Visione Artificiale (Computer Vision)

Liliana Lo Presti

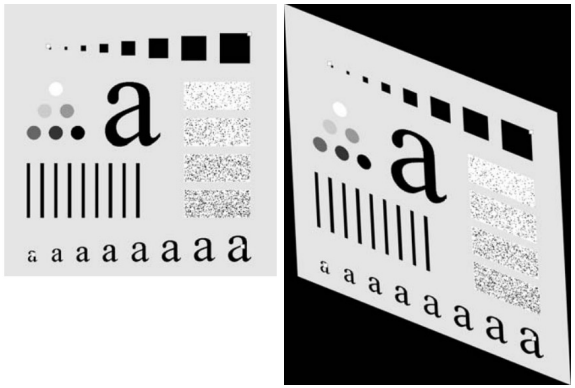
Image geometric transformation

- Geometric transformations modify the spatial relationship between pixels in an image.
- A geometric transformation consists of two basic operations:

  (1) a spatial transformation of coordinates

  (2) intensity interpolation

  it assigns intensity values to the spatially transformed pixels.



Spatial Transformation of Coordinates

- The transformation of coordinates may be expressed as

$$(x, y) = T\{ (v, w) \}$$

- where (v, w) are pixel coordinates in the original image and (x, y) are the corresponding pixel coordinates in the transformed image.
- For example, the transformation

$$(x, y) = T\{ (v, w) \} = (v/2, w/2)$$

- shrinks the original image to half its size in both spatial directions.
- One of the most commonly used spatial coordinate transformations is the affine transform

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ w \\ 1 \end{bmatrix}$$

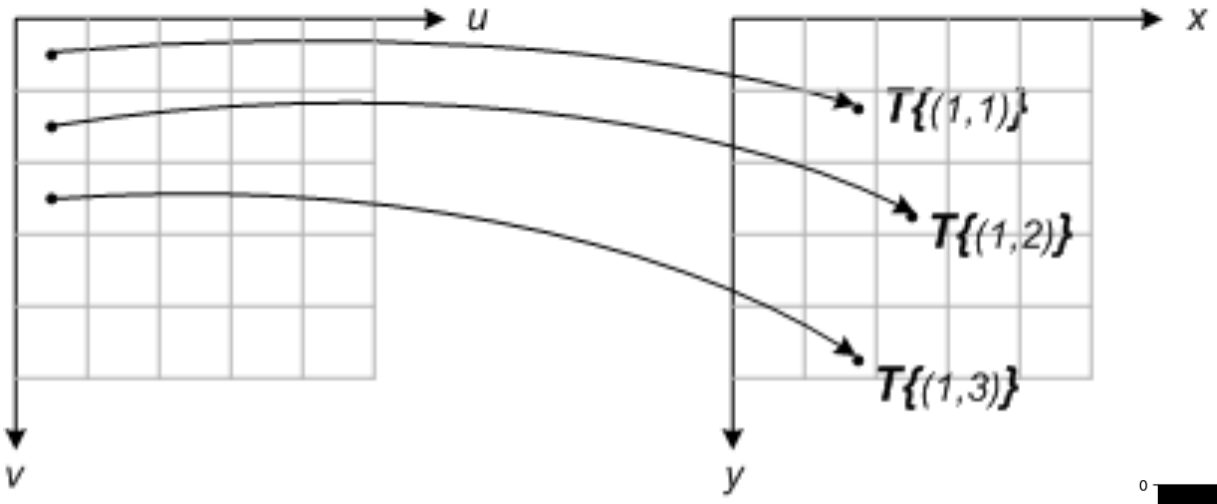
Spatial Transformation of Coordinates

- Geometric transformations relocate pixels on an image to new locations. Intensity values to those locations are assigned by using intensity interpolation.
- The equation of a geometric transformation T can be used in two basic ways:
  - **forward mapping:** it consists of scanning the pixels (v, w) of the input image and, at each location, computing the spatial location (x, y) of the corresponding pixel in the output image.

    There are 2 main issues:
    - When two or more pixels in the input image can be transformed to the same location in the output image, how should we combine multiple output values into a single output pixel?
    - Some output locations may not be assigned a pixel at all! Unassigned pixels are present in the image
  - **inverse mapping:** it scans the output pixel locations (x,y) and computes the corresponding location (v, w) in the input image using the inverse transformation T^{-1}. Then, it interpolates among the nearest input pixels to determine the intensity of the output pixel value.

