3D Transformations (II)

Previously, we have discussed linear and affine transformations in $\mathbb{R}^3$ and their matrix representations using homogeneous coordinates. Now is the time to discuss their applications in 3D viewing together with a final class of powerful transformations: projections.

3D viewing

A (geometric) model of an object can be stored in the computer as a list of 3D vertices $\mathbf{V} = \{\mathbf{v}_i\}$, faces as groupings of vertices, and face normals $\mathbf{N} = \{\hat{\mathbf{n}}_j\}$ specifying the normal direction of each face. On a high level, viewing is a transformation $\mathcal{V} : \mathbb{R}^3 \to \mathbb{R}^3$ that maps $\mathbf{V}$ and $\mathbf{N}$ to a "canonical" space named normalized device coordinates (NDC). The viewing transformation can be written as a composition of

1. **Model** transformation $\mathcal{M}$
2. **Camera (View)** transformation $\mathcal{C}$
3. **Projection** transformation $\mathcal{P}$,

in that $\mathcal{V} = \mathcal{P} \circ \mathcal{C} \circ \mathcal{M}$. To understand all these transforms, it helps conceptually to imagine a virtual camera that captures a scene. Say we want to take a photo of a toy Teddy Bear. What we have to do are the following:

1. Place Ted in a nice spot and pose (say close to the window, rotated a bit so his left body is lit by the Sun)
2. Pick a right place and angle for the camera (say pointing straight to Ted in a full body portrait)
3. Install the right lens onto the camera and "Cheese!"

These three steps basically correspond to the three transforms introduced above, as we shall see next.

Model transformation

The model transformation $\mathcal{M}$ is an affine transformation that takes a model to wherever you want it to be in the scene. It also rotates the model in any direction you want, stretching and shearing it if necessary (poor Ted). As we saw last time such a transformation can be expressed as a 4x4 matrix of the form

$$\mathbf{M} = \left[\begin{array}{c|c} \mathbf{T}_{3\times 3} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{array}\right] \tag{1}$$

encoding the transform

$$\begin{aligned} \mathcal{M}(\mathbf{v}) &= \mathbf{T}\mathbf{v} + \mathbf{t} && \text{for vertex } \mathbf{v} \\ \mathcal{M}(\mathbf{d}) &= \mathbf{T}\hat{\mathbf{d}} && \text{for direction } \hat{\mathbf{d}} \end{aligned}$$

Now suppose Ted feels a bit lonely and wants his buddy John in the photo next to him. We can use another model transformation $\mathcal{M}'$ to transform John's vertices and normals to where Ted wants him to be. In general, each model in a scene has its own model transform that mapping the vertices in its local coordinates to a shared, world coordinate space.

$$\left\{\mathbf{M}^{(1)} = \left[\begin{array}{c|c} \mathbf{T}^{(1)} & \mathbf{t}^{(1)} \\ \mathbf{0}^\top & 1 \end{array}\right], \ldots, \mathbf{M}^{(n)} = \left[\begin{array}{c|c} \mathbf{T}^{(n)} & \mathbf{t}^{(n)} \\ \mathbf{0}^\top & 1 \end{array}\right]\right\} \tag{2}$$

The coordinates system after applying model transform is called the *world space coordinates*.

Camera (View) transformation

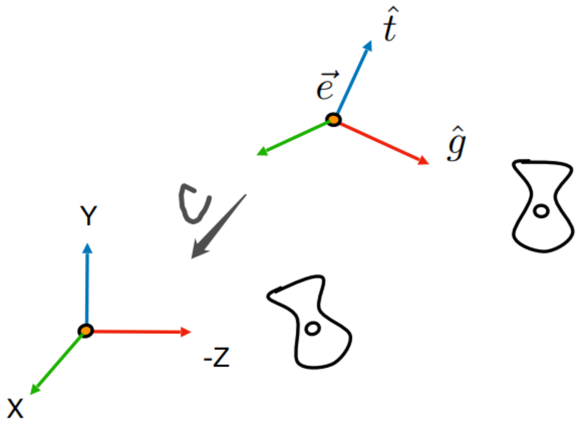
With Ted and his friends ready, time to take out the camera. The camera or view transform $\mathcal{C}$, as the name suggests, effectively "puts the camera at a nice angle". To fully determine a camera's orientation in space three parameters are needed:

