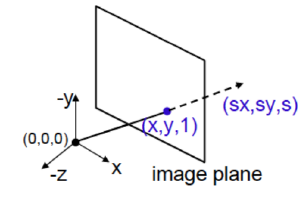
**Homography estimation**

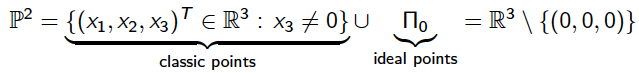
**Projective geometry and transformations:**

* 3D world are transformed into image objects through a projective transformation (multiplication by a matrix).
* **Image capture** 🡪 linear transformation.
* Intersection of two lines and the line that passes through two points is a linear operation.
* **Points at infinity** 🡪 natural representation.



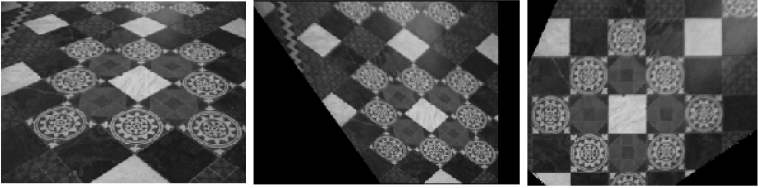
*Simulation of a projection of the real world in a plane*

* The plane can be at different distances, for that reason the points are equivalent if we multiply the homogeneous coordinates by a scalar.
* **Projective plane** 🡪 formed by the classical points (the projections) and the ideal points.
  + Classical points 🡪 those that have all the coordinates different from 0.
  + Ideal points 🡪 points at infinity, those that have the third coordinate equal to 0.



* The intersection of a line with the line at infinity is a point at infinity (b,-a,0)T.
* **2D transformation allows:**
  + Remove the projective distortion
  + Build image mosaics

**Affine and metric rectification:**



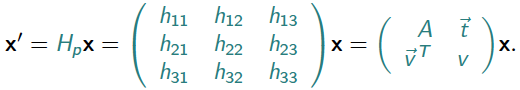
**First** 🡪 original image

**Second** 🡪 affine image, this image have a distorsion. Keeps paralelisms

but not angles. Lenghts are keept.

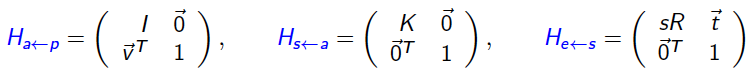
**Third** 🡪 perspective image. Lengths, angles, parallelisms are not kept.

* **Perspective transformation (planar transformation):**
  + 2D projective transformation.
  + Hp:
    - Matrix of the perspective transformation.
    - Non-singular matrix.



* **Projective matrix can be decomposed into:**
  + Matrix that transform from projective to affine space
  + Matrix that transform from affine to isometric space
  + Matrix that transform from isometric to Euclidean space

