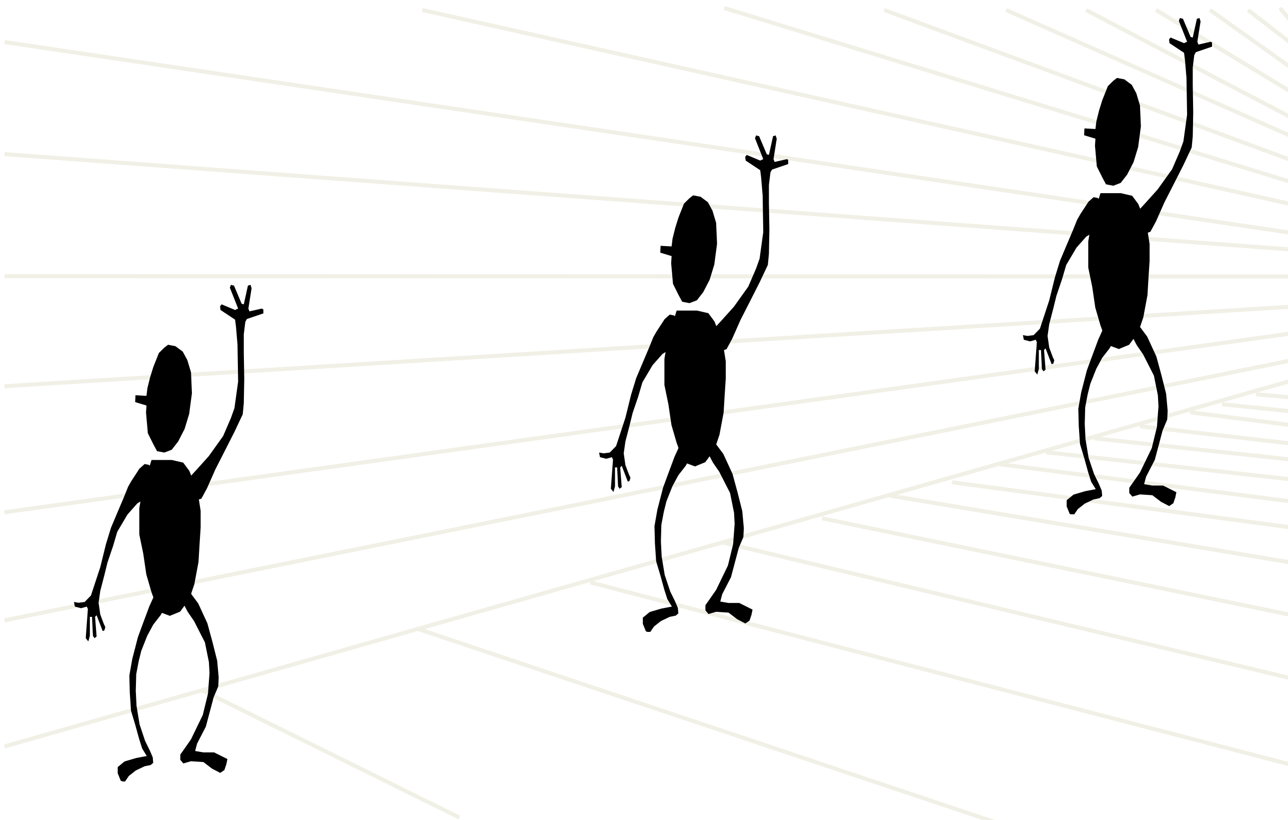




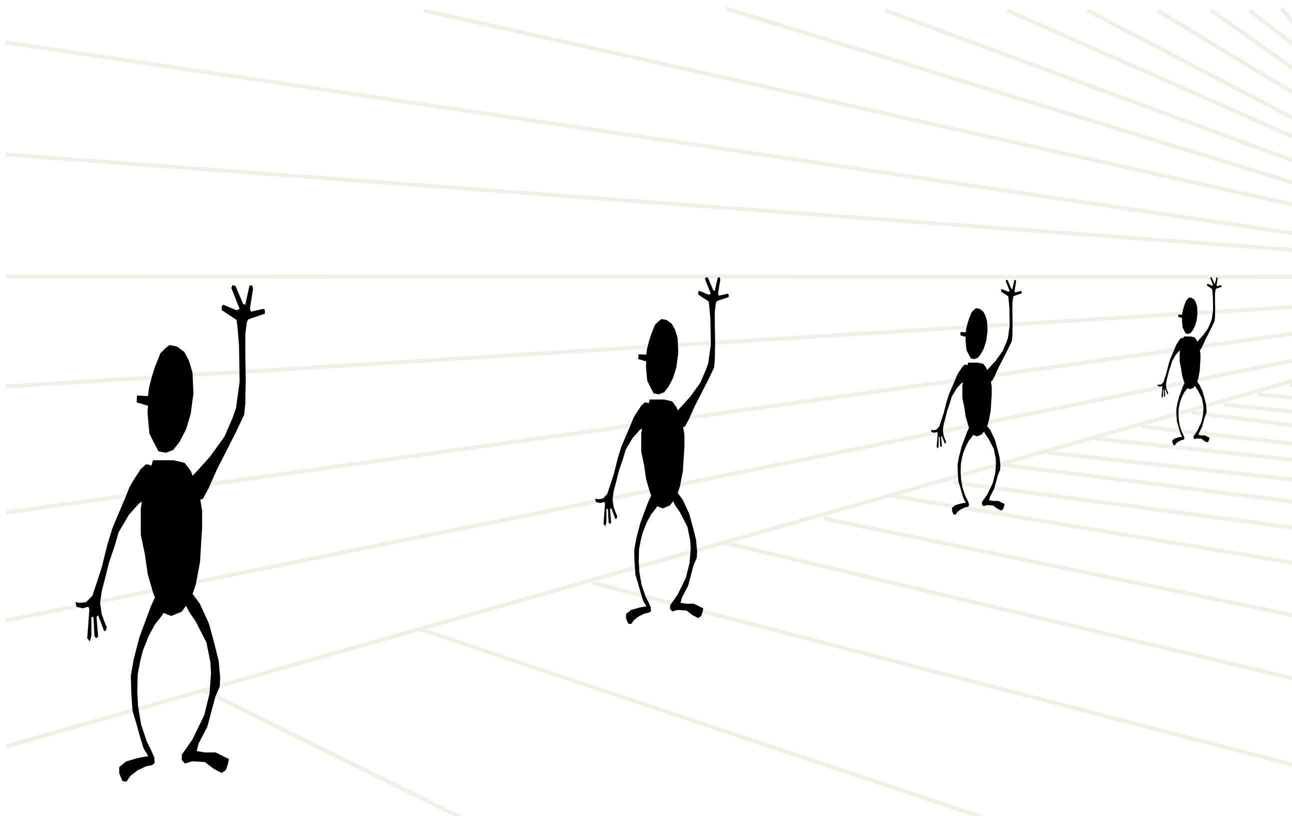