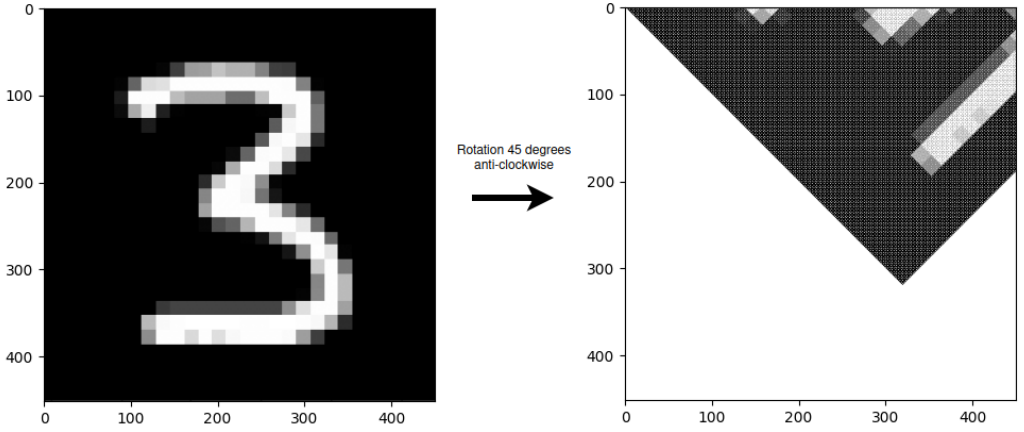
    Inverse mappings are more efficient to implement than forward mappings and are used in numerous commercial implementations of spatial transformations

