Image Formation - Fundamentals

Dr. Sander Ali Khowaja,
Assistant Professor, Department of Telecommunication Engineering
Faculty of Engineering and Technology, University of Sindh, Pakistan
Senior Member, IEEE – Member, ACM
https://sander-ali.github.io

Computer Vision & Image Processing

Computer Vision in the News

The Biggest News in Computer Vision in 2022

By Datagen, December 19, 2022

SHARE

2022 has been an incredibly eventful year for the world of computer vision and synthetic data. Here is a roundup of the most exciting developments from this year.

Generative AI models steal the spotlight

These generative models have impressed the world with their ability to generate high-resolution and high-fidelity images from text prompts. 2022 saw the release of multiple powerful text-to-image models that outperformed their predecessors.

Mid-2022, Open AI released DALL-E 2. It can make realistic edits to existing images, replicate the style of other images, and even expand a single image. Midjourney, another text-to-image model, released its open beta in 2022. Google's Imagen was also released this year.

Stability AI released its brainchild Stable Diffusion 1.0 in 2022, and even updated it to Stable Diffusion 2.0 in November. Based on a novel latent diffusion model, Stable Diffusion performs competitively with the state-of-the-art for various image generation tasks with a smaller computational footprint. Its developers made the code and model public, a marked departure from its competitors DALL-E 2 and Midjourney.

"A photograph of an astronaut riding a horse" by DALL-E 2 (left) and Stable Diffusion (right)

https://datagen.tech/blog/2022-computer-vision-year-in-review/

Biotech & Health

Guide dogs will be giving the side-eye to self-driving car tech coming for their jobs

Haje Jan Kamps @Haje / 6:39 PM GMT+5 • January 7, 2023

Comment

Image Credits: Guidi

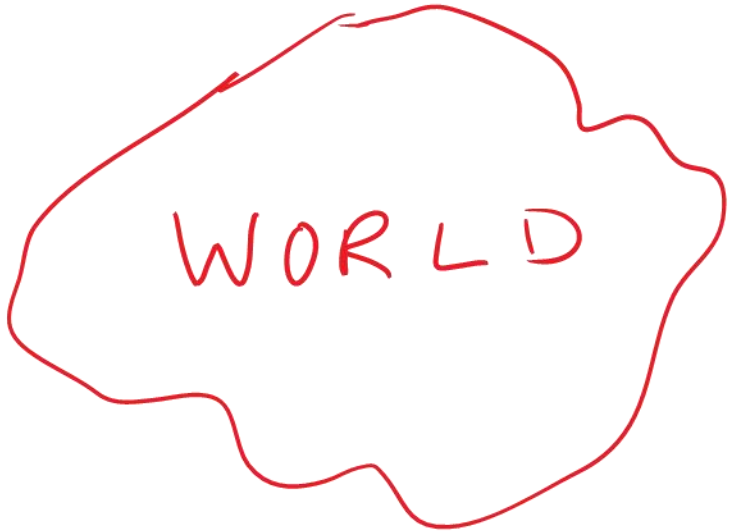
The visually impaired are getting a helping hand (or a helping belt, as it were) from Korean startup AI Guided. At CES in Las Vegas, the company was showing off some pretty neat tech that incorporates optical and lidar technology along with AI-powered on-device computing to identify obstacles and help with navigation.

The company claims to be able to do advanced object identification to help keep walkers safe, in addition to using gentle haptic feedback to help with wayfinding. The whole system is carried on a belt, leaving the users hands free.

AI-Guided's upcoming product Guidi, shown off at CES 2023. Image Credits: Haje Kamps/TechCrunch

https://techcrunch.com/2023/01/07/ai-guided/

A camera creates an image …

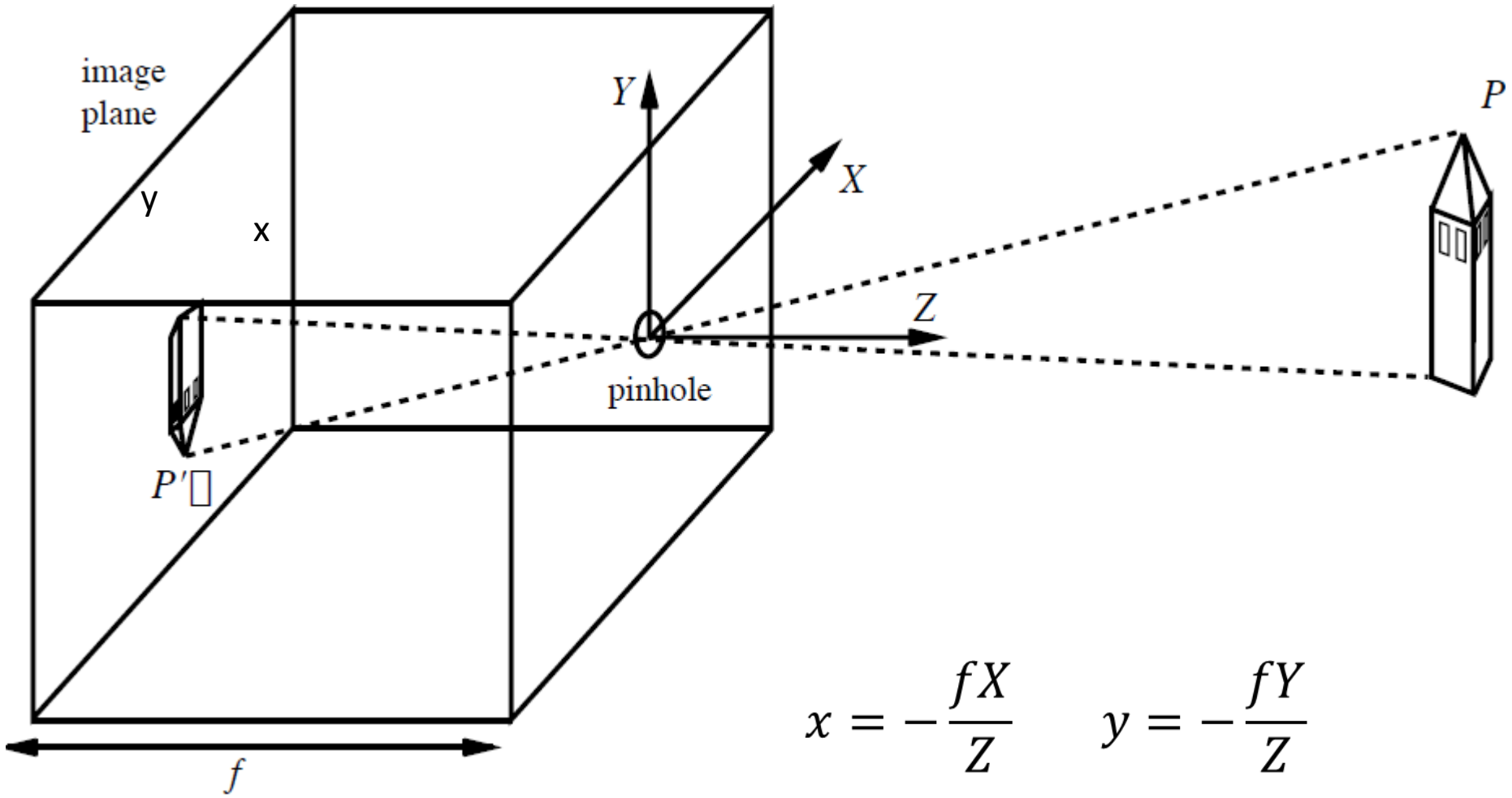


The image I(x,y) measures how much light is captured at pixel (x,y)

We want to know

- Where does a point (X,Y,Z) in the world get imaged?
- What is the brightness at the resulting point (x,y)?

Pinhole Camera

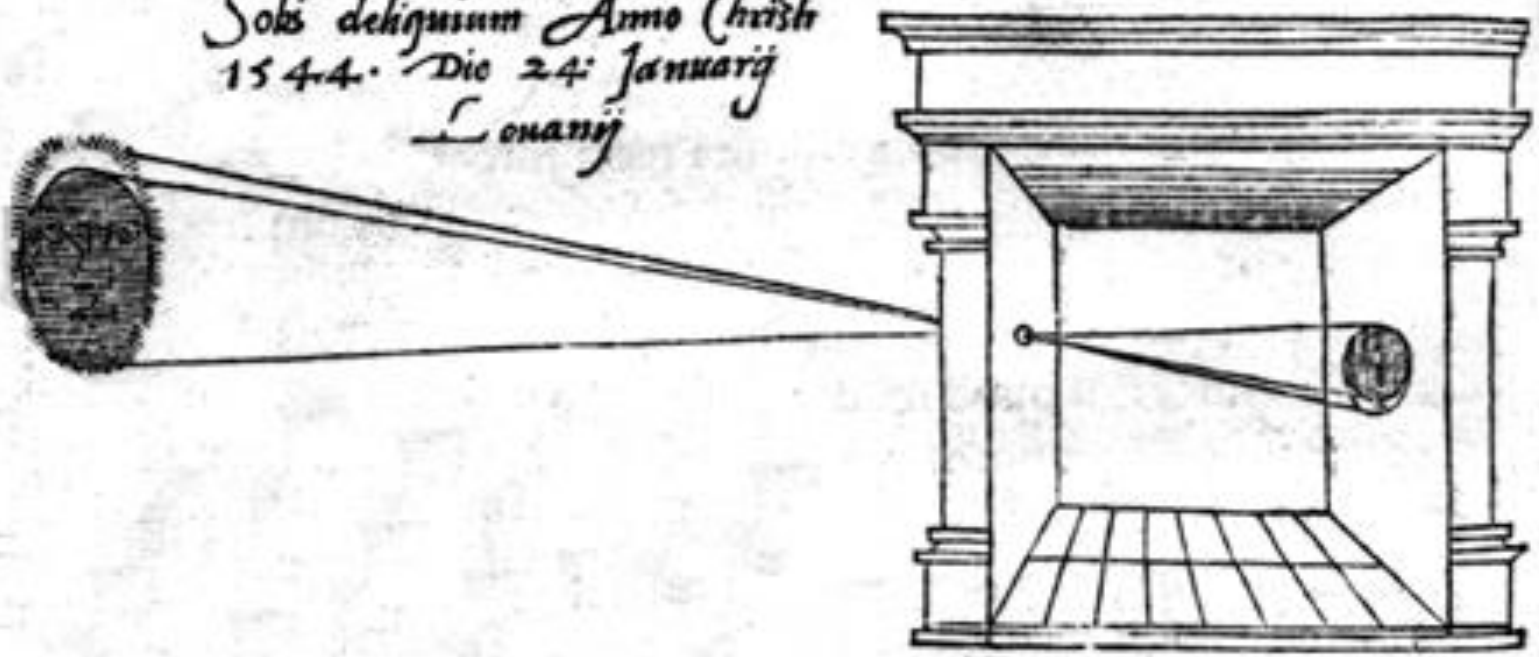


$$x = -\frac{fX}{Z} \qquad y = -\frac{fY}{Z}$$

Camera Obscura (Reinerus Gemma-Frisius, 1544)

illum in tabula per radios Solis, quàm in cœlo contingit: hoc eſt, ſi in cœlo ſuperior pars deliquiũ patiatur, in radiis apparebit inferior deficere, vt ratio exigit optica.

Solis deliquium Anno Christi 1544. Die 24. Januarij Louanij



Sic nos exactè Anno .1544. Louanii eclipſim Solis obſeruauimus, inuenimusq; deficere paulò plus q̃ dex-

Lets recall similar triangles concept

This diagram is for the special case of a point P in the Y-Z plane.
In the general case, consider the projection of P on the Y-Z plane.

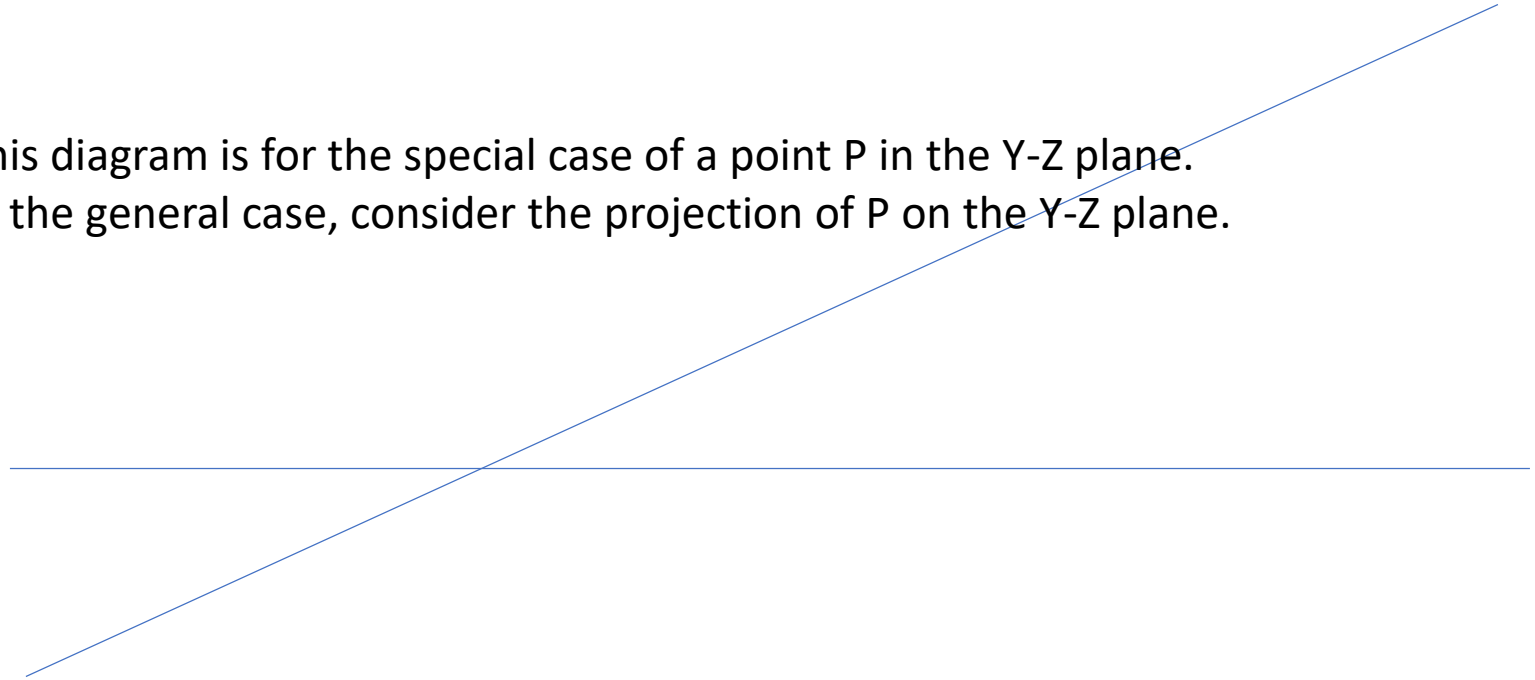
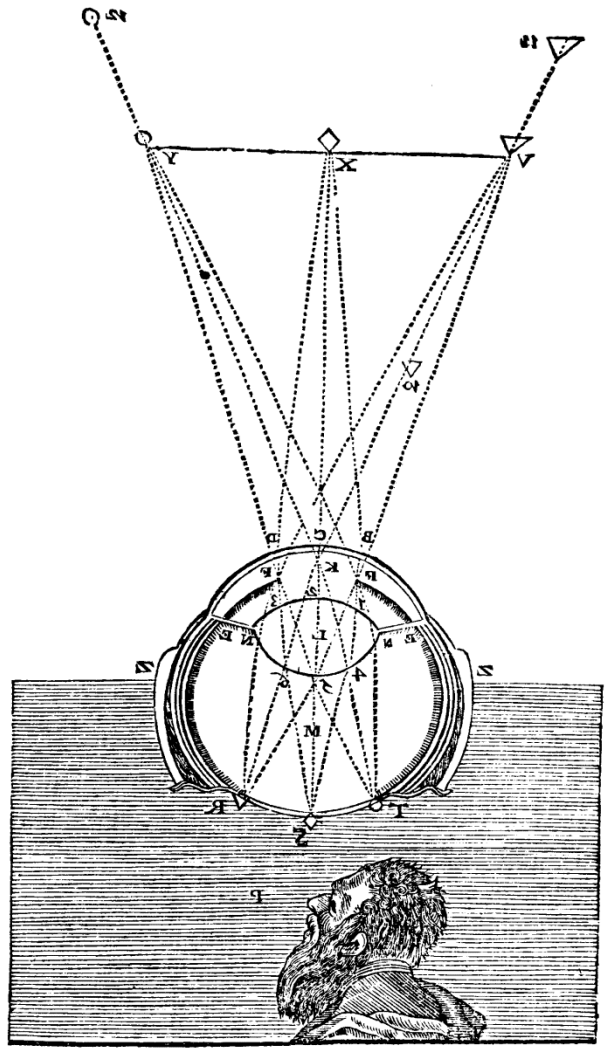


