Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix

August 13, 2013
Author's note: the source file for all of this post's diagrams is available under a creative commons license. Please feel free to modify and share!

Credit: Dave6163 (via Flickr)

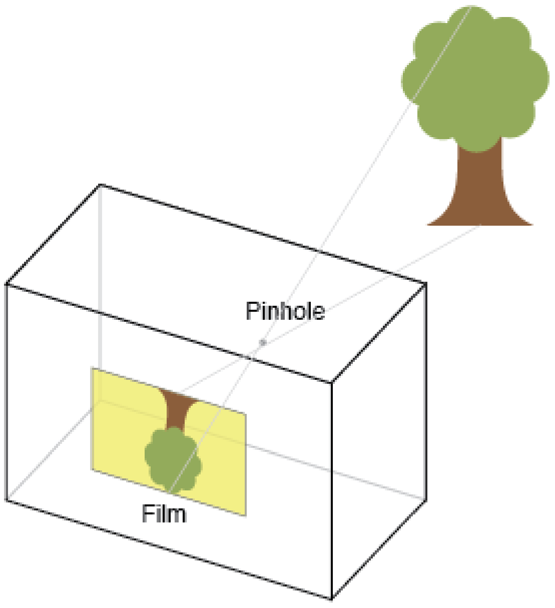
Today we'll study the intrinsic camera matrix in our third and final chapter in the trilogy "Dissecting the Camera Matrix." In the first article, we learned how to split the full camera matrix into the intrinsic and extrinsic matrices and how to properly handle ambiguities that arise in that process. The second article examined the extrinsic matrix in greater detail, looking into several different interpretations of its 3D rotations and translations. Today we'll give the same treatment to the intrinsic matrix, examining two equivalent interpretations: as a description of the virtual camera's geometry and as a sequence of simple 2D transformations. Afterward, you'll see an interactive demo illustrating both interpretations.

If you're not interested in delving into the theory and just want to use your intrinsic matrix with OpenGL, check out the articles Calibrated Cameras in OpenGL without glFrustum and Calibrated Cameras and gluPerspective.

All of these articles are part of the series "The Perspective Camera, an Interactive Tour." To read the other entries in the series, head over to the table of contents.

The Pinhole Camera

The intrinsic matrix transforms 3D camera cooordinates to 2D homogeneous image coordinates. This perspective projection is modeled by the ideal pinhole camera, illustrated below.



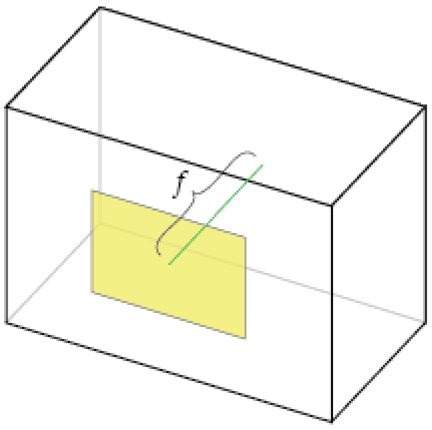
The intrinsic matrix is parameterized by Hartley and Zisserman as

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Each intrinsic parameter describes a geometric property of the camera. Let's examine each of these properties in detail.

Focal Length, f_x, f_y

The focal length is the distance between the pinhole and the film (a.k.a. image plane). For reasons we'll discuss later, the focal length is measured in pixels. In a true pinhole camera, both f_x and f_y have the same value, which is illustrated as f below.



In practice, f_x and f_y can differ for a number of reasons:

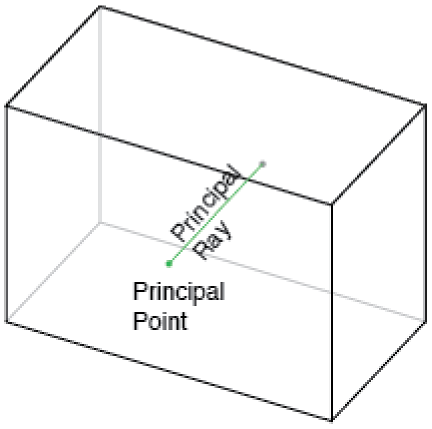
- Flaws in the digital camera sensor.
- The image has been non-uniformly scaled in post-processing.
- The camera's lens introduces unintentional distortion.
- The camera uses an anamorphic format, where the lens compresses a widescreen scene into a standard-sized sensor.
- Errors in camera calibration.

