Computer Vision
Spatial aspects in computer vision - lecture 10

Adam Szmigielski
aszmigie@pjwstk.edu.pl

materials: $ftp(public)://aszmigie/MWREnglish$

Geometric primitives - 2D points

Geometric primitives form the basic building blocks used to describe three-dimensional shapes.

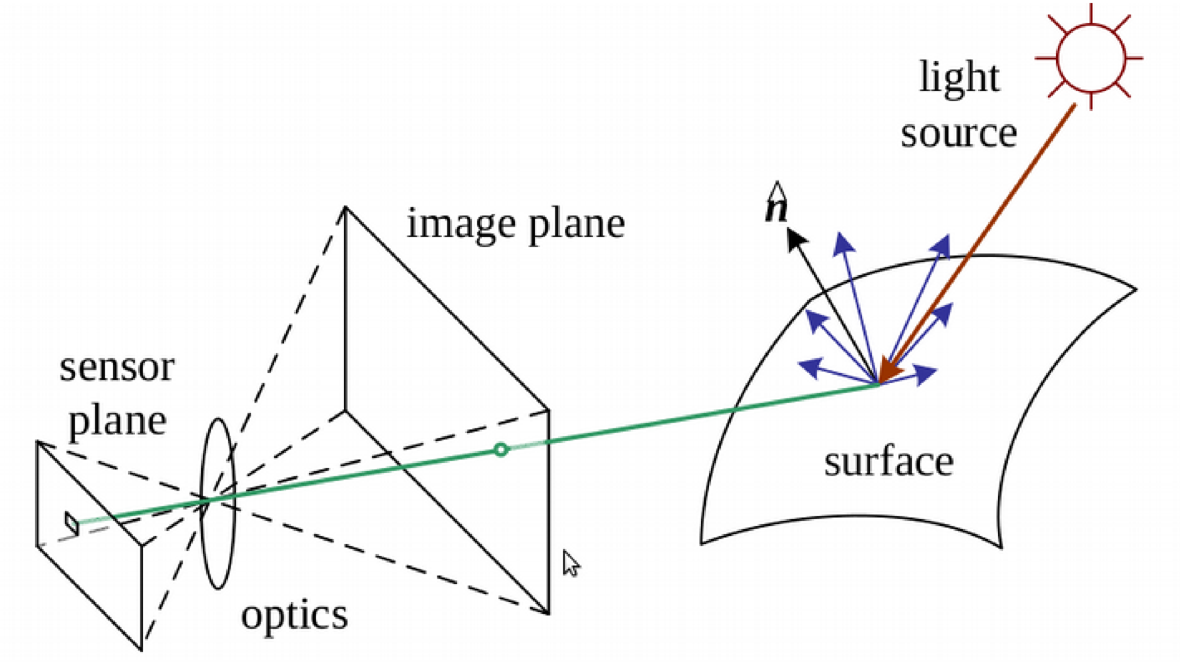
- **2D points** (pixel coordinates in an image) can be denoted using a pair of values, $x = (x, y) \in R^2$

  $2D$ points can also be represented using homogeneous coordinates,

  $$\tilde{x} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}\overline{x} \in P^2$$

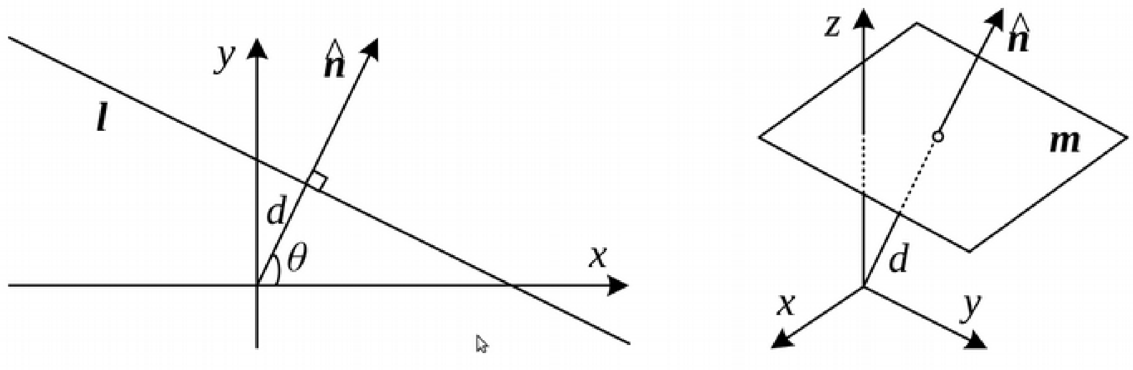
  where vectors $\tilde{x}$ that differ only by scale $\tilde{w}$ are considered to be equivalent, $\overline{x} = (x, y, 1)$ is the *augmented vector* and P^2 is called the $2D$ *projective space*.

Photometric image formation



- **Lighting** to produce an image, the scene must be illuminated with one or more light sources.
- Light sources can generally be divided into point and area light sources. A point light source originates at a single location in space, potentially at infinity (e.g., the sun).

Geometric primitives - 2D lines



(a) (b)

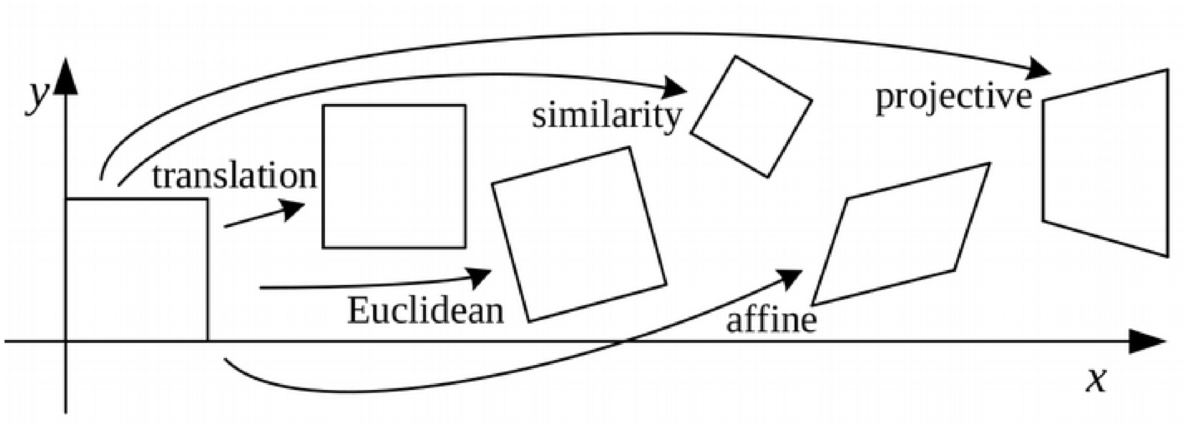
- **2D lines** can also be represented using homogeneous coordinates $\tilde{l} = (a, b, c)$. The corresponding line equation is

