Camera Models

Textbook

- If you are interested, this book has it all:

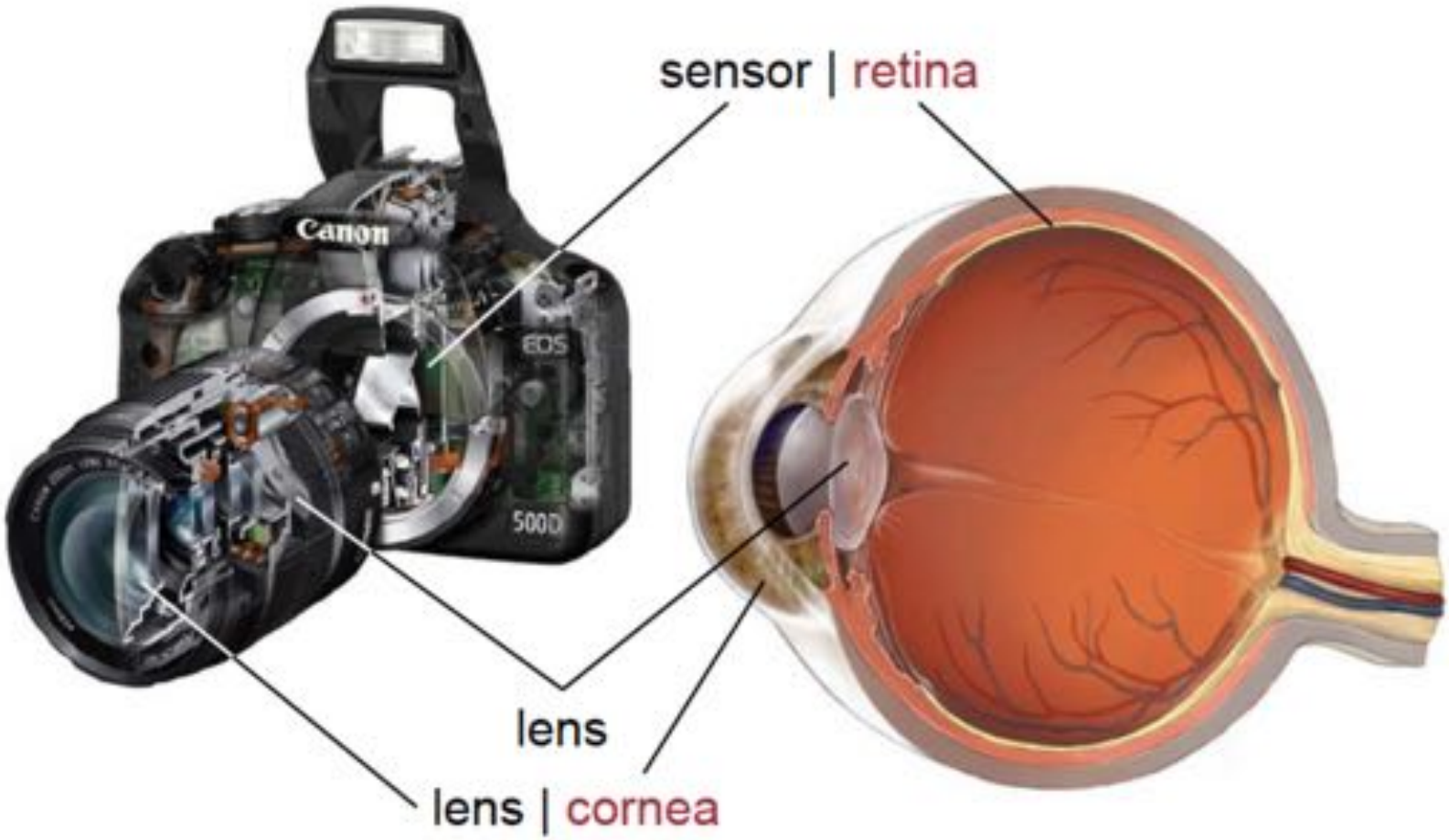
A. Zisserman and R. Hartley

Multiview Geometry

Cambridge University Press, 2003

Camera

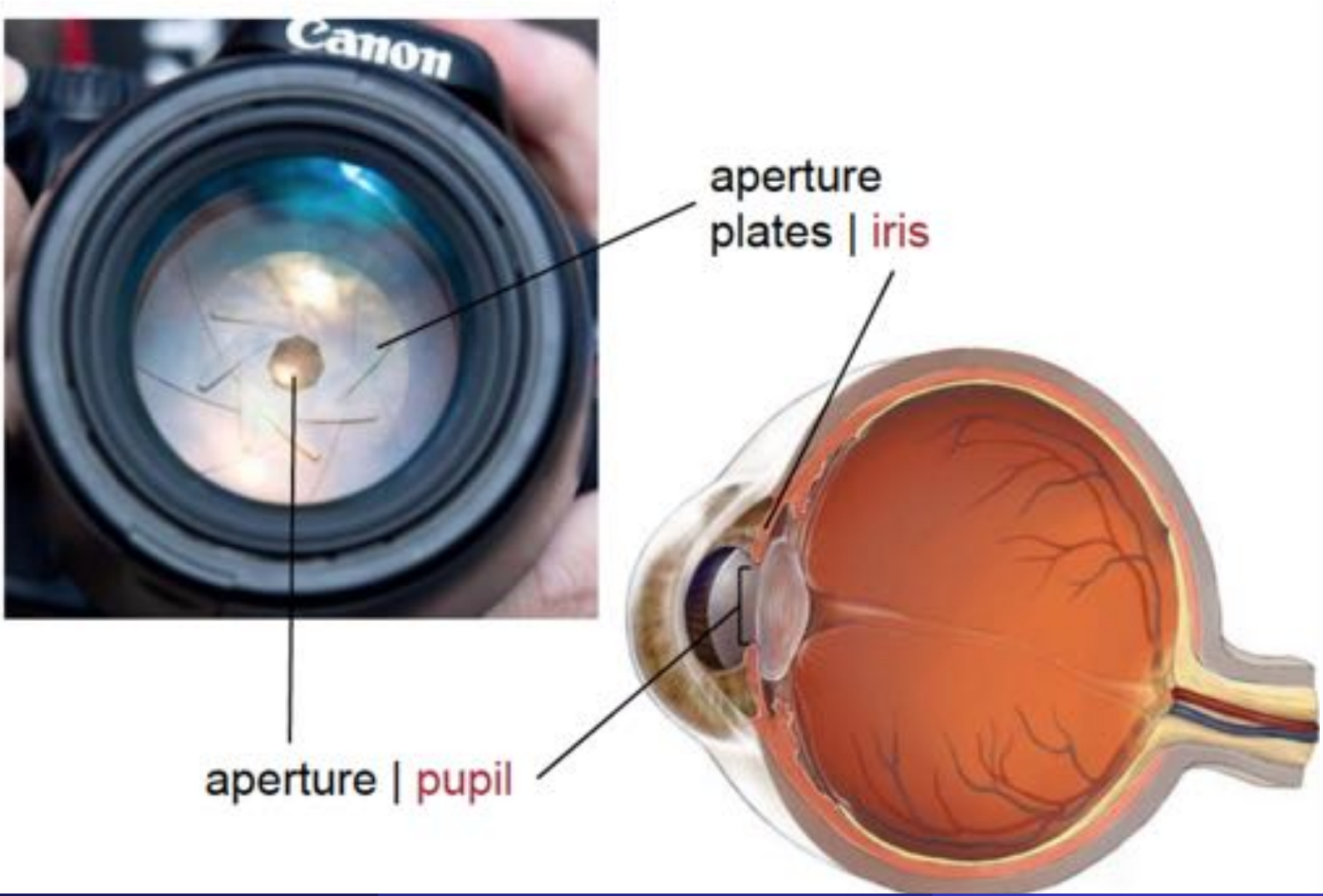
- Camera is structurally similar to the eye



[Source: L.W. Kheng]

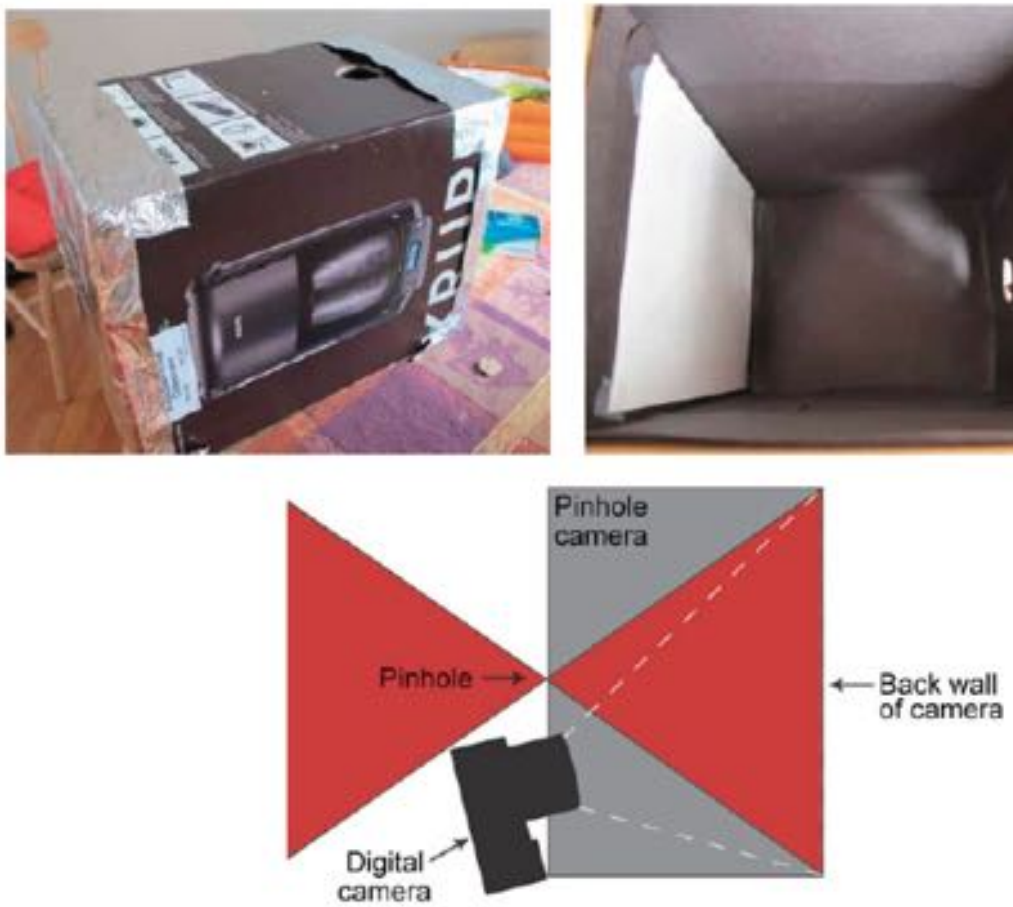
Camera

- Camera is structurally similar to the eye



Camera

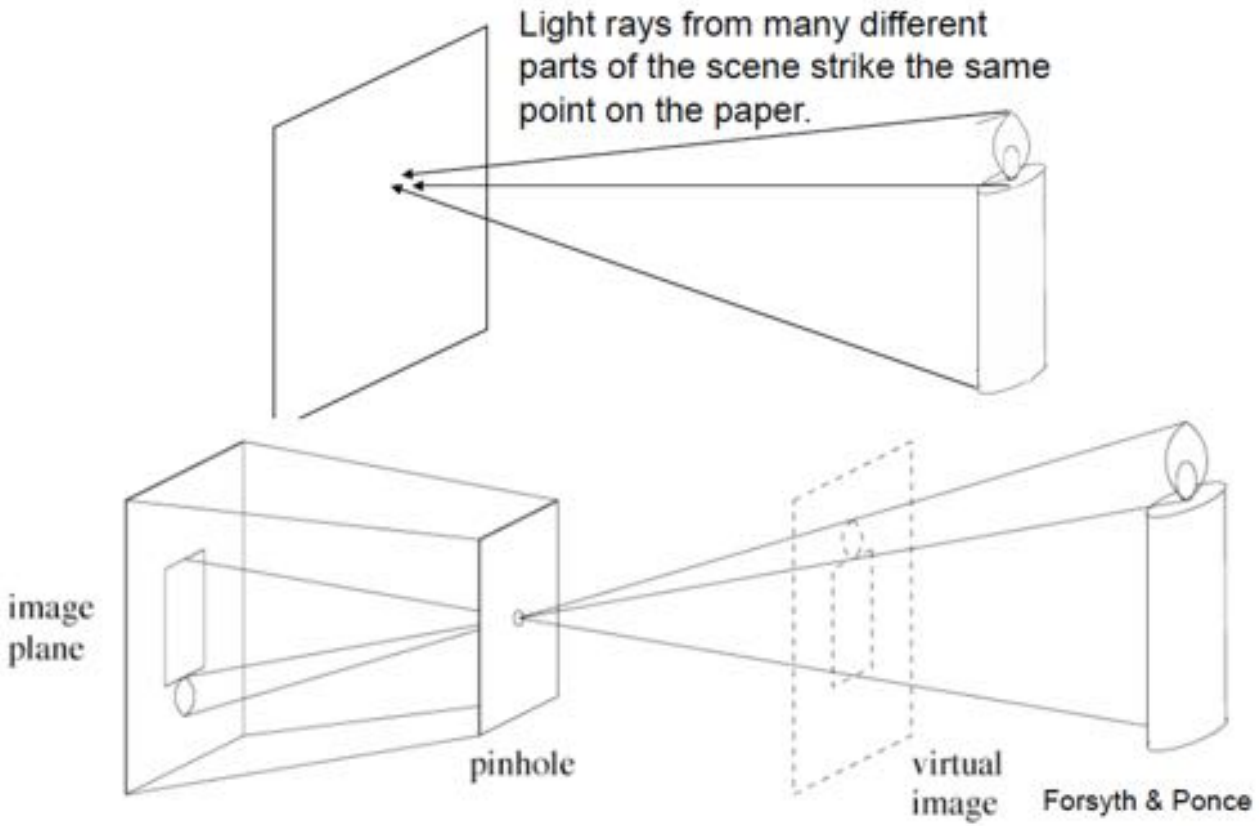
- Remember the pinhole camera from Lecture 2?



[Source: A. Torralba]

Camera

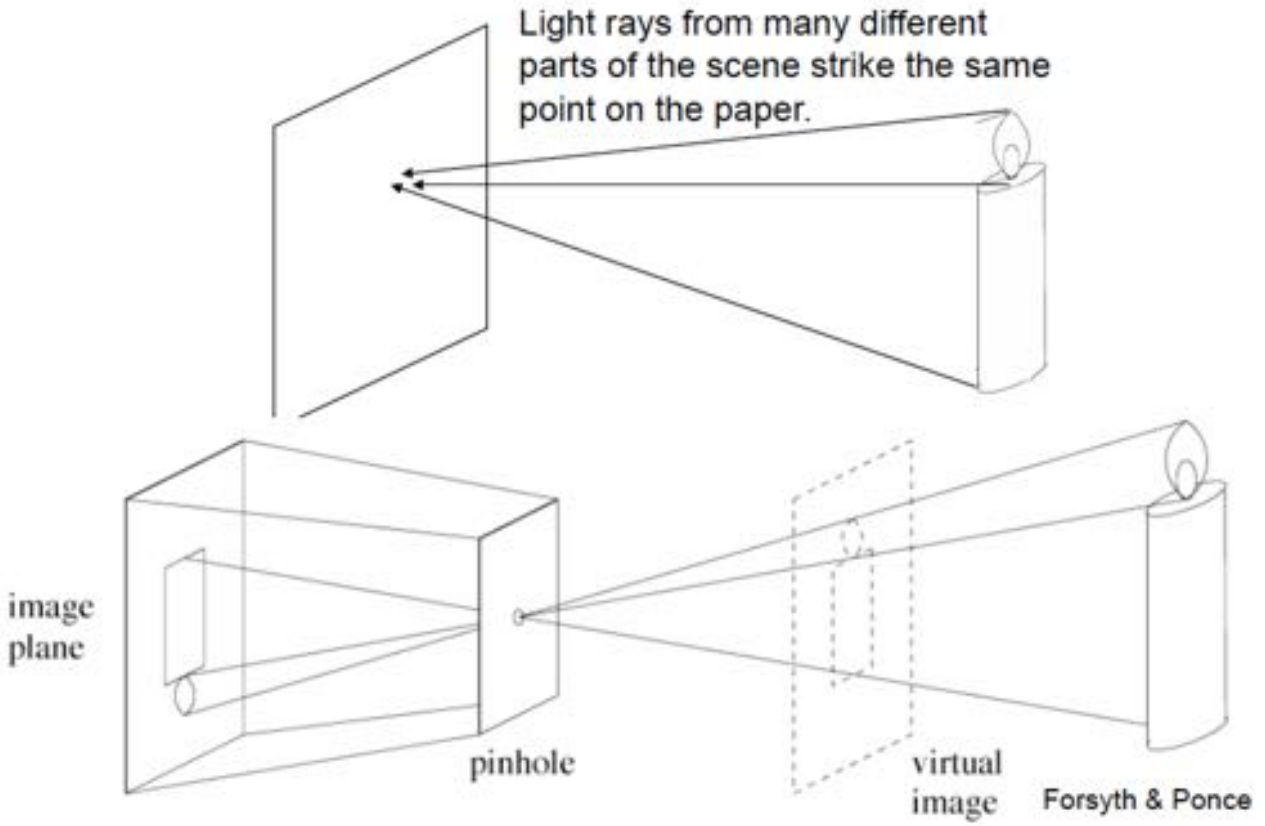
- Remember the pinhole camera from Lecture 2?



[Source: A. Torralba]

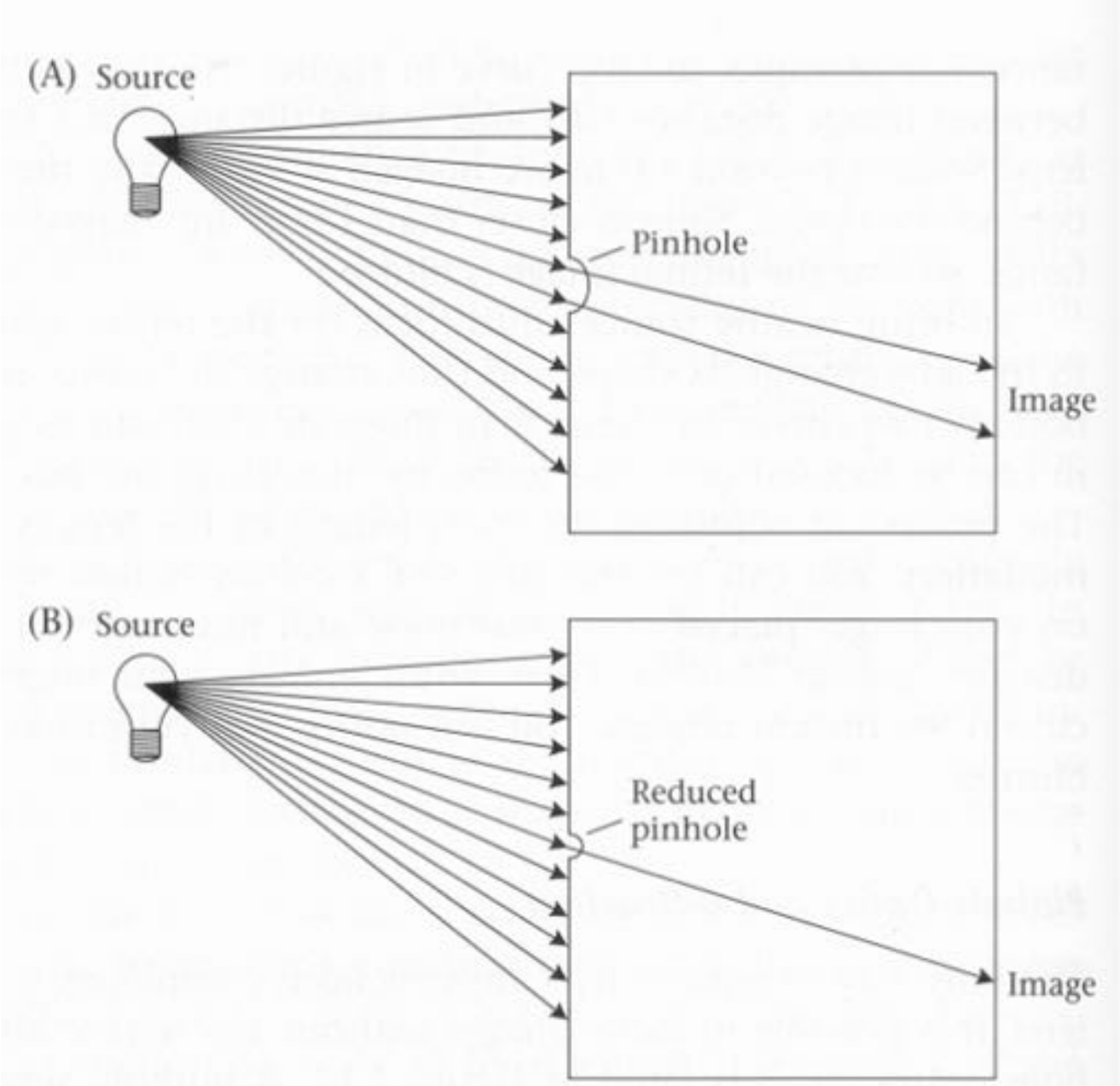
Camera

- Remember the pinhole camera from Lecture 2?
- Size of the pinhole is called **aperture**



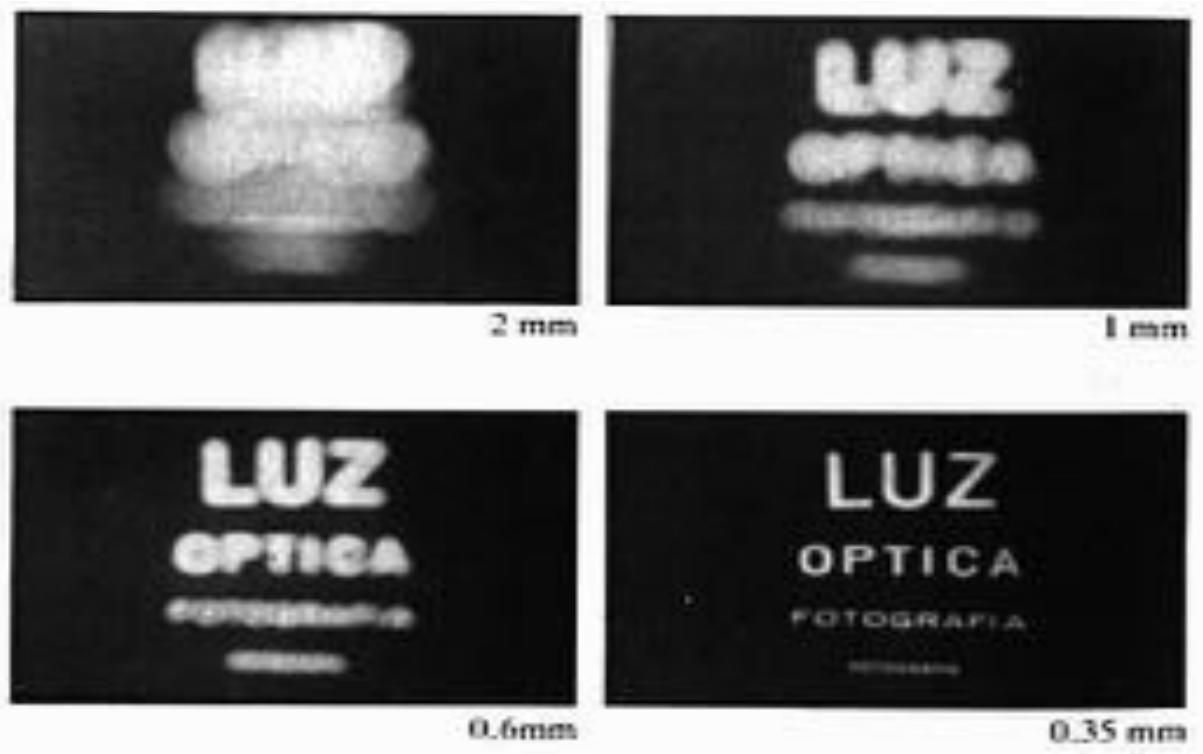
[Source: A. Torralba]

Pinhole Camera



[Source: A. Torralba]

Shrinking the Aperture



Why not make the aperture as small as possible?

- Less light gets through
- Diffraction effects...

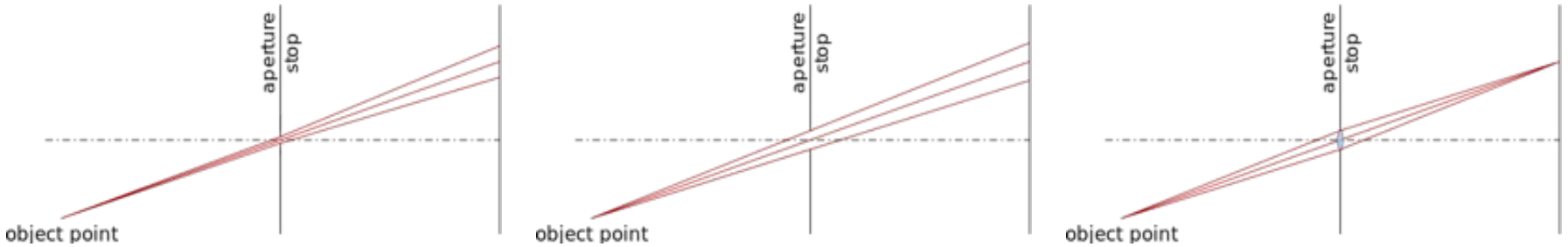
[Source: N. Snavely]

Shrinking the Aperture

[Source: N. Snavely]

Adding a Lens

[Pic from Wikipedia]

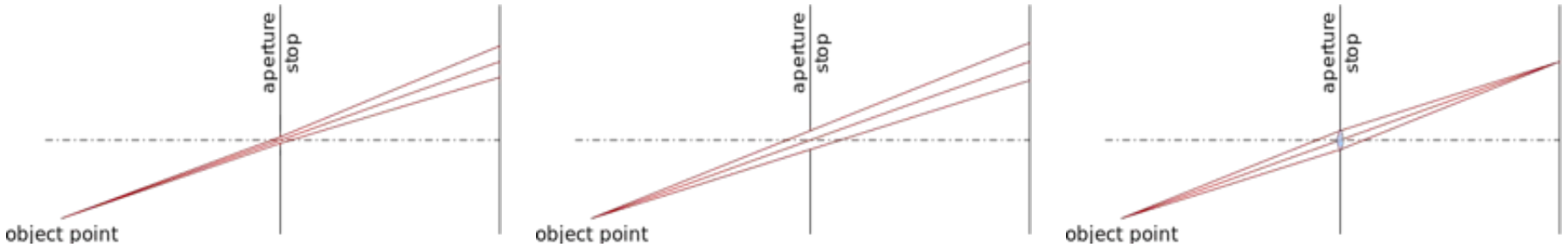


Small pinhole | Big pinhole | Lens

- A lens focuses light onto the film
- There is a specific distance at which objects are **in focus**

Adding a Lens

[Pic from Wikipedia]



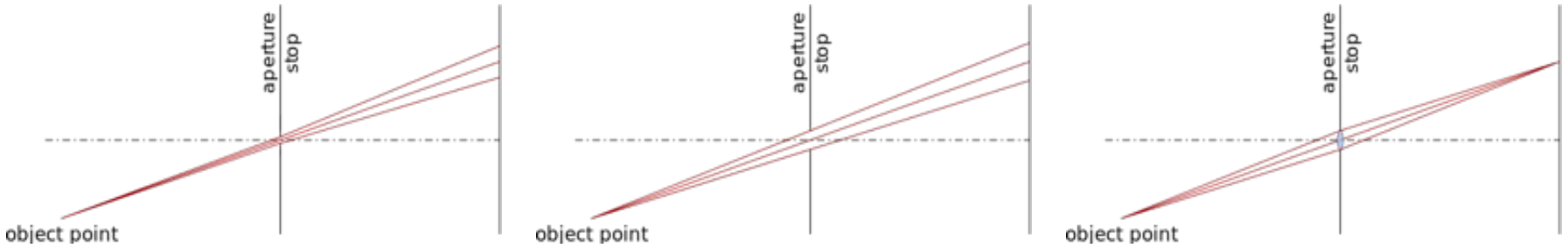
Small pinhole | Big pinhole | Lens

- A lens focuses light onto the film
- There is a specific distance at which objects are **in focus**
- Changing the shape of the lens changes this distance

[Source: N. Snavely]

Adding a Lens

[Pic from Wikipedia]



Small pinhole | Big pinhole | Lens

- A lens focuses light onto the film
- There is a specific distance at which objects are **in focus**
- Changing the shape of the lens changes this distance

[Source: N. Snavely]

Some "Cameras" Have Bigger Lenses than Others

http://www.use.com/images/s_2/thick_glasses_13b6941623c255ff400a_1.jpg?

