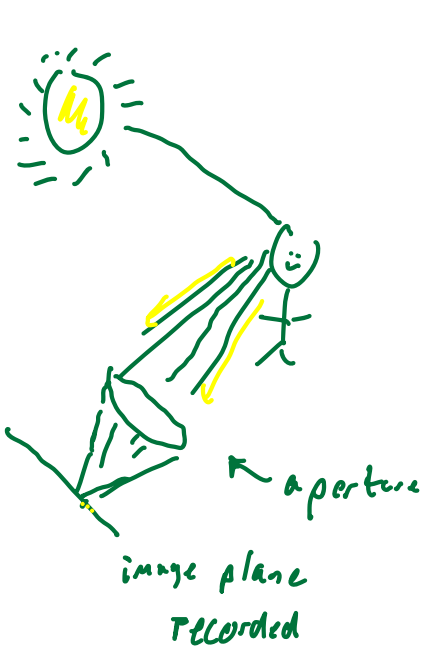
CSci 4270 and 6270
Computational Vision, Spring 2025
Lecture 5: Transformations
January 23, 2025

Transformations and Cameras

Questions to address:

- Where does a point in the "world" project to an image?
- How can we describe this geometrically?
- What information is lost, what distortions are created by the projections into an image?
- How can we describe the projections with simple approximations?



Topics

- Perspective projection
- Transformations in 2d
- Homogenous coordinates
- Transformations in 3d
- Camera matrix

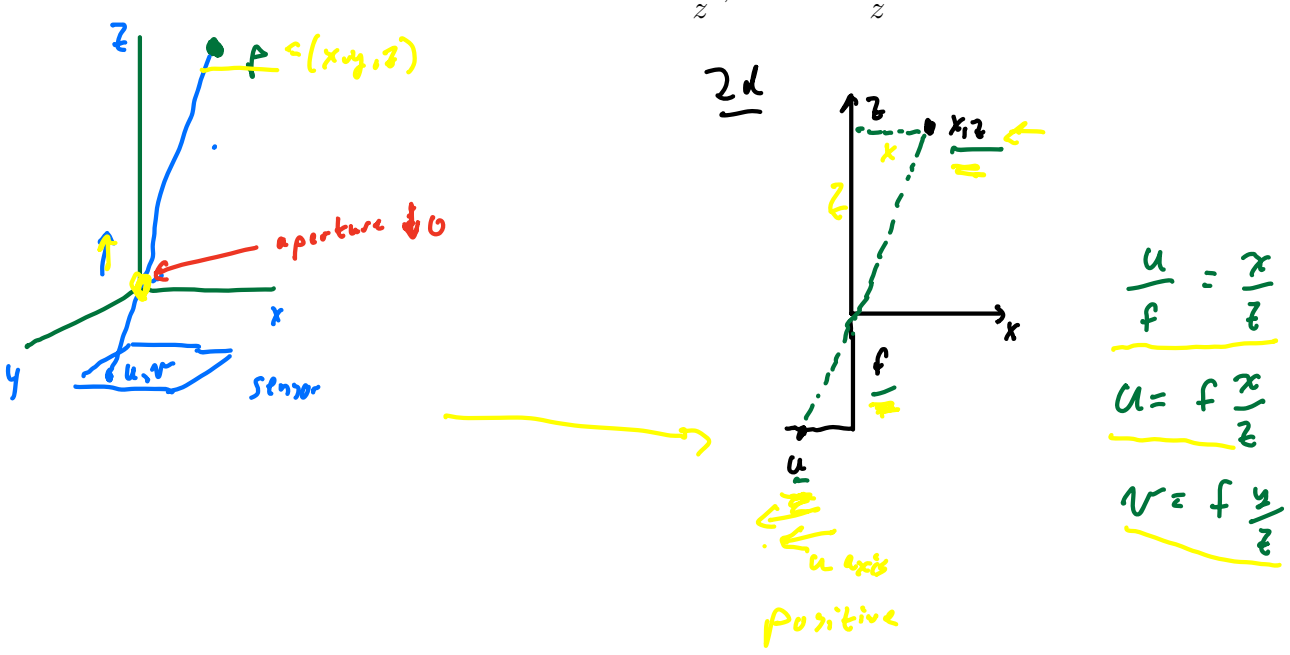
Reading Background in the Szeliski text

- Camera models and transformations: Section 2.1

Perspective Projection

- Given a point at location $\mathbf{p} = (x, y, z)^\top$ in the scene.
- Assume:
  - Camera is at origin of coordinate system
  - Optical axis of camera is the z axis
  - Infinitesmal aperture
  - Focal length of f
  - Projection follows a linear path
- Similar triangles (diagram in class) shows that the projected image location of the point is