Perspective cues



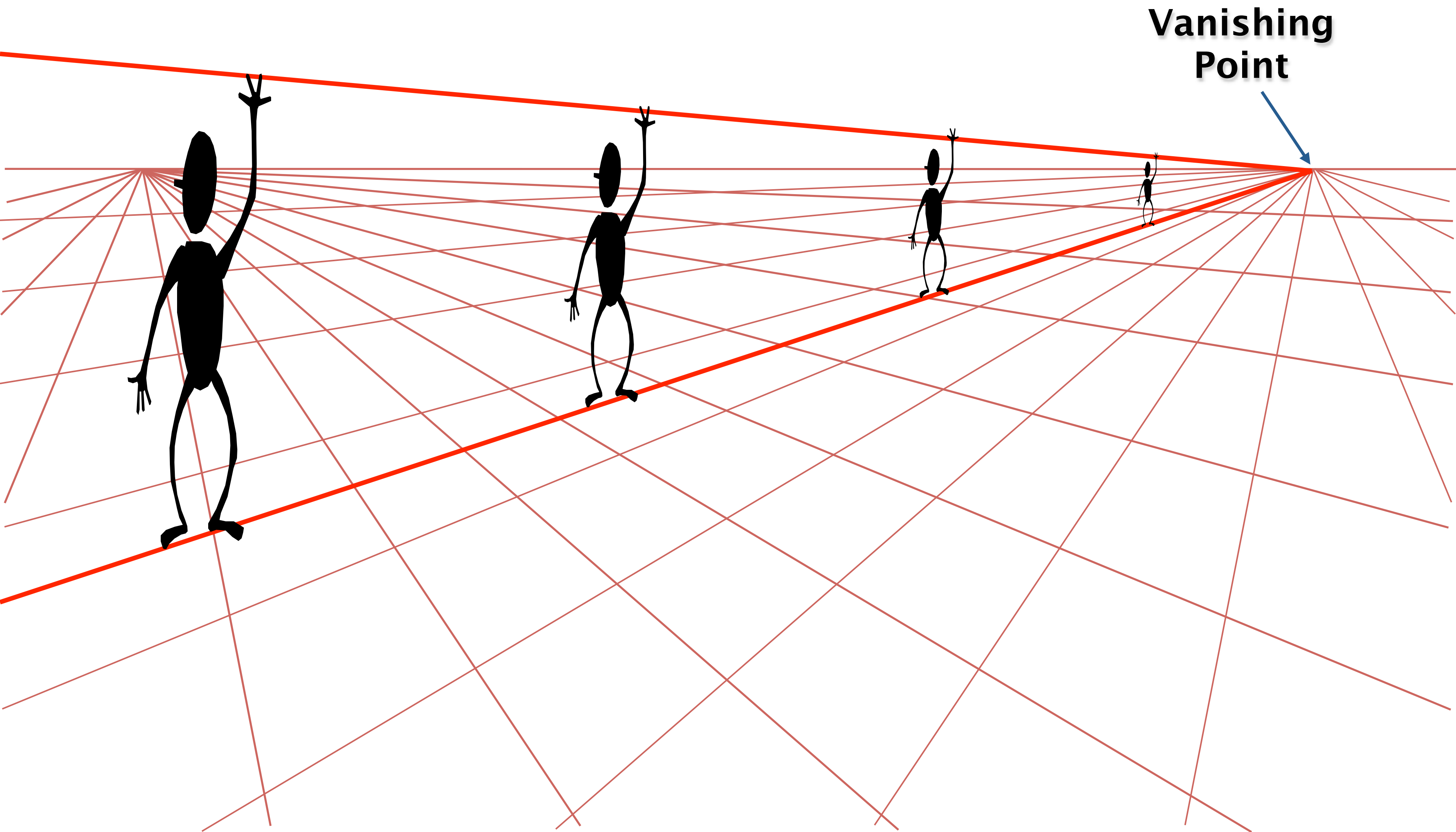
Perspective cues