Imaging

- Images are 2D projections of real world scene
- Images capture two kinds of information:
  - Geometric: positions, points, lines, curves, etc.
  - Photometric: intensity, color
- Complex 3D-2D relationships
- Camera models approximate these relationships

[Source: L.W. Kheng]

Projection

[Source: N. Snavely]

Projection

[Source: N. Snavely]

3D to 2D Projection

- How are 3D primitives projected onto the image plane?
- We can do this using a linear 3D to 2D projection matrix

3D to 2D Projection

- How are 3D primitives projected onto the image plane?
- We can do this using a linear 3D to 2D projection matrix
- Different types:
  - Perspective projection
  - Orthographic projection
  - Scaled orthographic projection
  - Paraperspective projection

[source: R. Urtasun]

3D to 2D Projection

- How are 3D primitives projected onto the image plane?
- We can do this using a linear 3D to 2D projection matrix
- Different types:
  - Perspective projection
  - Orthographic projection
  - Scaled orthographic projection
  - Paraperspective projection

[source: R. Urtasun]

3D to 2D Projection

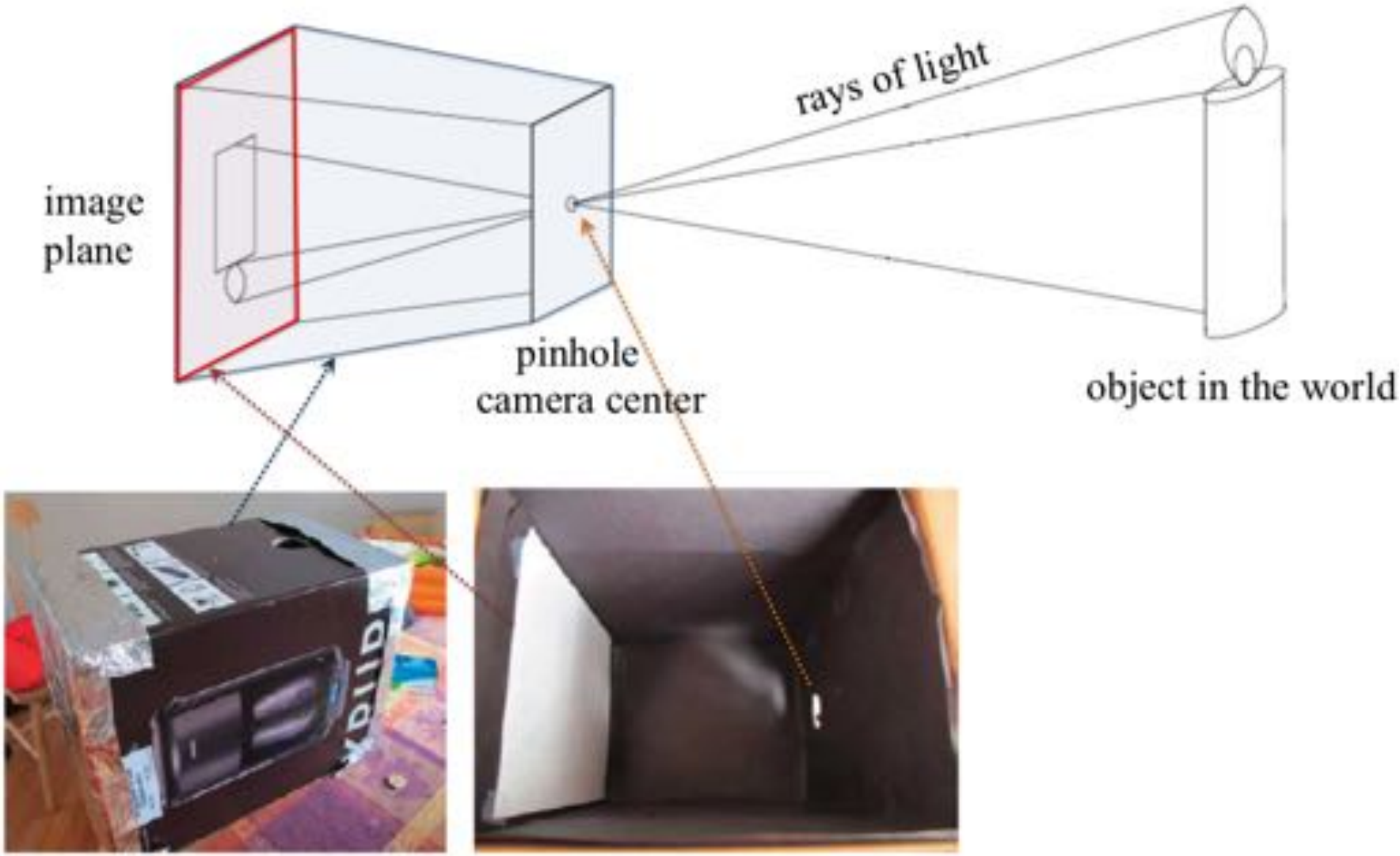
- How are 3D primitives projected onto the image plane?
- We can do this using a linear 3D to 2D projection matrix
- Different types, most common:
  - Perspective projection
  - Orthographic projection
  - Scaled orthographic projection
  - Paraperspective projection

[source: R. Urtasun]

Modeling Projection

[Pics from: A. Torralba, Forsyth & Ponce]

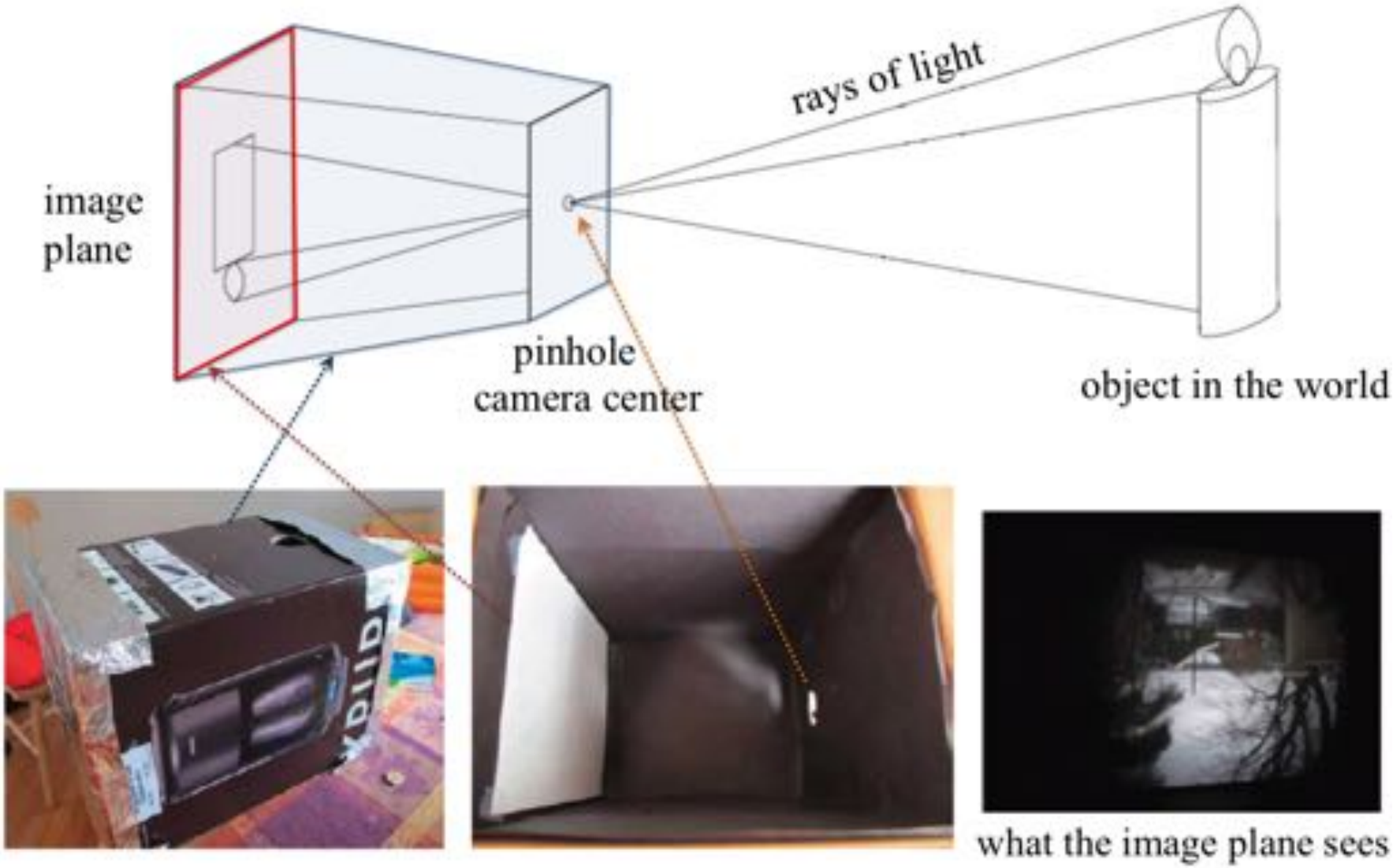
- We will use the pinhole model as an approximation



Modeling Projection

[Pics from: A. Torralba, Forsyth & Ponce]

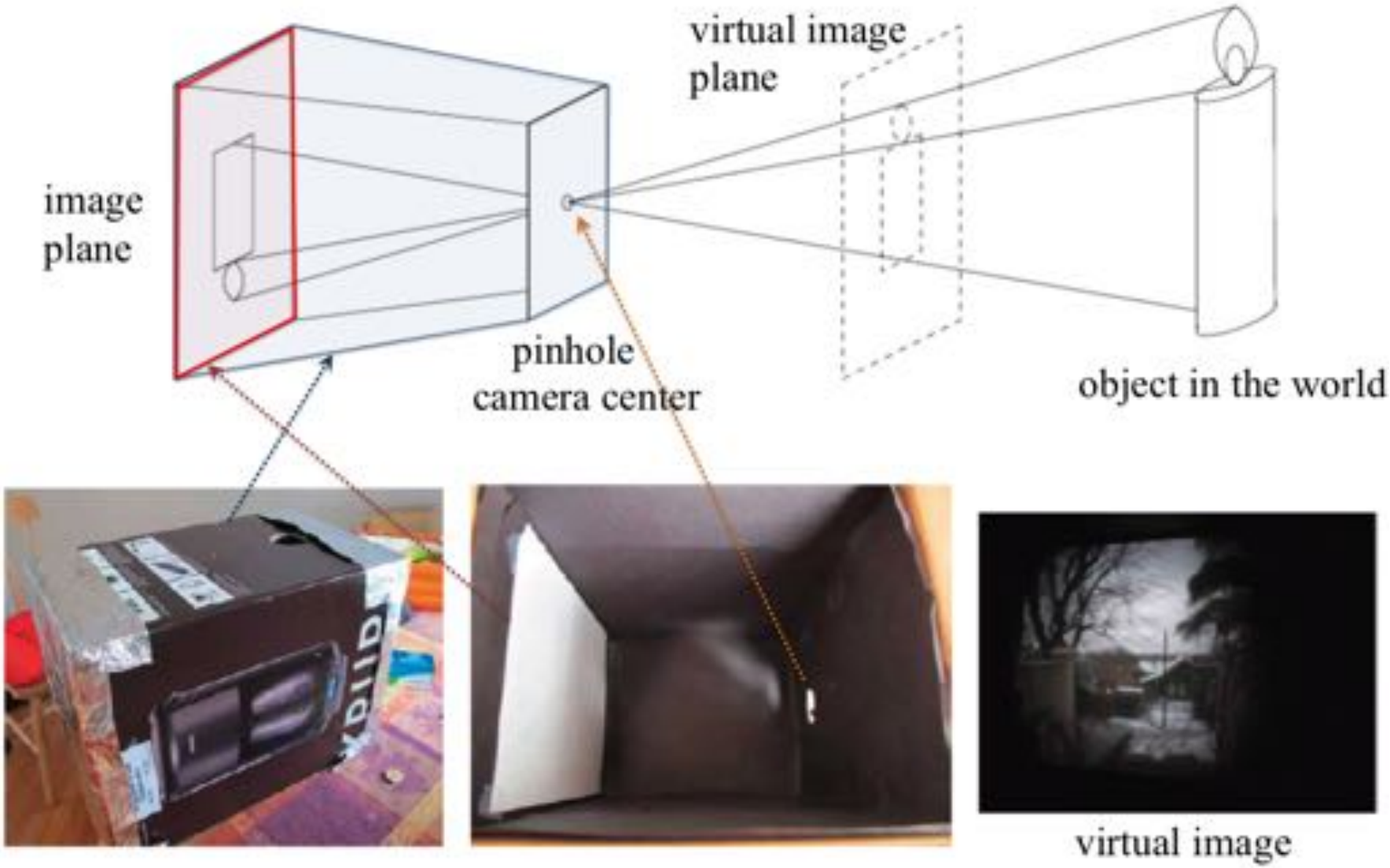
- We will use the pinhole model as an approximation



Modeling Projection

[Pics from: A. Torralba, Forsyth & Ponce]

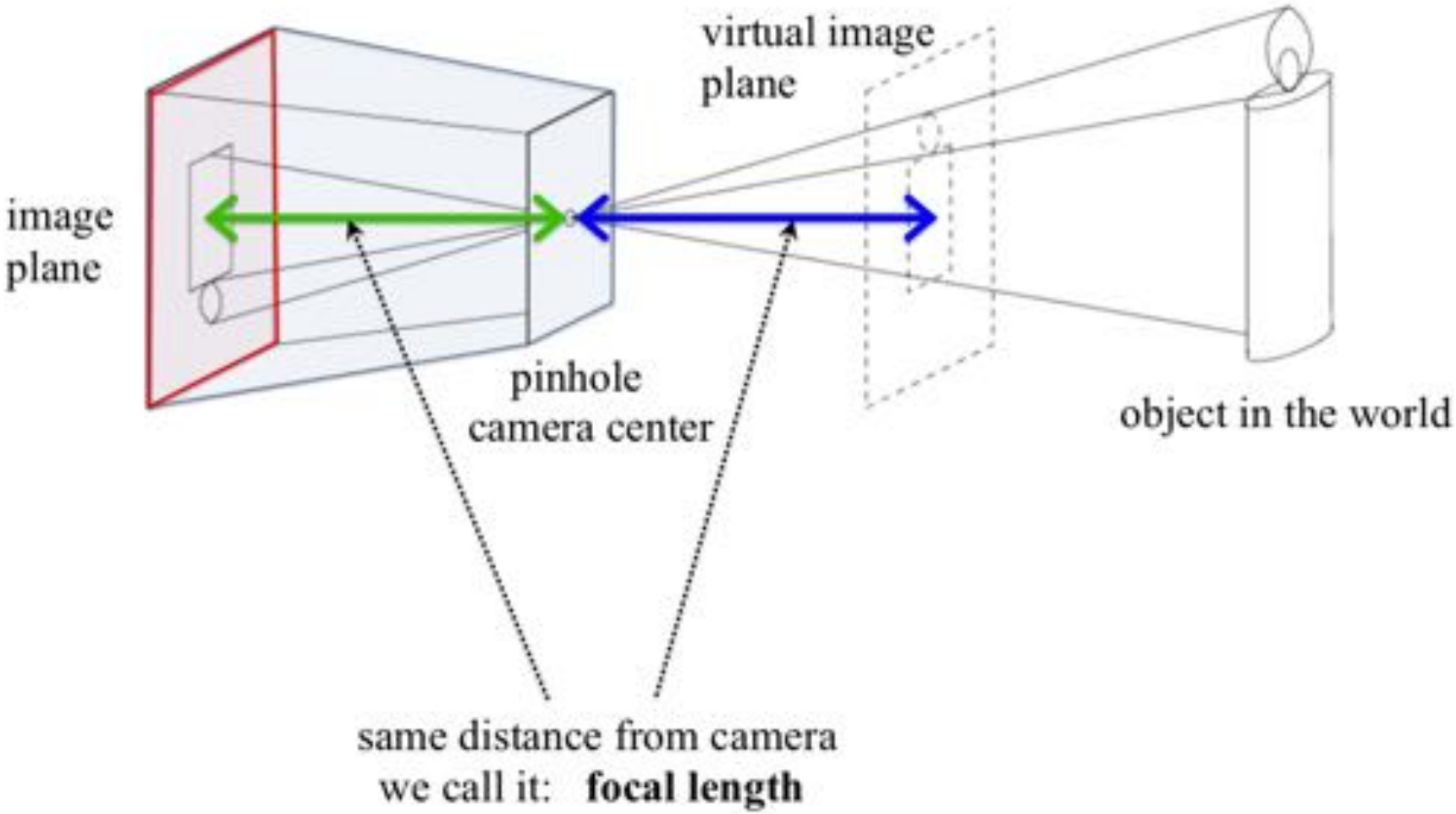
- We will use the pinhole model as an approximation



Modeling Projection

[Pics from: A. Torralba, Forsyth & Ponce]

- We will use the pinhole model as an approximation



Focal Length

- Can be thought of as **zoom**
- Related to the **field of view**

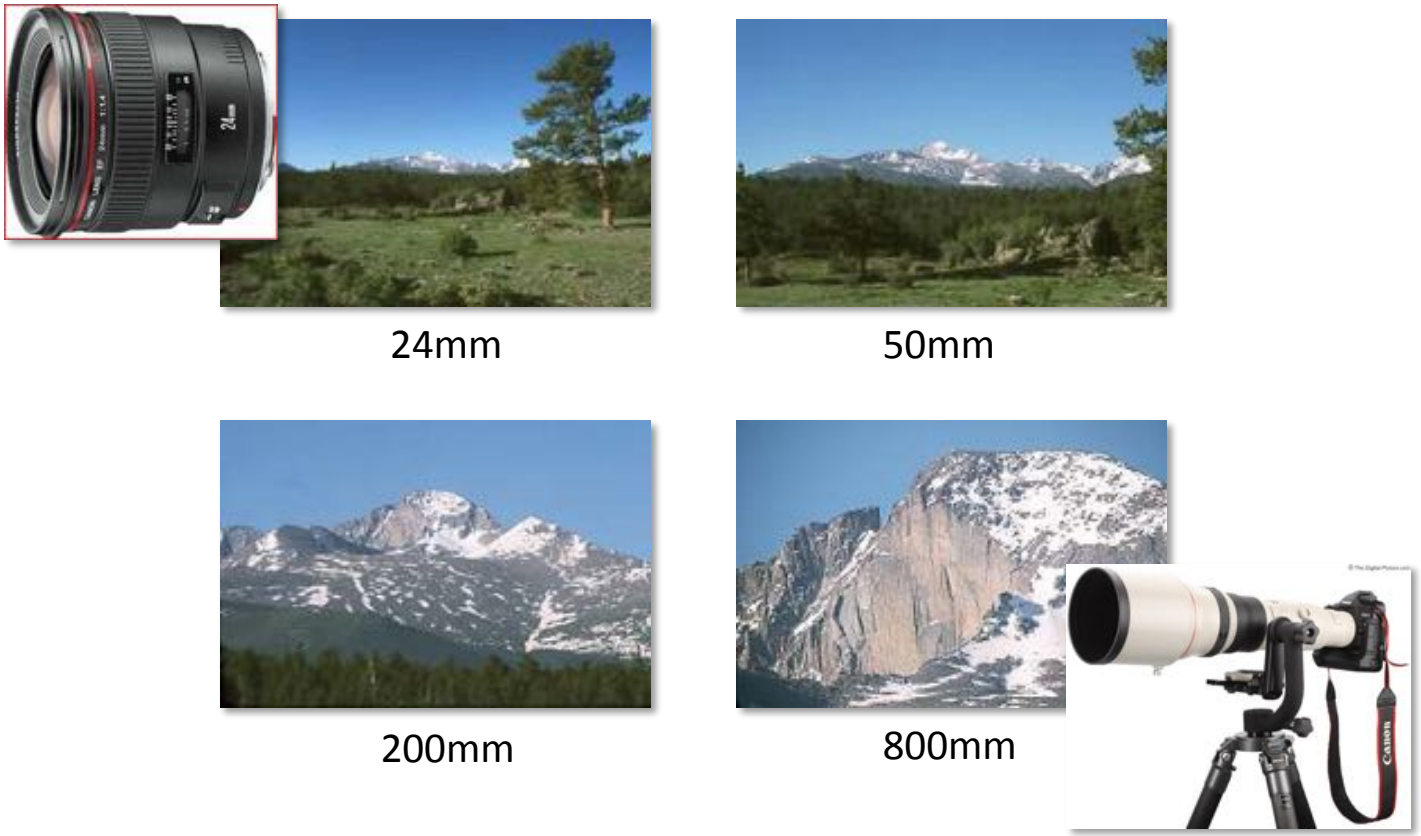


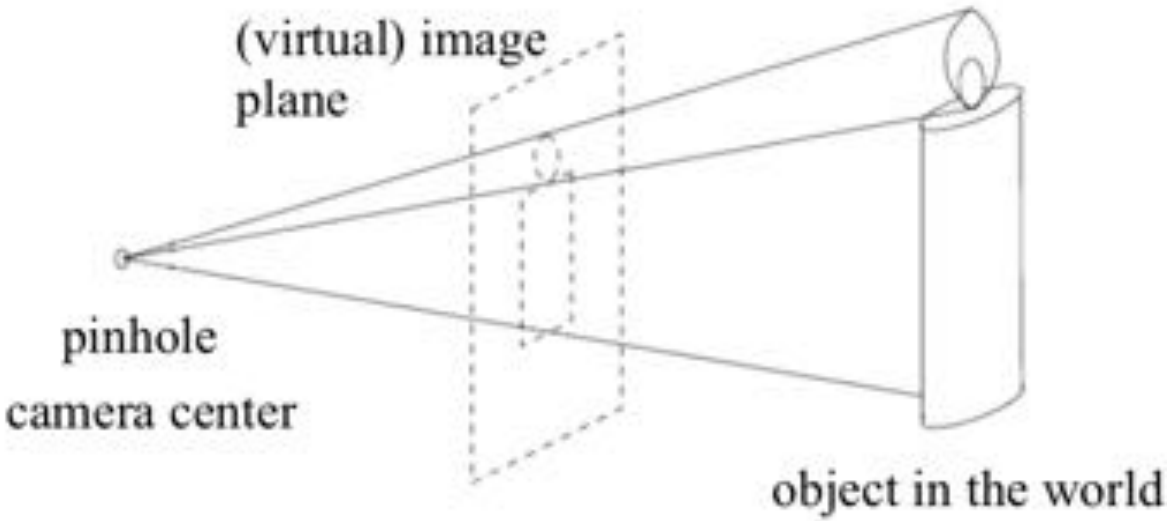
Figure: Image from N. Snavely

[Source: N. Snavely, slide credit: R. Urtasun]

Modeling Projection

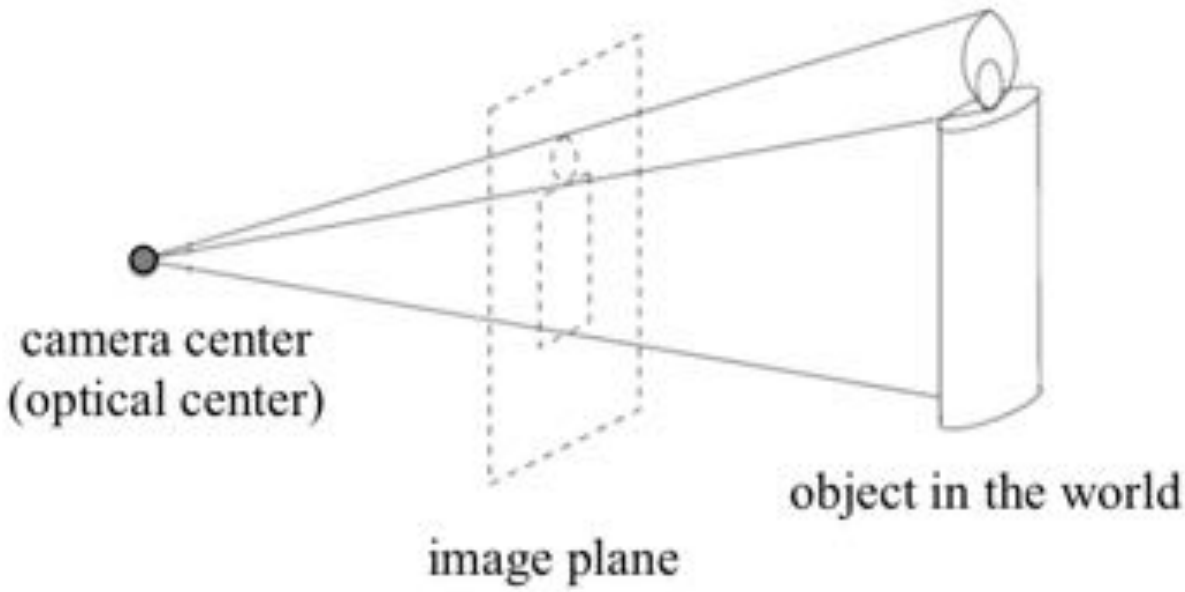
[Pics from: A. Torralba, Forsyth & Ponce]

- We will use the pinhole model as an approximation



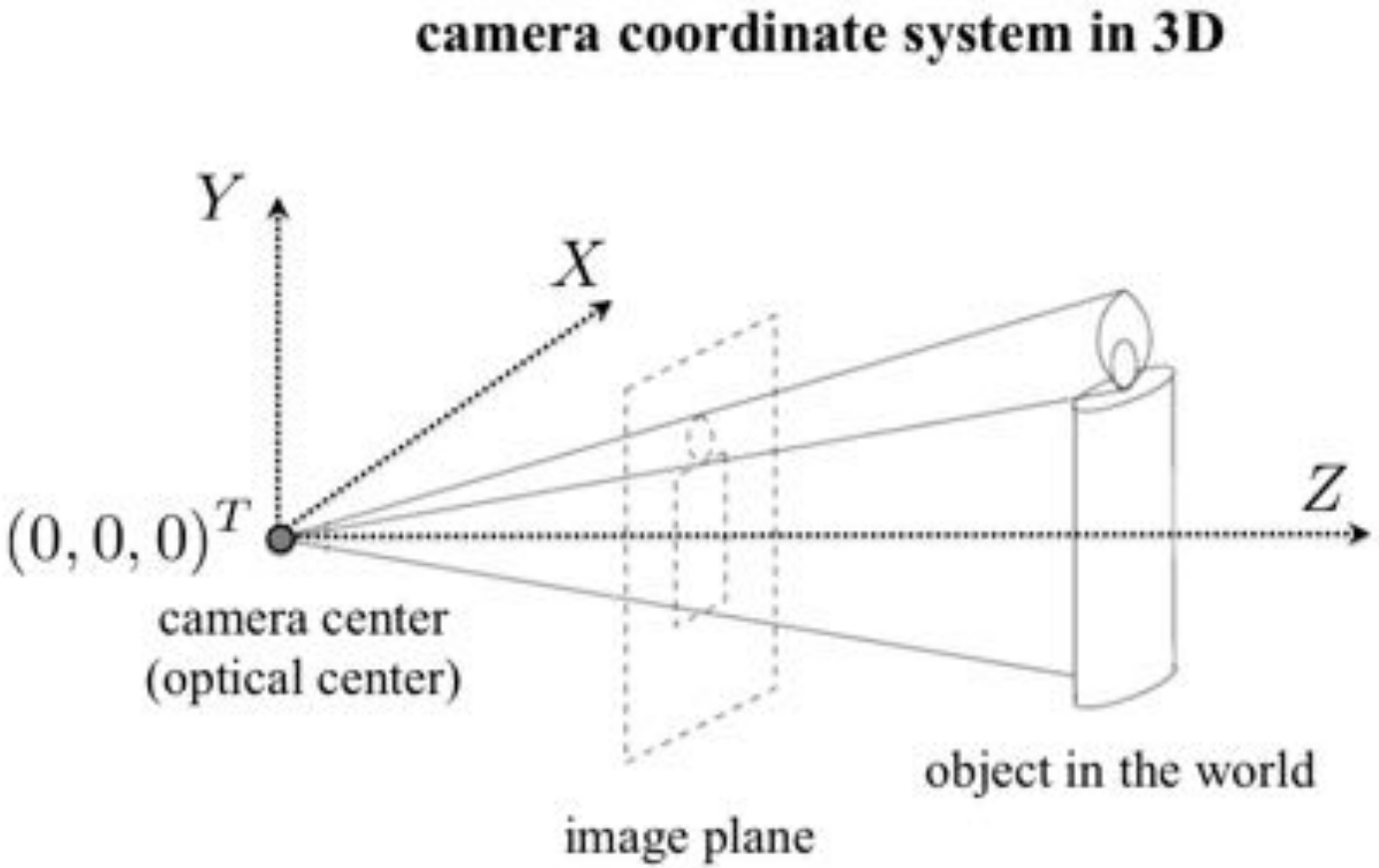
- Since it's easier to think in a non-upsidedown world, we will work with the virtual image plane, and just call it the image plane.
- How do points in 3D project to image plane? If I know a point in 3D, can I compute to which pixel it projects?

Modeling Projection



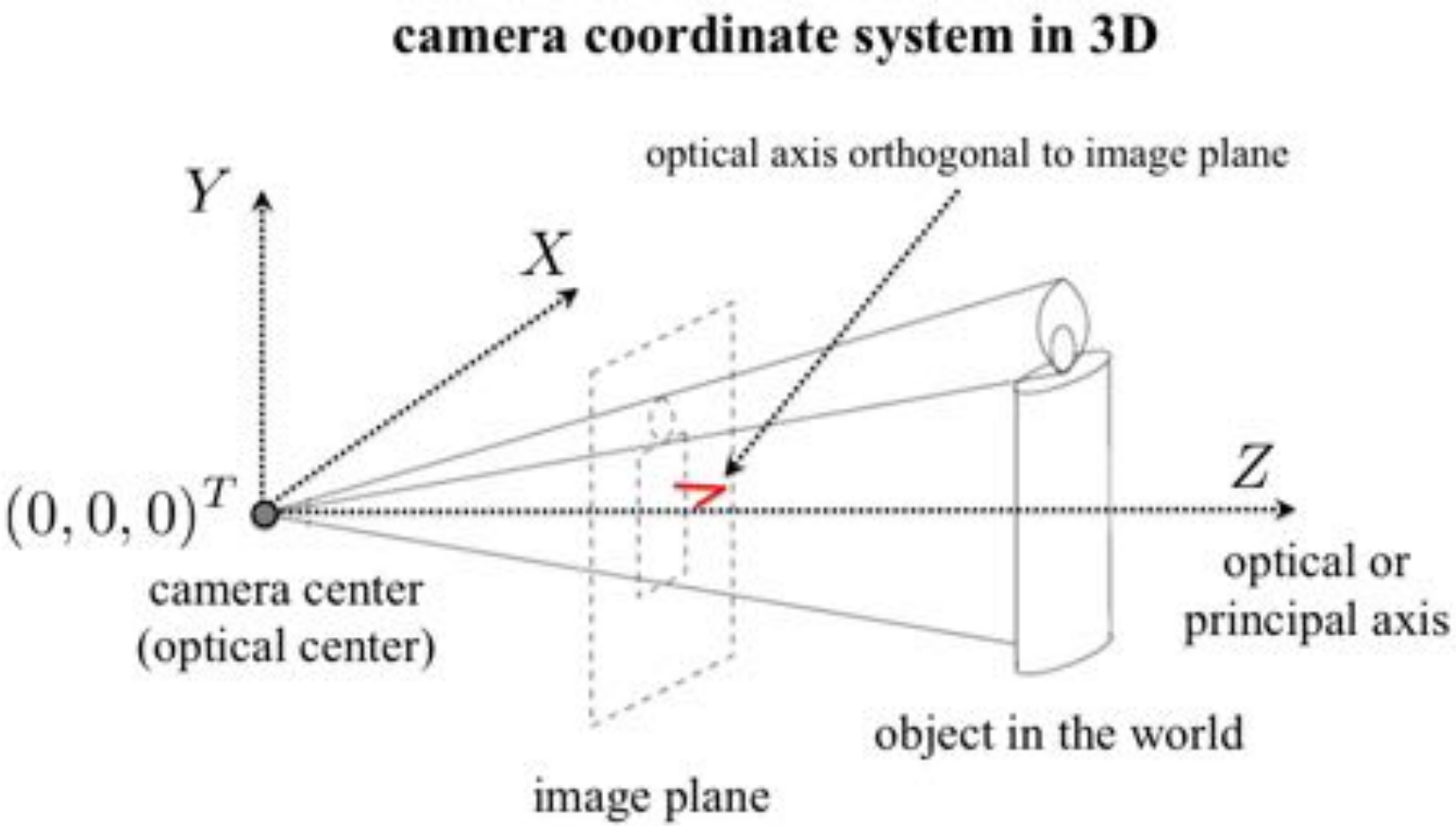
- First some notation which will help us derive the math
- To start with, we need a coordinate system

Modeling Projection



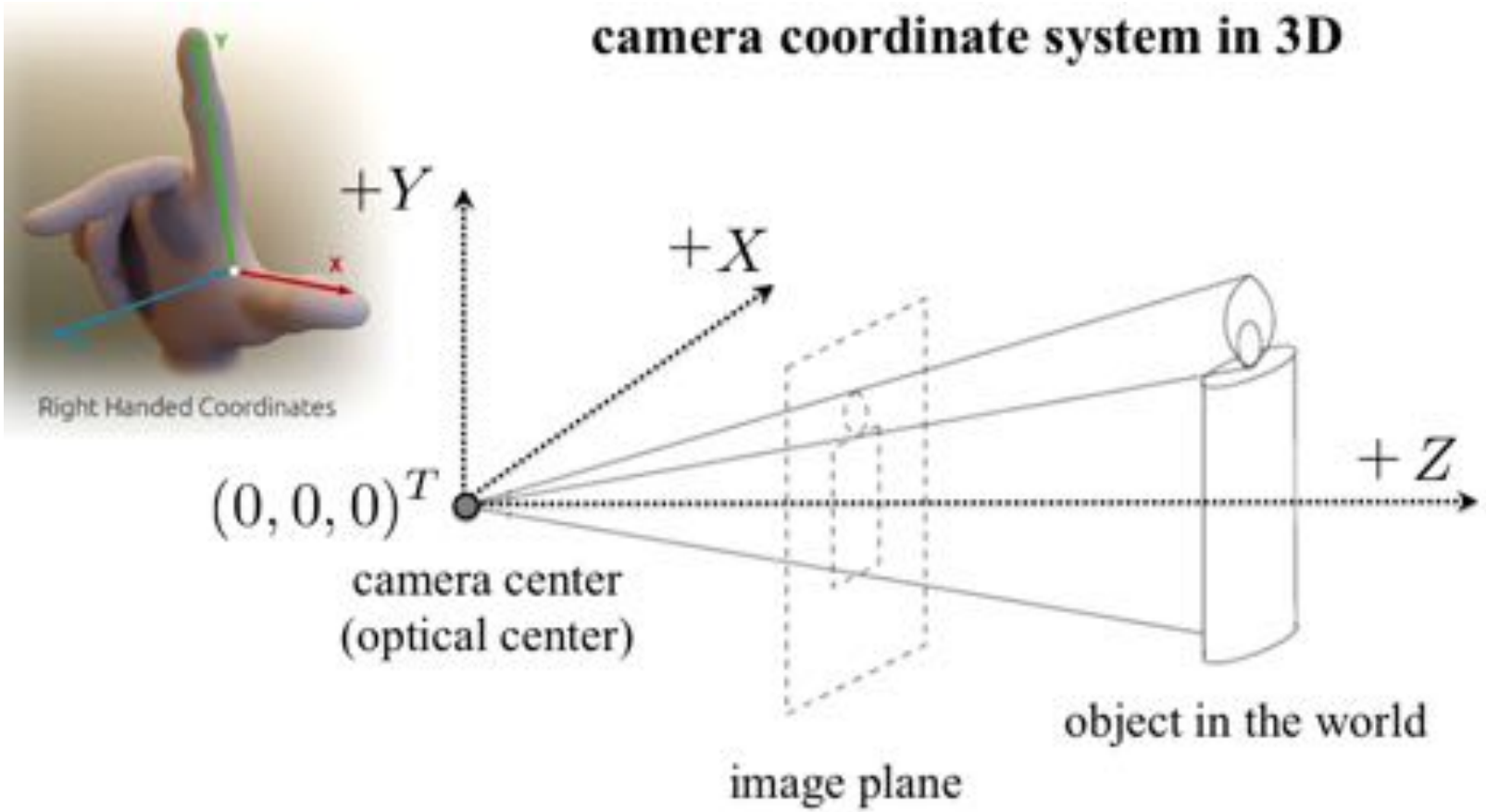
- We place a coordinate system relative to camera: **optical center** or **camera center C** is thus at origin $(0, 0, 0)$.