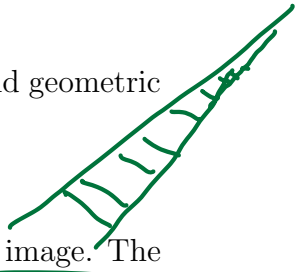
$$u = \frac{fx}{z}, \qquad v = \frac{fy}{z}$$



Implications

We can derive the following through a combination of algebraic and geometric arguments.

- Lines project onto lines
- Parallel lines in the world, in general, are not parallel in the image. The exception is lines on planes parallel to the image plane.
- Circles project onto ellipses
- A line through the origin (the camera center) projects onto a single point in the image.



Moving Away From Our Simple Perspective Model

Leads to discussion of transformations:

- Transformations to map between coordinate systems of different cameras and to map between world and camera coordinate systems
- Transformations of the (u, v) values ("camera coordinates") into pixel ("image") coordinates.
- Our discussion
  - Transformations in two dimensions
  - Homogeneous coordinates
  - Transformation in three dimensions, and especially representations of rotation

Transformations in 2d

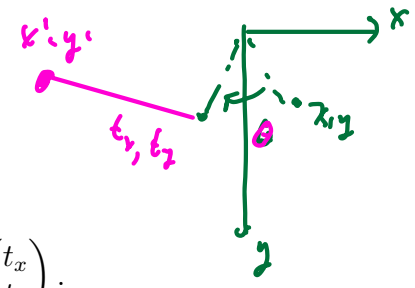
An important way to understand the different transformations is to understand what properties are preserved

- Rigid
- Similarity
- Affine
- Projective

Rigid Transformation

- Rotation and then translation (translate and then rotate is equivalent, but the values in the translation vector are different)
- Coordinate form:

$$x' = x\cos\theta - y\sin\theta + t_x$$
$$y' = x\sin\theta + y\cos\theta + t_y$$



- Matrix vector form:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}.$$

- Or, more compactly

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t}.$$

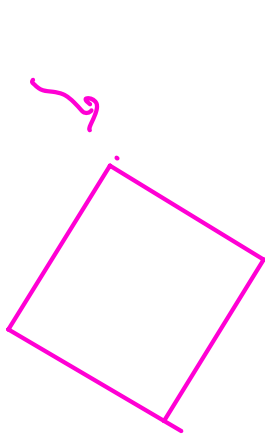
- Here we require that $\mathbf{R}^\top\mathbf{R} = \mathbf{I}$ and $\det\mathbf{R} = 1$.
- Lengths and angles are preserved.

Similarity Transformations

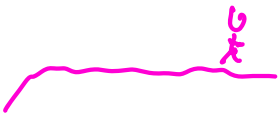
- Same as rigid, but allow scaling by a constant factor, $s > 0$:

$$\begin{aligned}\mathbf{x}' &= s\mathbf{R}\mathbf{x} + \mathbf{t} \\ &= \begin{pmatrix} s\cos\theta & -s\sin\theta \\ s\sin\theta & s\cos\theta \end{pmatrix}\mathbf{x} + \mathbf{t} \\ &= \begin{pmatrix} a & -b \\ b & a \end{pmatrix}\mathbf{x} + \mathbf{t}\end{aligned}$$

where $a = s\cos\theta$ and $b = s\sin\theta$.



- Similarity transformations preserve angles, parallelism between lines, and ratios of distances.
- Approximates the change in appearance of an image when
  - Camera is looking perpendicular to a planar surface
  - Camera movement is translation (any direction) or rotation around the optical axis
  - Think of taking photos from a plane or a UAV that is looking straight down.



