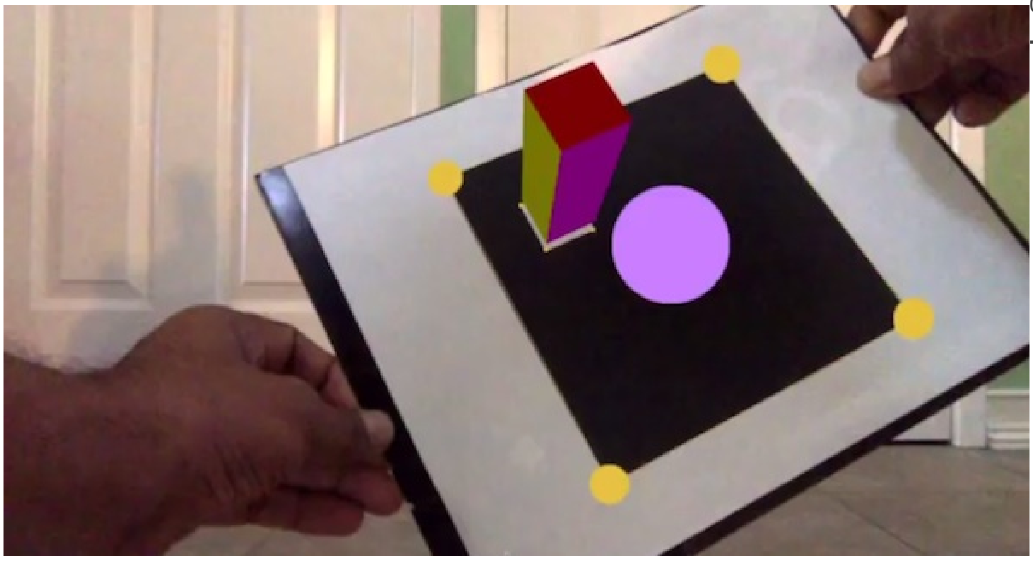
calculating OpenGL perspective matrix from OpenCV intrinsic matrix

How can we calculate the OpenGL perpsective matrix, from the camera calibration matrix (intrinsic matrix) and other parameters?



When we develop augmented reality applications, we have to display OpenGL graphics superimposed on the realtime video feed that you get from a camera. To do this we must do two steps:

- We must first calibrate our camera as an offline process to determine the intrinsic parameters of the camera as described by Hartley and Zisserman
- The augmented reality application, on every frame of the realtime video feedback, now uses the intrinsic matrix, and correspondence between the image and object-centric points of a fiducial marker and give you the rotation and translation (model-view matrix) of the OpenGL frame. For drawing an open OpenGL object, we need the current model-view matrix and the perspective matrix. We just mentioned how we can get the model-view matrix. The persective matrix conventionally stays the same for the duration of the application. **How do we calculate the perspective matrix, given the intrinsic matrix we got from our camera calibration?**

Please checkout my implementation as demonstrated in this video.

Before we must give any explanation, we must acknowledge the following excellent blog post by Kyle Simek which is a more general solution on this subject. What follows is my personal way to explain off the complexity, deriving largely from Kyle's work, assuming the specialization that the OpenGL view frsutum constructed is symmetrical wrt the co-ordinate axes.

Problem specification

What do we have?

- We have the α, β, c_x, c_y values from the intrinsic matrix.
- We assume a near and far plane distances n and f of the view frustum.
- We also assume that the image plane is symmetric wrt the focal plane of the pinhole camera.
- From OpenGL literature(See Song Ho Ahn), we have the formula for the OpenGL projection matrix as, $M_{proj} = \begin{bmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$.

What do we need to find?

- The only unknowns in M_{proj} are r and t. We need to find them from the above givens, so that we can completely calculate, M_{proj}.