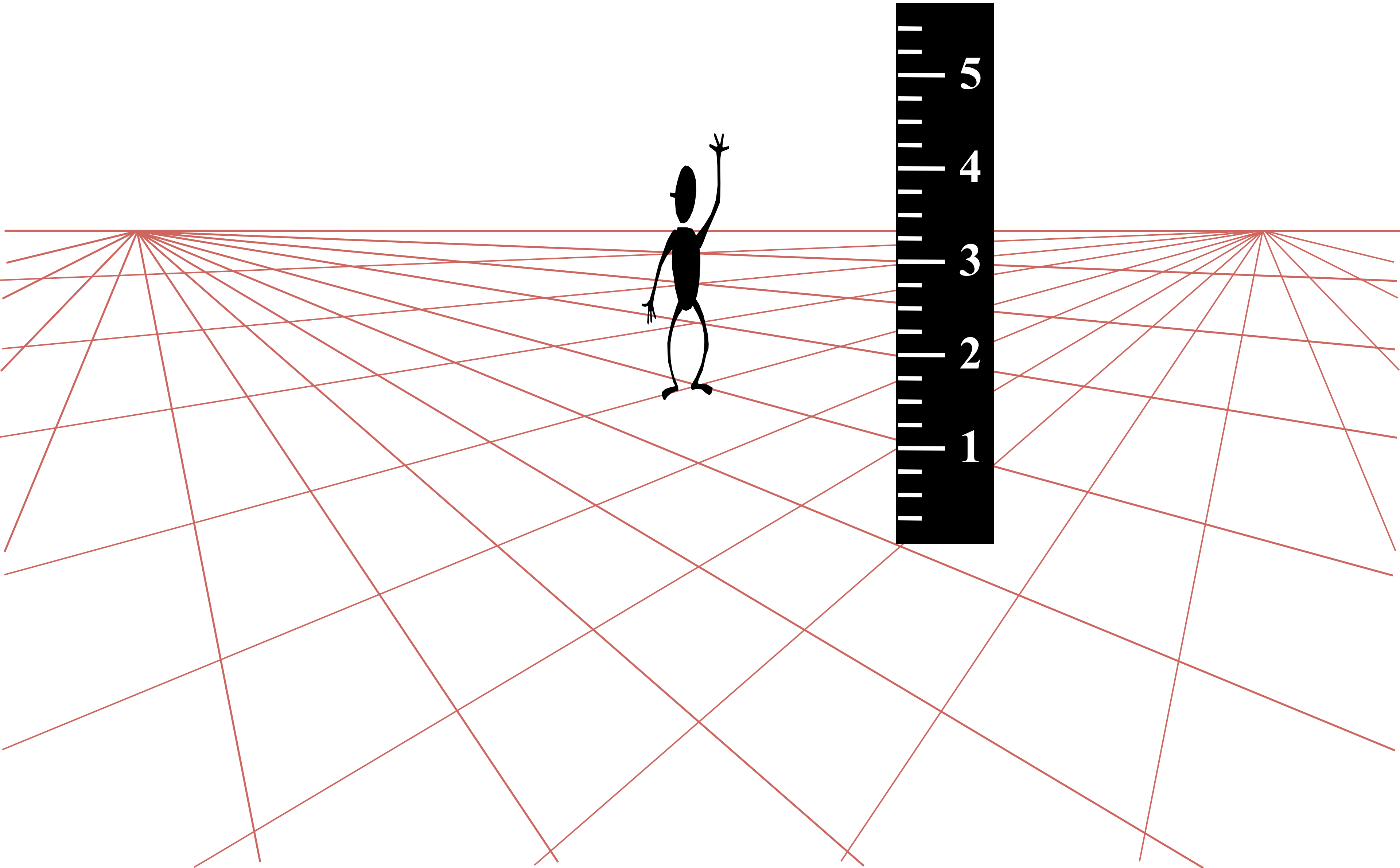
Perspective cues



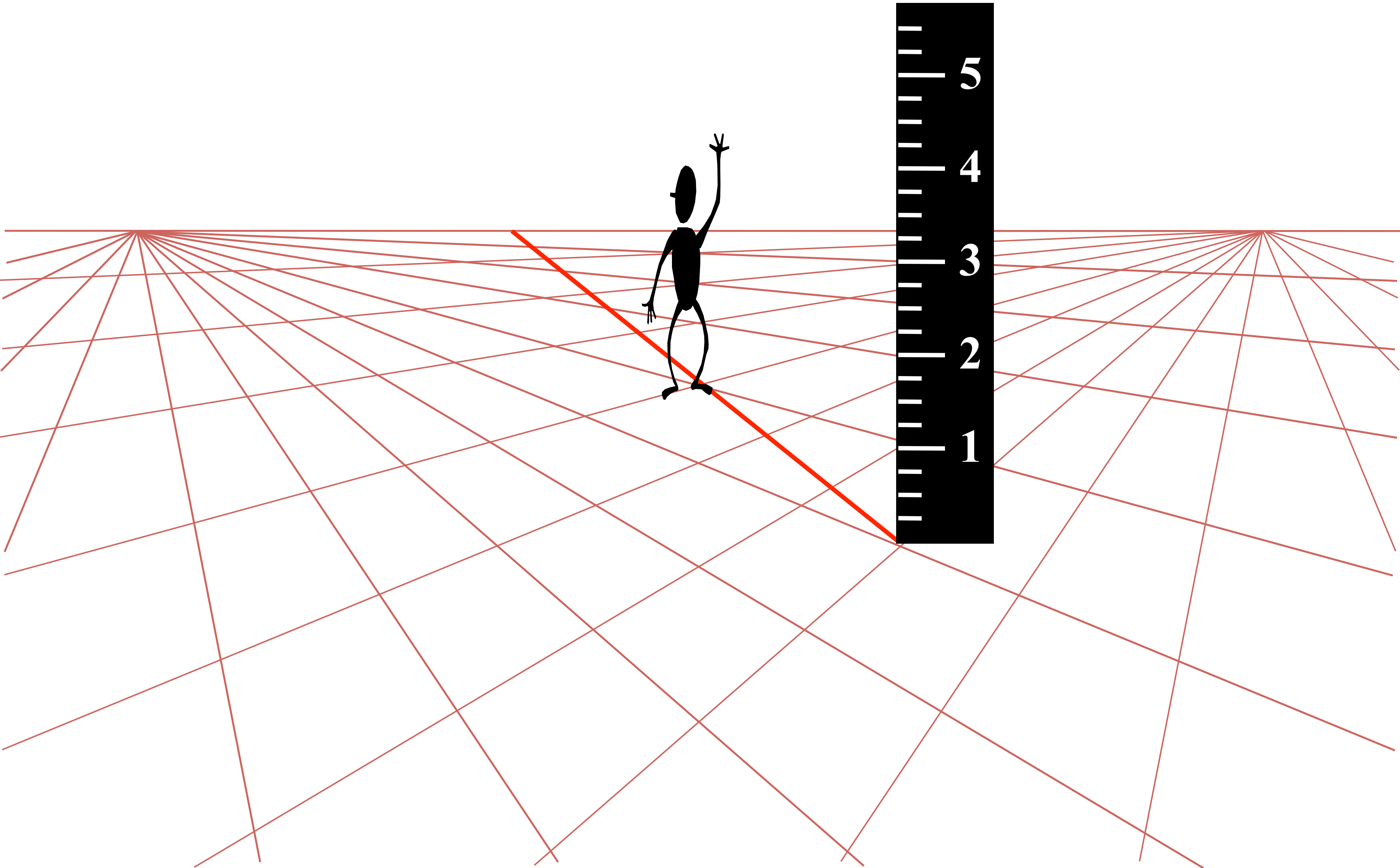