Image is inverted



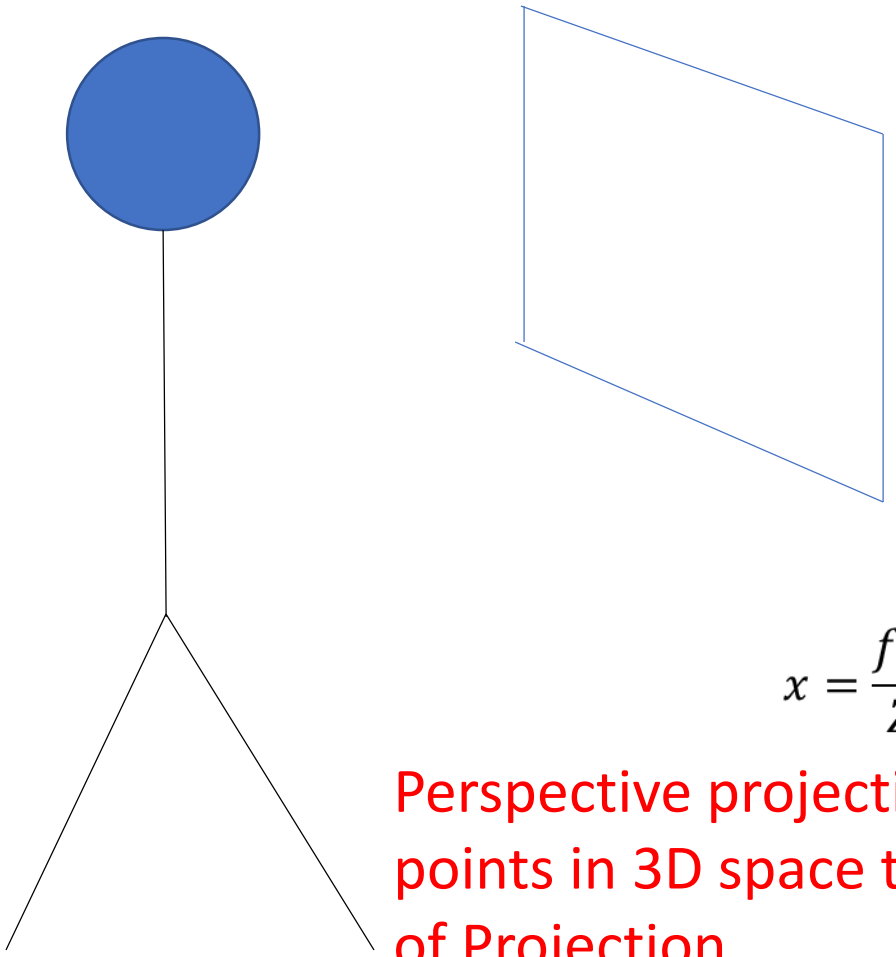
From Descartes(1637), La Dioptrique

This was pointed out by Kepler in 1604

But this is no big deal. The brain can interpret it the right way. And for a camera, software can simply flip the image top-down and right-left. After this trick, we get

$$x = \frac{fX}{Z} \qquad y = \frac{fY}{Z}$$

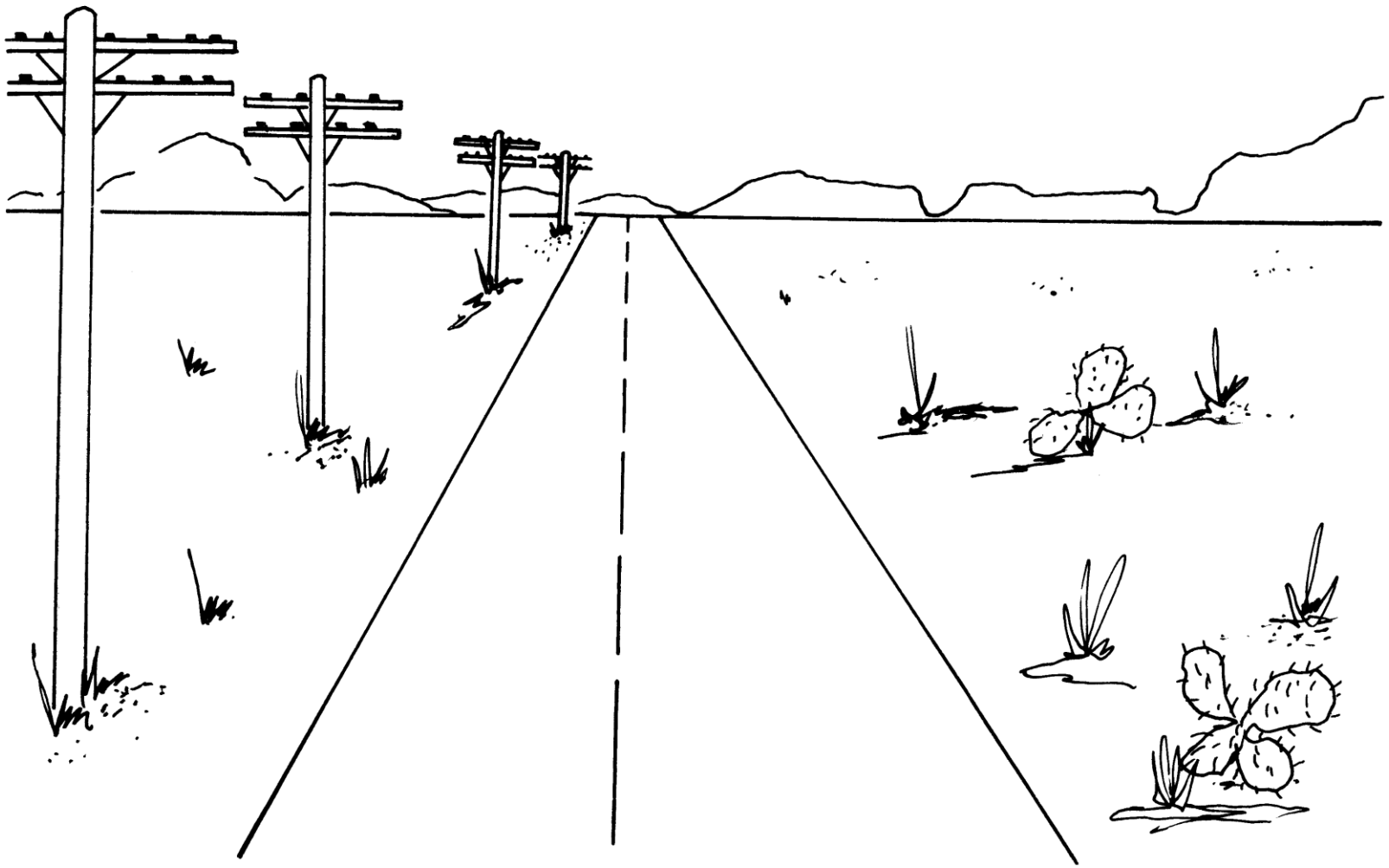
A Projection Model that avoids inversion



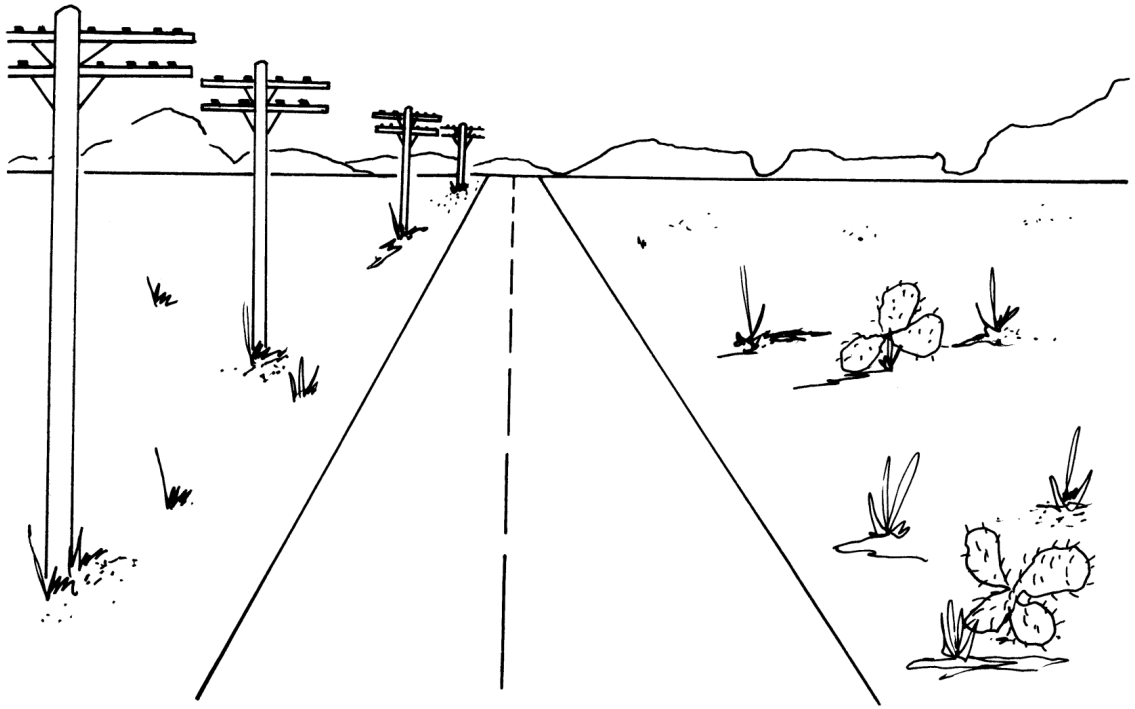
$$x = \frac{fX}{Z}, y = \frac{fY}{Z}$$

Perspective projection is a mapping from points in 3D space to rays through the Center of Projection

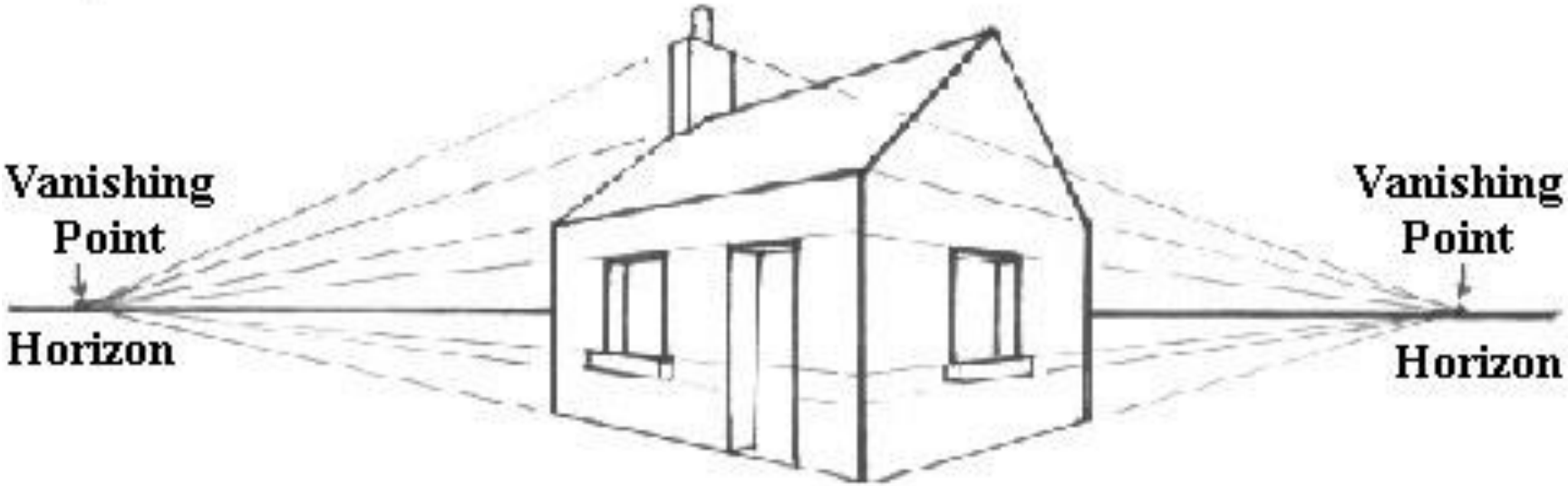
Some Perspective Phenomena



Parallel Lines Converge to a Vanishing Point

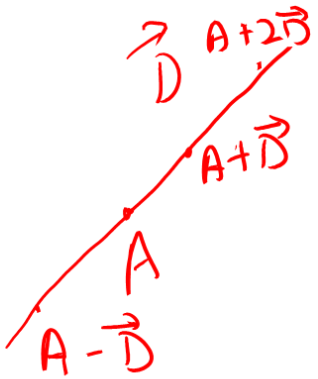


Each Family of Parallel Lines has their own vanishing points



Proof

Let there be a point A and a direction vector D in three dimensional space.



$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} + \lambda \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} \quad -\infty \leftarrow \lambda \rightarrow \infty$$

$$x = \frac{fX}{Z} = \frac{f(A_x + \lambda D_x)}{A_z + \lambda D_z}$$

Let us consider $\lambda \rightarrow \infty$

$$x = \frac{f \lambda D_x}{\lambda D_z} = \frac{f D_x}{D_z}$$

This expression does not depend on A

Coordinates of the projected point are

$\frac{f D_x}{D_z}$ for the x-coordinate

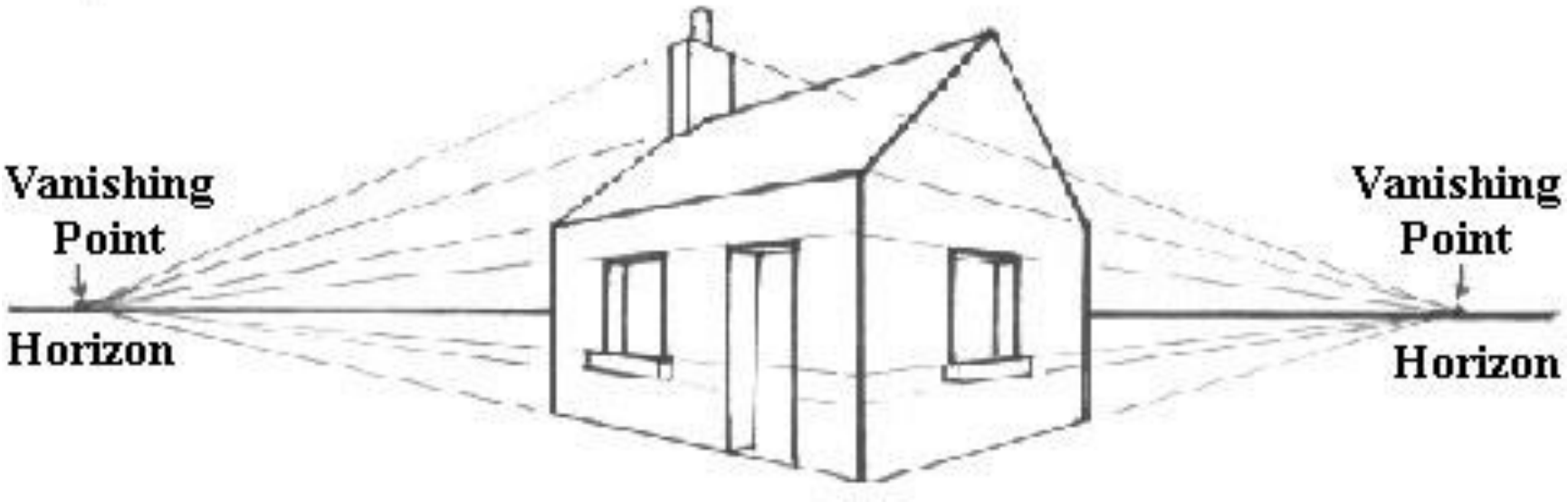
(and by doing the same process for y-coordinate

$\frac{f D_y}{D_z}$ for the y-coordinate.

Thus $\left(\frac{f D_x}{D_z}, \frac{f D_y}{D_z} \right)$ are the coordinates of the vanishing point

..

Each Family of Parallel line has its own vanishing point



But this isn't true of the vertical lines. They stay parallel. Why?

Vanishing point in Vector's notation

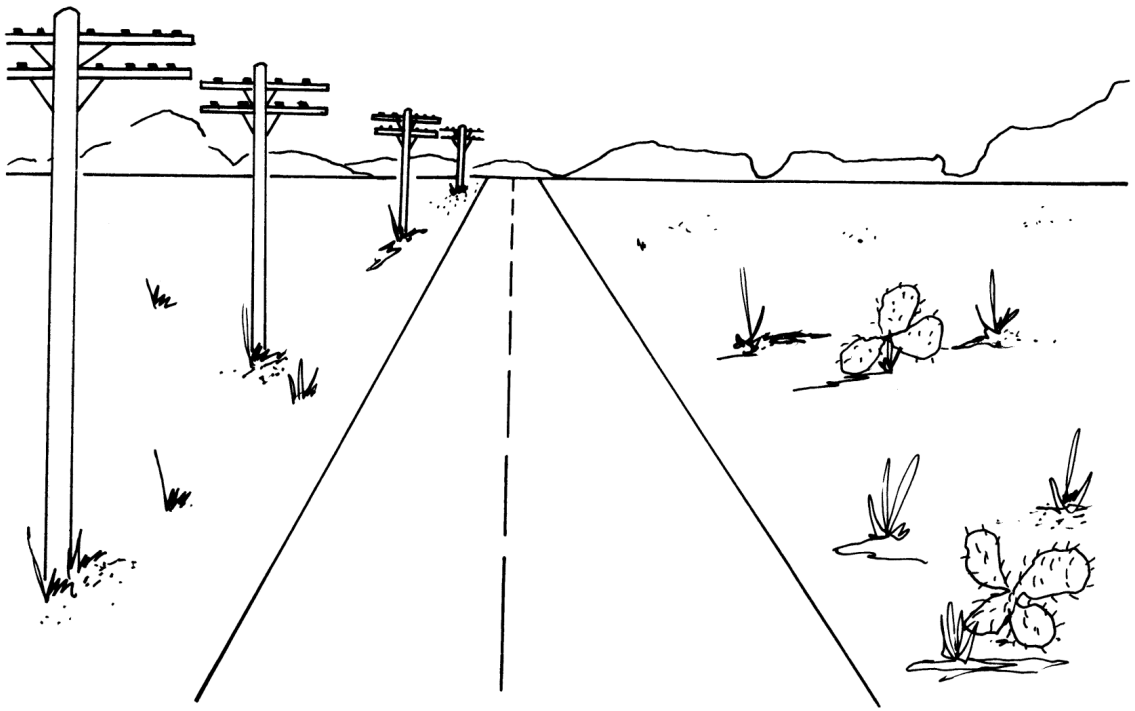
$$\mathbf{p} = f\frac{\mathbf{X}}{Z}$$

A line of points in 3D can be represented as $\mathbf{X} = \mathbf{A} + \lambda\mathbf{D}$, where $\mathbf{A}$ is a fixed point, $\mathbf{D}$ a unit vector parallel to the line, and λ a measure of distance along the line. As λ increases points are increasingly further away and in the limit:

$$\lim_{\lambda\to\infty} \mathbf{p} = f\frac{\mathbf{A} + \lambda\mathbf{D}}{A_Z + \lambda D_Z} = f\frac{\mathbf{D}}{D_Z}$$

i.e. the image of the line terminates in a *vanishing point* with coordinates $(fD_X/D_Z, fD_Y/D_Z)$, unless the line is parallel to the image plane ($D_Z = 0$). Note, the vanishing point is unaffected (invariant to) line position, $\mathbf{A}$, it only depends on line orientation, $\mathbf{D}$. Consequently, the family of lines parallel to $\mathbf{D}$ have the same vanishing point.

Nearer Objects are Lower in Image



Primitives and Transformation

- Geometric Primitives are the **basic building blocks** used to describe 3D shapes.
- In this unit, we introduce points, lines and planes
- Furthermore, the most basic transformations are discussed
- This unit covers the topics of the Szeliski book Chapter 2.1
- A more exhaustive introduction can be found in the book:

Hartley and Zisserman: Multiple View Geometry in Computer vision

2D Points

2D points can be written in **inhomogeneous coordinates** as

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2$$

or in **homogeneous coordinates** as

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^2$$

where $\mathbb{P}^2 = \mathbb{R}^3 \setminus \{(0,0,0)\}$ is called **projective space.**

Remark: Homogeneous vectors that differ only by scale are considered equivalent and define an equivalence class. $\Rightarrow$ Homogeneous vectors are defined only up to scale.