Modeling Projection



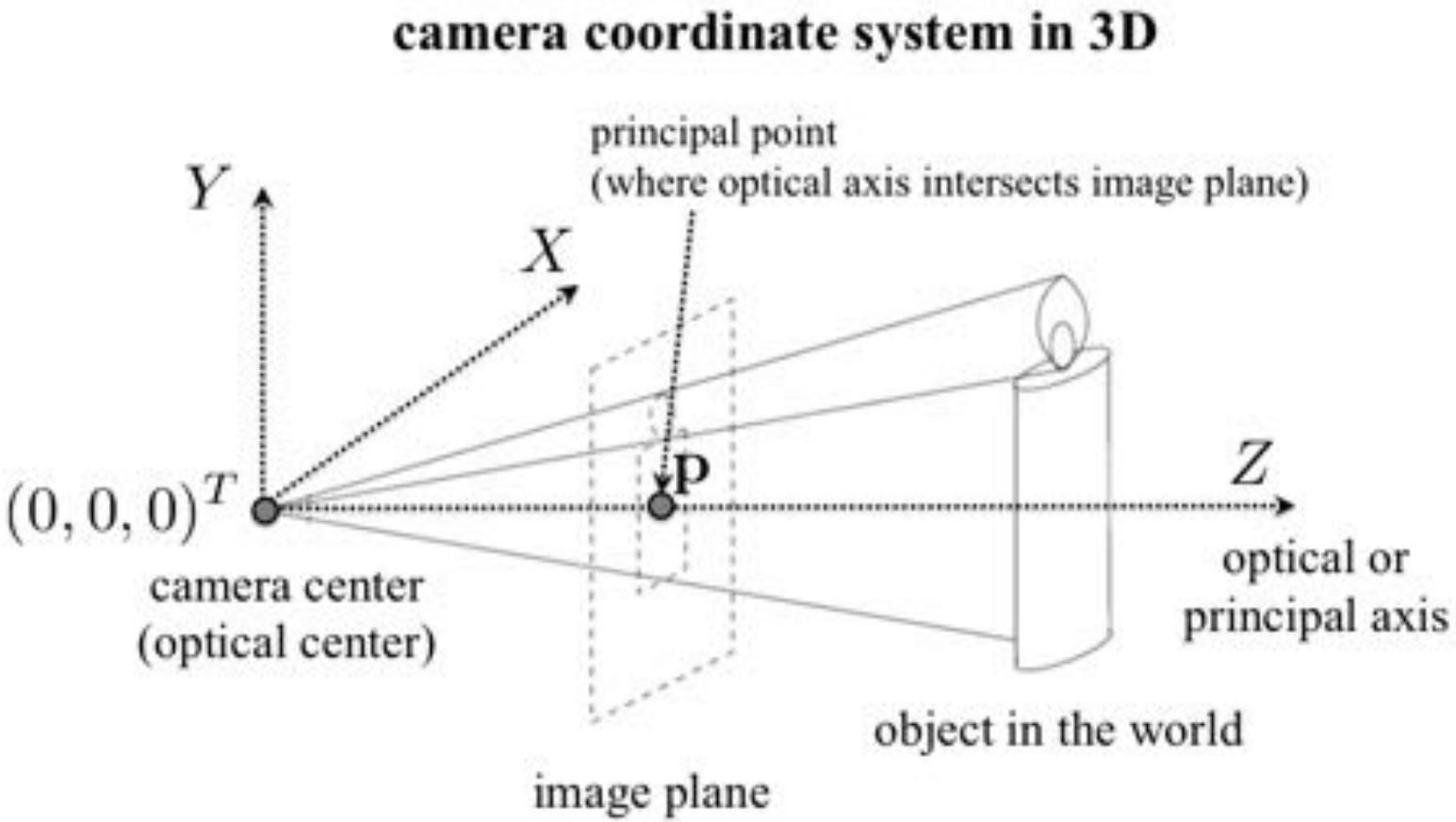
- The Z axis is called the **optical** or **principal axis**. It is orthogonal to the image plane. Axes X and Y are parallel to the image axes.

Modeling Projection



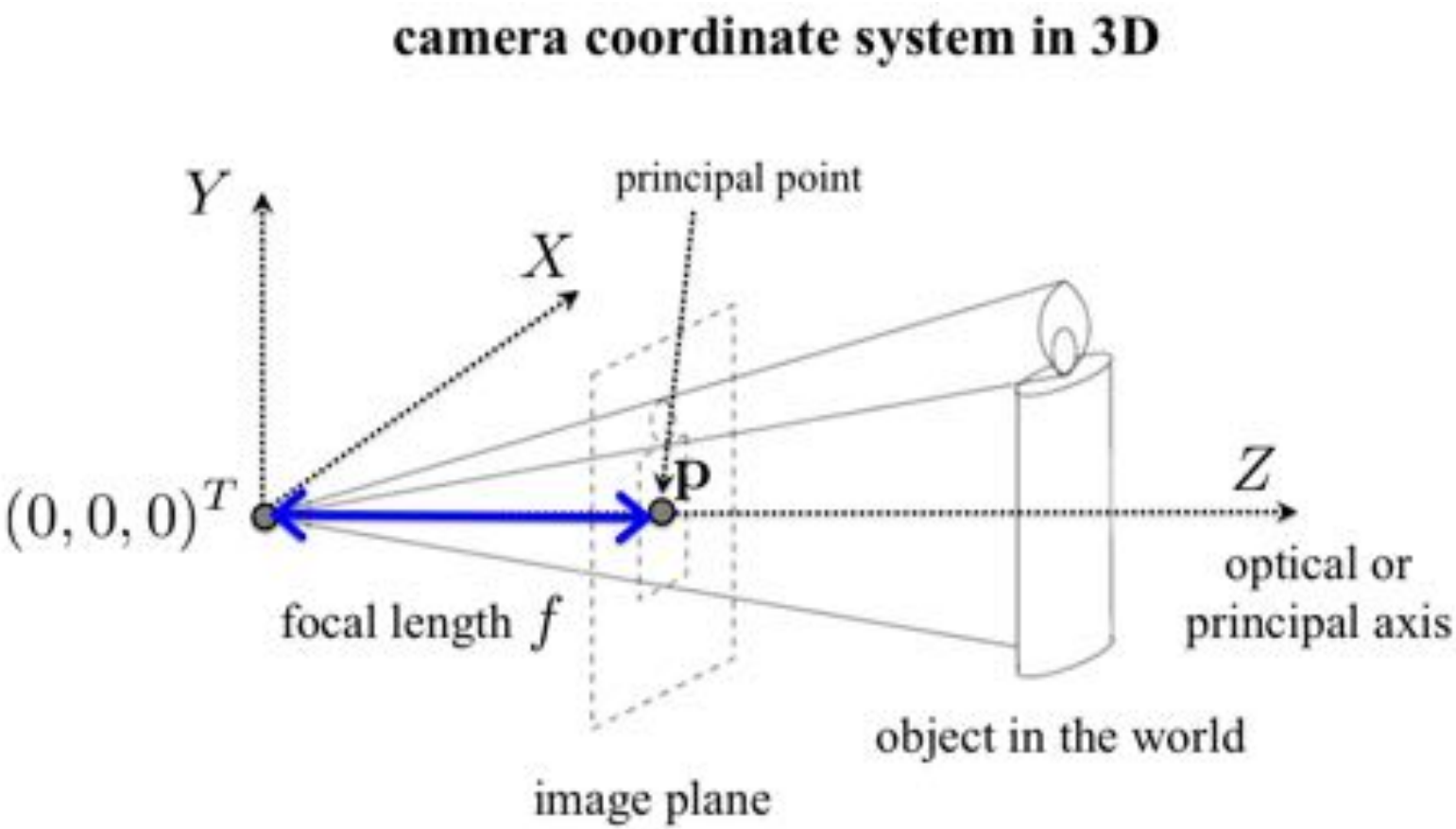
- We will use a **right handed** coordinate system

Modeling Projection



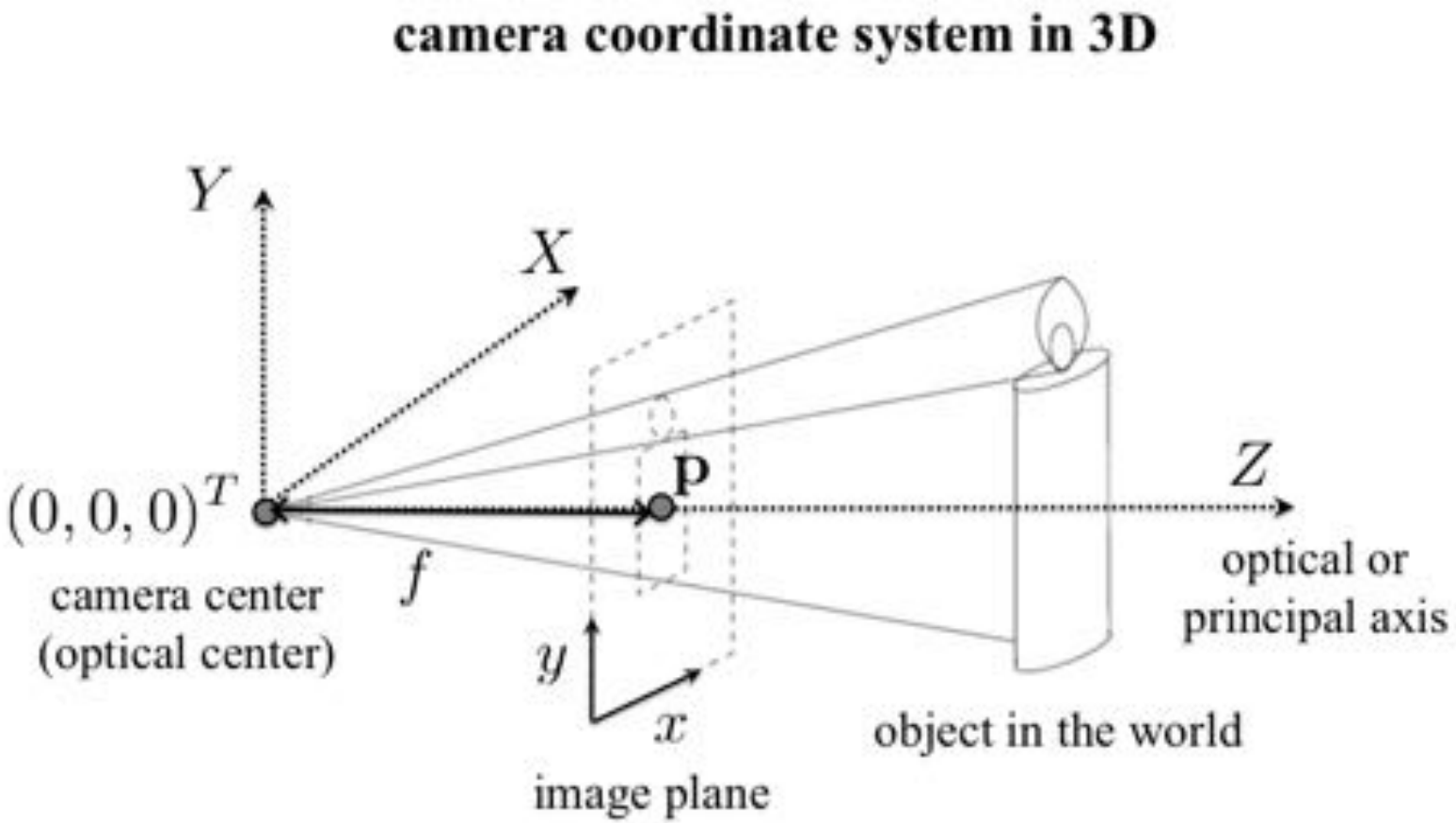
- The optical axis intersects the image plane in a point, **p**. We call this point a **principal point**. It's worth to remember the principal point since it will appear again later in the math.

Modeling Projection



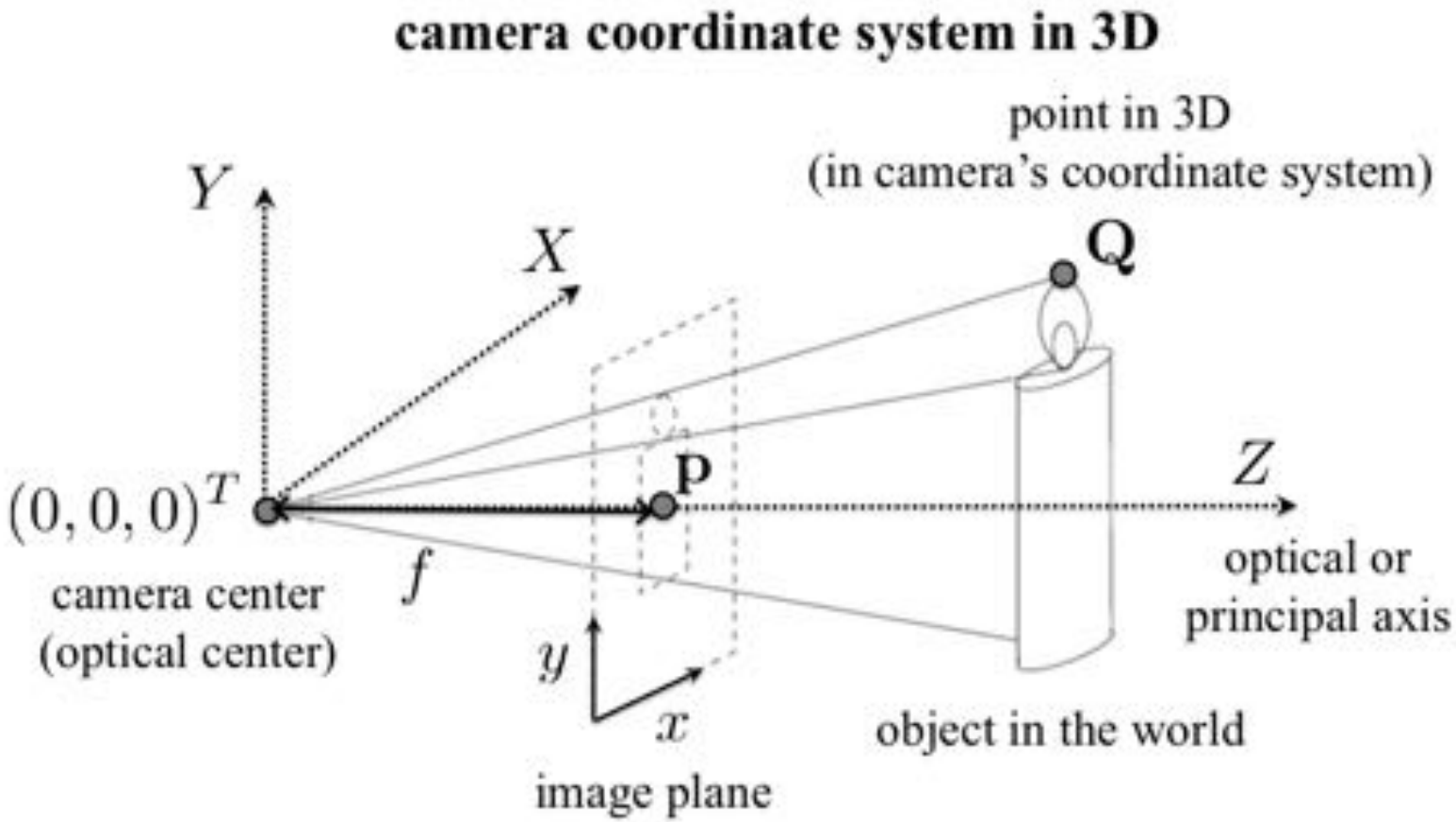
- The distance from the camera center to the principal point is called **focal length**, we will denote it with f. It's worth to remember the focal length since it will appear again later in the math.

Modeling Projection



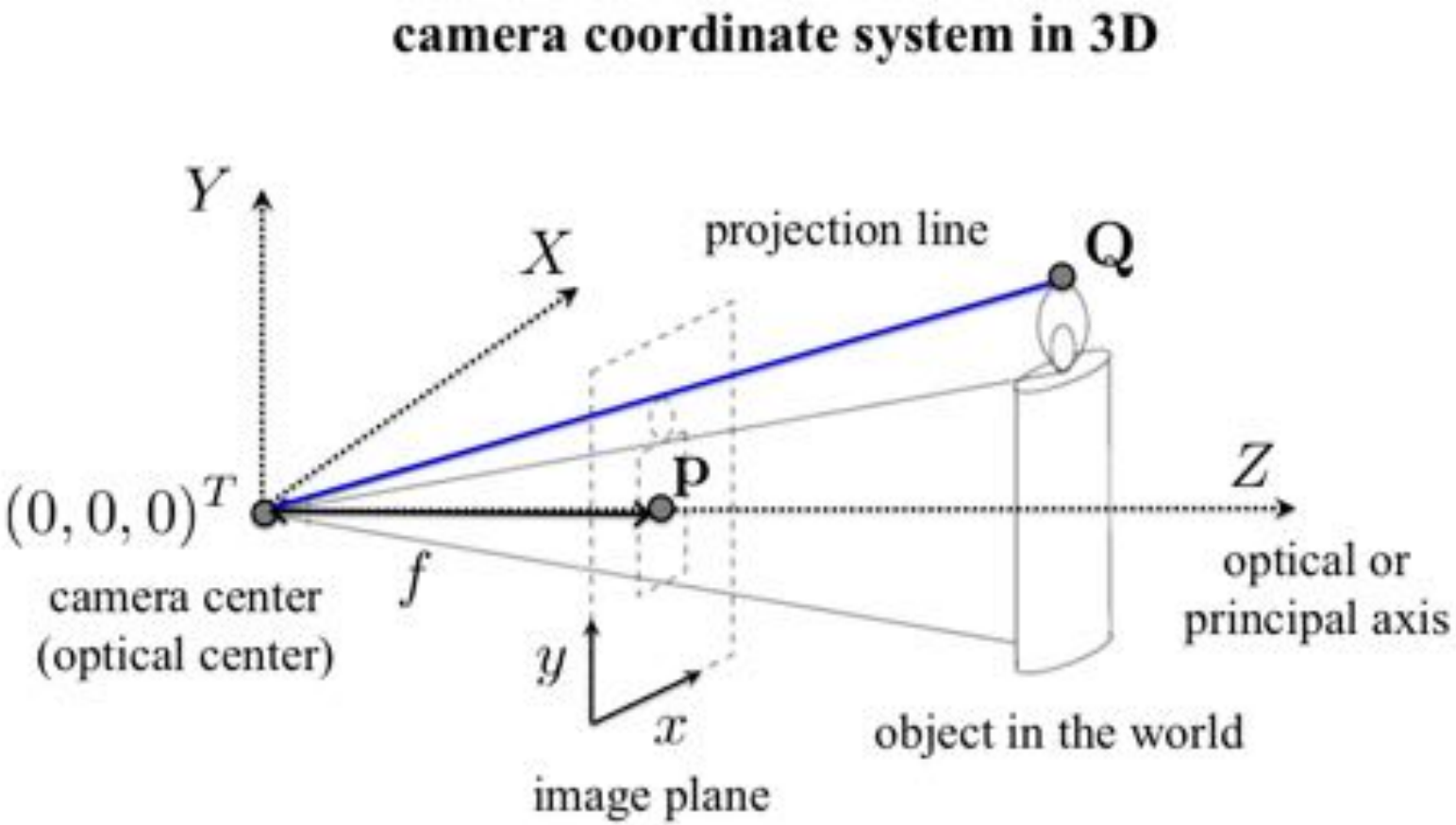
- We'll denote the image axes with x and y. An image we see is of course represented with these axes. We'll call this an **image coordinate system**.
- The tricky part is how to get from the camera's coordinate system (3D) to the image coordinate system (2D).

Modeling Projection



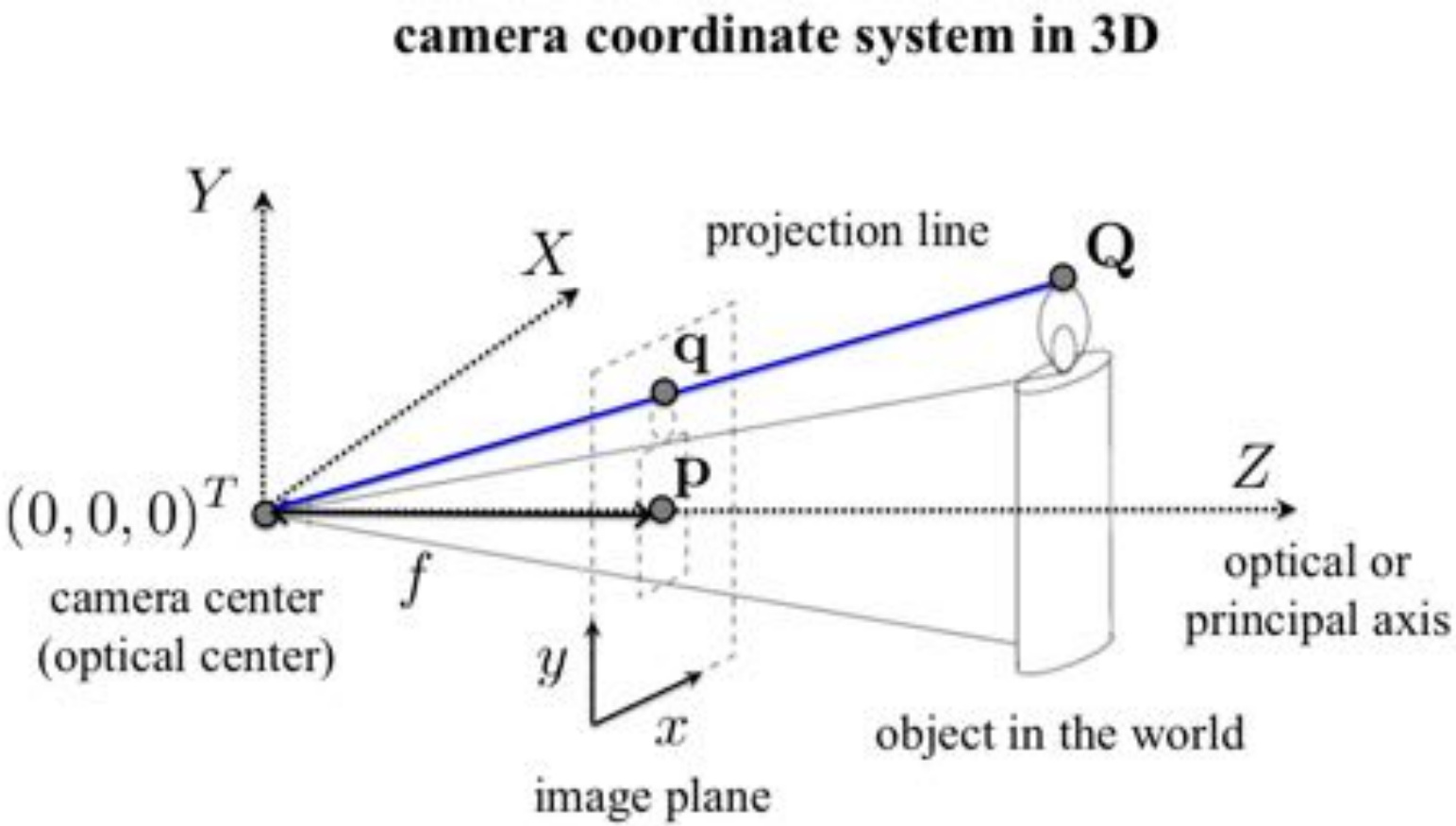
- Let's take some point **Q** in 3D. **Q** "lives" relative to the camera; its coordinates are assumed to be in camera's coordinate system.

Modeling Projection



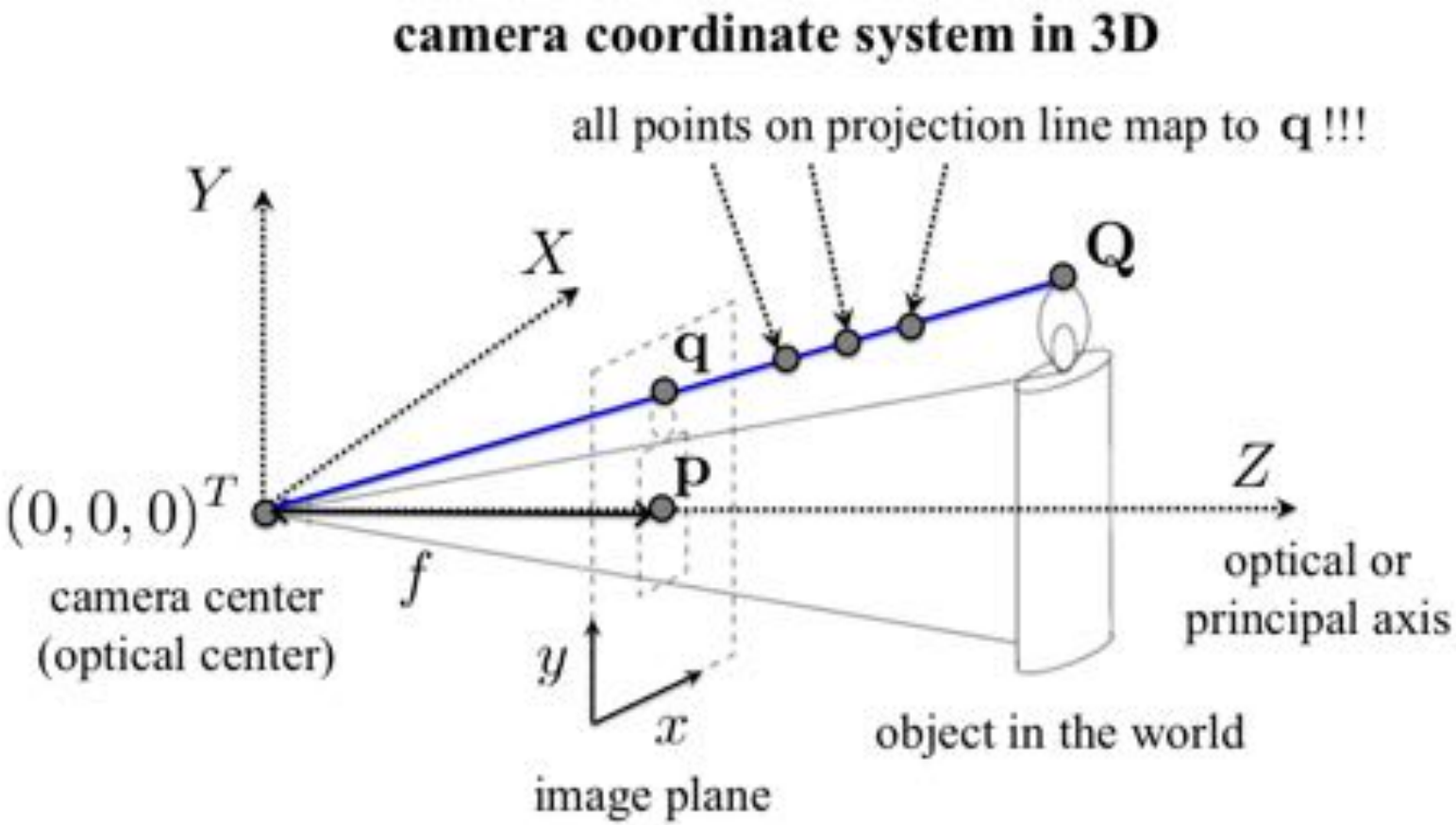
- We call the line from **Q** to camera center a **projection line**.