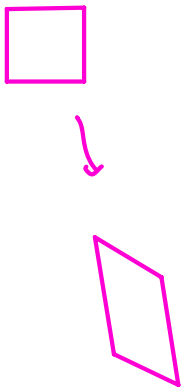
Affine Transformations

- Now the 2×2 matrix has 4 degrees of freedom:

$$\mathbf{x}' = \mathbf{A}\mathbf{x} + \mathbf{t}$$

  with the constraint that $\mathbf{A}$ is rank 2.

- Affine transformations preserve parallelism, but not angles.
- Ratios of distances of colinear line segments are unchanged.
- Good local approximation to the effects of more general transformations.
  - 1st order Taylor series expansion of the general 2d to 2d transformation.
- Approximates what happens to the image of a planar surface when the surface (or the camera) tilts, but not too much....



Homogeneous Coordinates / Augmented Vectors

Unify and simplify notation while considering more general transformations

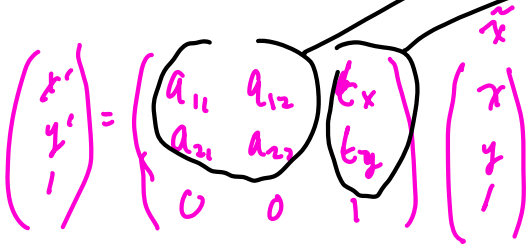
- If we add an extra component to the point vector, so that

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

- Then we can combine the affine and translations terms into a single matrix:

$$\tilde{\mathbf{x}}' = \begin{pmatrix} \mathbf{A} & \mathbf{t} \\ 0 & 1 \end{pmatrix} \tilde{\mathbf{x}}$$

- It is easy to see how to write rigid and similarity transformations using the same form.



More Than Just a Mathematical Convenience

What if we allow an arbitrary 3rd row of the matrix?

- Replace the $(0, 0, 1)$ in the bottom.
- Write a general form, without giving distinguishing names to the entries:

$$\mathbf{H} = \begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}$$

  with the constraint that $\mathbf{H}$ has rank 3!

- The transformation of a point becomes

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

- This is known as a "projective" transformation or "homography".

$\tilde{x}$ ← means homogeneous

$\vec{h}_1^T = (h_{11}, h_{12}, h_{13})$

$\vec{h}_2^T = (h_{21}, h_{22}, h_{23})$

$\vec{h}_3^T = (h_{31}, h_{32}, h_{33})$

$$H = \begin{pmatrix} \vec{h}_1^T \\ \vec{h}_2^T \\ \vec{h}_3^T \end{pmatrix}$$

$$\tilde{x}' = H\tilde{x} = \begin{pmatrix} \vec{h}_1^T \tilde{x} \\ \vec{h}_2^T \tilde{x} \\ \vec{h}_3^T \tilde{x} \end{pmatrix}$$

3x1 column vector

Handling the Last Term

- The 3rd entry in $\tilde{\mathbf{x}}'$ is no longer necessarily 1, so we must write it as

$$\tilde{\mathbf{x}}' = \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

- We then obtain the so-called "affine coordinates" that we are used to working with as

$$x' = \frac{u}{w}, \qquad y' = \frac{v}{w}.$$

- We have to be careful of special cases where $w = 0$. This gives us what are known as "points at infinity".

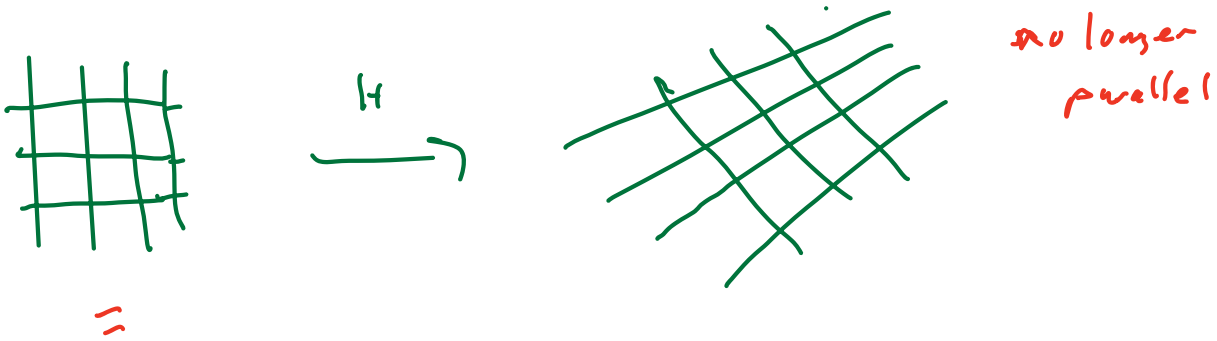
$u = \vec{h}_1^T \tilde{x}$ $\quad v = \vec{h}_2^T \tilde{x}$ $\quad w = \vec{h}_3^T \tilde{x}$