Published

08 March 2014

Tags

- augmented reality 1
- opencv 1
- opengl 1
- intrinsic matrix 1
- camera calibration 1

Haretley-Zisserman pinhole camera

We have the OpenCV intrinsic matrix to start with. It is expressed as, $I = \begin{bmatrix} \alpha & \mu & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix}$

Note that, for all our practical purposes μ , the skew factor is zero. So the above intrinsic matrix simplifies to: $I = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix}$

This is derived from the the basic pinhole camera description as shown in fig1 an fig2.

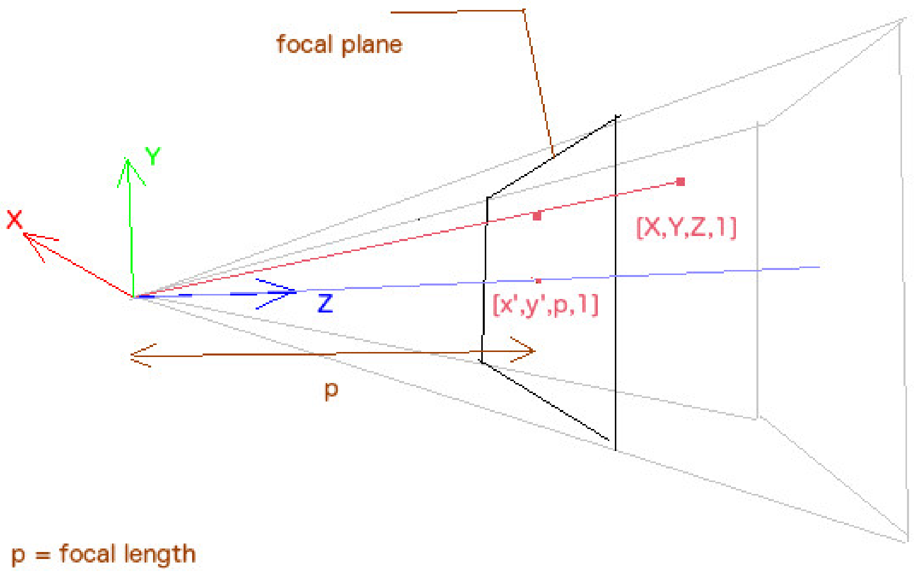


fig 1: pinhole camera as described in Hartley-Ziisserman

. Note that we use the symbol p for focal length, since the we need to use the regular symbol f for the far plane distance later. The transformation that transforms the point $[X, Y, Z, 1]$ in 3d space to the point $[x', y', p, 1]$ is a matrix multiplication followed by perspective division (division of result by the w component of the result).

$$\begin{bmatrix} x' \\ y' \\ p \\ 1 \end{bmatrix} \approx \begin{bmatrix} p & 0 & 0 & 0 \\ 0 & p & 0 & 0 \\ 0 & 0 & p & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

where '$\approx$' sign captures the perspective division.

Now, let us take a look at the transformation that relates the image (pixel) co-ordinates and the points in the focal plane.

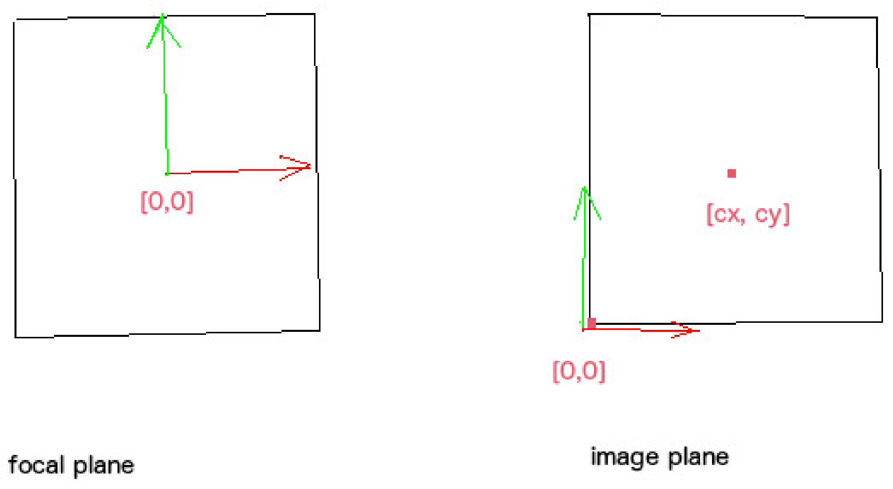


fig 2: correspondence between focal plane and image plane

The transformation from focal plane to the image plane is captured by the following matrix.

