Digital Image Processing

Image Formation – Part II

Prof. Tiago Vieira

Universidade Federal de Alagoas

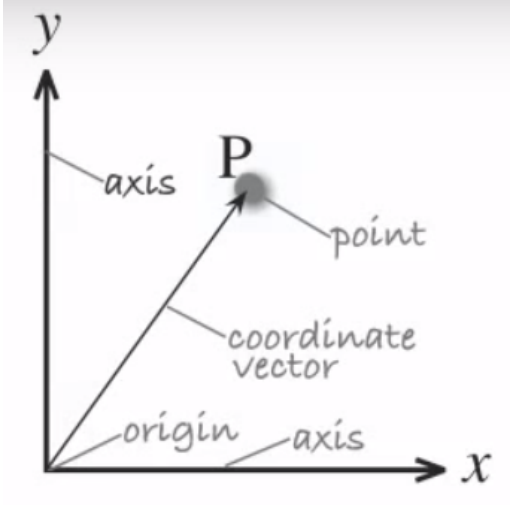
tvieira@ic.ufal.br

April 27, 2021

Contents

Quick geometry recap

- Some familiar concepts from geometry:
  - Euclidean plane.
    - A non-curved space where the rules of Euclidean geometry apply.
  - Cartesian coordinates.
    - Distances to a point with respect to the origin and measured along orthogonal axes.



Homogeneous coordinates

- Cartesian $\mapsto$ Homogeneous:

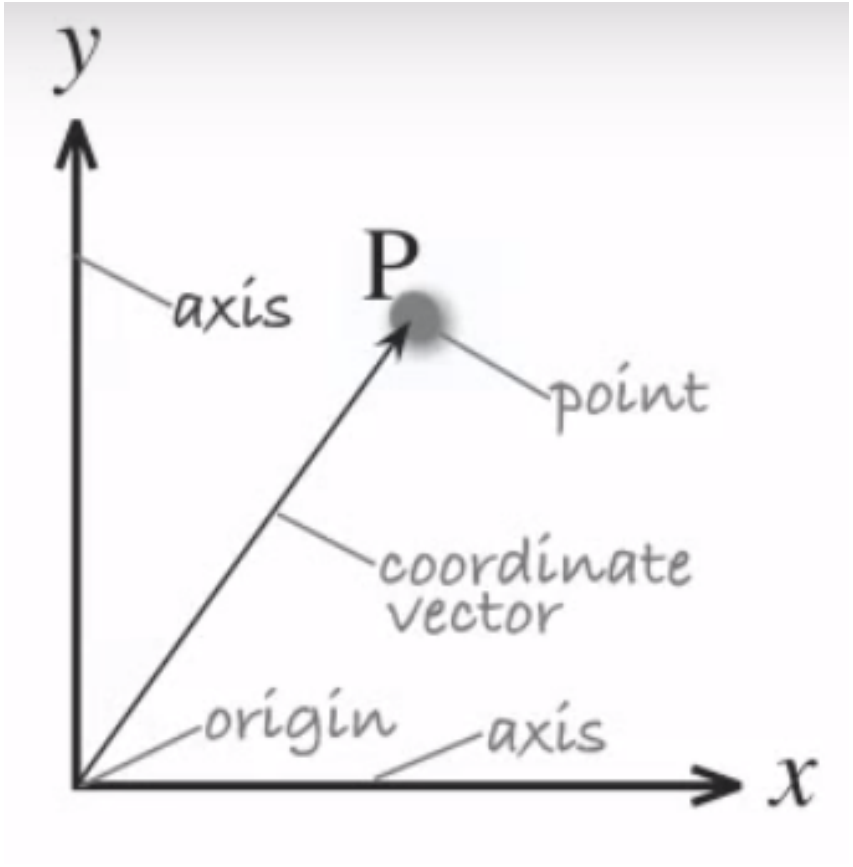
$$\mathbf{P} = (x, y) \quad \tilde{\mathbf{P}} = (x, y, 1)$$
$$\mathbf{P} \in \mathbb{R}^2 \qquad \tilde{\mathbf{P}} \in \mathbb{P}^2$$

- Homogeneous $\mapsto$ Cartesian:

$$\tilde{\mathbf{P}} = (\tilde{x}, \tilde{y}, \tilde{z}) \mapsto \mathbf{P} = (x, y)$$

where

$$x = \tilde{x}/\tilde{z}, \quad y = \tilde{y}/\tilde{z}.$$



In homogeneous coordinates

Lines and points are duals!

Pin-hole in homogeneous form

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

I.e.,

$$\tilde{x} = fX, \quad \tilde{y} = fY, \quad \tilde{z} = Z.$$

As before:

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

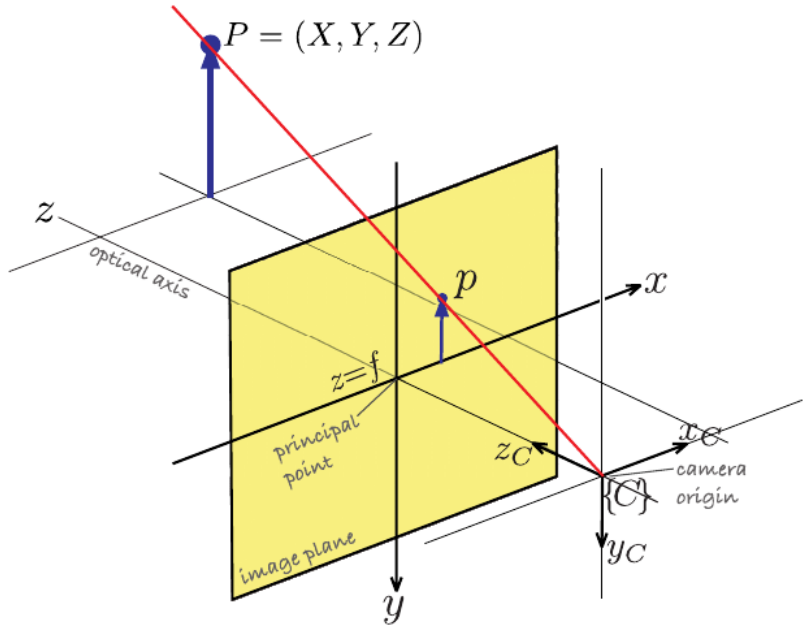
- Perspective transformation, with the divide by Z is **linear** in homogeneous coordinate form.

Pin-hole in homogeneous form

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

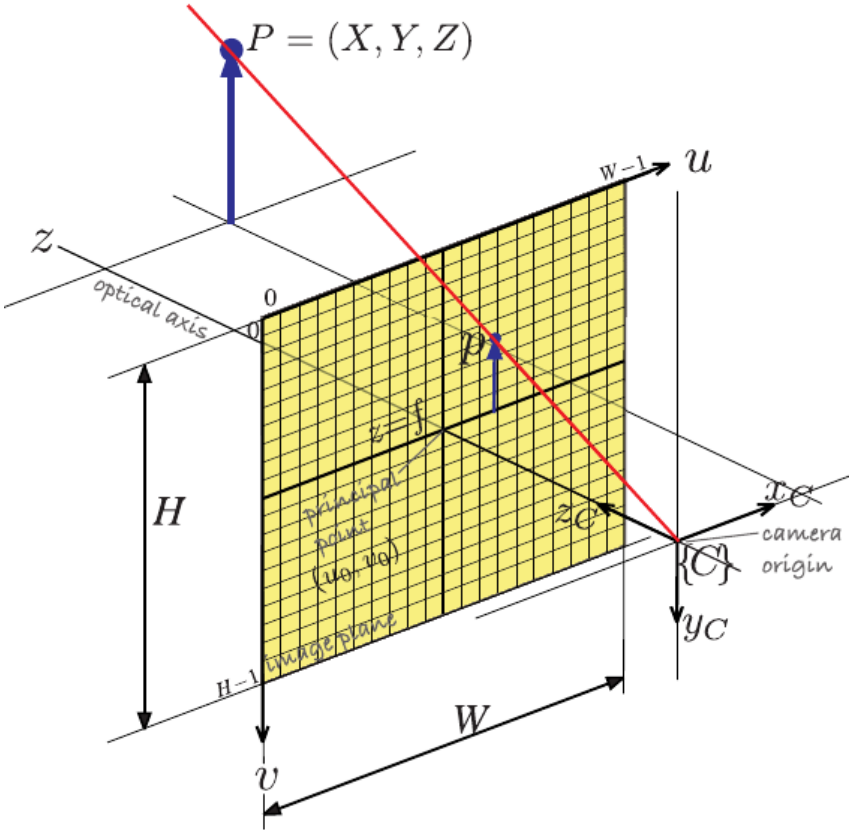
$$\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{3D \mapsto 2D} \underbrace{\begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{scaling/zooming}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Central Projection Model



