Camera Geometry

Computer Vision I (CSCI 3240U)

Faisal Z. Qureshi

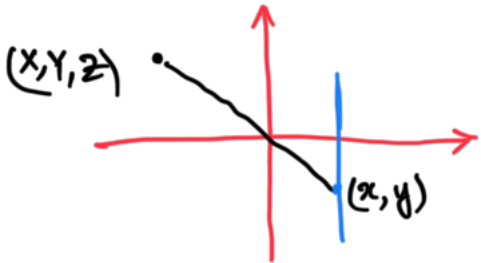
http://vclab.science.ontariotechu.ca

Week 2 at a glance

- **Lecture 1: Homogeneous coordinates and intrinsics (Sec. 2.1.4)**
  - Why projection is not a matrix, until we change coordinates
  - Homogeneous coordinates
  - The calibration matrix $\mathbf{K}$
  - Focal length in pixels; the principal point
- **Lecture 2: Extrinsics, the projection matrix, distortion (Sec. 2.1.5, 11.1)**
  - Rotation and translation
  - The full 3×4 matrix $\mathbf{P}$
  - Reading a real calibration file
  - Radial distortion; vanishing points

Reading: Szeliski Sec. 2.1.4, 2.1.5, 11.1. In-class exercises `w02a`, `w02b`, `w02c`. **Lab 2** uses all of this.

Lecture 1: Homogeneous coordinates and intrinsics



$$x = f\frac{X}{Z}$$

$$y = f\frac{Y}{Z}$$

The problem with last week's equation

Last week:

$$x = f\frac{X}{Z}, \qquad y = f\frac{Y}{Z}$$

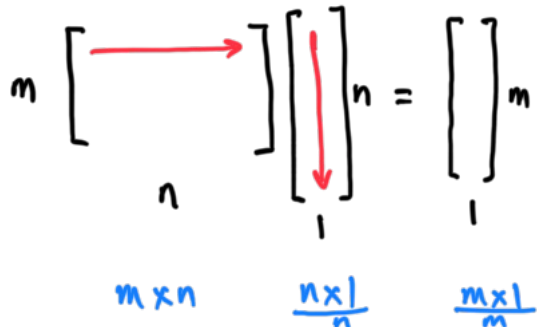
The problem with last week's equation

Last week:

$$x = f\frac{X}{Z}, \qquad y = f\frac{Y}{Z}$$

This is **not linear** in (X, Y, Z): there is a division.

That is inconvenient. We would like to write projection as a **matrix**, so we can compose it with rotations, translations and other transformations by multiplying matrices.



The problem with last week's equation

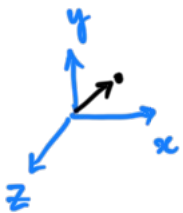
Last week:

$$x = f\frac{X}{Z}, \qquad y = f\frac{Y}{Z}$$

This is **not linear** in (X, Y, Z): there is a division.

That is inconvenient. We would like to write projection as a **matrix**, so we can compose it with rotations, translations and other transformations by multiplying matrices.

Homogeneous coordinates are the trick that makes this possible.



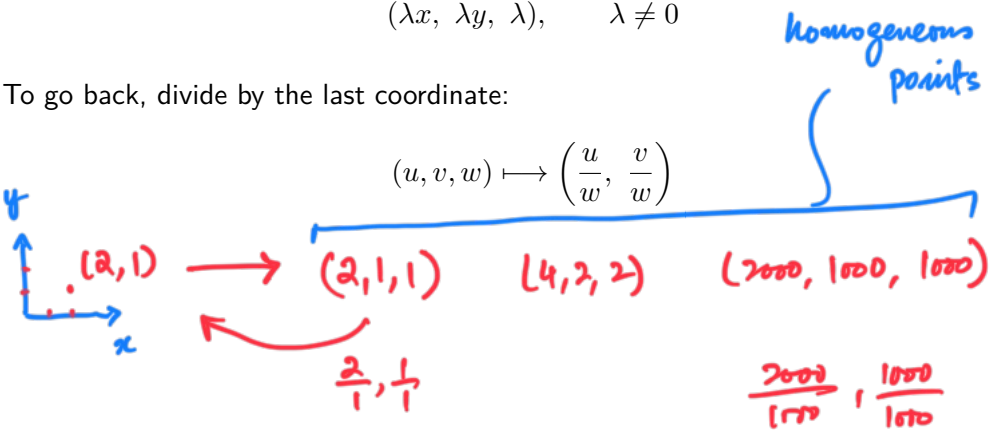
Homogeneous coordinates

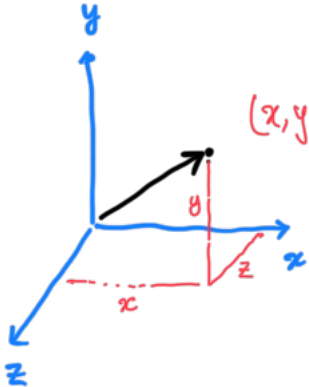
Represent a 2D point (x, y) by *any* triple