$$\tilde{x} \cdot \tilde{l} = ax + by + c = 0.$$

We can normalize the line equation vector so that $l = (\hat{n}_x, \hat{n}_y, d) = (\hat{n}, d)$ with $||\hat{n}|| = 1$ this case, $\hat{n}$ is the normal vector perpendicular to the line and d is its distance to the origin.

- The combination (Θ, d) is also known as *polar coordinates*.

2D transformations



- **2D translations** can be written as $x' = x + t$ or

$$x' = \left[\begin{array}{cc} I & t \end{array}\right] \cdot \bar{x},$$

where I is the 2×2 identity matrix.

- **Rotation + translation**. This transformation is also known as 2D rigid body motion or the 2D Euclidean transformation. It can be written as

$$x' = \begin{bmatrix} R & t \end{bmatrix} \overline{x}$$

where

$$R = \begin{bmatrix} \cos(\theta) & -sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

is an orthonormal rotation matrix with $R \cdot R^T = I$ and $|R| = 1$.

- **Scaled rotation**, also known as the similarity transform, this transformation can be expressed as $x' = sRx + t$ where s is an arbitrary scale factor. It can also be written as

$$x' = \begin{bmatrix} sR & t \end{bmatrix} \overline{x} = \begin{bmatrix} a & -b & t_x \\ b & a & t_y \end{bmatrix} \overline{x}$$

where we no longer require that $a^2 + b^2 = 1$. The similarity transform preserves angles between lines.

- **Affine transformation** is written as $x' = A\overline{x}$ where A is an arbitrary 2×3 matrix, i.e.,

$$x' = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{bmatrix} \overline{x}$$

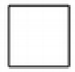
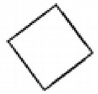
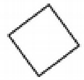
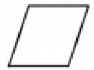
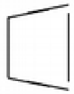
  Parallel lines remain parallel under affine transformations (does not keep angles between stight lines and distance between points)

- **Projective transformation**, also known as a perspective transform or homography, operates on homogeneous coordinates,

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

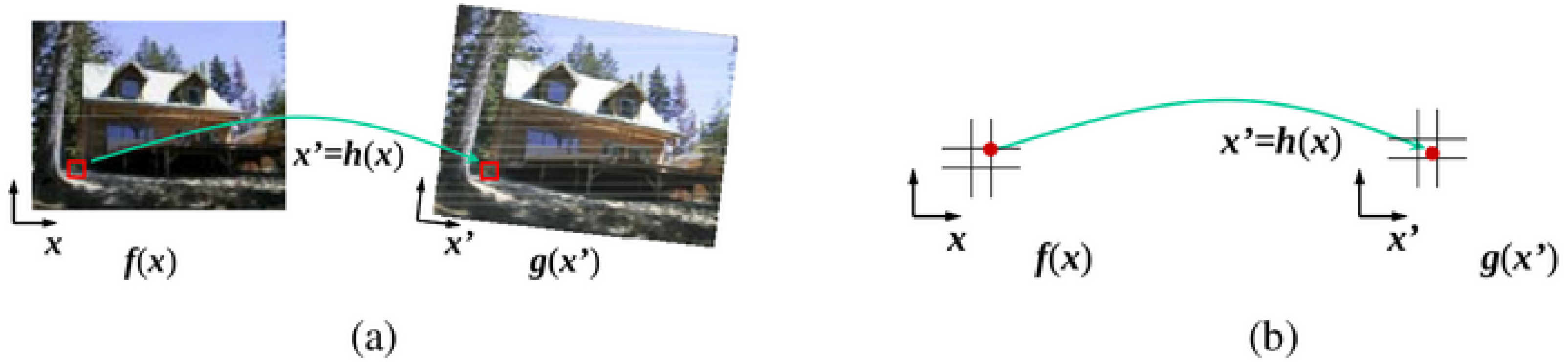
  where $\tilde{H}$ is an arbitrary 3×3 matrix. $\tilde{H}$ is only defined up to a scale, and that two $\tilde{H}$ matrices that differ only by scale are equivalent.

Hierarchy of 2D coordinate transformations

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

- Each transformation also preserves the properties (similarity preserves not only angles but also parallelism and straight lines).
- The 2×3 matrices are extended with a third $[0^T 1]$ row to form a full 3×3 matrix for homogeneous coordinate transformations.

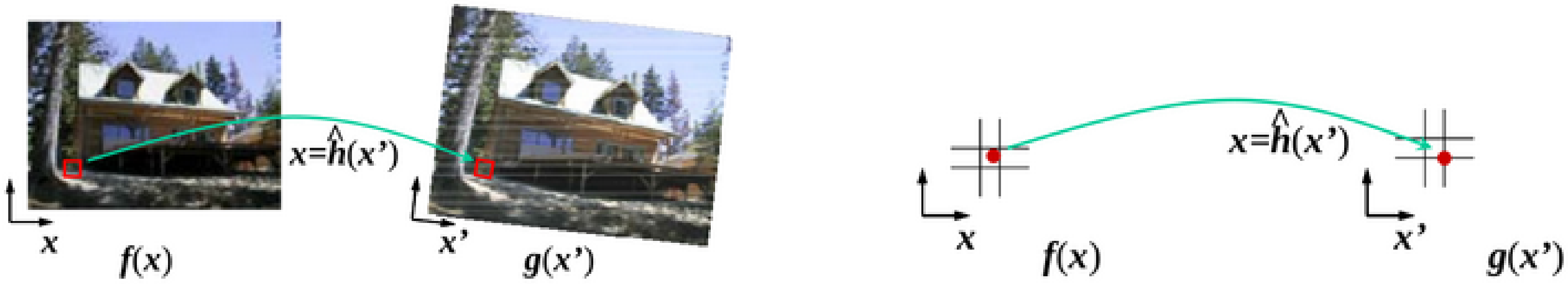
Forward warping (or mapping)