Forward Mapping

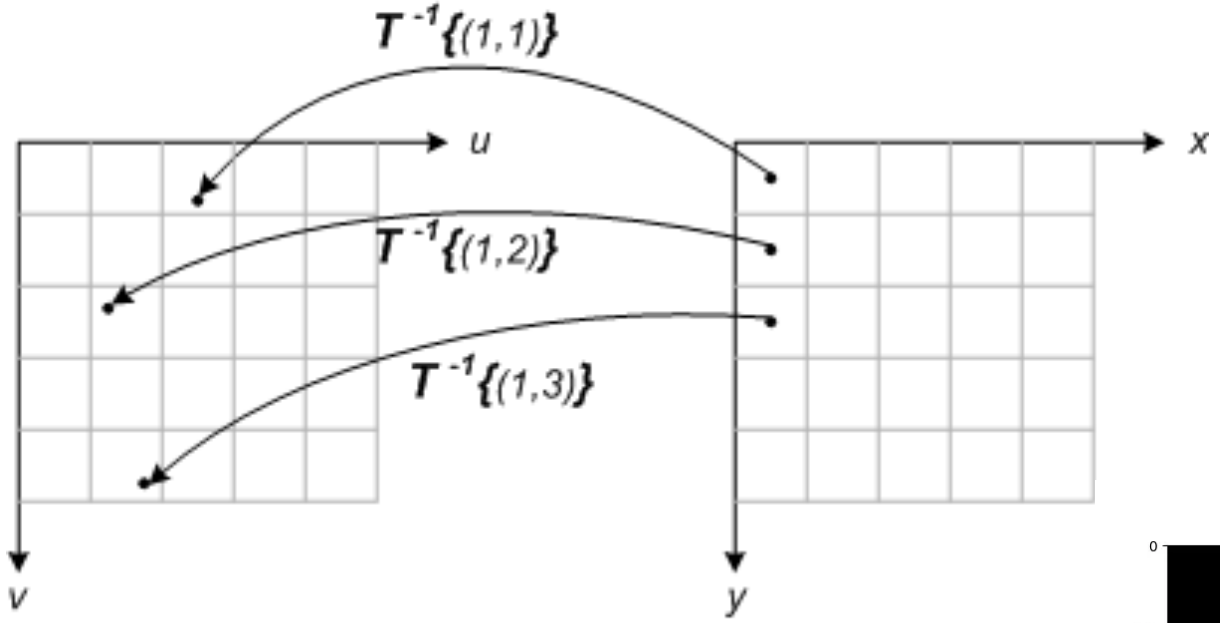


Input space

Output space

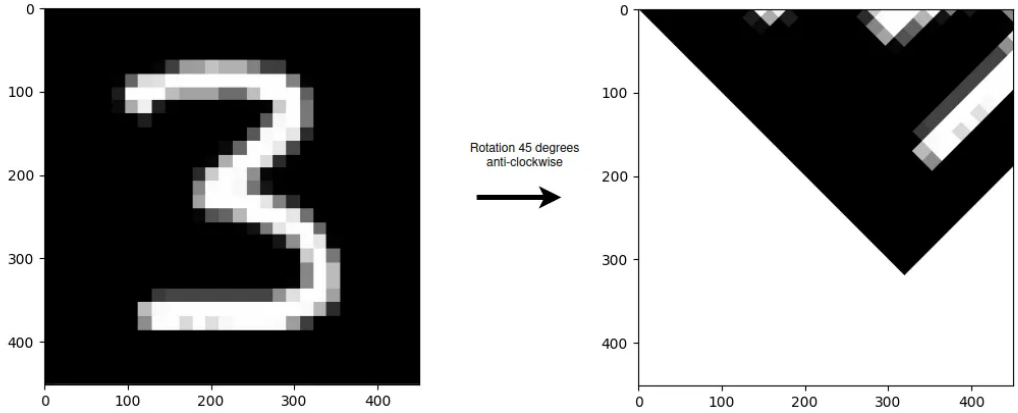


Inverse Mapping



Input space

Output space

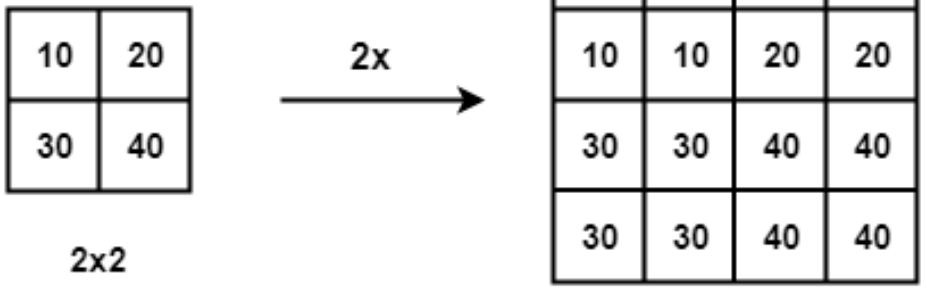


Interpolation

- Interpolation is a basic tool used extensively in tasks such as zooming, shrinking, rotating, and geometric corrections.
- Interpolation is the process of using known data to estimate values at unknown locations.
- Our problem is that of **assigning an intensity value to the pixel in the output image** after that, by means of a inverse geometric transformation, we get the **corresponding point in the input image**
- There are several techniques with different computational costs

Nearest Neighbor Interpolation

- To perform intensity-level assignment for any point, we look for its closest pixel in the original image and assign the intensity of that pixel to the new pixel in the grid. It is the faster but provide low quality images



2x2

(u, v)

Input Image I

4x4

(x, y)

Output Image J

Inverse mapping T^{-1}: u = x/2

v = y/2

(x, y)→(u,v)→assignment to output image

(0,0) → (0,0) → J(0,0) = I(0,0)
(0,1) → (0,0.5) → J(0,1) = I(0,0)
(0,2) → (0,1) → J(0,2) = I(0,1)
(0,3) → (0,1.5) → J(0,3) = I(0,1)

(1,0) → (0.5,0) → J(1,0) = I(0,0)
(1,1) → (0.5,0.5) → J(1,1) = I(0,0)
(1,2) → (0.5,1) → J(1,2) = I(0,1)
(1,3) → (0.5,1.5) → J(1,3) = I(0,1) ….

Bilinear Interpolation

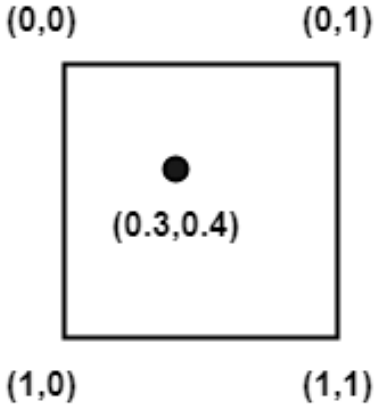
- In bilinear interpolation we use the four nearest neighbors to estimate the intensity at a given location by means of a weighted sum. It is more costly but provide better results
- Let (x,y) denote the pixel location to which we want to assign an intensity value J(x,y), and let (u,v) the point that corresponds to (x,y) in the original image
- We first need to find the 4-neighbors of (u,v):

