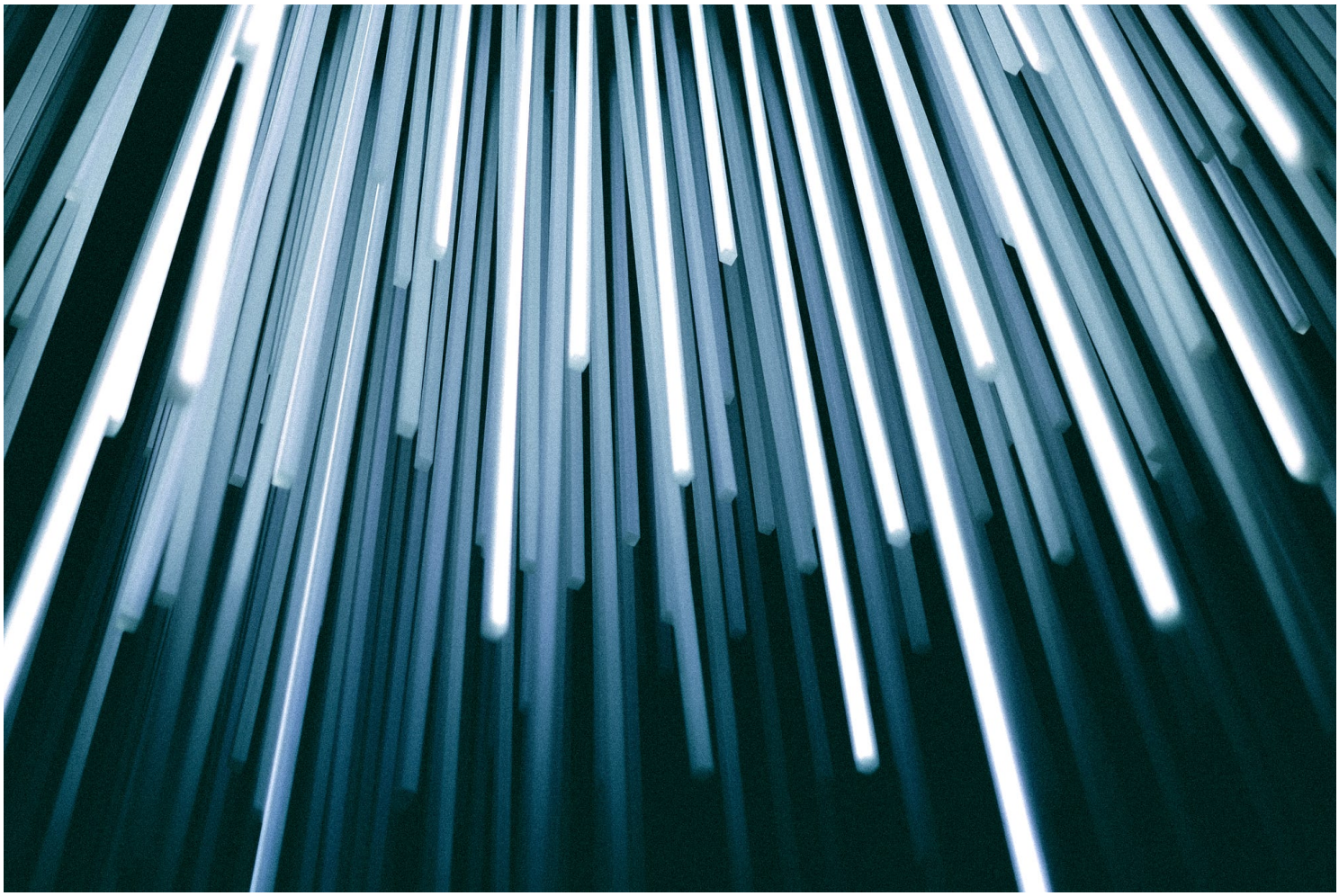


Photo by Christopher Burns on Unsplash

Vision:

An Overview of Affine Transformations and Homographies

Felix Liu · Follow

Published in Towards Data Science

9 min read · Aug 4, 2021

Contents

Transformations make up an important part of computer vision and understanding how they work lays the basis for much more advanced techniques.

Here, I will primarily cover the **affine transform** and **homography.**

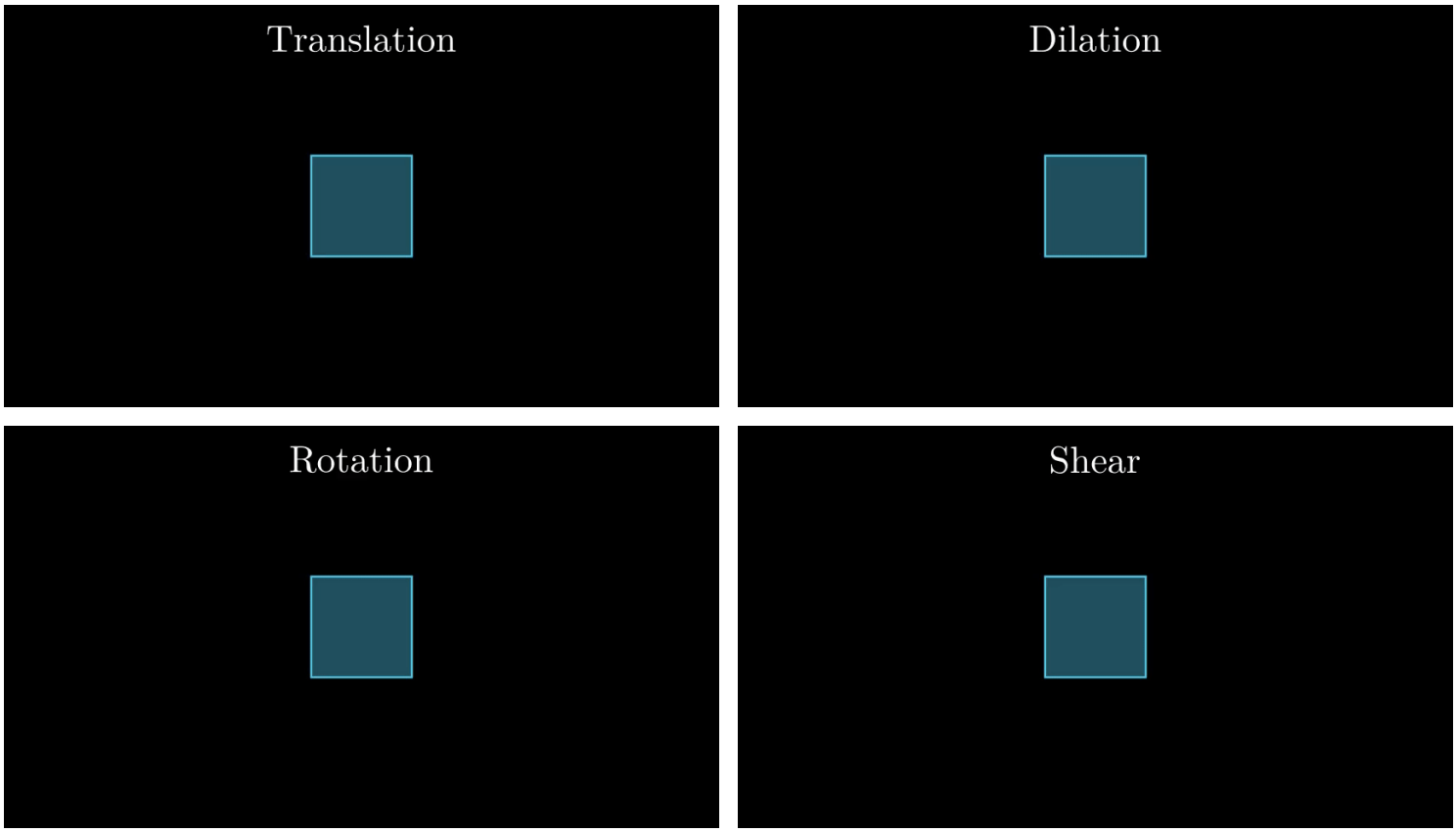
Affine Transformations:

Affine transformations are the simplest form of transformation. These transformations are also **linear** in the sense that they satisfy the following properties:

- Lines map to lines
- Points map to points
- Parallel lines stay parallel

Some familiar examples of affine transforms are **translations**, **dilations**, **rotations**, **shearing**, and **reflections.** Furthermore, any composition of these transformations

(like a rotation after a dilation) is another affine transform.



examples of basic affine transformations (image by author)

Rather than splitting up affine transforms into separate cases, it is much more elegant to have one unified definition. Thus, we turn towards the use of matrices as linear transformations to define affine transforms. If you are unfamiliar with interpreting matrices as linear transformations of space, 3Blue1Brown has an excellent video on the topic:

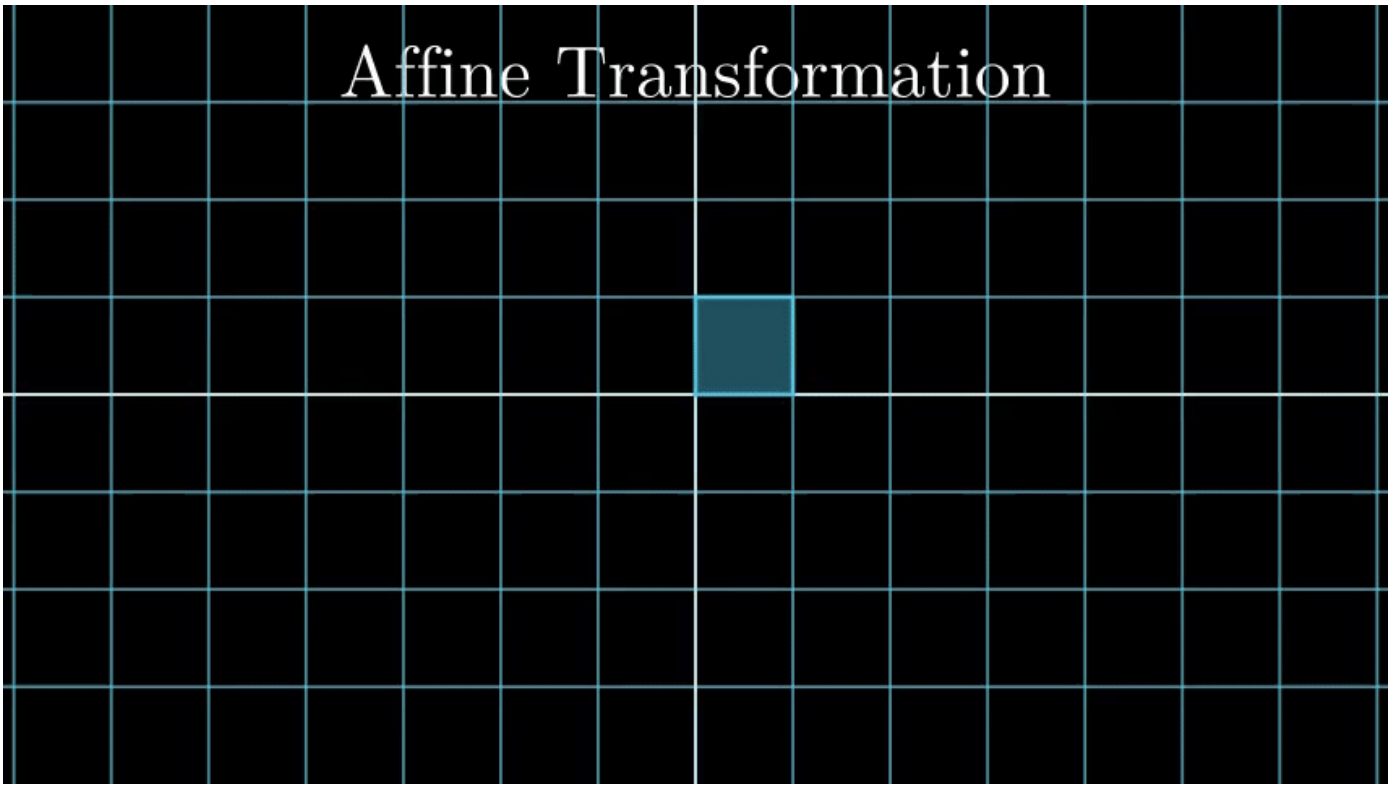
Linear transformations and matrices | Chapter 3, Essence of linear algebra

In the 2D case, the equation for an affine transform is given by:

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

equation for a 2D affine transform (image by author)

Here, the matrix represents some linear transform on the vector with entries ($x1$ and $x2$), such as a reflection, shear, rotation, dilation, or a combination of all four. It is important to note that, since the transformation is linear, it must also be invertible, so the determinant of the matrix is non-zero. The final step of the transform would be a translation by the vector [$t1$, $t2$], completing the transformation onto the vector [$y1$, $y2$].



example of an affine transform in 2D (image by author)

Affine transformations can also be generalized to n dimensions with the following equation:

$$y = Ax + b$$

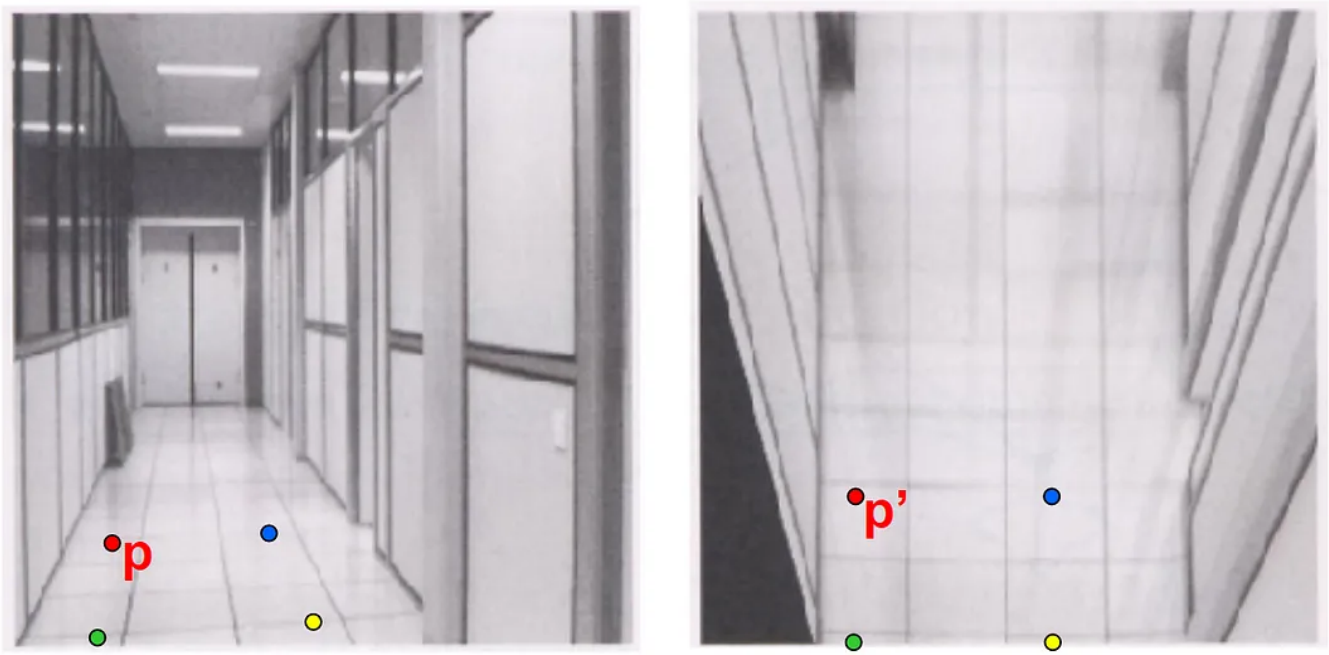
equation for n dimensional affine transform

This transformation maps the vector x onto the vector y by applying the linear transform A (where A is a $n \times n$, invertible matrix) and then applying a translation with the vector b (b has dimension $n \times 1$).

In conclusion, affine transformations can be represented as linear transformations composed with some translation, and they are extremely effective at modifying images for computer vision. In fact, image pre-processing relies heavily on affine transforms for scaling, rotating, shifting, etc.

Homography:

A homography is a type of projective transformation in that we take advantage of projections to relate two images. Homographies were originally introduced to study shifts in perspective, and they have enabled people to better understand how images change when we look at them from a different perspective.

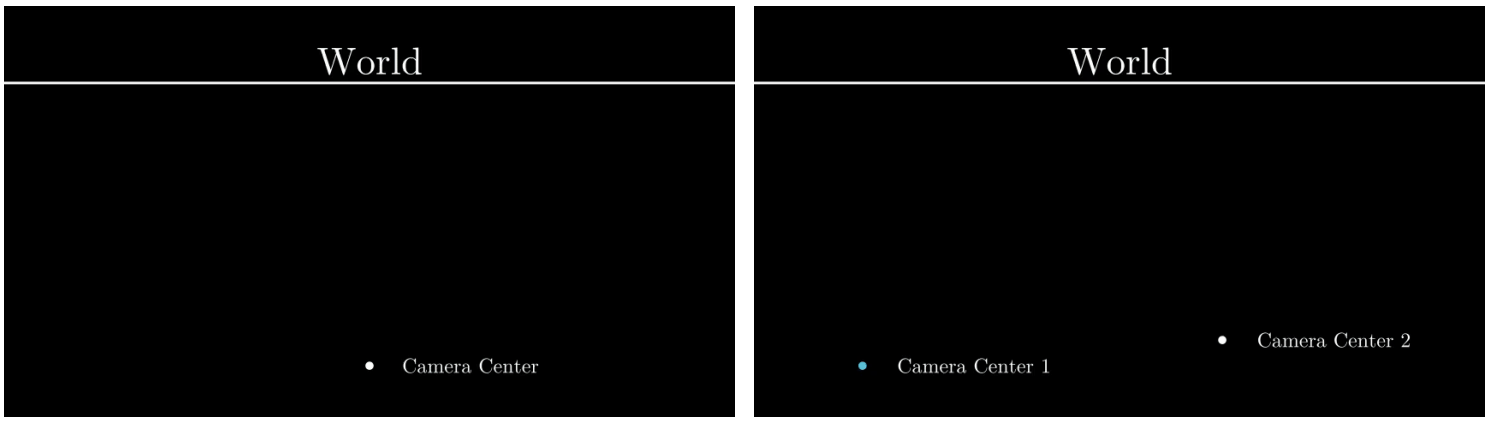


using a homography to shift perspectives (credit: University of Maryland)

Homographies are studied by examining the fact that cameras pick up different images depending on their position and orientation. In essence, a homography is a transformation between two images of the same scene, but from a different perspective. There are two only cases for which homography applies (both cases assume that the world view can be modeled by plane):

