Out[1]: (Show / Hide code)

Projections — Parallel

CSE 4303 / CSE 5365 Computer Graphics

Brian A. Dalio, 2019 Spring

Vertex Transformation

The vertices in the data file are all expressed in *world* coordinates, so they must be transformed to *screen* (also known as *viewport*) coordinates before they can be used to draw lines on a canvas.

In this case, we are using a (simple) parallel projection along the positive z axis. We can therefore just ignore the vertex's z coordinate in the transformation.

The transformation is straightforward. The steps in the transformation are:

- Translate the vertex to the world's coordinate origin with (f_x, f_y).
- Scale the vertex according to the ratio between the world coordinates and the screen coordinates with (s_x, s_y).
- Translate the vertex to the viewport's coordinate origin with (g_x, g_y).

$$\mathbf{v}' = \mathbf{T}_g \mathbf{S}_{xy} \mathbf{T}_f \mathbf{v}$$

Out[2]:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & g_x \\ 0 & 1 & g_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & f_x \\ 0 & 1 & f_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} s_x & 0 & f_x s_x + g_x \\ 0 & s_y & f_y s_y + g_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} f_x s_x + g_x + s_x x \\ f_y s_y + g_y + s_y y \\ 1 \end{bmatrix}$$

In this sort of transformation, the two translations are generally not the inverses of each other, so there will not usually be any sort of cancellation.

Each of these transformations is an affine transformation and so is the composition of all three of them. Therefore, the w element is 1 for the original vertex, all of the transformation matrices, and the resulting transformed vertex.

We can extract the expressions for x' and y' from the final form and, once f_x, f_y, s_x, s_y, g_x and g_y are known, compute the viewport coordinates for any vertex directly from its world coordinates.

Out[3]:

$$x' = f_x s_x + g_x + s_x x$$

$$y' = f_y s_y + g_y + s_y y$$

Since f_x, f_y, s_x, s_y, g_x and g_y are constants (once the canvas width and canvas height and the `w` and `s` parameters are known), we see that the transformation of each world coordinate is a multiplication by a constant (s_x for the x coordinate and s_y for the y coodinate) and an addition of a constant ($a_x = f_x s_x + g_x$ for the x coordinate and $a_y = f_y s_y + g_Y$ for the y coordinate.)

$$x' = a_x + s_x x$$

$$y' = a_y + s_y y$$

Reducing the transformation to this form results in a considerable savings in computation cost when transforming large numbers of vertices. Remember, however, that the f_x, f_y, s_x, s_y, g_x and g_y values (and therefore the a_x and a_y values) must be recalculated whenever any of the `w` or `s` parameters, or the canvas width or canvas height changes.

Computing f_x, f_y, s_x, s_y, g_x and g_y

Seems simple enough.

All we have to do now is determine the values of f_x, f_y, s_x, s_y, g_x and g_y (and a_x and a_y).

Each data file has a `w` line and an `s` line.

The `w` line provides the definition of how much of the *world* should be shown in the display; it's our *window* onto the *world*. We will be displaying all items that fit in the rectangle defined by $w_{xmin}, w_{ymin}, w_{xmax}, w_{ymax}$.

The translation-to-the-world-coordinate-origin parameters f_x and f_y are simply the negation of the w_{xmin} and w_{ymin} parameters given in the definition of the window onto the world by the `w` line in the data file. It's a negation because we are translating the vertex *to* the origin.

$$f_x = -w_{xmin}, \quad f_y = -w_{ymin}$$

The `s` line provides the *screen* (*viewport*) definition in *normalized* coordinates, that is, they will range from 0.0 to 1.0. This line defines how much of the *canvas* the display of the objects should take up, between $v_{xmin}, v_{ymin}, v_{xmax}, v_{ymax}$. When we actually display the objects, these normalized coordinates have to be transformed to pixel coordinates on the canvas. (These parameters are labelled v for *viewport*. Do not confuse them with v for *vertex*, as each vertex has three elements, its x, y, and z coordinates. We do not use s for *screen* here as s is used for the *scaling* factors below.)