$$x' = \frac{\vec{h}_1^T \tilde{x}}{\vec{h}_3^T \tilde{x}} \qquad y' = \frac{\vec{h}_2^T \tilde{x}}{\vec{h}_3^T \tilde{x}}$$

Note H has 8 degrees of freedom, even though it has 9 params

Properties of the H matrix and the Associated Transformation

- Most general linear transform
- Collinearity is preserved.
- Lengths, angles and parallelism are not preserved.
- Good approximation to effect of perspective projection of points on a surface, as we will soon see.



What Happens to Direction and Normal Vectors?

We'll consider for now only the affine transformation and think only about lines through the origin since the translation component has no effect on the line direction.

- If $\boldsymbol{\eta}$ is the unit normal to the line then a point $\mathbf{x}$ is on the line when

$$\boldsymbol{\eta}^\top \mathbf{x} = 0.$$

- Now, consider the affine transformation

$$\mathbf{x}' = \mathbf{A}\mathbf{x}$$

- In order for $\boldsymbol{\eta}'$ to be the line normal, we must have

$$\boldsymbol{\eta}'^\top \mathbf{x}' = \boldsymbol{\eta}'^\top \mathbf{A}\mathbf{x} = 0$$

- This leads to

$$\boldsymbol{\eta}' = \mathbf{A}^{-\top}\boldsymbol{\eta}$$

  Note that $\boldsymbol{\eta}'$ is not necessarily a unit vector, so we must renormalize.

Summary of 2d Transformations

- Hierarchy based on both algebraic properties and geometric "invariants".
- Each can be used as a special-case approximation to the movement of a point in an image as a camera moves.
- A wide range of properties may be derived.
  - We have only scratched the surface...

At this point we will switch to a Jupyter notebook to look at some of the effects of various transformations on points in 2d.

play around

Transformations in 3d

rigid, similarity, affine, projective

- Similar hierarchy as 2d
- Our primary focus is rigid transformations since when we discuss 3d we are mostly concerned with physical movement in the world rather than approximations.
- Order of discussion:
  - Aside on lines and planes
  - Representation of rotations in 3d

Lines and Planes

- Generalization of the 2d implicit form of the line

$$ax + by + cz + d = 0$$

describes a plane in 3d. This can be seen by considering the point and normal form.

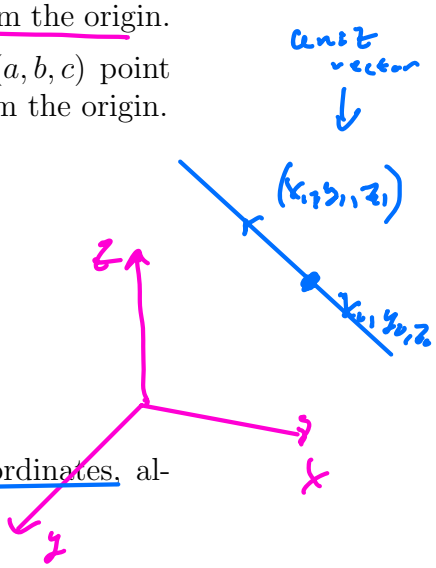
  - When (a, b, c) is a unit vector, then it is the normal to the plane, and d is the signed minimum distance of the plane from the origin.
  - The two-way sign ambiguity is removed by making (a, b, c) point away from the origin, and then $-d$ is the distance from the origin.

- The easiest-to-understand form of line in 3d is parametric:

$$x(t) = x_1 t + x_0$$
$$y(t) = y_1 t + y_0$$
$$z(t) = z_1 t + z_0$$

This has six parameters and four degrees of freedom.

- Other representations include two planes and Plücker coordinates, although we will not discuss these.