$\{(u_f, v_f),(u_f, v_c), (u_c, v_f),(u_c, v_c)\}$

u_f=floor(u) u_c=ceil(u)

$J(x,y) = a*I(u_f, v_f)+ b*I(u_f, v_c)+ c*I(u_c, v_f)+ d*I(u_c, v_c)$

The weights a, b, c, d provide information about the distance of (u,v) to the neighbor point



(u,v)=(0.3, 0.4)

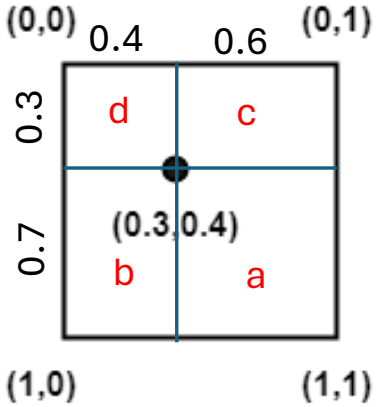
Bilinear Interpolation

- How can we compute the weights?

$$J(x,y) = a*I(u_f, v_f) + b*I(u_f, v_c) + c*I(u_c, v_f) + d*I(u_c, v_c)$$

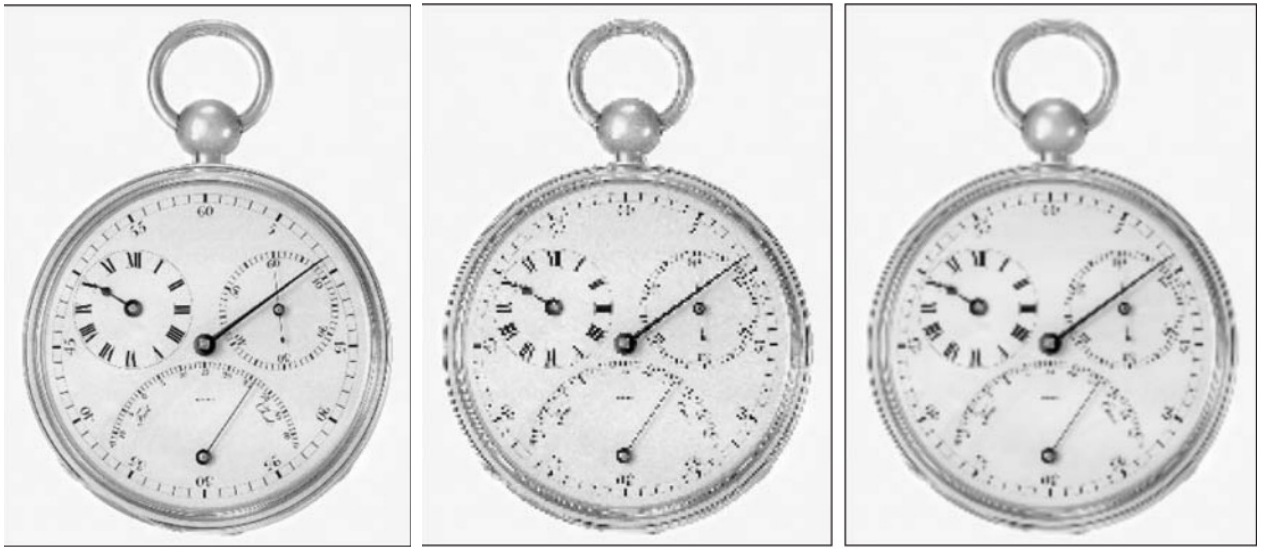
- Geometrically, each weight represents the area of the partial area diagonally opposite the corner within the unit square

a=0.7*0.6
b=0.7*0.4
c=0.3*0.6
d=0.3*0.4



(u,v)=(0.3, 0.4)