The translation-to-the-viewport-coordinate-origin parameters g_x and g_y are the v_{xmin} and v_{ymin} parameters given in the definition of the viewport by the `s` line in the data file scaled to fit the size of the canvas.

$$g_x = width \cdot v_{xmin}, \quad g_y = height \cdot v_{ymin}$$

where $width$ is the width of the canvas in pixels and $height$ is the height of the canvas in pixels. There is no negation here because we are translating the vertex *away from* the origin.

The scaling factors take care of transforming the scale of the world vertices to that of the viewport, which is measured in canvas pixels.

$$s_x = \frac{width \cdot (v_{xmax} - v_{xmin})}{w_{xmax} - w_{xmin}}$$

$$s_y = \frac{height \cdot (v_{ymax} - v_{ymin})}{w_{ymax} - w_{ymin}}$$

Once f_x, f_y, s_x, s_y, g_x and g_y are known, we can compute a_x and a_y.

$$a_x = f_x s_x + g_x$$

$$a_y = f_y s_y + g_y$$

While using a_x and a_y in performing the transformation of a vertex is not required (one can do the computation with f_x, f_y, s_x, s_y, g_x and g_y directly), such use can result in a considerable savings in computational cost when transforming large numbers of vertices. Transforming each coordinate costs two multiplies and two additions when using f_x, f_y, s_x, s_y, g_x and g_y directly but only one multiply and one addition when using a_x and a_y, for a savings of 50%.

A Numeric Example

Let's consider the data in the "pyramid.txt" as a specific numeric example.

```
v 0 0 0
v .8 0 0
v 0 .8 0
v .8 .8 0
v 0.4 0.4 .7
f 4 2 1
f 3 4 1
f 2 5 1
f 5 3 1
f 2 4 5
f 3 5 4
w -1.0 -1.0 1 1
s 0.1 0.1 0.9 0.9
```

We will assume a canvas size of $width, height = (500, 400)$.

Therefore, we have for the world space and screen space definitions:

Out[4]:

$$w_{xmin} = -1, w_{ymin} = -1, w_{xmax} = 1, w_{ymax} = 1$$

$$v_{xmin} = 0.1, v_{ymin} = 0.1, v_{xmax} = 0.9, v_{ymax} = 0.9$$

Which gives us the following f_x, f_y, s_x, s_y, g_x, and g_y values:

$$(f_x, f_y) = (-w_{xmin}, -w_{ymin}) = (1, 1)$$

$$(g_x, g_y) = (width \cdot v_{xmin}, height \cdot v_{ymin}) = (500 \cdot 0.1, 400 \cdot 0.1) = (50.0, 40.0)$$

$$s_x = \frac{width \cdot (v_{xmax} - v_{xmin})}{w_{xmax} - w_{xmin}} = \frac{500 \cdot (0.9 - 0.1)}{1 - -1} = 200.0$$

$$s_y = \frac{height \cdot (v_{ymax} - v_{ymin})}{w_{ymax} - w_{ymin}} = \frac{400 \cdot (0.9 - 0.1)}{1 - -1} = 160.0$$

Out[5]: The transformation matrix is therefore:

$$\mathbf{T} = \mathbf{T}_g \mathbf{S}_{xy} \mathbf{T}_f$$

$$= \begin{bmatrix} 1 & 0 & g_x \\ 0 & 1 & g_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & f_x \\ 0 & 1 & f_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} s_x & 0 & f_x s_x + g_x \\ 0 & s_y & f_y s_y + g_y \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 50.0 \\ 0 & 1 & 40.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 200.0 & 0 & 0 \\ 0 & 160.0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 200.0 & 0 & 250.0 \\ 0 & 160.0 & 200.0 \\ 0 & 0 & 1 \end{bmatrix}$$

The transform of a vertex $\mathbf{v}$ is therefore:

$$= \begin{bmatrix} 200.0 & 0 & 250.0 \\ 0 & 160.0 & 200.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 200.0x + 250.0 \\ 160.0y + 200.0 \\ 1 \end{bmatrix}$$

Or, more concisely since all of the transforms are affine and we therefore know the w element will stay 1, we have:

$$(x', y') = (200.0x + 250.0, 160.0y + 200.0)$$

We can get the same transformation by computing a_x and a_y from f_x, f_y, s_x, s_y, g_x and g_y directly.

$$a_x = f_x s_x + g_x = 1 \cdot 200.0 + 50.0 = 250.0$$

$$a_y = f_y s_y + g_Y = 1 \cdot 160.0 + 40.0 = 200.0$$

$$(x', y') = (s_x x + a_x, s_y y + a_y) = (200.0x + 250.0, 160.0y + 200.0)$$

As soon as we know the world space, screen space, and canvas size parameters, the entire transformation for one vertex is reduced to one multiply and one add for the x coordinate and one multiply and one add for the y coordinate. As was observed earlier, by doing this composition we save enormously in computational costs.