In all of these cases, the resulting image has non-square pixels.

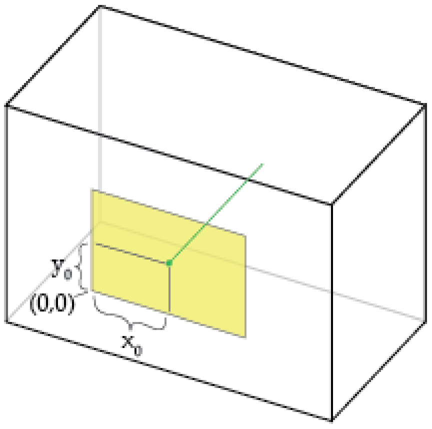
Having two different focal lengths isn't terribly intuitive, so some texts (e.g. Forsyth and Ponce) use a single focal length and an "aspect ratio" that describes the amount of deviation from a perfectly square pixel. Such a parameterization nicely separates the camera geometry (i.e. focal length) from distortion (aspect ratio).

Principal Point Offset, x_0, y_0

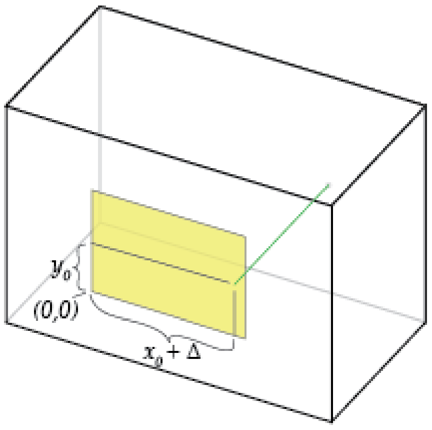
The camera's "principal axis" is the line perpendicular to the image plane that passes through the pinhole. Its itersection with the image plane is referred to as the "principal point," illustrated below.



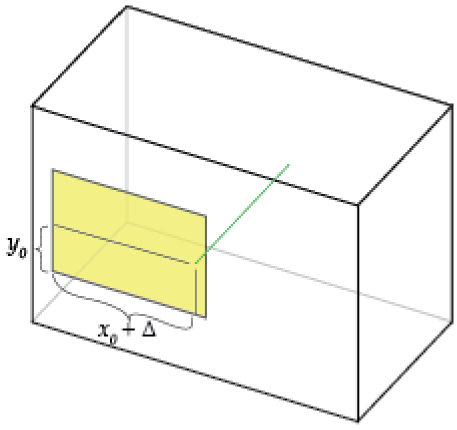
The "principal point offset" is the location of the principal point relative to the film's origin. The exact definition depends on which convention is used for the location of the origin; the illustration below assumes it's at the bottom-left of the film.



Increasing x_0 shifts the pinhole to the right:



This is equivalent to shifting the film to the left and leaving the pinhole unchanged.



Notice that the box surrounding the camera is irrelevant, only the pinhole's position relative to the film matters.

Axis Skew, s

Axis skew causes shear distortion in the projected image. As far as I know, there isn't any analogue to axis skew a true pinhole camera, but apparently some digitization processes can cause nonzero skew. We'll examine skew more later.

Other Geometric Properties

The focal length and principal point offset amount to simple translations of the film relative to the pinhole. There must be other ways to transform the camera, right? What about rotating or scaling the film?

Rotating the film around the pinhole is equivalent to rotating the camera itself, which is handled by the extrinsic matrix. Rotating the film around any other fixed point x is equivalent to rotating around the pinhole P, then translating by $(x - P)$.

What about scaling? It should be obvious that doubling all camera dimensions (film size and focal length) has no effect on the captured scene. If instead, you double the film size and *not* the focal length, it is equivalent to doubling both (a no-op) and then halving the focal length. Thus, representing the film's scale explicitly would be redundant; it is captured by the focal length.