$$(\lambda x,\ \lambda y,\ \lambda), \qquad \lambda \neq 0$$

To go back, divide by the last coordinate:

$$(u, v, w) \longmapsto \left(\frac{u}{w},\ \frac{v}{w}\right)$$





$(x, y, z) \rightsquigarrow (\lambda x, \lambda y, \lambda z, \lambda)$

homogeneous

$(7, 6, 8) \rightsquigarrow (77, 66, 88, 11)$

Homogeneous coordinates

Represent a 2D point (x, y) by *any* triple

$$(\lambda x,\ \lambda y,\ \lambda), \qquad \lambda \neq 0$$

To go back, divide by the last coordinate:

$$(u, v, w) \longmapsto \left(\frac{u}{w},\ \frac{v}{w}\right)$$

- A point is now an **equivalence class**, not a single triple.
- $(2, 4, 2)$, $(1, 2, 1)$ and $(-3, -6, -3)$ are all the point $(1, 2)$.
- Everything is defined **up to scale**. Remember this; it is why $\mathbf{H}$ has 8 degrees of freedom, not 9 (Week 9).

What we gain

The division moves out of the equation and into the **interpretation**:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} \longrightarrow \begin{matrix} \frac{fX}{Z} \\ \frac{fY}{Z} \end{matrix}$$

homogeneous

Camera: $f = 7$

world point: $8, 3, 6$

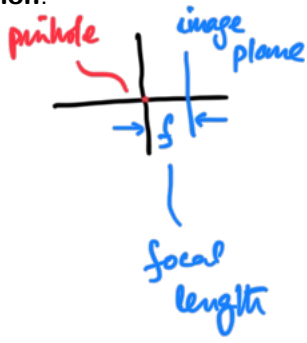
$$\begin{bmatrix} 7 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 8 \\ 3 \\ 6 \\ 1 \end{bmatrix} = \begin{bmatrix} 56 \\ 21 \\ 6 \end{bmatrix} \longrightarrow \left(\frac{56}{6}, \frac{21}{6} \right)$$

What we gain

The division moves out of the equation and into the **interpretation**:

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix}$$

$\lambda = 1$



and then

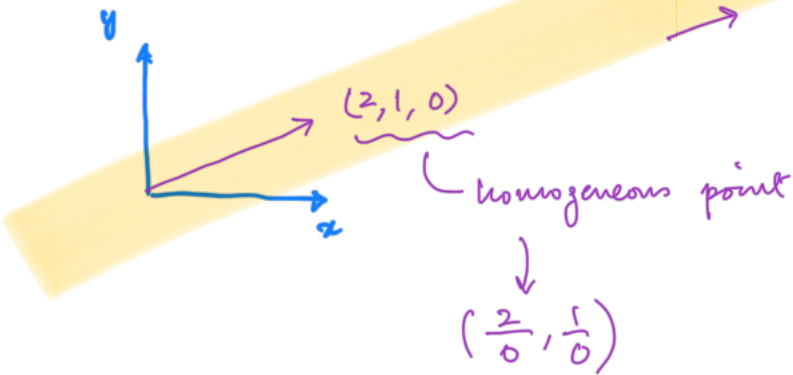
$$(x, y) = \left(\frac{x'}{z'}, \frac{y'}{z'} \right) = \left(\frac{fX}{Z}, \frac{fY}{Z} \right)$$

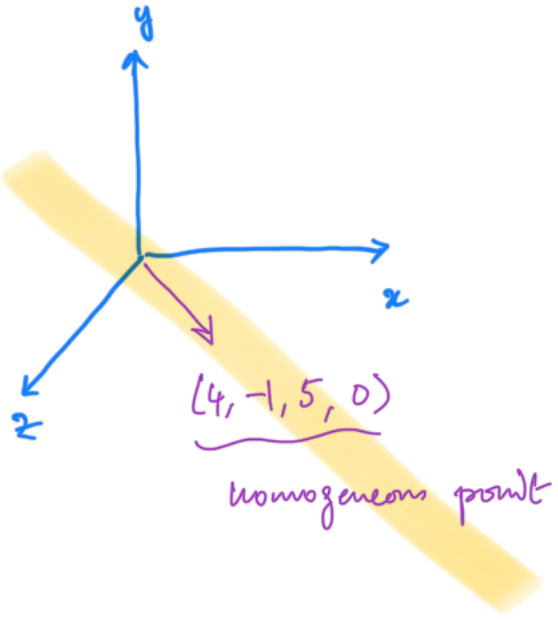
which is exactly last week's result.

Projection is now a matrix multiply followed by one divide.

Points at infinity

What does $w = 0$ mean? $(u, v, 0)$ cannot be divided back; it corresponds to a point infinitely far away in the direction (u, v).