2D Points

An **inhomogeneous vector** $\mathbf{x}$ is converted to a **homogeneous vector** $\tilde{\mathbf{x}}$ as follows

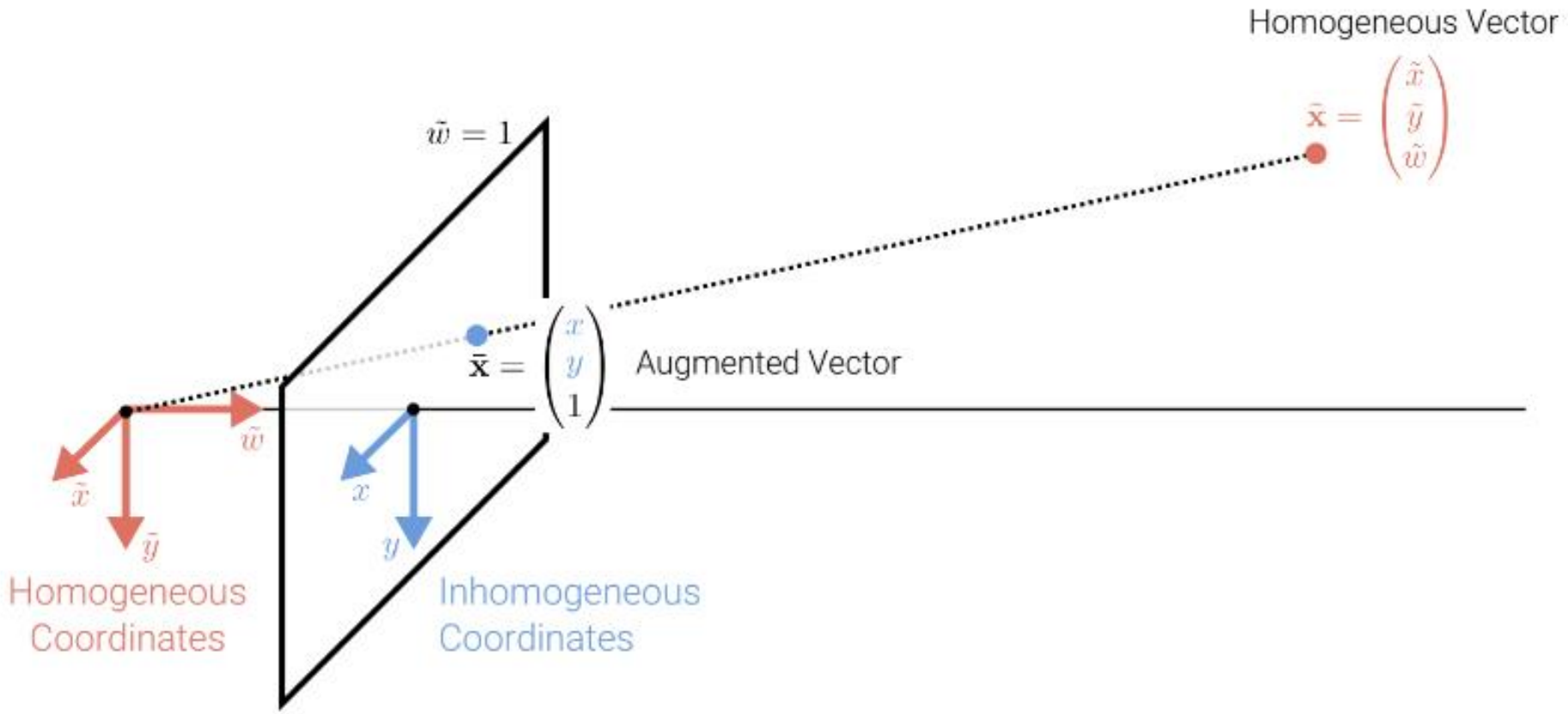
$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} = \bar{\mathbf{x}}$$

with **augmented vector** $\bar{\mathbf{x}}$.To convert in the opposite direction we divide by $\tilde{w}$:

$$\bar{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \frac{1}{\tilde{w}} \tilde{\mathbf{x}} = \frac{1}{\tilde{w}} \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \\ 1 \end{pmatrix}$$

Homogeneous points whose last element is $\tilde{w} = 0$ are called **ideal points** or **points at infinity.** These points can't be represented with inhomogeneous coordinates!

2D points


Homogeneous Vector
$\bar{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix}$
$\bar{w} = 1$
$\bar{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$
Augmented Vector
$\tilde{w}$
$\tilde{x}$
$\tilde{y}$
x
y
Homogeneous Coordinates
Inhomogeneous Coordinates

2D points

2D lines can also be expressed using homogeneous coordinates $\tilde{\mathbf{l}} = (a, b, c)^\top$:

$$\{\bar{\mathbf{x}} \mid \tilde{\mathbf{l}}^\top \bar{\mathbf{x}} = 0\} \quad \Leftrightarrow \quad \{x, y \mid ax + by + c = 0\}$$

We can **normalize** $\tilde{\mathbf{l}}$ so that $\tilde{\mathbf{l}} = (n_x, n_y, d)^\top = (\mathbf{n}, d)^\top$ with $\|\mathbf{n}\|_2 = 1$. In this case, $\mathbf{n}$ is the normal vector perpendicular to the line and d is its distance to the origin.

An exception is the **line at infinity** $\tilde{\mathbf{l}}_\infty = (0, 0, 1)^\top$ which passes through all ideal points.

Cross Product

Cross product expressed as the product of a skew-symmetric matrix and a vector:

$$\mathbf{a} \times \mathbf{b} = [\mathbf{a}]_\times \mathbf{b} = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_2\, b_3 - a_3\, b_2 \\ a_3\, b_1 - a_1\, b_3 \\ a_1\, b_2 - a_2\, b_1 \end{pmatrix}$$

Remark: In this course, we use squared brackets to distinguish matrices from vectors.

2D Line Arithmetic

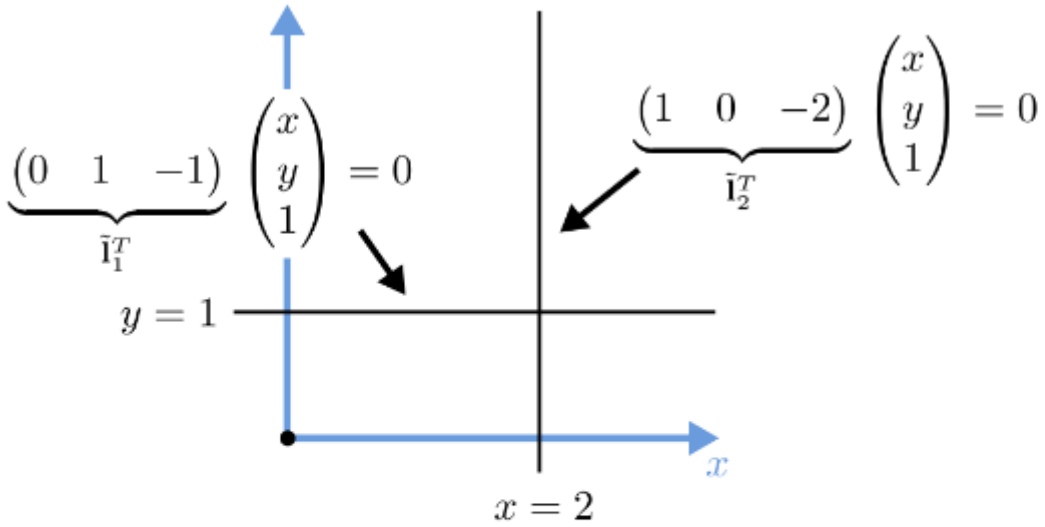
In homogeneous coordinates, the **intersection** of two lines is given by:

$$\tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$$

Similarly, the **line joining two points** can be compactly written as:

$$\tilde{\mathbf{l}} = \bar{\mathbf{x}}_1 \times \bar{\mathbf{x}}_2$$

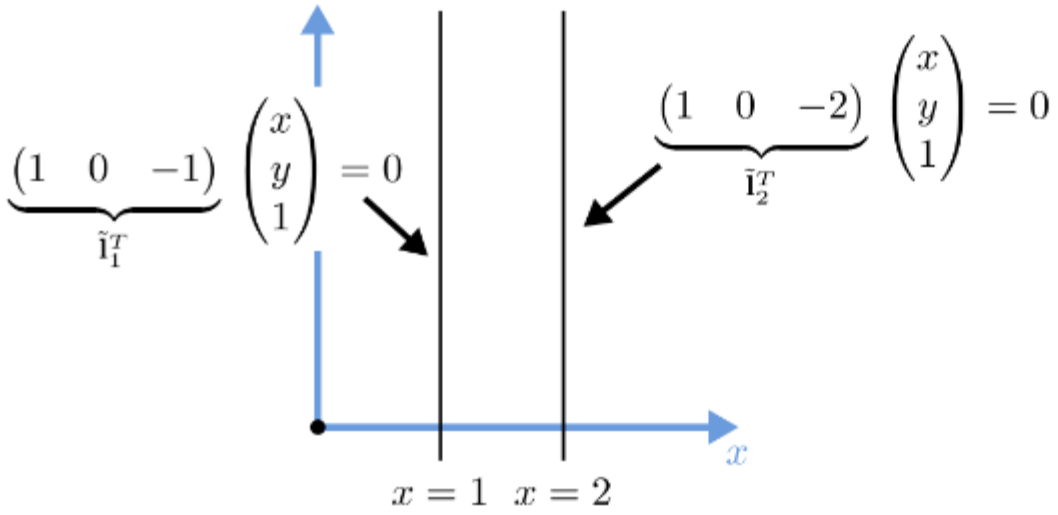
2D Arithmetic



$$[\mathbf{x}]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

$$\tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2 = [\tilde{\mathbf{l}}_1]_\times \tilde{\mathbf{l}}_2 = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 0 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} -2 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

2D Line Arithmetic



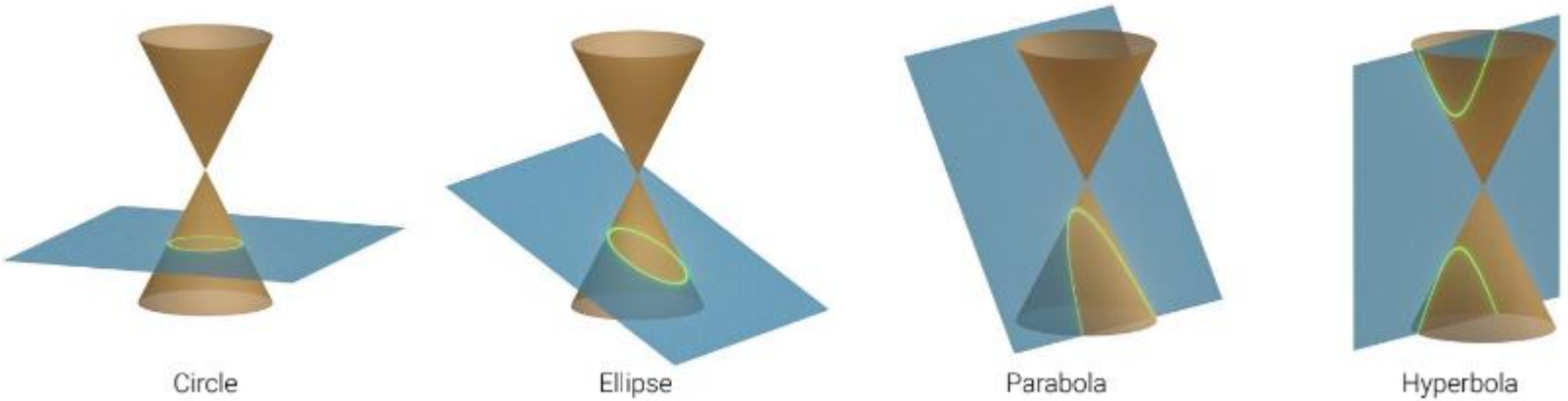
$$[\mathbf{x}]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}$$

$$\tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2 = [\tilde{\mathbf{l}}_1]_\times \tilde{\mathbf{l}}_2 = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \qquad \underbrace{\begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^\top}_{\tilde{\mathbf{l}}_\infty} \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = 0$$

2D Conics

More complex algebraic objects can be represented using **polynomial homogeneous equations.** For example, **conic sections** (arising as the intersection of a plane and a 3D cone) can be written using quadric equations:

$$\{\bar{\mathbf{x}} \mid \bar{\mathbf{x}}^\top \mathbf{Q}\, \bar{\mathbf{x}} = 0\}$$



Circle Ellipse Parabola Hyperbola

Useful for multi-view geometry and camera calibration, see Hartley and Zisserman.

3D Points

3D points can be written in **inhomogeneous coordinates** as

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3$$

or in **homogeneous coordinates** as

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^3$$

with **projective space** $\mathbb{P}^3 = \mathbb{R}^4 \setminus \{(0,0,0,0)\}$.

3D Planes

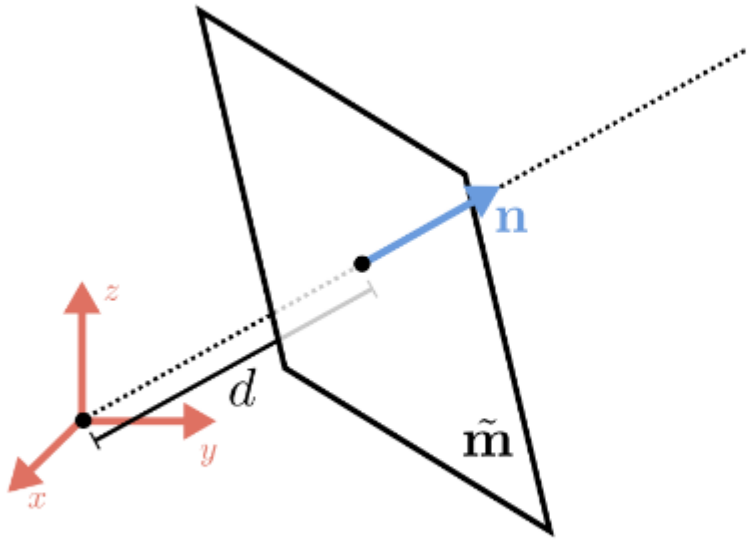
3D planes can also be represented as homogeneous coordinates $\tilde{\mathbf{m}} = (a, b, c, d)^\top$:

$$\{\bar{\mathbf{x}} \mid \tilde{\mathbf{m}}^\top \bar{\mathbf{x}} = 0\} \quad \Leftrightarrow \quad \{x, y, z \mid ax + by + cz + d = 0\}$$

Again, we can **normalize** $\tilde{\mathbf{m}}$ so that $\tilde{\mathbf{m}} = (n_x, n_y, n_z, d)^\top = (\mathbf{n}, d)^\top$ with $\|\mathbf{n}\|_2 = 1$. In this case, $\mathbf{n}$ is the normal perpendicular to the plane and d is its distance to the origin.

An exception is the **plane at infinity** $\tilde{\mathbf{m}} = (0, 0, 0, 1)^\top$ which passes through all ideal points (= points at infinity) for which $\tilde{w} = 0$.

3D Planes



$$\underbrace{\begin{pmatrix} n_x & n_y & n_z & d \end{pmatrix}}_{\tilde{\mathbf{m}}^T} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = 0$$

3D Lines

3D lines are less elegant than either 2D lines or 3D planes. One possible representation is to express points on a line as a **linear combination** of two points $\mathbf{p}$ and $\mathbf{q}$ on the line:

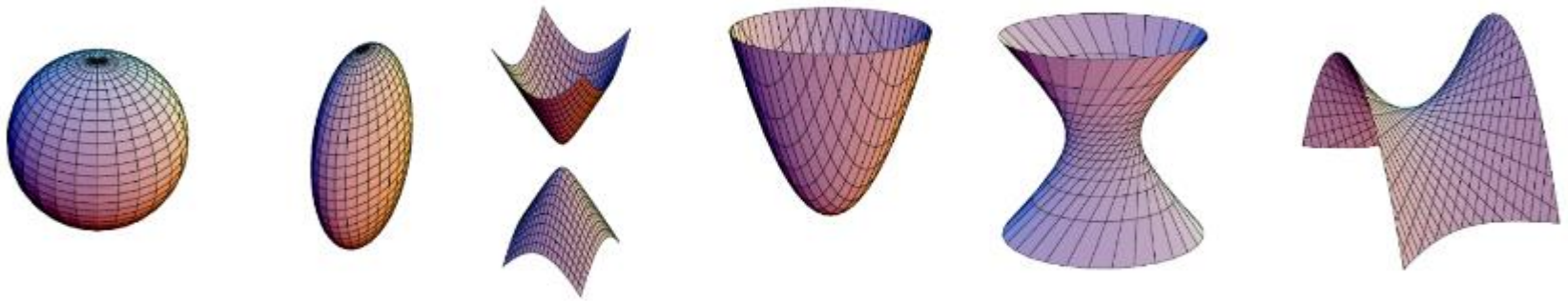
$$\{\mathbf{x} \mid \mathbf{x} = (1-\lambda)\mathbf{p} + \lambda\mathbf{q} \wedge \lambda \in \mathbb{R}\}$$

However, this representation uses 6 parameters for 4 degrees of freedom.
Alternative minimal representations are the **two-plane parameterization** or **Plücker coordinates.** See Szeliski, Chapter 2.1 and Hartley/Zisserman, Chapter 2 for details.

3D Quadrics

The 3D analog of 2D conics is a **quadric surface:**

$$\{\bar{\mathbf{x}} \mid \bar{\mathbf{x}}^\top \mathbf{Q}\, \bar{\mathbf{x}} = 0\}$$



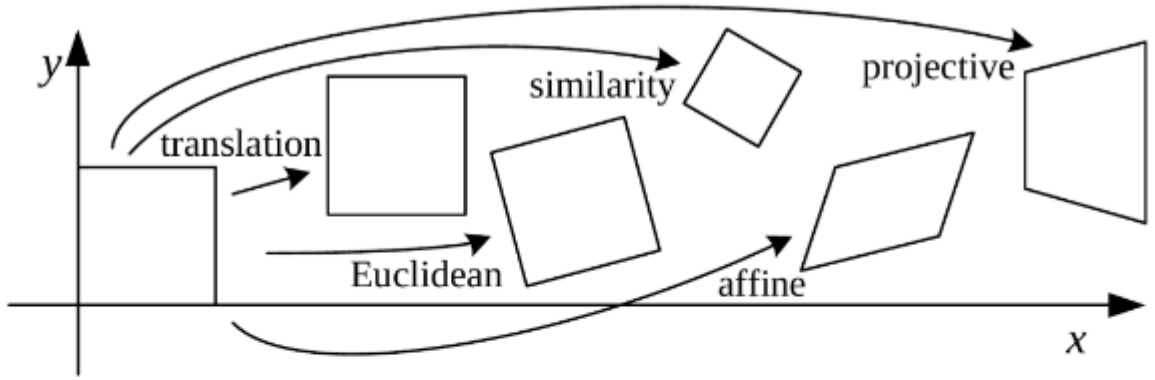
Useful in the study of multi-view geometry. Also serves as useful modeling primitives (spheres, ellipsoids, cylinders), see Hartley and Zisserman, Chapter 2 for details.