Comparing heights



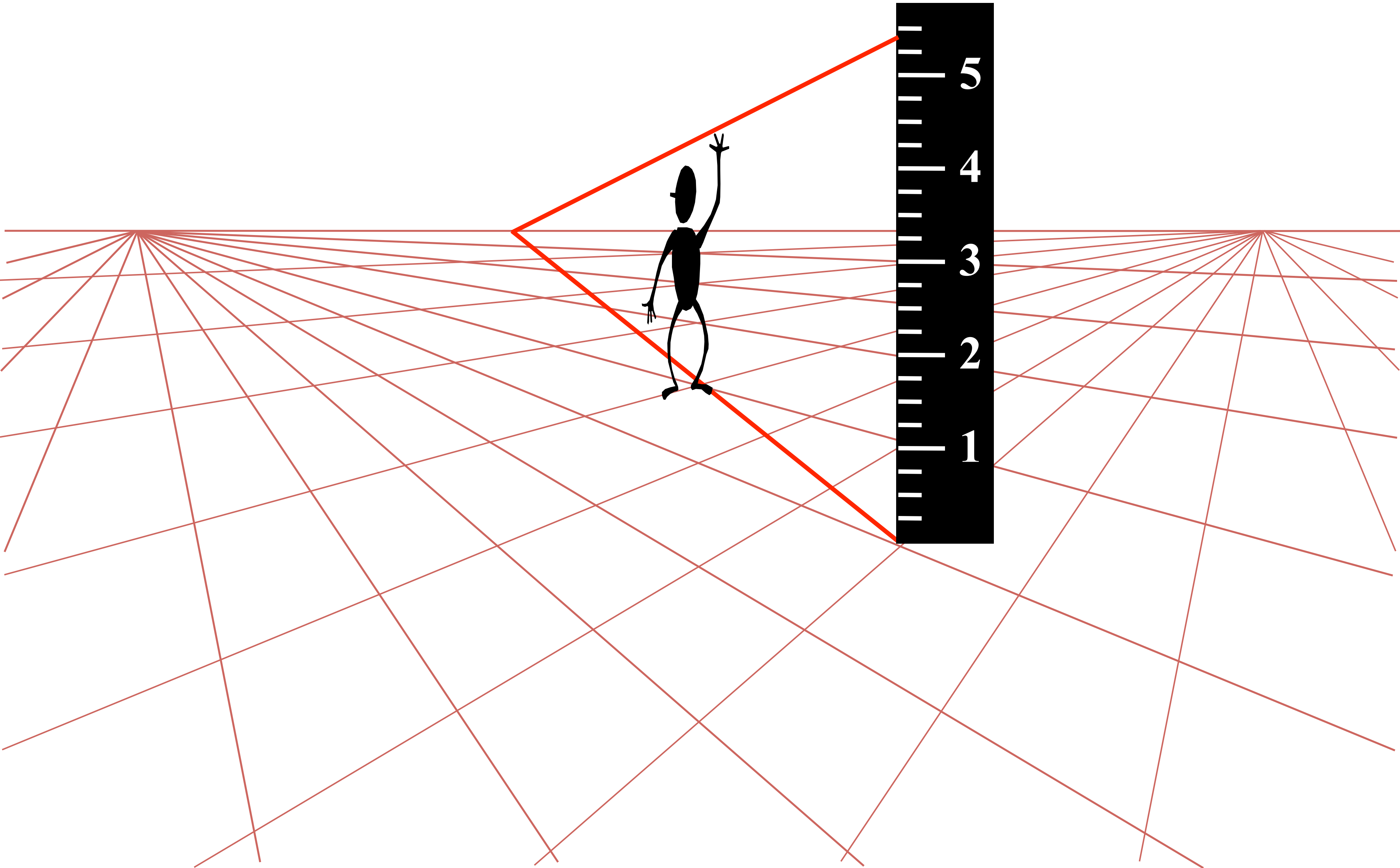
Measuring height



