<u>Sightations</u> ← <u>A Computer Vision Blog</u>

- Home
- About
- Contact
- <u>Code</u>
- Archive
- 🔊

Calibrated Cameras and gluPerspective

June 18, 2013

After posting my last article <u>relating glFrustum to the intrinsic camera matrix</u>, I receieved some emails asking how the (now deprecated) <u>gluPerspective</u> function relates to the intrinsic matrix. We can show a similar result with gluPerspective as we did with glFrustum, namely that it is the product of a glOrtho matrix and a (modified) intrinsic camera matrix, but in this case the intrinsic matrix has different constraints. I'll be re-using notation and concepts from the previous article, so if you aren't familiar with them, I recommend reading it first.

Decomposing gluPerspective

The matrix generated by gluPerspective is

$$\left(egin{array}{cccc} rac{f}{ ext{aspect}} & 0 & 0 & 0 \ 0 & f & 0 & 0 \ 0 & 0 & C' & D' \ 0 & 0 & -1 & 0 \end{array}
ight)$$

where

$$f=\cot(fovy/2) \ C'=-rac{far+near}{far-near} \ D'=-rac{2\ far\ near}{far-near}$$

Like with glFrustum, gluPerspective permits no axis skew, but it also restricts the viewing volume to be centered around the camera's principal (viewing) axis. This means that the principal point offsets x_0 and y_0 must be zero, and the matrix generated by glOrtho must be centered, i.e. bottom = -top and left = -right. The Persp matrix corresponding to the intrinsic matrix is:

$$Persp = egin{pmatrix} lpha & 0 & 0 & 0 \ 0 & eta & 0 & 0 \ 0 & 0 & A & B \ 0 & 0 & -1 & 0 \end{pmatrix}$$

where

$$A = near + far$$

 $B = near * far$

and the NDC matrix is

$$NDC = egin{pmatrix} rac{2}{right-left} & 0 & 0 & t_x \ 0 & rac{2}{top-bottom} & 0 & t_y \ 0 & 0 & -rac{2}{far-near} & t_z \ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$=egin{pmatrix} rac{2}{width} & 0 & 0 & 0 \ 0 & rac{2}{height} & 0 & 0 \ 0 & 0 & -rac{2}{far-near} & t_z \ 0 & 0 & 0 & 1 \end{pmatrix}$$

where

$$t_x = -rac{right + left}{right - left} \ t_y = -rac{top + bottom}{top - bottom} \ t_z = -rac{far + near}{far - near}$$

It is easy to show that the product $(NDC \times Persp)$ is equivalent to the matrix generated by gluPerspective(fovy, aspect, near, far) with

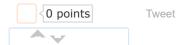
$$ext{fovy} = 2 ext{arctan} \left(rac{ ext{height}}{2 eta}
ight) \ ext{aspect} = rac{eta}{lpha} rac{ ext{width}}{ ext{height}}.$$

glFrustum vs. gluPerpsective

In my experience, the zero-skew assumption is usually reasonable, so glfrustum can provide a decent approximation to the full intrinsic matrix. However there is quite often a non-negligible principal point offset ($\sim 2\%$ of the image size), even in high-quality cameras. For this reason, gluperspective might be a good choice for quick-and-dirty demos, but for the most accurate simulation, you should use the full camera matrix like I described previously.

Posted by <u>Kyle Simek</u>

<u>Dissecting the Camera Matrix</u>, <u>Part 3: The Intrinsic Matrix</u> → ← <u>Calibrated Cameras in OpenGL without glFrustum</u>





Be the first to comment.

ALSO ON SIGHTATIONS

Q & A: Recovering pose of a calibrated camera - Algebraic vs. Geometric method?

1 comment • 4 years ago

dissertation writing service — Learning this 3D Avatarpoints projections is ideal in math subject. It gives the easy way to determine the ...

Dissecting the Camera Matrix, Part 2: The Extrinsic Matrix

16 comments • 6 years ago

Rajesh — Hi, I am looking for a method to
Avatarcompute homography matrix between
images(left & right) of stereo camera using ...

Compiling ELSD (Ellipse and Line Segment Detector) on OS X

4 comments • 5 years ago

philip andrew — Do you know anything better Avatanhan elsd now?

Calibrated Cameras in OpenGL without glFrustum

21 comments • 6 years ago

ksimek — Thanks for the clarifying photos.

AvatarSorry, I misunderstood the scenario -- I thought your calibration image was larger ...

Content by <u>Kyle Simek</u>. Original design by <u>Mark Reid</u> (<u>Some rights reserved</u>) Powered by <u>Jekyll</u>