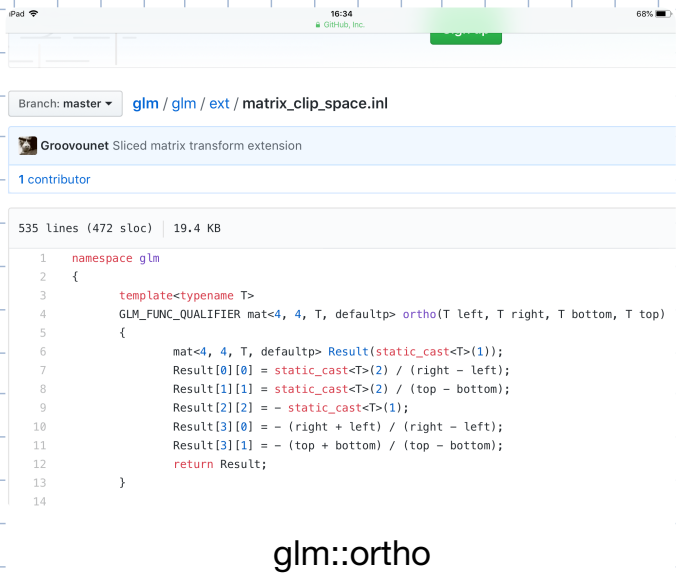
ORTHOGRAPHIC PROJECTION MATRIX

HJALMAR BASILE
SEP 2018
PISA



Branch: master ▾ glm / glm / ext / matrix_clip_space.inl

Groovounet Sliced matrix transform extension

1 contributor

535 lines (472 sloc) | 19.4 KB

```
1   namespace glm
2   {
3           template<typename T>
4           GLM_FUNC_QUALIFIER mat<4, 4, T, defaultp> ortho(T left, T right, T bottom, T top)
5           {
6                   mat<4, 4, T, defaultp> Result(static_cast<T>(1));
7                   Result[0][0] = static_cast<T>(2) / (right - left);
8                   Result[1][1] = static_cast<T>(2) / (top - bottom);
9                   Result[2][2] = - static_cast<T>(1);
10                  Result[3][0] = - (right + left) / (right - left);
11                  Result[3][1] = - (top + bottom) / (top - bottom);
12                  return Result;
13          }
14
```

glm::ortho

The code above will construct the following matrix:

$$\begin{pmatrix} \frac{2}{R-L} & & & -\frac{R+L}{R-L} \\ & \frac{2}{T-B} & & -\frac{T+B}{T-B} \\ & & -1 & 0 \\ & & & 1 \end{pmatrix}$$

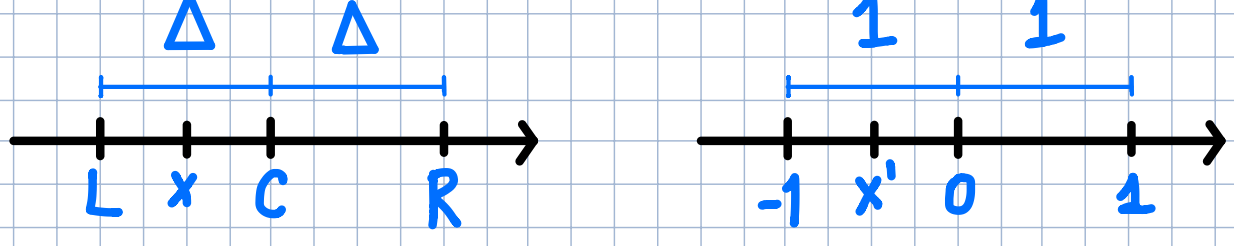
Let's see how a point in space will be transformed

$$\begin{pmatrix} \frac{2}{R-L} & & & -\frac{R+L}{R-L} \\ & \frac{2}{T-B} & & -\frac{T+B}{T-B} \\ & & -1 & 0 \\ & & & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{2x-(R+L)}{R-L} \\ \frac{2y-(T+B)}{T-B} \\ -z \\ 1 \end{pmatrix}$$

$$P \longmapsto P'$$

Let's focus on P'_x:

$$P'_x = \frac{2x-(R+L)}{R-L} = \frac{x - \frac{R+L}{2}}{\frac{R-L}{2}} = \frac{x-C}{\Delta} = x'$$



In short: points from interval $[L,R]$ will be mapped proportionally to points in $[-1,1]$.

Similar conclusions hold for y coordinate.

Notice also that this transformation is equivalent to:

1) Scaling by the vector $\left(\frac{2}{R-L}, \frac{2}{T-B}, -1\right)$

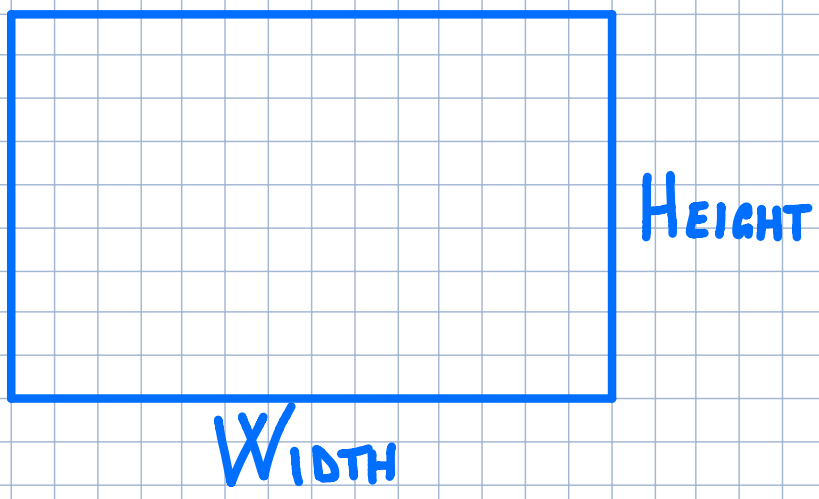
THEN

2) Translating by the vector $\left(-\frac{R+L}{R-L}, -\frac{T+B}{T-B}, 0\right)$