- (a) a pixel $f(x)$ is copied to its corresponding location $x' = h(x)$ in image $g(x')$;
- (b) detail of the source and destination pixel locations.
- Forward warping algorithm for transforming an image $f(x)$ into an image $g(x')$:

```
procedure forwardWarp(f, h, out g):
        For every pixel x in f(x)
                1. Compute the destination location x' = h(x).
                2. Copy the pixel f(x) to g(x').
```

Inverse warping (or mapping)



(a) (b)

- (a) a pixel $g(x')$ is sampled from its corresponding location $x = \hat{h}(x')$ in image $f(x)$;
- (b) detail of the source and destination pixel locations.
- Inverse warping algorithm for creating an image $g(x')$ from an image $f(x)$ using the parametric transform $x' = h(x)$.:

```
procedure inverseWarp(f, h, out g):
    For every pixel x' in g(x')
        1. Compute the source location x = h^(x')
        2. Resample f(x) at location x and copy to g(x')
```

2D and 3D feature-based alignment

Feature-based alignment is the problem of estimating the motion between two or more sets of matched 2D or 3D points.

Transform	Matrix	Parameters p	Jacobian J
translation	$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$	(t_x, t_y)	$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
Euclidean	$\begin{bmatrix} c_\theta & -s_\theta & t_x \\ s_\theta & c_\theta & t_y \end{bmatrix}$	(t_x, t_y, θ)	$\begin{bmatrix} 1 & 0 & -s_\theta x - c_\theta y \\ 0 & 1 & c_\theta x - s_\theta y \end{bmatrix}$
similarity	$\begin{bmatrix} 1+a & -b & t_x \\ b & 1+a & t_y \end{bmatrix}$	(t_x, t_y, a, b)	$\begin{bmatrix} 1 & 0 & x & -y \\ 0 & 1 & y & x \end{bmatrix}$
affine	$\begin{bmatrix} 1+a_{00} & a_{01} & t_x \\ a_{10} & 1+a_{11} & t_y \end{bmatrix}$	$(t_x, t_y, a_{00}, a_{01}, a_{10}, a_{11})$	$\begin{bmatrix} 1 & 0 & x & y & 0 & 0 \\ 0 & 1 & 0 & 0 & x & y \end{bmatrix}$
projective	$\begin{bmatrix} 1+h_{00} & h_{01} & h_{02} \\ h_{10} & 1+h_{11} & h_{12} \\ h_{20} & h_{21} & 1 \end{bmatrix}$	$(h_{00}, h_{01}, \ldots, h_{21})$	

- For simplicity we reduce problem to global *parametric transformations*,
- Jacobians of the 2D coordinate transformations $x' = f(x; p)$.

2D alignment using least squares

Given a set of matched feature points $\{x_i, x_i'\}$ and a planar parametric transformation of the form $x' = f(x; p)$
How can we estimate the parameters p?

- The usual way is least squares (LS to minimize the sum of squared errors)

$$E_{LS} = \sum_i ||r_i||^2 = \sum_i ||f(x_i; p) - x_i'||^2,$$

where

$$r_i = f(x_i; p) - x_i' = \hat{x_i}' - \tilde{x_i}'$$

is the residual between the measured location $\hat{x_i}'$ and its corresponding predicted location $\tilde{x_i}' = f(x_i; p)$.

Linear motion model

For linear operations (translation, similarity, and affine) relationship between the amount of motion $\triangle x = x' - x$ and the unknown parameters p,

$$\triangle x = x' - x = J(x)p,$$

where $J = \frac{\partial f}{\partial p}$ is the Jacobian of the transformation f with respect to the motion parameters p.

- In linear least squares problem:

$$E_{LS} = \sum_i ||J(x_i)p - \triangle x_i'||^2 = p^T A p - 2p^T b + c$$

  where $A = \sum_i J^T(x_i)J(x_i)$ is the Hessian and $b = \sum_i J^T(x_i) \triangle x_i$

- The minimum can be found by solving

$$A \cdot p = b$$

Robust least squares

More robust version of LS required analysis of outliers between the correspondences:

- In this case, it is preferable to use an M-estimator, which involves applying a robust penalty function $\rho(r)$ to the residuals:

$$E_{RLS}(\triangle p) = \sum_i \rho(||r_i||)$$

- Instead of squaring them.We can take the derivative of this function with respect to p and set it to 0

$$\sum_i \psi(||r_i||) \frac{\partial ||r_i||}{\partial p} = \sum_i \frac{\psi(||r_i||)}{||r_i||} r_i^T \frac{\partial r_i}{\partial p} = 0$$

where $\psi(r) = \rho'(r)$ is the derivative of ρ and is called the *influence function.*

Iteratively Reweighted Least Squares (IRLS)

- If we introduce **weight function** $w(r) = \frac{\psi(r)}{r}$,
- We observe that finding the stationary point of $E_{RLS}(\triangle p) = \sum_i \rho(||r_i||)$ is equivalent to minimizing IRLS:

$$E_{IRLS} = \sum_i w(||r_i||)||r_i||^2,$$

where the $w(||r_i||)$ is inverse proportional to squared standard deviation $\sim \frac{1}{\sigma_i^2}$.

RANdom SAmple Consensus - RANSAC

A better approach is to find correspondences with a dominant motion estimate.

- It starts by selecting (at random) a subset of k correspondences, which is used to compute an initial estimate for p.
- The residuals of the full set of correspondences are then computed as:

$$r_i = \tilde{x}_i{}'(x_i; p) - \hat{x}_i{}'$$

  where $\tilde{x}_i{}'$ are the estimated (mapped) locations and $\hat{x}_i{}'$ are the sensed (detected) feature point locations.

