James' Blog

Software development, graphics, scientific computing and building stuff.

Wednesday, November 30, 2011

Matching calibrated cameras with OpenGL

When working with calibrated cameras it is often useful to be able to display things on screen for debugging purposes. However the camera model used by OpenGL is quite different from the calibration parameters from, for example, OpenCV. The linear parameters that OpenCV provides are the following:

$$\begin{bmatrix} \alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

where (from http://en.wikipedia.org/wiki/Camera_resectioning) γ is the skew between the x and y axes, $[u_0, v_0]^T$ are the image principle point m_x, m_y with *f* being the focal length and m_x, m_y being scale factors relating pixels to distance. Multiplying a point $[x, y, z]^T$ by this matrix and dividing by resulting z-coordinate then gives the point projected into the image.

The OpenGL parameters are quite different. Generally the projection is set using the glFrustum command, which takes the left, right, top, bottom, near and far clip plane locations as parameters and maps these into "normalized device coordinates" which range from [-1, 1]. The normalized device coordinates are then transformed by the current viewport, which maps them onto the final image plane. Because of the differences, obtaining an OpenGL projection matrix which matches a given set of intrinsic parameters is somewhat complicated.

Roughly following this post, (***update: a much-improved update from Kyle, the post's author is available here***) the following code will produce an OpenGL projection matrix and viewport. I have tested this code against the OS-X OpenGL implementation (using gluProject) to verify that for randomly generated intrinsic parameters, the corresponding OpenGL frustum and viewport reproduce the x and y coordinates of the projected point. The code works by multiplying a perspective projection matrix by an orthographic projection to map into normalized device coordinates, and setting the appropriate box for the glViewport command.

```
/**
 @brief basic function to produce an OpenGL projection matrix and associ
 which match a given set of camera intrinsics. This is currently writter
 algebra library, however it should be straightforward to port to any 4x
 @param[out] frustum Eigen::Matrix4d projection matrix.  Eigen stores th
 @param[out] viewport 4-component OpenGL viewport values, as might be re
 @param[in]  alpha x-axis focal length, from camera intrinsic matrix
 @param[in]  alpha y-axis focal length, from camera intrinsic matrix
 @param[in]  skew  x and y axis skew, from camera intrinsic matrix
 @param[in]  u0 image origin x-coordinate, from camera intrinsic matrix
 @param[in]  v0 image origin y-coordinate, from camera intrinsic matrix
 @param[in]  img_width image width, in pixels
 @param[in]  img_height image height, in pixels
 @param[in]  near_clip near clipping plane z-location, can be set arbitr
 @param[in]  far_clip  far clipping plane z-location, can be set arbitra
*/
void build_opengl_projection_for_intrinsics( Eigen::Matrix4d &frustum, i

    // These parameters define the final viewport that is rendered into
    // the camera.
    double L = 0;
    double R = img_width;
    double B = 0;
    double T = img_height;

    // near and far clipping planes, these only matter for the mapping f
    // world-space z-coordinate into the depth coordinate for OpenGL
    double N = near_clip;
    double F = far_clip;

    // set the viewport parameters
    viewport[0] = L;
    viewport[1] = B;
    viewport[2] = R-L;
    viewport[3] = T-B;

    // construct an orthographic matrix which maps from projected
    // coordinates to normalized device coordinates in the range
    // [-1, 1].  OpenGL then maps coordinates in NDC to the current
    // viewport
    Eigen::Matrix4d ortho = Eigen::Matrix4d::Zero();
    ortho(0,0) =  2.0/(R-L); ortho(0,3) = -(R+L)/(R-L);
```