Rotations in 3d

- Three degrees of freedom
- Can be described as a 3×3 matrix $\mathbf{R}$ where $\det \mathbf{R} = 1$, $\mathbf{R}^\top \mathbf{R} = \mathbf{I}$ and, as a result, $\mathbf{R}^{-1} = \mathbf{R}^\top$.
  - As discussed in Lecture 3, one eigenvalue is 1 and the correspondence eigenvector is the axis of rotation.
  - The other two eigenvalues are $\cos\theta \pm i \sin\theta$, where θ is the axis of rotation.
- One (somewhat ugly) way to parameterize $\mathbf{R}$ is in terms of rotations around the three axes:

$$\mathbf{R} = \mathbf{R}_x \mathbf{R}_y \mathbf{R}_z$$

where

$$\mathbf{R}_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}, \quad \mathbf{R}_y = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix},$$

$$\mathbf{R}_z = \begin{pmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Order matters

$\vec{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$

$R_x R_y R_z \vec{p}$

rot around z axis ; changes x and y

rotation around new y axis

rotation around new x axis

$R_x R_y R_z \neq R_z R_y R_x$

approx equal when ϕ, θ, α are all small magnitude

Small Angle Approximation

[We will skip this in lecture.]

- Fortunately, when the rotations are small (estimated incrementally), this can be approximated as

$$\mathbf{R} \approx \begin{pmatrix} 1 & -\phi & \alpha \\ \phi & 1 & -\theta \\ -\alpha & \theta & 1 \end{pmatrix}$$

- This is not strictly a rotation matrix, but it can be converted to one using the singular value decomposition (which we haven't discussed).

Other Representations for 3d Rotations

- Axis-angle
- Unit quaternions
- See discussion in Szeliski
- We will not consider these further here.

Back to Camera Modeling

- External parameters and matrix representation
- Internal parameters
- Approximations

Simple Perspective Model as Matrix

Make use of homogeneous representation

- Recall

$$u = \frac{fx}{z} \qquad v = \frac{fy}{z}.$$

- Although it seems unnecessary at this point, we can write this as a 3×4 matrix, using homogeneous coordinates:

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}.$$

M

$u' = fx$
$v' = fy$
$w' = z$

- Since this is homogeneous coordinates, we obtain

$$u = \frac{u'}{w'} \qquad \text{and} \qquad v = \frac{v'}{w'}.$$

$u' = fx/z \quad v' = fy/z$

- We will use the variable $\mathbf{M}$ to denote the 3×4 "camera" matrix.

Writing the Camera Matrix as a Product

- We split $\mathbf{M}$ as

$$\mathbf{M} = \begin{pmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} = \mathbf{K} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

- $\mathbf{K}$ will incorporate the intrinsic parameters of the camera (currently only the focal length).
- The second matrix, which is 3×4, will describe the extrinsic parameters.
- We will consider these in turn.

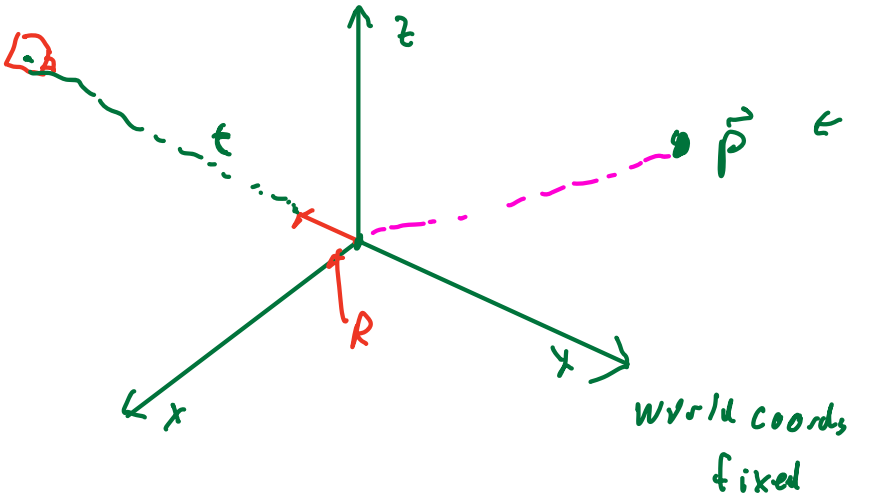
Extrinsic Parameters

Rotation and translation in world coordinates

- Camera is rotated by $\mathbf{R}$ and translated by $\mathbf{t}$ relative to the world coordinate system.
- Therefore, a point $\mathbf{p}$ in this coordinate system appears at

$$\mathbf{R}^\top(\mathbf{p} - \mathbf{t})$$

relative to the camera coordinate system



Extrinsic Parameters, Continued

Writing this in matrix form:

- If we use homogeneous coordinates for $\mathbf{p}$, writing $\tilde{\mathbf{p}}$, we can express this as a 3×4 matrix:

$$\begin{pmatrix} \mathbf{R}^\top & -\mathbf{R}^\top \mathbf{t} \end{pmatrix} \tilde{\mathbf{p}}$$

- Our 3×4 perspective projection $\mathbf{M}$ now becomes

$$\mathbf{M} = \mathbf{K} \begin{pmatrix} \mathbf{R}^\top & -\mathbf{R}^\top \mathbf{t} \end{pmatrix}$$

- For convenience we usually replace $\mathbf{R}' = \mathbf{R}^\top$ by $\mathbf{R}$ and $\mathbf{t}' = -\mathbf{R}\mathbf{t}$, reversing the transformation. Then we drop the $'$ and just write

$$\mathbf{M} = \mathbf{K} \begin{pmatrix} \mathbf{R} & \mathbf{t} \end{pmatrix}$$

Intrinsic Parameters

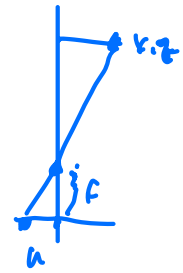
Convert from camera coordinates (metric) to image coordinates (pixels)

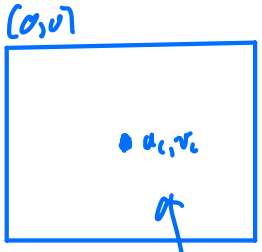
- Let (u_c, v_c) be the optical center in pixel coordinates.
- Let (d_x, d_y) be the pixel dimensions (e.g. meters per pixel)
- Then the pixel location of $u = fx/z$ and $v = fy/z$ becomes

$$u = \frac{fx}{z}\frac{1}{d_x} + u_c \qquad v = \frac{fy}{z}\frac{1}{d_y} + v_c$$

or, letting $s_x = f/d_x$ and $s_y = f/d_y$, we get

$$u = s_x\frac{x}{z} + u_c, \qquad v = s_y\frac{y}{z} + v_c$$





The Intrinsic Parameter Matrix

- Using homogenous coordinates, we can write this as

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = \begin{pmatrix} s_x & 0 & u_c \\ 0 & s_y & v_c \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$u' = s_x x + u_c z$
$v' = s_y y + v_c z$
$w' = z$

- We now have our upper-triangular intrinsic parameter matrix

$$\mathbf{K} = \begin{pmatrix} s_x & 0 & u_c \\ 0 & s_y & v_c \\ 0 & 0 & 1 \end{pmatrix}$$

$\frac{u'}{w'} = s_x \frac{x}{z} + u_c$
$\frac{v'}{w'} = s_y \frac{y}{z} + v_c$

- Sometimes the $(1, 2)$ coordinate is non-zero; it represents the cosine of the angle between the x and y axes. In most cases this angle is very close to 90°.

Putting It All Together

- We write our 3×4 camera matrix as

$$\mathbf{M} = \mathbf{K}\begin{pmatrix}\mathbf{R} & \mathbf{t}\end{pmatrix}$$

- When $\mathbf{M}$ is known we map a point in the world into the camera as:

  1. Write it in homogeneous coordinates
  2. Rotate and translate into the camera coordinates
  3. Multiply by the intrinsic matrix
  4. Convert to affine coordinates

- We get a simplified view of this by just considering $\mathbf{M}$ as a 3×4 matrix. Then we form homogenous coordinates, apply matrix multiplication, and convert back to affine coordinates.

Looking Ahead

- Finished with background:
  - OpenCV and NumPy
  - Linear algebra
  - Points, lines, distance
  - Estimation
  - Transformations and cameras
- Delved some into computer vision problems as illustrations.
- Now we are going back to the beginning, starting with image processing.
- Techniques we have discussed will be used repeatedly.