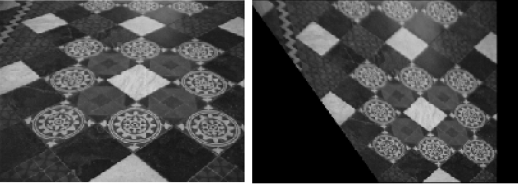




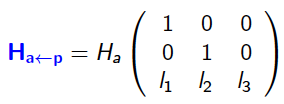


*Composing the matrixes we obtain that expression*

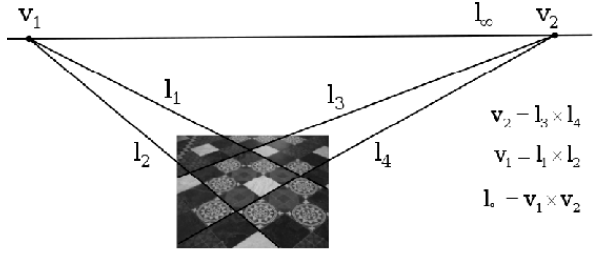
* **From projective to affine:**



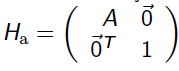
* + Parallelism will be recovered.
  + The fact is that points at infinity weren’t at infinity.
  + Trick: parallel lines meets at infinity (vanishing points).
    - Map ideal point to infinity.
    - Parallel lines meets at a point at infinity (vanishing point).
    - In a projective image parallel lines are not parallel, so if we know two lines are parallel we can compute the meeting point, mapping it to infinity (natural expression).
    - Having two points at infinity we can compute the correspondent line at infinity.
    - Matrix transformation is written as a 3x3 matrix with a 2x2 identity matrix where the last row corresponds to a line that is not at infinity but must be at infinity.
  + Basic idea:
    - In a projection ideal point is closest to the image plane.
    - The cross product of two points are a line, so the cross product of two ideal points is the line at infinity.
    - This line has to be (0,0,1)T (or a scaled version).
    - Due to the projection a line that should be at infinity won’t.
    - By a transformation matrix we can move the line to infinity.



* + Algorithm:
    - Take two parallel lines.
    - Compute the vanishing point by the cross product of the lines.
    - Take two points, and compute the line that joints them by computing the cross product.
    - Built the matrix.



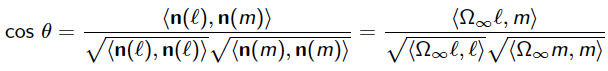
* **Metric rectification:**
  + Now we have parallel lines, but we want to recover the metric properties (angles,…).
  + We assume the image has been affinely rectified.
  + Matrix Ha represents the affine transformation of the isometric image.



* + In this step the infinity line is at infinity (affine transformation was applied).
  + Trick: angles in the image must be equal to the angles of the real world.
    - We can impose that an angle in the image is 90º (because we know) and apply a transformation.

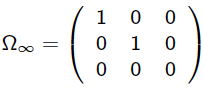


* + Basic idea:
    - Having two lines: ‘l’ and ‘m’, the angle between them can be computed with the dot (scalar) product of them.
    - **n(l)** 🡪 normal vector to ‘l’, the Euclidean coordinates of the line.



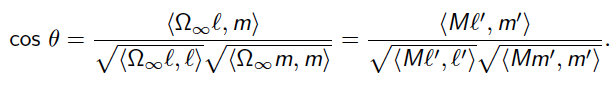
*Compute the angle in the Euclidean representation,*

*without transformation*



*This last matrix delete the third coordinate*

* + - If we have a transformation of the Euclidean lines, the expression for the angle computation change:

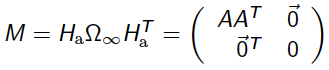


*For points in the distorted plane (not Euclidean plane)*



*Transformation applied*

* + Given a affine matrix transformation Ha:
    - A matrix M can be described as:

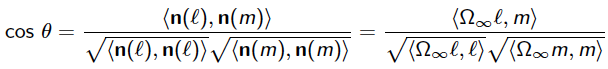




H is a similarity transformation



* + Algorithm:
    - Having ‘l’ and ‘m’ two lines that are orthogonal in the real world, the angle between them are 90º, following the next equation:



*Has to be equal to 0*

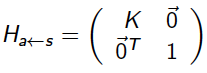




*s=(s1,s2,s3)T*

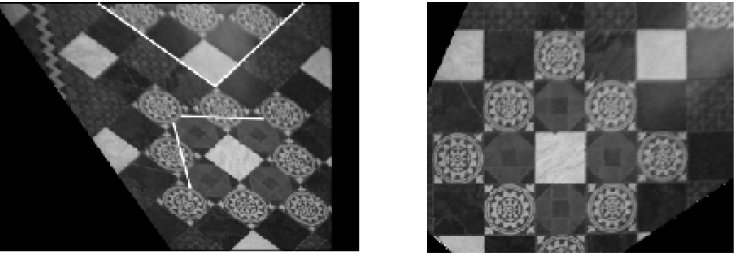


* + - With these two lines we can build two equations allowing us to compute ‘s’.
    - With the matrix S 🡪 using Cholesky decomposition we found an upper triangular matrix K such that: S=KKT
    - K can be a possible matrix A that can be used to metrically rectify the image.





*Matrix to transform from affine to isometric image*

**

*Highlighted, lines used to compute the matrix*