Superquadrics Revisited

Superquadrics (generalization of quadrics) for shape abstraction and compression.

Paschalidou, Ulusoy and Geiger: Superquadrics Revisited: Learning 3D Shape Parsing beyond Cuboids. CVPR, 2019.

2D Transformations

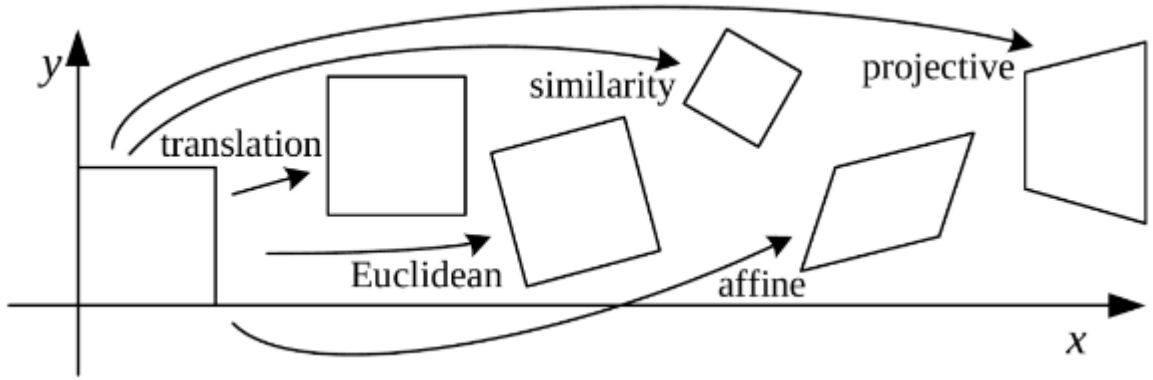


Translation: (2D Translation of the Input, 2 DoF)

$$\mathbf{x}' = \mathbf{x} + \mathbf{t} \quad \Leftrightarrow \quad \bar{\mathbf{x}}' = \begin{bmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \bar{\mathbf{x}}$$

- Using homogeneous representations allows to chain/invert transformations
- Augmented vectors $\bar{\mathbf{x}}$ can always be replaced by general homogeneous ones $\tilde{\mathbf{x}}$

2D Transformations

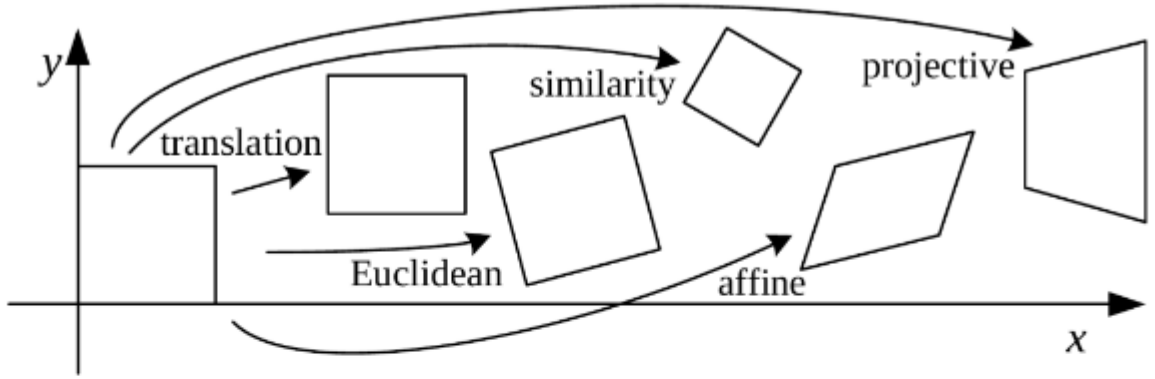


Euclidean: (2D Translation + 2D Rotation, 3 DoF)

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t} \quad \Leftrightarrow \quad \bar{\mathbf{x}}' = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \bar{\mathbf{x}}$$

- $\mathbf{R} \in SO(2)$ is an orthonormal rotation matrix with $\mathbf{R}\mathbf{R}^\top = \mathbf{I}$ and $\det(\mathbf{R}) = 1$
- Euclidean transformations preserve Euclidean distances

2D Transformations

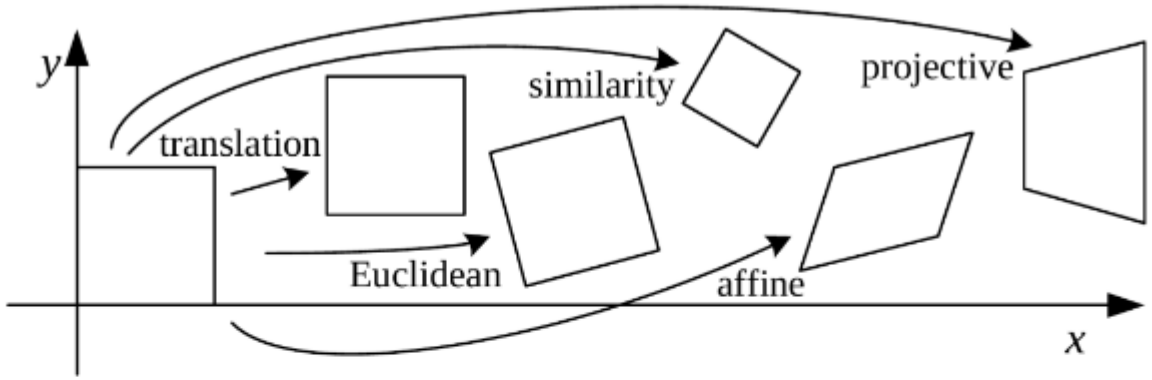


Similarity: (2D Translation + Scaled 2D Rotation, 4 DoF)

$$\mathbf{x}' = s\mathbf{R}\mathbf{x} + \mathbf{t} \quad \Leftrightarrow \quad \bar{\mathbf{x}}' = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \bar{\mathbf{x}}$$

- $\mathbf{R} \in SO(2)$ is a rotation matrix and s is an arbitrary scale factor
- The similarity transform preserves angles between lines

2D Transformations

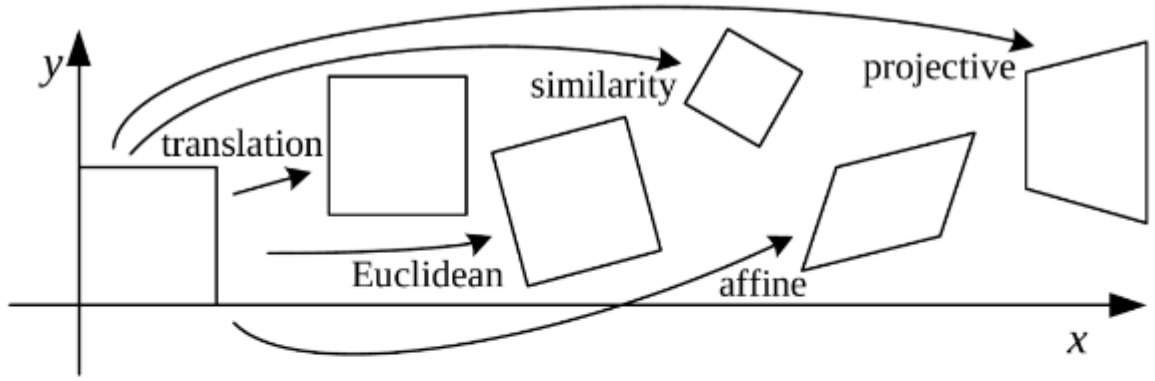


Affine: (2D Linear Transformation, 6 DoF)

$$\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{t} \quad \Leftrightarrow \quad \bar{\mathbf{x}}' = \begin{bmatrix} \mathbf{A} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \bar{\mathbf{x}}$$

- $\mathbf{A} \in \mathbb{R}^{2\times 2}$ is an arbitrary 2×2 matrix
- Parallel lines remain parallel under affine transformations

2D Transformations



Projective: (Homography, 8 DoF)

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}} \qquad \left(\bar{\mathbf{x}} = \frac{1}{\tilde{w}}\tilde{\mathbf{x}}\right)$$

- $\tilde{\mathbf{H}} \in \mathbb{R}^{3\times3}$ is an arbitrary homogeneous 3×3 matrix (specified up to scale)
- Projective transformations preserve straight lines

2D Transformations on Co-Vectors

Considering any perspective 2D transformation

$$\tilde{\mathbf{x}}' = \tilde{\mathbf{H}}\tilde{\mathbf{x}}$$

the transformed 2D line equation is given by:

$$\tilde{\mathbf{l}}'^{\top}\tilde{\mathbf{x}}' = \tilde{\mathbf{l}}'^{\top}\tilde{\mathbf{H}}\tilde{\mathbf{x}} = (\tilde{\mathbf{H}}^{\top}\tilde{\mathbf{l}}')^{\top}\tilde{\mathbf{x}} = \tilde{\mathbf{l}}^{\top}\tilde{\mathbf{x}} = 0$$

Therefore, we have:

$$\tilde{\mathbf{l}}' = \tilde{\mathbf{H}}^{-\top}\tilde{\mathbf{l}}$$

Thus, the action of a projective **transformation on a co-vector** such as a 2D line or 3D normal can be represented by the transposed inverse of the matrix.

Overview of 2D Transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{2\times3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{2\times3}$	3	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{2\times3}$	4	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{2\times3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{3\times3}$	8	straight lines	

- Transformations form **nested set of groups** (closed under composition, inverse)
- Interpret as restricted 3×3 matrices operating on 2D homogeneous coordinates
- Transformations preserve properties below (similarity: parallelism, straight lines)

Overview of 3D Transformations

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \mathbf{I} & \mathbf{t} \end{bmatrix}_{3\times4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}_{3\times4}$	6	lengths	
similarity	$\begin{bmatrix} s\mathbf{R} & \mathbf{t} \end{bmatrix}_{3\times4}$	7	angles	
affine	$\begin{bmatrix} \mathbf{A} \end{bmatrix}_{3\times4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{\mathbf{H}} \end{bmatrix}_{4\times4}$	15	straight lines	

- 3D transformations are defined analogously to 2D transformations
- 3×4 matrices are extended with a fourth $[\mathbf{0}^\top\ 1]$ row for homogeneous transforms
- Transformations preserve properties below (similarity: parallelism, straight lines)

Direct Linear Transform for Homography Estimation

How can we estimate a homography from a set of 2D correspondences?

Let $\mathcal{X} = \{\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}'_i\}_{i=1}^{N}$ denote a set of N 2D-to-2D correspondences related by $\tilde{\mathbf{x}}'_i = \tilde{\mathbf{H}}\tilde{\mathbf{x}}_i$. As the correspondence vectors are homogeneous, they have the same direction but differ in magnitude. Thus, the equation above can be expressed as $\tilde{\mathbf{x}}'_i \times \tilde{\mathbf{H}}\tilde{\mathbf{x}}_i = \mathbf{0}$. Using $\tilde{\mathbf{h}}_k^\top$ to denote the k'th row of $\tilde{\mathbf{H}}$, this can be rewritten as a linear equation in $\tilde{\mathbf{h}}$:

$$\underbrace{\begin{bmatrix} \mathbf{0}^\top & -\tilde{w}'_i\tilde{\mathbf{x}}_i^\top & \tilde{y}'_i\tilde{\mathbf{x}}_i^\top \\ \tilde{w}'_i\tilde{\mathbf{x}}_i^\top & \mathbf{0}^\top & -\tilde{x}'_i\tilde{\mathbf{x}}_i^\top \\ -\tilde{y}'_i\tilde{\mathbf{x}}_i^\top & \tilde{x}'_i\tilde{\mathbf{x}}_i^\top & \mathbf{0}^\top \end{bmatrix}}_{\mathbf{A}_i} \underbrace{\begin{bmatrix} \tilde{\mathbf{h}}_1 \\ \tilde{\mathbf{h}}_2 \\ \tilde{\mathbf{h}}_3 \end{bmatrix}}_{\tilde{\mathbf{h}}} = \mathbf{0}$$

The last row is linearly dependent (up to scale) on the first two and can be dropped.

Direct Linear Transform for Homography Estimation

Each point correspondence yields two equations. Stacking all equations into a $2N \times 9$ dimensional matrix $\mathbf{A}$ leads to the following **constrained least squares problem**

$$\begin{aligned}\tilde{\mathbf{h}}^* &= \underset{\tilde{\mathbf{h}}}{\operatorname{argmin}} \ \|\mathbf{A}\tilde{\mathbf{h}}\|_2^2 + \lambda(\|\tilde{\mathbf{h}}\|_2^2 - 1) \\ &= \underset{\tilde{\mathbf{h}}}{\operatorname{argmin}} \ \tilde{\mathbf{h}}^\top \mathbf{A}^\top \mathbf{A}\tilde{\mathbf{h}} + \lambda(\tilde{\mathbf{h}}^\top \tilde{\mathbf{h}} - 1)\end{aligned}$$

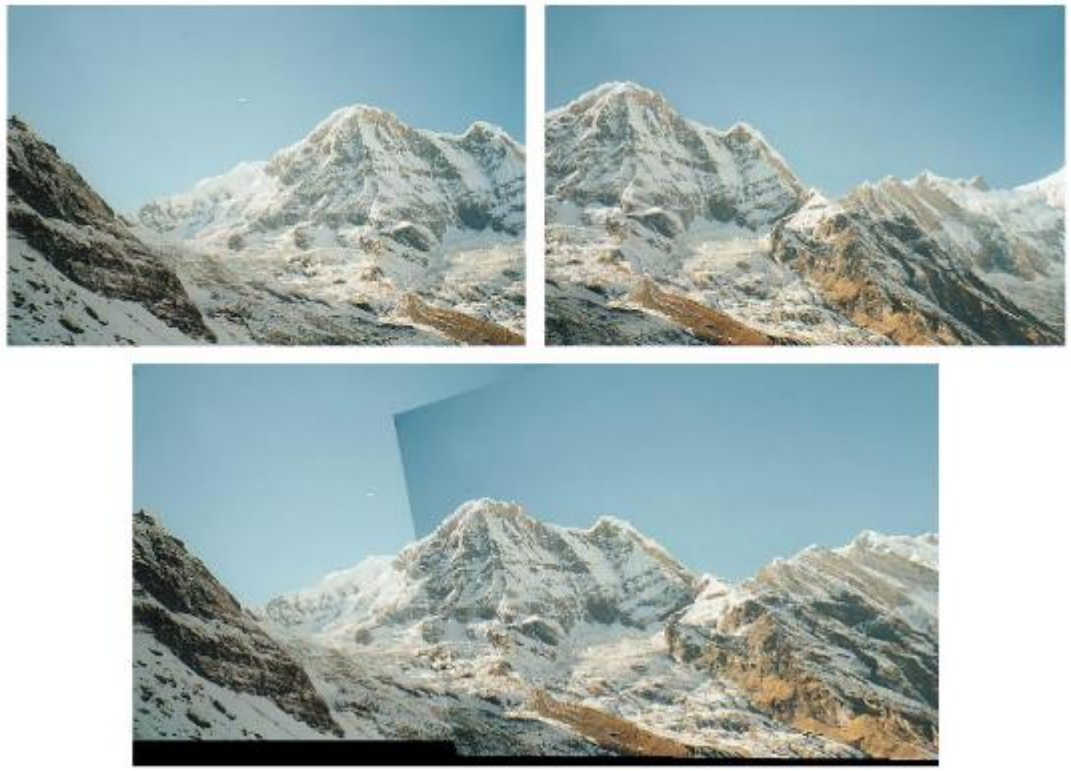
where we have fixed $\|\tilde{\mathbf{h}}\|_2^2 = 1$ as $\tilde{\mathbf{H}}$ is homogeneous (i.e., defined only up to scale) and the trivial solution to $\tilde{\mathbf{h}} = 0$ is not of interest. The solution to the above optimization problem is the **singular vector** corresponding to the smallest singular value of $\mathbf{A}$ (i.e., the last column of $\mathbf{V}$ when decomposing $\mathbf{A} = \mathbf{U}\mathbf{D}\mathbf{V}^\top$

The resulting algorithm is called **Direct Linear Transformation.**

Resource for detailed solution using Direct Linear Transformation

https://www.cs.cmu.edu/~16385/s17/Slides/10.2_2D_Alignment__DLT.pdf

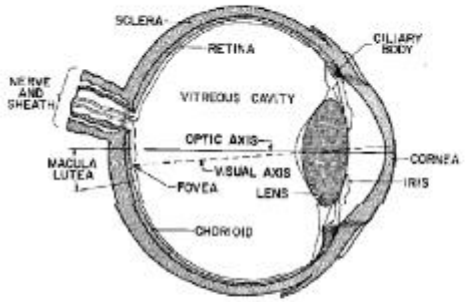
Application: Panorama Stitching



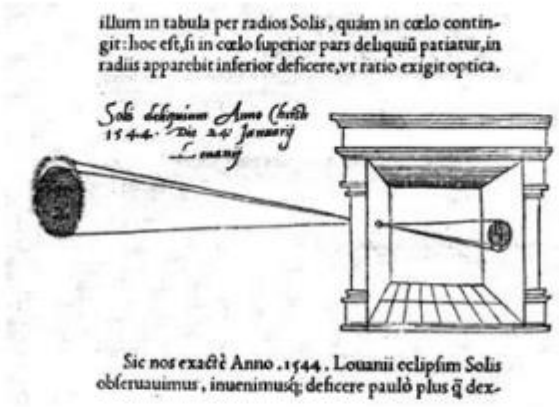
Brown and Lowe: Recognising Panoramas. ICCV, 2003

Geometric Image Formation

Origins of the Pinhole Camera



Animal Eye:
A long time ago


illum in tabula per radios Solis, quàm in cœlo contingit: hoc eſt, ſi in cœlo ſuperior pars deliquiũ patiatur, in radiis apparebit inferior deficere, vt ratio exigit optica.