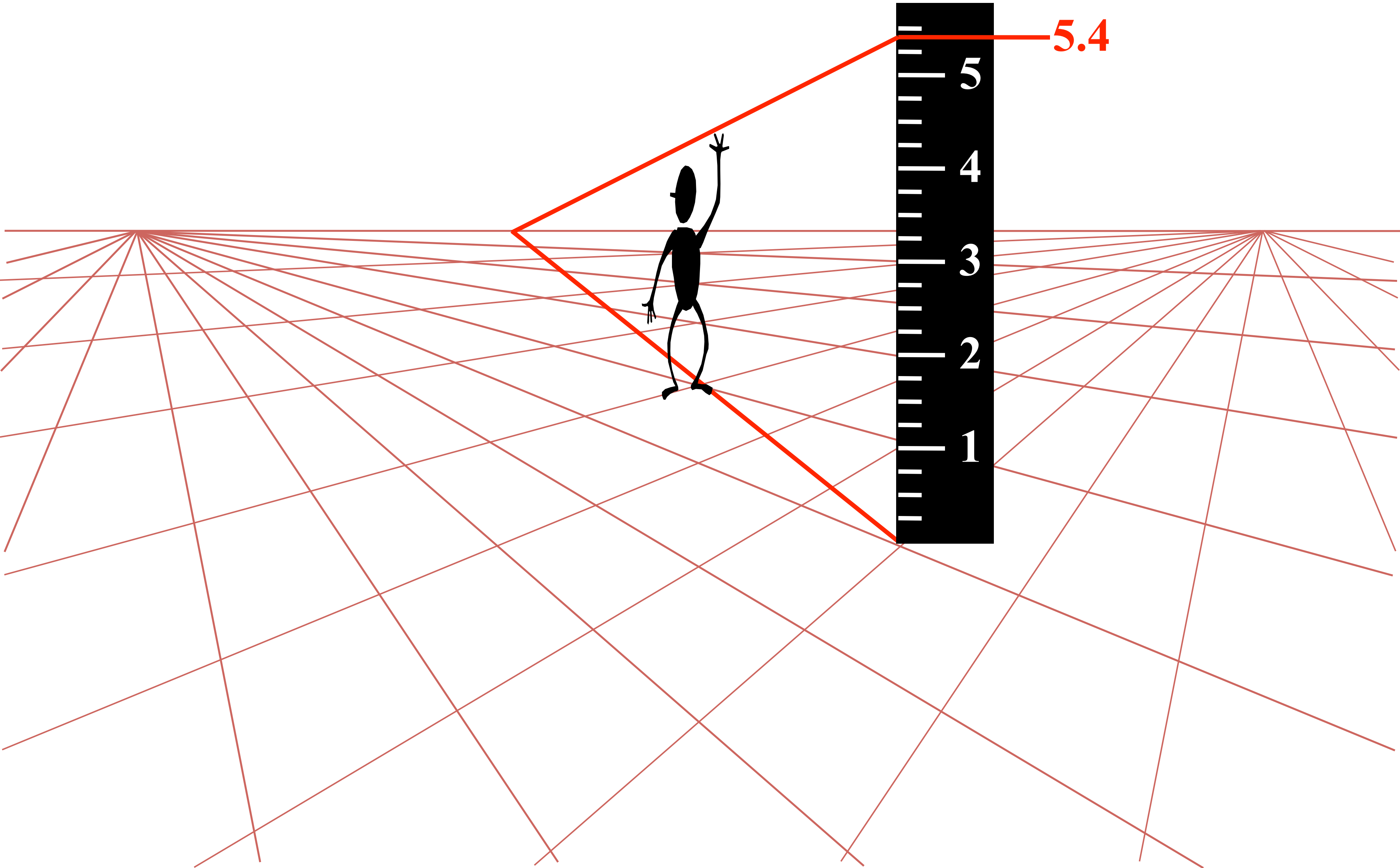
Measuring height



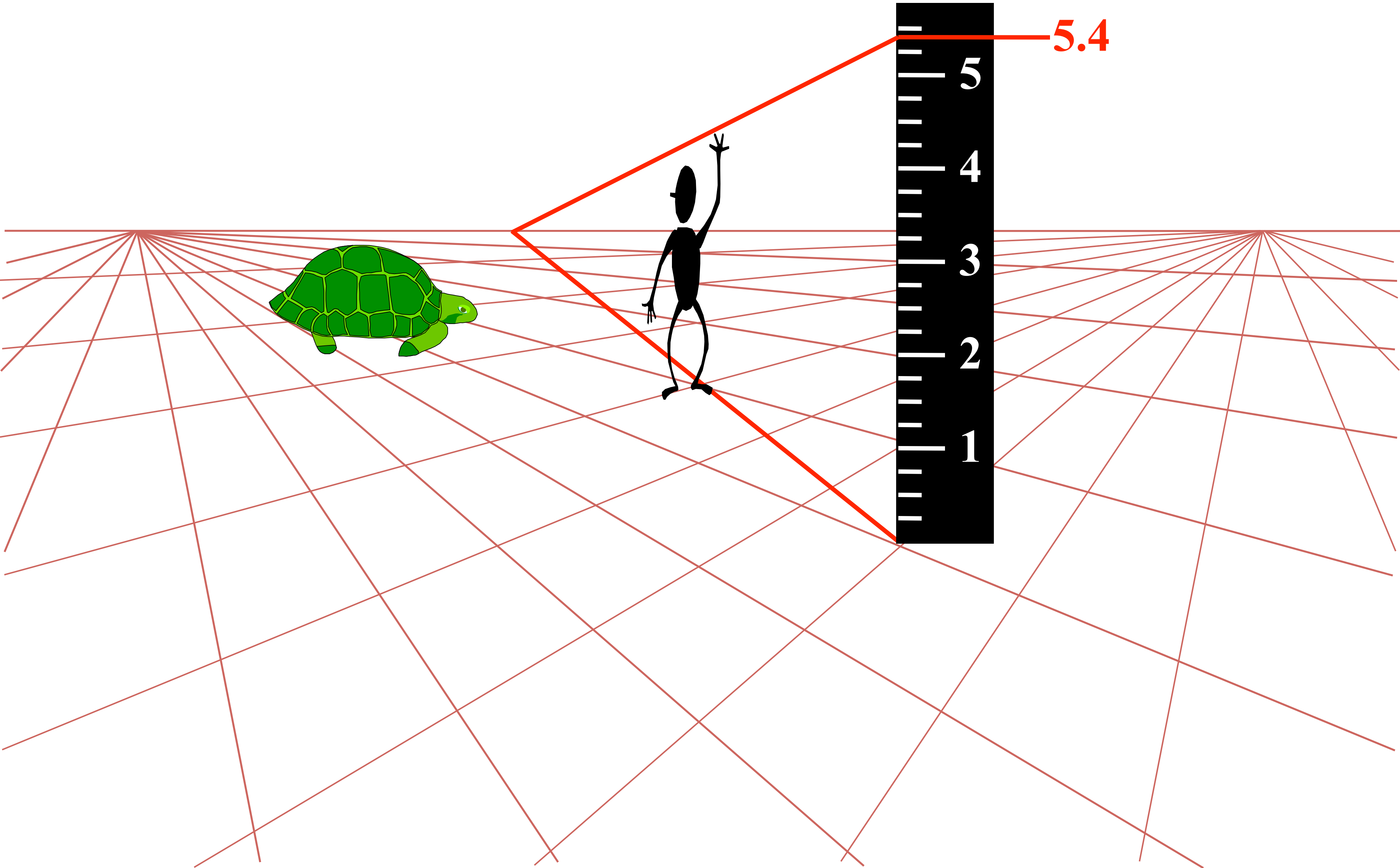