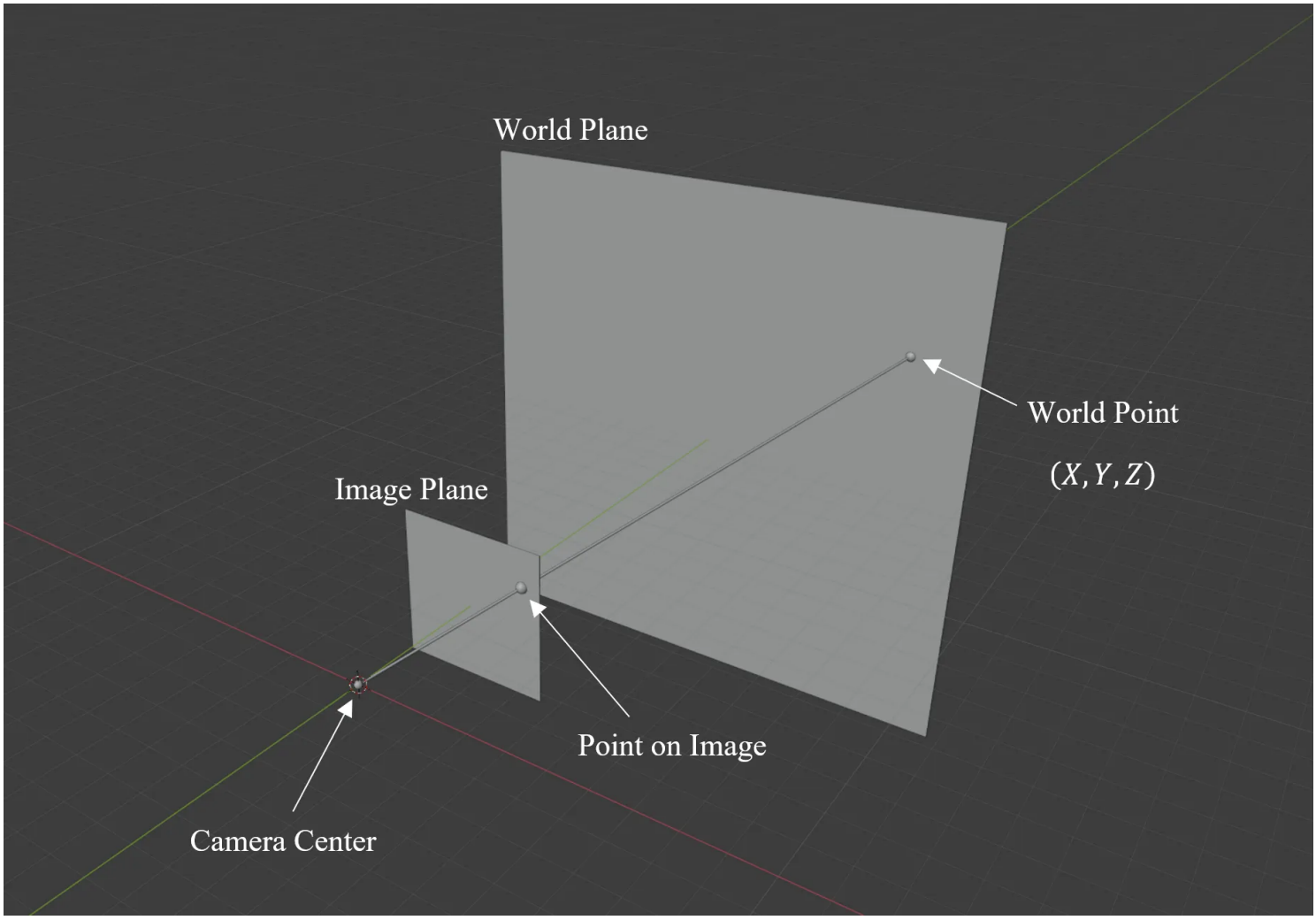
1. Images are captured by the **same camera** but at a **different angle** (the world is now essentially a plane)

2. Two cameras are viewing the **same plane** from a **different location**



two cases for a homography from a bird's-eye view (image by author)

However, in order to use a homography, we must first devise tools to describe how an image would turn out on a camera. More specifically, if we are given the location/orientation of a **camera** and an **image plane** (which is where the image appears and is a property of the camera), we must find where a point in the world (called the **world point**) appears on the image plane.



finding the coordinates for the point on the image is key for homography (image by author)

Because light travels in a straight line, we know that a world point with Cartesian coordinates (X, Y, Z) appears on the image plane at the intersection of the image plane with the line going through the camera center and world point (this line is the path that light takes to the camera).

In order to properly define formulas for this, however, we must take detour to a concept in projective geometry called **homogeneous coordinates.**

Homogeneous Coordinates:

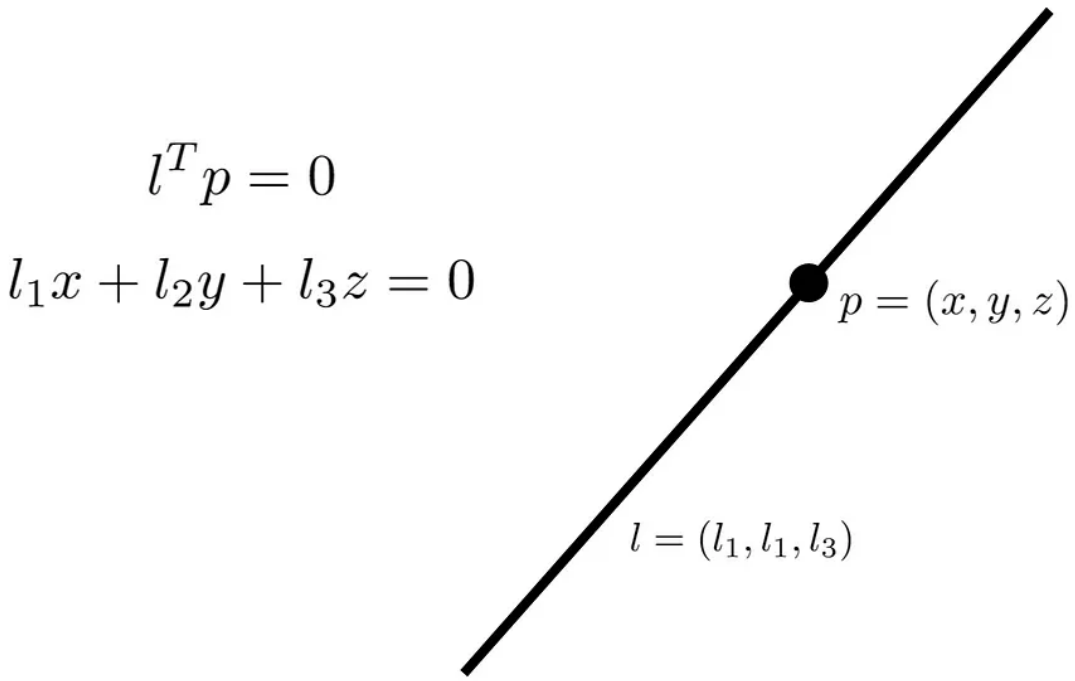
Homogeneous coordinates (or **projective coordinates**) are another coordinate system with the advantage that formulas with homogeneous coordinates are often much simpler than in **Cartesian coordinates** (points on the *x*-*y* plane). In contrast, homogeneous coordinates use 3 coordinates (*x'*, *y'*, *z'*) to represent points in space (in general, they use one more coordinate than their Cartesian counterparts). As a bonus, converting homogeneous coordinates to Cartesian coordinates is relatively simple:

$$(x', y', z') \to (x, y): \quad \begin{matrix} x = x'/z' \\ y = y'/z' \end{matrix}$$

converting homogeneous coordinates into Cartesian coordinates

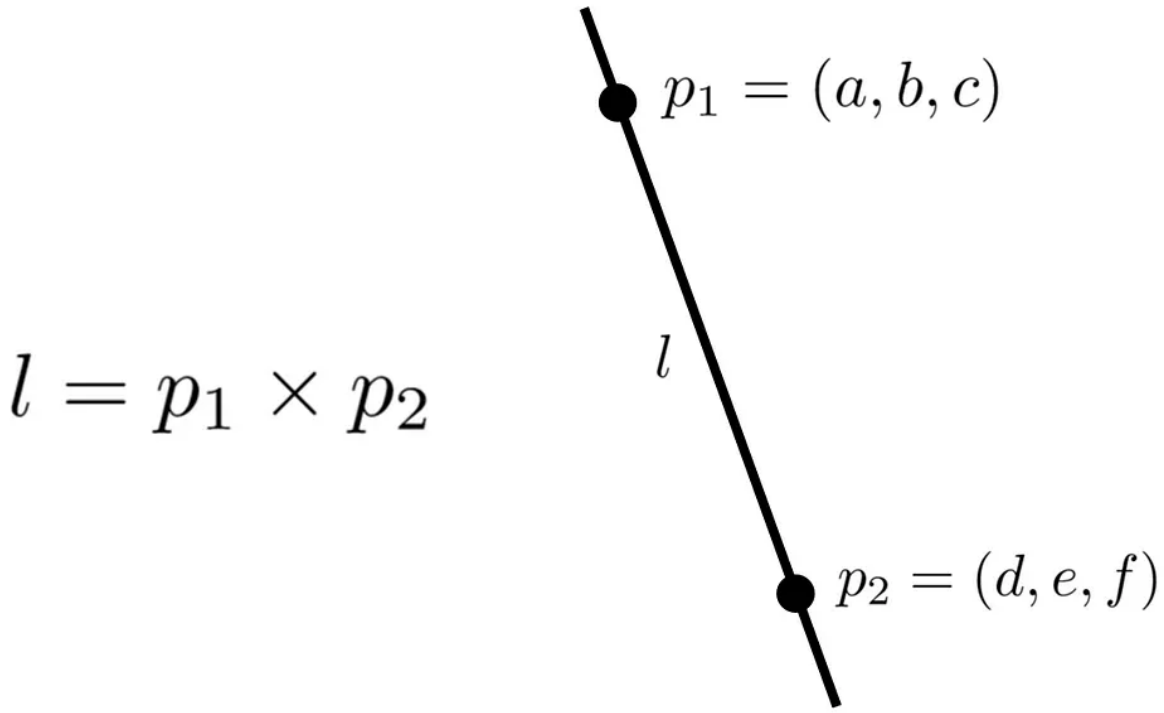
Notice that when *z'*=1, the homogeneous coordinates map nicely to Cartesian coordinates. Furthermore, when *z'*=0, we interpret the corresponding Cartesian coordinate as a point at infinity. With this in mind, let us examine some applications of homogenous coordinates.

First of all, lines in homogeneous coordinates can be represented with only 3 points. In particular, you can represent a line *l* as (*l*1, *l*2, *l*3) in homogeneous coordinates. Then, the line is simply the set of all points *p*=(*x*, *y*, *z*) such that the dot product of *l* with *p* is 0.