Focal Length - From Pixels to World Units

This discussion of camera-scaling shows that there are an infinite number of pinhole cameras that produce the same image. The intrinsic matrix is only concerned with the relationship between camera coordinates and image coordinates, so the absolute camera dimensions are irrelevant. Using pixel units for focal length and principal point offset allows us to represent the relative dimensions of the camera, namely, the film's position relative to its size in pixels.

Another way to say this is that the intrinsic camera transformation is invariant to uniform scaling of the camera geometry. By representing dimensions in pixel units, we naturally capture this invariance.

You can use similar triangles to convert pixel units to world units (e.g. mm) if you know at least one camera dimension in world units. For example, if you know the camera's film (or digital sensor) has a width W in millimiters, and the image width in pixels is w, you can convert the focal length f_x to world units using:

$$F_x = f_x \frac{W}{w}$$

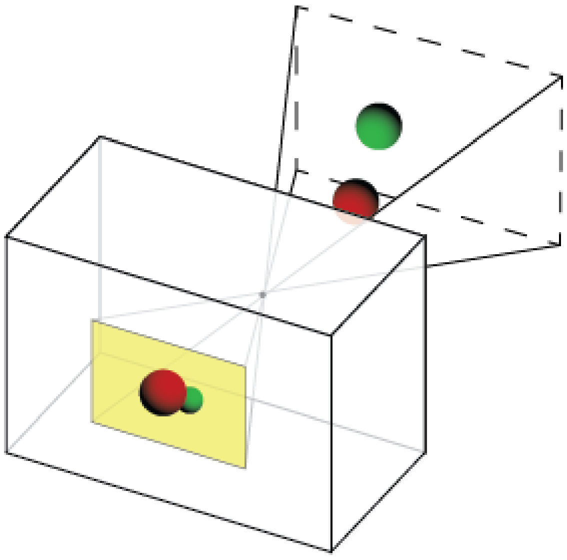
Other parameters f_y, x_0, and y_0 can be converted to their world-unit counterparts F_y, X_0, and Y_0 using similar equations:

$$F_y = f_y \frac{H}{h} \qquad X_0 = x_0 \frac{W}{w} \qquad Y_0 = y_0 \frac{H}{h}$$

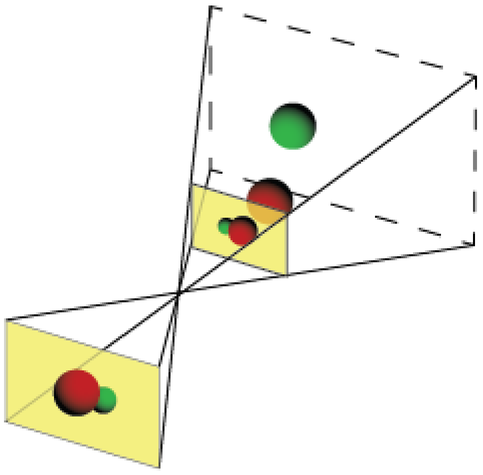
The Camera Frustum - A Pinhole Camera Made Simple

As we discussed earlier, only the arrangement of the pinhole and the film matter, so the physical box surrounding the camera is irrelevant. For this reason, many discussion of camera geometry use a simpler visual representation: the camera frustum.

The camera's viewable region is pyramid shaped, and is sometimes called the "visibility cone." Lets add some 3D spheres to our scene and show how they fall within the visibility cone and create an image.

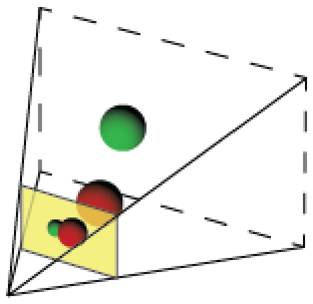


Since the camera's "box" is irrelevant, let's remove it. Also, note that the film's image depicts a mirrored version of reality. To fix this, we'll use a "virtual image" instead of the film itself. The virtual image has the same properties as the film image, but unlike the true image, the virtual image appears in front of the camera, and the projected image is unflipped.



Note that the position and size of the virtual image plane is arbitrary — we could have doubled its size as long as we also doubled its distance from the pinhole.

After removing the true image we're left with the "viewing frustum" representation of our pinhole camera.