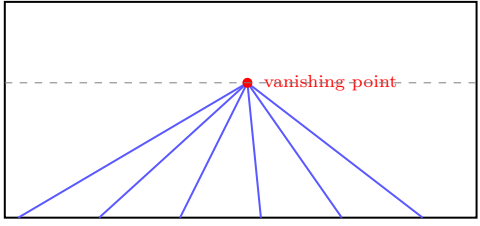

y
x
z
(4, -1, 5, 0)
homogeneous point

Points at infinity

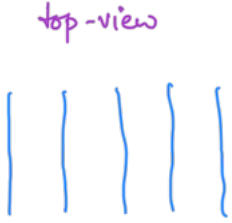
What does $w = 0$ mean? $(u, v, 0)$ cannot be divided back; it corresponds to a point infinitely far away in the direction (u, v).

These are **points at infinity**, or **ideal points**, and they are genuinely useful:

- A **direction** in the world is a point at infinity.
- Its image is the **vanishing point** of all lines with that direction.



parallel world lines converge in the image

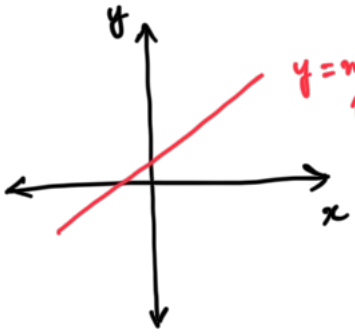


We use this at the end of the week, and again in Week 9.

Lines are also vectors

A line $ax + by + c = 0$ is the triple $\mathbf{l} = (a, b, c)$, again up to scale.

- Point $\mathbf{p}$ lies on line $\mathbf{l}$ iff $\mathbf{l} \cdot \mathbf{p} = 0$.
- The line **through** two points: $\mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$.
- The point **where** two lines meet: $\mathbf{p} = \mathbf{l}_1 \times \mathbf{l}_2$.

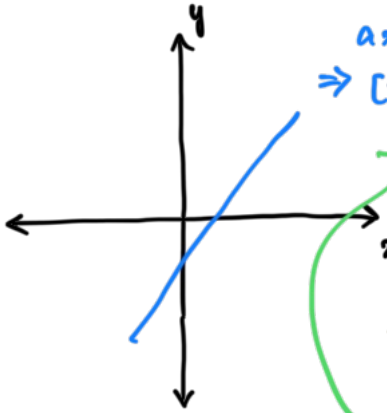


$y = mx + b$

co-efficients

$y = \begin{bmatrix} x & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix}$

$1 \times 2 \quad 2 \times 1$



$ax + by + c = 0$

$\Rightarrow \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = 0$

Co-efficient of a line: a, b, c

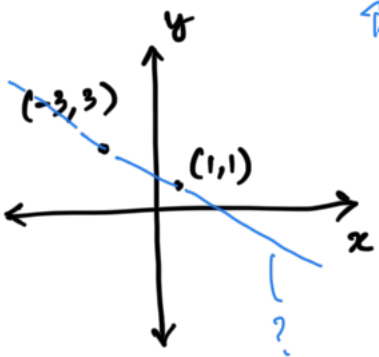
dot product $= 0$

$\therefore (x, y, 1) \perp (a, b, c)$

$\vec{p} \cdot \vec{l} = \phi$

$\vec{p} = (x, y, 1)$

$\vec{l} = (a, b, c)$



Typically: $\frac{y-y_1}{y_2-y_1} = \frac{x-x_1}{x_2-x_1}$

$(x_1, y_1) = (-3, 3)$

$(x_2, y_2) = (1, 1)$

$(-3, 3, 1) \times (10, 10, 10) = (a, b, c)$

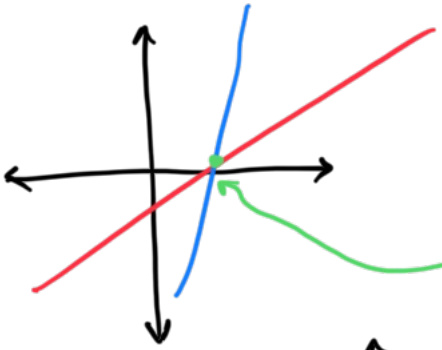
$$\begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ -3 & 3 & 1 \\ 10 & 10 & 10 \end{vmatrix} = \hat{i}(\quad) - \hat{j}(\quad) + \hat{k}(\quad)$$

a b c

$$(a, b, c) \times (d, e, f)$$

$$= \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ a & b & c \\ d & e & f \end{vmatrix} = \hat{i}(bf - ec) - \hat{j}(af - dc) + \hat{k}(ae - db)$$

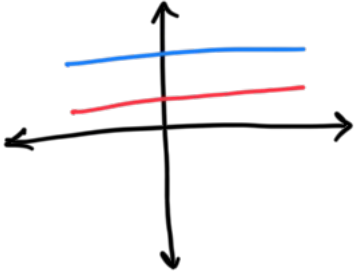
$$= (bf - ac,\ dc - af,\ ae - db)$$



$(a_1, b_1, c_1) \times (a_2, b_2, c_2) =$ _____

x, y, w

$\left(\frac{x}{w}, \frac{y}{w}\right)$



Lines are also vectors

A line $ax + by + c = 0$ is the triple $\mathbf{l} = (a, b, c)$, again up to scale.