Measuring height



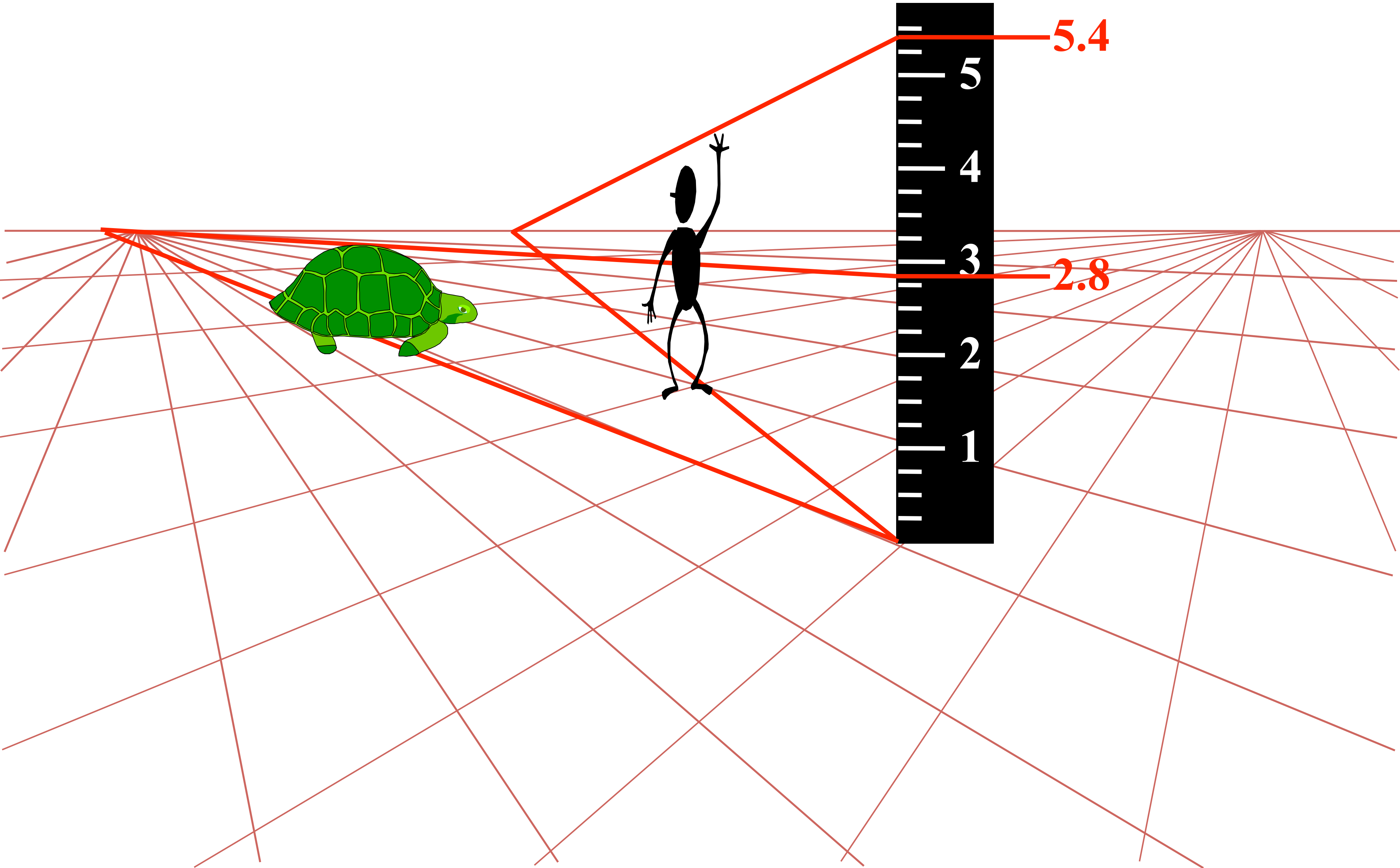
Measuring height