Modeling Projection



- The projection line intersects the image plane in a point **q**. This is the point we see in our image.

Modeling Projection



- First thing to notice is that all points from **Q**'s projection line project to the same point **q** in the image!
- **Ambiguity**: It's impossible to know how far a 3D point is from the camera along the projection line by looking only at the image (point **q**).

Modeling projection

From the movie Bone Collector

- **Ambiguity**: It's impossible to know how far a 3D point is from the camera along the projection line by looking only at the image (point **q**).
- It's impossible to know the real 3D size of objects just from an image
- Why did the detective put a dollar bill next to the footprint?
- How would you compute the shoe's dimensions?

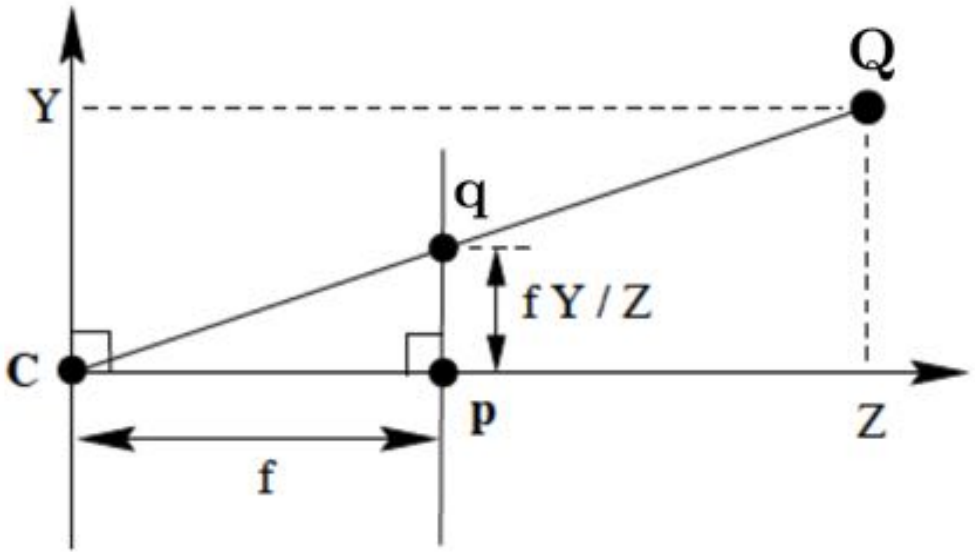
Modeling projection

From the movie Bone Collector

- **Ambiguity**: It's impossible to know how far a 3D point is from the camera along the projection line by looking only at the image (point **q**).
- It's impossible to know the real 3D size of objects just from an image
- Why did the detective put a dollar bill next to the footprint?
- How would you compute the shoe's dimensions?

Projection: Ready for Math

[Pic from: Zisserman & Hartley]



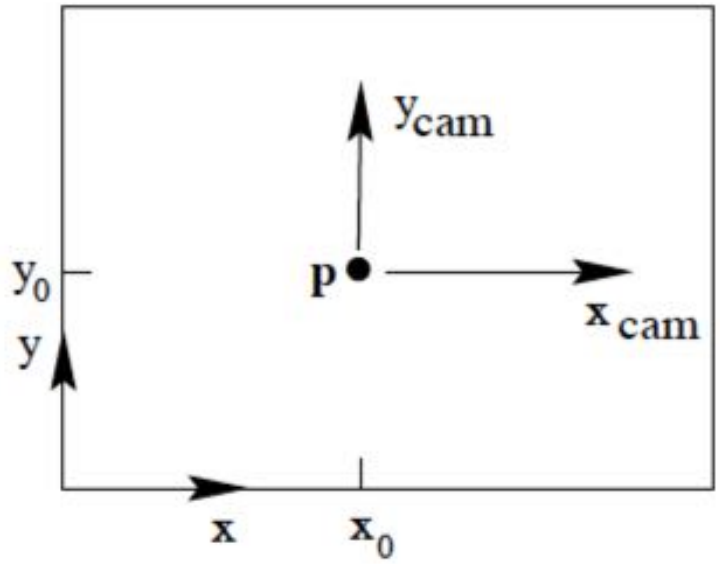
Projection Equations

- Using similar triangles:

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \left(\frac{f \cdot X}{Z}, \frac{f \cdot Y}{Z}, f\right)^T$$

Projection: Ready for Math

[Pic from: Zisserman & Hartley]



Projection Equations

- Using similar triangles:

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \left(\frac{f \cdot X}{Z}, \frac{f \cdot Y}{Z}, f\right)^T$$

- This is relative to principal point $\mathbf{p}$. To move the origin to $(0, 0)$ in image:

$$\mathbf{q} = (X, Y, Z)^T \quad \rightarrow \quad \left(\frac{f \cdot X}{Z} + p_x, \frac{f \cdot Y}{Z} + p_y, f\right)^T$$

where $\mathbf{p} = (p_x, p_y)$ is the principal point

Projection: Ready for Math

Projection Equations

- Using similar triangles:

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \left(\frac{f \cdot X}{Z}, \frac{f \cdot Y}{Z}, f\right)^T$$

- This is relative to principal point $\mathbf{p}$. To move the origin to $(0, 0)$ in image:

$$\mathbf{q} = (X, Y, Z)^T \quad \rightarrow \quad \left(\frac{f \cdot X}{Z} + p_x, \frac{f \cdot Y}{Z} + p_y, f\right)^T$$

where $\mathbf{p} = (p_x, p_y)$ is the principal point

Projection: Ready for Math

Projection Equations

- Using similar triangles:

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \left(\frac{f \cdot X}{Z}, \frac{f \cdot Y}{Z}, f\right)^T$$

- This is relative to principal point $\mathbf{p}$. To move the origin to $(0, 0)$ in image:

$$\mathbf{q} = (X, Y, Z)^T \quad \rightarrow \quad \left(\frac{f \cdot X}{Z} + p_x, \frac{f \cdot Y}{Z} + p_y, f\right)^T$$

  where $\mathbf{p} = (p_x, p_y)$ is the principal point

- Get the projection by throwing the last coordinate:

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \mathbf{q} = \left(\frac{f \cdot X}{Z} + p_x, \frac{f \cdot Y}{Z} + p_y\right)^T$$

- This is NOT a linear transformation as a division by Z is non-linear

Homogeneous Coordinates!

- We will use homogeneous coordinates, which simply append a 1 to the vector

Homogeneous coordinates to the rescue!

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates

$$(x, y, z) \Rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

homogeneous scene coordinates

Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \Rightarrow (x/w, y/w, z/w)$$

[Source: N. Snavely]

Homogeneous Coordinates!

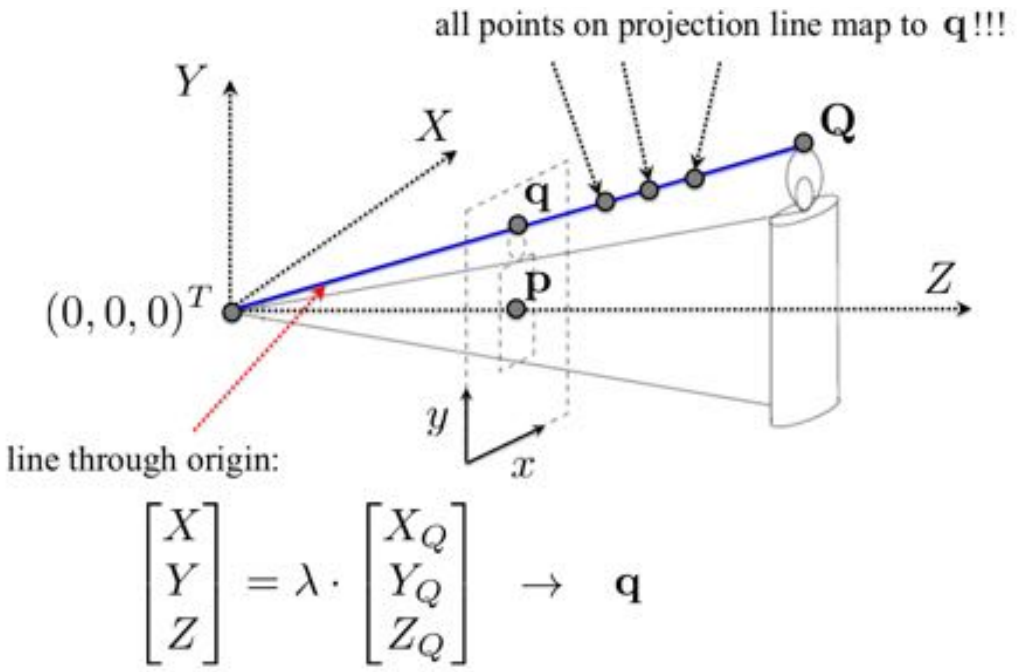
- We will use homogeneous coordinates, which simply append a 1 to the vector
- In homogeneous coordinates, scaling doesn't affect anything:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} w \cdot x \\ w \cdot y \\ w \end{bmatrix}$$

Homogeneous Coordinates!

- We will use homogeneous coordinates, which simply append a 1 to the vector
- In homogeneous coordinates, scaling doesn't affect anything:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} w \cdot x \\ w \cdot y \\ w \end{bmatrix}$$



- In Projective Geometry, all points are equal under scaling

Useful Trivia about Homogeneous Coordinates

- Homogeneous coordinates are quite useful in general. Let's see why
- Let's look at equation of a line in 2D: $ax + by + c = 0$

Useful Trivia about Homogeneous Coordinates

- Homogeneous coordinates are quite useful in general. Let's see why
- Let's look at equation of a line in 2D: $ax + by + c = 0$
- I can represent the line with a homogeneous vector $\mathbf{l} := (a, b, c)^T$ (why homogeneous?) and a homogeneous vector $\mathbf{x} := (x, y, 1)$. How?

Useful Trivia about Homogeneous Coordinates

- Homogeneous coordinates are quite useful in general. Let's see why
- Let's look at equation of a line in 2D: $ax + by + c = 0$
- I can represent the line with a homogeneous vector $\mathbf{l} := (a, b, c)^T$ (why homogeneous?) and a homogeneous vector $\mathbf{x} := (x, y, 1)$. How?
- Dot product is 0: $\mathbf{l}^T \cdot \mathbf{x} = 0$!

Useful Trivia about Homogeneous Coordinates

- Homogeneous coordinates are quite useful in general. Let's see why
- Let's look at equation of a line in 2D: $ax + by + c = 0$
- I can represent the line with a homogeneous vector $\mathbf{l} := (a, b, c)^T$ (why homogeneous?) and a homogeneous vector $\mathbf{x} := (x, y, 1)$. How?
- Dot product is 0: $\mathbf{l}^T \cdot \mathbf{x} = 0$!
- So if I have a line and someone gives me a point (x, y), how do I quickly check if the point lies on the line? Homogeneous coordinates, and check if dot product is 0.

Useful Trivia about Homogeneous Coordinates

- Homogeneous coordinates are quite useful in general. Let's see why
- Let's look at equation of a line in 2D: $ax + by + c = 0$
- I can represent the line with a homogeneous vector $\mathbf{l} := (a, b, c)^T$ (why homogeneous?) and a homogeneous vector $\mathbf{x} := (x, y, 1)$. How?
- Dot product is 0: $\mathbf{l}^T \cdot \mathbf{x} = 0$!
- So if I have a line and someone gives me a point (x, y), how do I quickly check if the point lies on the line? Homogeneous coordinates, and check if dot product is 0.
- Ok, what if I give you two points (x_1, y_1) and (x_2, y_2) and I ask you to write an equation for the line between them?

Useful Trivia about Homogeneous Coordinates

- Homogeneous coordinates are quite useful in general. Let's see why
- Let's look at equation of a line in 2D: $ax + by + c = 0$
- I can represent the line with a homogeneous vector $\mathbf{l} := (a, b, c)^T$ (why homogeneous?) and a homogeneous vector $\mathbf{x} := (x, y, 1)$. How?
- Dot product is 0: $\mathbf{l}^T \cdot \mathbf{x} = 0$!
- So if I have a line and someone gives me a point (x, y), how do I quickly check if the point lies on the line? Homogeneous coordinates, and check if dot product is 0.
- Ok, what if I give you two points (x_1, y_1) and (x_2, y_2) and I ask you to write an equation for the line between them?
- You can solve a linear system. But it's much easier in homogeneous coordinates: Since both points lie on a line, the homogeneous vectors are both "orthogonal" to the line vector $\mathbf{l}$ (dot product 0).

Useful Trivia about Homogeneous Coordinates

- Homogeneous coordinates are quite useful in general. Let's see why
- Let's look at equation of a line in 2D: $ax + by + c = 0$
- I can represent the line with a homogeneous vector $\mathbf{l} := (a, b, c)^T$ (why homogeneous?) and a homogeneous vector $\mathbf{x} := (x, y, 1)$. How?
- Dot product is 0: $\mathbf{l}^T \cdot \mathbf{x} = 0$!
- So if I have a line and someone gives me a point (x, y), how do I quickly check if the point lies on the line? Homogeneous coordinates, and check if dot product is 0.
- Ok, what if I give you two points (x_1, y_1) and (x_2, y_2) and I ask you to write an equation for the line between them?
- You can solve a linear system. But it's much easier in homogeneous coordinates: Since both points lie on a line, the homogeneous vectors are both "orthogonal" to the line vector $\mathbf{l}$ (dot product 0).
- We know that a vector orthogonal to two vectors is a cross product between them: $\mathbf{l} = (x_1, y_1, 1)^T \times (x_2, y_2, 1)^T$. And this is easy to compute.

Back to Perspective Projection

- We currently have this (the nasty division by Z):

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \mathbf{q} = \begin{bmatrix} \frac{f \cdot X}{Z} + p_x \\ \frac{f \cdot Y}{Z} + p_y \end{bmatrix}$$

Back to Perspective Projection

- We currently have this (the nasty division by Z):

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \mathbf{q} = \begin{bmatrix} \frac{f \cdot X}{Z} + p_x \\ \frac{f \cdot Y}{Z} + p_y \end{bmatrix}$$

- Write this with homogeneous coordinates:

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \mathbf{q} = \begin{bmatrix} \frac{f \cdot X}{Z} + p_x \\ \frac{f \cdot Y}{Z} + p_y \\ 1 \end{bmatrix} \quad \sim \quad \begin{bmatrix} f \cdot X + Z \cdot p_x \\ f \cdot Y + Z \cdot p_y \\ Z \end{bmatrix}$$

Back to Perspective Projection

- We currently have this (the nasty division by Z):

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \mathbf{q} = \begin{bmatrix} \frac{f \cdot X}{Z} + p_x \\ \frac{f \cdot Y}{Z} + p_y \end{bmatrix}$$

- Write this with homogeneous coordinates:

$$\mathbf{Q} = (X, Y, Z)^T \quad \rightarrow \quad \mathbf{q} = \begin{bmatrix} \frac{f \cdot X}{Z} + p_x \\ \frac{f \cdot Y}{Z} + p_y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f \cdot X + Z \cdot p_x \\ f \cdot Y + Z \cdot p_y \\ Z \end{bmatrix}$$

- We can now write this as matrix multiplication:

$$\mathbf{Q} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} f \cdot X + Z \cdot p_x \\ f \cdot Y + Z \cdot p_y \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Camera Intrinsics

- From previous slide:

$$\mathbf{Q} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} f \cdot X + Z \cdot p_x \\ f \cdot Y + Z \cdot p_y \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

- Write:

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

This is called a **camera calibration matrix** or **intrinsic parameter matrix**. The parameters in K are called **internal camera parameters**.

Camera Intrinsics

- From previous slide:

$$\mathbf{Q} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} f \cdot X + Z \cdot p_x \\ f \cdot Y + Z \cdot p_y \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

- Write:

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

  This is called a **camera calibration matrix** or **intrinsic parameter matrix**. The parameters in K are called **internal camera parameters**.

- Finally: $\begin{bmatrix} w \cdot x \\ w \cdot y \\ w \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \quad \rightarrow \quad \mathbf{q} = \begin{bmatrix} x \\ y \end{bmatrix}$

[Source: Zisserman & Hartley]

Camera Intrinsics

- Camera calibration matrix:

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Camera Intrinsics

- Camera calibration matrix:

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

- It can be a little more complicated. Pixels may not be square:

$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Camera Intrinsics

- Camera calibration matrix:

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

- It can be a little more complicated. Pixels may not be square:

$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

- And there might be a skew angle θ between x and y image axis:

$$K = \begin{bmatrix} f_x & -f_x \cot\theta & p_x \\ 0 & f_y / \sin\theta & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

[Source: Zisserman & Hartley]

Camera Intrinsics

- Camera calibration matrix:

$$K = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

We'll work with this one

- It can be a little more complicated. Pixels may not be square:

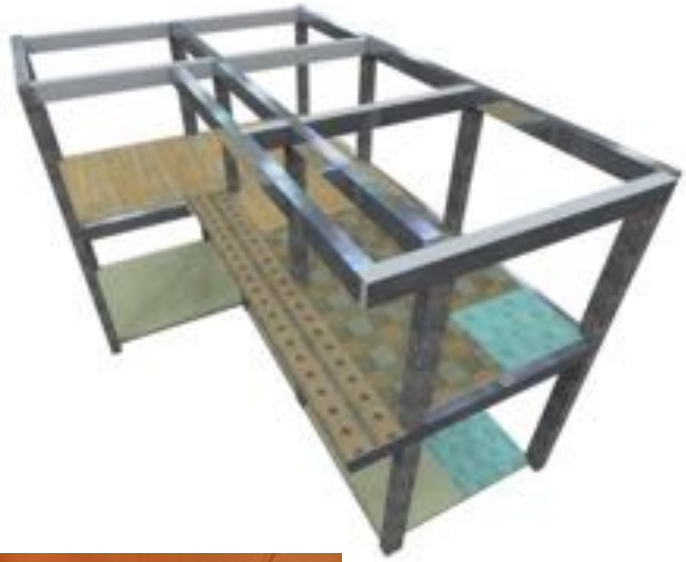
$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

- And there might be a skew angle θ between x and y image axis:

$$K = \begin{bmatrix} f_x & -f_x \cot\theta & p_x \\ 0 & f_y / \sin\theta & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

[Source: Zisserman & Hartley]

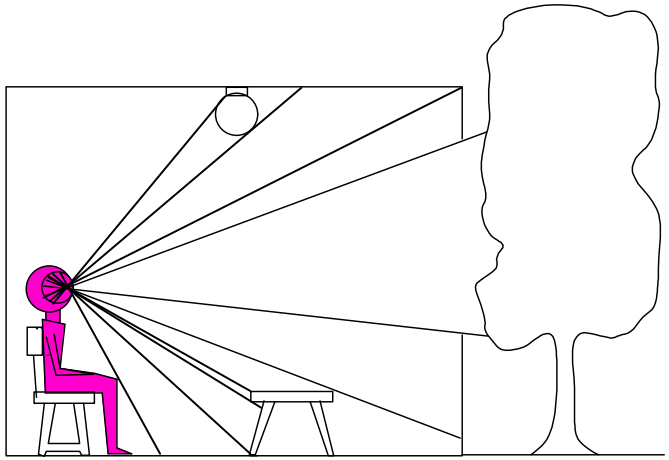
Perspective Projection



[Source: N. Snavely]

Dimensionality Reduction Machine (3D to 2D)

3D world



Point of observation

2D image

What have we lost?

- Angles
- Distances (lengths)

Slide by A. Efros

Figures © Stephen E. Palmer, 2002

Projection properties

- **Many-to-one**: any points along same ray map to same point in image

Projection properties

- **Many-to-one**: any points along same ray map to same point in image
- Points $\rightarrow$ points

Projection properties

- **Many-to-one**: any points along same ray map to same point in image
- Points $\rightarrow$ points
- Lines $\rightarrow$ lines. Why?

Projection properties

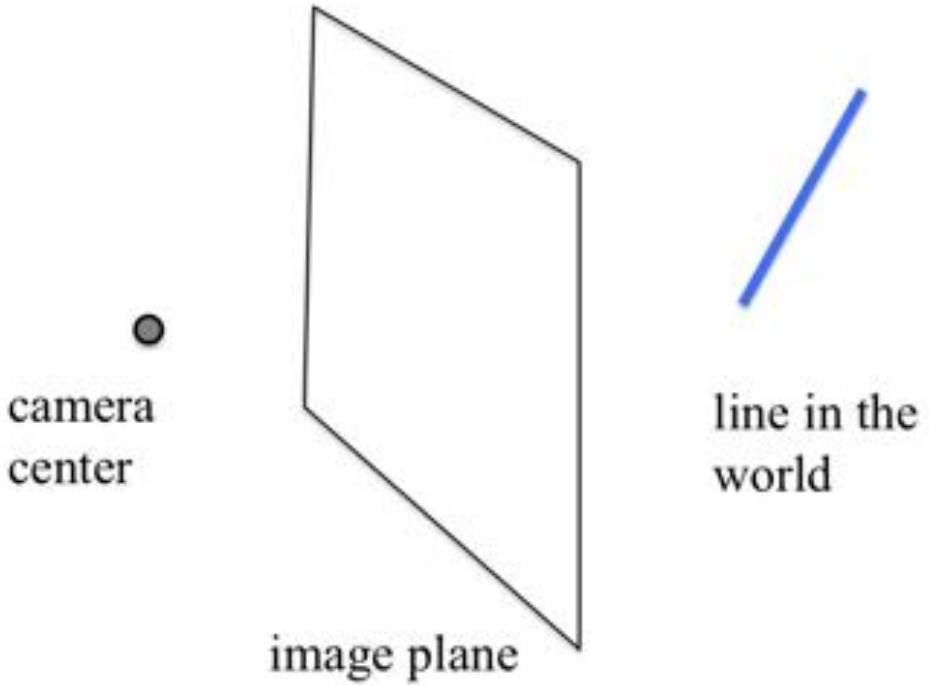


Figure: Proof by drawing

Projection properties

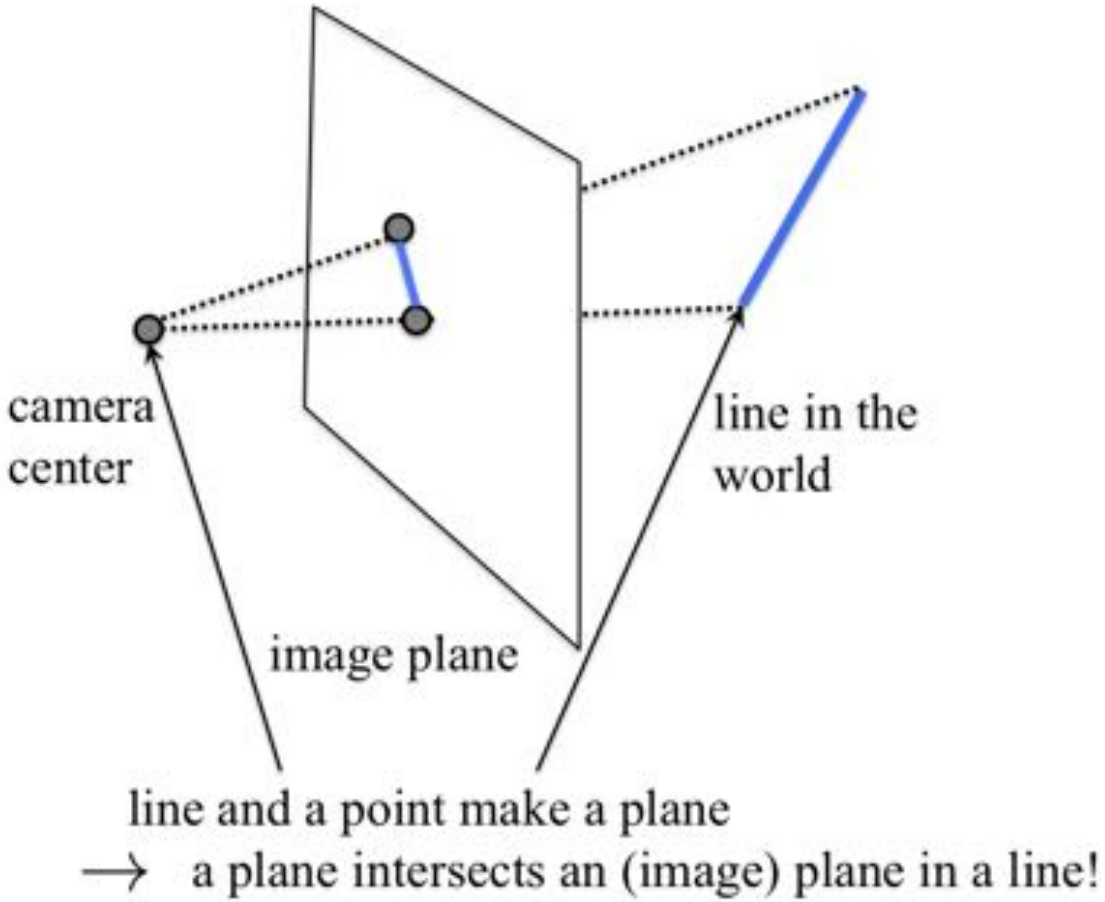


Figure: Proof by drawing

Projection properties

- **Many-to-one**: any points along same ray map to same point in image
- Points $\rightarrow$ points
- Lines $\rightarrow$ lines
- But line through principal point projects to a point. Why?

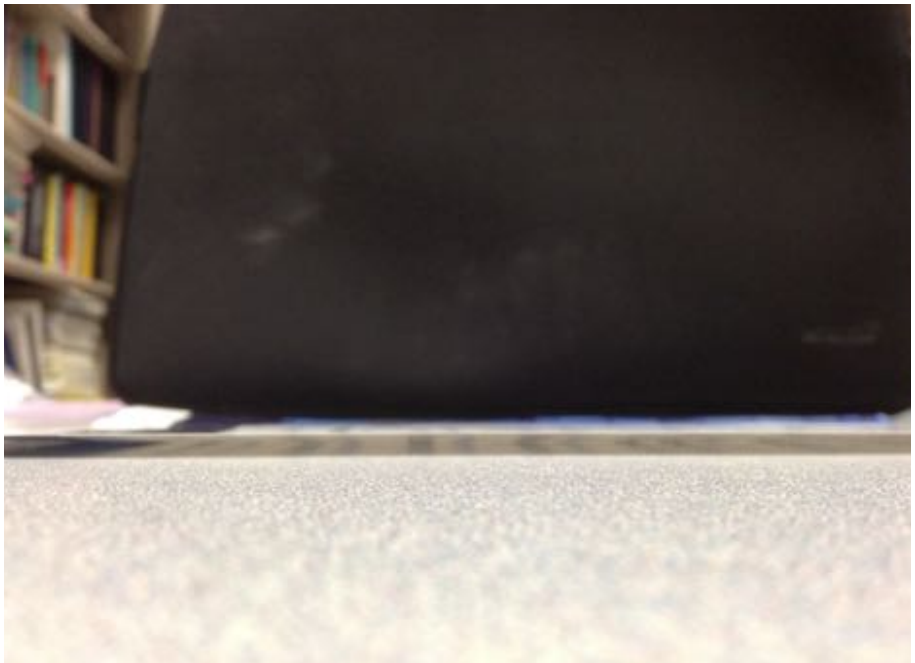
Figure: Can you tell where is the principal point?

Projection properties

- **Many-to-one**: any points along same ray map to same point in image
- Points $\rightarrow$ points
- Lines $\rightarrow$ lines
- But line through principal point projects to a point. Why?
- Planes $\rightarrow$ planes

Projection properties

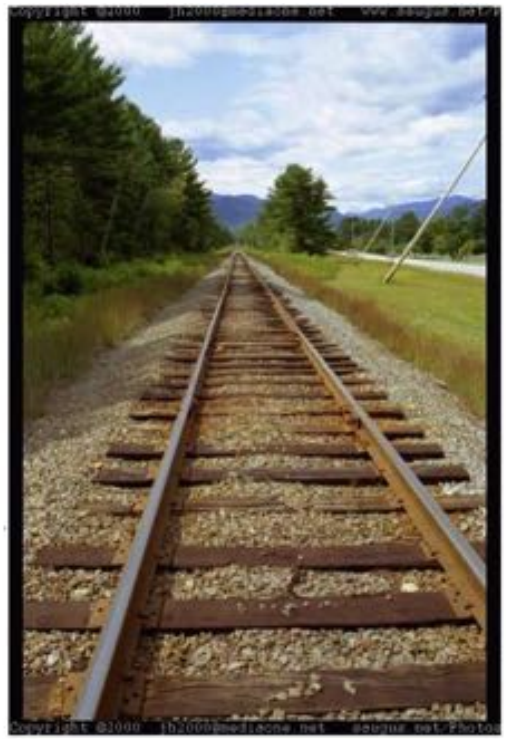
- **Many-to-one**: any points along same ray map to same point in image
- Points $\rightarrow$ points
- Lines $\rightarrow$ lines
- But line through principal point projects to a point. Why?
- Planes $\rightarrow$ planes
- But plane through principal point which is orthogonal to image plane projects to line. Why?