The pinhole has been replaced by the tip of the visibility cone, and the film is now represented by the virtual image plane. We'll use this representation for our demo later.

Intrinsic parameters as 2D transformations

In the previous sections, we interpreted our incoming 3-vectors as 3D image coordinates, which are transformed to homogeneous 2D image coordinates. Alternatively, we can interpret these 3-vectors as 2D homogeneous coordinates which are transformed to a new set of 2D points. This gives us a new view of the intrinsic matrix: a sequence of 2D affine transformations.

We can decompose the intrinsic matrix into a sequence of shear, scaling, and translation transformations, corresponding to axis skew, focal length, and principal point offset, respectively:

$$\begin{aligned} K &= \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \\ &= \underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Scaling}} \times \underbrace{\begin{pmatrix} 1 & s/f_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Shear}} \end{aligned}$$

An equivalent decomposition places shear *after* scaling:

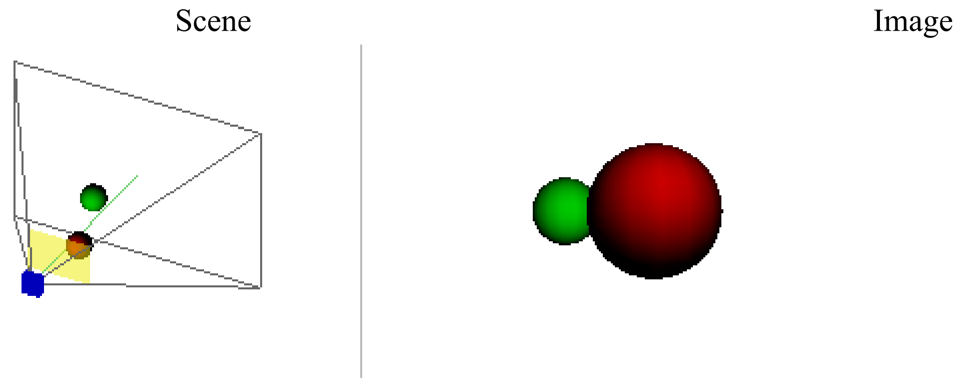
$$K = \underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{pmatrix} 1 & s/f_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Shear}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Scaling}}$$

This interpretation nicely separates the extrinsic and intrinsic parameters into the realms of 3D and 2D, respactively. It also emphasizes that the intrinsic camera transformation occurs *post-projection*. One notable result of this is that **intrinsic parameters cannot affect visibility** — occluded objects cannot be revealed by simple 2D transformations in image space.

Demo

The demo below illustrates both interpretations of the intrinsic matrix. On the left is the "camera-geometry" interpretation. Notice how the pinhole moves relative to the image plane as x_0 and y_0 are adjusted.

On the right is the "2D transformation" interpretation. Notice how changing focal length results causes the projected image to be scaled and changing principal point results in pure translation.



Left: scene with camera and viewing volume. Virtual image plane is shown in yellow. *Right*: camera's image.

- Intrinsic

Focal Length
Axis Skew
x_0
y_0

Dissecting the Camera Matrix, A Summary

Over the course of this series of articles we've seen how to decompose

1. the full camera matrix into intrinsic and extrinsic matrices,
2. the extrinsic matrix into 3D rotation followed by translation, and
3. the intrinsic matrix into three basic 2D transformations.

We summarize this full decomposition below.

$$P = \overbrace{K}^{\text{Intrinsic Matrix}} \times \overbrace{[R \mid \mathbf{t}]}^{\text{Extrinsic Matrix}}$$

$$= \overbrace{\underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Translation}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Scaling}} \times \underbrace{\begin{pmatrix} 1 & s/f_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{\text{2D Shear}}}^{\text{Intrinsic Matrix}} \times \overbrace{\underbrace{\left(\begin{array}{c|c} I & \mathbf{t} \end{array} \right)}_{\text{3D Translation}} \times \underbrace{\left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array} \right)}_{\text{3D Rotation}}}^{\text{Extrinsic Matrix}}$$

To see all of these transformations in action, head over to my Perpective Camera Toy page for an interactive demo of the full perspective camera.

Do you have other ways of interpreting the intrinsic camera matrix? Leave a comment or drop me a line!