$$\begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} k_x & 0 & \frac{c_x}{p} & 0 \\ 0 & k_y & \frac{c_y}{p} & 0 \\ 0 & 0 & \frac{1}{p} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x' \\ y' \\ p \\ 1 \end{bmatrix} \qquad (2)$$

where

- k_x is the pixel-coordinate/camera-space scale factor along X axis.
- k_y is defined similarly along the Y axis.
- c_x and c_y are points in the image plane corresponding to the origin of the focal plane.
- the image plane origin is at left-bottom corner.
- According to our initial assumption, our image plane is symmetrically arranged wrt to the image plane.

Equations (1) and (2) are combined in the following equation.

$$\begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} \approx \begin{bmatrix} p.k_x & 0 & c_x & 0 \\ 0 & p.k_y & c_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \qquad (3)$$

The upper 3×3 portion of the above matrix is in fact our intrinsic matrix.

$$\begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} p.k_x & 0 & c_x \\ 0 & p.k_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

pinhole camera in OpenGL view frustum.

Let us put this pinhole camera in the OpenGL coordinate system, which will help us to visualize the integration of the two (HZ-pinhole camera vs OpenGL} formulations.

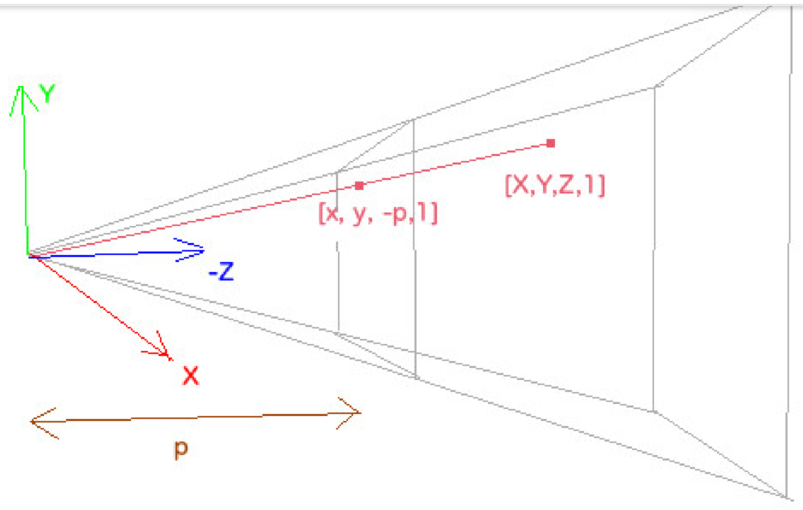


fig 3: pinhole camera in OpenGL co-ordinate system

This is different from fig 1 in that, the camera is looking down the negative Z-axis just as in OpenGL. Also, the image plane origin should correspond to the (minimum, minimum) point of the focal plane origin. With these considerations, the effective intrisic matrix in OpenGL co-ordinate system is

$$\begin{bmatrix} x \\ y \\ 1 \\ 1 \end{bmatrix} \approx \begin{bmatrix} p.k_x & 0 & -c_x & 0 \\ 0 & p.k_y & -c_y & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4}$$

The effective intrinsic matrix is unimportant to our eventual calculation.