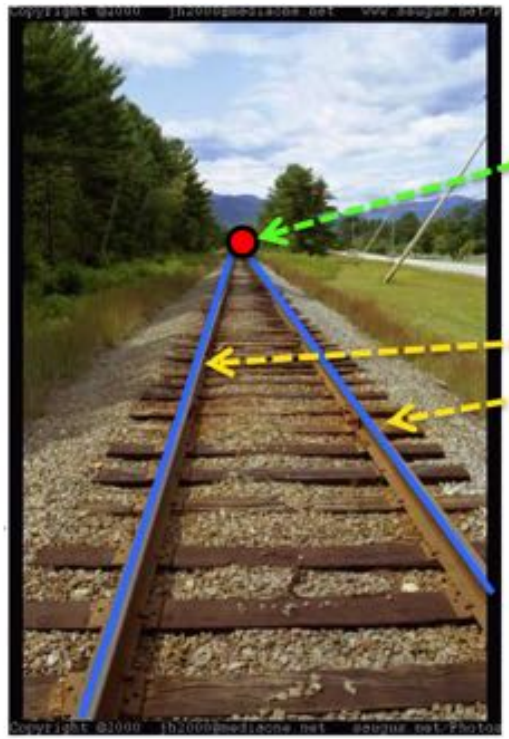


Projection Properties: Cool Facts

Parallel lines converge at a **vanishing point**

- Each different direction in the world **has its own vanishing point**





[Adopted from: N. Snavely, R. Urtasun]

Projection Properties: Cool Facts

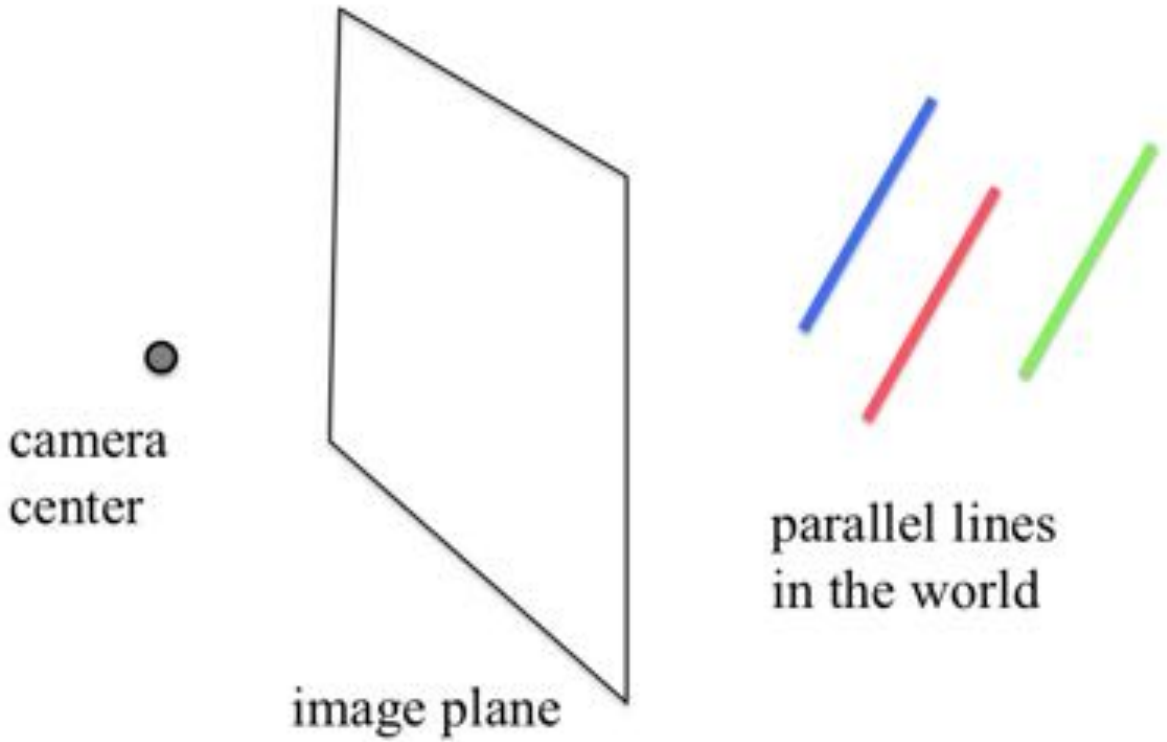
Parallel lines converge at a **vanishing point**

- Each different direction in the world **has its own vanishing point**
- All lines with the same 3D direction intersect at the **same vanishing point**

[Pic: R. Szeliski]

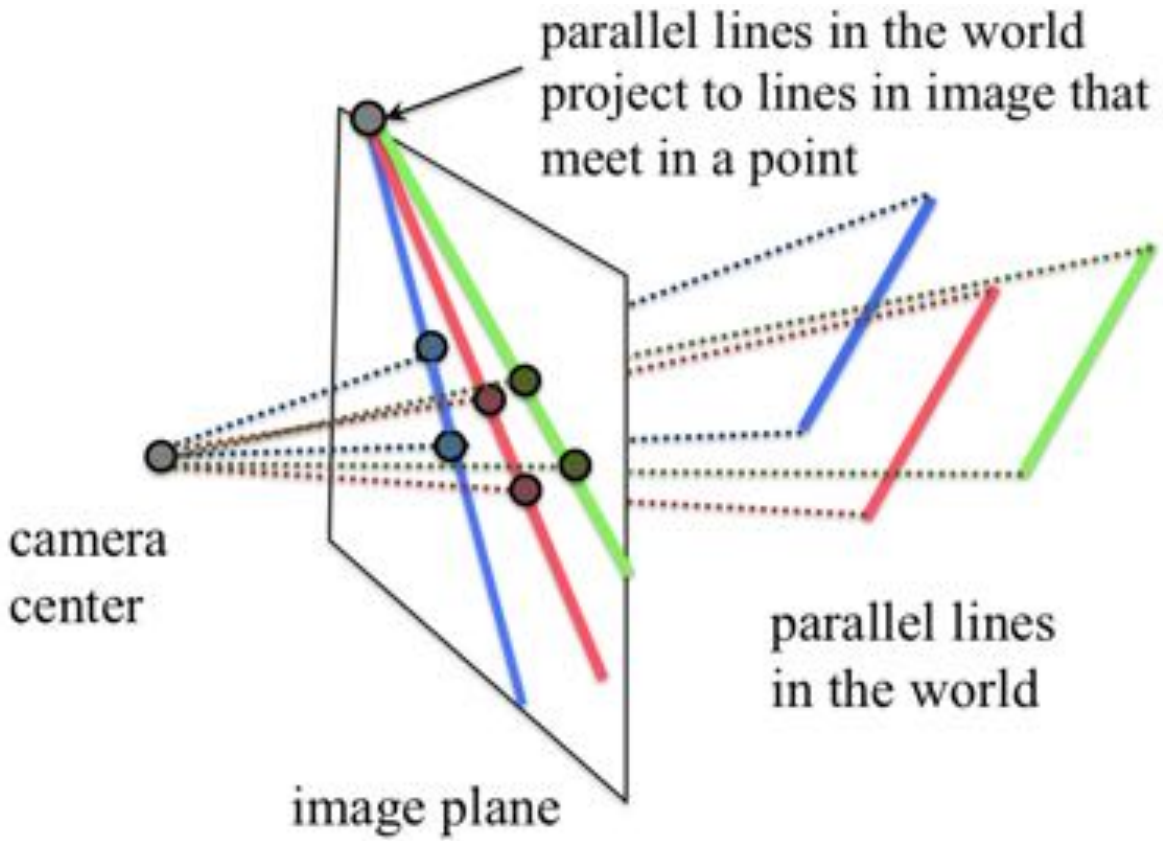
Projection Properties: Vanishing Point

- All lines with the same 3D direction intersect at the **same vanishing point**. **Why?**



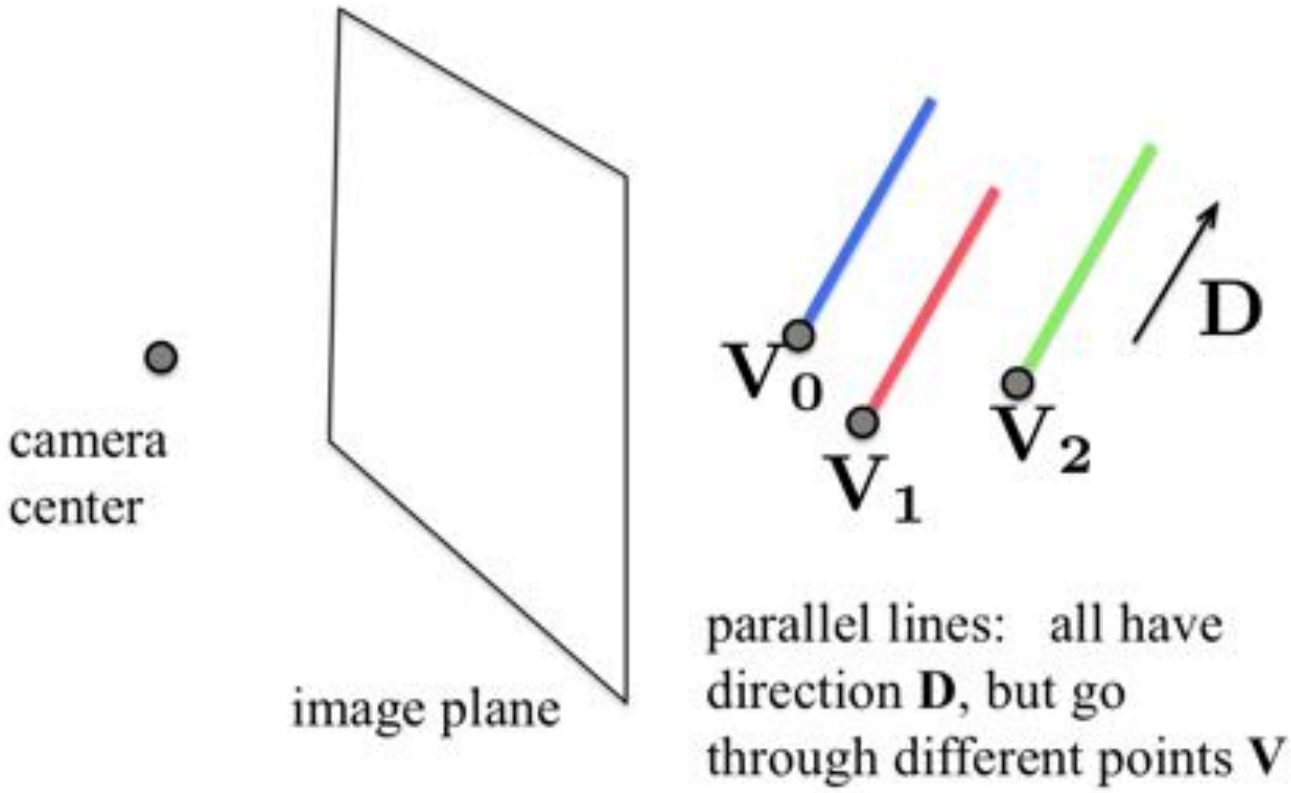
Projection Properties: Vanishing Point

- All lines with the same 3D direction intersect at the **same vanishing point**. **Why?**



Projection Properties: Vanishing Point

- All lines with the same 3D direction intersect at the **same vanishing point**. **Why?**



Projection Properties: Vanishing Point

- All lines with the same 3D direction intersect at the **same vanishing point**. **Why?**
- Line that passes through $\mathbf{V}$ with direction $\mathbf{D}$: $\mathbf{X} = \mathbf{V} + t\mathbf{D}$.

Projection Properties: Vanishing Point

- All lines with the same 3D direction intersect at the **same vanishing point**. **Why?**
- Line that passes through $\mathbf{V}$ with direction $\mathbf{D}$: $\mathbf{X} = \mathbf{V} + t\mathbf{D}$.
- Project it:

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = K\mathbf{X} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x + tD_x \\ V_y + tD_y \\ V_z + tD_z \end{bmatrix} = \begin{bmatrix} fV_x + ftD_x + p_x V_z + tp_x D_z \\ fV_y + ftD_y + p_y V_z + tp_y D_z \\ V_z + tD_z \end{bmatrix}$$

Projection Properties: Vanishing Point

- All lines with the same 3D direction intersect at the **same vanishing point**. **Why?**
- Line that passes through $\mathbf{V}$ with direction $\mathbf{D}$: $\mathbf{X} = \mathbf{V} + t\mathbf{D}$.
- Project it:

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = K\mathbf{X} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x + tD_x \\ V_y + tD_y \\ V_z + tD_z \end{bmatrix} = \begin{bmatrix} fV_x + ftD_x + p_x V_z + tp_x D_z \\ fV_y + ftD_y + p_y V_z + tp_y D_z \\ V_z + tD_z \end{bmatrix}$$

- Send $t \to \infty$ and compute x and y:

$$x = \lim_{t \to \infty} \frac{fV_x + ftD_x + p_x V_z + tp_x D_z}{V_z + tD_z} = \frac{fD_x + p_x D_z}{D_z}$$

$$y = \lim_{t \to \infty} \frac{fV_y + ftD_y + p_y V_z + tp_y D_z}{V_z + tD_z} = \frac{fD_y + p_y D_z}{D_z}$$

Projection Properties: Vanishing Point

- All lines with the same 3D direction intersect at the **same vanishing point**. **Why?**
- Line that passes through $\mathbf{V}$ with direction $\mathbf{D}$: $\mathbf{X} = \mathbf{V} + t\mathbf{D}$.
- Project it:

$$\begin{bmatrix} wx \\ wy \\ w \end{bmatrix} = K\mathbf{X} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x + tD_x \\ V_y + tD_y \\ V_z + tD_z \end{bmatrix} = \begin{bmatrix} fV_x + ftD_x + p_xV_z + tp_xD_z \\ fV_y + ftD_y + p_yV_z + tp_yD_z \\ V_z + tD_z \end{bmatrix}$$

- Send $t \to \infty$ and compute x and y:

$$x = \lim_{t\to\infty} \frac{fV_x + ftD_x + p_xV_z + tp_xD_z}{V_z + tD_z} = \frac{fD_x + p_xD_z}{D_z}$$

$$y = \lim_{t\to\infty} \frac{fV_y + ftD_y + p_yV_z + tp_yD_z}{V_z + tD_z} = \frac{fD_y + p_yD_z}{D_z}$$

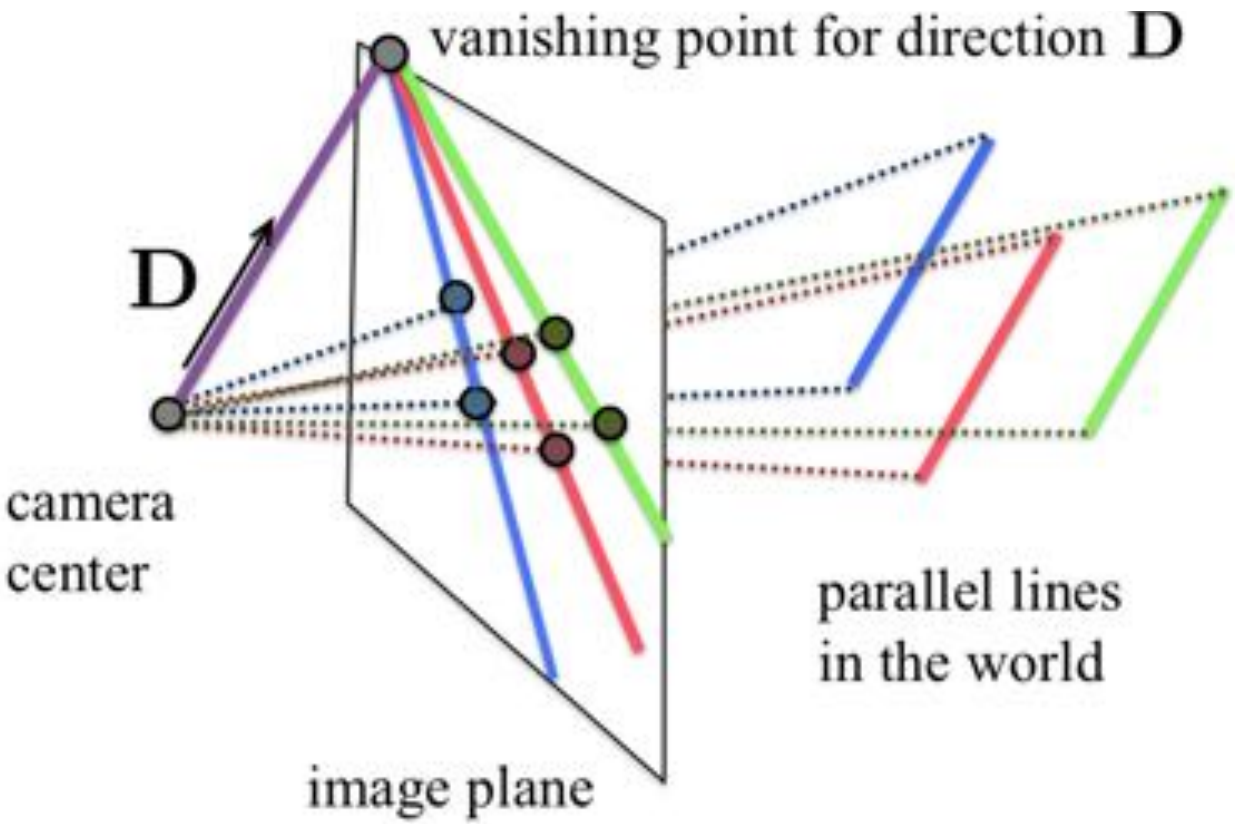
- This doesn't depend on $\mathbf{V}$! So all lines with direction $\mathbf{D}$ go to this point!

Projection Properties: Vanishing Point

- All lines with the same 3D direction intersect at the **same vanishing point**.

Projection Properties: Vanishing Point

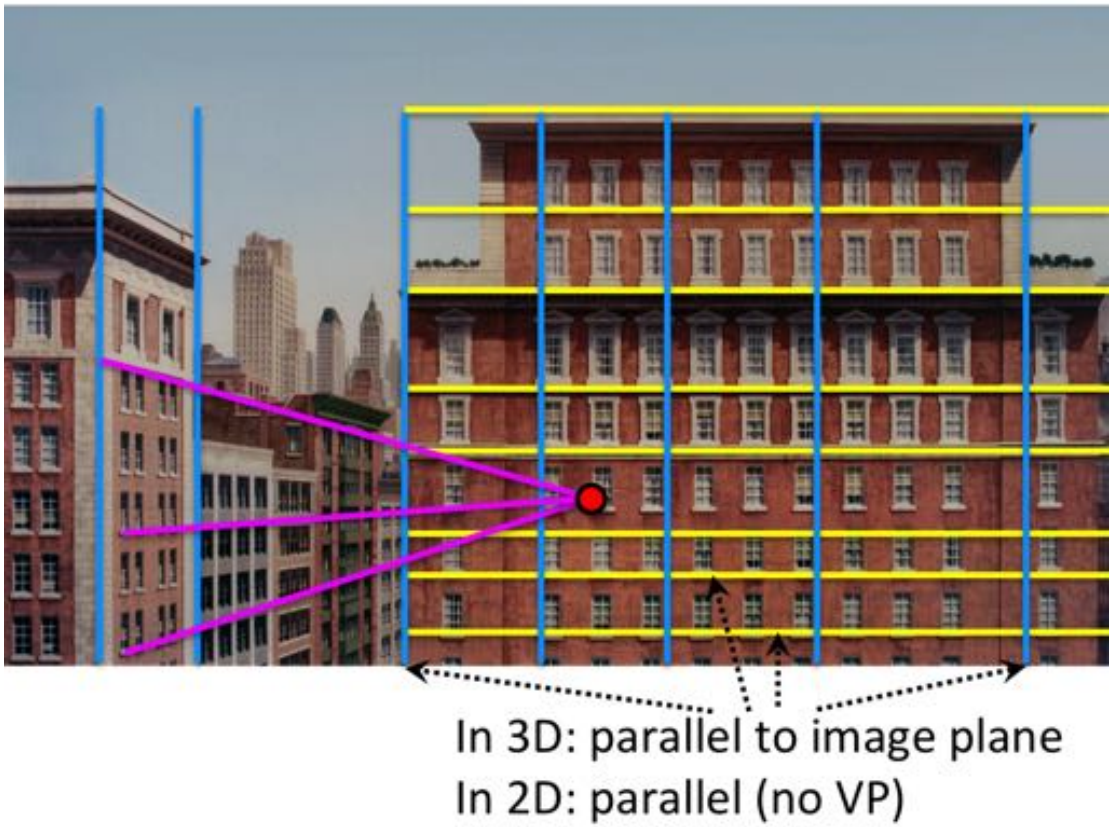
- All lines with the same 3D direction intersect at the **same vanishing point**.
- The easiest way to find this point: Translate line with direction **D** to the camera center. This line intersects the image plane in the vanishing point corresponding to direction **D**! Why?



Projection Properties: Cool Facts

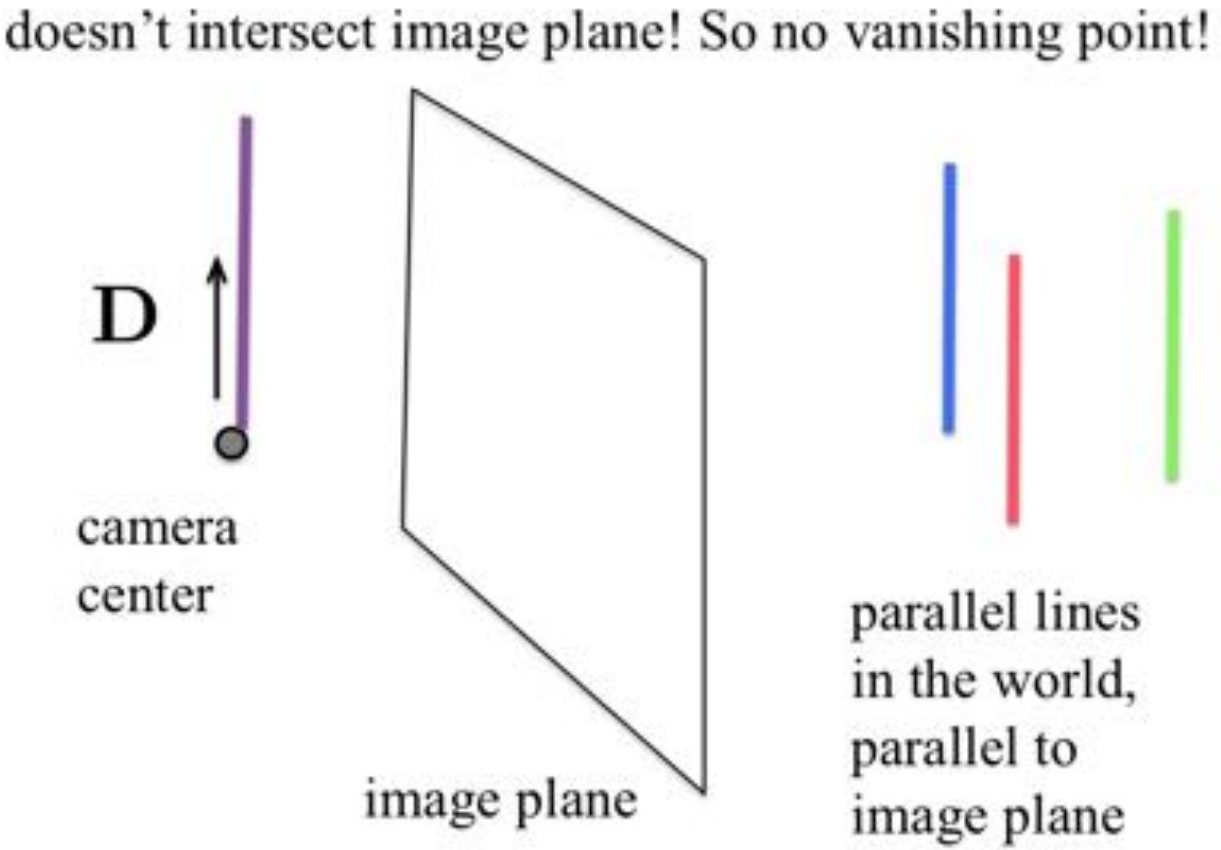
Parallel lines converge at a **vanishing point**

- Each different direction in the world **has its own vanishing point**
- Lines parallel to image plane are also parallel in the image (we say that they intersect at infinity). Why?



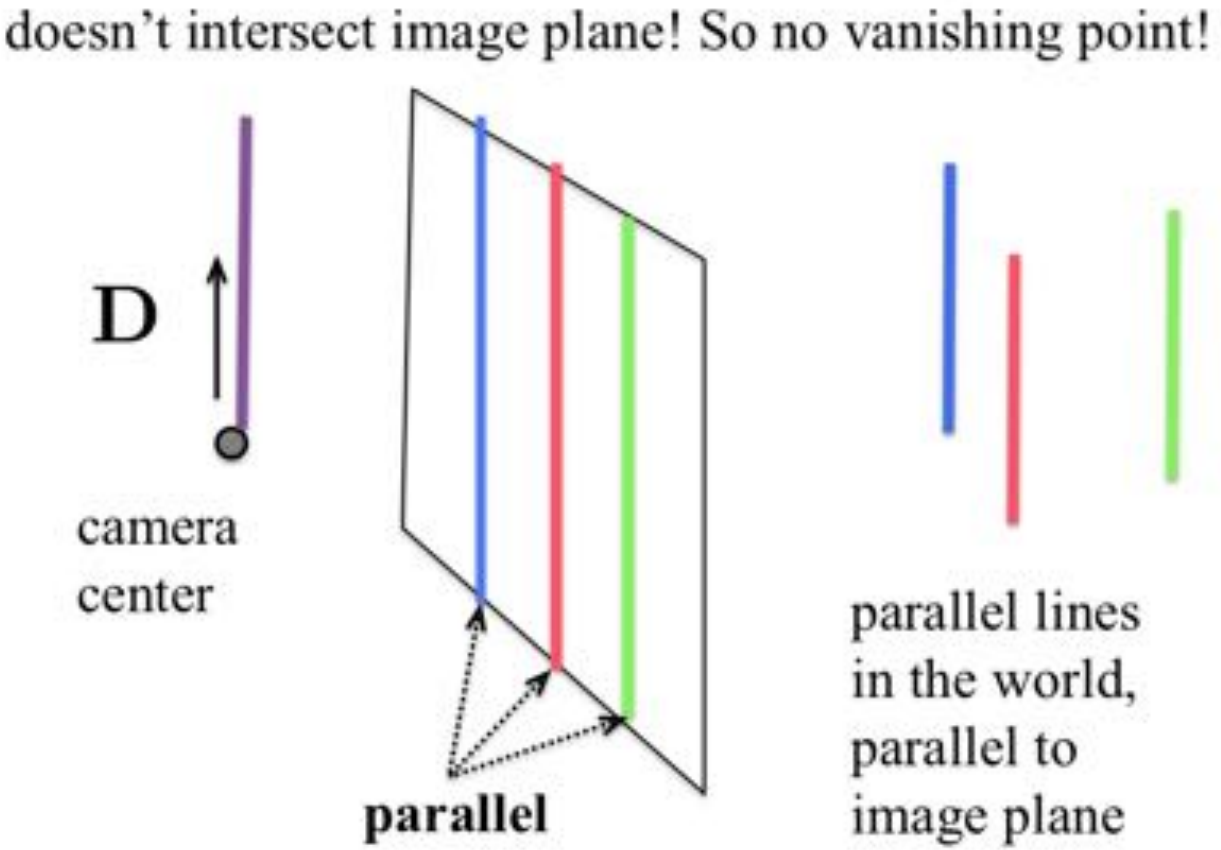
Projection Properties: Cool Facts

- Lines parallel to image plane are also parallel in the image. We say that they intersect at infinity.



Projection Properties: Cool Facts

- Lines parallel to image plane are also parallel in the image. We say that they intersect at infinity.



Projection Properties: Cool Tricks

- This picture has been recorded from a car with a camera on top. We know the camera intrinsic matrix K.
- Can we tell the incline of the hill we are driving on?
- How?

Projection Properties: Cool Tricks

- Can we tell the incline of the hill we are driving on?

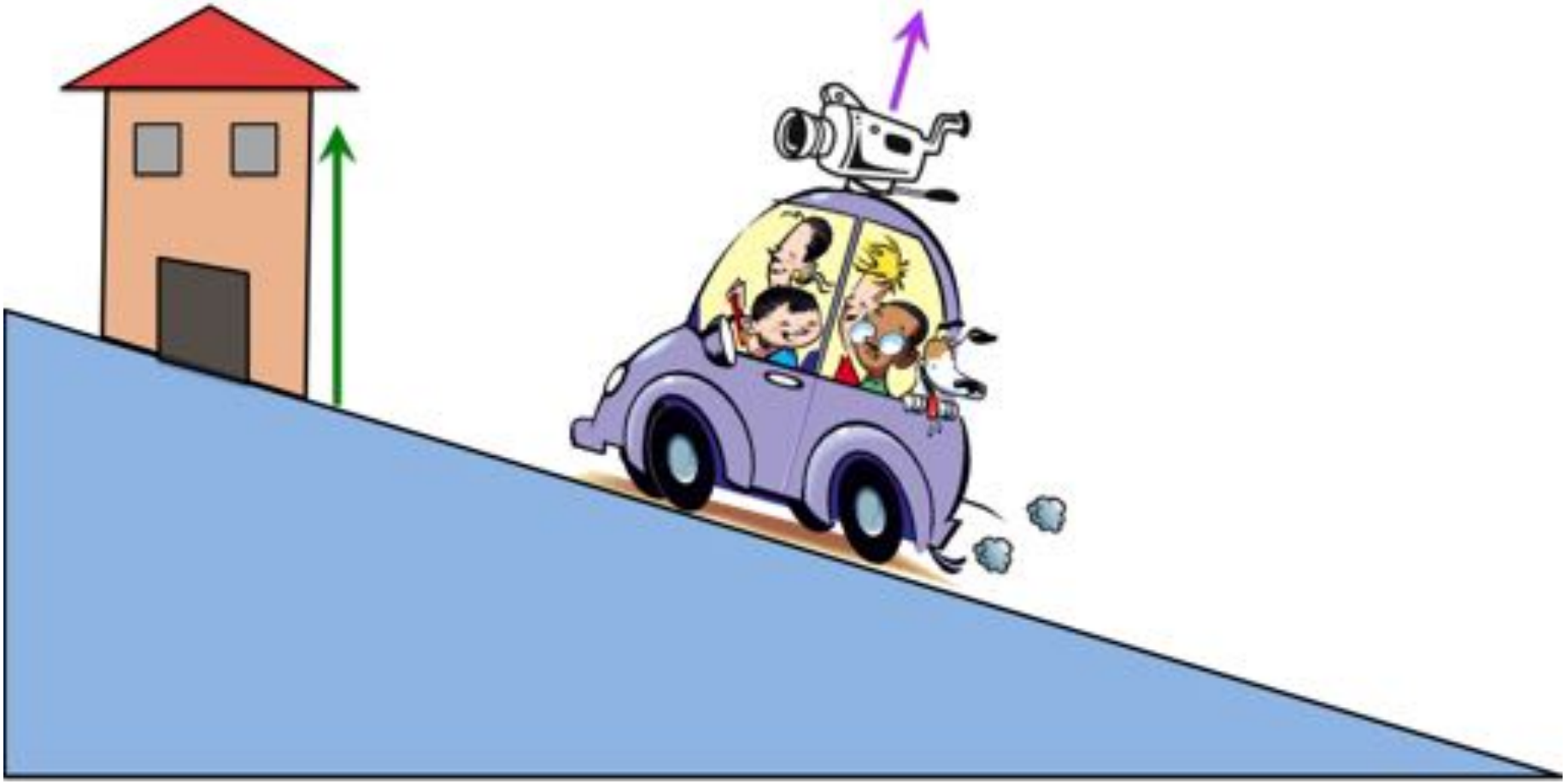


Figure: This is the 3D world behind the picture.

Projection Properties: Cool Tricks

- Can we tell the incline of the hill we are driving on?