Sic nos exactè Anno .1544. Louanii eclipſim Solis obſeruauimus, inuenimusq; deficere paulò plus q̃ dex-

Pinhole Perspective Projection:
Brunelleschi, 15th Century

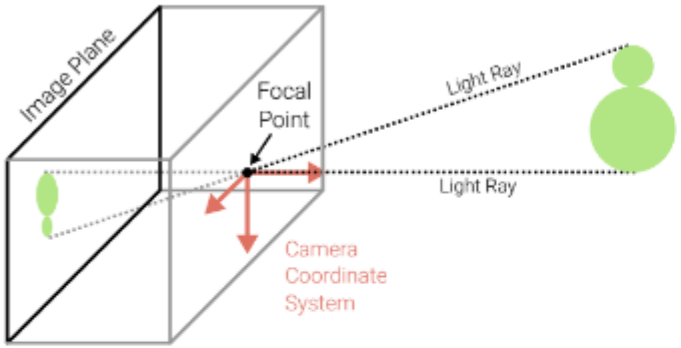
Photographic Camera:
Nicéphore Niépce, 1816

Camera Obscura:
4th Century BC

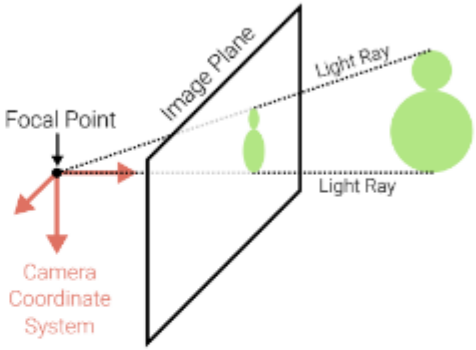
Origins of the Pinhole Camera

Origins of the Pinhole Camera

Physical Camera Model



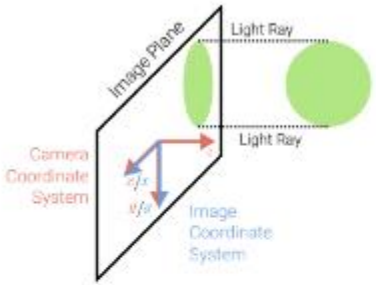
Mathematical Camera Model

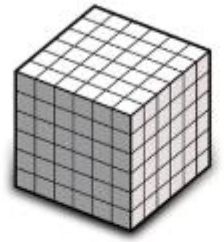


- In a physical pinhole camera the image is projected up-side down onto the image plane which is located **behind** the focal point
- When modeling perspective projection, we assume the image plane **in front**
- Both models are **equivalent,** with appropriate change of image coordinates

Projection Models

Orthographic Projection

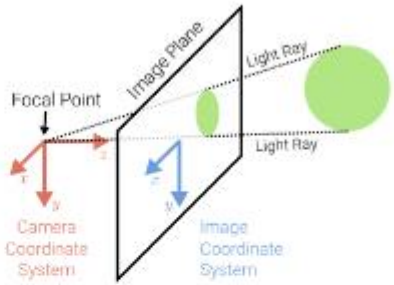




Opto Engineering Telecentric Lens

Canon 800mm Telephoto Lens

Perspective Projection



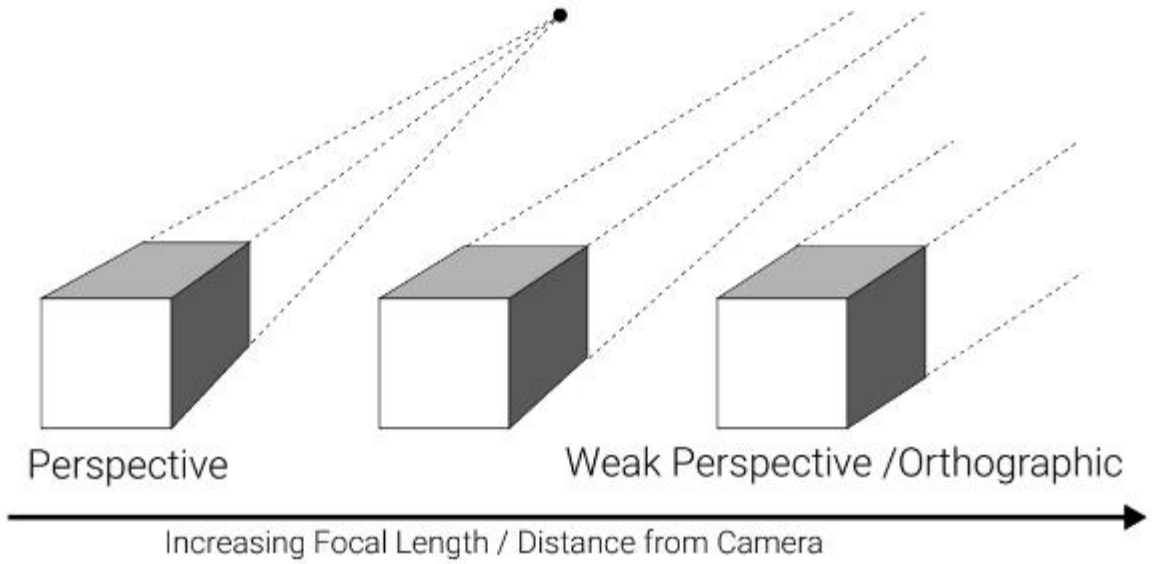
Nikon AF-S Nikkor 50mm

Sony DSC-RX100 V

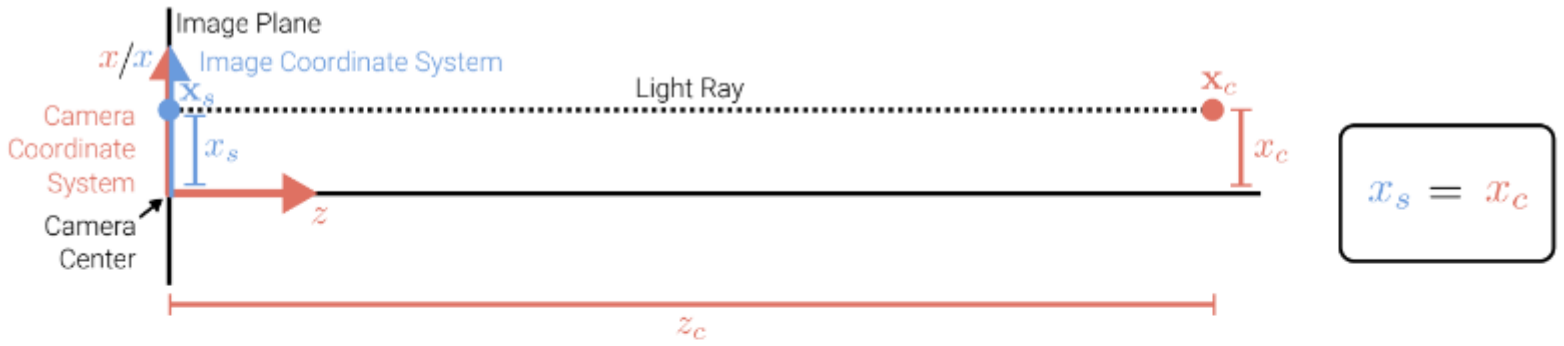
Samsung Galaxy S20

- These two are the most important projections, see Szeliski Ch. 2.1.4 for others

Projection Models



Orthographic Projection



Orthographic projection of a 3D point $\mathbf{x}_c \in \mathbb{R}^3$ to pixel coordinates $\mathbf{x}_s \in \mathbb{R}^2$:

- The x and y axes of the camera and image coordinate systems are shared
- Light rays are parallel to the z-coordinate of the camera coordinate system
- During projection, the z-coordinate is dropped, x and y remain the same
- Remark: the y coordinate is not shown here for clarity, but behaves similarly

Orthographic Projection

An **orthographic projection** simply **drops the z component** of the 3D point in camera coordinates $\mathbf{x}_c$ to obtain the corresponding 2D point on the image plane (= screen) $\mathbf{x}_s$.

$$\mathbf{x}_s = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{x}_c \quad \Leftrightarrow \quad \bar{\mathbf{x}}_s = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{x}}_c$$

Orthography is exact for telecentric lenses and an approximation for telephoto lenses.
After projection the distance of the 3D point from the image can't be recovered.

Scaled Orthographic Projection

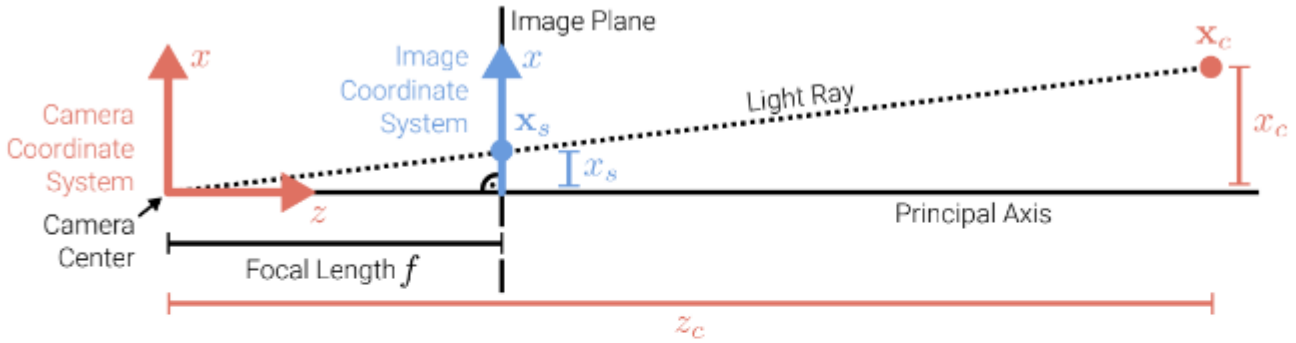
In practice, world coordinates (which may measure dimensions in meters) must be scaled to fit onto an image sensor (measuring in pixels) ⇒ **scaled orthography:**

$$\mathbf{x}_s = \begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \end{bmatrix} \mathbf{x}_c \quad \Leftrightarrow \quad \bar{\mathbf{x}}_s = \begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \bar{\mathbf{x}}_c$$

Remark: The unit for s is px/m or px/mm to convert metric 3D points into pixels.

Under orthography, structure and motion can be estimated simultaneously using factorization methods (e.g., via singular value decomposition).

Perspective Projection



$$\frac{x_s}{f} = \frac{x_c}{z_c}$$

Perspective projection of a 3D point $\mathbf{x}_c \in \mathbb{R}^3$ to pixel coordinates $\mathbf{x}_s \in \mathbb{R}^2$:

- The light ray passes through the camera center, the pixel $\mathbf{x}_s$ and the point $\mathbf{x}_c$
- Convention: the principal axis (orthogonal to image plane) aligns with the z-axis
- Remark: the y coordinate is not shown here for clarity, but behaves similarly

Perspective Projection

In **perspective projection,** 3D points in camera coordinates are mapped to the image plane by **dividing** them **by their z component** and multiplying with the focal length:

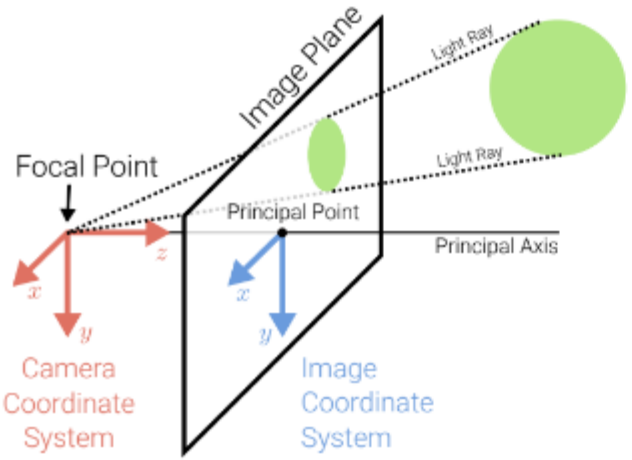
$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} = \begin{pmatrix} f x_c / z_c \\ f y_c / z_c \end{pmatrix} \quad \Leftrightarrow \quad \tilde{\mathbf{x}}_s = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \bar{\mathbf{x}}_c$$

Note that this projection is **linear** when using **homogeneous coordinates.** After the projection it is not possible to recover the distance of the 3D point from the image.

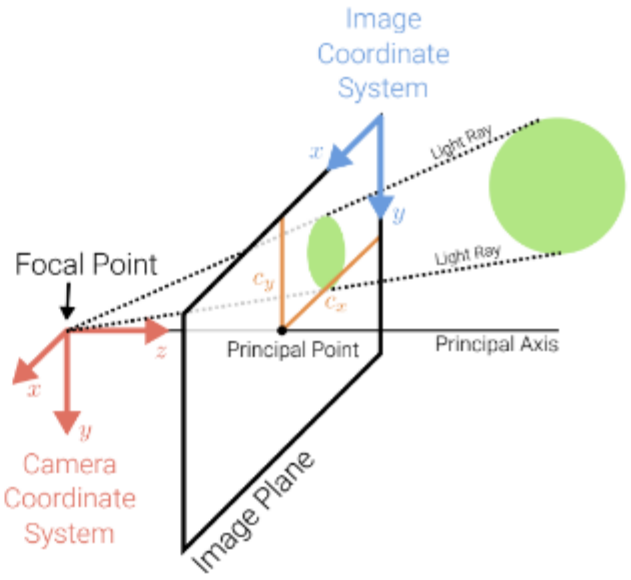
Remark: The unit for f is px (=pixels) to convert metric 3D points into pixels.

Perspective Projection

Without Principal Point Offset



With Principal Point Offset



- To ensure positive pixel coordinates, a **principal point offset** $\mathbf{c}$ is usually added
- This moves the image coordinate system to the corner of the image plane

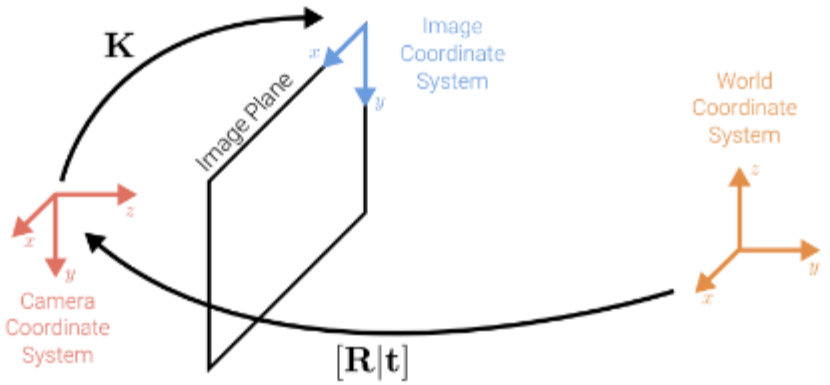
Perspective Projection

The **complete perspective projection model** is given by:

$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} = \begin{pmatrix} f_x\, x_c/z_c + s\, y_c/z_c + c_x \\ f_y\, y_c/z_c + c_y \end{pmatrix} \quad \Leftrightarrow \quad \tilde{\mathbf{x}}_s = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \bar{\mathbf{x}}_c$$

- The left 3×3 submatrix of the projection matrix is called **calibration matrix** $\mathbf{K}$
- The parameters of $\mathbf{K}$ are called camera intrinsics (as opposed to extrinsic pose)
- Here, f_x and f_y are independent, allowing for different pixel aspect ratios
- The skew s arises due to the sensor not mounted perpendicular to the optical axis
- In practice, we often set $f_x = f_y$ and $s = 0$, but model $\mathbf{c} = (c_x, c_y)^\top$

Chaining Transformations



Let $\mathbf{K}$ be the calibration matrix (intrinsics) and $[\mathbf{R}|\mathbf{t}]$ the camera pose (extrinsics).
We **chain both transformations** to project a point in world coordinates to the image:

$$\tilde{\mathbf{x}}_s = \begin{bmatrix} \mathbf{K} & \mathbf{0} \end{bmatrix} \bar{\mathbf{x}}_c = \begin{bmatrix} \mathbf{K} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \bar{\mathbf{x}}_w = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix} \bar{\mathbf{x}}_w = \mathbf{P}\, \bar{\mathbf{x}}_w$$

Remark: The 3×4 projection matrix $\mathbf{P}$ can be pre-computed.