- Position $\mathbf{e}$ (Where is the camera?)
- Look-at / gaze direction $\hat{\mathbf{g}}$ (Which direction is the camera pointing to?)
- Up direction $\hat{\mathbf{t}}$ (Which direction is the top of the camera pointing to?)

The camera transformation is defined as the rigid transformation mapping the vectors

$$\begin{gathered}\mathcal{C}\\ \mathbf{e} \mapsto \mathbf{0}\\ \mathbf{e}+\hat{\mathbf{t}} \mapsto \hat{\mathbf{y}}\\ \mathbf{e}+\hat{\mathbf{g}} \mapsto -\hat{\mathbf{z}}\end{gathered}$$

Equivalently, it can be understood as a change of coordinates from the $[\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}}]$ system to the new system $\mathbf{Q} = [\hat{\mathbf{g}} \times \hat{\mathbf{t}}, \hat{\mathbf{t}}, -\hat{\mathbf{g}}]$ centered at $\mathbf{e}$. In plain English, we want the camera to be located at the origin, looking at $-\hat{\mathbf{z}}$, with its "up" direction aligned with $\hat{\mathbf{y}}$.



Let's derive a matrix representation for $\mathcal{C}$. We first translate $\mathbf{e}$ to $\mathbf{0}$ with

$$\mathbf{T} = \left[\begin{array}{c|c} \mathbf{I} & -\mathbf{e} \\ \mathbf{0}^\top & 1 \end{array}\right] \tag{3}$$

Next, we need a rotation matrix $\mathbf{R}$ to turn $\hat{\mathbf{t}}$ to $\hat{\mathbf{y}}$ and $\hat{\mathbf{g}}$ to $-\hat{\mathbf{z}}$. The inverse matrix is easy to write:

$$\mathbf{R}^{-1} = \left[\begin{array}{c|c} \mathbf{Q} & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{array}\right] = \left[\begin{array}{ccc|c} \hat{\mathbf{g}} \times \hat{\mathbf{t}} & \hat{\mathbf{t}} & -\hat{\mathbf{g}} & \mathbf{0} \\ 0 & 0 & 0 & 1 \end{array}\right] \tag{4}$$

This is an orthogonal matrix. Thus

$$\mathbf{R} = (\mathbf{R}^{-1})^\top = \left[\begin{array}{c|c} \mathbf{Q}^\top & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{array}\right] \tag{5}$$

from which we get

$$\mathbf{C} = \mathbf{R}\mathbf{T} = \left[\begin{array}{c|c} \mathbf{Q}^\top & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{array}\right] \left[\begin{array}{c|c} \mathbf{I} & -\mathbf{e} \\ \mathbf{0}^\top & 1 \end{array}\right] = \left[\begin{array}{c|c} \mathbf{Q}^\top & -\mathbf{Q}^\top\mathbf{e} \\ \mathbf{0}^\top & 1 \end{array}\right] \tag{6}$$

which encodes the transform

$$\begin{aligned} \mathcal{C}(\mathbf{v}) &= \mathbf{Q}^\top(\mathbf{v}-\mathbf{e}) && \text{for vertex } \mathbf{v} \\ \mathcal{C}(\mathbf{d}) &= \mathbf{Q}^\top\hat{\mathbf{d}} && \text{for direction } \hat{\mathbf{d}} \end{aligned}$$

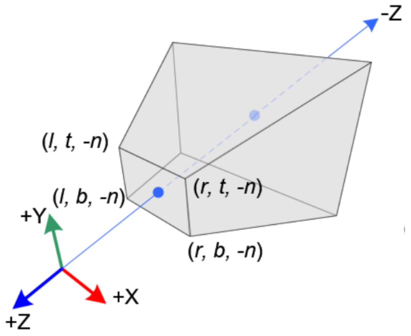
The coordinates system after applying model and camera transforms is called the *camera space coordinates*.