Next time, we'll show how to prepare your calibrated camera to generate stereo image pairs. See you then!

Posted by *Kyle Simek*
Compiling ELSD (Ellipse and Line Segment Detector) on OS X → ← Calibrated Cameras and gluPerspective

0 points Tweet

25 Comments Signtations Login

Recommend 13 Tweet Share Sort by Best

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Ben Hur Nascimento • 3 years ago

Thank you very much for this post, it helped me a lot on a computional vision project.

1 • Reply • Share ›

Milton Wong • 4 years ago

Hi, ksimek,

Thank you for answering previous question.
Now I came across one more question: If the operation is not cropping, but a scaling. e.g. from 1280x480 to 640x240 (aspect ratio remained) using a image size convertor. Except cx' and cy' should change to 320 and 120 respectively. Should parameters such as fx and fy be changed?

Furthermore, if aspect ratio isn't remained, such as from 1280x480 to 1024x768, and how about for this situation?

Thanks in advance.

Milton

1 • Reply • Share ›

ksimek Mod → Milton Wong • 4 years ago

Yes, in this case, both will change. The easiest way to think about image scaling is as an extra linear transformation applied after camera projection, which you can achieve by left-multiplying the camera by diag([s,s,1]). The new camera will have all the parameters changed, including the axis skew if it is non-zero. The non-uniform scaling is the same, but the scaling constants in the x and y direction will differ, i.e. diag([sx, sy, 1]).

• Reply • Share ›

Milton Wong → ksimek • 4 years ago

@keimek, thank you so much:)
Let me confirm my understanding.

Case1: uniform scaling (aspect-ratio remained)
orginal intrinsic matrix K = [fx,0,cx;0,fy,cy;0,0,1] = [520,0,640;0,520,240;0,0,1], original resolution is 1280x480, now convert it into 640x240 resolution, so scaling factor s = 0.5.
Then apply diag([s,s,1])*K = diag([0.5,0.5,1])*[520,0,640;0,520,240;0,0,1] = [260,0,320;0,260,120;0,0,1], which is the updated intrinsic matrix K'

Case2: non-uniform scaling

original resolution is 1280x480, now convert it into 1024x768 resolution, so scaling factor sx = 1024/1280 = 0.8 , sy = 768/480 = 1.6.
Then apply diag([sx,sy,1])*K = diag([0.8,1.6,1])*[520,0,640;0,520,240;0,0,1] = [416,0,512;0,832,384;0,0,1], which is the updated intrinsic matrix K'

Right?

Also , one can observe that fx and cx are scaled by sx, fy and cy are scaled by sy.

see more

Reply • Share ›

ksimek Mod → Milton Wong • 4 years ago

Exactly, the elements of the diagonal matrix scale the rows of K.

I agree, the normalized intrinsic matrix is a nice way of removing the question of resolution entirely. Resolution has nothing to do with the geometry of a pinhole camera, so one might argue it shouldn't be included in K. It also avoids the need to store the image dimensions along with the camera parameters.

Of course, using K_normal, you'll need to scale your points appropriately before displaying them. It may be worthwhile to save the original image aspect ratio, especially if your camera has significantly different unnormalized fx and fy, so you can remember how to display the image without distortion.

see more

Reply • Share ›

get2jils • a month ago

Can someone explain me this statement?. I couldn't understand.

"Rotating the film around the pinhole is equivalent to rotating the camera itself, which is handled by the extrinsic matrix. Rotating the film around any other fixed point x is equivalent to rotating around the pinhole P, then translating by (x−P)."

Reply • Share ›

samrat chakravarthi • a year ago

How to Find out the instrinsic parameter of the camera using any two images?

Reply • Share ›

Arvind Keerthi • 2 years ago

This demo is just fabulous!

Reply • Share ›

Zhixin Shu • 2 years ago

Thanks!

• Reply • Share ›

pk • 2 years ago

Hi,thank you for your work. I have a question, How about the intrinsic matrix of hyperspectral camera?

• Reply • Share ›

Shangxuan Wu • 2 years ago

Thanks!

• Reply • Share ›

Clemens Tolboom • 3 years ago

Thanks for this writing. I wondered where to place the screen / pixels. Now I know. It doesn't matter unless placed into real world space right?