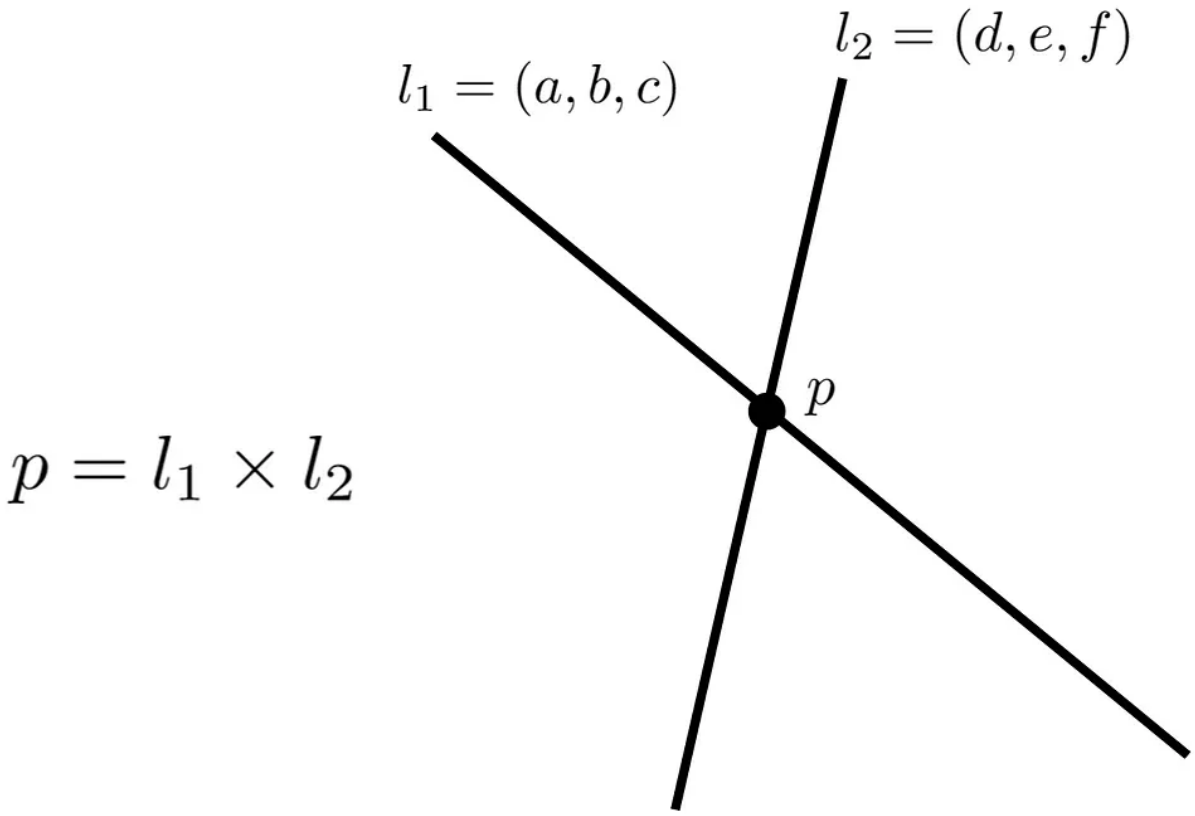
line represented in homogeneous coordinates (image by author)

The second property of note is that, given two points $p1$=(a, b, c) and $p2$ = (d, e, f), the line l going through $p1$ and $p2$ is given by the **cross-product** (an operation on two vectors defined in elementary linear algebra) of the two points.



line between two points in homogeneous coordinates (image by author)

The final property of note is that the intersection of two lines $l1$ and $l2$ (in homogeneous coordinates) is the point p given by the cross-product of the two lines.



intersection of two lines in homogeneous coordinates (image by author)

With these properties, we are now well-equipped to understand the formulas behind homographies.

Pinhole camera model:

We now return to our problem of finding where a **world point** (*X*, *Y*, *Z*) — this is in Cartesian coordinates — lies the **image plane** of a camera given the location and orientation of said camera.

In particular, we find that the world point (*X*, *Y*, *Z*, 1) lies on the point (*x'*, *y'*, *z'*) of the image plane, where both points are in homogeneous coordinates and are related by the following equation (*f* is a constant describing the focal length of the camera):

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

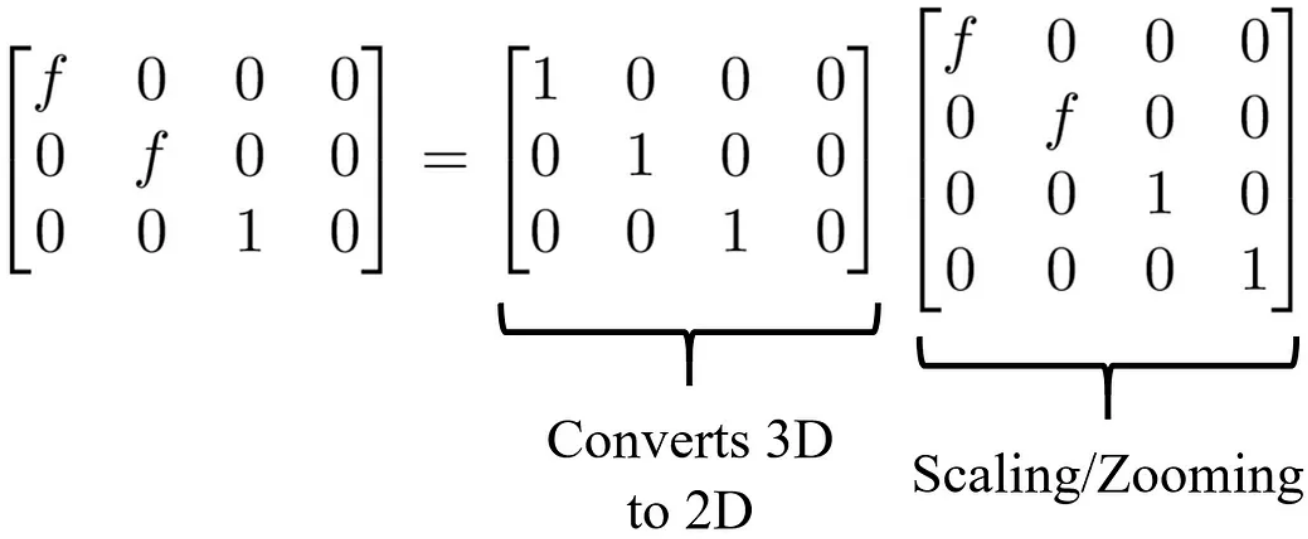
$$\downarrow$$

$$x' = fX$$
$$y' = fY$$
$$z' = Z$$

equation relating homogeneous coordinate of world point to homogenous coordinate on image plane

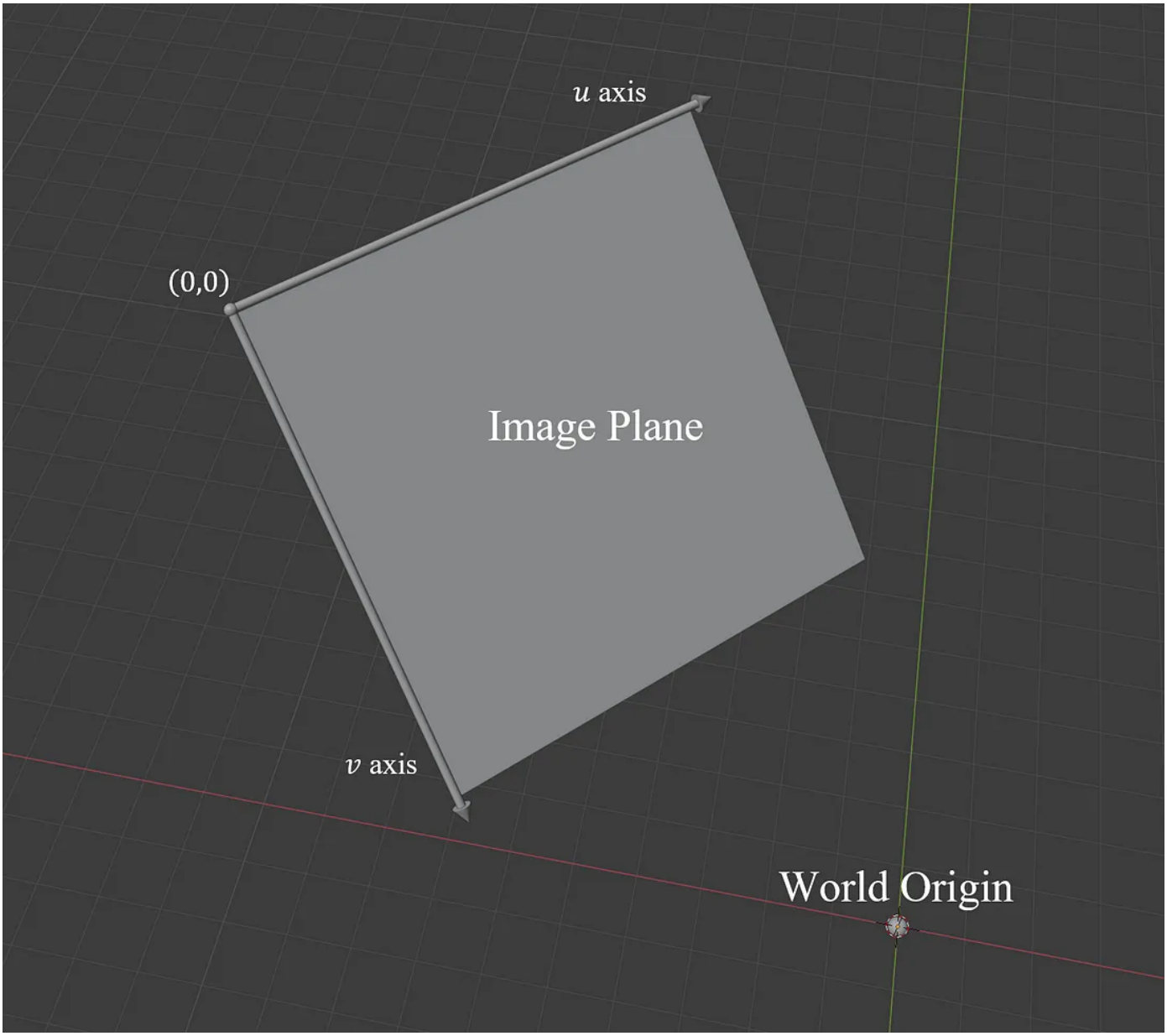
From here, we can convert (x', y', z') into Cartesian coordinates (x, y) on the image plane as described earlier.