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

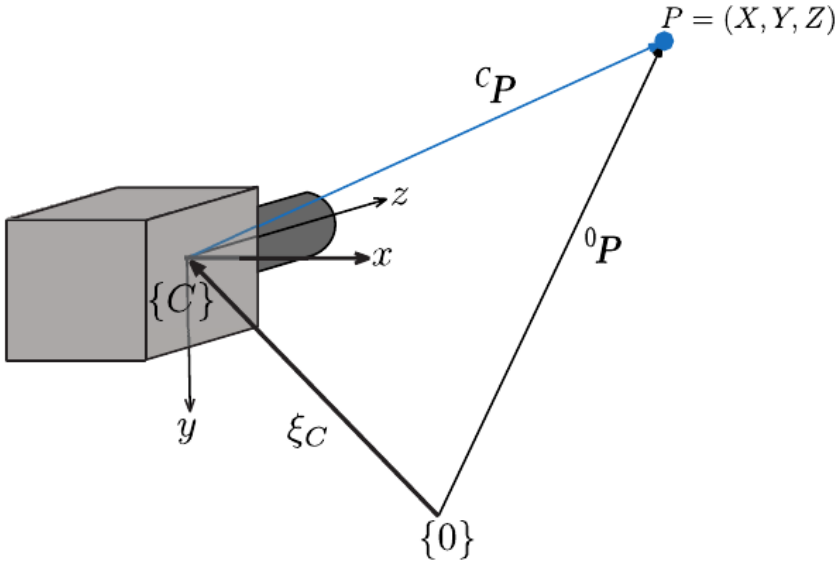
Central Projection Model



$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \frac{1}{\rho_u} & 0 & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix}$$

- Scale from meters to pixels.
- Shift the origin to top left corner.

$$p = \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} = \begin{pmatrix} \tilde{u}/\tilde{w} \\ \tilde{v}/\tilde{w} \end{pmatrix}$$



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

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \underbrace{\underbrace{\begin{pmatrix} \frac{1}{\rho_u} & 0 & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{K=\text{intrinsic parameters}} \underbrace{\begin{pmatrix} \mathbf{R} & t \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix}^{-1}}_{\text{extrinsic parameters}}}_{C=\text{camera matrix}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \underbrace{\underbrace{\begin{pmatrix} \frac{1}{\rho_u} & 0 & u_0 \\ 0 & \frac{1}{\rho_v} & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}}_{K=\text{intrinsic parameters}} \underbrace{\begin{pmatrix} \mathbf{R} & t \\ \mathbf{0}_{1\times 3} & 1 \end{pmatrix}^{-1}}_{\text{extrinsic parameters}}}_{C=\text{camera matrix}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- Intrinsic parameters (5): f, ρ_u, ρ_v, u_0 and v_0.
- Extrinsic parameters (6): Rotation and translation.

Camera parameters

Initially unknown and estimated using a calibration procedure.

Camera matrix

- Mapping points from the world to an image (pixel) coordinate is simply a matrix multipliaction using

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

And:

$$p = \begin{pmatrix} \tilde{u} \\ \tilde{v} \end{pmatrix} = \begin{pmatrix} \tilde{u}/\tilde{w} \\ \tilde{v}/\tilde{w} \end{pmatrix}$$

Field of View (FOV)

- The field of view of a lens is an open rectangular pyramid that subtends angles θ_h and θ_v in the horizontal and vertical planes respectively.
- The dimension of a sensor chip d is measured diagonally between its corners and is typically expressed in inches.
- Common dimensions are $1/4$, $1/3$ and $1/2$ inch.
- A normal lens has $f \approx d$ and a wide-angle lens generally has $f > d/3$ giving a maximum angular field of view of $\approx 110^\circ$.

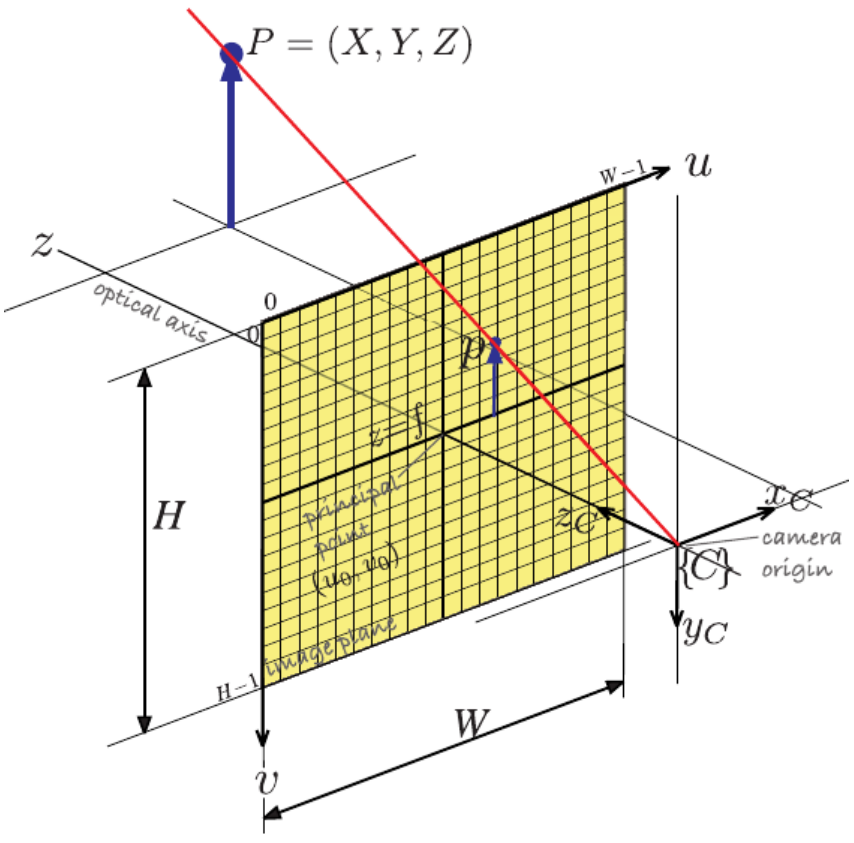
Field of View (FOV)

- For wide-angle lenses it is more common to describe the field of view as a solid angle which is measured in units of steradians (or sr).
- This is the area of the field of view projected onto the surface of a unit sphere.
- A hemispherical field of view is 2π sr and a full spherical view is 4π sr.
- If we approximate the camera's field of view by a cone with apex angle θ the corresponding solid angle is $\Omega = 2\pi(1 - cos\theta/2)$ sr.
- A camera with a field of view greater than a full hemisphere is termed omni-directional or panoramic.

Field of View (FOV)

- The field of view of a camera is a function of its focal length f.
- A wide-angle lens has a small focal length.
- A telephoto lens has a large focal length
- A zoom lens has an adjustable focal length.

Field of View (FOV)



- Horizontal angle view:

$$\theta_h = 2\tan^{-1}\frac{\rho_u W/2}{f}$$

- Vertical angle view:

$$\theta_v = 2\tan^{-1}\frac{\rho_v H/2}{f}$$

- Note that the FOV depends on the dimensions of the camera chip, ie., $W\rho^u \times H\rho_v$.

Camera matrix

Given the following camera matrix

```
C = [...
    512     -800    0       800; ...
    512     0       -800    1600; ...
    1       0       0       0
    ];
```

Find the image coordinates of point $P = \begin{pmatrix} 4 & 0 & 0 \end{pmatrix}^T$

Camera matrix

```
C = [...
    512     -800    0       800; ...
    512     0       -800    1600; ...
    1       0       0       0
    ];
P = [4, 0, 0];
W = C * [P, 1]'
W = W/W(3)
```

Scale invariance

- Consider an arbitrary scalar scale factor

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \lambda \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & C_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

- $\tilde{u}, \tilde{v}, \tilde{w}$ will all be scaled by λ.
- but

$$u = \frac{\tilde{u}}{\tilde{w}}, \; v = \frac{\tilde{v}}{\tilde{w}}.$$

- So the result is unchanged.

Normalized camera matrix

Since scale factor is arbitrary we can fix the value of one element, typically $C(3,4)$ to one.

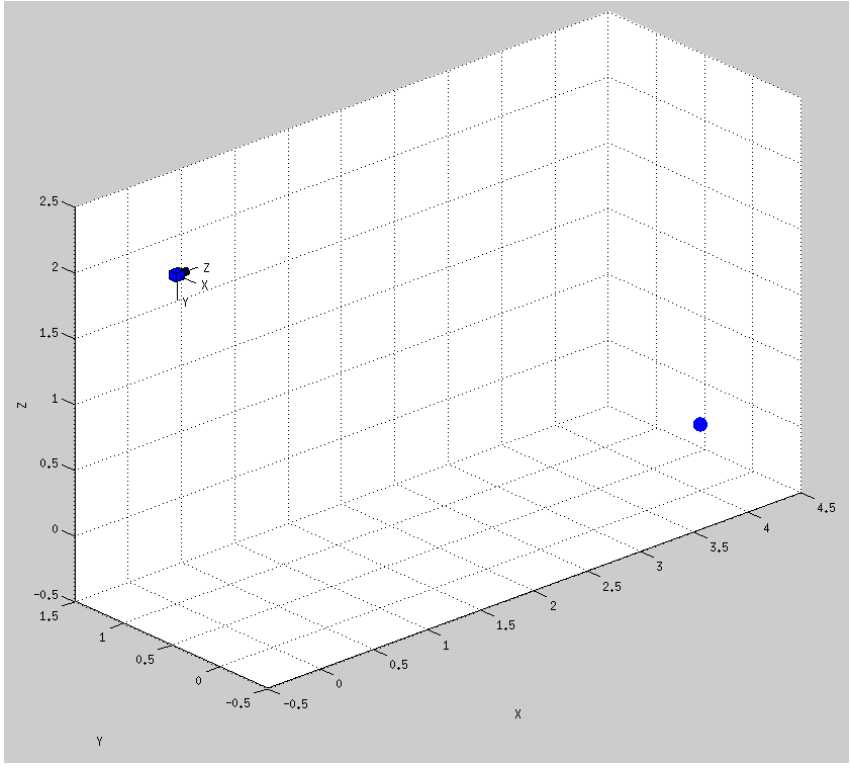
$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \lambda \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Example

```
cam = CentralCamera('default', 'pose', ...
    transl(0,1,2)*troty(pi/2)*trotz(-pi/2));
cam.C
P = [4,0,0]';
cam.project(P)
```