- Point $\mathbf{p}$ lies on line $\mathbf{l}$ iff $\mathbf{l} \cdot \mathbf{p} = 0$.
- The line **through** two points: $\mathbf{l} = \mathbf{p}_1 \times \mathbf{p}_2$.
- The point **where** two lines meet: $\mathbf{p} = \mathbf{l}_1 \times \mathbf{l}_2$.

Two remarks worth the time:

- The last two rules are the *same operation*. Points and lines are interchangeable in 2D: this is **duality**.
- Parallel lines meet at a point with $w = 0$: a point at infinity. No special case needed.

This is in-class exercise `w02b`.

Parallel lines meet at a point at infinity

Take two parallel lines with slope 3:

$$\mathbf{l}_1 : 3x - y + 7 = 0, \qquad \mathbf{l}_2 : 3x - y - 3 = 0$$

so $\mathbf{l}_1 = (3, -1, 7)$ and $\mathbf{l}_2 = (3, -1, -3)$.

Parallel lines meet at a point at infinity

Take two parallel lines with slope 3:

$$\mathbf{l}_1 : 3x - y + 7 = 0, \qquad \mathbf{l}_2 : 3x - y - 3 = 0$$

so $\mathbf{l}_1 = (3, -1, 7)$ and $\mathbf{l}_2 = (3, -1, -3)$.

Their meeting point is the cross product:

$$\mathbf{l}_1 \times \mathbf{l}_2 = \begin{pmatrix} (-1)(-3) - (7)(-1) \\ (7)(3) - (3)(-3) \\ (3)(-1) - (-1)(3) \end{pmatrix} = \begin{pmatrix} 10 \\ 30 \\ 0 \end{pmatrix}$$

Parallel lines meet at a point at infinity

Take two parallel lines with slope 3:

$$\mathbf{l}_1 : 3x - y + 7 = 0, \qquad \mathbf{l}_2 : 3x - y - 3 = 0$$

so $\mathbf{l}_1 = (3, -1, 7)$ and $\mathbf{l}_2 = (3, -1, -3)$.

Their meeting point is the cross product:

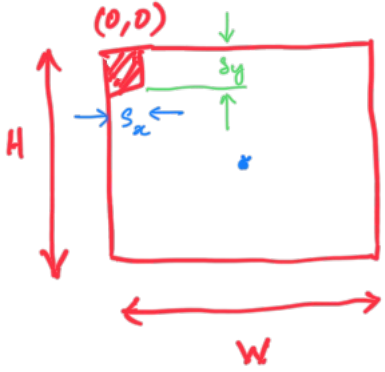
$$\mathbf{l}_1 \times \mathbf{l}_2 = \begin{pmatrix} (-1)(-3) - (7)(-1) \\ (7)(3) - (3)(-3) \\ (3)(-1) - (-1)(3) \end{pmatrix} = \begin{pmatrix} 10 \\ 30 \\ 0 \end{pmatrix}$$

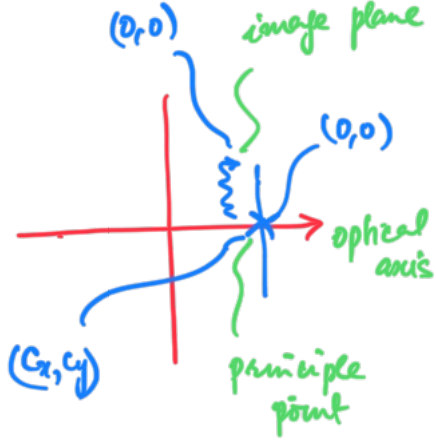
$w = 0$, so we cannot divide back to a finite (x, y). The **direction** is $(10, 30) \propto (1, 3)$, which is exactly the direction both lines run in.

Parallel lines meet at the point at infinity in their shared direction. No special case, no exception: the cross product just returns $w = 0$ and we read the direction off.

From metres to pixels

So far x and y are in the same units as f (millimetres, say). Sensors are indexed in **pixels**. Three corrections:



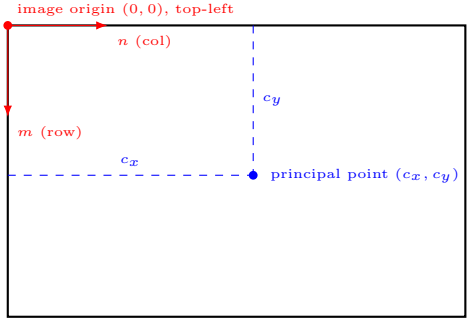


From metres to pixels

So far x and y are in the same units as f (millimetres, say). Sensors are indexed in **pixels**. Three corrections:

1. **Scale.** Pixels have a physical size. If a pixel is s_x wide and s_y tall, then $\alpha_x = f/s_x$ and $\alpha_y = f/s_y$ are focal lengths *in pixels*.
2. **Origin.** Image indices start at a corner, not at the optical axis. Add the principal point (c_x, c_y).
3. **Skew.** A term s for non-perpendicular axes. In practice **zero**.