- The RANSAC technique counts the number of inliers that are within ϵ of their predicted location (whose $||r_i|| \leq \epsilon$).

Implementation of RANSAC

- The value of ϵ is application dependent (often around 1–3 pixels),
- Least median of squares finds the median value of the $||r_i||^2$ values,
- The random selection process is repeated S times and the sample set with the largest number of inliers (or with the smallest median residual) is kept as the final solution.
- The initial parameter p is passed on to the next data fitting stage.
- When the number of measurements is large the most plausible points can be selected.

Implementation of RANSAC - number of trials

- To ensure that the random sampling has a good chance of finding a true value we have to make the sufficient number of trials S.
- The likelihood in one trial that all k random samples are inliers is p_k .
- Let p be the probability of valid correspondence and P the total probability of success after S trials. The likelihood that S such trials will all fail is

$$1 - P = (1 - p^k)^S$$

- and the required minimum number of trials is

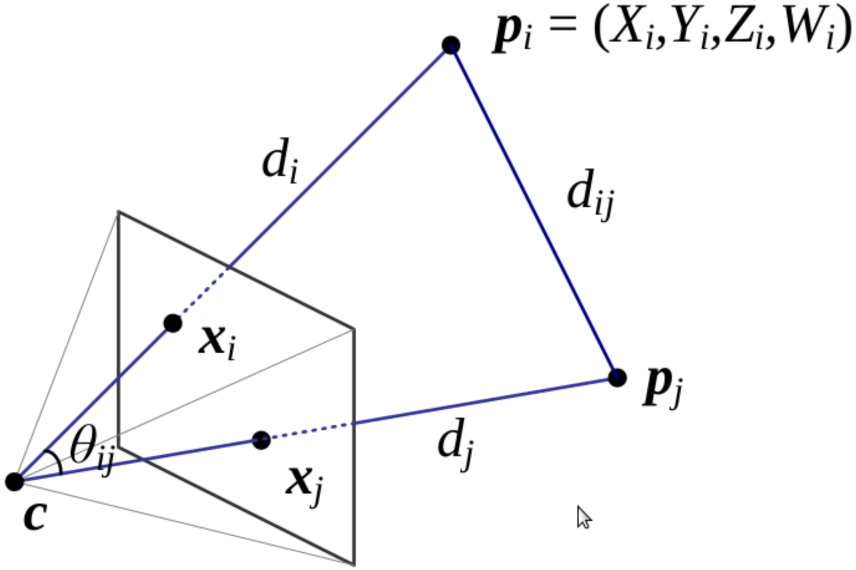
$$S = \frac{log(1 - P)}{log(1 - p^k)}$$

Pose estimation

A particular instance of feature-based alignment, which occurs very often, is estimating an object's 3D pose from a set of 2D point projections.

- This **pose estimation** problem is also known as **extrinsic calibration** (as opposed to the *intrinsic calibration* of internal camera parameters such as focal length etc.)
- The problem of recovering pose from three correspondences is known as the perspective-3-point-problem (P3P)

Linear algorithms



The simplest way to recover the pose of the camera is to form a set of linear equations from the camera matrix of perspective projection,

$$x_i = \frac{p_{00}X_i + p_{01}Y_i + p_{02}Z_i + p_{03}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$
$$y_i = \frac{p_{10}X_i + p_{11}Y_i + p_{12}Z_i + p_{13}}{p_{20}X_i + p_{21}Y_i + p_{22}Z_i + p_{23}}$$

where (x_i, y_i) are the measured $2D$ feature locations and (X_i, Y_i, Z_i) are the known $3D$ feature locations.

Linear algorithms

- In order to compute the 12 unknowns in P, at least six correspondences between $3D$ and $2D$ locations must be known.
- Once the entries in P have been recovered, it is possible to recover both the intrinsic calibration matrix K and the rigid transformation (R, t):

$$P = K[R|t].$$

- Since K is by convention upper-triangular, both K and R can be obtained from the front 3×3 sub-matrix of P using RQ factorization,
- In the case when the camera is already calibrated the matrix K is known,
- Visual angle θ_{ij} between any pair of $2D$ points $\hat{x_i}$ and $\hat{x_j}$ must be the same as the angle between their corresponding $3D$ points p_i and p_j.

Linear algorithms

- Given a set of corresponding $2D$ and $3D$ points $\{(\hat{x_i}, pi)\}$, where the $\hat{x_i}$ are unit directions obtained by transforming $2D$ pixel measurements x_i to unit norm 3D directions $\hat{x_i}$ through the inverse calibration matrix K

$$\hat{x_i} = \frac{K^{-1} x_i}{||K^{-1} x_i||},$$

  the unknowns are the distances d_i from the camera origin c to the $3D$ points p_i, where

$$p_i = d_i \hat{x_i} + c$$

- Once the individual estimates of the d_i distances we can generate a $3D$ structure consisting of the scaled point directions $d_i \hat{x_i}$, to obtained the desired pose estimate.

Iterative algorithms

The most accurate way to estimate pose is to directly minimize the squared reprojection error for the 2D points using non-linear least squares.

- We can write the projection equations as

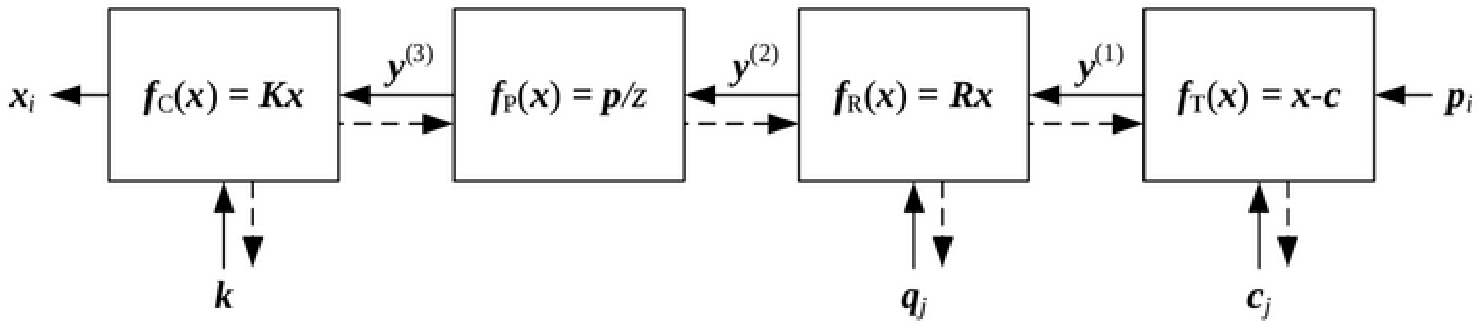
$$x_i = f(p_i; R, t, K)$$

and iteratively minimize the linearized reprojection errors:

$$E_{NLP} = \sum_i \rho\left(\frac{\partial f}{\partial R} \triangle R + \frac{\partial f}{\partial t} \triangle t + \frac{\partial f}{\partial K} \triangle K - r_i\right)$$

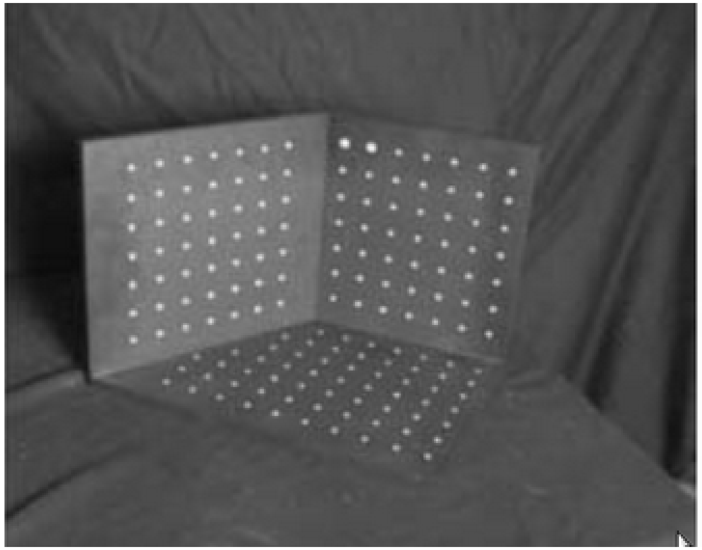
where $r_i = \tilde{x}_i - \hat{x}_i$ is 2D error in predicted position.

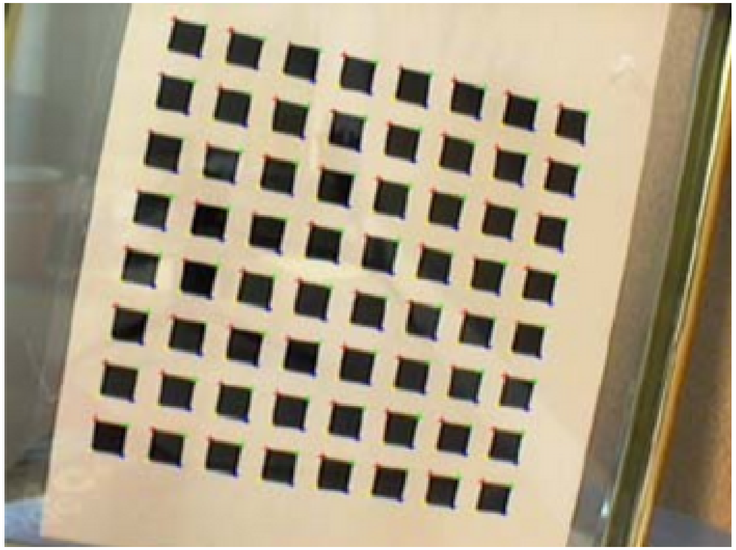
Implementation of iterative algorithms



- A set of chained transforms for projecting a $3D$ point p_i to a $2D$ measurement x_i through a series of transformations $f^{(k)}$, each of which is controlled by its own set of parameters.
- The dashed lines indicate the flow of information as partial derivatives are computed during a backward pass.

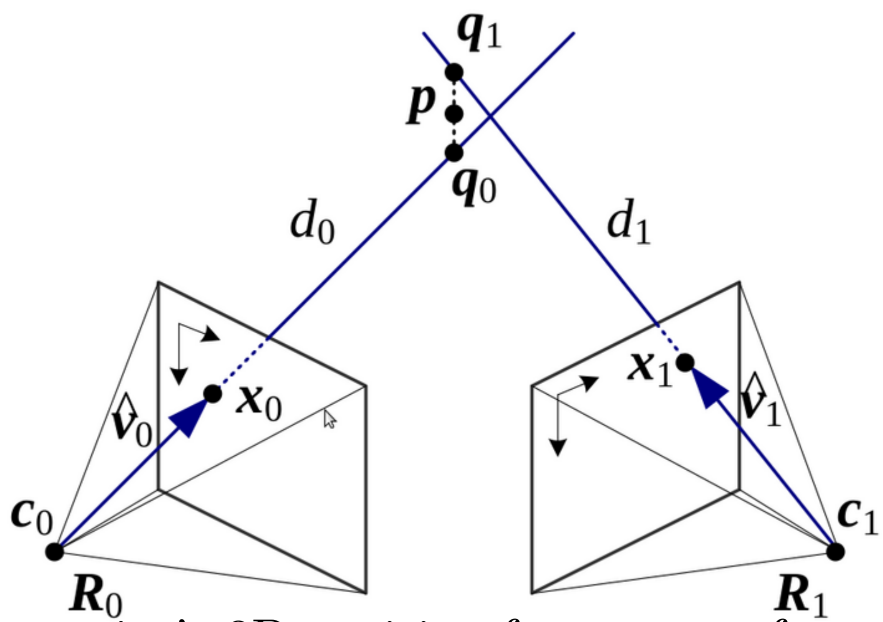
Geometric intrinsic calibration





- **Calibration patterns -** use of a calibration pattern or set of markers is one of the more reliable ways to estimate a camera's intrinsic parameters,
- **Planar calibration patterns -** a good way to perform calibration is to move a planar calibration target in a controlled fashion through the workspace volume.