OpenGL perspective matrix

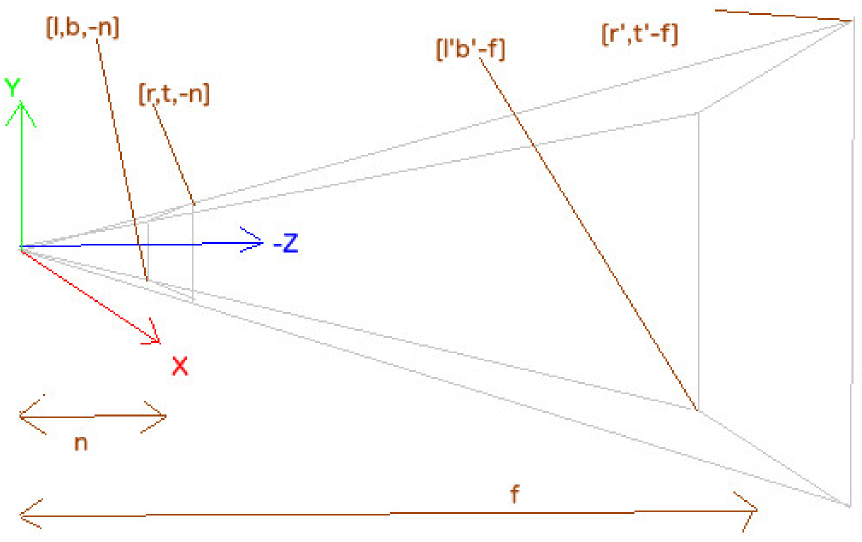


fig 4: basic OpenGL view frustum

Shown above is the OpenGL view frustum. From this we should be able to generate a pespective matrix. OpenGL will use the perspective matrix to transform a 3d point to the normalized device coordinate space below.

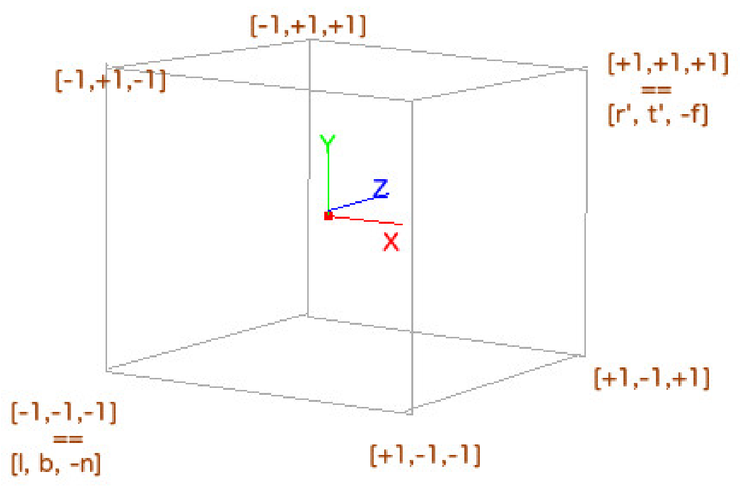


fig 5: normalized device coordinate space, after applying OpenGL perspective process

For mathematical sanity, please be assured that l', r', b' and t' can be computed from $l, r, b, , t, n$ and f by means of similarity transforms.

We can write the above OpenGL perspective process as

$$\begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} = M_{proj} \times \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \approx \begin{bmatrix} x_{clip} \\ y_{clip} \\ z_{clip} \\ w_{clip} \end{bmatrix} \tag{6}$$

All the components of the resultant vector $[x, y, z, 1]^T$ will be in the range -1.0 to 1.0, (ie: normalized device coordinates). The $\approx$ sign denote division by w component. The derivation of M_{proj} is ver well described by Song Ho Ahn and we wont be repeating that. So we can write M_{proj} as,

$$M_{proj} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{7}$$

Specialization of the OpenGL perspective matrix

Please note that, when trying to apply the perspective projection to our needs, the following constraints apply, $l < 0, r > 0, b < 0, t > 0$. Also, we have $n > 0, f > 0$, and $n > p > f$. Also, since our image plane is symmetrically arranged wrt the focal plane, we have $l = -r, b = -t$. So we have $r + l = 0$, $r - l = 2r$, $t + b = 0$ and $t - b = 2t$. So (7) will simplify to

$$M_{proj} = \begin{bmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \tag{8}$$