NN vs Bilinear Interpolation



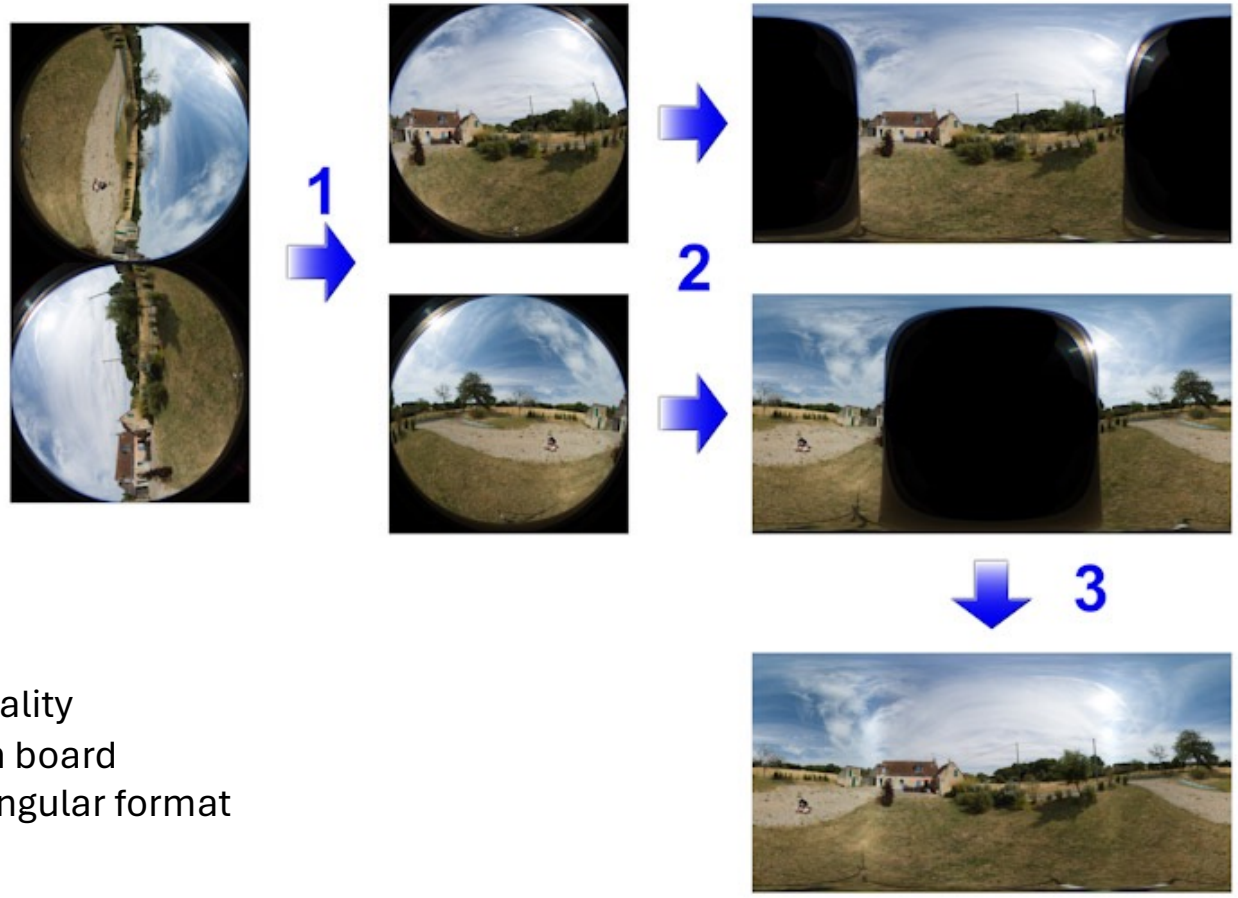
Original NN Bilinear

Image shrinked and then zoomed back to the original size

Original size 444 x 338, reduced size 213 x 162

Spherical Cameras - examples

Spherical Images



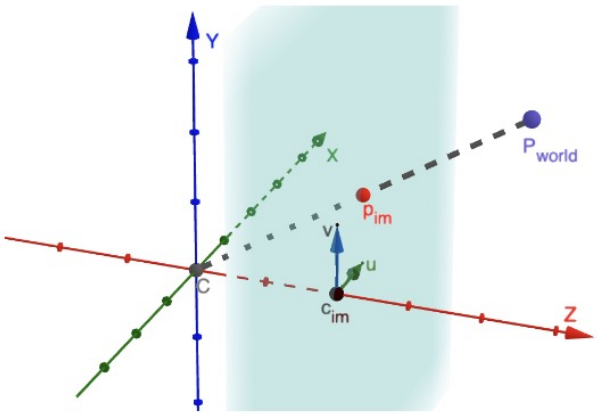
1. Each fisheye lens acquire an image
2. Images are corrected to improve quality
3. Images are rectified and stitched on board
4. It produce an image in an equirectangular format