Example

```
%%
cam = CentralCamera(...
    'default', 'pose', ...
    transl(0,1,2)*...
    troty(pi/2)*...
    \\
    trotz(-pi/2));
cam.C
P = [4,0,0]';
cam.project(P)
cam.plot_camera
plot_sphere(P, 0.05)
grid on
```



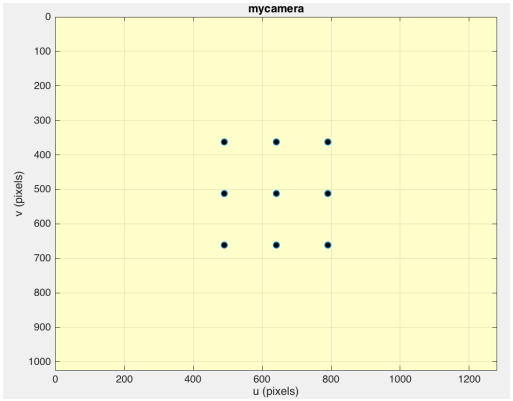
Camera matrix

Find the camera's matrices:

```
cam.C
cam.K
cam.fov() * 180/pi
```

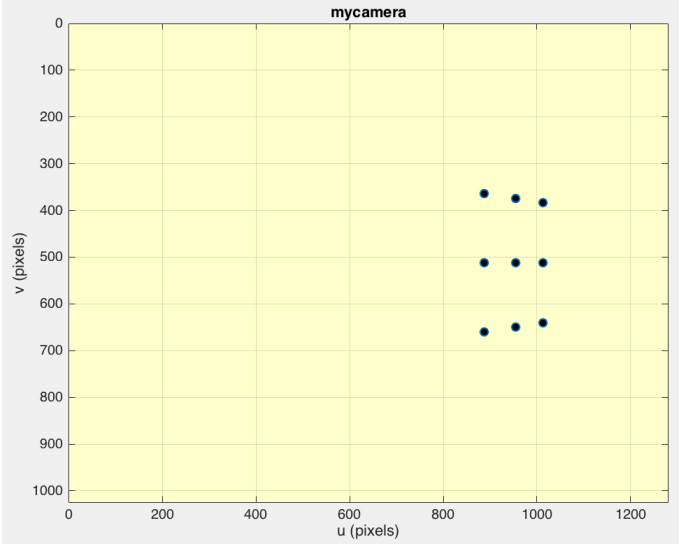
Example

```
cam = CentralCamera('focal', 0.015, ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024], ...
    'centre', [640 512], 'name', 'mycamera')
P = mkgrid(3, 0.2, 'T', transl(0, 0, 1.0));
cam.project(P)
cam.plot(P)
```

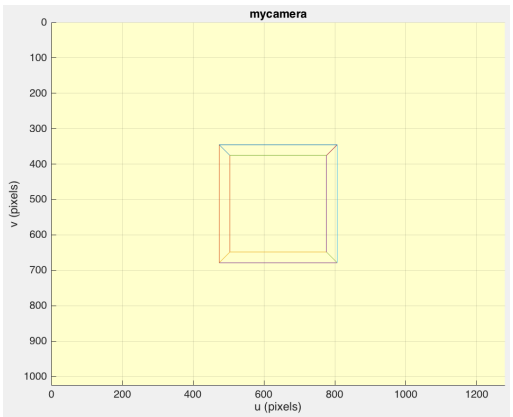


Example

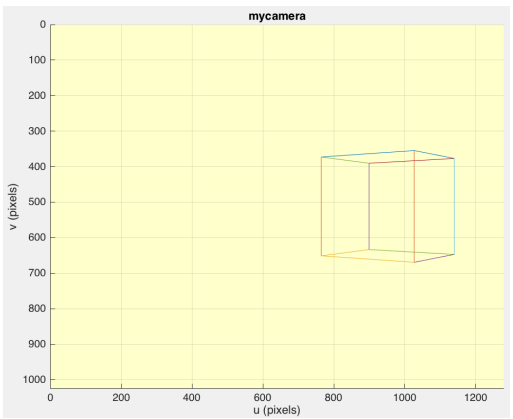
```
Tcam = transl(-1,0,0.5)*troty(0.9);
cam.plot(P, 'Tcam', Tcam)
```



```
cam = CentralCamera('focal', 0.015, ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024], ...
    'centre', [640 512], 'name', 'mycamera')
cube = mkcube(0.2, 'T', transl([0, 0, 1]) );
[X,Y,Z] = mkcube(0.2, 'T', transl([0, 0, 1.0]), 'edge');
cam.mesh(X, Y, Z)
```

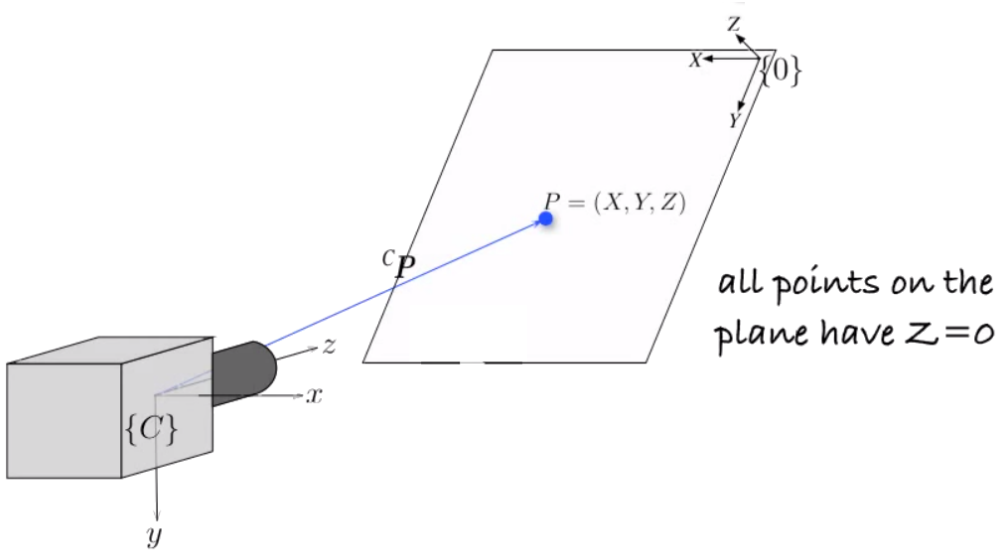


```
cam = CentralCamera('focal', 0.015, ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024], ...
    'centre', [640 512], 'name', 'mycamera')
cube = mkcube(0.2, 'T', transl([0, 0, 1]) );
[X,Y,Z] = mkcube(0.2, 'T', transl([0, 0, 1.0]), 'edge');
cam.T = transl(-1,0,0.5)*troty(0.8);
cam.mesh(X, Y, Z, 'Tcam', Tcam);
```



```
cam = CentralCamera('focal', 0.015, ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024], ...
    'centre', [640 512], 'name', 'mycamera')
theta = [0:20]/100*2*pi;
[X,Y,Z] = mkcube(0.2, [], 'edge');
for th = theta
    T_cube = transl(0, 0, 1.5)*trotx(th)*...
        troty(th*1.2)*trotz(th*1.3)
    [X,Y,Z] = mkcube(0.2, 'T', T_cube, 'edge');
    cam.mesh(X, Y, Z, 'Tcam', Tcam);
    pause(.01);
end
```

Planar homography



$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix}$$

Planar homography

$$\begin{pmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

- Once again the scale factor is arbitrary.
- 8 unique numbers in the homography matrix.
- Can be estimated from 4 world points and their corresponding image points.