Triangulation



The problem of determining a point's 3D position from a set of corresponding image locations and known camera positions is known as *triangulation.*

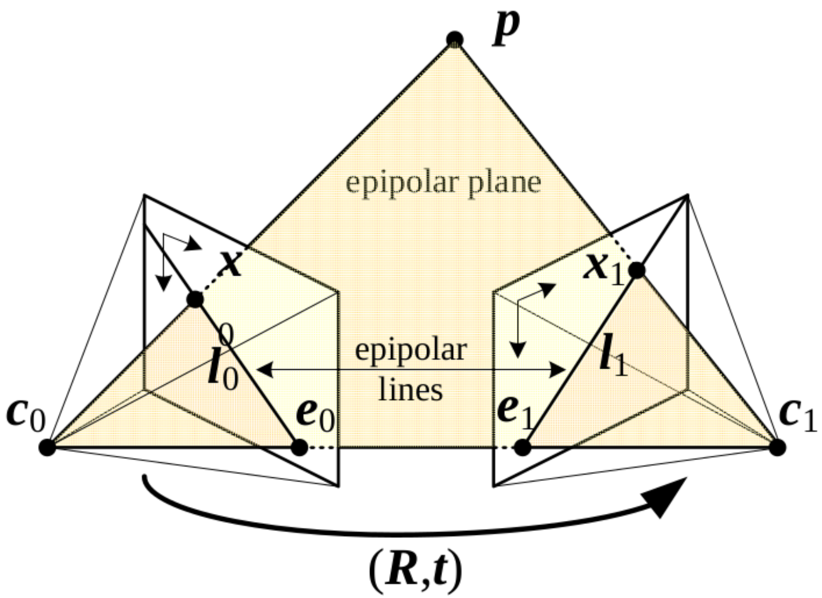
- This problem is the reverse to the pose estimation,
- To solve this problem we need to find the $3D$ point p that lies closest to all of the $3D$ rays corresponding to the $2D$ matching feature locations $\{x_j\}$ observed by cameras $\{P_j = K_j[R_j|t_j]\}$, where $t_j = -R_j c_j$ and c_j is the jth camera center.

Triangulation

- The nearest point to p on this ray, which we denote as q_1 , minimizes the distance $||c_1 + d_1 \hat{v}_1 - p||^2$,
- The optimal value for p, which lies closest to all of the rays, can be computed as a regular least squares problem by summing over all the r_j^2 and finding the optimal value of p:

$$p = [\sum_i (I - \hat{v}_j \hat{v}_j{}^T)]^{-1} [\sum_i (I - \hat{v}_j \hat{v}_j{}^T) c_j].$$

Two-frame structure from motion



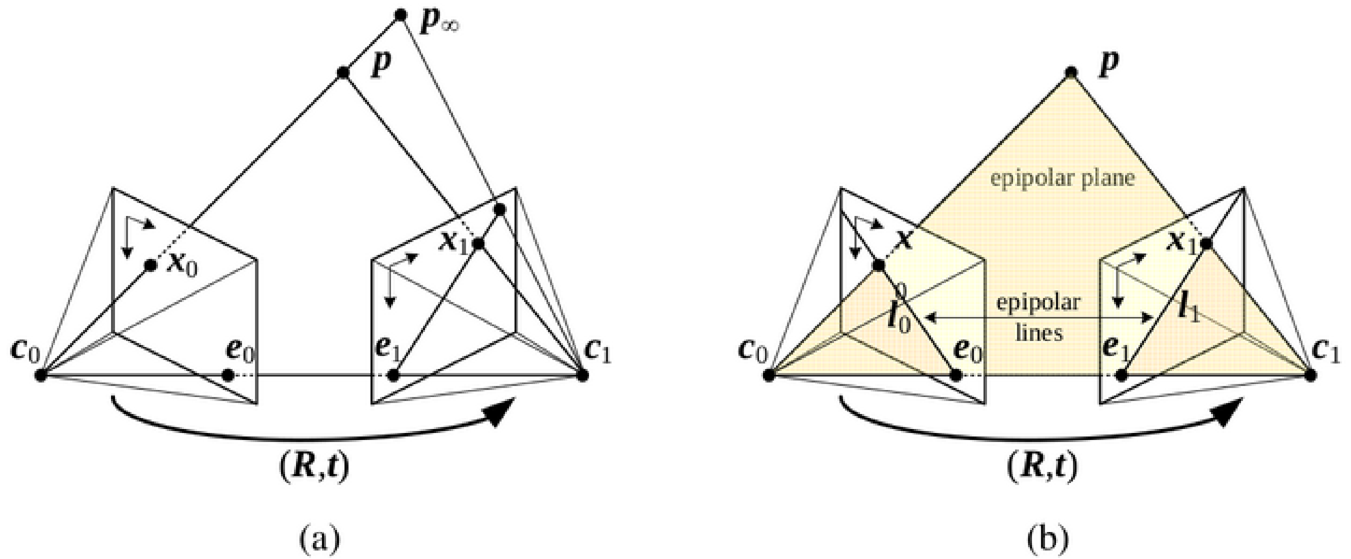
In that task we simultaneous recovery of $3D$ structure and pose from image correspondences.

- Relative position can be encoded by a rotation R and a translation t,
- The vectors $t = c_1 - c_0$, $p - c_0$ and $p - c_1$ are co-planar and define the basic epipolar constraint expressed in terms of the pixel measurements x_0 and x_1.

Stereo matching

Stereo matching is the process of taking two or more images and estimating a $3D$ model of the scene by finding matching pixels in the images and converting their $2D$ positions into $3D$ depths.