Projection

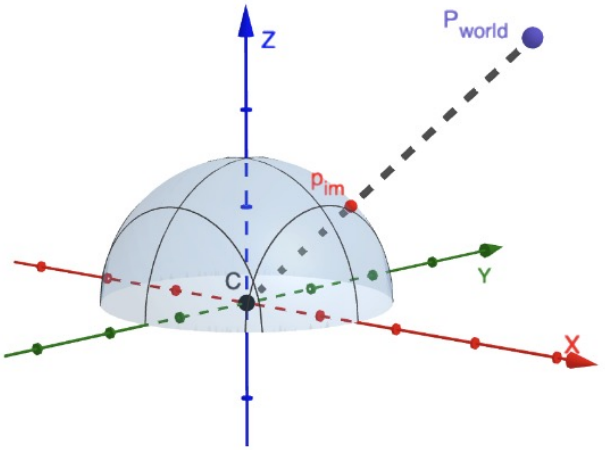
(a) Spherical Image

(b) Equirectangular Image

Projection: pinhole vs spherical cameras



(a) Central Projection in Pinhole Camera



(b) Central Projection in Spherical Camera

To find projection on the plane we use

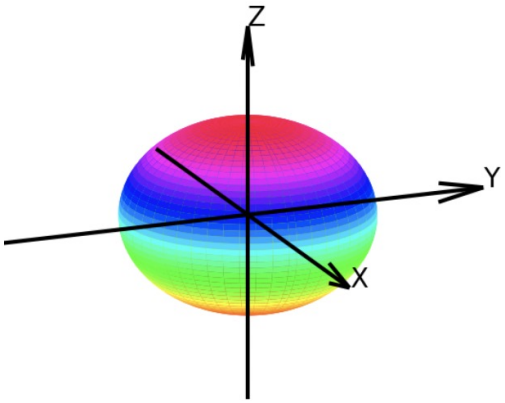
$$\bar{x} = \begin{bmatrix} f\frac{x}{z} \\ f\frac{y}{z} \\ 1 \end{bmatrix}$$

To find projection on the sphere we use

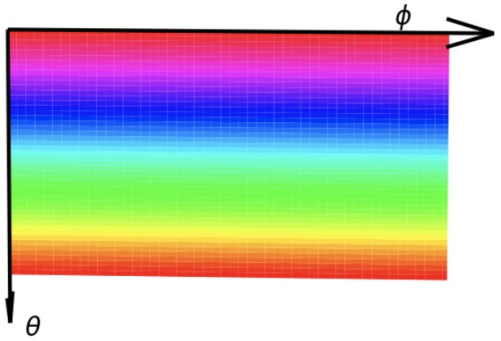
$$\bar{x} = \frac{x}{\|x\|}$$

Which is a 3D point

From spherical to equirectangular image



(a) Unit Sphere in Cartesian Coordinates



(b) Equirectangular image of the unit Sphere

The most commonly used projection to map a spherical image to an image plane is the non-linear equirectangular projection which simply transform the Cartesian coordinates into spherical coordinates as usual

$$\phi = \arctan \frac{y'}{x'};$$
$$\theta = \arccos \frac{z'}{f};$$
$$\rho = f.$$

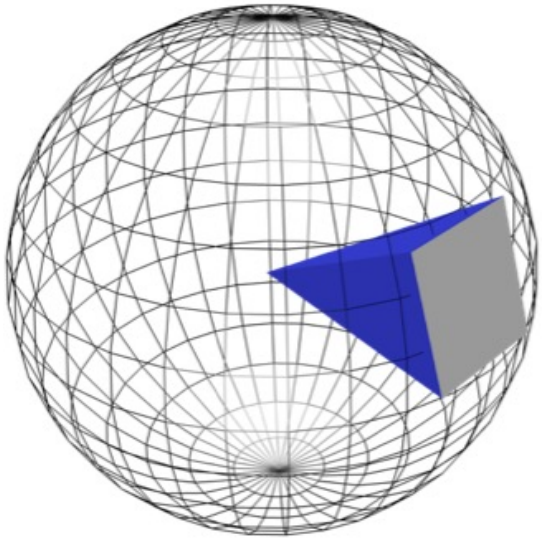
With f=1

Since rho is constant, it can be dropped and angles are rescaled to obtain pixel coordinates of an image of size HxW

$$\begin{bmatrix} x_e \\ y_e \end{bmatrix} = \begin{bmatrix} \frac{W}{2\pi} & 0 \\ 0 & \frac{H}{\pi} \end{bmatrix} \begin{bmatrix} \phi \\ \theta \end{bmatrix}$$