Perspective rectification

Perspective rectification

```
[x, y] = ginput(4)
```

Perspective rectification

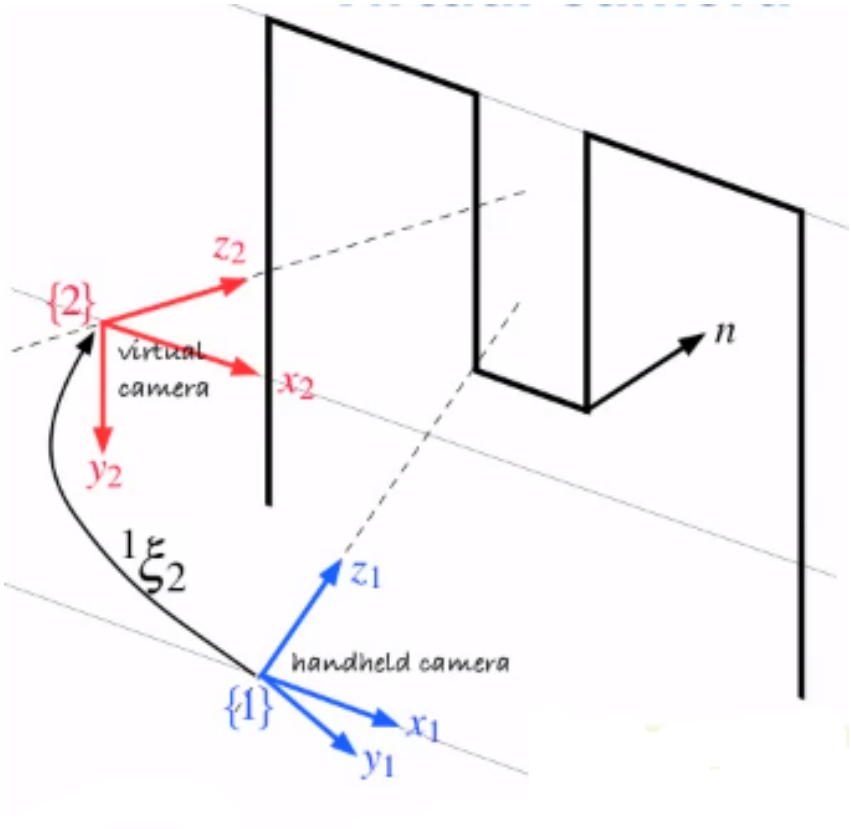
```
H = homography([x, y]', [x2, y2]')
```

$$^{2}p = \mathbf{H}^{1}p$$

Perspective rectification

```
homwarp(H, f, 'full')
```

Virtual camera



$$H_E \approx R + \frac{t}{d} n^T$$


London 2012
London 201
3
TATEISH
2
JAMIESON
WR
GYURTA
WR
2:07.31
OR
2:07.64
NEW WR
2:07.28

Lens distortion

No lenses are perfect:

- Chromatic aberration (color fringing).
- Spherical aberration astigmatism (variation in focus accross the scene).

Geometric distortion

Most problematic effect in computer vision.

Two components:

1. Radial.
2. Tangential.

Lens distortion

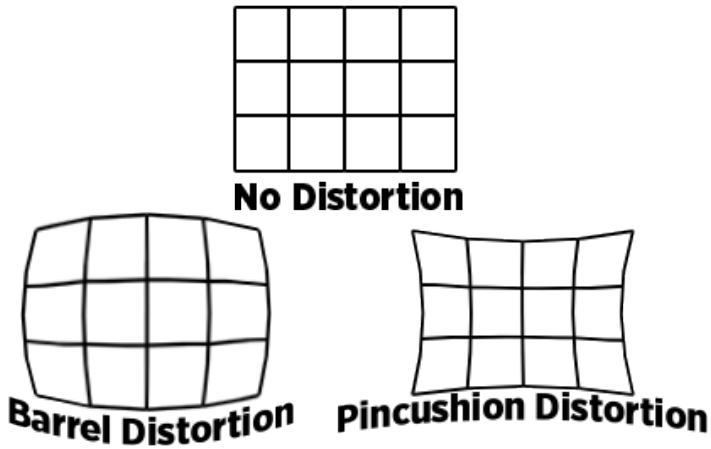
Tangential distortion. Displacement between the lens principal axis and the sensor's geometric center.

Lens distortion

The radial error is approximated by

$$\delta r = k_1 r^3 + k_2 r^5 + k_3 r^7 + \dots \quad (1)$$

where r is the distance from the image point to the sensor's principal point.



Lens distortion

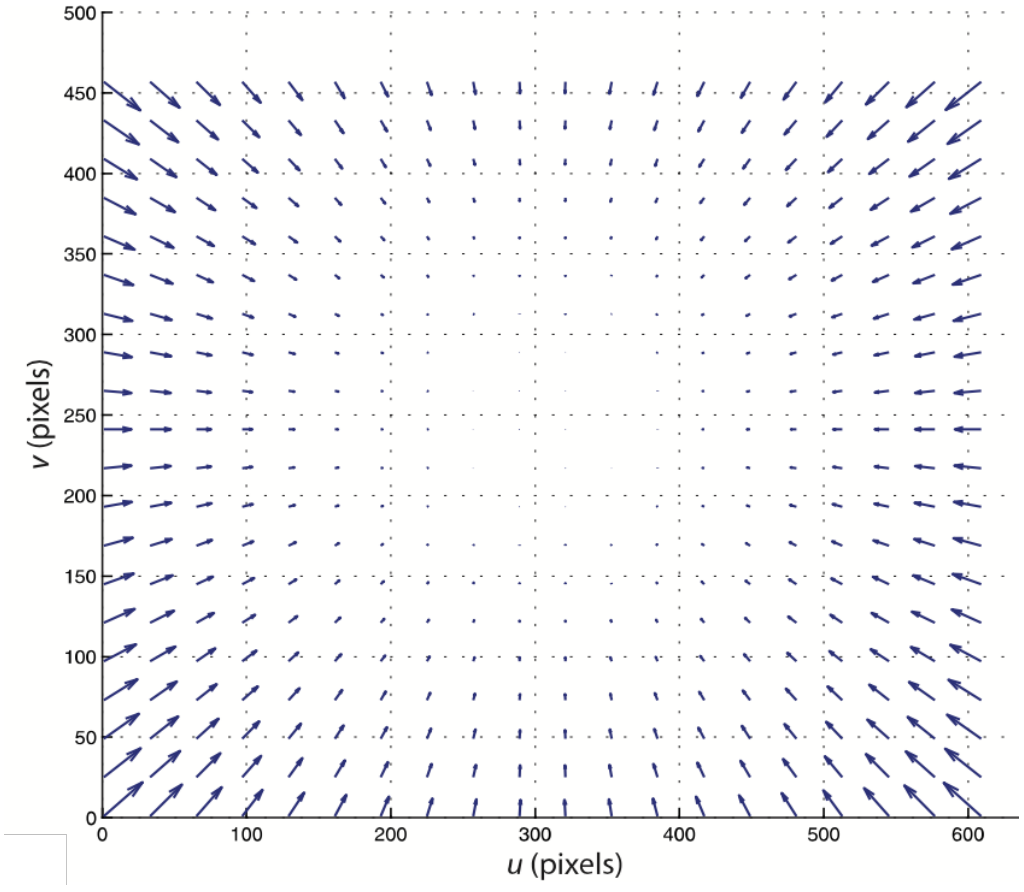
The coordinates (u, v) of a point, after distortion are given by

$$\begin{aligned} u^{(d)} &= u + \delta_u \\ v^{(d)} &= v + \delta_v \end{aligned} \tag{2}$$

Where the displacement is

$$\begin{pmatrix} \delta_u \\ \delta_v \end{pmatrix} = \underbrace{\begin{pmatrix} u\left(k_1 r^2 + k_2 r^4 + k_3 r^6 + \ldots\right) \\ v\left(k_1 r^2 + k_2 r^4 + k_3 r^6 + \ldots\right) \end{pmatrix}}_{\text{radial}} + \underbrace{\begin{pmatrix} 2p_1 uv + p_2(r^2 + 2u^2) \\ p_1(r^2 + 2v^2) + 2p_1 uv \end{pmatrix}}_{\text{tangential}} \tag{3}$$

Lens distortion



Typically three coefficients are sufficient to describe the radial distortion and the distortion model is parameterized by (k1, k2, k3, p1, p2) which are considered as additional intrinsic parameters. Distortion can be modeled by the CentralCamera class using the 'distortion' option, for example

```
cam = CentralCamera('focal', 0.015, ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024], ...
    'centre', [512 512], ...
    'distortion', [k1 k2 k3 p1 p2] )
```

Camera calibration

A linear calibration model (book).
Example:

```
P = mkcube(0.2);
T_unknown = transl(0.1, 0.2, 1.5) * ...
    rpy2tr(0.1, 0.2, 0.3);
cam = CentralCamera('focal', 0.015, ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024], ...
    'centre', [512 512], ...
    'noise', 0.05);
p = cam.project(P, 'Tobj', T_unknown);
C = camcald(P, p)
```

```
% Decomposing the Camera Calibration Matrix
est = invcamcal(C) % Attention! invcamcal_updated
est.f/est.rho(1)
cam.f/cam.rho(2)

T_unknown * est.T

est.plot_camera()
```

Pose Estimation

```
cam = CentralCamera('focal', 0.015, ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024], ...
    'centre', [640 512]);
P = mkcube(0.2);
T_unknown = transl(0,0,2)*trotx(0.1)*troty(0.2)
p = cam.project(P, 'Tobj', T_unknown);
T_est = cam.estpose(P, p)
```

Pose Estimation

```
T_unknown =

    0.9801         0    0.1987         0
    0.0198    0.9950   -0.0978         0
   -0.1977    0.0998    0.9752    2.0000
         0         0         0    1.0000


T_est =

    0.9801    0.0000    0.1987   -0.0000
    0.0198    0.9950   -0.0978    0.0000
   -0.1977    0.0998    0.9752    2.0000
         0         0         0    1.0000
```

Pose Estimation

```
cam = CentralCamera('focal', 0.015, ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024], ...
    'centre', [640 512], ...
    'noise', 0.05);
P = mkcube(0.2);
T_unknown = transl(0,0,2)*trotx(0.1)*troty(0.2)
p = cam.project(P, 'Tobj', T_unknown);
T_est = cam.estpose(P, p)
```

```
T_unknown =

    0.9801         0    0.1987         0
    0.0198    0.9950   -0.0978         0
   -0.1977    0.0998    0.9752    2.0000
         0         0         0    1.0000


T_est =

    0.9794    0.0002    0.2020   -0.0001
    0.0204    0.9948   -0.0999    0.0000
   -0.2009    0.1019    0.9743    2.0019
         0         0         0    1.0000
```