Result

We assume the values of the near and far planes. So let us say that n and t are known. (In our implementation we used $n = 0.1$ and $f = 100.0$.) How can we compute the values of r and t from

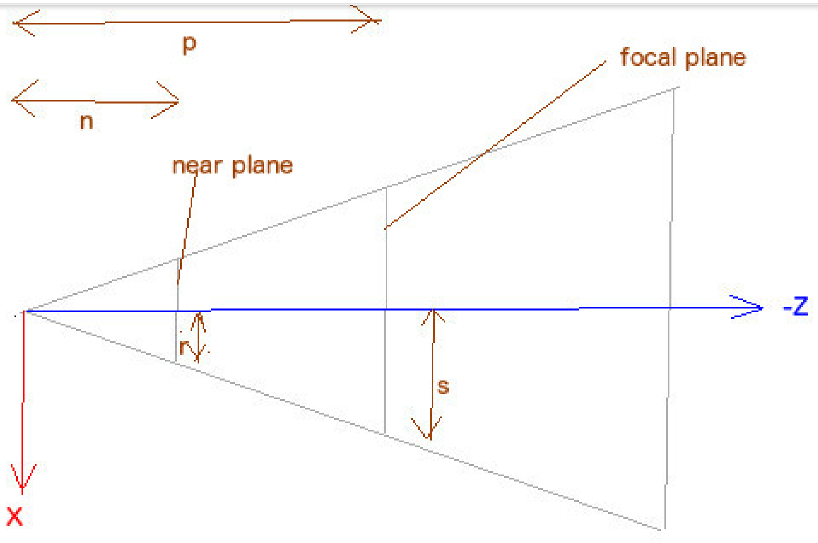


fig 6: correspondence of focal plane and near plane

Shown above is a top view of the frustum. The unknown s when transformed to the image plane has a value of c_x. So $s = \frac{c_x}{k_x}$

From the above figure, we have

$$r = \frac{s.n}{p}$$

Hence

$$r = \frac{n.c_x}{p.k_x}$$

Or

$$r = \frac{n.c_x}{\alpha}$$

Similarly,

$$t = \frac{n.c_y}{\beta}$$

So we can write our projection matrix as,

$$M_{proj} = \begin{bmatrix} \frac{\alpha}{c_x} & 0 & 0 & 0 \\ 0 & \frac{\beta}{c_y} & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

Please checkout the following video.

References

- Kyle Simek's excellent post on OpenGL-opencv camera calibration
- Song Ho Ahn's excellent derivation of OpenGL perspective matrix

← Previous Archive Next →

Recommend Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Sérgio Agostinho • 9 months ago

Hey Koshy. Thanks for the post. Just a quick note which I felt that is important to state on the list of early assumptions you wrote. There's an underlying assumption that cx and cy in the HZ model are actually half the width and height of your screen. This rarely happens in real cameras, there's always an offset of a couple of pixels. Given that the envisioned use case is AR, in which real cameras are used, it might be worth to write an alternative formulation for Mproj without that assumption.

Reply • Share ›

Stephan Vedder • 2 years ago

Hey, this only seems to work with Left-Handed Coordinate systems (it's always inverting the z coordinate because of M[2][2])

Reply • Share ›

Vitaly Oganesyan • 2 years ago

Hi, kgeorge! Thanks for the article! Can you please share source code of the project from the video?

Reply • Share ›

ALSO ON KGEORGE

HOG implementation and object detection

1 comment • 5 years ago

Andrew — Hey, just came across this project and it looks very interesting. I am trying to do some …

Understanding Gaobor Filter

1 comment • 3 years ago

wei wang — why the fourier transform of equation 105 I got by another method is different ,which …

calculating opengl perspective matrix from opencv intrinsic matrix

3 comments • 5 years ago

Ram — Hi, Can you please share the source code for verification? my mail ID is kgram007@gmail.com

Subscribe Add Disqus to your siteAdd DisqusAdd

© Koshy George 2012 with help from Jekyll Bootstrap and Twitter Bootstrap