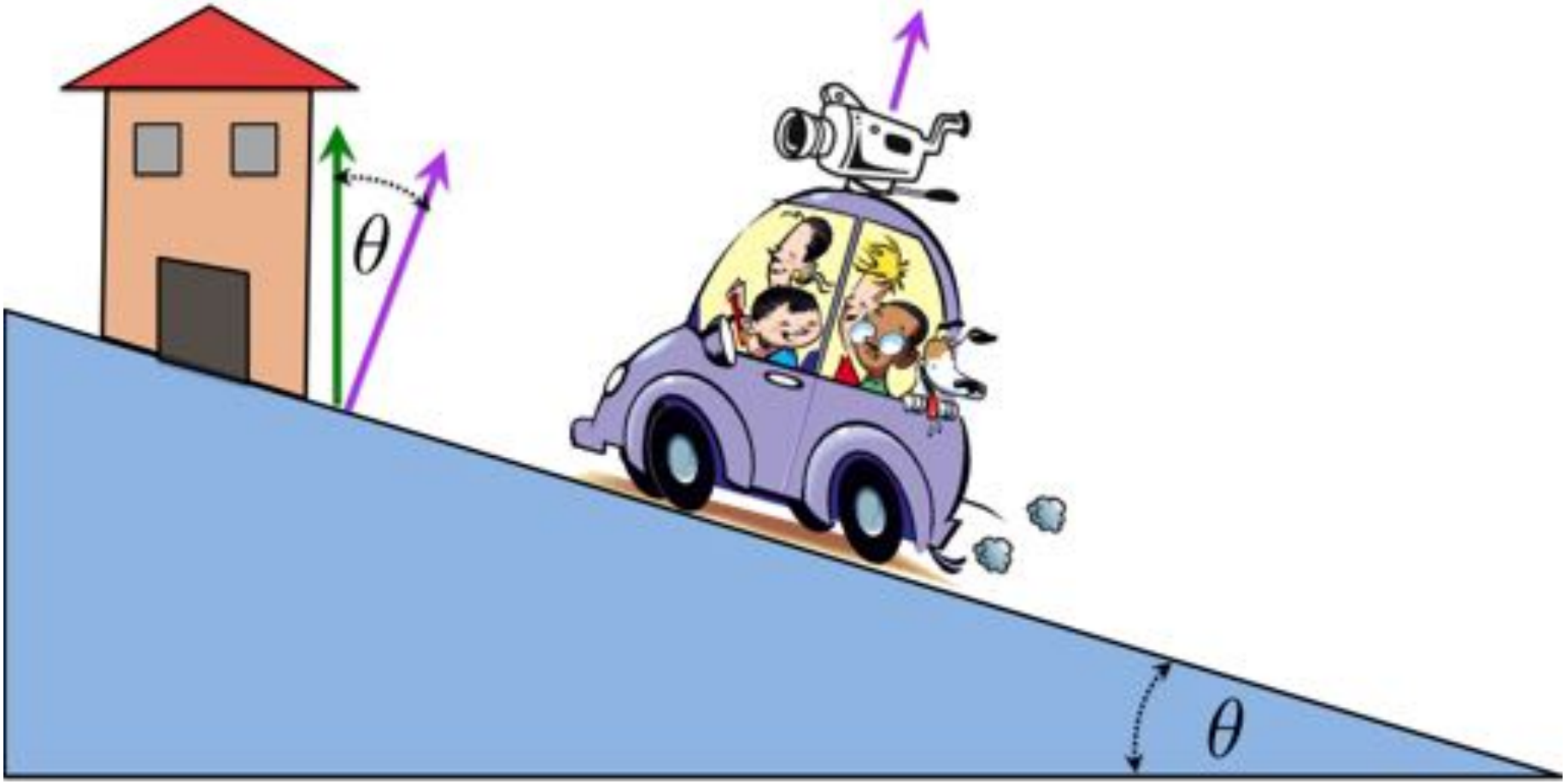


Figure: If we compute the 3D direction of the house's vertical lines relative to camera, we have the incline! How can we do that?

Projection Properties: Cool Tricks

- Can we tell the incline of the hill we are driving on?

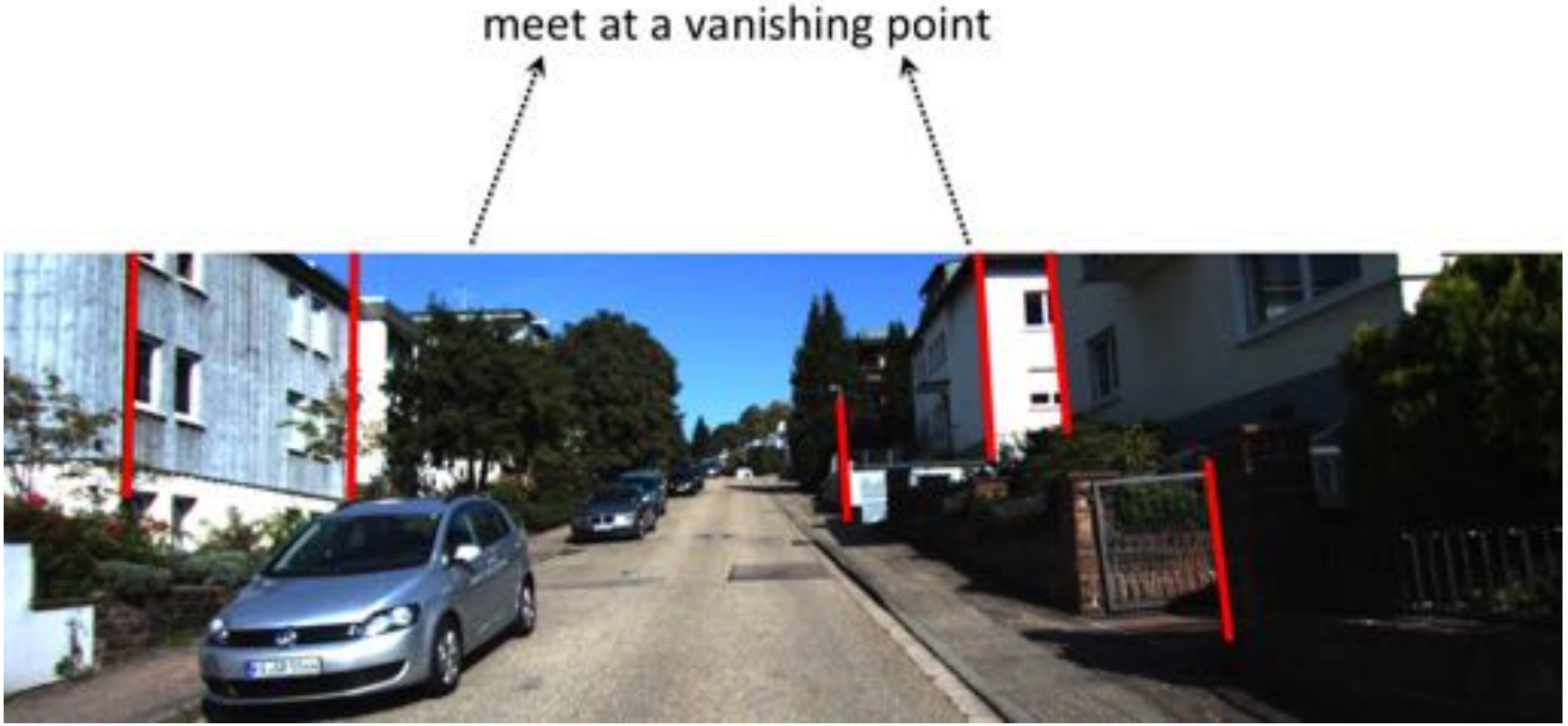


Figure: Extract "vertical" lines and compute vanishing point. How can we compute direction in 3D from vanishing point (if we have K)?

Projection Properties: Cool Tricks

- Can we tell the incline of the hill we are driving on?

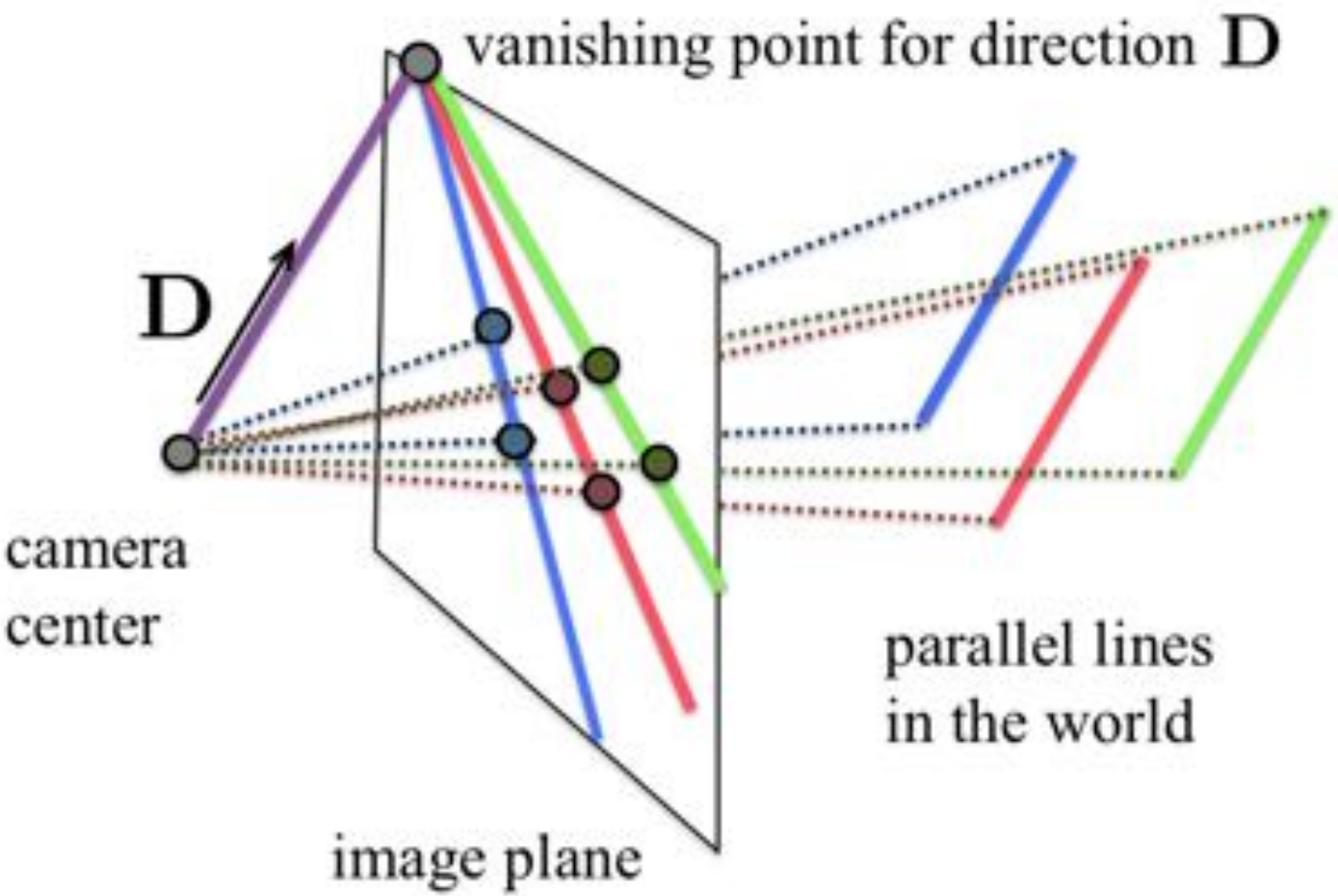
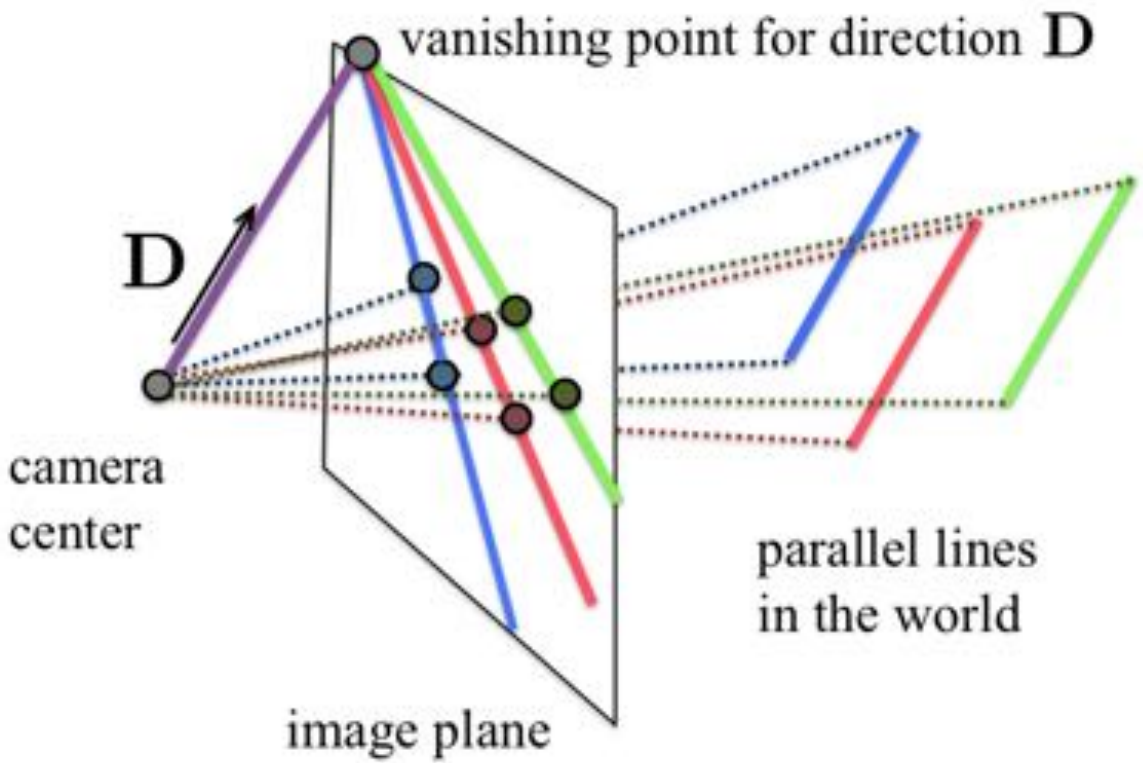


Figure: This picture should help.

Projection Properties: Cool Tricks

- Can we tell the incline of the hill we are driving on?

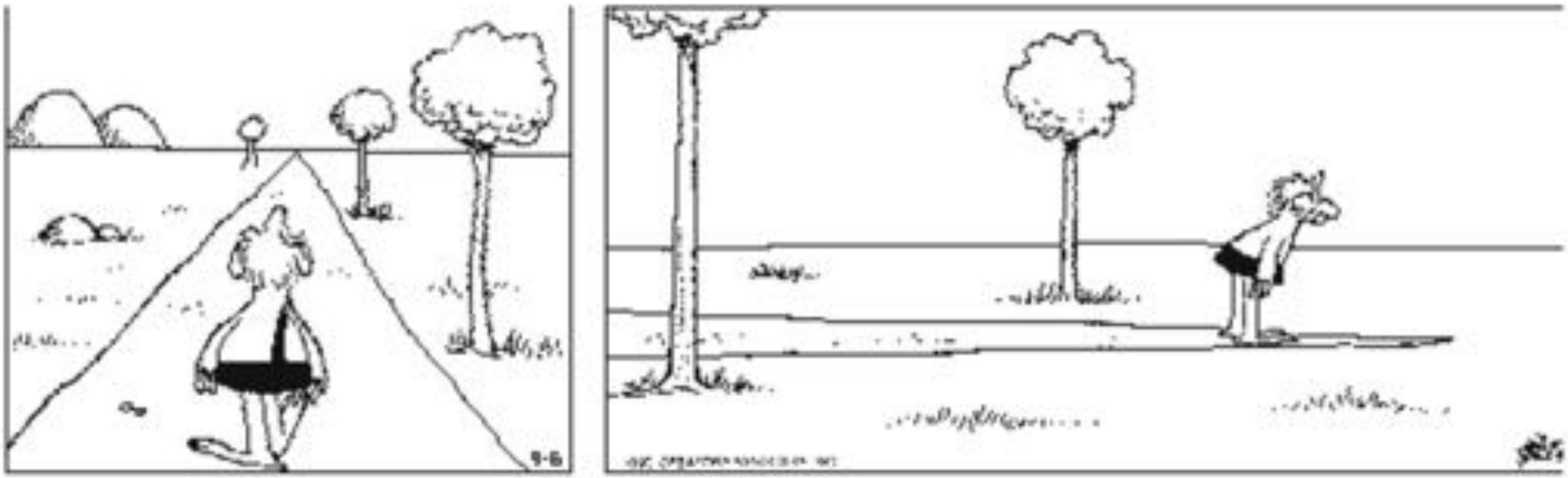


- We have:

$$\begin{bmatrix} w \cdot vp_x \\ w \cdot vp_y \\ w \end{bmatrix} = K\mathbf{D} \quad \rightarrow \quad \mathbf{D} = wK^{-1} \begin{bmatrix} vp_x \\ vp_y \\ 1 \end{bmatrix} \quad \rightarrow \quad \text{normalize } \mathbf{D} \text{ to norm 1}$$

Vanishing Points Can be Deceiving

- Parallel lines converge at a **vanishing point**.
- But intersecting lines in 2D are not necessary parallel in 3D.

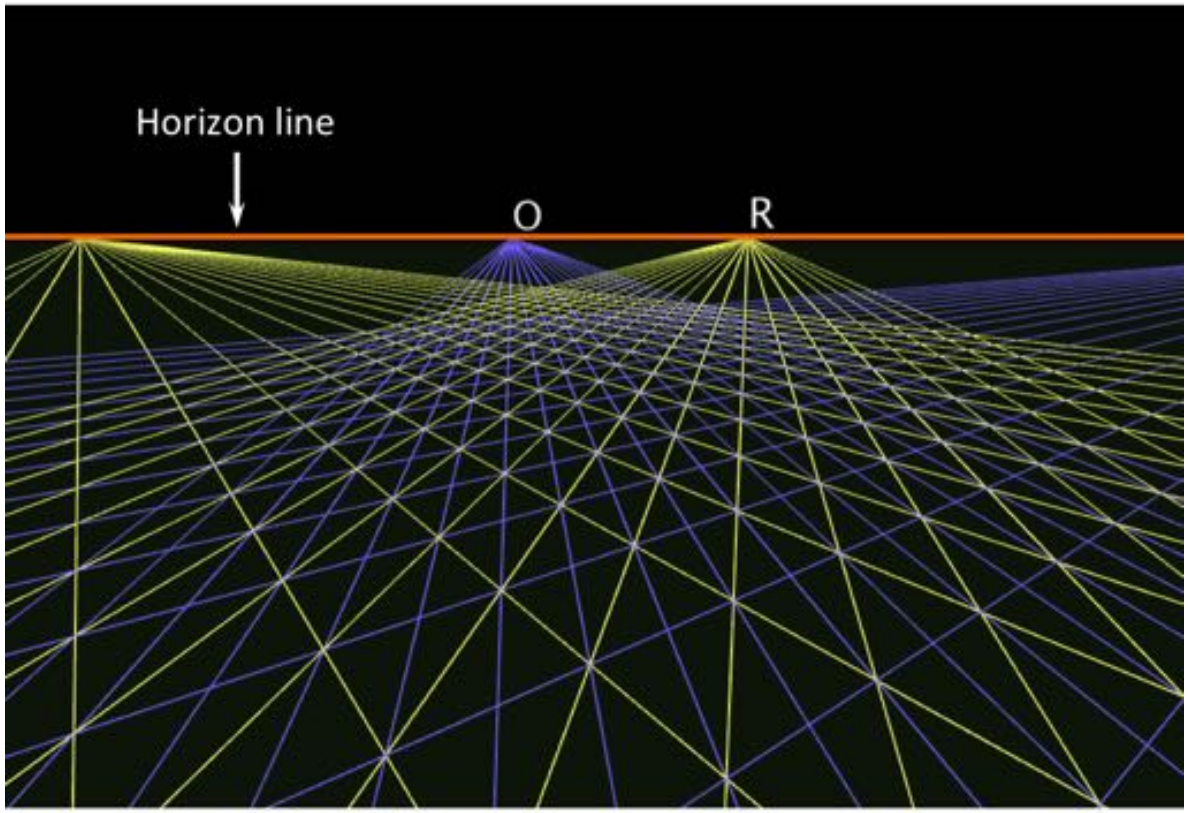


[Source: A. Jepson]

Projection Properties: Cool Facts

Parallel lines converge at a **vanishing point**

- Each different direction in the world **has its own vanishing point**
- For lines on the same 3D plane, the vanishing points lie on a **line**. We call it a **vanishing line**. Vanishing line for the ground plane is a **horizon line**.



http://4.bp.blogspot.com/-0Jm9d9j35Tc/T5ESbVpKI7I/AAAAAAAACEk/nVAiTxBuiyc/s1600/perspectiveGrid-01.png

Projection Properties: Cool Facts

Parallel lines converge at a **vanishing point**

- For lines on the same 3D plane, the vanishing points lie on a **line**. We call it a **vanishing line**. Vanishing line for the ground plane is a **horizon line**.
- Some horizon lines are nicer than others ;)

Punta Cana

Projection Properties: Cool Facts

Parallel lines converge at a **vanishing point**

- For lines on the same 3D plane, the vanishing points lie on a **line**. We call it a **vanishing line** or a **horizon line**.
- Parallel planes in 3D have the **same horizon line** in the image.

Projection Properties: Cool Facts

- Can I tell how much above ground this picture was taken?

Projection Properties: Cool Facts

- Can I tell how much above ground this picture was taken?

Projection Properties: Cool Facts

- Same distance as where the horizon intersects a building

Projection Properties: Cool Facts

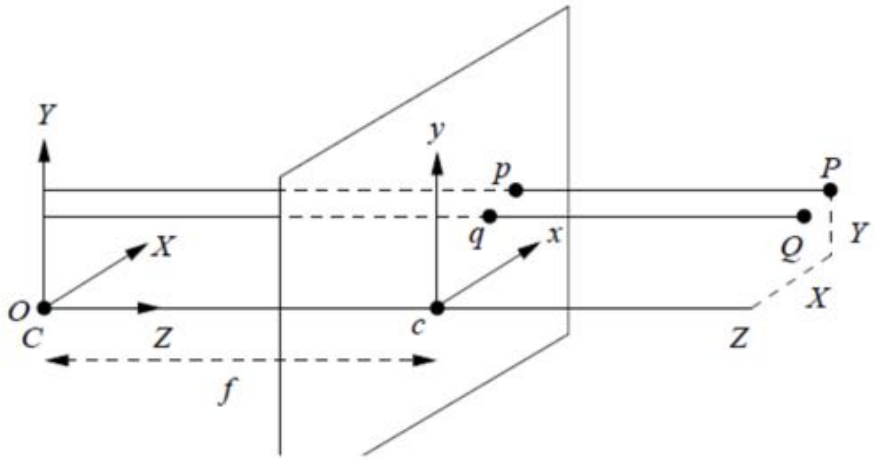
- Same distance as where the horizon intersects a building: 50 floors up

Projection Properties: Cool Facts

- This is only true when the camera (image plane) is orthogonal to the ground plane. And the ground plane is flat.
- A very nice explanation of this phenomena can be find by Derek Hoiem here: `https://courses.engr.illinois.edu/cs543/sp2011/materials/3dscene_book_svg.pdf`

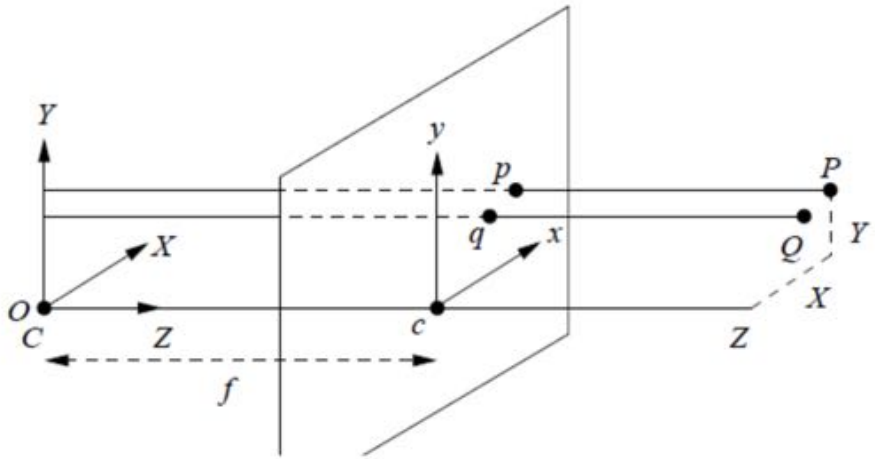
Orthographic Projection

[Source: R. Urtasun]



Orthographic Projection

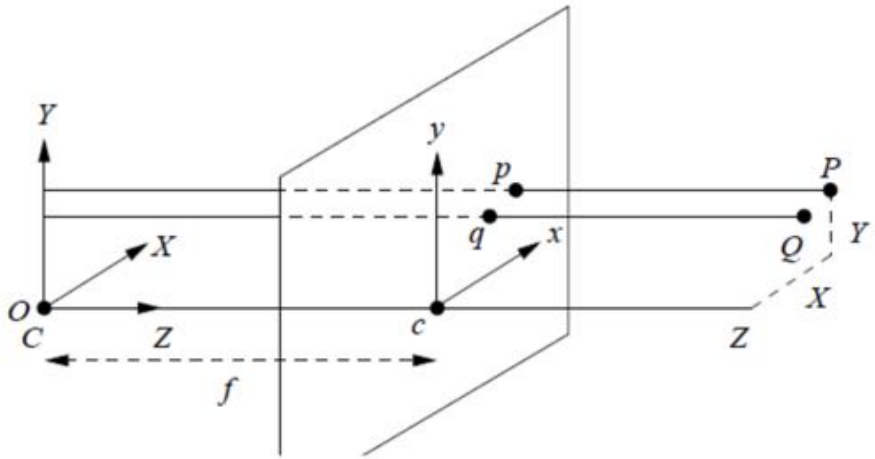
[Source: R. Urtasun]



- Requires no division and simply drops the Z coordinate.

Orthographic Projection

[Source: R. Urtasun]

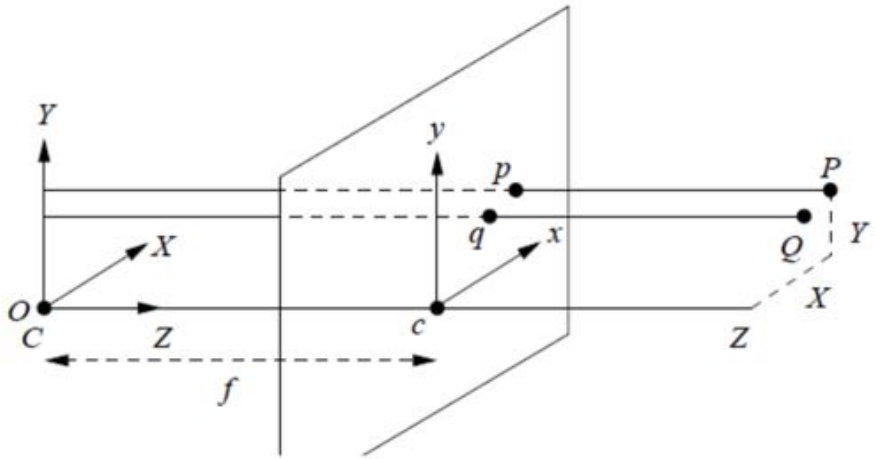


- Requires no division and simply drops the Z coordinate.
- Orthographic projection:

$$\mathbf{Q} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \rightarrow \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Orthographic Projection

[Source: R. Urtasun]

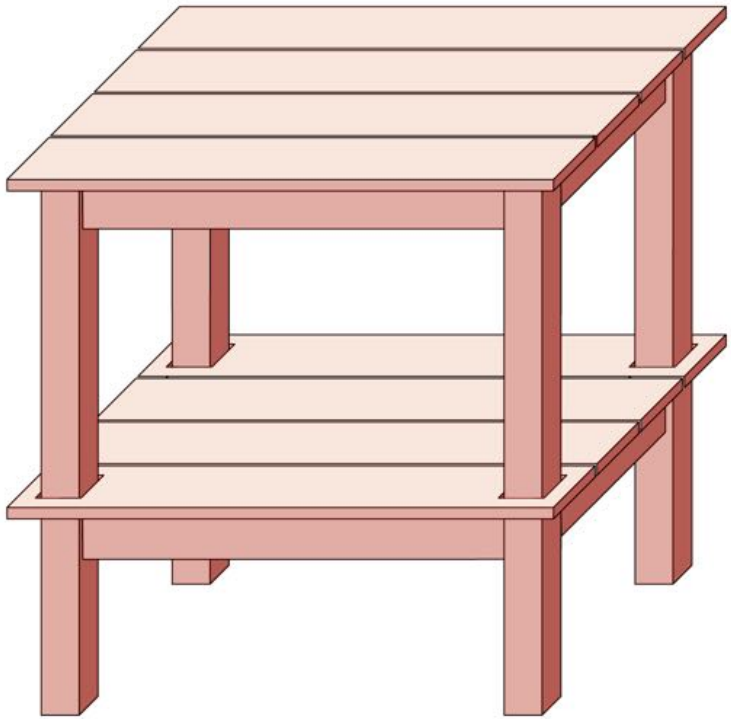


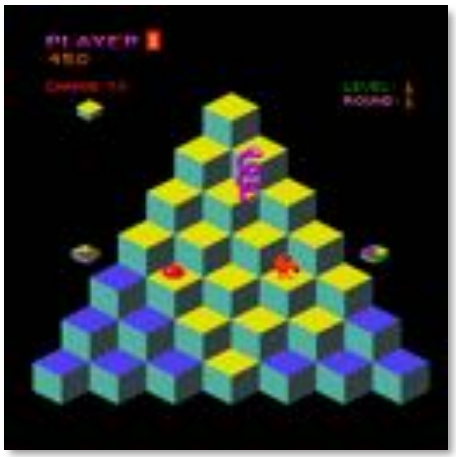
- Requires no division and simply drops the Z coordinate.
- Orthographic projection:

$$\mathbf{Q} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad \rightarrow \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- Special case of perspective projection where the distance from the camera center to the image plane is infinity

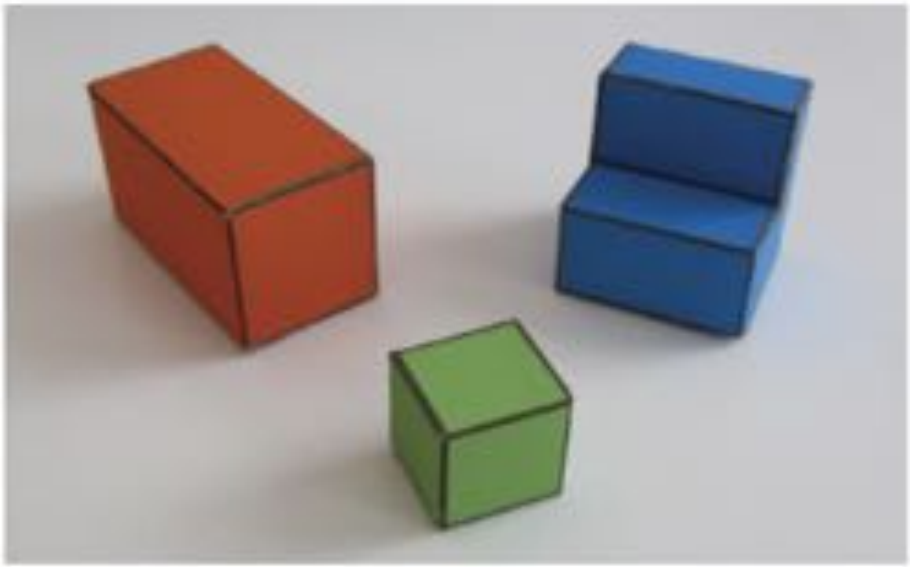
Orthographic Projection



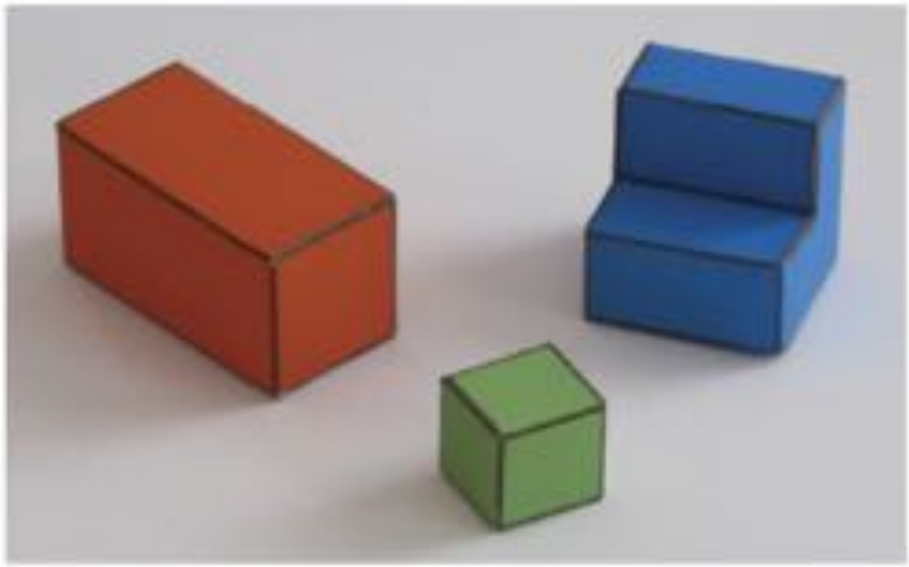


[Source: N. Snavely]

Orthographic Projection



Perspective projection



Parallel (orthographic) projection

- For perspective projection lines parallel in 3D **are not** parallel in the image.
- For orthographic projection lines parallel in 3D **are** parallel in the image.