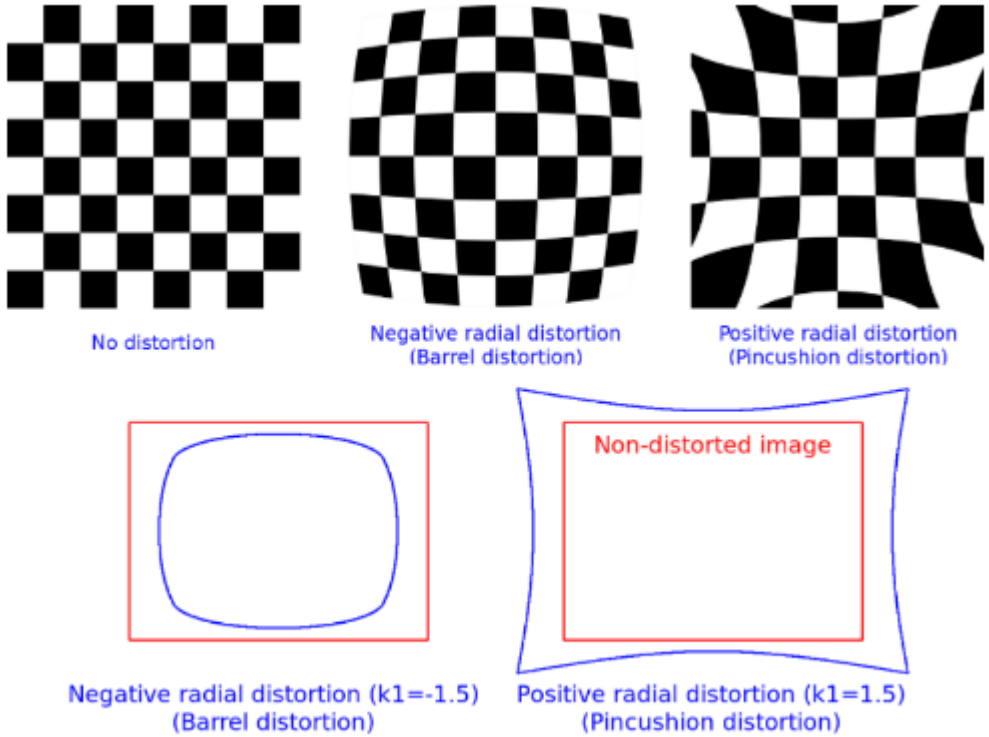
Lens Distortion

The assumption of linear projection (straight lines remain straig
practice due to the properties of the camera lens which introdu
Both **radial and tangential distortion** effects can be modeled r
Let $x = x_c/z_c$, $y = y_c/z_c$ and $r^2 = x^2 + y^2$. The distorted point is

$$\mathbf{x}' = \underbrace{(1 + \kappa_1 r^2 + \kappa_2 r^4)}_{\text{Radial Distortion}} \begin{pmatrix} x \\ y \end{pmatrix} + \underbrace{\begin{pmatrix} 2\kappa_3 x y + \kappa_4(r^2 \\ 2\kappa_4 x y + \kappa_3(r^2 \end{pmatrix}}_{\text{Tangential Dist}}$$

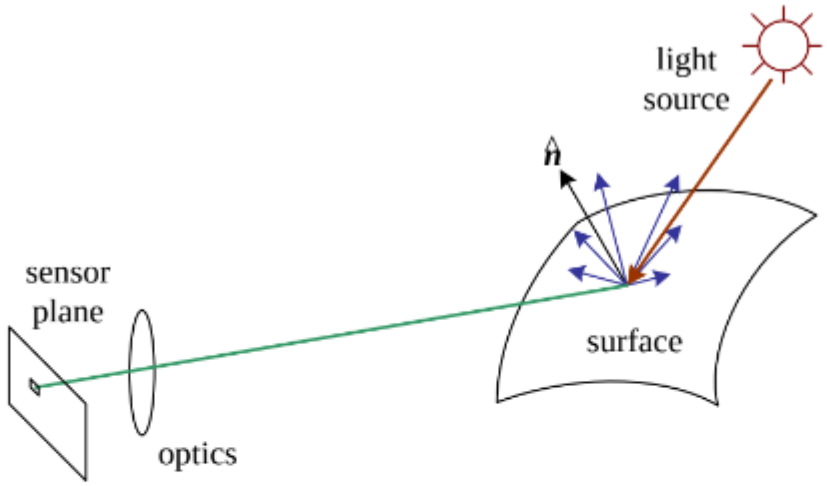
$$\mathbf{x}_s = \begin{pmatrix} f_x x' + c_x \\ f_y y' + c_y \end{pmatrix}$$

Images can be **undistorted** such that the perspective projectio
More complex distortion models must be used for wide-angle l



Photometric Image Formation

Photometric Image Formation



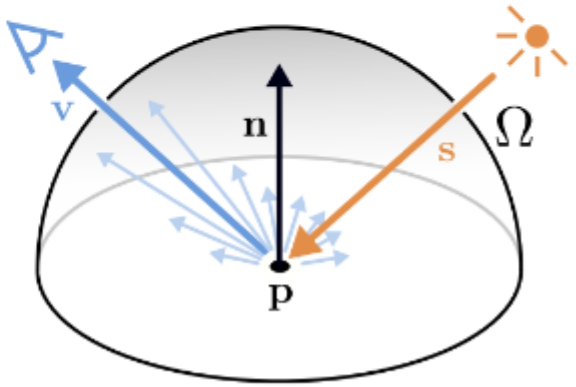
- So far we have discussed how individual light rays travel through space
- We now discuss how an image is formed in terms of **pixel intensities and colors**
- Light is **emitted** by one or more light sources and **reflected** or **refracted** (once or multiple times) at surfaces of objects (or media) in the scene

Rendering Equation

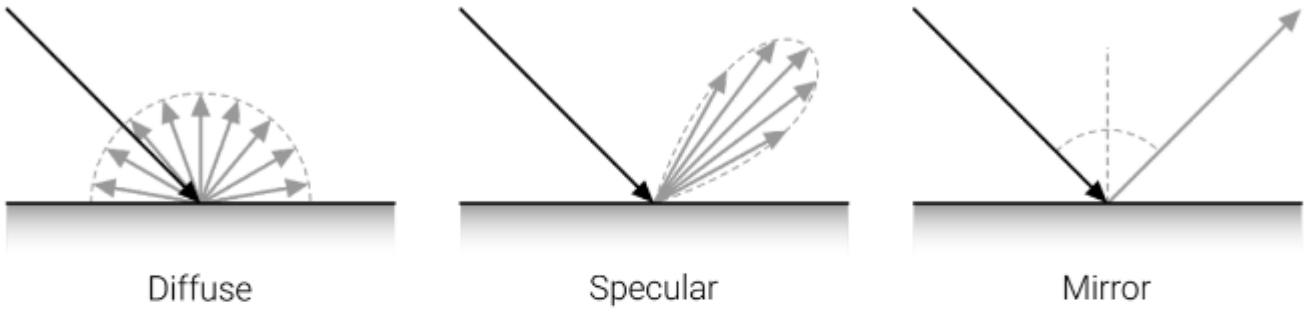
Let $\mathbf{p} \in \mathbb{R}^3$ denote a 3D surface point, $\mathbf{v} \in \mathbb{R}^3$ the viewing direction and $\mathbf{s} \in \mathbb{R}^3$ the incoming light direction. The **rendering equation** describes how much of the light L_{in} with wavelength λ arriving at $\mathbf{p}$ is reflected into the viewing direction $\mathbf{v}$:

$$L_{\text{out}}(\mathbf{p}, \mathbf{v}, \lambda) = L_{\text{emit}}(\mathbf{p}, \mathbf{v}, \lambda) + \int_{\Omega} \text{BRDF}(\mathbf{p}, \mathbf{s}, \mathbf{v}, \lambda) \cdot L_{\text{in}}(\mathbf{p}, \mathbf{s}, \lambda) \cdot (-\mathbf{n}^{\top}\mathbf{s})\, d\mathbf{s}$$

- Ω is the unit hemisphere at normal $\mathbf{n}$
- The bidirectional reflectance distribution function $\text{BRDF}(\mathbf{p}, \mathbf{s}, \mathbf{v}, \lambda)$ defines how light is reflected at an opaque surface.
- $L_{\text{emit}} > 0$ only for light emitting surfaces

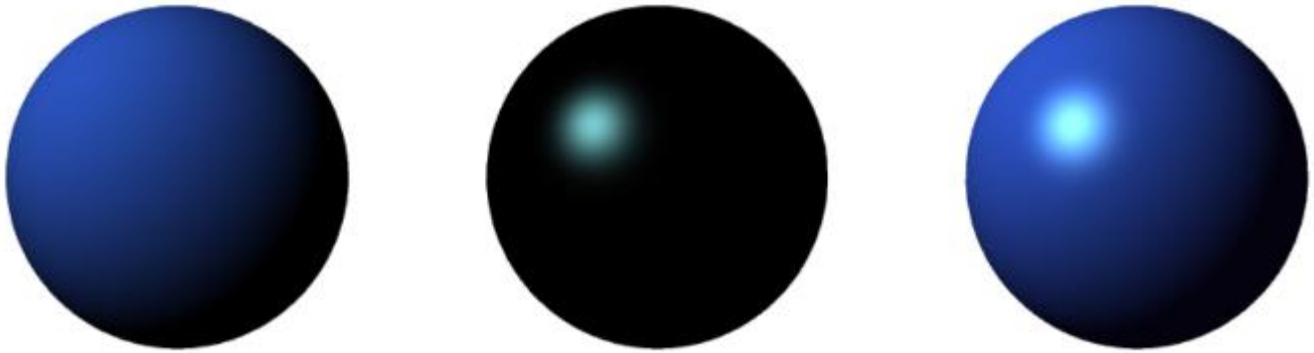


Diffuse and Specular Reflection



- Typical BRDFs have a **diffuse** and a **specular** component
- The diffuse (=constant) component scatters light uniformly in all directions
- This leads to shading, i.e., smooth variation of intensity wrt. surface normal
- The specular component depends strongly on the outgoing light direction

Diffuse and Specular Reflection



Diffuse Specular Combined

- Typical BRDFs have a **diffuse** and a **specular** component
- The diffuse (=constant) component scatters light uniformly in all directions
- This leads to shading, i.e., smooth variation of intensity wrt. surface normal
- The specular component depends strongly on the outgoing light direction

BRDF Examples

- BRDFs can be very complex and spatially varying

Slide Credits: Svetlana Lazebnik

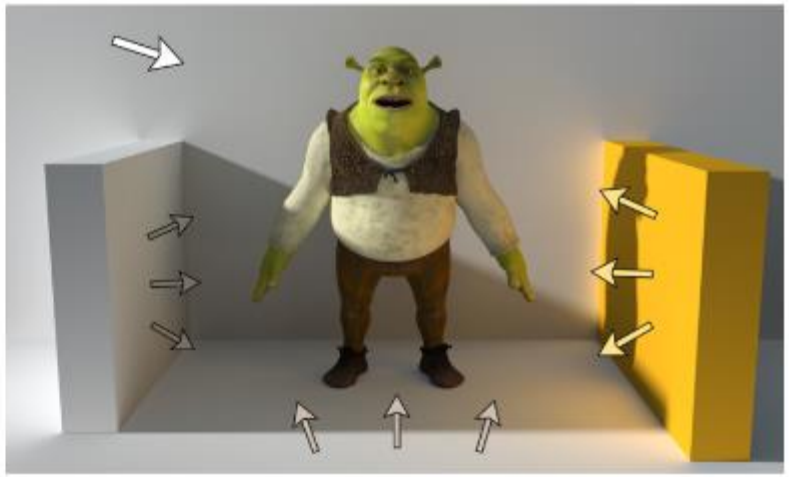
Fresnel Effect

- The amount of light reflected from a surface depends on the viewing angle

Slide Credits: Filament Documentation

Global Illumination

Rendering with Direct Lighting



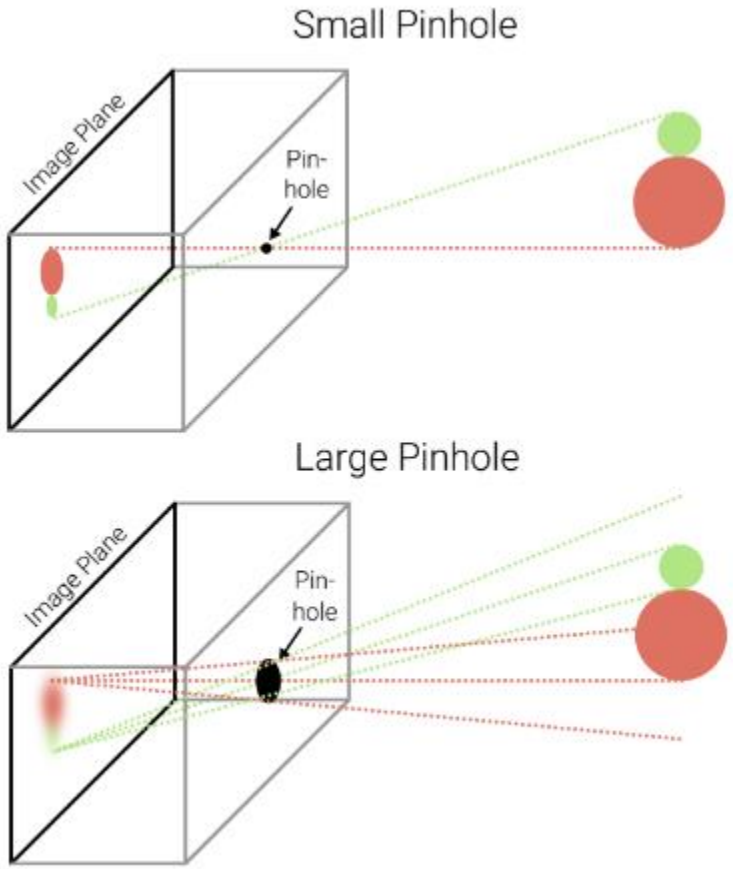
Rendering with Global Illumination

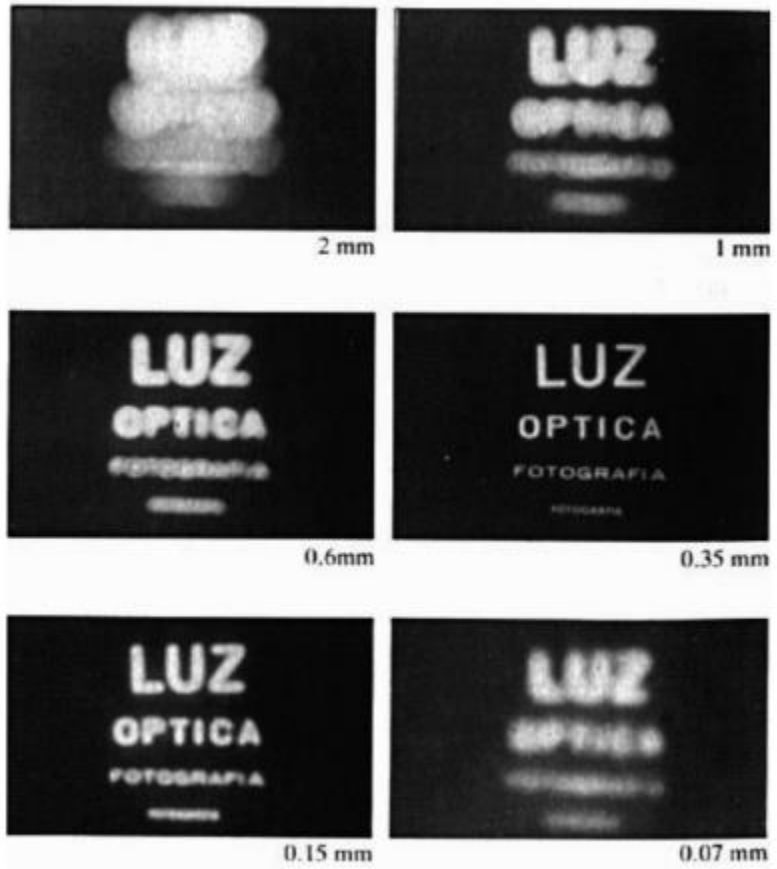
- Modeling one light bounce is insufficient for rendering complex scenes
- Light sources can be shadowed by occluders and rays can bounce multiple times
- **Global illumination** techniques also take indirect illumination into account

Why Camera Lenses

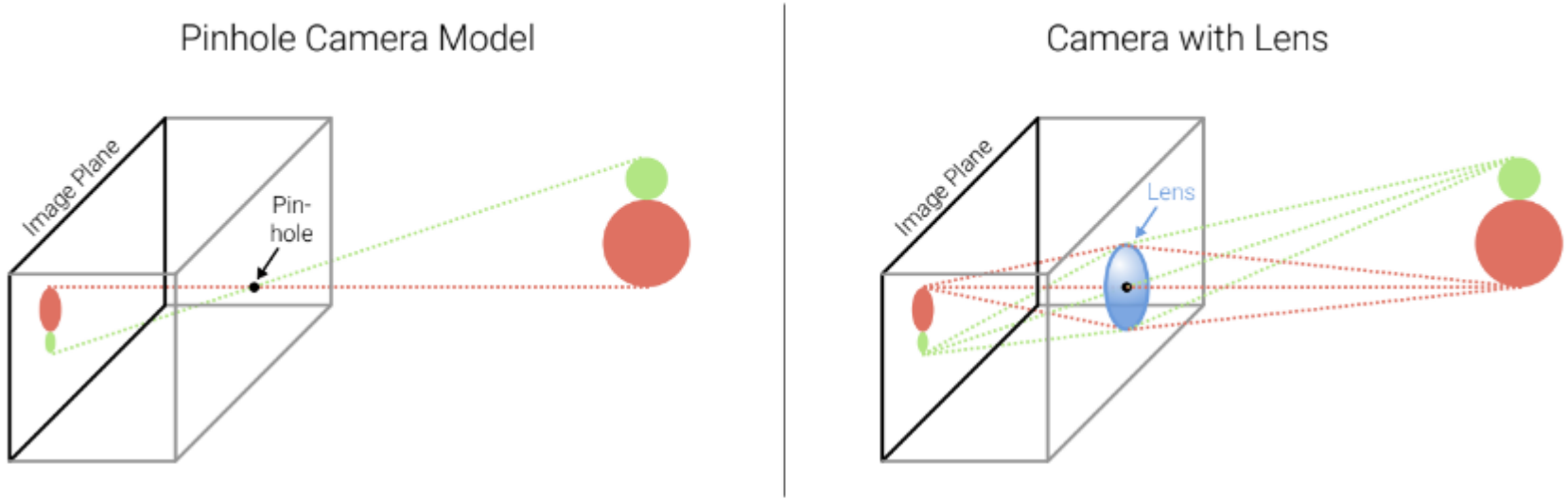
- Large and very small pinholes result in **image blur** (averaging, diffraction)
- Small pinholes require very **long shutter times** (⇒ motion blur)
- `http://www.pauldebevec.com/Pinhole/`

Why Camera Lenses



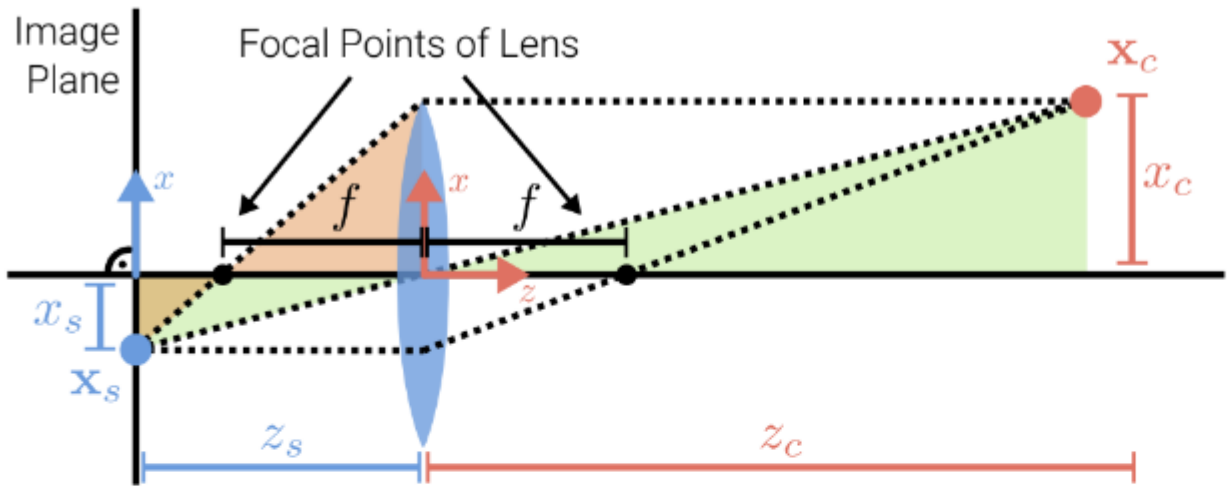


Optics



- Cameras use one or multiple **lenses** to accumulate light on the sensor plane
- Importantly, if a 3D point is in **focus**, all light rays arrive at the same 2D pixel
- For many applications it suffices to model lens cameras with a pinhole model
- However, to address **focus, vignetting and aberration** we need to model lenses