Where the image origin sits



KITTI: 1242×375, principal point $(609.6,\ 172.9)$, near but not at the centre

The calibration matrix

Collecting those into one upper-triangular matrix:

$$\mathbf{K} = \begin{pmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

intrinsic matrix

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \rightsquigarrow \begin{bmatrix} \alpha_x & s & c_x & 0 \\ 0 & \alpha_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

The calibration matrix

Collecting those into one upper-triangular matrix:

$$\mathbf{K} = \begin{pmatrix} \alpha_x & s & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

- α_x, α_y: focal length in pixels. Equal for square pixels.
- (c_x, c_y): principal point, near but rarely exactly the image centre.
- s: skew. Assume 0 unless you have a reason not to.

These are the **intrinsic** parameters: everything internal to the camera.

Sanity checks on $\mathbf{K}$

When you parse a real calibration file, check:

- $\alpha_x \approx \alpha_y$: if not, either the pixels are not square or you have mis-parsed.
- $s = 0$.
- (c_x, c_y) near the image centre. *Near*, not equal; cropping after rectification shifts it.

Sanity checks on $\mathbf{K}$

When you parse a real calibration file, check:

- $\alpha_x \approx \alpha_y$: if not, either the pixels are not square or you have mis-parsed.
- $s = 0$.
- (c_x, c_y) near the image centre. *Near*, not equal; cropping after rectification shifts it.

Field of view follows directly:

$$\mathrm{FOV}_x = 2 \arctan\left(\frac{W}{2\alpha_x}\right)$$

A number you can check against what the pictures look like.

$\theta = \tan^{-1}(y/x)$

In-class exercise: w02b

Show that the line through two 2D points (x_1, y_1) and (x_2, y_2) is given by

$$(x_1, y_1, 1) \times (x_2, y_2, 1)$$

and use it to compute the line through $(1, 1)$ and $(2, 4)$.

In-class exercise: w02b

Show that the line through two 2D points (x_1, y_1) and (x_2, y_2) is given by

$$(x_1, y_1, 1) \times (x_2, y_2, 1)$$

and use it to compute the line through $(1, 1)$ and $(2, 4)$.

Hint: the cross product is perpendicular to both arguments, so $\mathbf{l} \cdot \mathbf{p}_1 = \mathbf{l} \cdot \mathbf{p}_2 = 0$, which is precisely the statement that both points lie on the line.

Always substitute your two points back into $ax + by + c = 0$ to check.

Lecture 1 recap

- Homogeneous coordinates turn projection into a matrix multiply plus a divide.
- Everything is defined **up to scale**.
- $w = 0$ gives points at infinity: directions, whose images are vanishing points.
- Points and lines are dual; the cross product does both jobs.
- $\mathbf{K}$ holds focal length in pixels, principal point, and skew.

Next: where the camera *is*.

Lecture 2: Extrinsics, $\mathbf{P}$, and distortion

The camera is not at the origin

So far the camera sits at the world origin looking along $+Z$. In general it has a **pose**: a rotation $\mathbf{R}$ and a position $\mathbf{o}$ in world coordinates.

A world point $\mathbf{P}_w$ expressed in camera coordinates is

$$\mathbf{P}_c = \mathbf{R}\,(\mathbf{P}_w - \mathbf{o})$$

The camera is not at the origin

So far the camera sits at the world origin looking along $+Z$. In general it has a **pose**: a rotation $\mathbf{R}$ and a position $\mathbf{o}$ in world coordinates.

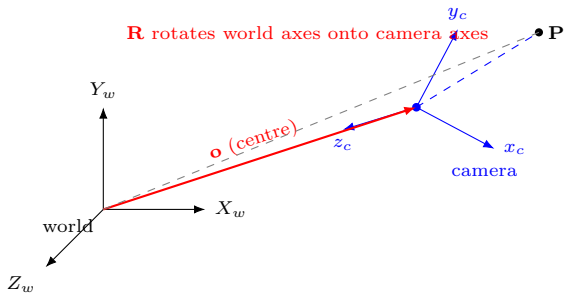
A world point $\mathbf{P}_w$ expressed in camera coordinates is

$$\mathbf{P}_c = \mathbf{R}\,(\mathbf{P}_w - \mathbf{o})$$

- The **rows of** $\mathbf{R}$ are the camera's x, y, z axes written in world coordinates.
- Equivalently $\mathbf{P}_c = \mathbf{R}\mathbf{P}_w + \mathbf{t}$ with $\mathbf{t} = -\mathbf{R}\mathbf{o}$.

Both forms appear in the literature. Always state which you are using. Mixing them is the single most common error in this material.