Camera Calibration Toolbox

- Camera Calibration Toolbox for Matlab https://www.vision.caltech.edu/bouguetj/calib_doc/

  `calib_gui`

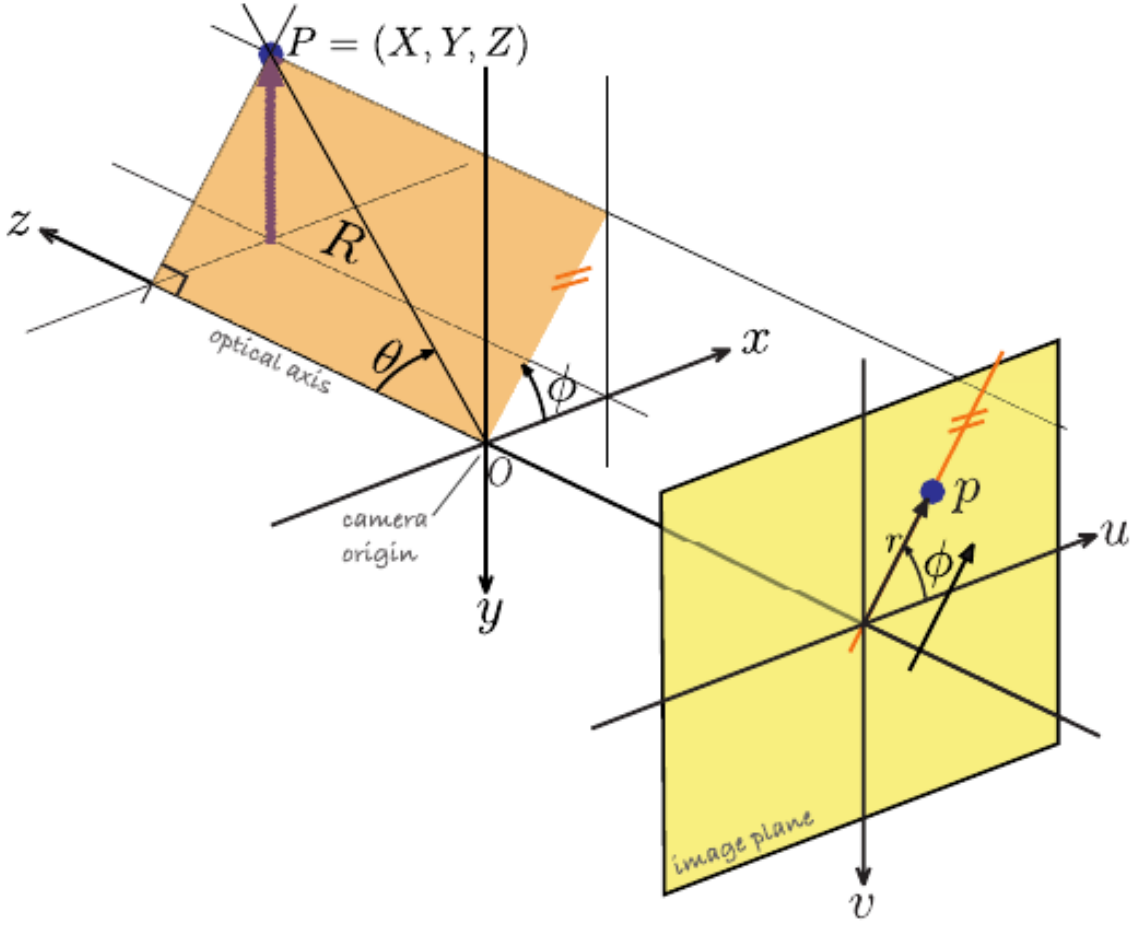
- Camera calibration APP (Matlab).
- Stereo camera calibration APP (Matlab).

Non-Perspective Imaging Models

- Perspective camera model mimics the way human-eye works.
- FOV limited to one hemisfere.
- Radial distortion increasingly higher for smaller focal length f.
- Let's drop the perspective contrain and use reflection rather than refraction.
- Mirrors are free of color fringing.
- Easier to scale.

Figure: Escher (1935).

Fisheye Lens Camera



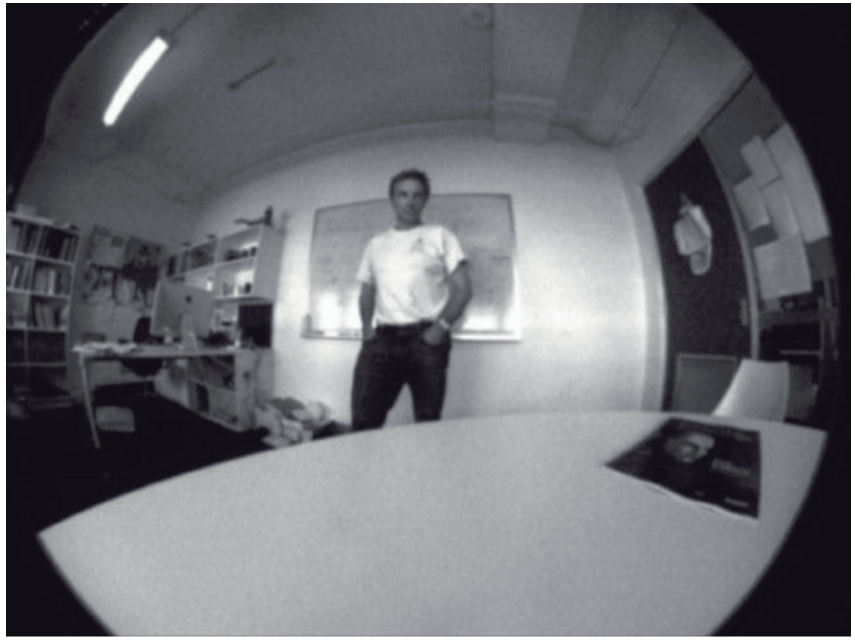
$$R = \sqrt{X^2 + Y^2 + Z^2}$$

$$\theta = \cos^{-1} \frac{R}{Z}$$

$$\phi = \tan^{-1} \frac{Y}{X}$$

$$u = r \cos \phi$$

$$v = r \sin \phi$$



Mapping	Equation
Equiangular	$r = k\,\theta$
Stereographic	$r = k \tan(\theta/2)$
Equisolid	$r = k \sin(\theta/2)$
Polynomial	$r = k_1\theta + k_2\theta^2 + \cdots$

```
cam = FishEyeCamera('name', 'fisheye', ...
    'projection', 'equiangular', ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024])

[X,Y,Z] = mkcube(0.2, 'centre', [0.2, 0, 0.3], 'edge');
cam.mesh(X, Y, Z)
```

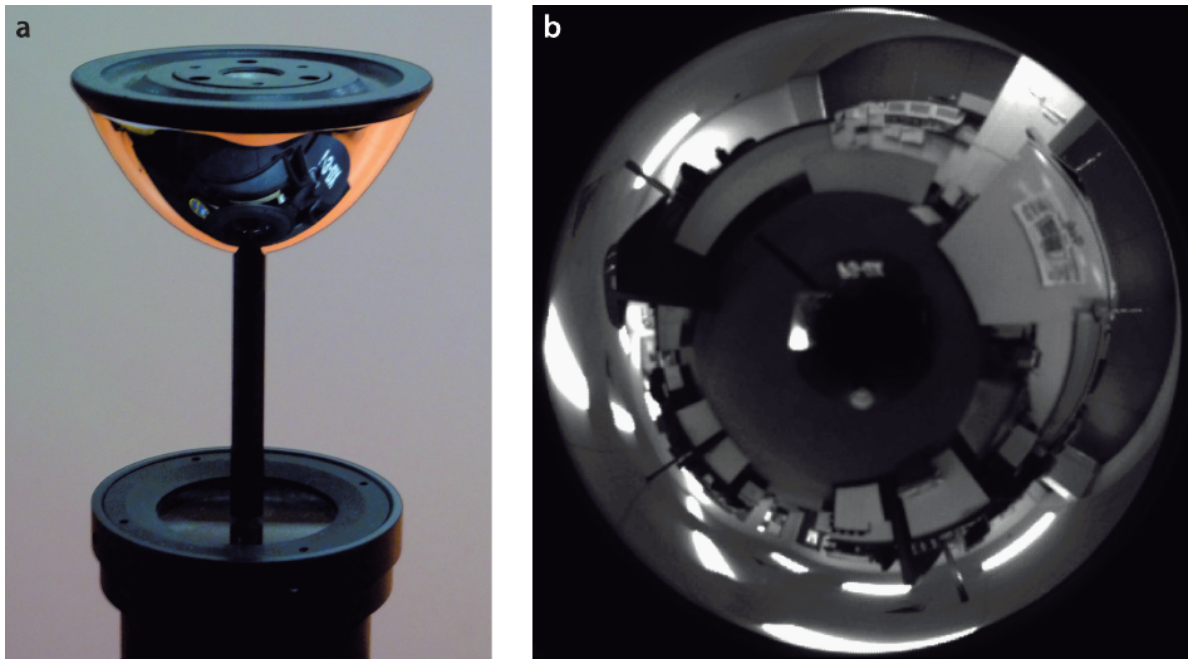
Wide-angle lenses:

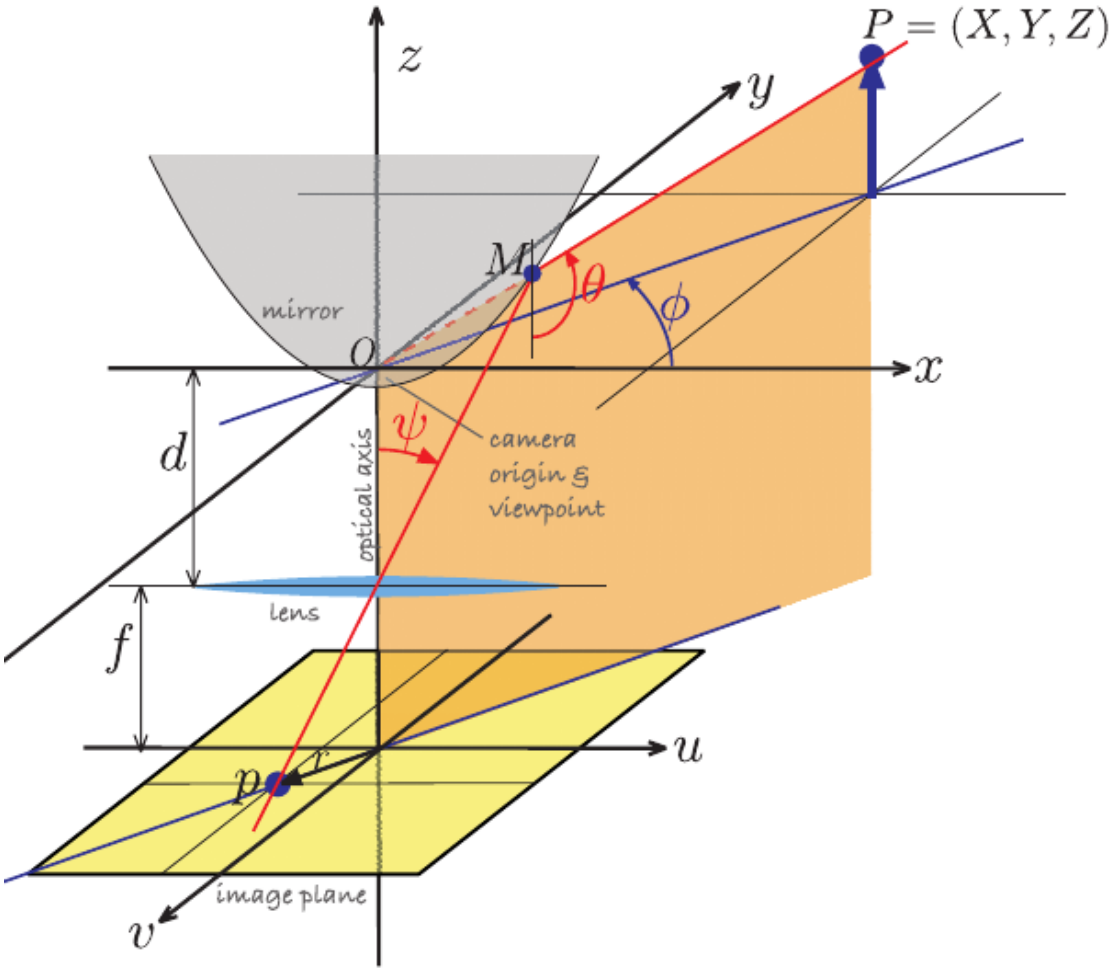
- Can have 180° or 190° FOV.
- Spatial resolution is lower since the camera pixels are spread over a wider FOV.
- The FOV is a circular region, i.e., nearly 25% of the rectangular image plane is effectively wasted.
- Outdoor images may reduce too much the exposure to avoid saturation.

Catadioptric Camera

Catadioptric

A catadioptric imaging system comprises both reflective and refractive elements, a mirror and a lens





- Elevation angle:

$$\theta = \tan^{-1} \frac{Z}{X^2 + Y^2} + \frac{\pi}{2}$$

a

- Relation between ψ and θ is determined by the mirror shape at M.
  - Spherical, parabolic, elliptical and hyperbolic.

Figure: Catadioptric image formation

Equiangular mirror

Each pixel spans an equal angle, irrespective of its distance from the center of the image.

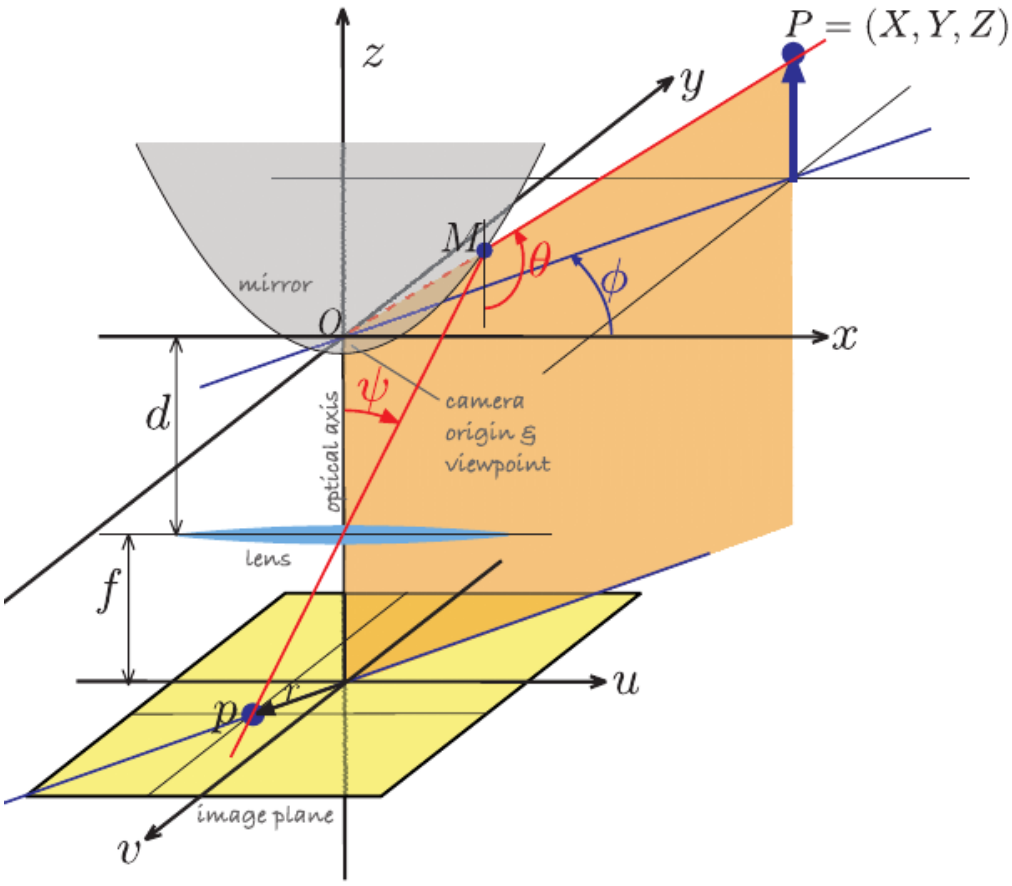


Figure: Catadioptric image formation

For equiangular mirrors, we have

$$\theta = \alpha\psi$$

$$r = f \tan\psi$$

$$\mathbf{p} = (r, \phi)$$

$$u = r\cos\phi$$

$$v = r\sin\phi$$

where

$$\phi = \tan^{-1}(Y/X)$$

```
cam = CatadioptricCamera(...
    'name', 'panocam', ...
    'projection', 'equiangular', ...
    'maxangle', pi/4, ...
    'pixel', 10e-6, ...
    'resolution', [1280 1024])
[X,Y,Z] = mkcube(1, ...
    'centre', [1, 1, 0.8], 'edge');
cam.mesh(X, Y, Z)
```