I fixed a typo through https://github.com/ksimek/k...

• Reply • Share ›

Lei Wen • 3 years ago

I have some confusion here that if all I have is the MVP matrix inside one opengl vertex shader, which convert the model point into the clipping space.

Could we use the same tech introduced here, that using the QR decomposition method to extract the M/V/P separately from the pre-calculated MVP matrix?

• Reply • Share ›

pay for a paper • 3 years ago

It's a pretty wonderful thing that you were able to discuss this kind of information that will be going to help a lot of people in understanding more about intrinsic. From this, they could totally improve their technique and some of their projects in becoming better.

• Reply • Share ›

Francesco • 3 years ago

Hi ksimek,

congratulation, good explanation about physical camera characteristic!!

I have a question for you about that, is possible to calculate the metric distance of one pixel (located in a generic position from middle to base of image) that represent an object knowing intrinsic and extrinsic parameters? how can I do this?

thank you!

• Reply • Share ›

Amer Qarabsa → Francesco • 3 years ago

Hi Franesco,

I have a smiliar requirment , were you lucky to find the answer?

I have a similar requirement , were you lucky to find the answer?

Reply • Share ›

ksimek Mod → Amer Qarabsa • 3 years ago

Getting metric distance is not possible without additional information. You need to know the measurement of at least one distance, then you can rescale all of your coordinates to agree with that. That usually means manually measuring distances, using depth sensors like laser range finder or structured light, or using prior knowledge like the heights of people or doorways.

see more

Reply • Share ›

puneet • 3 years ago

well explained

Reply • Share ›

Andy • 4 years ago

Fantastic series of posts! Thanks for taking the time.

Reply • Share ›

Milton Wong • 4 years ago

Hi, ksimek,

I had a question about intrinsic matrix.

Q: I had calibrated my input image and get intrinsic matrix K = [fx,0,cx;0,fy,cy;0,0,1] = [520,0,640;0,520,240;0,0,1], the resolution of image is 1280x480.

Now if I center crop the original image into a 640x480 image. What'll be the updated intrinsic matrix K'?

Will the value of fx, fy change? Obviously, the updated cx' and cy' should be 320 and 240 respectively.

Could you give me some hints?

Thanks in advance.

Milton

see more

^ | ∨ • Reply • Share ›

ksimek Mod ➦ Milton Wong • 4 years ago

Hi Milton. The intrinsic matrix doesn't encode anything about the clipping region, so in theory cropping shouldn't affect the intrinsic parameters. However, usually when cropping, we also update the image origin to the corner of the new image, which requires the change in the principal point offset that you suggested. The other parameters shouldn't change.

^ | ∨ • Reply • Share ›

Milton Wong ➦ ksimek • 4 years ago

@ksimek , thank you very much. I'll test on that.

^ | ∨ • Reply • Share ›

HPH • 4 years ago

This is wonderful. Can you please give a more detailed explanation on the axis skew? Like what exactly is s? Thank you very much!

^ | ∨ • Reply • Share ›

ksimek Mod ➦ HPH • 4 years ago

Regarding what axis skew *is*, I can't say much more than I already have. The interactive demo above shows how it affects the viewing frustum and the projected image by it introducing a shear transformation. Practically speaking, it's just an extra free parameter in the linear camera model that has no obvious analogue in the physical camera model.

When you do camera calibration, allowing non-zero skew could give better fit to the data, but more likely than not, the extra free parameter is just fitting to noise in the observations or flaws in the calibration target. In that case, the resulting camera will perform worse than a camera with zero skew in modeling points located away from the training set. This is probably why OpenCV's calibration routines force the skew to be zero.

All 3D reconstruction tools that I'm aware of also assume zero skew, including the Bundler structure from motion library, the Point Cloud Library, and Ceres Solver's structure-from-motion demo.

In fact, in most structure-from-motion and self-calibration applications, the axis skew *must* be constrained to be zero in order to avoid projective ambiguity. More generally, at

see more

^ | ∨ • Reply • Share ›

ram • 5 years ago

Thanks a lot........... great post!

^ | ∨ • Reply • Share ›