Measuring height

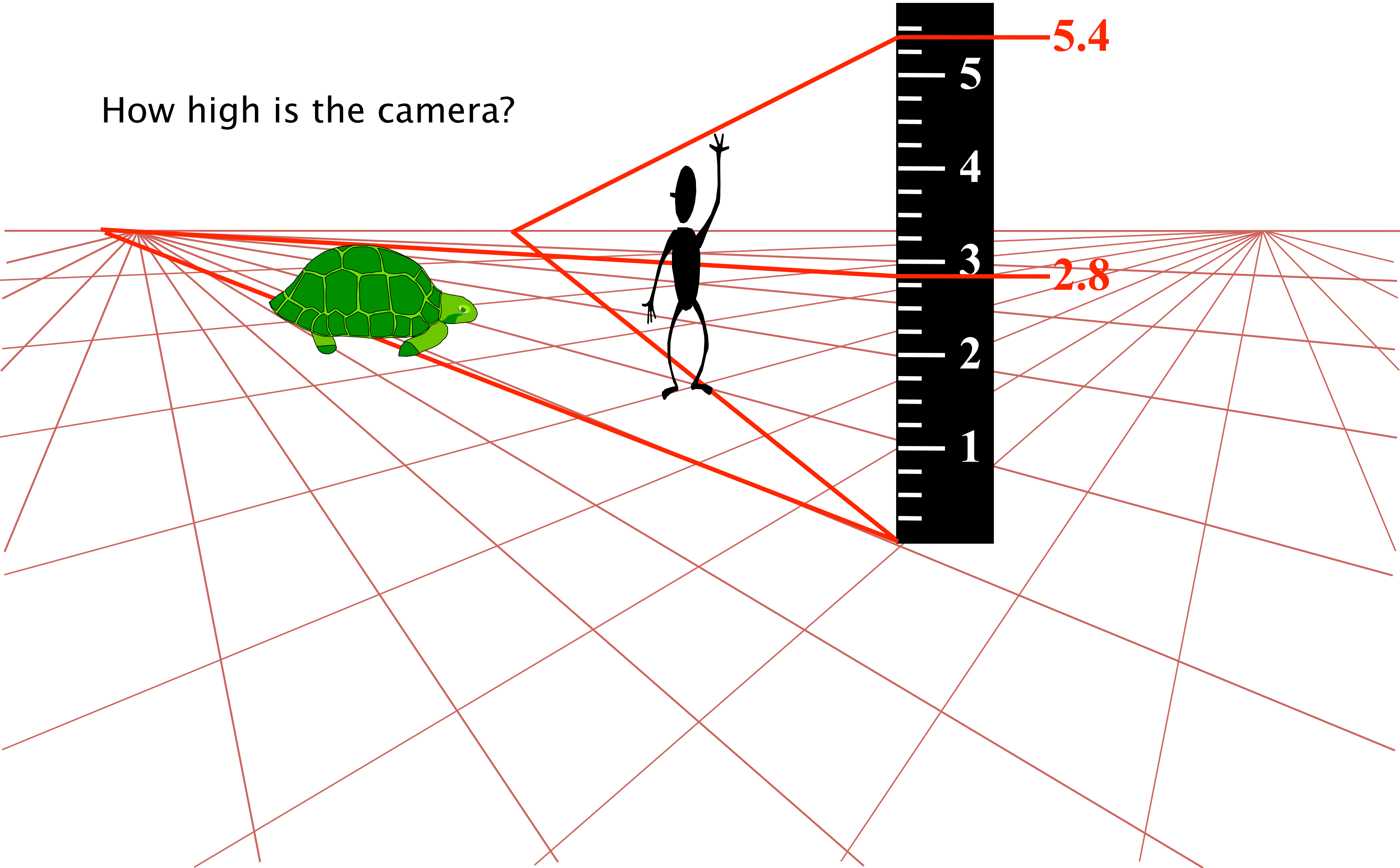


Measuring height



Measuring height

How high is the camera?



Measuring height

How high is the camera?