Epipolar geometry



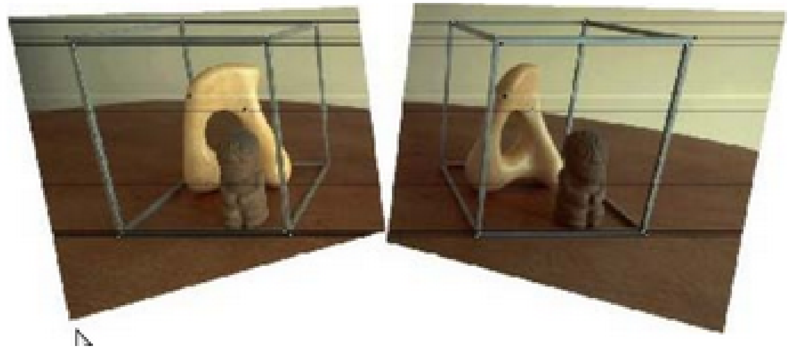
- (a) epipolar line segment corresponding to one ray. Pixel x_0 in one image projects to an epipolar line segment in the other image.
- (b) corresponding set of epipolar lines and their epipolar plane.

Rectification

The epipolar geometry for a pair of cameras is implicit in the relative pose and calibrations of the cameras, and can be computed from point matches.

- One way to do this is to use a general correspondence algorithm, such as *optical flow*.
- A more efficient algorithm can be obtained by rectifying the input images so that corresponding horizontal scanlines are epipolar lines,
- Afterwards, it is possible to match horizontal scanlines independently or to shift images horizontally while computing matching scores.

Rectification - example



- A simple way to rectify the two images is to first rotate both cameras so that they are looking perpendicular to the line joining the camera centers c_0 and c_1,
- Next, to determine the desired twist around the optical axes, make the up vector perpendicular to the camera center line.
- Finally, re-scale the images, if necessary, to account for different focal lengths,

Standard rectified geometry

is employed in a lot of stereo camera setups and stereo algorithms, and leads to a very simple inverse relationship between 3D depths Z and disparities d:

$$d = f\frac{B}{Z}$$

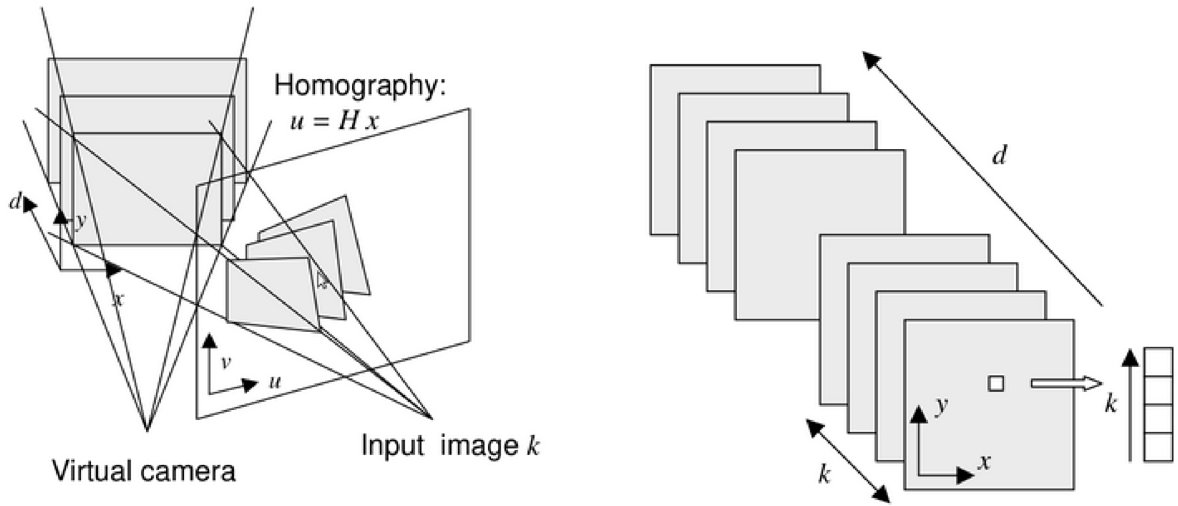
where f is the focal length (measured in pixels), B is the baseline.

- The relationship between corresponding pixel coordinates in the left and right images:

$$x' = x + d(x, y), y' = y$$

- The task of extracting depth from a set of images then becomes one of estimating the **disparity map** $d(x, y)$.
- After rectification, we can easily compare the similarity of pixels at corresponding locations (x, y) and $(x', y') = (x + d, y)$.

Plane sweep



An alternative to pre-rectifying the images before matching is to **sweep a set of planes** through the scene and to measure the photoconsistency of different images as they are re-projected onto these planes

- The set of planes seen from a virtual camera induces a set of homographies in any other source (input) camera image.
- The warped images from all the other cameras can be stacked into a generalized disparity space volume $\tilde{I}(x, y, d, k)$, disparity d of camera k, (x, y) - pixel location

3D to 2D projections

We can do this using a linear $3D$ to $2D$ projection matrix.

- An orthographic projection simply drops the z component of the three-dimensional coordinate p to obtain the $2D$ point x.

$$x = [I_{2\times 2}|0]p$$

- Often image need to be scaled by s to fit onto an image sensor (eg. cm to pixels). For this reason, scaled orthography is actually more commonly used,

$$x = [sI_{2\times 2}|0]p$$

Perspective

The most commonly used projection in computer graphics and computer vision is true $3D$ perspective

- Here, points are projected onto the image plane by dividing them by their z component
- In homogeneous coordinates, the projection has a simple linear form

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{p}$$

- Into normalized device coordinates in the range $(x, y, z) \in [-1, -1] \times [-1, 1] \times [0, 1]$, and then rescales these coordinates to integer pixel coordinates using a viewport transformation
- The (initial) perspective projection is then represented using a 4×4 matrix

$$\tilde{x} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-z_{far}}{z_{range}} & \frac{z_{near} \cdot z_{far}}{z_{range}} \\ 0 & 0 & 1 & 0 \end{bmatrix} \tilde{p}$$

  where z_{near} and z_{far} are the near and far z *clipping planes* and $z_{range} = z_{far} - z_{near}$.
- If we set $z_{near} = 1$, $z_{far} = \inf$, the third element of the normalized screen vector becomes the inverse depth (*disparity*)
- It is then convenient to be able to map disparity value d directly back to a $3D$ location using the inverse of a 4×4 matrix.