World and camera frames



The full projection matrix

Compose pose with intrinsics:

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

a 3×4 matrix taking homogeneous world points to homogeneous image points:

$$\tilde{\mathbf{p}} = \mathbf{P} \, \tilde{\mathbf{P}}_w$$

The full projection matrix

Compose pose with intrinsics:

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

a 3×4 matrix taking homogeneous world points to homogeneous image points:

$$\tilde{\mathbf{p}} = \mathbf{P} \, \tilde{\mathbf{P}}_w$$

Degrees of freedom:

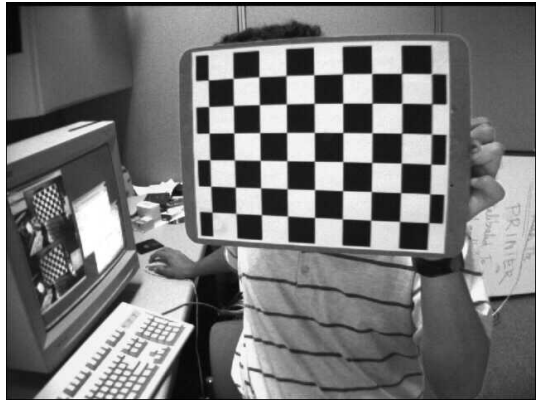
- Intrinsics: 5 (or 4 with zero skew)
- Extrinsics: 6 (3 rotation, 3 translation)
- Total: **11**, matching a 3×4 matrix up to scale.

How do you get $\mathbf{K}$ in the first place?

Calibrate the camera against an object of known geometry, usually a printed **checkerboard**.

The recipe in OpenCV:

1. Print a checkerboard, glue it to a rigid board.
2. Take 10 to 20 photos from different angles.
3. `cv.findChessboardCorners`, then `cv.cornerSubPix` for sub-pixel accuracy.
4. `cv.calibrateCamera` returns $\mathbf{K}$, distortion coefficients, and one pose per photo.



Rectified datasets like KITTI ship the result, so we do not have to redo it.

Reading a real calibration file

KITTI ships one calibration file per image. It contains, among others,

$$\mathbf{P}_2 = \begin{bmatrix} \mathbf{K} & \mathbf{K}\mathbf{t} \end{bmatrix}$$

for the left colour camera.

To recover the pieces:

- $\mathbf{K}$ is the leftmost 3×3 block.
- $\alpha_x = \mathbf{P}_2[0,0]$, and $(c_x, c_y) = (\mathbf{P}_2[0,2],\ \mathbf{P}_2[1,2])$.
- The horizontal offset of the camera is $\mathbf{P}_2[0,3]/\alpha_x$.

You will do exactly this in Lab 2, and again in Lab 9 to get the stereo baseline.

Reading a real calibration file

$$\mathbf{P}_2 = \begin{bmatrix} \mathbf{K} & \mathbf{K}\mathbf{t} \end{bmatrix} \text{ is the special case of } \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \text{ when } \mathbf{R} = \mathbf{I}.$$

KITTI ships rectified stereo. Rectification chooses a common camera frame for all four cameras ($\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$), so every camera's rotation relative to that frame is the identity by construction. Each camera differs from the reference only by a horizontal translation t.

What KITTI's numbers look like

For the road benchmark, every frame shares:

Quantity	Value
$\alpha_x = \alpha_y$	721.5 px
(c_x, c_y)	$(609.6,\ 172.9)$ px
skew	0
image size	1242×375

- Square pixels, zero skew: the file passes our sanity checks.
- $c_x = 609.6$ against a half-width of 621: close, but **not** the centre. The images are cropped after rectification.
- $\mathrm{FOV}_x = 2\arctan(1242/(2 \cdot 721.5)) \approx 81°$.

Radial distortion

Real lenses are not thin, and projection is not exactly perspective. The dominant deviation is **radial**, since displacement depends only on distance from the optical axis:

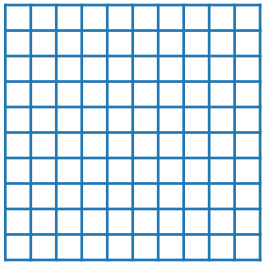
$$x_d = x\,(1 + \kappa_1 r^2 + \kappa_2 r^4 + \cdots) \qquad r^2 = x^2 + y^2$$
$$y_d = y\,(1 + \kappa_1 r^2 + \kappa_2 r^4 + \cdots)$$

- $\kappa_1 < 0$: **barrel** distortion (wide angle).
- $\kappa_1 > 0$: **pincushion**.

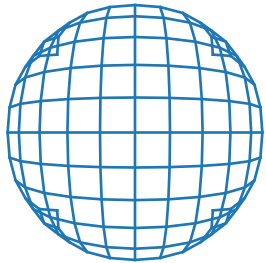
Straight lines become curved, so *undistort first*, then do geometry. Rectified datasets such as KITTI have this already removed, which is why we can treat them as ideal pinholes.