For reference, while the pyramid object has only five vertices and six faces, the teapot object has 1,536 vertices and 1,176 faces, and the cow object has 2,904 vertices and 5,804 faces. And these are *tiny* objects in the CG world.

Out[6]: We can use this transform to convert the vertices of the pyramid object from world space to screen space with a canvas size of $width, height = (500, 400)$ thusly,

$$\mathbf{v}'_1 = \begin{bmatrix} 200.0 & 0 & 250.0 \\ 0 & 160.0 & 200.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.0 \\ 0.0 \\ 1 \end{bmatrix} = \begin{bmatrix} 250.0 \\ 200.0 \\ 1 \end{bmatrix} = (250.0, 200.0)$$

$$\mathbf{v}'_2 = \begin{bmatrix} 200.0 & 0 & 250.0 \\ 0 & 160.0 & 200.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.8 \\ 0.0 \\ 1 \end{bmatrix} = \begin{bmatrix} 410.0 \\ 200.0 \\ 1 \end{bmatrix} = (410.0, 200.0)$$

$$\mathbf{v}'_3 = \begin{bmatrix} 200.0 & 0 & 250.0 \\ 0 & 160.0 & 200.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.0 \\ 0.8 \\ 1 \end{bmatrix} = \begin{bmatrix} 250.0 \\ 328.0 \\ 1 \end{bmatrix} = (250.0, 328.0)$$

$$\mathbf{v}'_4 = \begin{bmatrix} 200.0 & 0 & 250.0 \\ 0 & 160.0 & 200.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.8 \\ 0.8 \\ 1 \end{bmatrix} = \begin{bmatrix} 410.0 \\ 328.0 \\ 1 \end{bmatrix} = (410.0, 328.0)$$

$$\mathbf{v}'_5 = \begin{bmatrix} 200.0 & 0 & 250.0 \\ 0 & 160.0 & 200.0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0.4 \\ 0.4 \\ 1 \end{bmatrix} = \begin{bmatrix} 330.0 \\ 264.0 \\ 1 \end{bmatrix} = (330.0, 264.0)$$

This conversion shows the use of the transformation matrix itself, but each one of those calculations could have been done using s_x, s_y, a_x, and a_y in the simplified equation shown above.

Out[7]:

$$(x', y') = (s_x x + a_x, s_y y + a_y) = (200.0x + 250.0, 160.0y + 200.0)$$

$$\mathbf{v}'_1 = (200.0 \cdot 0.0 + 250.0, 160.0 \cdot 0.0 + 200.0) = (250.0, 200.0)$$

$$\mathbf{v}'_2 = (200.0 \cdot 0.8 + 250.0, 160.0 \cdot 0.0 + 200.0) = (410.0, 200.0)$$

$$\mathbf{v}'_3 = (200.0 \cdot 0.0 + 250.0, 160.0 \cdot 0.8 + 200.0) = (250.0, 328.0)$$

$$\mathbf{v}'_4 = (200.0 \cdot 0.8 + 250.0, 160.0 \cdot 0.8 + 200.0) = (410.0, 328.0)$$

$$\mathbf{v}'_5 = (200.0 \cdot 0.4 + 250.0, 160.0 \cdot 0.4 + 200.0) = (330.0, 264.0)$$

Notice how we get the same results no matter which way we do the calculation. (Whew!)

You can use these results to check your calculations for the pyramid test case. If your code works correctly for that case, you should not have any trouble with the teapot or cow data sets.

These results are correct only for the given world space and screen space definitions and the canvas size of $width, height = (500, 400)$. While the world space and screen space definitions are set by the data file, the canvas size changes whenever the user resizes the display window. Therefore each time the display window size changes, the transformation matrix has to be recalculated, all of the vertices have to be retransformed, and all of the faces have to be redrawn.