It turns out that the matrix that transforms the world point into a point on the image plane can be factored nicely, giving us an intuitive way to understand what the matrix is doing:



the matrix can be understand as two actions: (1) scaling, (2) compressing to a lower dimension

However, since the coordinate system on images is different from Cartesian coordinates (the top left corner is (0,0) in pixel coordinates), we must perform one final transform to convert from homogeneous image plane coordinates (*x'*, *y'*, *z'*) into homogeneous **pixel coordinates** (*u'*, *v'*, *w'*).



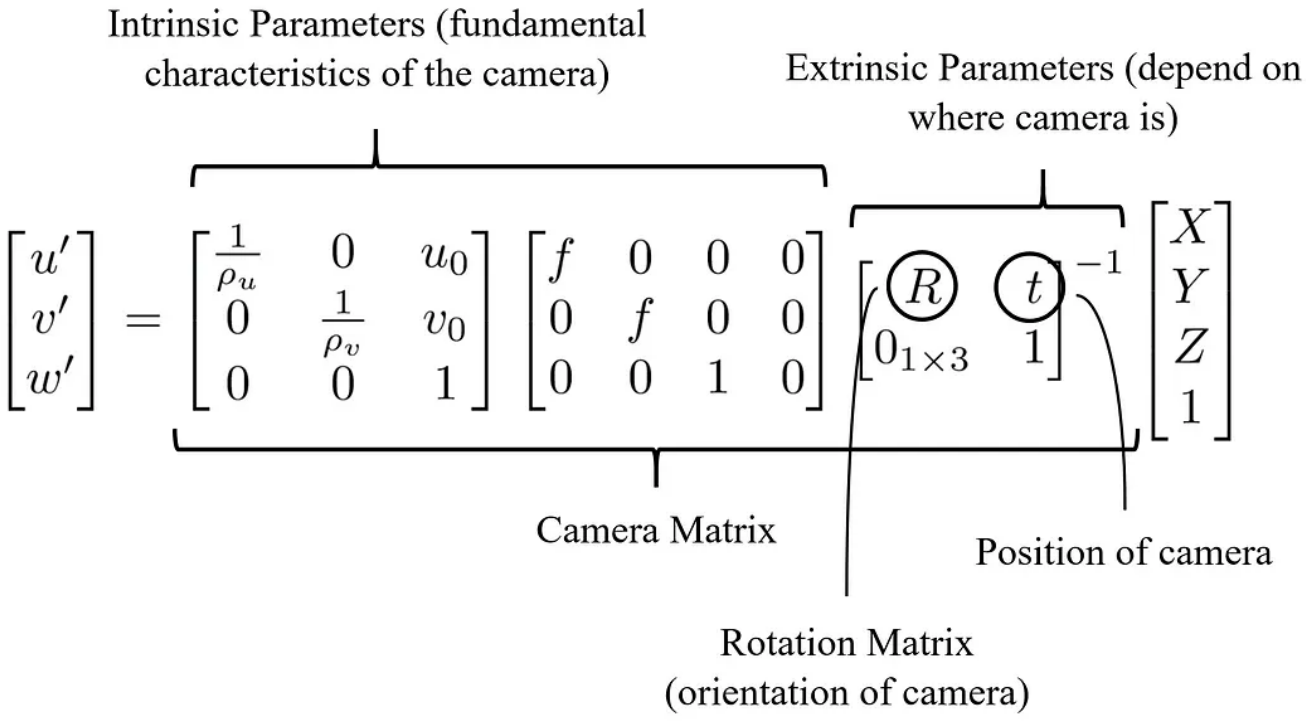
representation of pixel coordinates (image by author)

Thus, we first scale our image plane coordinates (in order to convert them into pixels) by dividing by the size of the pixel, which is ρu in the u direction and ρv in the v direction. You can think of this as converting from a unit like meters into pixels. Next, we must translate the coordinates by some $u0$ and $v0$ such that the origin of the pixel coordinates is in the proper spot. Putting these two transformations, we get the following formula:

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

transforming image plane coordinates into pixel coordinates.

Finally, putting everything together, we can complete our camera model by converting world point (*X, Y, Z,* 1) into pixel coordinates (*u', v', w'*), where both are given in homogeneous coordinates:



equation for the camera model

The extrinsic parameters can be thought of as a term that gives us information about the orientation (stored in the rotation matrix *R*) and the position (stored in the vector *t*) of the camera.

In practice, we don't often know all the parameters, so we can approximate the camera matrix instead, which is just some 3×4 matrix (by matrix multiplication properties); this is accomplished by calibrating the camera.

Furthermore, one important property of the camera matrix is that scaling the entries of C gives a new camera matrix that describes the same camera. This is because if we multiply the C by some constant λ, then all the entries of the **homogeneous** pixel coordinate will also be scaled by λ. However, when we convert from homogeneous to Cartesian coordinates (which was shown earlier), the λ's cancel out, leaving the same Cartesian coordinate as before the scaling. Thus, it has become convention to let the lower right entry of the camera matrix be 1, since scale factors are arbitrary.

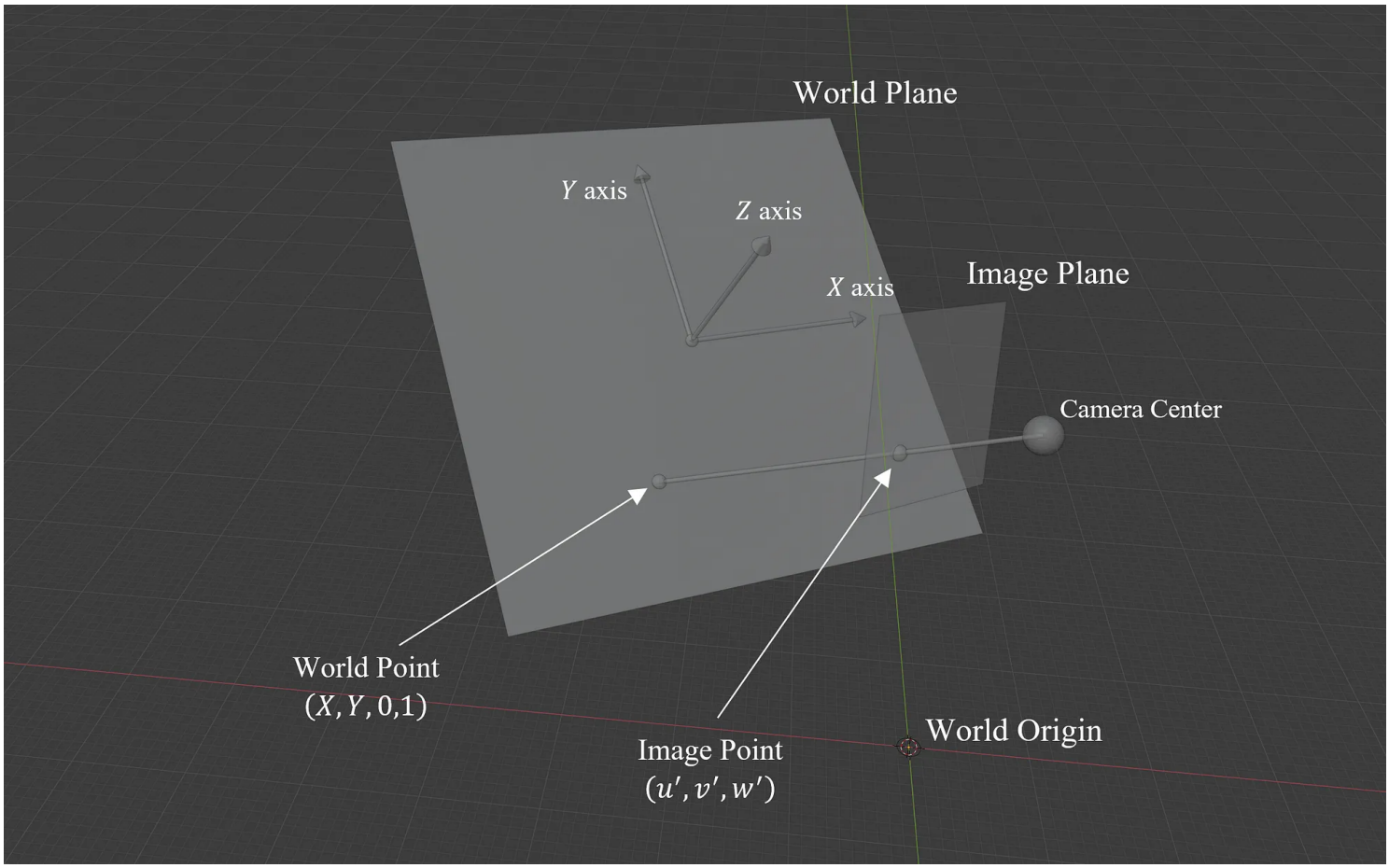
$$C = \begin{bmatrix} C_{11} & C_{12} & C_{13} & C_{14} \\ C_{21} & C_{22} & C_{23} & C_{24} \\ C_{31} & C_{32} & C_{33} & 1 \end{bmatrix}$$

standard form of the camera matrix

With these equations defined, we are very close to attaining a formula for the homography.