Projection transformation

The (perspective) projection transformation $\mathcal{P}$ "squishes", translates and scales a so-called the *view frustum* to the canonical cube $K = [-1, 1]^3$. The view frustum is a right, rectangular frustum in front of the camera whose height is parallel to the z axis of the camera space.

Denote the rectangular face of the frustum closer to the camera by N and the father one F. The frustum, and thus transform $\mathcal{P}$, can be determined by the following parameters:

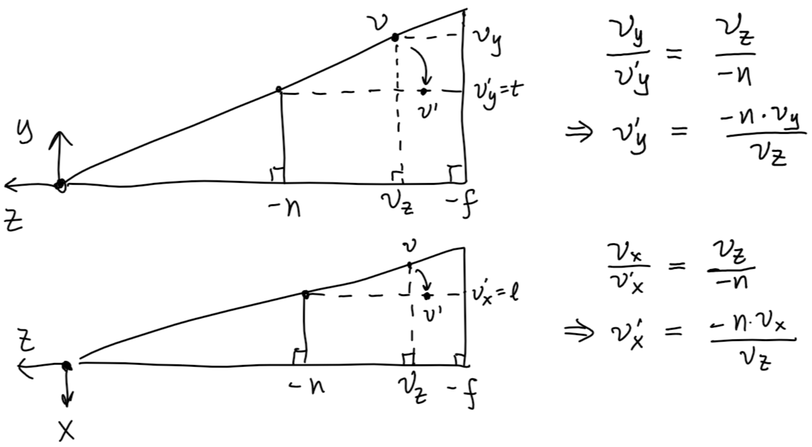
Parameter	n **(near)**	f **(far)**	t **(top)**	b **(bottom)**	r **(right)**	l **(left)**
means...	$-N_z$	$-F_z$	$\max N_y$	$\min N_y$	$\max N_x$	$\min N_x$



"Squishing"

Let's assume the squishing transform $\mathcal{S}$ can be represented by a 4x4 matrix $\mathbf{S}$. We shall derive what $\mathbf{S}$ should be. We wish to map the view frustum to the cuboid $C = [l, r] \times [b, t] \times [-f, -n]$. Additionally, we require the transform $\mathcal{S}$ to

- scale the x and y coordinates of each points so that every cross section of the frustum after transformation has the same dimension as N, and
- keep the z coordinates of the near and far planes unchanged.


v
v_y
$v'_y = t$
v'
y
z
$-n$
v_z
$-f$
$\frac{v_y}{v'_y} = \frac{v_z}{-n}$
$\Rightarrow v'_y = \frac{-n \cdot v_y}{v_z}$
v
$v'_x = l$
v'
z
x
$-n$
v_z
$-f$
$\frac{v_x}{v'_x} = \frac{v_z}{-n}$
$\Rightarrow v'_x = \frac{-n \cdot v_x}{v_z}$

By inspecting two sets of similar triangles above, we know $\mathcal{S}$ should scale each x-y plane uniformly by a factor $-n/z$ where z is the z coordinate of the plane. That is,

$$\mathcal{S}(\mathbf{v}) = \mathcal{S}(v_x, v_y, v_z) = \left(\frac{-nv_x}{v_z}, \frac{-nv_x}{v_z}, v_z' \right) \tag{7}$$