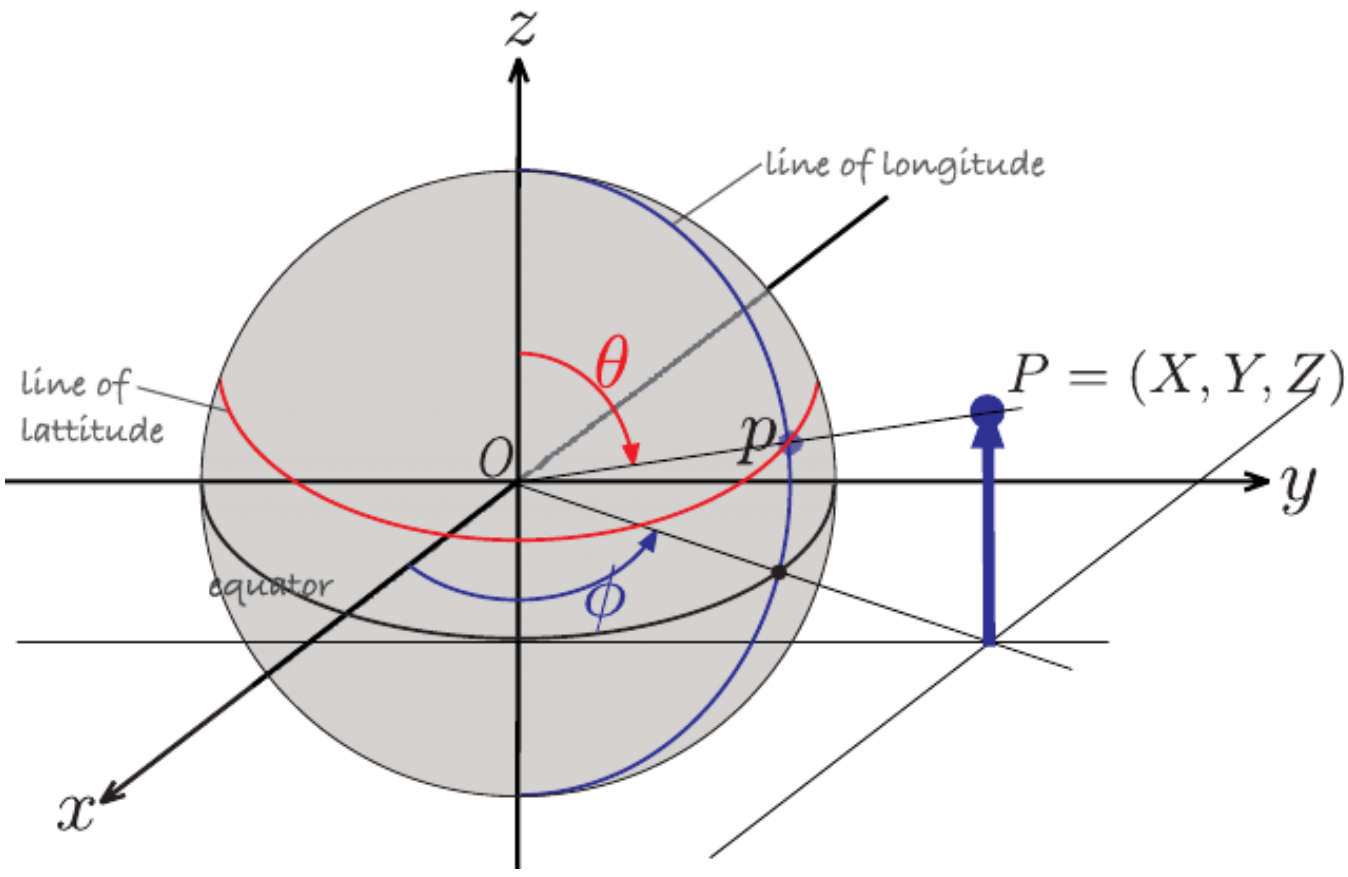
Advantages of catadioptric cameras:

- Can view 360° in azimuth.

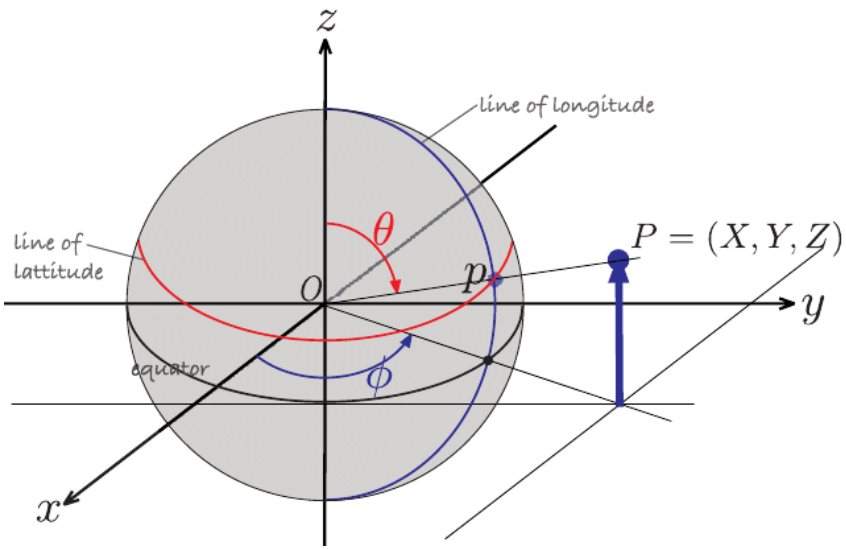
Disadvantages of catadioptric cameras:

- Reduced spatial resolution.
- Wasted image plane pixels.
- Exposure control.
- Blindspot.

Sherical camera



Sherical camera



$$x = \frac{X}{R},\ y = \frac{Y}{R},\ z = \frac{Z}{R}$$

$$R = \sqrt{X^2 + Y^2 + Z^2}$$

$$\theta = \sin^{-1} r,\ \theta \in [0, \pi]$$

$$r = \sqrt{x^2 + y^2}$$

$$\phi = \tan^{-1} \frac{y}{x},\ \phi \in [-\pi, \pi)$$

$$x = r \cos \phi$$

$$y = r \sin \phi$$

$$z = \cos \theta$$

Sherical camera

```
cam = SphericalCamera('name', 'spherical')
[X,Y,Z] = mkcube(1, 'centre', [2, 3, 1], 'edge');
cam.mesh(X, Y, Z)
```

- Spherical cameras have been demonstrated in laboratories.
- It is more useful as a conceptual constructt to simplify the discussion of wide-angle imaging.

Unified Imaging

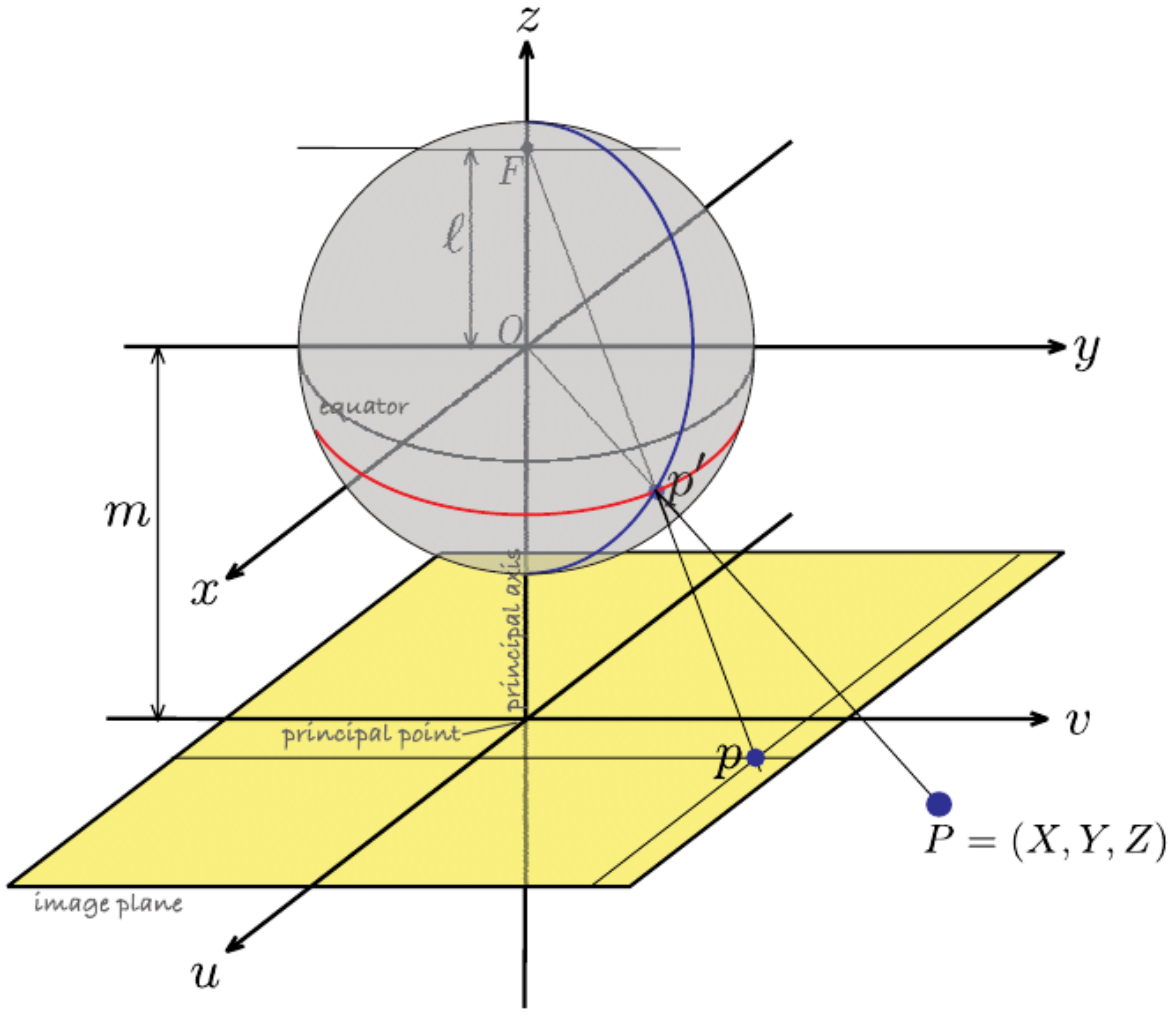


Figure: Geyer and Daniilidis (2000)