Equation of Homography:

Once again, let us examine the pinhole camera setup. In this setup, our camera will be taking a picture of points $P=(X, Y, Z)$ that lie on a plane, where a point P appears as (u', v', w') in homogeneous pixel coordinates on the image plane. Now, we can apply a clever trick with our choice of coordinate system. Namely, we let the X and Y axes lie in the plane such that the Z axis points out of the plane. Therefore, the Z coordinate for all points on the plane is 0, meaning that P is of the form $P=(X, Y, 0)$.



convenient choice of coordinate systems (**coordinates shown are homogeneous**) (image by author)

Now, substituting this into our camera equation and using with Z=0, we get the following simplification:

$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & \cancel{C_{13}} & C_{14} \\ C_{21} & C_{22} & \cancel{C_{23}} & C_{24} \\ C_{31} & C_{32} & \cancel{C_{33}} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ \emptyset \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} C_{11} & C_{12} & C_{14} \\ C_{21} & C_{22} & C_{24} \\ C_{31} & C_{32} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

taking advantage of Z=0 in the camera equation

This new camera matrix is known as the **homography matrix** *H* and has 8 unknown entries (since the 1 at the lower right is fixed). Conventionally, the entries of the planar homography matrix are denoted with *H* rather than *C* as shown below:

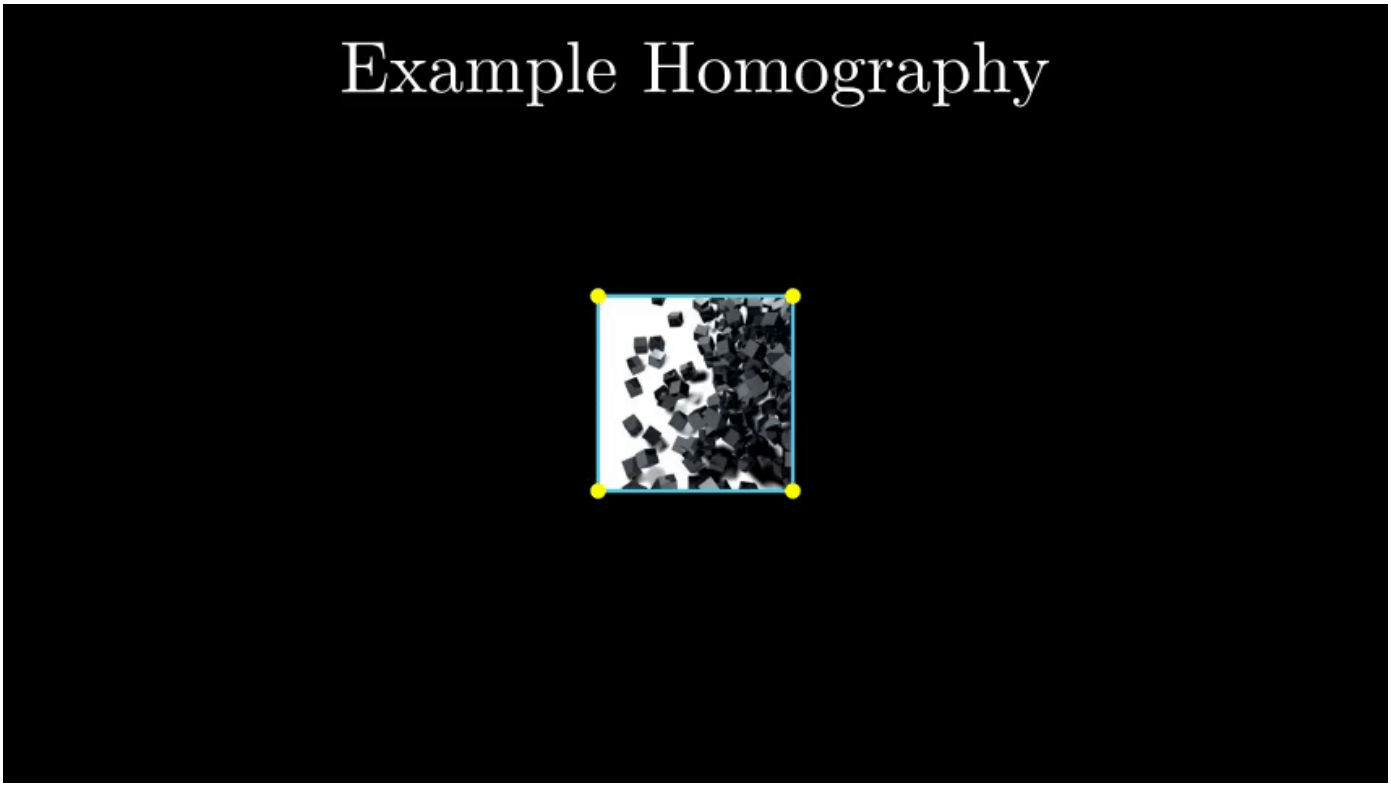
$$\begin{bmatrix} u' \\ v' \\ w' \end{bmatrix} = H \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

$$H = \begin{bmatrix} H_{11} & H_{12} & H_{14} \\ H_{21} & H_{22} & H_{24} \\ H_{31} & H_{32} & 1 \end{bmatrix}$$

equation for a planar homography

Our final task is now to estimate the entries of H. Since there are 8 unknowns in the homography matrix, and each world point on the plane has 2 coordinates of note, we will need to **calibrate the camera on 4 world points** in order to estimate all the entries of H.

Recapping, homographies are a transformation that we can use to transform one image into another when both images are pictures of the same plane, but from a different perspective. Mathematically, this transformation is carried out by the homography matrix, which is 3×3 matrix that has 8 unknowns and can be estimated by calibrating the images with 4 corresponding points (using more points gets a better approximation, but 4 is the minimum required).



example homography with calibration points in yellow (image by author)

All in all, homgraphies are a powerful tool with applications in areas like **augmented reality** (to project certain images onto the environment) and **image stitching** (combining multiple images to create a larger panorama).

References

https://en.wikipedia.org/wiki/Homogeneous_coordinates

http://www.cs.umd.edu/class/fall2019/cmsc426-0201/files/16_Homography-estimation-and-decomposition.pdf

https://medium.com/swlh/image-processing-with-python-image-warping-using-homography-matrix-22096734f09a

https://www.youtube.com/watch?v=l_qjO4cM74o

https://www.youtube.com/watch?v=fVJeJMWZcq8

Computer Vision | Machine Learning | Mathematics | Artificial Intelligence | Math

Follow

Written by Felix Liu

24 Followers · Writer for Towards Data Science

Hi, my name is Felix Liu and I am a student interested in mathematics and how it applies to the real world.

More from Felix Liu and Towards Data Science

Felix Liu in Towards Data Science

The Math Behind Deepfakes

This article will provide an introductory look into the foundational math that goes into creating a deepfake with a GAN.

· 17 min read · Aug 7, 2020

Jacob Marks, Ph.D. in Towards Data Science

How I Turned My Company's Docs into a Searchable Database with OpenAI

And how you can do the same with your docs

15 min read · Apr 25

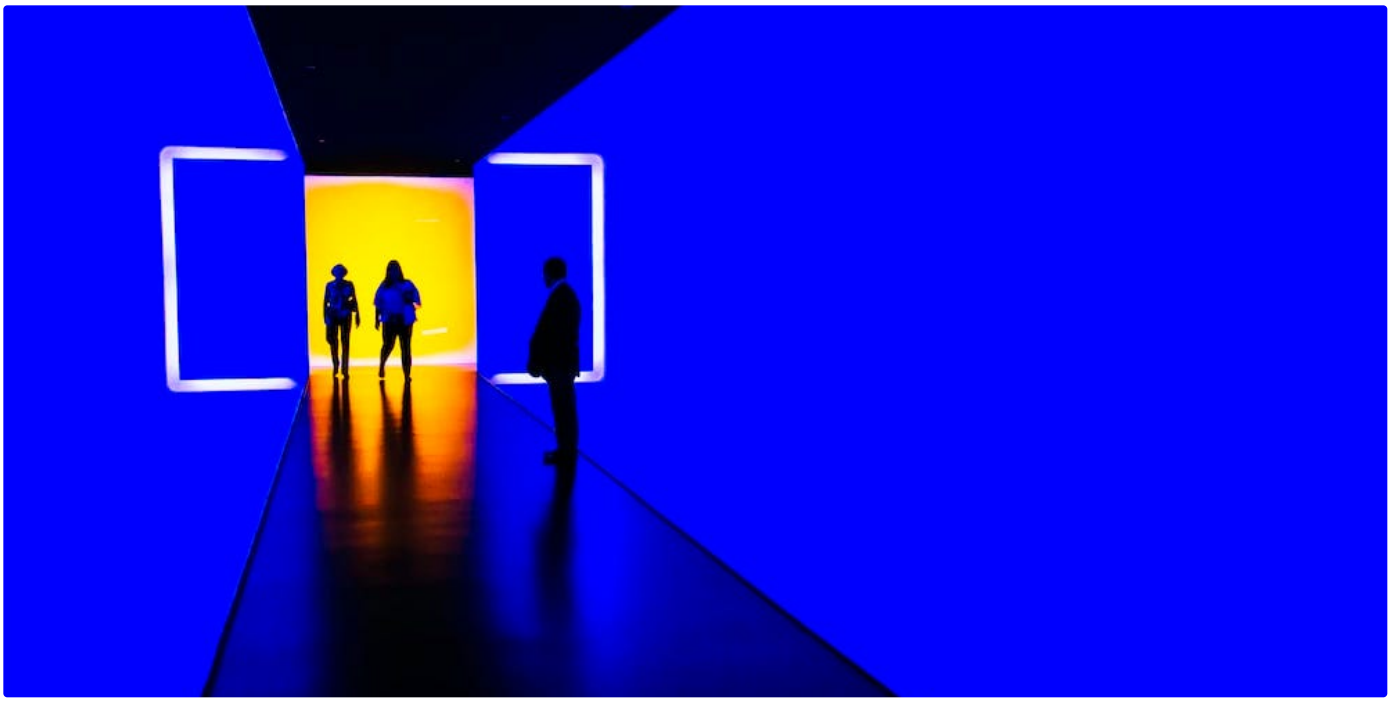
Leonie Monigatti in Towards Data Science

Getting Started with LangChain: A Beginner's Guide to Building LLM-Powered Applications