Labels

3D printing (8) Arduino (10) AVR (5) AVR Dragon (2) camera calibration (2) casting (8) CNC (20) code (40) computer vision (12) electrical (10) embedded (14) fluids (3) graphics (21) linear motion (17) making (40) math (1) matlab (5) mechanical (21) OpenGL (2) python (3) random (16) scientific computing (29) Shapelock (1) software (43) suppliers (1) Teensy (3)

```
    // additional row is inserted to map the z-coordinate to
    // OpenGL.
    Eigen::Matrix4d tproj = Eigen::Matrix4d::Zero();
    tproj(0,0) = alpha; tproj(0,1) = skew; tproj(0,2) = u0;
                        tproj(1,1) = beta; tproj(1,2) = v0;
                                           tproj(2,2) = -(N+F); tproj(2,
                                           tproj(3,2) = 1.0;

    // resulting OpenGL frustum is the product of the orthographic
    // mapping to normalized device coordinates and the augmented
    // camera intrinsic matrix
    frustum = ortho*tproj;
}
```

The code uses the Eigen linear algebra library, which conveniently stored matrices in column-major order, so applying the resulting *frustum* matrix is as simple as:

```
glMatrixMode(GL_PROJECTION);
glLoadMatrixd( &frustum(0,0) );
```

Posted by James Gregson at 4:09 PM
Labels: camera calibration, code, computer vision, graphics, software

10 comments:

Maikon said...

Hi,

How about modelview matrix? What do you think of this matrix?
|R t|
|0 1|

where R is 3x3 rotation matrix and t is a translation vector (t= -RC)

August 17, 2012 at 5:50 AM

James Gregson said...

Hello,

The focus of this post was really on just the projection matrix, but matching the modelview matrix is fairly straightforward if your cameras are calibrated.

For example, OpenCV (www.opencv.org) has functions that will estimate the modelview parameters for the camera (camera extrinsic parameters) from checkerboard patterns in the images. These extrinsics are just the rotation and translation matrices you're referring to.

Hope this is helpful,

James

August 17, 2012 at 6:46 AM

Maikon said...

James,
thank you for answer. My cameras are calibrated. I have a R rotation matrix (3x3) and a t translation vector. My doubt is how can i convert this matrices to modedelview 4x4 for use in OpenGL. Do you know how?

Another question. Why the both tproj(2,2) and tproj(2,3) are negative?

Maikon.

August 17, 2012 at 11:47 AM

James Gregson said...

Sorry for the delay,

To generate the modelview matrix you simply store your 3x3 rotation matrix R in the top left submatrix and the translation t in the first three rows of the right column, and set the bottom row to [0,0,0,1]^T.

Note that OpenGL stores the data by column first, then by row, so a positions [row,col] in the matrix will be the index row+col*4.

I don't recall offhand why the negatives are there except to make the frustum point the right way, i.e. so the camera looks forward rather than backwards.

James

August 22, 2012 at 8:54 AM

Kyle said...

Hi, James. I'm glad my blog post was helpful. If you're interested, I've written an updated version of that post that better explains how to set the parameters to glOrtho and discusses why the third column is negated. It is available here:

http://ksimek.github.io/2013/06/03/calibrated_cameras_in_opengl/

June 28, 2013 at 12:25 PM

James Gregson said...

Thanks! I've added a link to your new post. Great work by the way, the original post was the only useful reference that I was able to find and your new post improves on it considerably!

October 11, 2013 at 10:51 AM

skylook said...

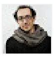
This comment has been removed by the author.

June 21, 2017 at 8:08 PM

skylook said...

Seems that there is a difference between your implementation and Kyle's:
In his article top = 0.0 and bottom = height, but yours top = height and bottom = 0.0.
Does that matter?

June 21, 2017 at 8:09 PM



b4silio said...

@skylook swapping the bottom and top is used to invert the Y axis depending on how you're working on your original space. (Basically: if things are upside down, then you probably need to swap the two of them).

January 11, 2018 at 4:59 AM

Post a Comment

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

Blog Archive

- ► 2015 (1)
- ► 2014 (1)
- ► 2013 (28)
- ► 2012 (46)
- ▼ 2011 (7)
  - ▼ November (4)
    - Matching calibrated cameras with OpenGL
    - Got my face did!
    - Getting things done with qmake
    - 4x4 transformation matching OpenGL
  - ► July (1)
  - ► April (1)
  - ► March (1)
- ► 2010 (4)

About Me

James Gregson

View my complete profile