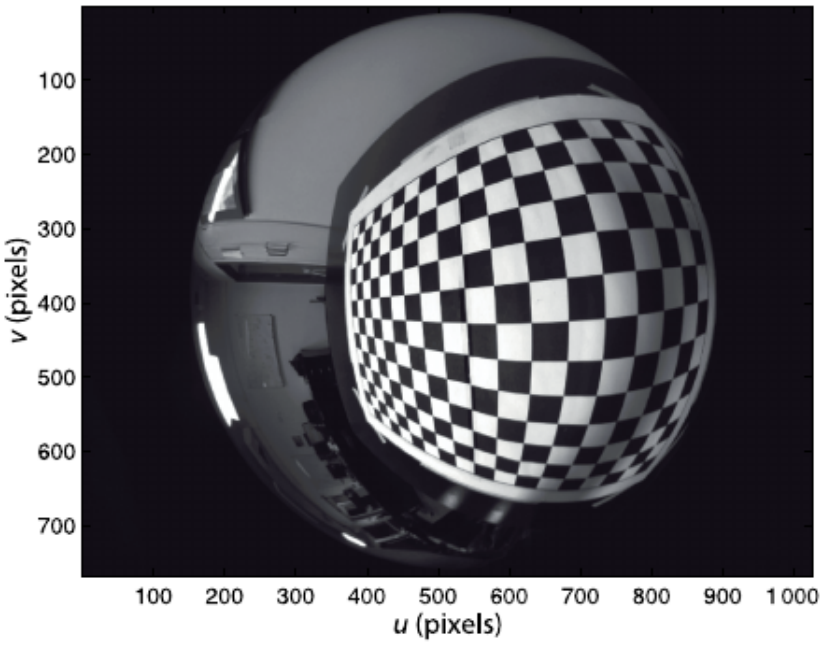
$$r = \frac{(l + m)\sin\theta}{l + \cos\theta}$$

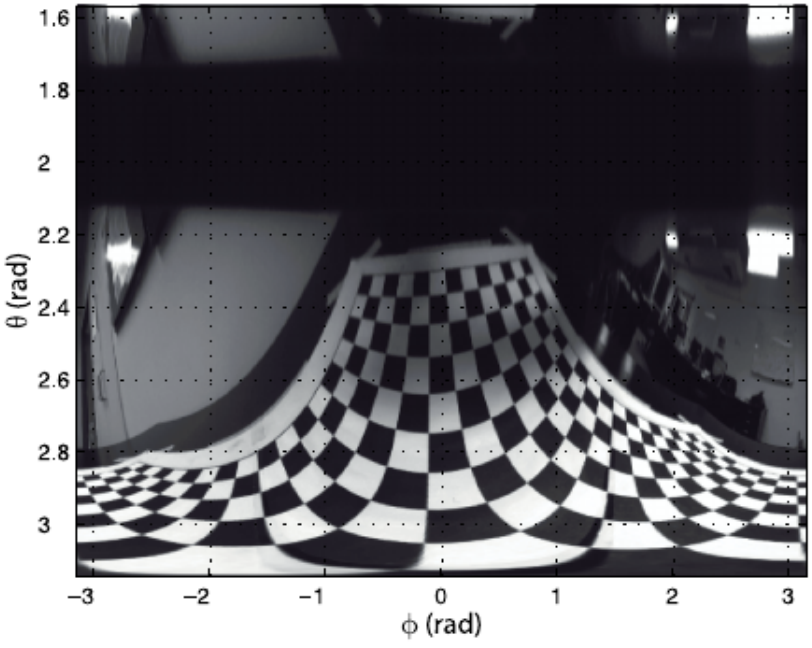
- It has two parameters m and l.
- For a perspective camera, the two points O and F coincide.
- For catadioptric cameras with mirrors that are conics the focus F lies between the center of the sphere and the north pole, that is, $0 < l < 1$.

Imaging	ℓ	m
Perspective	0	f
Stereographic	1	f
Fisheye	>1	f
Catadioptric (elliptical, $0 < \varepsilon < 1$)	$\frac{2\varepsilon}{1+\varepsilon^2}$	$\frac{2\varepsilon(2p-1)}{1+\varepsilon^2}$
Catadioptric (parabolic, $\varepsilon = 1$)	1	$2p-1$
Catadioptric (hyperbolic, $\varepsilon > 1$)	$\frac{2\varepsilon}{1+\varepsilon^2}$	$\frac{2\varepsilon(2p-1)}{1+\varepsilon^2}$

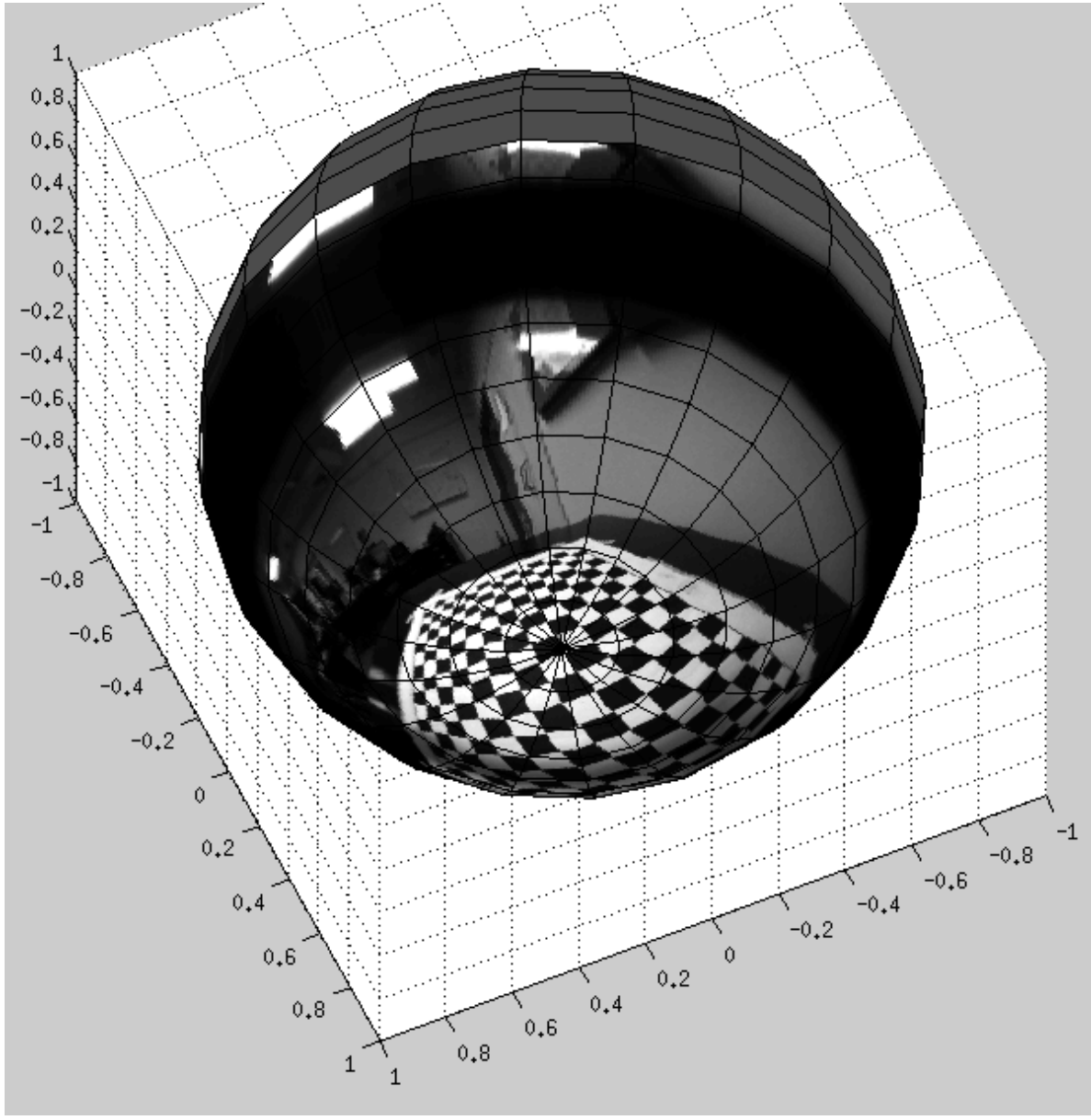
From an image obtained with a fisheye lens to a "pseudo"-spherical lens.

```
fisheye = iread('fisheye_target.png', 'double', 'grey');
u0 = 528.1214; v0 = 384.0784;
l=2.7899;
m=996.4617;
[Ui,Vi] = imeshgrid(fisheye);
n = 500;
theta_range = (0:n)/n*pi/2 + pi/2;
phi_range = (-n:2:n)/n*pi;
[Phi,Theta] = meshgrid(phi_range, theta_range);
r = (l+m)*sin(Theta) ./ (l+cos(Theta));
U = r.*cos(Phi) + u0;
V = r.*sin(Phi) + v0;
spherical = interp2(Ui, Vi, fisheye, U, V);
idisp(spherical)
```


100
200
300
400
500
600
700
v (pixels)
100 200 300 400 500 600 700 800 900 1000
u (pixels)


1.6
1.8
2
2.2
2.4
2.6
2.8
3
θ (rad)
−3 −2 −1 0 1 2 3
ϕ (rad)

```
sphere_paint(spherical, 'south')
```



Summary

- Mapping points from 3D (world) to 2D (image) is achieved by a matrix multiplaction in homogeneneous coordinates.
- Homogeneous coordinates are scale invariant.
- Mapping points from one plane to another is achieved by a matrix multiplication with the planar homography matrix.

Thank you!

tvieira@ic.ufal.br