/ ——— /

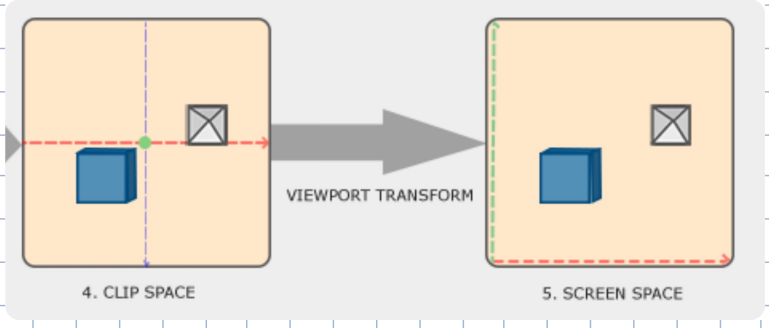
USING GLM::ORTHO IN A WINDOW

Suppose we had a GLFWwindow in our app:



Let's take some steps back first...

REMINDER: Last step of coordinate transformation is VIEWPORT TRANSFORM, which goes from CLIP SPACE to SCREEN SPACE:



Clip space uses NORMALIZED DEVICE COORDINATES, which is basically $[-1,1] \times [-1,1]$.

Screen space coordinates are defined by glViewport, we will assume the default here: $[0,W] \times [0,H]$.

After each shader run, OpenGL will discard every point outside of CLIP SPACE, and will use VIEWPORT TRANSFORM to map the remaining points to SCREEN SPACE.

Below the VT function:

(CLIP SPACE) (SCREEN SPACE)

$$VT : [-1,1] \times [-1,1] \longrightarrow [0,W] \times [0,H]$$

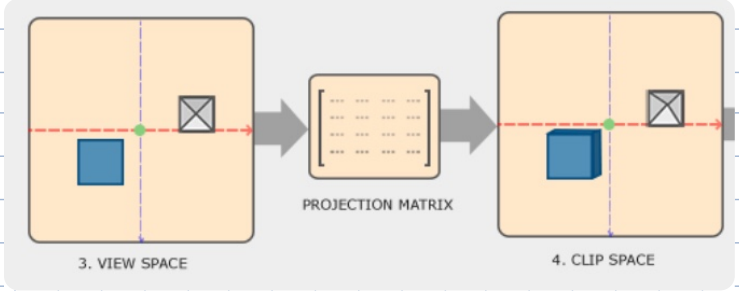
$$(x,y) \longmapsto \left(\frac{W}{2}\cdot(x+1) , \frac{H}{2}(y+1) \right)$$

Notice that every polygon in clip space will be scaled of a factor $\frac{W}{2}$ horizontally and of a factor $\frac{H}{2}$ vertically.

/——/

Back to our problem: we want to study the final result of applying our orthographic projection, considering the Viewport Transform as well.

NOTE: The projection matrix step is just the step preceding the VT step:



The VIEW SPACE is the 3D-coordinates system as seen by the camera.

Now suppose that our vertex shader sets as gl_Position (the final vertex position of a vertex in clip space) the product of the above matrix and the original vertex position. What happens to a rectangle (IN THE XY-PLANE) $a \times b$?

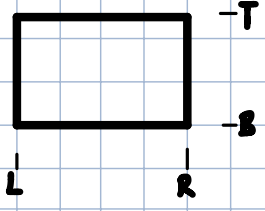
1.) $a' = \frac{2}{R-L} a$ $\quad b' = \frac{2}{T-B} b$ (PROJECTION MATRIX)

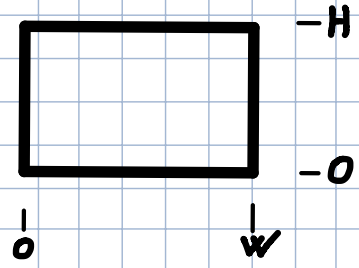
2.) $a'' = \frac{W}{2} a'$ $\quad b'' = \frac{H}{2} b'$ (VIEWPORT TRANSFORM)

$$\longrightarrow a'' = \frac{W}{R-L} a \qquad b'' = \frac{H}{T-B} b$$

What about the aspect ratio?

$$\frac{a''}{b''} = \frac{W}{R-L} \cdot \frac{T-B}{H} \cdot \frac{a}{b} = \frac{W/H}{(R-L)/(T-B)} \frac{a}{b}$$

So if choose the parameters for our matrix in a way that 

is proportional to 

then we'll get $\frac{a''}{b''} = \frac{a}{b}$: ratio will be preserved!

EXAMPLE

WINDOW

WIDTH = 720

HEIGHT = 540

ORTHO-PROJ

L = 0 R = 720

B = 0 T = 540

In this case we get $\frac{a''}{b''} = \frac{a}{b}$, so

if our vertex buffer position defines a square (IN THE XY-PLANE),

then we'll get a square as well to the screen,

in general the ratio of every polygon in the

XY-plane won't change.