Thin Lens Model



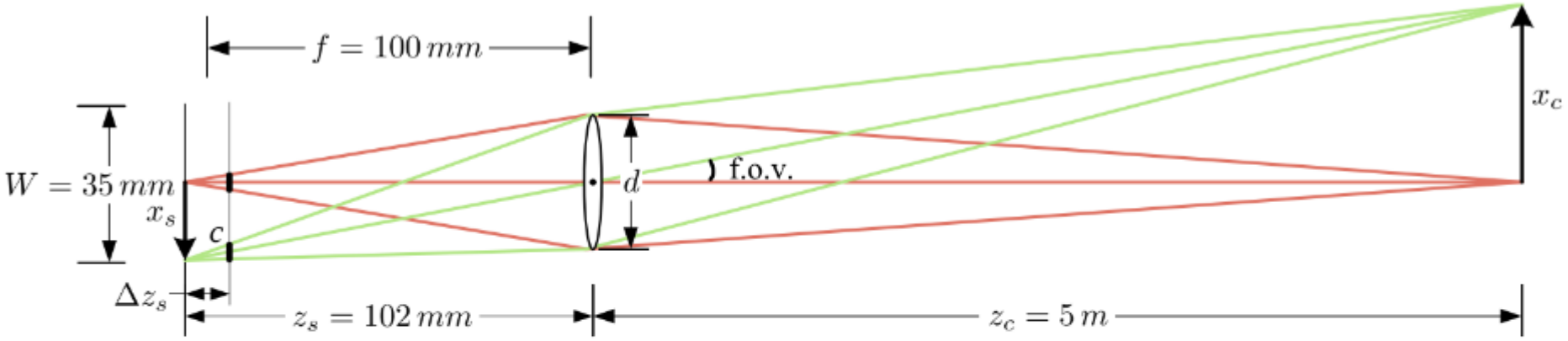
$$\frac{x_s}{z_s - f} = \frac{x_c}{f}$$

$$\frac{x_s}{z_s} = \frac{x_c}{z_c}$$

$$\frac{x_s}{x_c} = \frac{z_s - f}{f} \quad \wedge \quad \frac{x_s}{x_c} = \frac{z_s}{z_c} \quad \Rightarrow \quad \frac{z_s - f}{f} = \frac{z_s}{z_c} \quad \Rightarrow \quad \frac{z_s}{f} - 1 = \frac{z_s}{z_c} \quad \Rightarrow \quad \frac{1}{z_s} + \frac{1}{z_c} = \frac{1}{f}$$

- The **thin lens model** with spherical lens is often used as an approximation
- Properties: Axis-parallel rays pass the focal point, rays via center keep direction
- From Snell's law we obtain $f = \frac{R}{2(n-1)}$ with radius R and index of refraction n

Depth of Field (DOF)

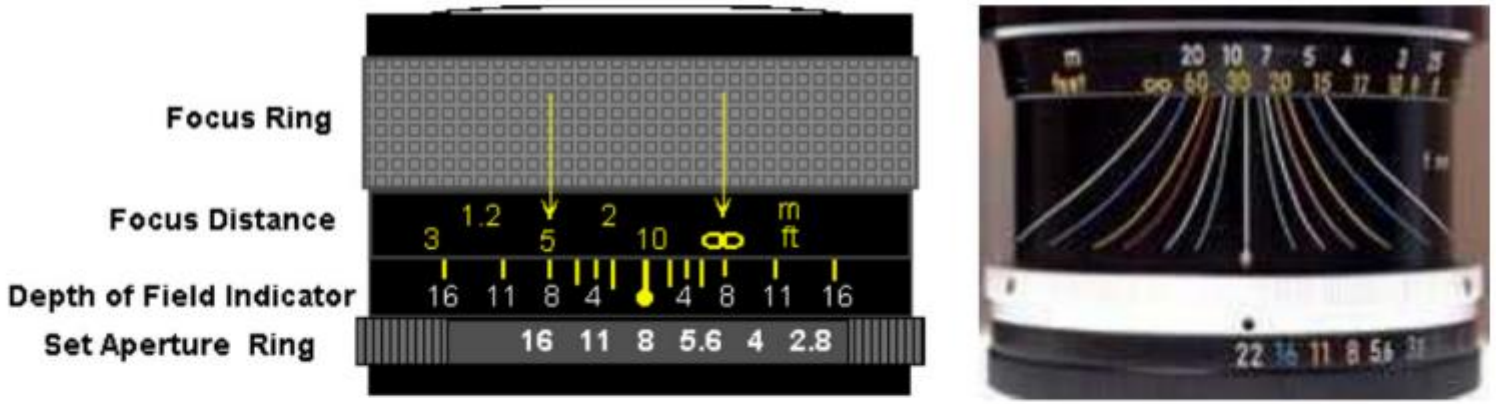


- The image is **in focus** if $\frac{1}{z_s} + \frac{1}{z_c} = \frac{1}{f}$ where f is the focal length of the lens
- For $z_c \to \infty$ we obtain $z_s = f$ (lens with focal length $f \approx$ pinhole at distance f)
- If the image plane is **out of focus**, a 3D point projects to the **circle of confusion** c

Depth of Field (DOF)

- To control the **size of the circle of confusion,** we change the lens **aperture**
- An aperture is a hole or an opening through which light travels
- The aperture limits the amount of light that can reach the image plane
- Smaller apertures lead to sharper, but more noisy images (less photons)

Depth of Field (DOF)



- The allowable depth variation that limits the circle of confusion c is called **depth of field** and is a function of both the focus distance and the lens aperture
- Typical DSLR lenses have depth of field indicators
- The commonly displayed **f-number** is defined as

$$N = \frac{f}{d} \qquad \text{(often denoted as } f/N\text{, e.g.: } f/1.4\text{)}$$

- In other words, it is the lens focal length f divided by the aperture diameter d

Depth of Field (DOF)

Aperture = f/1.4
DOF = 0.8 cm

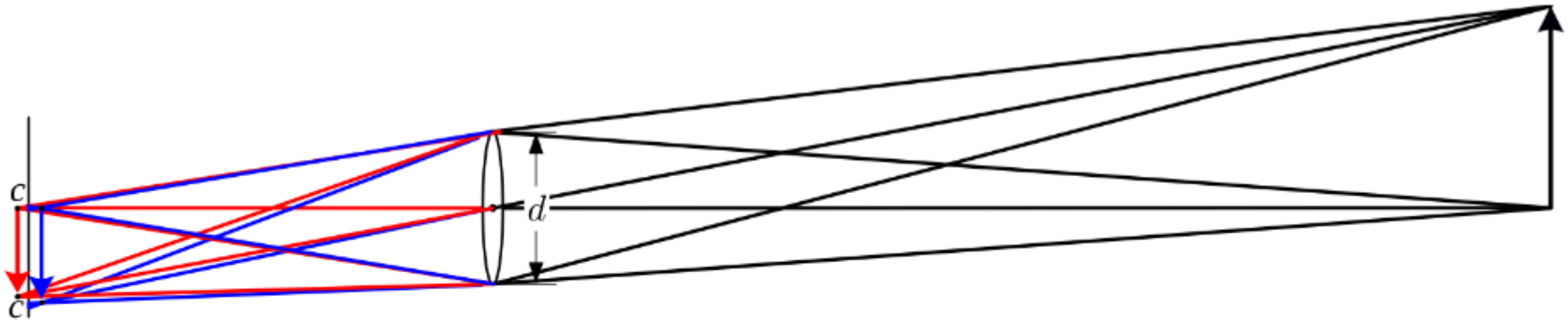
Aperture = f/4.0
DOF = 2.2 cm

Aperture = f/22
DOF = 12.4 cm

Depth of Field (DOF):

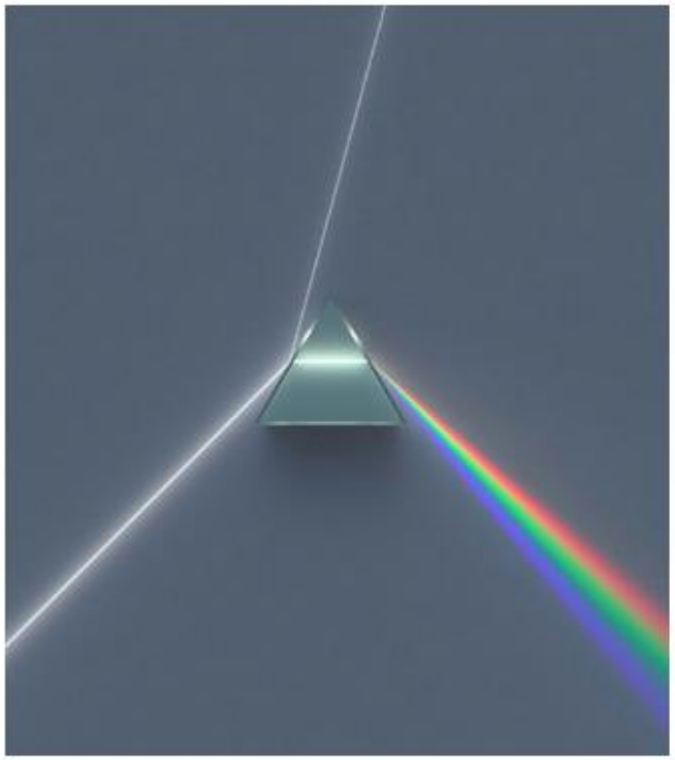
- Distance between the nearest and farthest objects that are acceptably sharp
- Decreasing the aperture diameter (increasing the f-number) increases the DOF

Chromatic Aberration



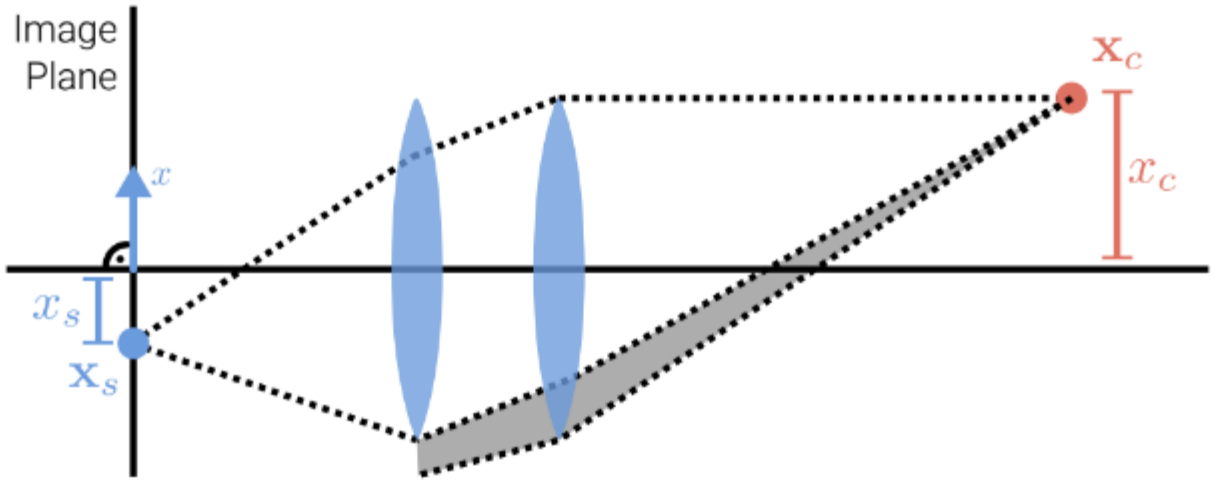
- The **index of refraction** for glass varies slightly as a function of wavelength
- Thus, simple lenses suffer from **chromatic aberration** which is the tendency for light of different colors to focus at slightly different distances (blur, color shift)
- To reduce chromatic and other kinds of aberrations, most photographic lenses are compound lenses made of different glass elements (with different coatings)

Chromatic Aberration



- Top: High-quality lens Bottom: Low-quality lens (blur, rainbow edges)

Vignetting

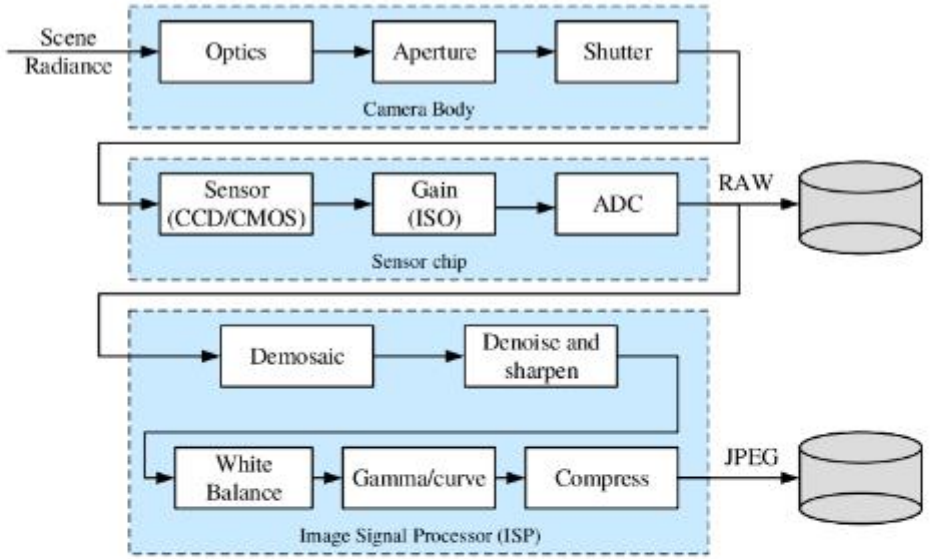


- Vignetting is the tendency for the brightness to fall off towards the image edge
- Composition of two effects: natural and mechanical vignetting
- Natural vignetting: foreshortening of object surface and lens aperture
- Mechanical vignetting: the shaded part of the beam never reaches the image
- Vignetting can be calibrated (i.e., undone)

Vignetting

Image Sensing Pipeline

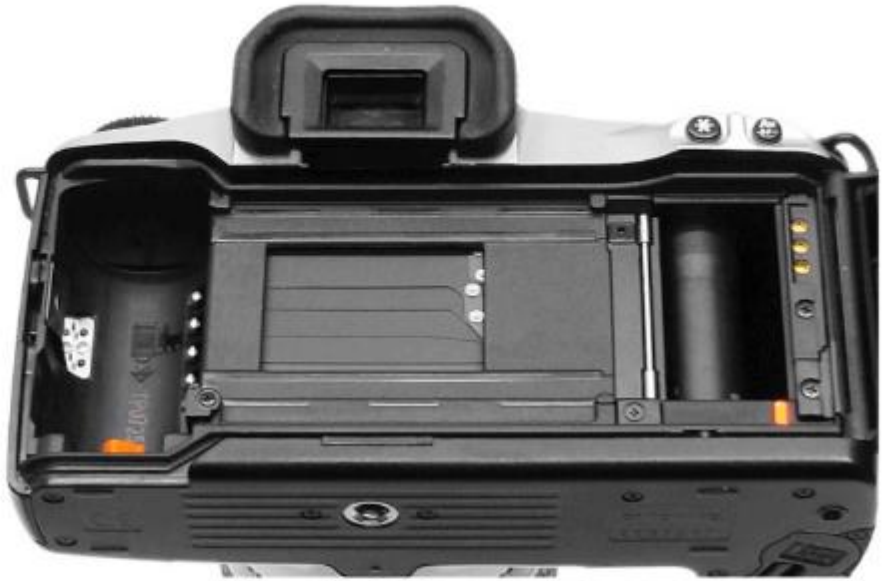
Image Sensing Pipeline



The **image sensing pipeline** can be divided into three stages:

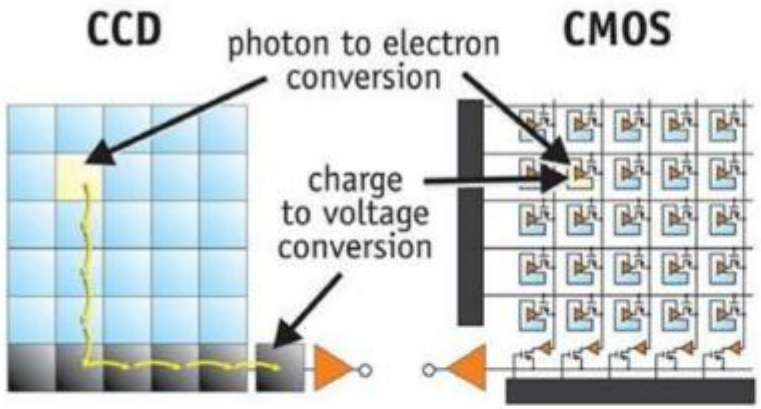
- **Physical light transport** in the camera lens/body
- **Photon measurement** and conversion on the sensor chip
- **Image signal processing (ISP)** and image compression

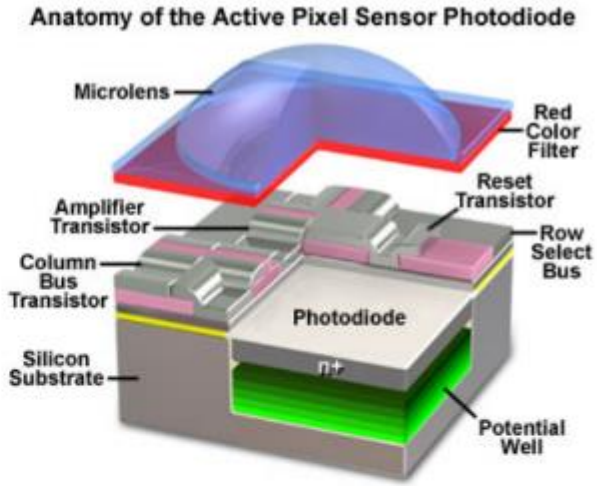
Shutter



- A **focal plane shutter** is positioned just in front the image sensor / film
- Most digital cameras use a combination of mechanical and electronic shutter
- The shutter speed (exposure time) controls how much light reaches the sensor
- It determines if an image appears over-/underexposed, blurred or noisy

Sensor





- **CCDs** move charge from pixel to pixel and convert it to voltage at the output node
- **CMOS** images convert charge to voltage inside each pixel and are standard
- Larger chips (full frame = 35 mm) are more photo-sensitive ⇒ less noise