In homogeneous coordinates,

$$\mathbf{S}\tilde{\mathbf{v}} = \mathbf{S} \begin{bmatrix} v_x \\ v_y \\ v_z \\ 1 \end{bmatrix} \sim \begin{bmatrix} -nv_x/v_z \\ -nv_y/v_z \\ v_z' \\ 1 \end{bmatrix} \sim \begin{bmatrix} -nv_x \\ -nv_y \\ v_z' v_z \\ v_z \end{bmatrix} \tag{8}$$

Thus we know $\mathbf{S}$ must have the form

$$\mathbf{S} = \begin{bmatrix} -n & 0 & 0 & 0 \\ 0 & -n & 0 & 0 \\ a & b & c & d \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{9}$$

where a, b, c, d are unknown. To find out these unknowns, observe that after squishing

1. The entire near plane remains unchanged
2. The intersection point on the far plane $(0, 0, -f)$ remains unchanged

The first condition implies that

$$\mathbf{S} \begin{bmatrix} v_x \\ v_y \\ -n \\ 1 \end{bmatrix} = \begin{bmatrix} -nv_x \\ -nv_y \\ av_x + bv_y - cn + d \\ -n \end{bmatrix} \sim \begin{bmatrix} v_x \\ v_y \\ \frac{av_x + bv_y - cn + d}{-n} \\ 1 \end{bmatrix} = \begin{bmatrix} v_x \\ v_y \\ -n \\ 1 \end{bmatrix} \tag{10}$$

for all v_x, v_y. Hence $a = b = 0$ and $d - cn = n^2$.

The second condition implies

$$\mathbf{S} \begin{bmatrix} 0 \\ 0 \\ -f \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -cf + d \\ -f \end{bmatrix} \sim \begin{bmatrix} 0 \\ 0 \\ \frac{-cf+d}{-f} \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -f \\ 1 \end{bmatrix} \tag{11}$$

Hence $d - cf = f^2$. Solving the above system yields

$$\begin{array}{l} a = b = 0 \\ c = -(n+f), d = -nf \end{array} \implies \mathbf{S} = \begin{bmatrix} -n & 0 & 0 & 0 \\ 0 & -n & 0 & 0 \\ 0 & 0 & -(n+f) & -nf \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{12}$$

Translation & Scaling

Now that the frustum has been squished to the cuboid C, all there's left to do is map the cuboid to the canonical cube K. This is straightforward to accomplish. First, translate the center of the cuboid $(\frac{l+r}{2}, \frac{t+b}{2}, -\frac{f+n}{2})$ to the origin. Then stretch along each axis (respectively by $\frac{2}{r-l}, \frac{2}{t-b}, \frac{2}{f-n}$) so that all side-lengths of the cuboid become 2. The overall transform is given by the matrix

$$\mathbf{A} = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & \frac{l+r}{l-r} \\ 0 & \frac{2}{t-b} & 0 & \frac{b+t}{b-t} \\ 0 & 0 & \frac{2}{f-n} & \frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}$$

Putting it together

The projection matrix is therefore the product

$$\mathbf{P} = \mathbf{AS} = \begin{bmatrix} \frac{-2n}{r-l} & 0 & \frac{l+r}{l-r} & 0 \\ 0 & \frac{-2n}{t-b} & \frac{b+t}{b-t} & 0 \\ 0 & 0 & \boxed{\dfrac{n+f}{n-f}} & \frac{2nf}{n-f} \\ 0 & 0 & \boxed{1} & 0 \end{bmatrix} \tag{14}$$

The coordinates system after applying the model, camera, and projection transformations is called the *normalized device coordinates* (NDC).

Note: In some graphics API's like OpenGL, the NDC system is defined to be left-handed (camera pointing to $\hat{\mathbf{z}}$ instead of $-\hat{\mathbf{z}}$) so that a larger z coordinate in NDC corresponds to greater depth from the camera. To get this type of projection matrix, simply negate the boxed elements in (14).

References

[1] https://sites.cs.ucsb.edu/~lingqi/teaching/games101.html

[2] http://www.songho.ca/opengl/gl_projectionmatrix.html

Next: Rasterization

To be continued.