- We can do this if we represent perspective projection using a full-rank 4×4 matrix:

$$\tilde{P} = \begin{bmatrix} K & 0 \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} = \tilde{K} \cdot E$$

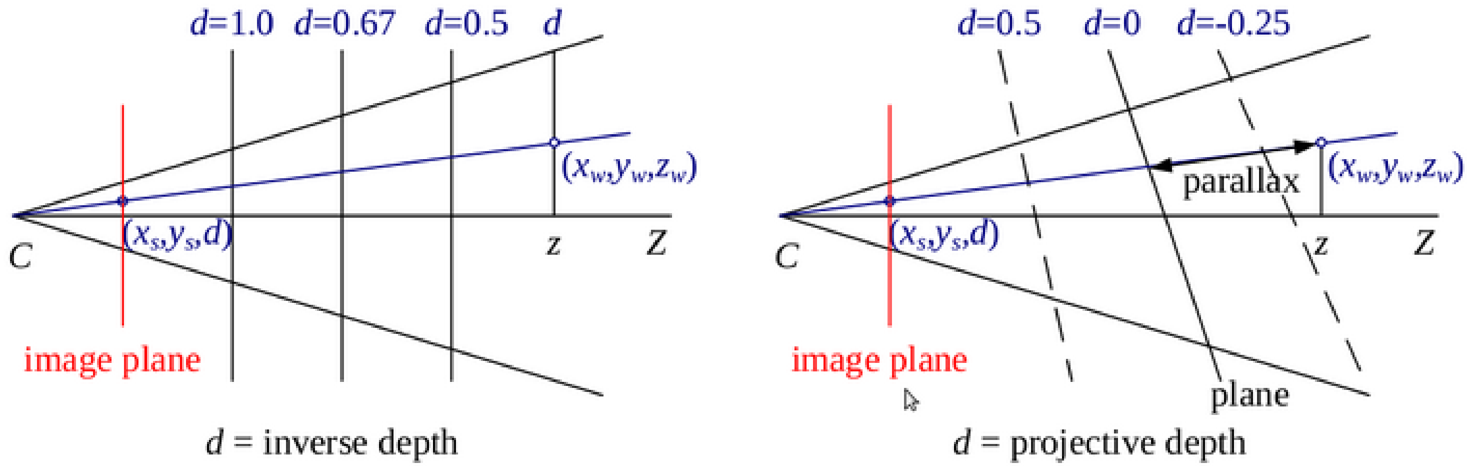
where E is a $3D$ rigid-body (Euclidean) transformation and $\tilde{K}$ is the full-rank calibration matrix.

- The 4×4 camera matrix $\tilde{P}$ can be used to map directly from $3D$ world coordinates $\overline{p_w} = (x_w, y_w, z_w, 1)$ to screen coordinates (plus disparity), $x_s = (x_s, y_s, 1, d)$,

$$s_s \sim \tilde{P}\overline{p_w}$$

where $\sim$ indicates equality up to scale.

Plane plus parallax (projective depth)

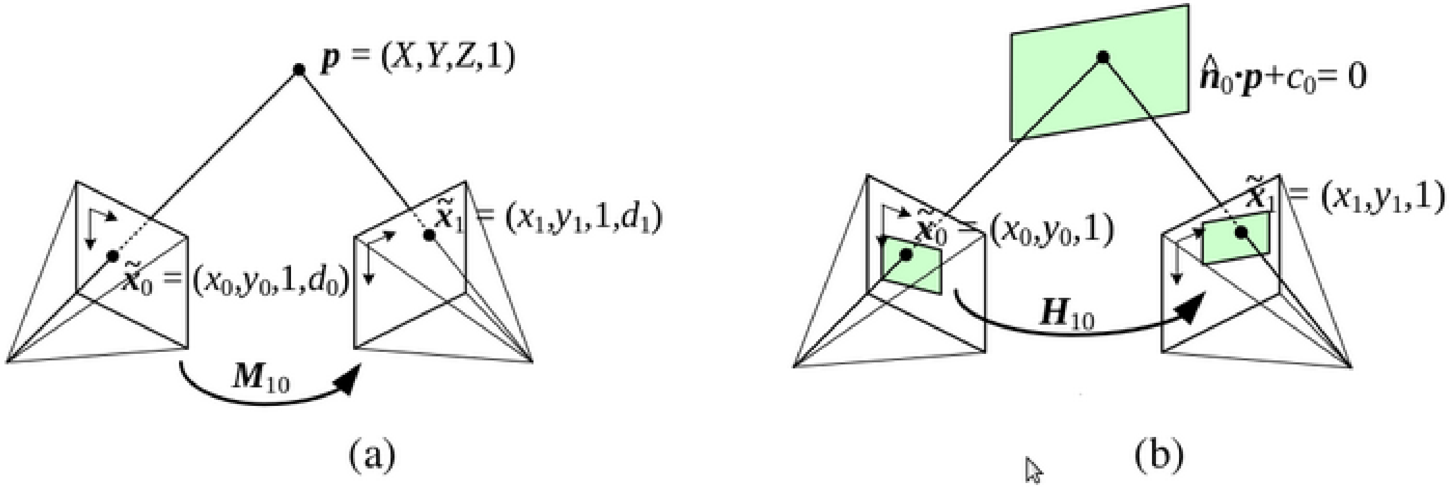


- In general, when using the 4×4 matrix $\tilde{P}$ we have the freedom to remap the last row to whatever suits our purpose,
- Let the last row of $\tilde{P}$ be $p_3 = s_3[\hat{n_0}|c_0]$, where $||\hat{n_0}|| = 1$, then we have:

$$d = \frac{s_3}{z}(\hat{n_0} p_w + c_0),$$

where $z = p_2 \cdot \overline{p_w} = r_z \cdot (p_w - c)$ is the distance of p_w from the camera center C along the optical axis Z.

Mapping from one camera to another



- Using the full rank 4×4 camera matrix $\tilde{P} = \tilde{K}E$ we can write the projection from world to screen coordinates as

$$\tilde{x_0} \sim \tilde{K_0}E_0 p = \tilde{P_0}p.$$

- Assuming that we know disparity value d_0 for a pixel in one image, we can compute the $3D$ point location p using

$$p \sim E_0^{-1}\tilde{K_0}^{-1}\tilde{x_0}.$$