A LangChain tutorial to build anything with large language models in Python

· 12 min read · Apr 25

2.2K 18



Matt Chapman in Towards Data Science

How I Stay Up to Date With the Latest AI Trends as a Full-Time Data Scientist

No, I don't just ask ChatGPT to tell me

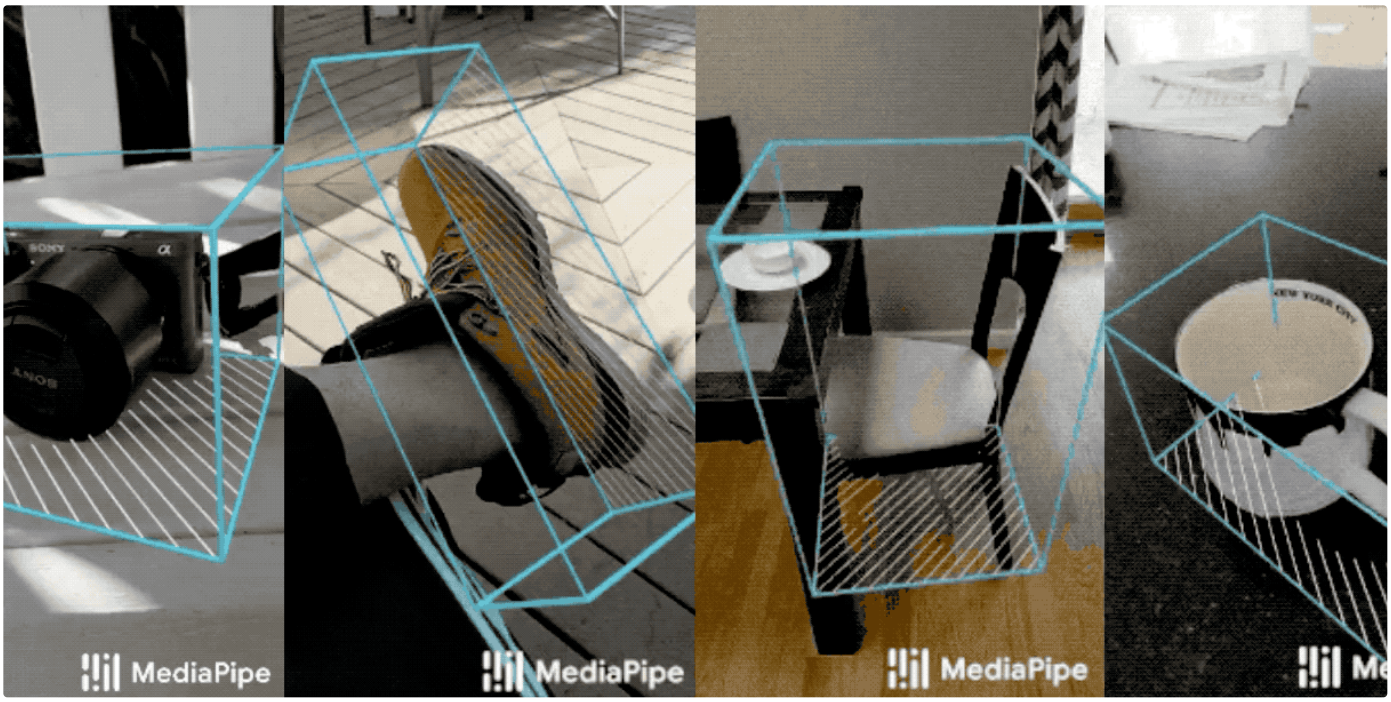
· 8 min read · May 1

1.4K 21

See all from Felix Liu

See all from Towards Data Science

Recommended from Medium



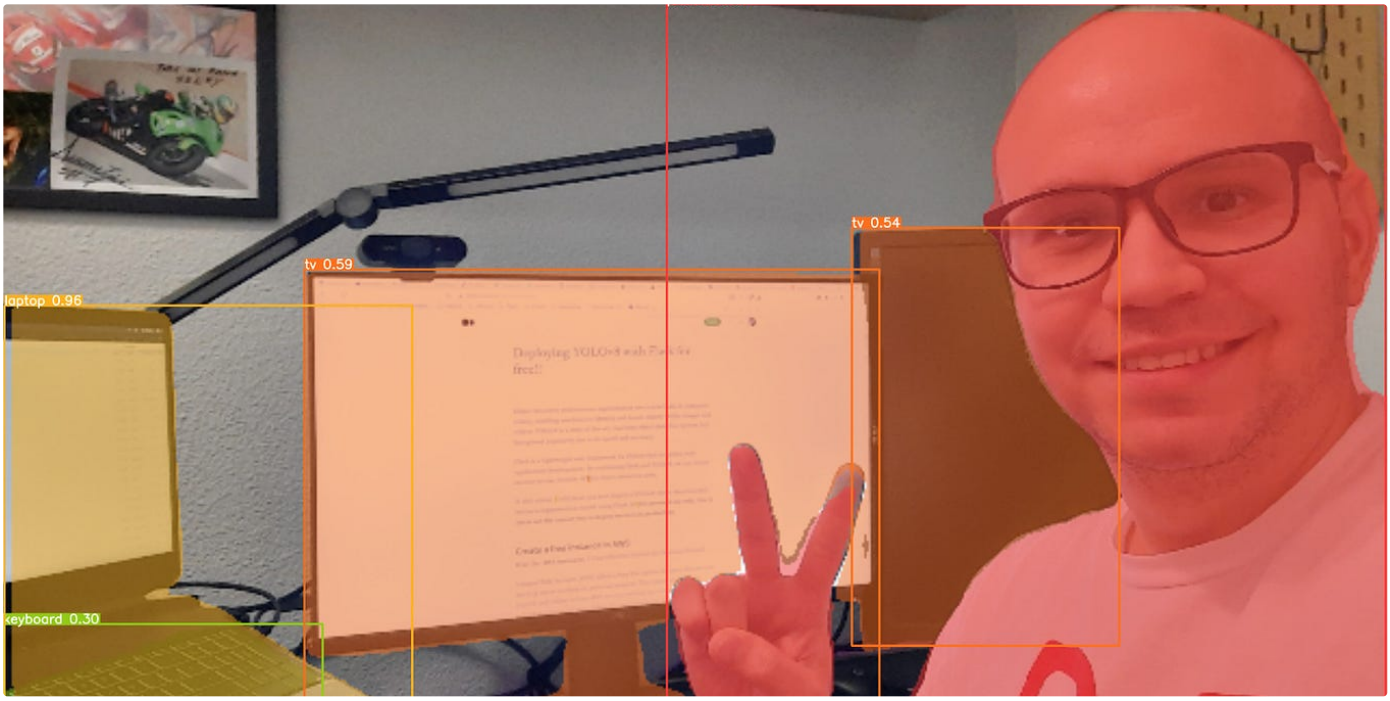
Şevval İLHAN

3D Object Detection with MediaPipe

3D Object Detection is a task of identify and locate objects based on their shape, location, and also orientation. In 2D Object detection...

4 min read · May 4

12 1



Henry Navarro

How to deploy YOLOv8 with FlaskAPI for free!

Simplified Object Detection and Instance Segmentation: Step-by-Step Guide to Deploying YOLOv8 with FlaskAPI for Personal Use

5 min read · May 8

5 1

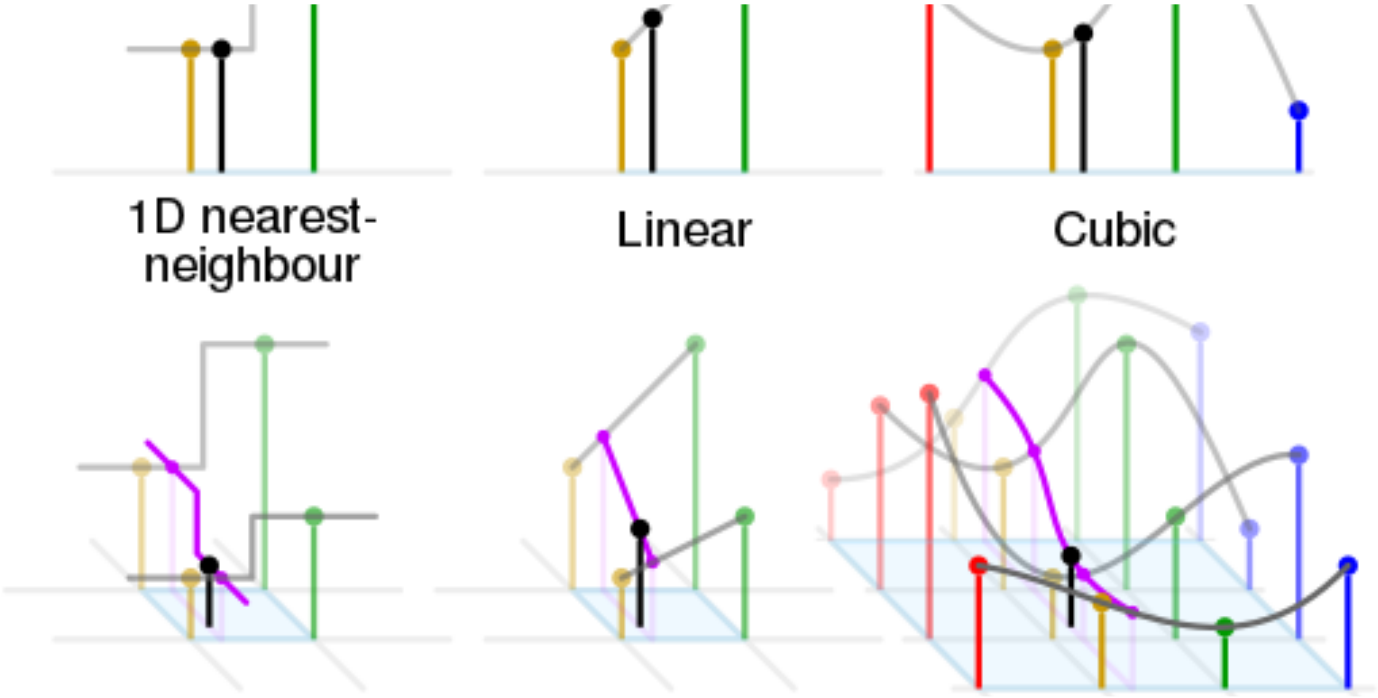
Lists

What is ChatGPT?