Barrel and pincushion

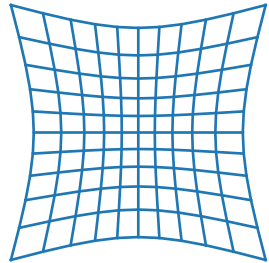
no distortion



barrel $(\kappa_1 < 0)$



pincushion $(\kappa_1 > 0)$



Projecting homogeneous world points

Full projection of a homogeneous world point $(\mathbf{X}, W)$:

$$\mathbf{v} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] \begin{pmatrix} \mathbf{X} \\ W \end{pmatrix}$$

A point at infinity in direction $\mathbf{d}$ has $W = 0$:

$$\mathbf{v} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] \begin{pmatrix} \mathbf{d} \\ 0 \end{pmatrix} = \mathbf{K}(\mathbf{R}\mathbf{d} + \mathbf{t} \cdot 0) = \mathbf{K}\mathbf{R}\mathbf{d}$$

- $\mathbf{R}\mathbf{d}$: rotate the world direction into the camera frame.
- $\mathbf{K}(\mathbf{R}\mathbf{d})$: turn that camera-frame direction into an image point (homogeneous, so divide by the third entry).
- $\mathbf{t}$ drops out because $W = 0$. The vanishing point depends on the camera's orientation, not its position, and parallel lines share it.

Vanishing points, properly

Take a world direction $\mathbf{d}$ and follow points $\mathbf{P} + t\mathbf{d}$ as $t \to \infty$. In homogeneous coordinates the limit is the point at infinity $(\mathbf{d}, 0)$, whose image is

$$\mathbf{v} = \mathbf{K}\mathbf{R}\,\mathbf{d}$$

Special case: the direction **along the optical axis**, $\mathbf{d} = (0, 0, 1)$ in camera coordinates. Then

$$\mathbf{v} = \mathbf{K}(0, 0, 1)^\top = (c_x,\ c_y)$$

The vanishing point of lines parallel to the optical axis is the principal point.

Why that matters

On a straight road with the camera looking along it:

- The lane markings and road edges are parallel in the world.
- Their images converge at the principal point.

In **Lab 2** you will:

1. predict the vanishing point from the calibration, and
2. measure it from the image by intersecting two fitted lines,

and compare. Expect 10 to 30 px of disagreement; the prediction is exact, but the measurement depends on your picked points and on how straight the road is.

In **Lab 6** the vanishing point becomes a *constraint*: a candidate road boundary that misses it can be rejected.

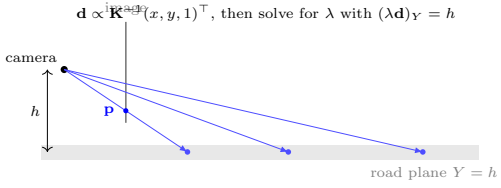
Going backwards: pixel to ray

Projection loses depth, but $\mathbf{K}$ is invertible. Given a pixel $\mathbf{p}$,

$$\mathbf{d} \propto \mathbf{K}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

is the direction of the ray in camera coordinates.

We still cannot say **where** along that ray the point lies, unless we add a constraint. If the point is known to lie on the road plane $Y = h$, solve for the scalar λ with $(\lambda \mathbf{d})_Y = h$.



Why $\mathbf{K}^{-1}(x, y, 1)^\top$ is the ray direction

For a camera at the origin looking along $+Z$, a point (X, Y, Z) in camera coordinates projects to

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \propto \mathbf{K} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

Invert:

$$\mathbf{K}^{-1} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \propto \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

This is the point itself, up to an unknown scale. Any point of the form $\lambda \mathbf{d}$ with $\lambda > 0$ projects to the same pixel. That family of points is the **viewing ray**: the camera cannot tell them apart.

Fixing the scale with a plane

The ray is $\lambda \mathbf{d}$ with $\mathbf{d} = \mathbf{K}^{-1}(x, y, 1)^\top$ and λ unknown.

On the road plane $Y = h$ (with h the camera height above the road):

$$(\lambda \mathbf{d})_Y = h \;\Rightarrow\; \lambda = h / d_Y$$

and the 3D point is $(h/d_Y)\,\mathbf{d}$. For KITTI, $h \approx 1.60$ m from `Tr_cam_to_road`.

Two cautions:

- The recipe assumes the point *is* on the road. A pixel on a car returns the ground beneath the car, not the car itself.
- $d_Y \to 0$ near the horizon: the ray runs parallel to the road, λ blows up. Mask the horizon before dividing.

Going backwards: pixel to ray

That single extra fact turns any road pixel into metres. It is Part 4 of Lab 2.

The $\mathbf{K}$ here is the 3×3 intrinsic

Two things called the *projection matrix*:

- $\mathbf{K}$ (3×3), the **intrinsic** matrix. Camera-frame 3-vector to pixel.
- $\mathbf{P} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]$ (3×4), the **full** projection. World-frame homogeneous 4-vector to pixel.