[Source: A. Torralba]

Camera Parameters

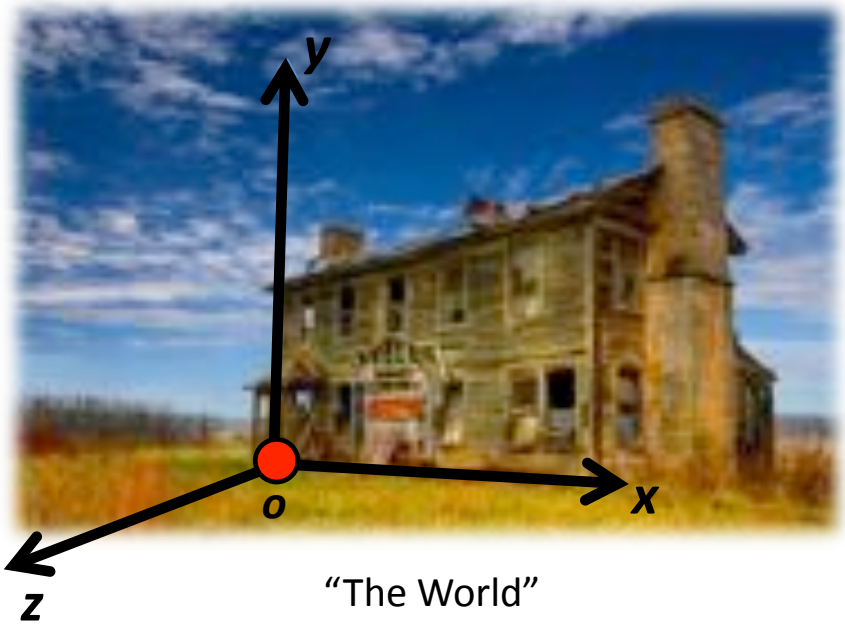
We are not yet done with projection. To fully specify projection, we need to:

- Describe its **internal parameters** (we know this, this is our **K**)

Camera Parameters

We are not yet done with projection. To fully specify projection, we need to:

- Describe its **internal parameters** (we know this, this is our **K**)
- Describe its **pose in the world**. Two important coordinate systems:
  - World coordinate system
  - Camera coordinate system



[Source: N. Snavely, slide credit: R. Urtasun]

Camera Parameters

- Why two coordinate systems?

Figure: Imagine this is your room.

Camera Parameters

- Why two coordinate systems?

Figure: When you were furnishing you measured everything in detail.

Camera Parameters

- Why two coordinate systems?

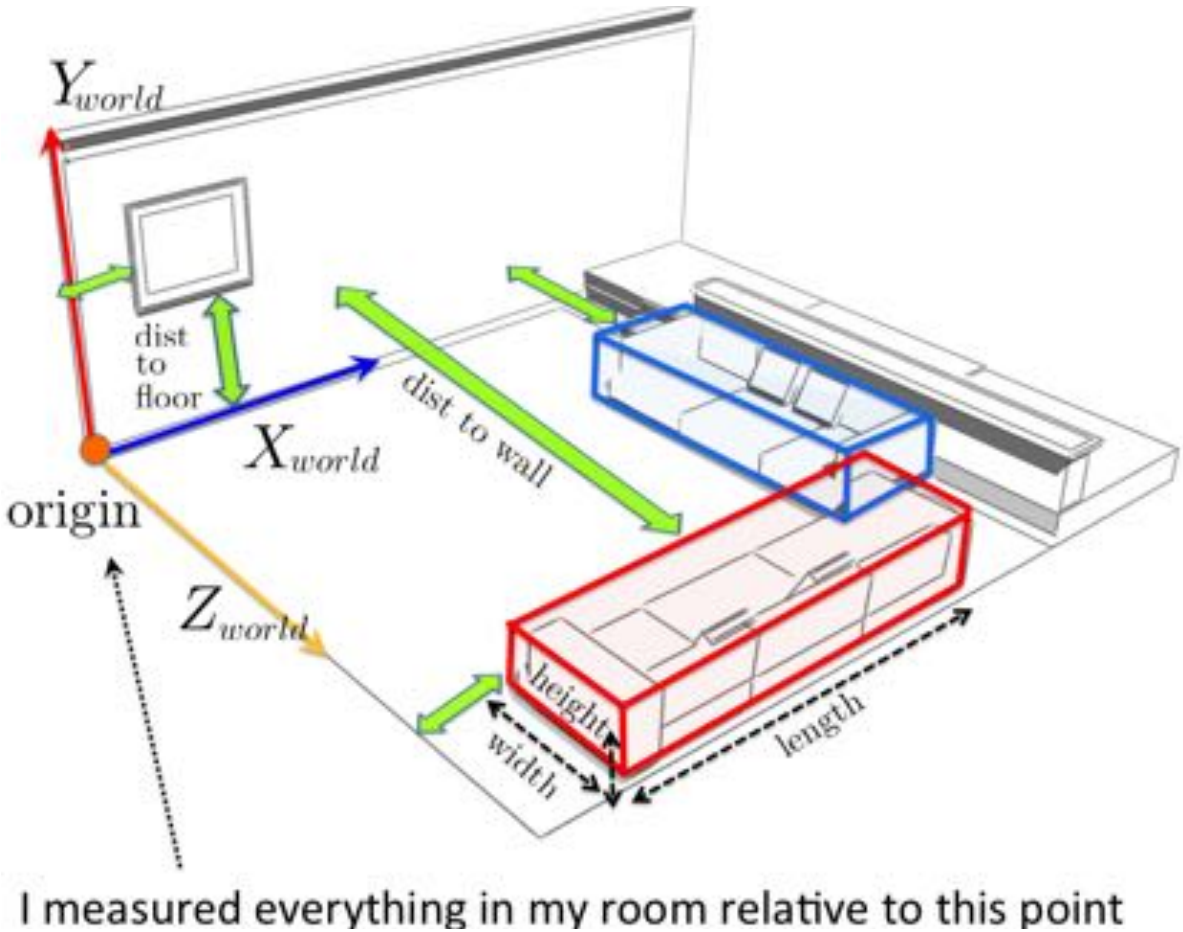


Figure: Thus you know all coordinates relative to a special point (origin) and coordinate system in the room. This is your **room's (world) coordinate system**.

Camera Parameters

- Why two coordinate systems?

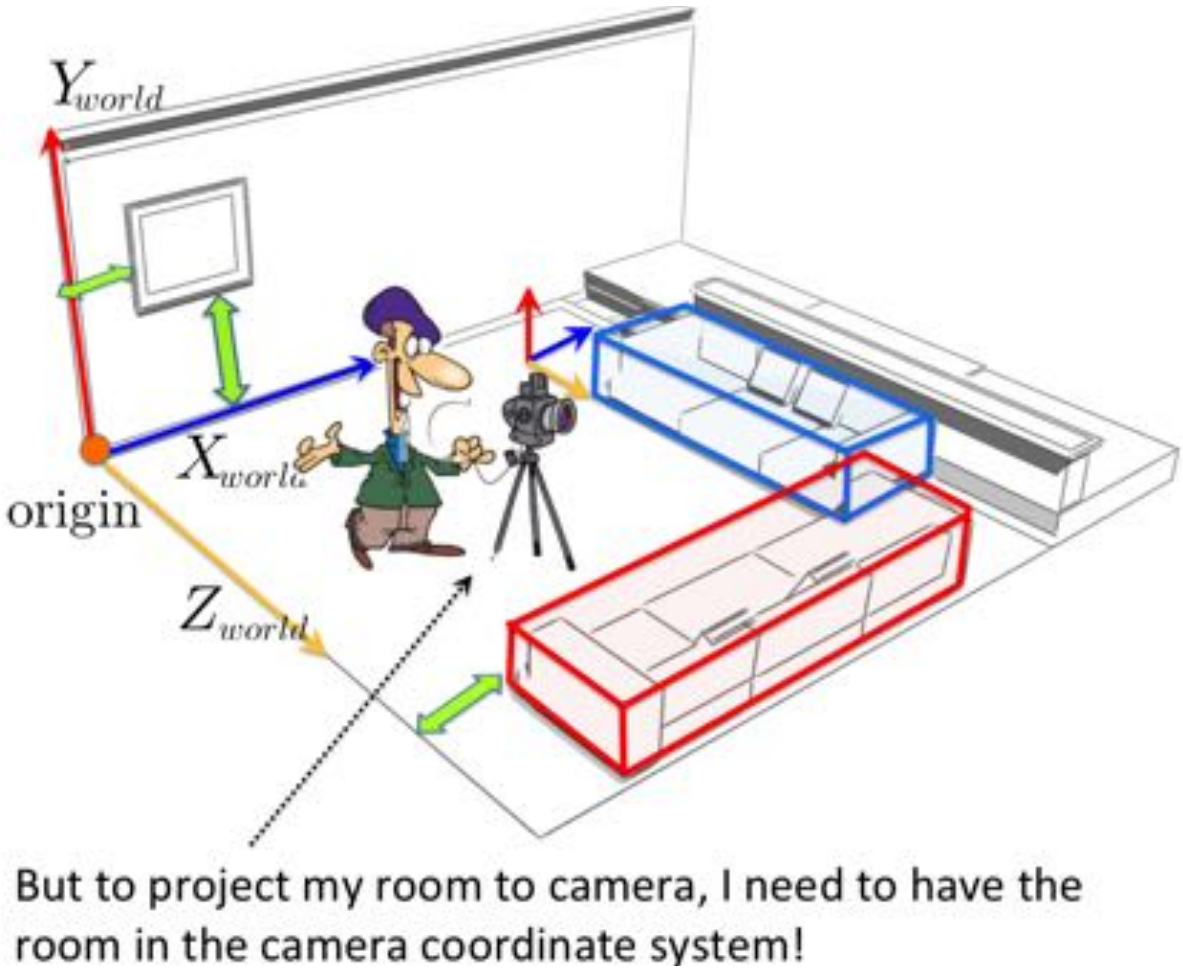


Figure: Now you take a picture and you wonder how points project to camera. In order to project, you need all points in **camera's coordinate system**.

Camera Parameters

- Why two coordinate systems?

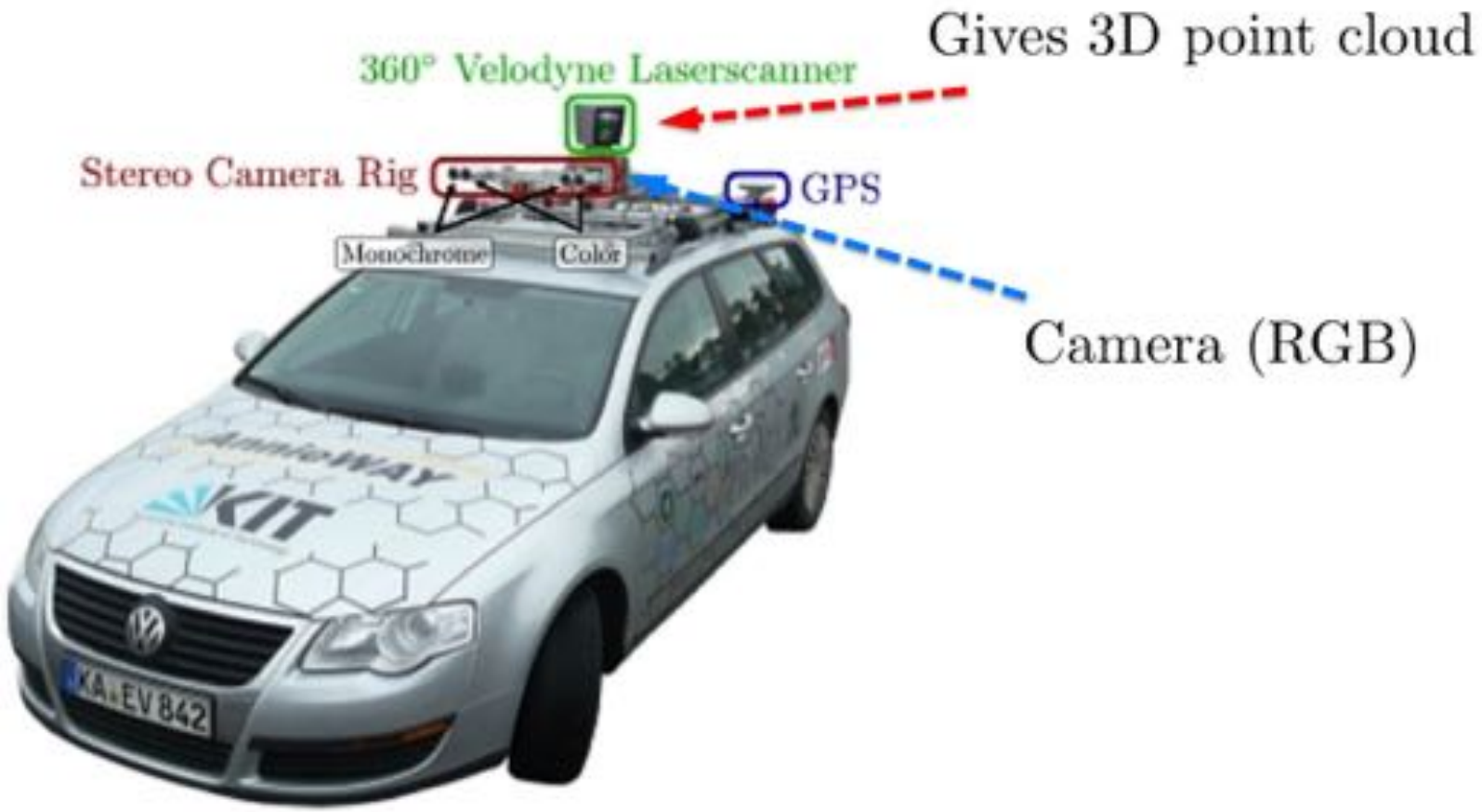


Figure: For e.g. self-driving cars, 3D points are typically measured with Velodyne.

Camera Parameters

- Why two coordinate systems?

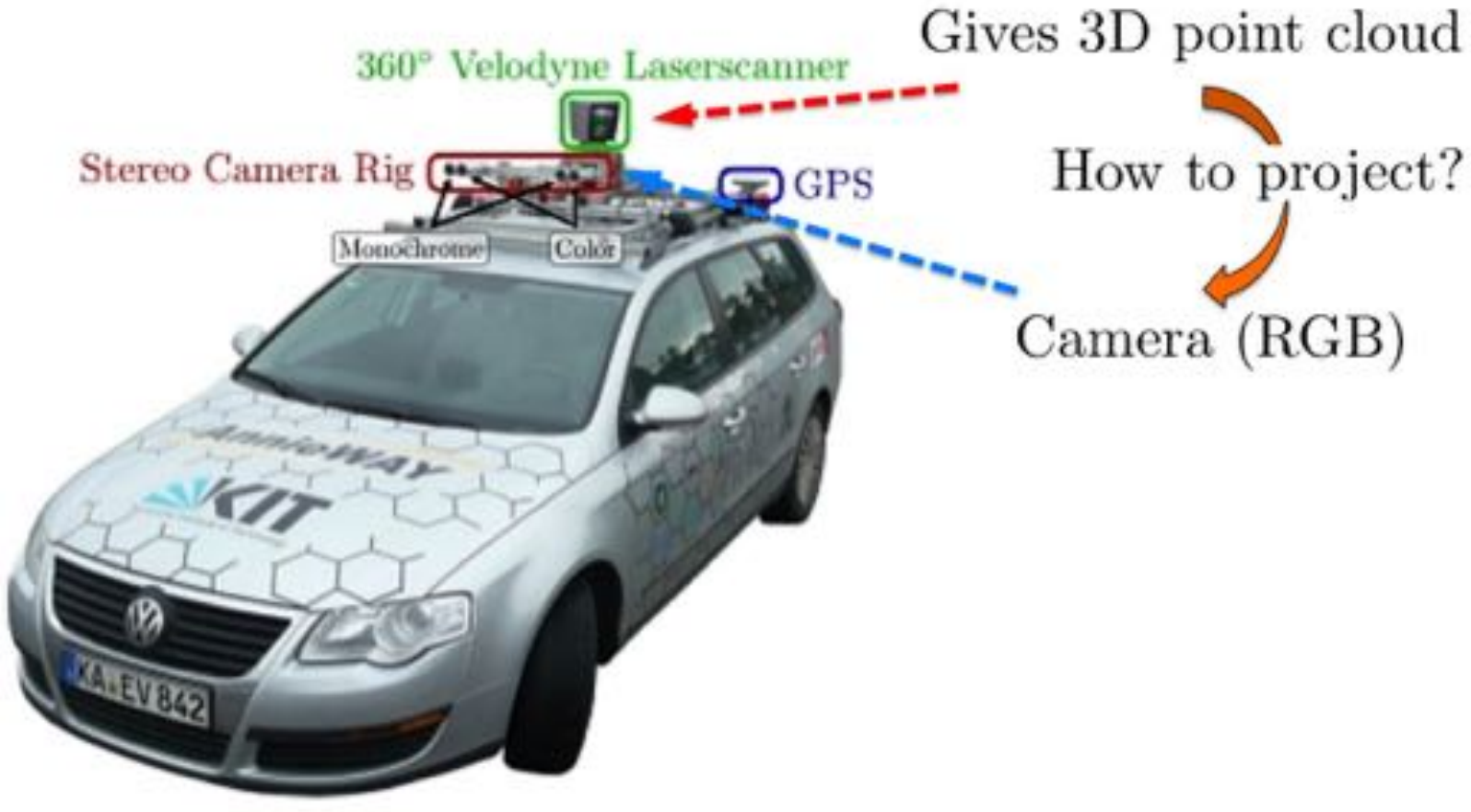


Figure: We want to be able to project the 3D points in Velodyne's coordinate system onto an image captured by a camera.

Camera Parameters

- Why two coordinate systems?

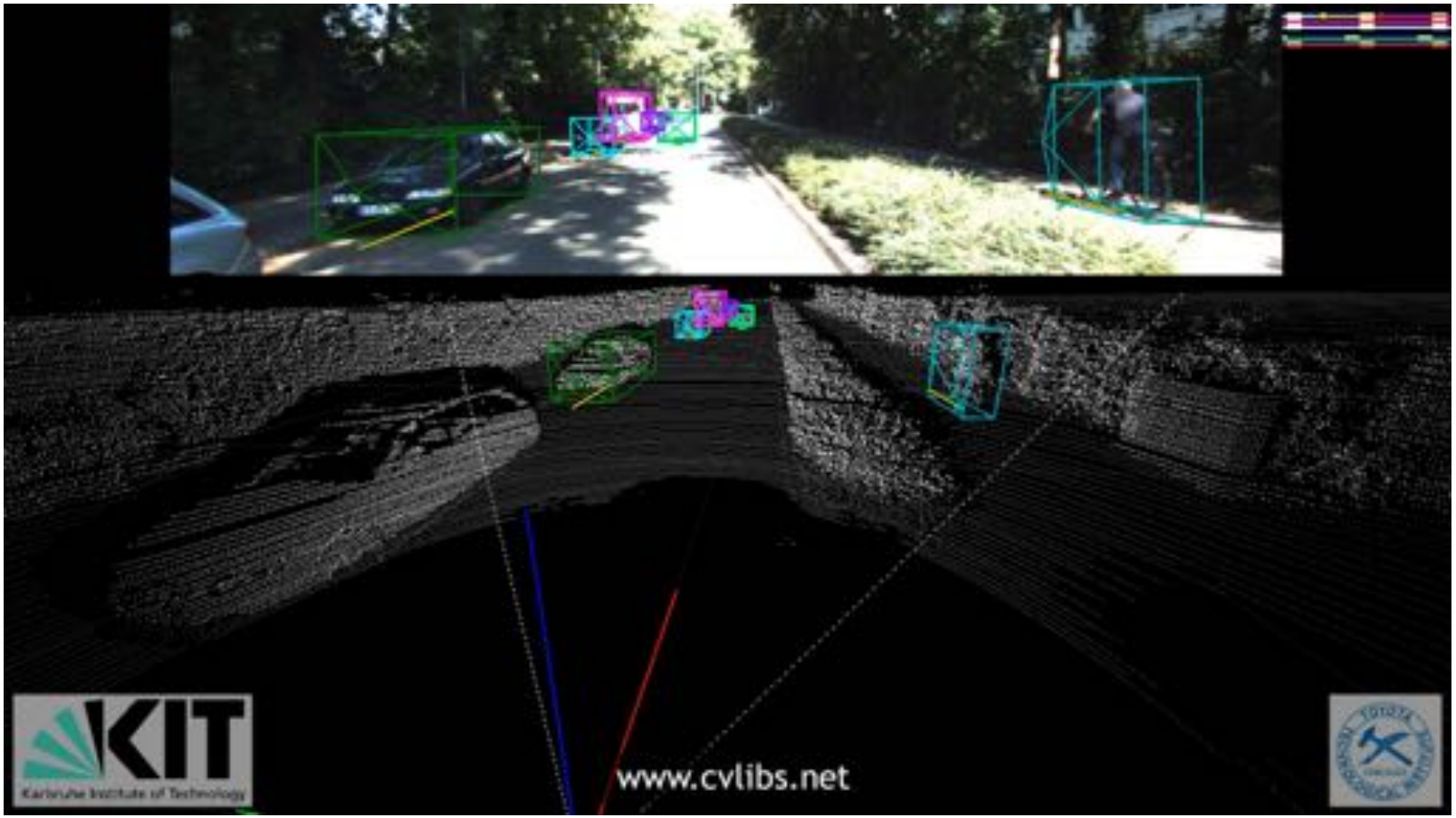


Figure: We want to be able to project the 3D points in Velodyne's coordinate system onto an image captured by a camera.

Projection

To project a point (X, Y, Z) in world coordinates on the image plane, we need to:

Projection

To project a point (X, Y, Z) in world coordinates on the image plane, we need to:

- Transform (X, Y, Z) into camera coordinates. We thus need:

Projection

To project a point (X, Y, Z) in world coordinates on the image plane, we need to:

- Transform (X, Y, Z) into camera coordinates. We thus need:
  - Camera **position** (in world coordinates)

Projection

To project a point (X, Y, Z) in world coordinates on the image plane, we need to:

- Transform (X, Y, Z) into camera coordinates. We thus need:
  - Camera **position** (in world coordinates)
  - Camera **orientation** (in world coordinates)

Projection

To project a point (X, Y, Z) in world coordinates on the image plane, we need to:

- Transform (X, Y, Z) into camera coordinates. We thus need:
  - Camera **position** (in world coordinates)
  - Camera **orientation** (in world coordinates)
- To project into the image plane
  - Need to know **camera intrinsics**

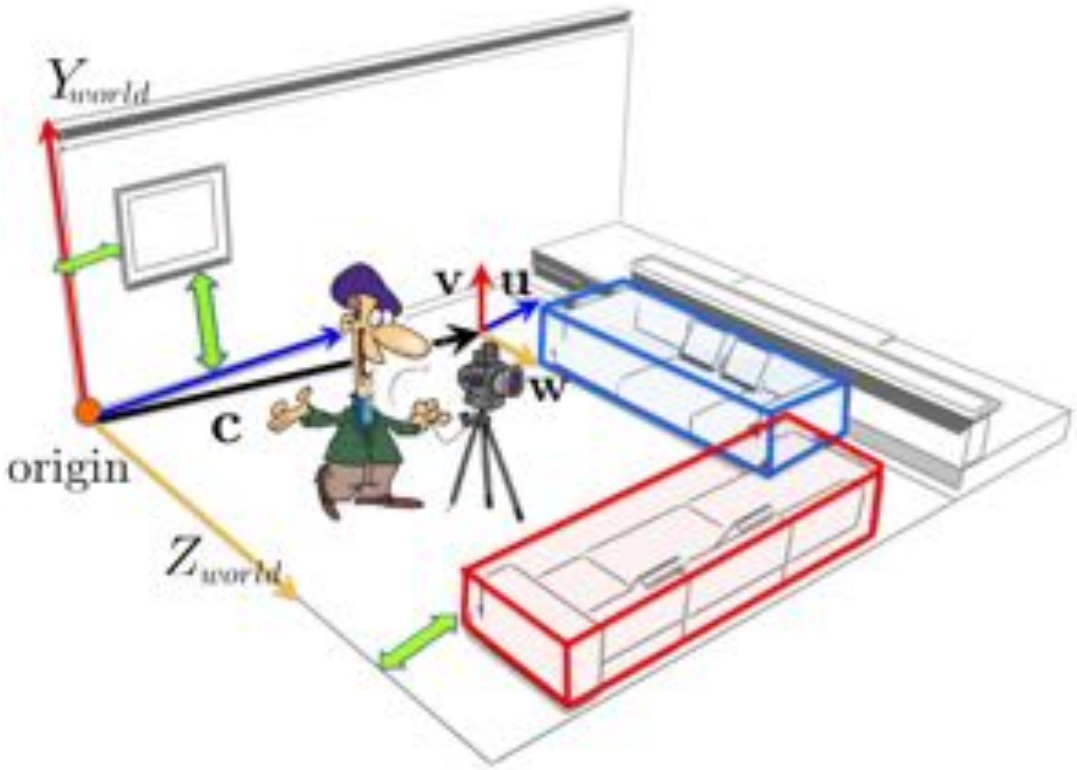
Projection

To project a point (X, Y, Z) in world coordinates on the image plane, we need to:

- Transform (X, Y, Z) into camera coordinates. We thus need:
  - Camera **position** (in world coordinates)
  - Camera **orientation** (in world coordinates)
- To project into the image plane
  - Need to know **camera intrinsics**
- These can all be described with matrices!

[Source: N. Snavely, slide credit: R. Urtasun]

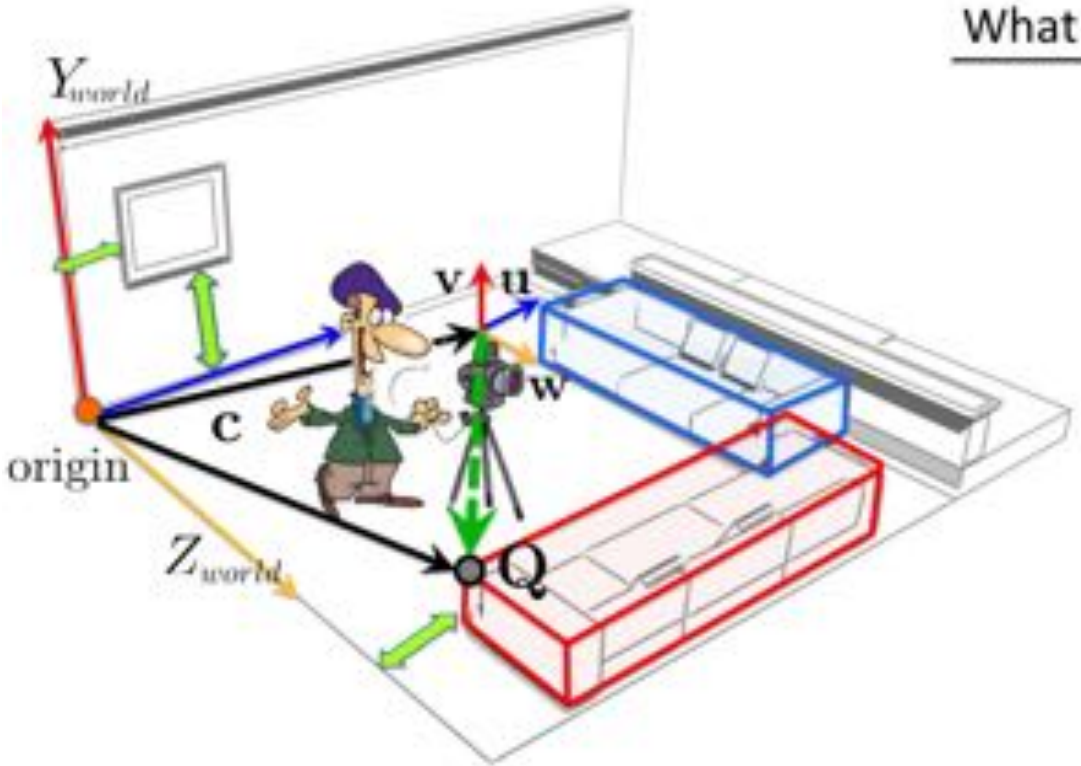
Camera Extrinsics



c	...	camera position in room coordinate system
u, v, w	...	3 orthogonal directions of camera in room coordinate system

Figure: We first need our camera position and orientation in the room's world.