9 stories · 62 saves

Staff Picks

329 stories · 83 saves



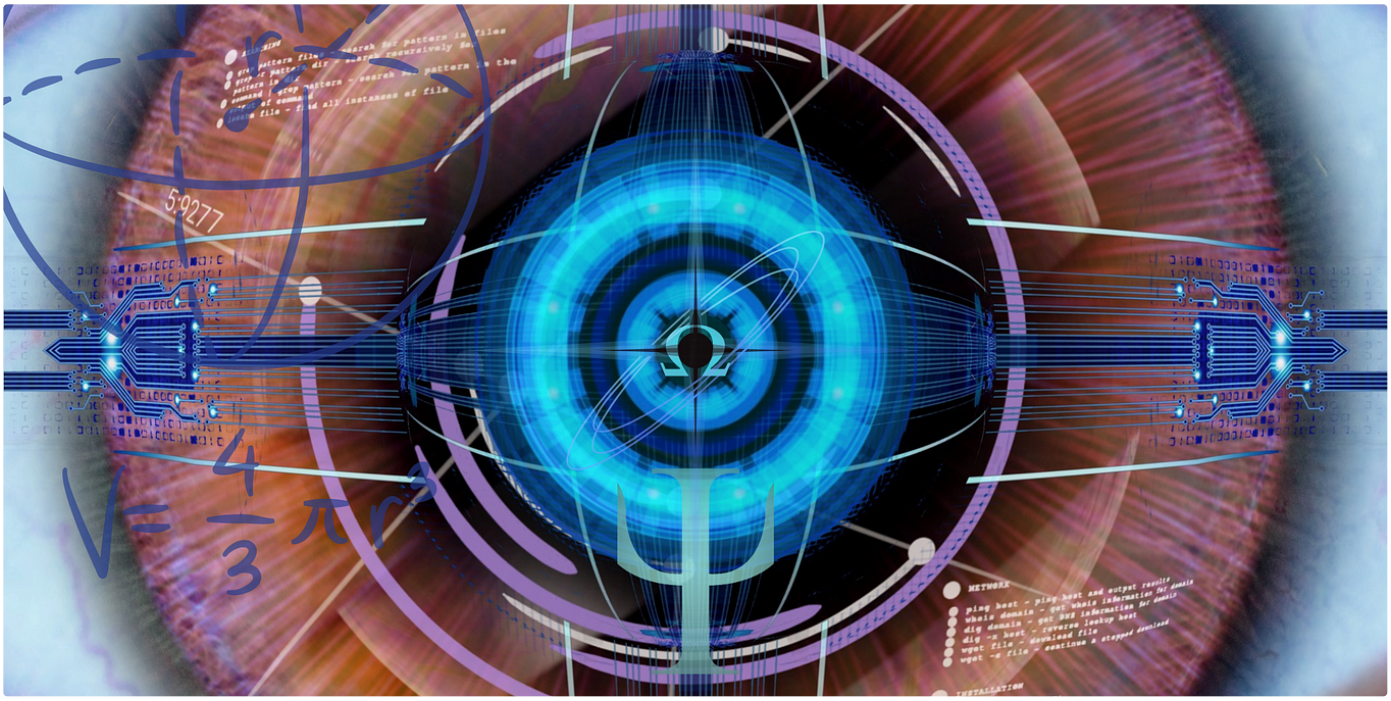
Zahid Parvez

Image interpolation in OpenCV

Some form of image interpolation is always taking place when manipulating digital images—whether it's resizing them to increase or...

· 3 min read · Jan 8

70 1



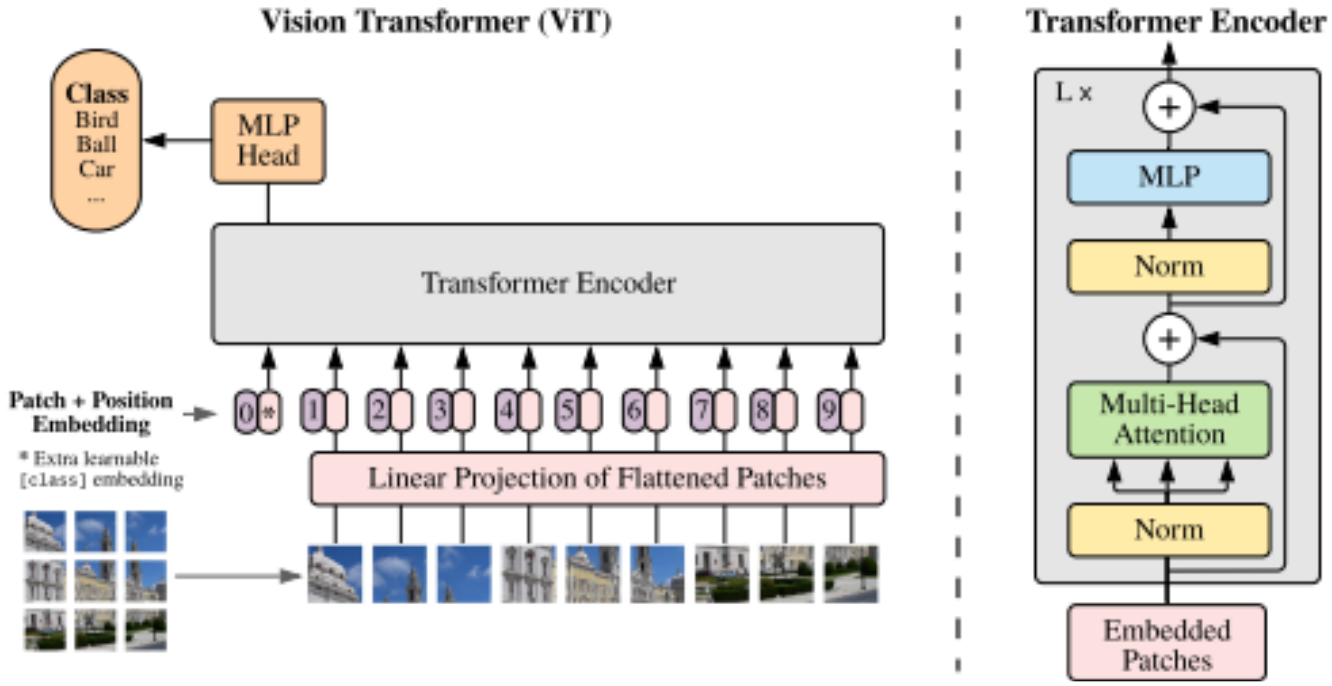
Nabeel Khan

Navigating Math for Computer Vision: Your Ultimate Roadmap

I got myself occupied with developing an understanding of Convolutional Neural Networks, as part of my final year project themed around...

5 min read · May 15

3



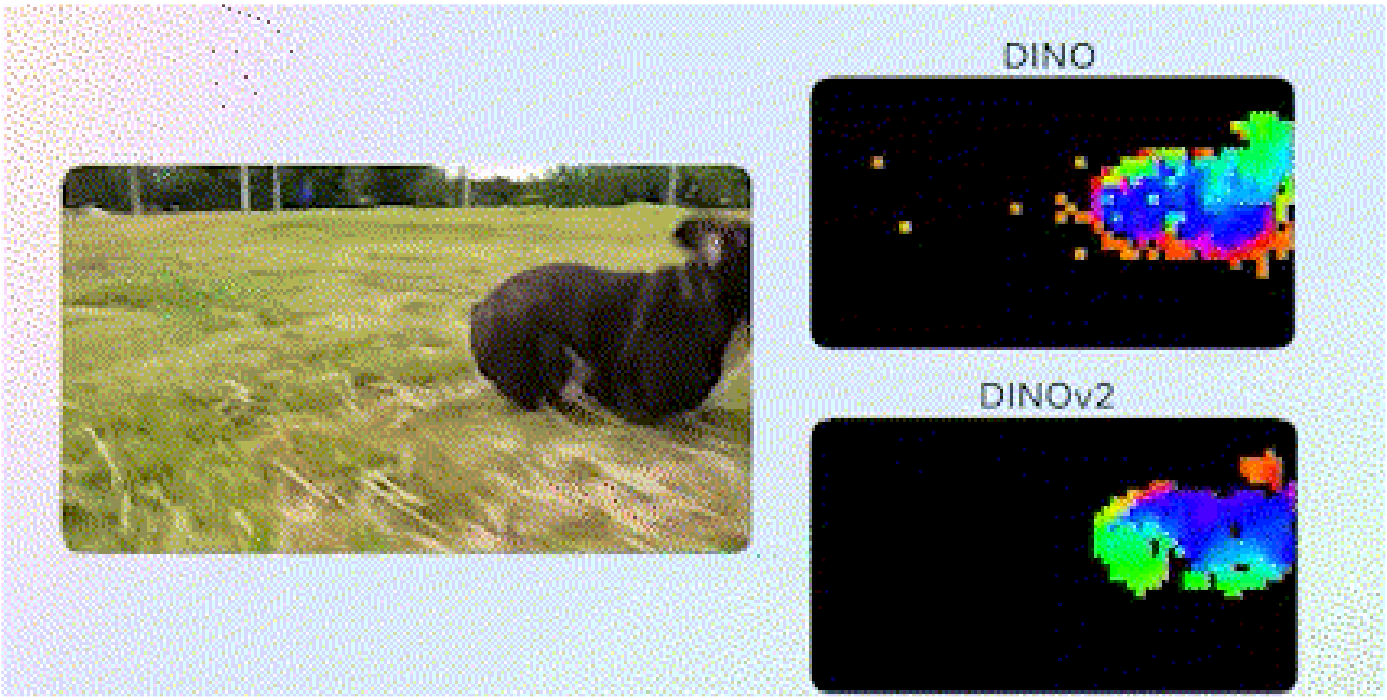
Vikram Pande

CNNs and Vision Transformers: Analysis and Comparison

Exploring the effectiveness of Vision Transformers and Convolutional Neural Networks (CNNs) in image classification tasks.

6 min read · Apr 29

153 4



Ashwanth Ravi in Better Programming

DINOv2: The New Frontier in Self-Supervised Learning?

Understanding the improvements in DINOv2

8 min read · May 11

 117  2

See more recommendations