The ray derivation uses the first. $\mathbf{R}$ and $\mathbf{t}$ are missing because the setup is **in camera coordinates**, so $\mathbf{R} = \mathbf{I}$ and $\mathbf{t} = \mathbf{0}$:

$$\mathbf{P} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \mathbf{K}[\mathbf{I} \mid \mathbf{0}] \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \mathbf{K} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}.$$

The homogeneous 1 is multiplied by the zero column and drops. The viewing ray then comes out in camera coordinates for free.

Worked example: setup

KITTI: $\alpha_x = \alpha_y = 721.5$, $(c_x, c_y) = (609.6, 172.9)$, $h \approx 1.60$ m.

The intrinsic matrix and its inverse:

$$\mathbf{K} = \begin{pmatrix} \alpha_x & 0 & c_x \\ 0 & \alpha_y & c_y \\ 0 & 0 & 1 \end{pmatrix}, \qquad \mathbf{K}^{-1} = \begin{pmatrix} 1/\alpha_x & 0 & -c_x/\alpha_x \\ 0 & 1/\alpha_y & -c_y/\alpha_y \\ 0 & 0 & 1 \end{pmatrix}$$

Check: multiply and use $\alpha_x \cdot (-c_x/\alpha_x) + c_x = 0$; the diagonal becomes 1 and the last column collapses to $(0, 0, 1)^\top$.

Worked example: pixel to ray

Pick a pixel at $(x, y) = (621, 300)$: image centre horizontally, 127 rows below the horizon.

$$\mathbf{d} = \mathbf{K}^{-1} \begin{pmatrix} 621 \\ 300 \\ 1 \end{pmatrix} = \begin{pmatrix} (621 - 609.6)/721.5 \\ (300 - 172.9)/721.5 \\ 1 \end{pmatrix} \approx \begin{pmatrix} 0.016 \\ 0.176 \\ 1 \end{pmatrix}.$$

Scale for the road plane: $\lambda = h/d_Y = 1.60/0.176 \approx 9.09$.

3D point $\lambda\mathbf{d} \approx (0.14,\ 1.60,\ 9.09)$ m: **9.1 m ahead**, 14 cm right, and 1.60 m below camera height (on the road, as required).

The columns and rows of $\mathbf{P}$ have a meaning

Write $\mathbf{P} = [\mathbf{p}_1\ \mathbf{p}_2\ \mathbf{p}_3\ \mathbf{p}_4]$ by columns. Apply it to each world axis at infinity in turn:

$$\mathbf{P}\,(1,0,0,0)^\top = \mathbf{p}_1, \quad \mathbf{P}\,(0,1,0,0)^\top = \mathbf{p}_2, \quad \mathbf{P}\,(0,0,1,0)^\top = \mathbf{p}_3$$

and to the world origin,

$$\mathbf{P}\,(0,0,0,1)^\top = \mathbf{p}_4.$$

- $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ are the **vanishing points of the world X, Y, Z axes**.
- $\mathbf{p}_4$ is the **image of the world origin**.

And the **rows** matter too. Write $\mathbf{P} = [\mathbf{r}_1;\ \mathbf{r}_2;\ \mathbf{r}_3]^\top$. The third row $\mathbf{r}_3$ defines the **principal plane** of the camera: the plane through the camera centre, parallel to the image plane. World points with $\mathbf{r}_3 \cdot \tilde{\mathbf{P}}_w = 0$ project to $w = 0$, that is, to infinity.

Handy for sanity checks. If your recovered $\mathbf{P}$ projects the world Z-axis to something far from the principal point, and the axes should be aligned, something is off.

In-class exercises this week

- `w02a`: given $\mathbf{R}$, the camera position, and f, project a world point into image coordinates. *State your convention for $\mathbf{R}$ and $\mathbf{t}$, and check the sign of Z_c: a negative value means the point is behind the camera.*
- `w02b`: lines through points via the cross product.
- `w02c`: recover a rotation angle from a single correspondence. *Use `atan2`, not `arctan`, or you lose the quadrant.*

Week 2 recap

- Homogeneous coordinates make projection linear, at the cost of working up to scale.
- $\mathbf{K}$: focal length in pixels, principal point, skew.
- $\mathbf{P} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]$, 11 degrees of freedom.
- Real calibration files are readable, and worth sanity-checking.
- Radial distortion is the main departure from the pinhole model.
- The vanishing point of the optical-axis direction is the principal point.
- $\mathbf{K}^{-1}$ gives a ray; a known plane turns that ray into a point.

This week's work

- **Reading:** Szeliski Sec. 2.1.4, 2.1.5, 11.1.
- **In-class exercises:** `w02a`, `w02b`, `w02c`.
- **Lab 2:** camera geometry on KITTI. Parse a real $\mathbf{P}_2$, project a metric grid onto the road, predict *and* measure the vanishing point, then back-project pixels to metres.

Next week: we leave geometry alone for a while and start processing the image itself: convolution and linear filtering.

Copyright and License

©Faisal Z. Qureshi

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.