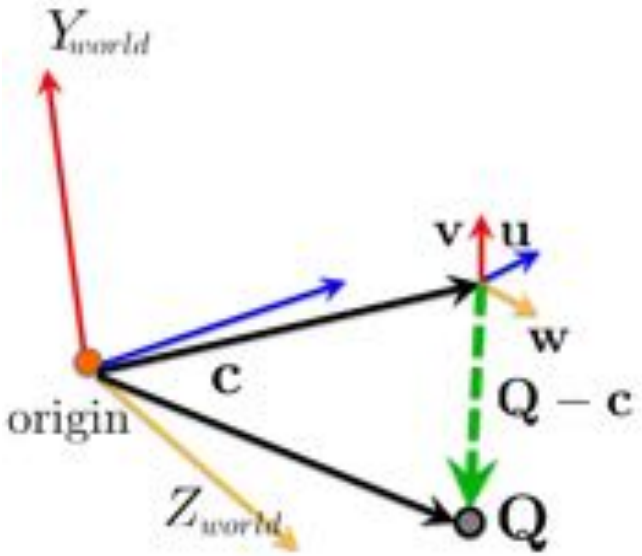
Camera Extrinsics



What is Q in camera's coordinate system??

c	...	camera position in room coordinate system
u, v, w	...	3 orthogonal directions of camera in room coordinate system

Camera Extrinsics



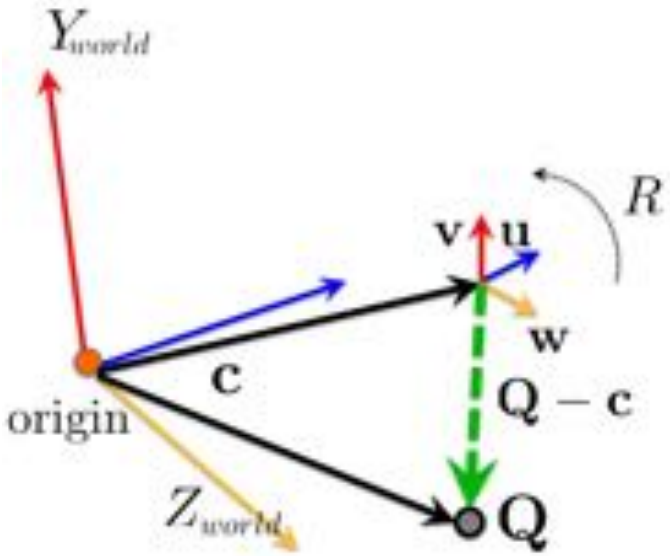
What is Q in camera's coordinate system??

$\mathbf{Q} - \mathbf{c}$ ... makes **position** relative to camera

$\mathbf{c}$ ... camera position in room coordinate system

$\mathbf{u}, \mathbf{v}, \mathbf{w}$ ... 3 orthogonal directions of camera in room coordinate system

Camera Extrinsics



What is Q in camera's coordinate system??

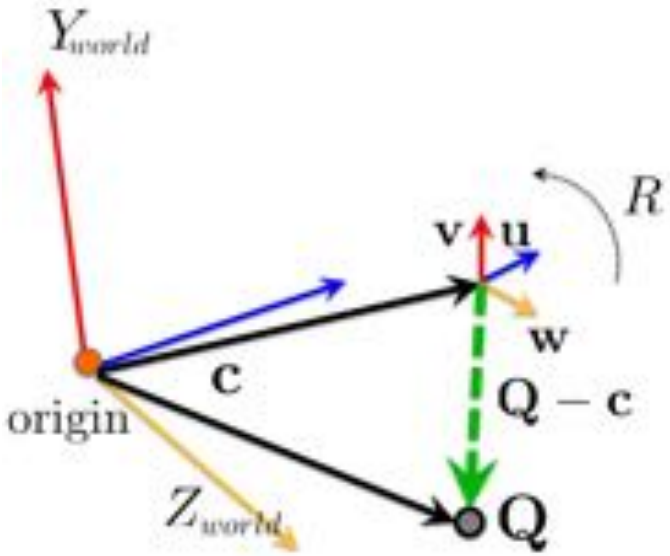
$\mathbf{Q} - \mathbf{c}$ ... makes **position** relative to camera

$R\begin{bmatrix}\mathbf{u} & \mathbf{v} & \mathbf{w}\end{bmatrix} = I$ (looking for rotation to canonical orientation)

$\mathbf{c}$ ... camera position in room coordinate system

$\mathbf{u}, \mathbf{v}, \mathbf{w}$ ... 3 orthogonal directions of camera in room coordinate system

Camera Extrinsics



What is Q in camera's coordinate system??

$\mathbf{Q} - \mathbf{c}$ ... makes **position** relative to camera

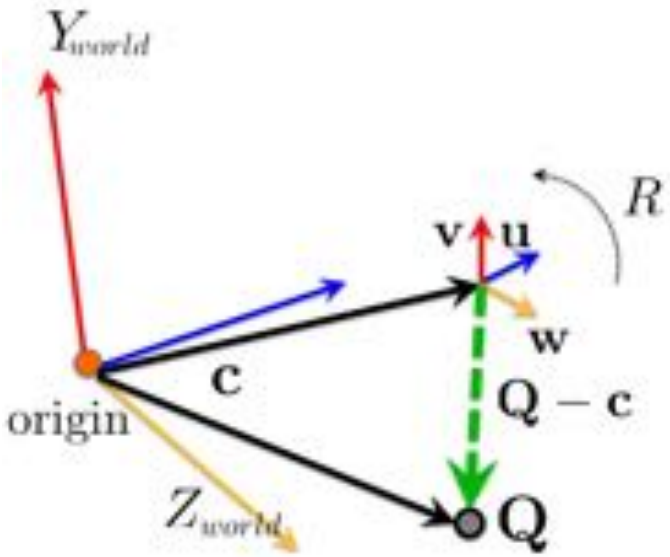
$R \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} \end{bmatrix} = I$ (looking for rotation to canonical orientation)

$R \cdot R^T = I$ (since orientation is orthogonal matrix)

$\mathbf{c}$ ... camera position in room coordinate system

$\mathbf{u}, \mathbf{v}, \mathbf{w}$ ... 3 orthogonal directions of camera in room coordinate system

Camera Extrinsics



What is Q in camera's coordinate system??

$\mathbf{Q} - \mathbf{c}$ ... makes **position** relative to camera

$R \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} \end{bmatrix} = I$ (looking for rotation to canonical orientation)

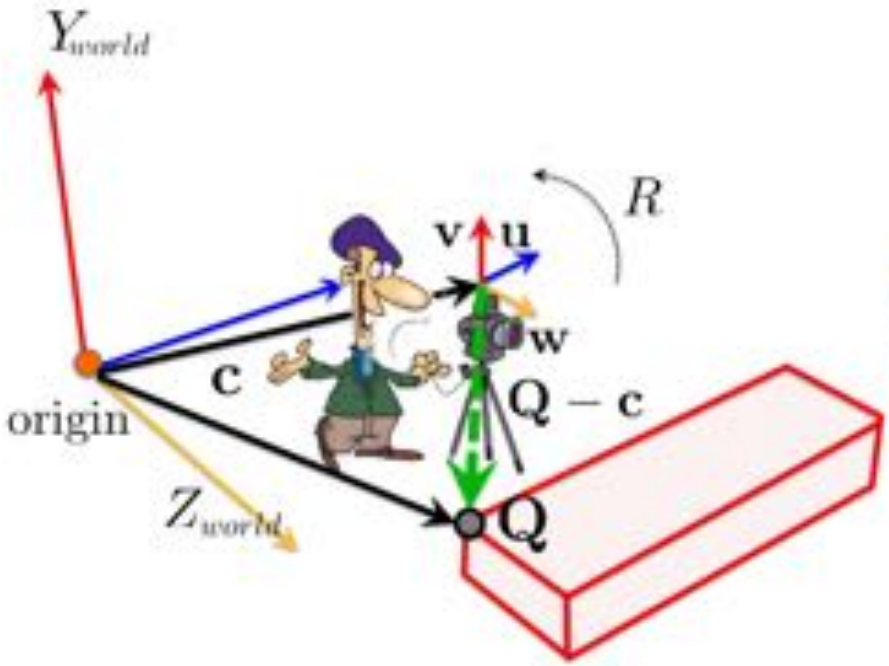
$R \cdot R^T = I$ (since orientation is orthogonal matrix)

$R = \begin{bmatrix} \mathbf{u}^T & \mathbf{v}^T & \mathbf{w}^T \end{bmatrix}$

$\mathbf{c}$ ... camera position in room coordinate system

$\mathbf{u}, \mathbf{v}, \mathbf{w}$ ... 3 orthogonal directions of camera in room coordinate system

Camera Extrinsics



What is Q in camera's coordinate system??

$\mathbf{Q} - \mathbf{c}$ ... makes **position** relative to camera

$R\begin{bmatrix}\mathbf{u} & \mathbf{v} & \mathbf{w}\end{bmatrix} = I$ (looking for rotation to canonical orientation)

$R \cdot R^T = I$ (since orientation is orthogonal matrix)

$R = \begin{bmatrix}\mathbf{u}^T & \mathbf{v}^T & \mathbf{w}^T\end{bmatrix}$

$$\begin{bmatrix}X'\\Y'\\Z'\end{bmatrix} = R\left(\begin{bmatrix}X\\Y\\Z\end{bmatrix} - \mathbf{c}\right) = \begin{bmatrix}R & -Rc\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$

camera coordinates

room (world) coordinates

$\mathbf{c}$ ... camera position in room coordinate system

$\mathbf{u}, \mathbf{v}, \mathbf{w}$ ... 3 orthogonal directions of camera in room coordinate system

Figure: Final Transformation

Projection Equations

- **Projection matrix** P models the cumulative effect of all intrinsic and extrinsic parameters. We use homogeneous coordinates for 2D and 3D:

$$\mathbf{q} = \begin{bmatrix} ax \\ ay \\ a \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Projection Equations

- **Projection matrix** P models the cumulative effect of all intrinsic and extrinsic parameters. We use homogeneous coordinates for 2D and 3D:

$$\mathbf{q} = \begin{bmatrix} ax \\ ay \\ a \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- It can be computed as

$$\mathbf{P} = \underbrace{\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsics } K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{projection}} \underbrace{\begin{bmatrix} \mathbf{R}_{3\times 3} & 0_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{T}_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix}}_{\text{translation}}$$

Projection Equations

- **Projection matrix** P models the cumulative effect of all intrinsic and extrinsic parameters. We use homogeneous coordinates for 2D and 3D:

$$\mathbf{q} = \begin{bmatrix} ax \\ ay \\ a \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

- It can be computed as

$$\mathbf{P} = \underbrace{\begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsics } K} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{projection}} \underbrace{\begin{bmatrix} \mathbf{R}_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{T}_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}}_{\text{translation}}$$

- To get a point $\mathbf{q}$ in the image plane, I need to compute $\mathbf{P}(X, Y, Z, 1)^T$, where $\mathbf{P}$ is a 3×4 matrix. This gives me a 3×1 vector. Now I divide all coordinates with the third coordinate (making the third coordinate equal to 1), and then drop the last coordinate. As simple as that.

The Projection Matrix

- The projection matrix is defined as

$$\mathbf{P} = \underbrace{\mathbf{K}}_{\text{intrinsics}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{projection}} \underbrace{\underbrace{\begin{bmatrix} \mathbf{R}_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} \mathbf{I}_{3\times3} & \mathbf{T}_{3\times3} \\ 0_{1\times3} & 1 \end{bmatrix}}_{\text{translation}}}_{\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}}$$

- More compactly

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}$$

- Sometimes you will see notation:

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} \mid \mathbf{t} \end{bmatrix}$$

  It's the same thing.

The Projection Matrix

- The projection matrix is defined as

$$\mathbf{P} = \underbrace{\mathbf{K}}_{\text{intrinsics}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{projection}} \underbrace{\underbrace{\begin{bmatrix} \mathbf{R}_{3\times 3} & 0_{3\times 1} \\ 0_{1\times 3} & 1 \end{bmatrix}}_{\text{rotation}} \underbrace{\begin{bmatrix} \mathbf{I}_{3\times 3} & \mathbf{T}_{3\times 3} \\ 0_{1\times 3} & 1 \end{bmatrix}}_{\text{translation}}}_{\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}}$$

- More compactly

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}$$

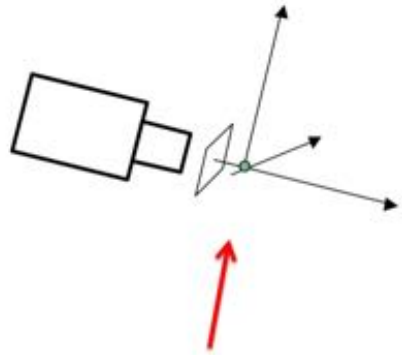
- Sometimes you will see notation:

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} \mid \mathbf{t} \end{bmatrix}$$

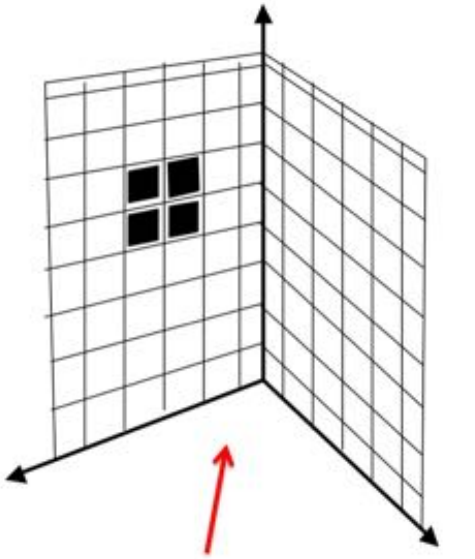
  It's the same thing.

- This might look complicated. Truth is, in most cases you don't have $\mathbf{P}$ at all, so you can't really compute any projections. When you have a calibrated camera, then someone typically gives you $\mathbf{P}$. And then projection is easy.

A Short Note on Camera Calibration



Detect corners in image and figure out which corner corresponds to which point in the 3D pattern.



You measured all distances for this pattern.

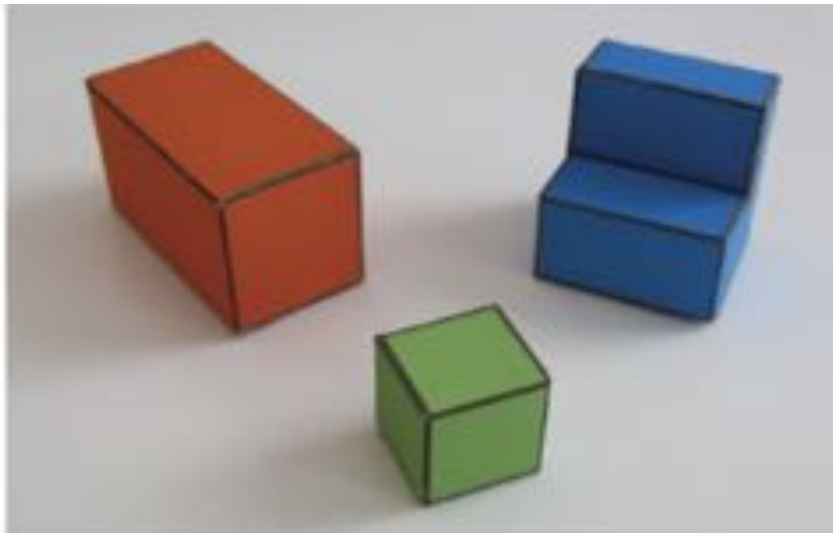
The general procedure:

- Place a 3D pattern (for which you know all distances) in front of camera.
- Take a picture. Detect corners in image and find correspondences with the points in the pattern.
- Go to the internet and check out the math that tells you how to compute K from these 2D-3D correspondences. ;) We won't cover in class.

[Pic from: R. Duraiswami]

Camera Calibration: Interesting Fact

- Let's say you have an image but you don't know **anything** about the camera (for example, image downloaded from the web).
- For images where you see lines corresponding to 3 orthogonal directions, like cubes or rooms, you can compute the camera matrix K as well as R and t!



- How to do this is explained in the Zisserman & Hartley book.

Projection Properties: Cool Facts

- As a consequence, for scenes with lots of lines (e.g. man-made scenes) one can reconstruct the scene in 3D from a **single image**!

Projection Properties: Cool Facts

- As a consequence, for scenes with lots of lines (e.g. man-made scenes) one can reconstruct the scene in 3D from a **single image**!

Projection Properties: Cool Facts

- As a consequence, for scenes with lots of lines (e.g. man-made scenes) one can reconstruct the scene in 3D from a **single image**!
- For those interested, check out the math here:

A. Criminisi, I. Reid, and A. Zisserman

Single View Metrology

International Journal of Computer Vision, vol 40, num 2, 2000

http://www.cs.cmu.edu/ ph/869/papers/Criminisi99.pdf

Inserting Objects

[Karsch et al., 2011]

K. Karsch, V. Hedau, D. Forsyth, D. Hoiem, Rendering synthetic objects into legacy photographs, SIGGRAPH'11

link to video

Camera Calibration: Another Interesting Fact

- From a longer **video** in which the sun travels across the sky you can compute the camera intrinsic matrix, as well as extrinsic, i.e., the GPS location where you are! Well, up to a 100km accuracy...

Camera Calibration: Another Interesting Fact

J.-F. Lalonde, S. G. Narasimhan, and A. A. Efros

What Do the Sun and Sky Tell Us About the Camera?

International Journal on Computer Vision, 88(1), May 2010
Paper: `http://vision.gel.ulaval.ca/~jflalonde/projects/sky/index.html`
Code: `https://github.com/jflalonde/webcamCalibration`

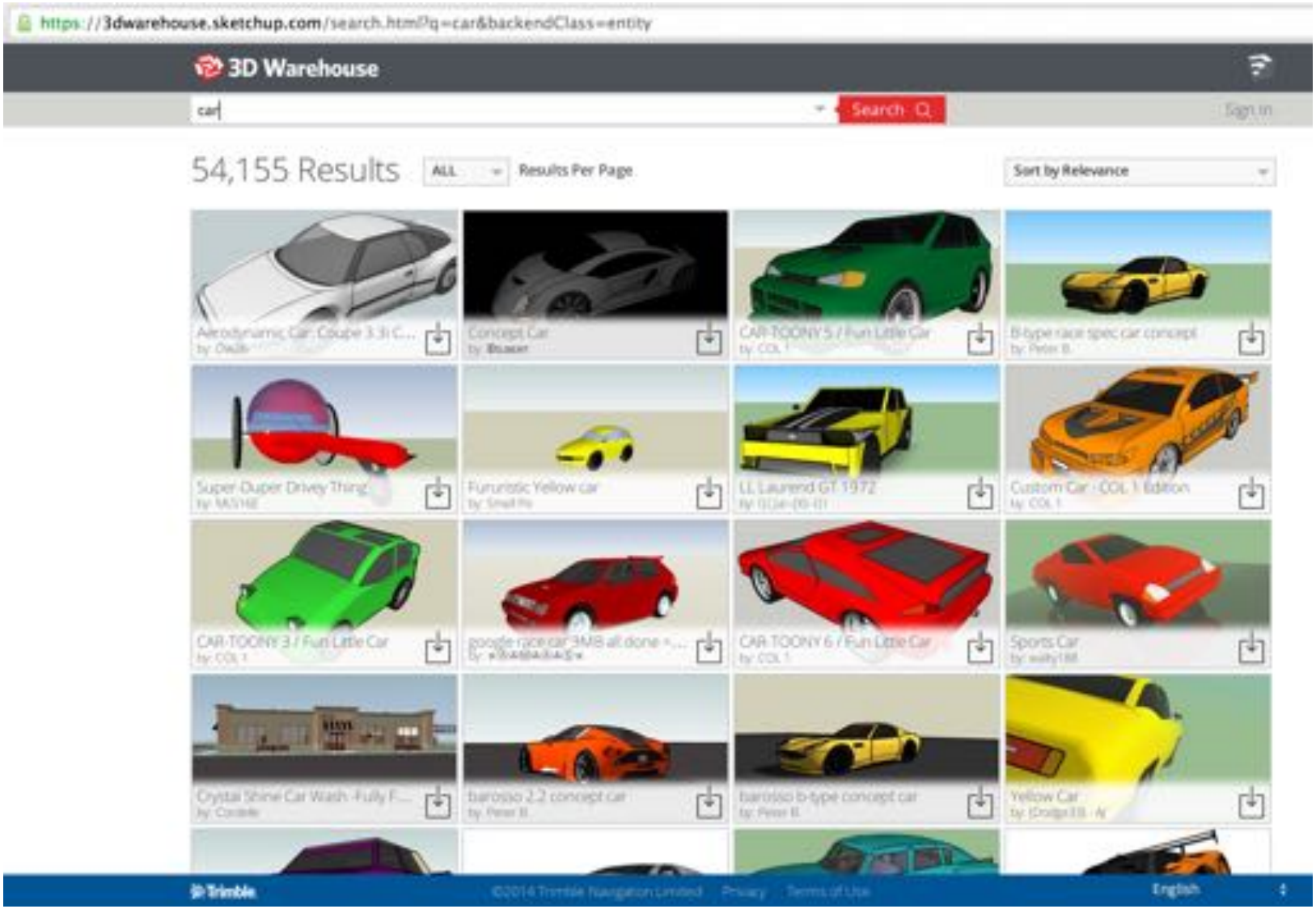
- From a longer **video** in which the sun travels across the sky you can compute the camera intrinsic matrix, as well as extrinsic, i.e., the GPS location where you are! Well, up to a 100km accuracy...
- Is this useful? Maybe, to catch terrorists that record their videos outside.

Exercise (Not Very Easy, But Fun)

- We want to render (project) a 3D CAD model of a car to this image in a realistic way
- How?

Exercise

- First get a CAD model. There are tones of them, e.g. 3D Warehouse (free)



Exercise

- We downloaded this model. Now what?

Figure: A CAD model is a collection of 3D vertices and faces that connect the vertices. Each face represents a small triangle. It typically has color.

Exercise

- Our image was collected with a car on the road:
  - A camera was on top of the car, approximately $1.7m$ above ground
  - Image plane is orthogonal to the ground
  - We have the internal parameters of the camera, $\mathbf{K}$.

Exercise

- Our image was collected with a car on the road:
  - A camera was on top of the car, approximately $1.7m$ above ground
  - Image plane is orthogonal to the ground
  - We have the internal parameters of the camera, $\mathbf{K}$.
- With a little bit of math, we can compute the ground plane in 3D, relative to camera. We a bit more math we can compute which point on the ground plane projects to an image point (x, y).

  How?
- We can now "place" our CAD model to this point (compute R and $\mathbf{t}$)

Exercise

- Our image was collected with a car on the road:
  - A camera was on top of the car, approximately $1.7m$ above ground
  - Image plane is orthogonal to the ground
  - We have the internal parameters of the camera, $\mathbf{K}$.
- With a little bit of math, we can compute the ground plane in 3D, relative to camera. We a bit more math we can compute which point on the ground plane projects to an image point (x, y).

  How?
- We can now "place" our CAD model to this point (compute R and $\mathbf{t}$)
- Rendering:

  Compute $[ax, ay, a]^T = K\big[R \mid \mathbf{t}\big][X, Y, Z, 1]^T$ for each CAD vertex$[X, Y, Z]^T$. Divide $[ax, ay, a]^T$ with third coord and drop it.

Exercise

- That's it. Make a video for more fun

Exercise

- That's it. Make a video for more fun

Exercise

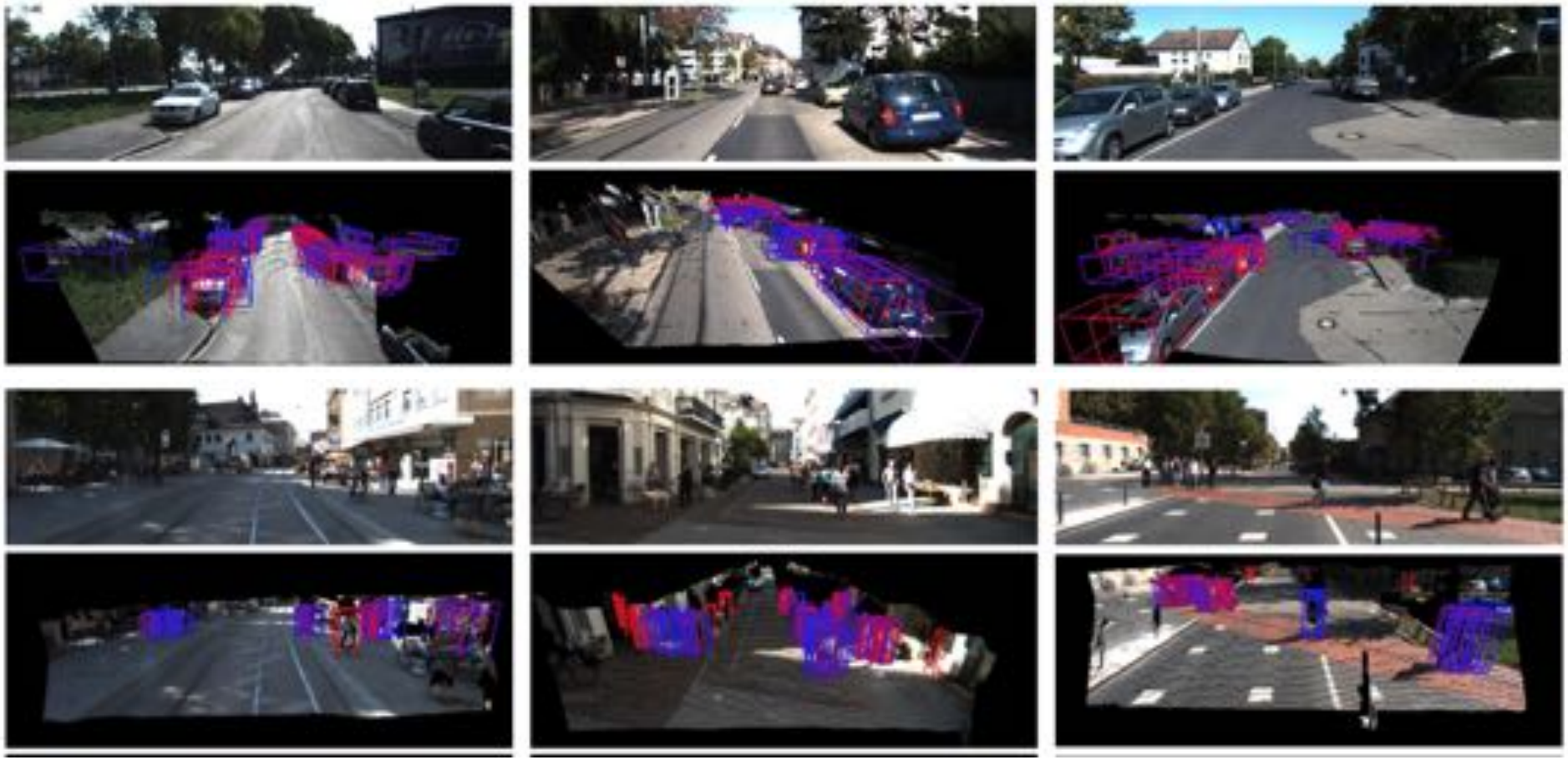
- That's it. Make a video for more fun

Can we Turn Fun into Useful?

- Can we use this in some more practical (useful) application?

Can we Turn Fun into Useful?

- Generate 3D boxes and score (classifier on some image features)



X. Chen, K. Kundu, Y. Zhu, A. Berneshawi, H. Ma, S. Fidler, R. Urtasun. 3D Object Proposals for Accurate Object Class Detection. NIPS'15

Can we Turn Fun into Useful?

Car

	Method	Setting	Code	Moderate	Easy	Hard	Runtime	Environment	Compare
1	DenseBox2			89.32 %	93.94 %	79.81 %	5 s	GPU @ 2.5 Ghz (C/C++)	
	L. Huang, Y. Yang, Y. Deng and Y. Yu: DenseBox: Unifying Landmark Localization with End to End Object Detection. ArXiv e-prints 2015.								
2	DJML			88.79 %	91.31 %	77.73 %	x s	GPU @ 1.5 Ghz (Matlab + C/C++)	
3	3DOP			88.64 %	93.04 %	79.10 %	3s	GPU @ 2.5 Ghz (Matlab + C/C++)	
	X. Chen, K. Kundu, Y. Zhu, A. Berneshawi, H. Ma, S. Fidler and R. Urtasun: 3D Object Proposals for Accurate Object Class Detection. NIPS 2015.								
4	SubCNN			88.16 %	90.89 %	77.06 %	2 s	GPU @ 2.5 Ghz (Python + C/C++)	
	Anonymous submission								
5	Mono3D			86.72 %	90.08 %	77.88 %	x s	GPU @ 2.5 Ghz (Matlab + C/C++)	
	Anonymous submission								
6	DenseBoxV2 L			84.92 %	88.77 %	70.00 %	0.04 s	GPU @ 2.0 Ghz (C/C++)	
	L. Huang, Y. Yang, Y. Deng and Y. Yu: DenseBox: Unifying Landmark Localization with End to End Object Detection. ArXiv e-prints 2015.								

Cyclist

	Method	Setting	Code	Moderate	Easy	Hard	Runtime	Environment	Compare
1	3DOP			68.94 %	78.39 %	61.37 %	3s	GPU @ 2.5 Ghz (Matlab + C/C++)	
	X. Chen, K. Kundu, Y. Zhu, A. Berneshawi, H. Ma, S. Fidler and R. Urtasun: 3D Object Proposals for Accurate Object Class Detection. NIPS 2015.								
2	Mono3D			63.17 %	76.82 %	55.68 %	x s	GPU @ 2.5 Ghz (Matlab + C/C++)	
	Anonymous submission								
3	SDP			61.87 %	73.59 %	54.29 %	1 s	GPU @ 2.5 Ghz (C/C++)	
	Anonymous submission								
4	SubCNN			59.70 %	74.15 %	53.08 %	2 s	GPU @ 2.5 Ghz (Python + C/C++)	
	Anonymous submission								
5	Regionlets			58.72 %	70.41 %	51.83 %	1 s	>8 cores @ 2.5 Ghz (C/C++)	
	X. Wang, M. Yang, S. Zhu and Y. Lin: Regionlets for Generic Object Detection. International Conference on Computer Vision 2013. C. Long, X. Wang, G. Hua, M. Yang and Y. Lin: Accurate Object Detection with Location Relaxation and Regionlets Relocalization. Asian Conference on Computer Vision 2014.								
6	MV-RGBD-RF			42.61 %	52.97 %	37.42 %	4 s	4 cores @ 2.5 Ghz (C/C++)	
	A. Gonzalez, G. Villalonga, J. Xu, D. Vazquez, J. Amores and A. Lopez: Multiview Random Forest of Local Experts Combining RGB and LIDAR data for Pedestrian Detection. IEEE Intelligent Vehicles Symposium (IV) 2015.								

X. Chen, K. Kundu, Y. Zhu, A. Berneshawi, H. Ma, S. Fidler, R. Urtasun. 3D Object Proposals for Accurate Object Class Detection. NIPS'15

A Little More on Camera Models