`https://meroli.web.cern.ch/lecture_cmos_vs_ccd_pixel_sensor.html`

Color Filter Arrays

G	R	G	R
B	G	B	G
G	R	G	R
B	G	B	G

Bayer RGB Pattern

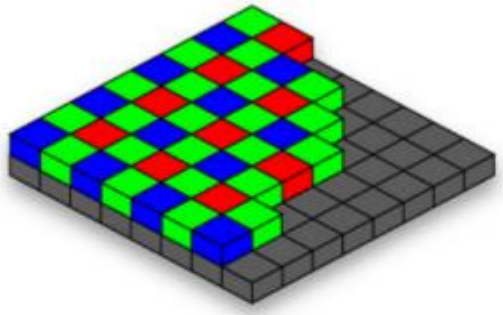
rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb
rGb	Rgb	rGb	Rgb
rgB	rGb	rgB	rGb

Interpolated Pixels

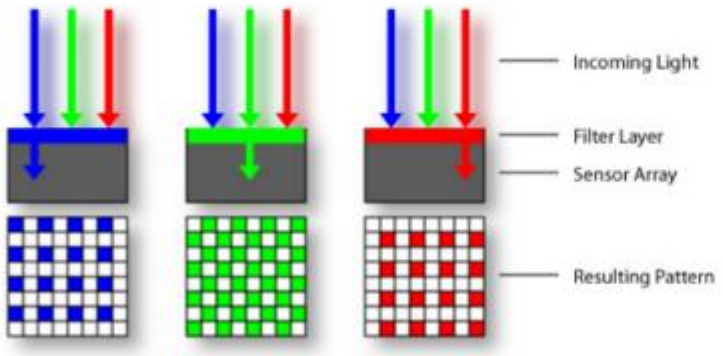
- To measure color, pixels are arranged in a **color array,** e.g.: Bayer RGB pattern
- Missing colors at each pixel are interpolated from the neighbors (demosaicing)

Color Filter Arrays

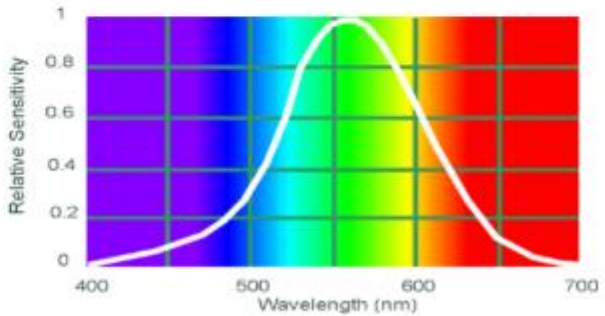
Bayer grid



Estimate missing components from neighboring values (demosaicing)

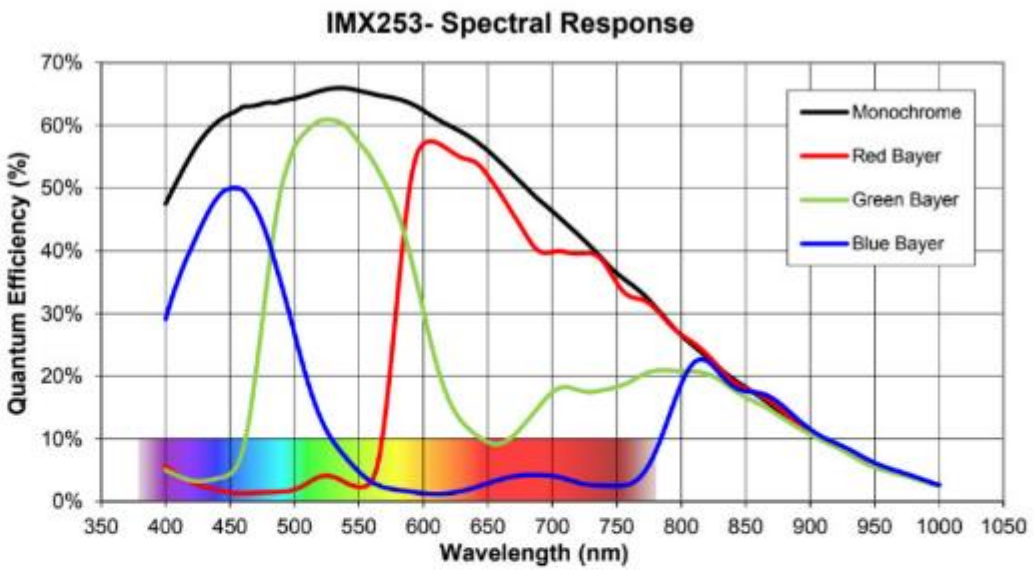


Why more green?



Slide Credits: Steve Seitz

Color Filter Arrays

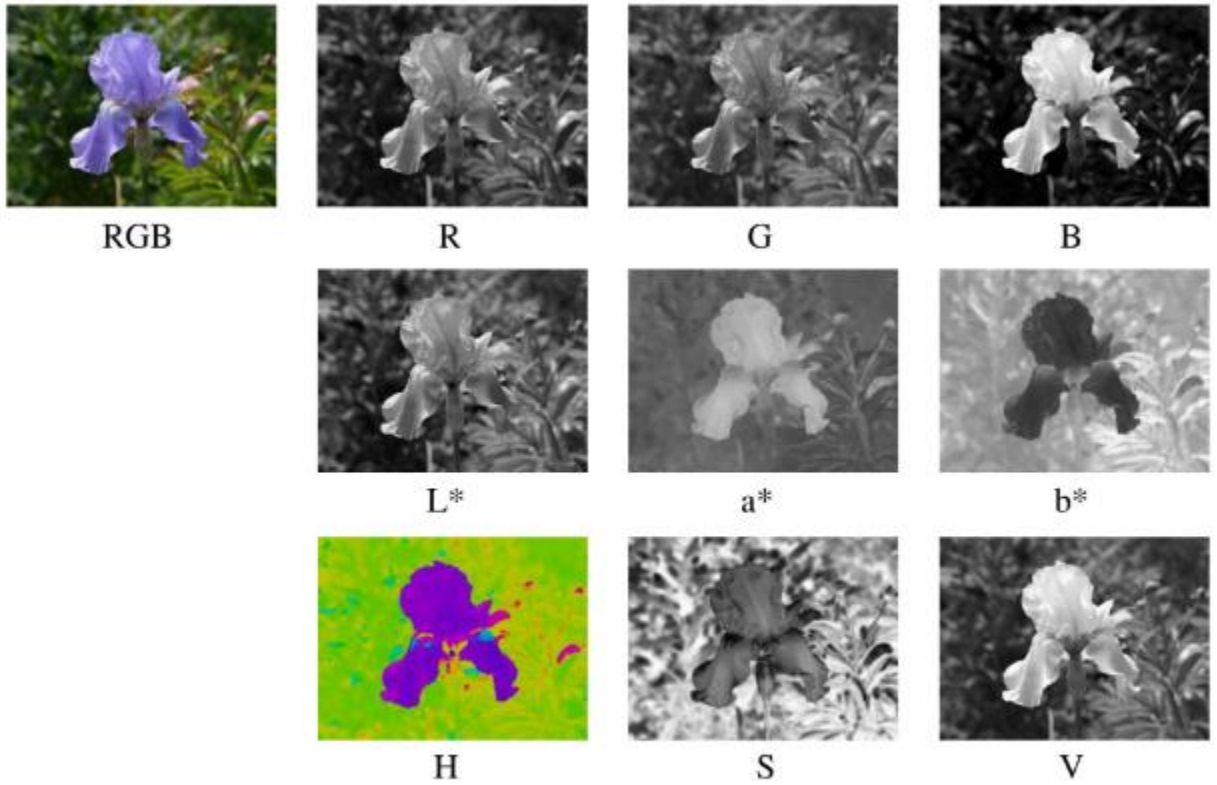


- Each pixel **integrates the light spectrum** L according to its spectral sensitivity S:

$$R = \int L(\lambda)\, S_R(\lambda)\, d\lambda$$

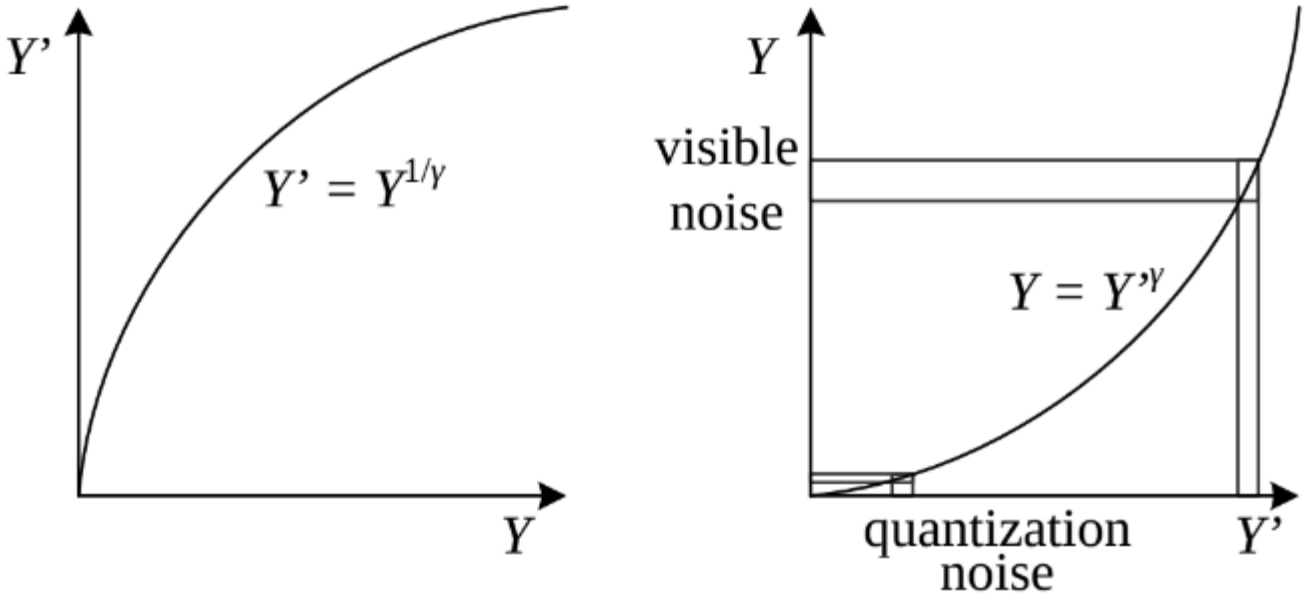
- The spectral response curves are provided by the camera manufacturer

Color Spaces



- Various different **color spaces** have been developed and are used in practice

Gamma Compression



- Humans are more sensitive to intensity differences in darker regions
- Therefore, it is beneficial to **nonlinearly transform** (left) the intensities or colors prior to discretization (left) and to undo this transformation during loading

Image Compression

- Typically luminance is compressed with higher fidelity than chrominance
- Often, (8×8 pixel) patch-based discrete cosine or wavelet transforms are used
- **Discrete Cosine Transform (DCT)** is an approximation to PCA on natural images
- The coefficients are quantized to integers that can be stored with Huffman codes
- More recently, deep network based compression algorithms are developed

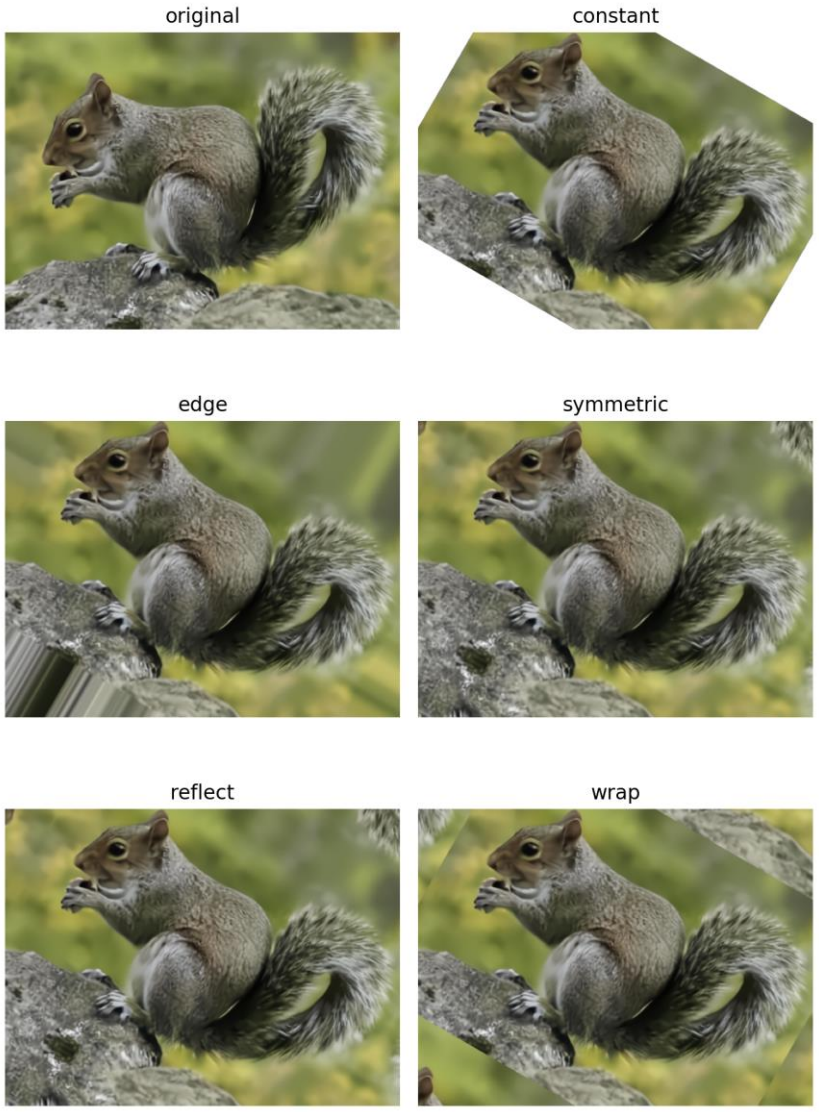
Python Code

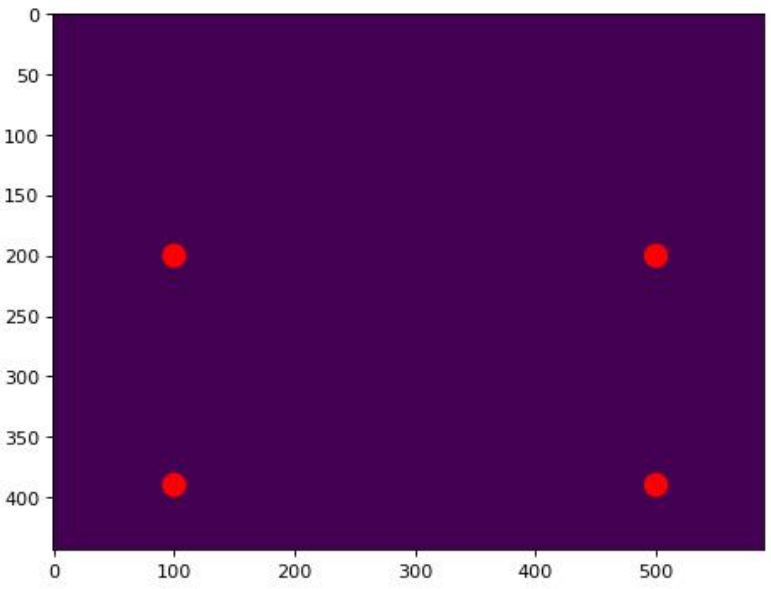
```python
#import packages
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.patches import Circle
from skimage import transform
from skimage.io import imread, imshow
#Read Image
sign = imread('0810.png')
imshow(sign)
#Rotate Image
sign_rotate = transform.rotate(sign, 330)
imshow(sign_rotate)

def rotate_fills(image):
    modes = ['constant', 'edge','symmetric','reflect','wrap']
    fig, ax = plt.subplots(3,2, figsize=(7, 10), dpi = 200)
    for n, ax in enumerate(ax.flatten()):
        n = n-1
        if n == -1:
            ax.set_title(f'original', fontsize = 12)
            ax.imshow(image)
            ax.set_axis_off()
        else:
            ax.set_title(f'{modes[n]}', fontsize = 12)
            ax.imshow(transform.rotate(image, 330, mode = modes[n]))
            ax.set_axis_off()

    fig.tight_layout();
rotate_fills(sign)

points_of_interest =[[360, 60],
                     [380, 270],
                     [130, 380],
                     [50, 150]]
projection = [[500, 200],
              [500, 390],
              [100, 390],
              [100, 200]]
color = 'red'
```

```python
fig, ax = plt.subplots(1,2, figsize=(15, 10), dpi = 80)
patch1 = Circle((points_of_interest[0][0],points_of_interest[0][1]),
                10, facecolor = color)
patch2 = Circle((points_of_interest[1][0],points_of_interest[1][1]),
                10, facecolor = color)
patch3 = Circle((points_of_interest[2][0],points_of_interest[2][1]),
                10, facecolor = color)
patch4 = Circle((points_of_interest[3][0],points_of_interest[3][1]),
                10, facecolor = color)
patch5 = Circle((projection[0][0],projection[0][1]), 10,
                facecolor = color)
patch6 = Circle((projection[1][0],projection[1][1]), 10,
                facecolor = color)
patch7 = Circle((projection[2][0],projection[2][1]), 10,
                facecolor = color)
patch8 = Circle((projection[3][0],projection[3][1]), 10,
                facecolor = color)
ax[0].add_patch(patch1)
ax[0].add_patch(patch2)
ax[0].add_patch(patch3)
ax[0].add_patch(patch4)
ax[0].imshow(sign);
ax[1].add_patch(patch5)
ax[1].add_patch(patch6)
ax[1].add_patch(patch7)
ax[1].add_patch(patch8)
ax[1].imshow(np.ones((sign.shape[0], sign.shape[1])));

points_of_interest = np.array([[360, 60],
                               [380, 270],
                               [130, 380],
                               [50, 150]])
projection = np.array([[500, 100],
                       [500, 300],
                       [100, 300],
                       [100, 100]])
```

```python

tform = transform.estimate_transform('projective', points_of_interest, projection)
tf_img_warp = transform.warp(sign, tform.inverse, mode = 'symmetric')
plt.figure(num=None, figsize=(8, 6), dpi=80)
fig, ax = plt.subplots(1,2, figsize=(15, 10), dpi = 80)
ax[0].set_title(f'Original', fontsize = 15)
ax[0].imshow(sign)
ax[0].set_axis_off();
ax[1].set_title(f'Transformed', fontsize = 15)
ax[1].imshow(tf_img_warp)
ax[1].set_axis_off();
```

Example from Python





Original

Transformed