(be careful with the origin of the reference system)

From spherical to planar images

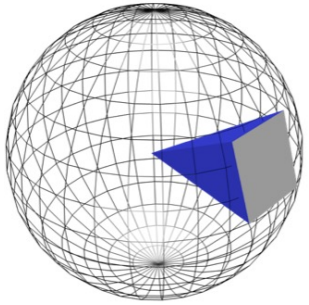


Our goal is to project part of the spherical surface onto a tangent plane to rectify the image and reduce deformation

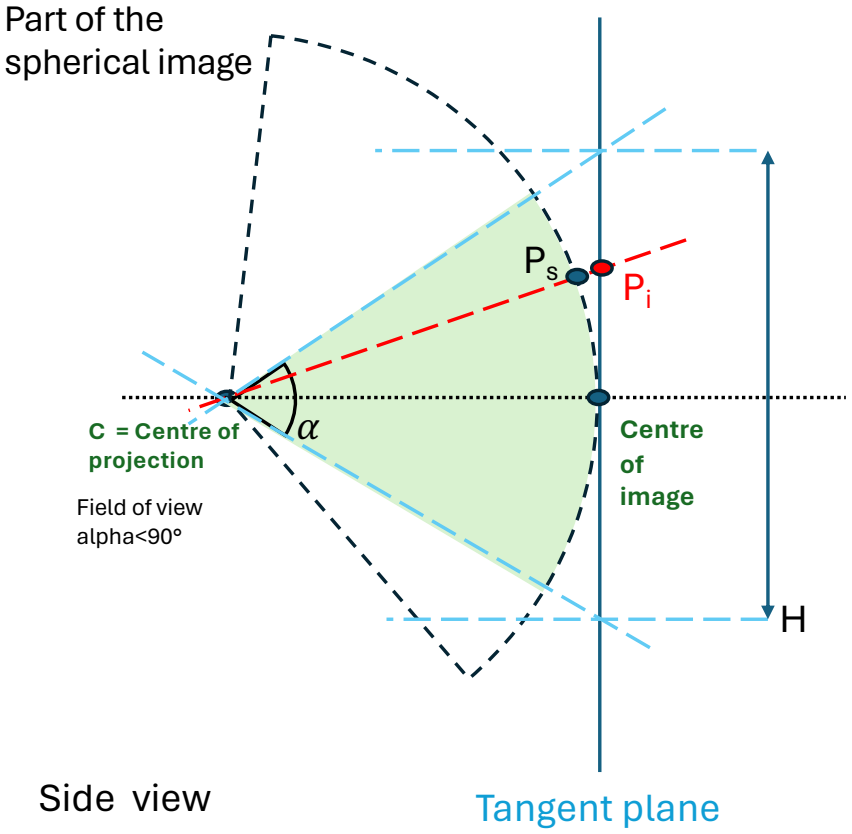
The plane is tangent to the sphere at a 3D point

Only a portion of the defined sphere can be projected across a field of view expressed in degrees (or radiants)

From spherical to planar images



Ray = 1



P_s is a point on the sphere
P_i is its projection on the tangent plane
H is the height of the planar image
α is the field of view

Note that $\frac{H}{2} = \tan\frac{\alpha}{2}$

Given a point P_s on the sphere, we compute

- the tangent plane k
- The line L through C and P_s
- The intersection between L and the plane k

From spherical to planar images

- To compute the planar image, we use inverse mapping
- Thus, for each pixel (u,v) of the planar image:
  - We find the corresponding point P_i on the tangent plane
    - Here it is convenient to consider a «**canonical**» plane tangential to the sphere at the point (1, 0, 0) (in practice x = 1, be careful on how you set the axis in your reference system)
    - Then, the point on this canonical plane is rotated by a 3D rotation matrix such that the plane is tangent to the sphere at the given input point (expressed as a point on the sphere in spherical coordinates)
  - Instead of finding the line that intersects the sphere, it is sufficient to normalize the point of the rotated tangent plane
  - Once the projection of the point on the sphere in Cartesian coordinates is obtained, it is easy to convert it to spherical coordinates and subsequently compute the location of the pixel on the equirectangular image
  - You may wanna take a look at the remap function in openCV to avoid implementing the bilinear interpolation

Assignment