Viewing II: Camera,Projections and their Relations

Positioning and Orienting the Camera

- Positioned (VRP=view reference point) by

  `set_view_reference_point(x,y,z)`

- Orientation (VPN=view plane normal and VUP = view-up vector) by

  `set_view_plane_normal(nx,ny,nz)  and set_view_up(vup_x,vup_y,vup_z)`

- The projection of the VUP onto the view-plane is a up-direction vector v
- u = v x n a vector orthogonal to v and n . The (u,v,n) and the VRP yields the viewing coordinate system
- Camera is usually located at a point e called the eye point, and it is pointed at the at point a. This defines VRP, and VPN = e - a. Finally use the OpenGL utitility function gluLookAt ()

```
glMatrixMode(GL_MODELVIEW); glLoadIdentity();
gluLookAt(eyex,eyey,eyez,atx,aty,atz,upx,upy,upz);
```

Projections

- Mapping from 3 dimensional space to 2 dimensional subspace
- Range of any projection $\mathcal{P} : R^3 \longrightarrow R^2$ called a *projection plane*
- $\mathcal{P}$ maps lines to points
- The image of any point $\mathbf{p}$ under $\mathcal{P}$ is the intersection of a *projection line* through $\mathbf{p}$ with the projection plane.

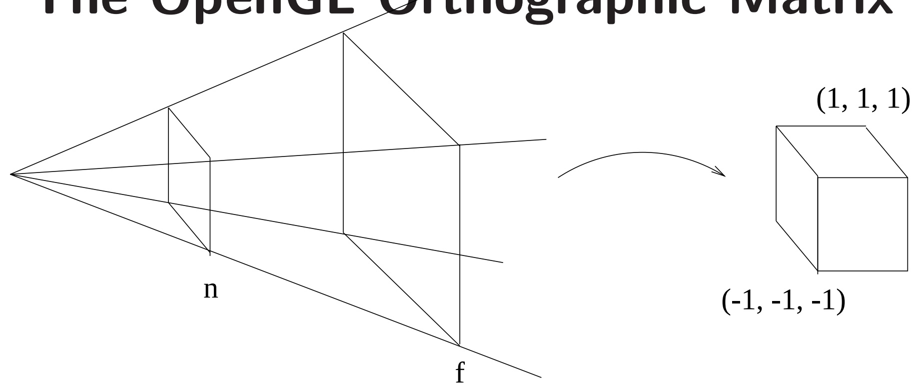
Taxonomy of Projections

- Parallel
  - orthographic
  - oblique
- Perspective
  - 1-pt
  - 2-pt
  - 3-pt

Parallel Projections

- All projection lines are parallel.
- An *orthographic projection* has projection lines orthogonal to projection plane.
- Otherwise a parallel projection is an *oblique projection*
- Particularly interesting oblique projections are the *cabinet projection* and the *cavalier projection*.

The OpenGL Orthographic Matrix



- The visible volume in world space is known as the *viewing volume*.
- Specify with the call **glOrtho**(l, r, b, t, n, f)
- In OpenGL, the window is in the *near* plane
- l and r are u-coordinates of left and right window boundaries in the near plane
- b and t are v-coordinates of bottom and top window boundaries in the near plane
- n and f are *positive distances* from the eye along the viewing ray to the near and far planes
- The left and right clipping planes are $x = -1$ and $x = 1$
- The bottom and top clipping planes are $y = -1$ and $y = 1$
- The near and far clipping planes are $z = -1$ and $z = 1$

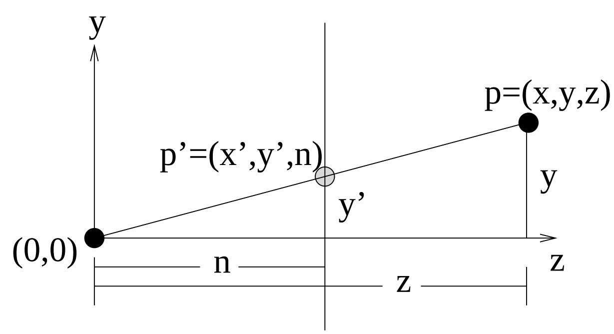
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Perspective Projection

- All projection lines pass through the *center of projection* (eyepoint).
- Therefore also called *central projection*
- This is *not* affine, but rather a *projective transformation*, (also discussed in previous lecture).
- Differences in Perspective
  - 1-pt
  - 2-pt
  - 3-pt

Perspective Transform in Eye Coordinates

- Given a point $\mathbf{p}$, find its projection $\mathcal{P}(\mathbf{p})$
- Convenient to do this in *eye coordinates*, with center of projection at origin and $z = n$ projection plane
- Note that eye coordinates are left-handed

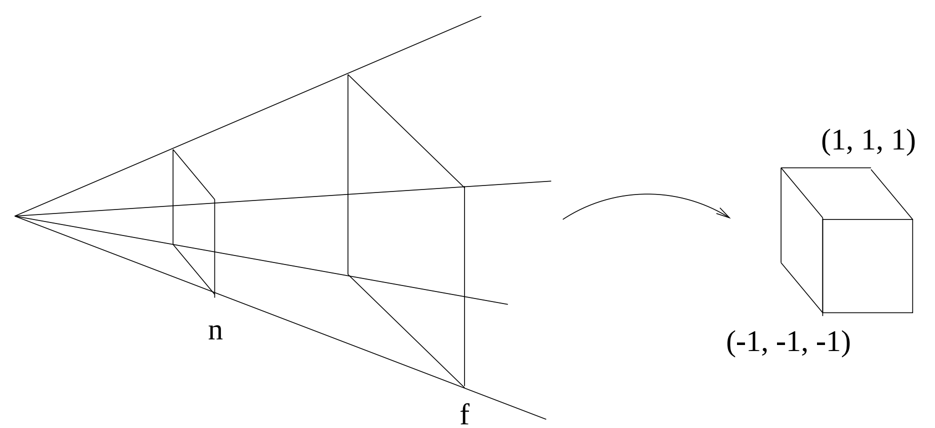


Projection plane, z = n

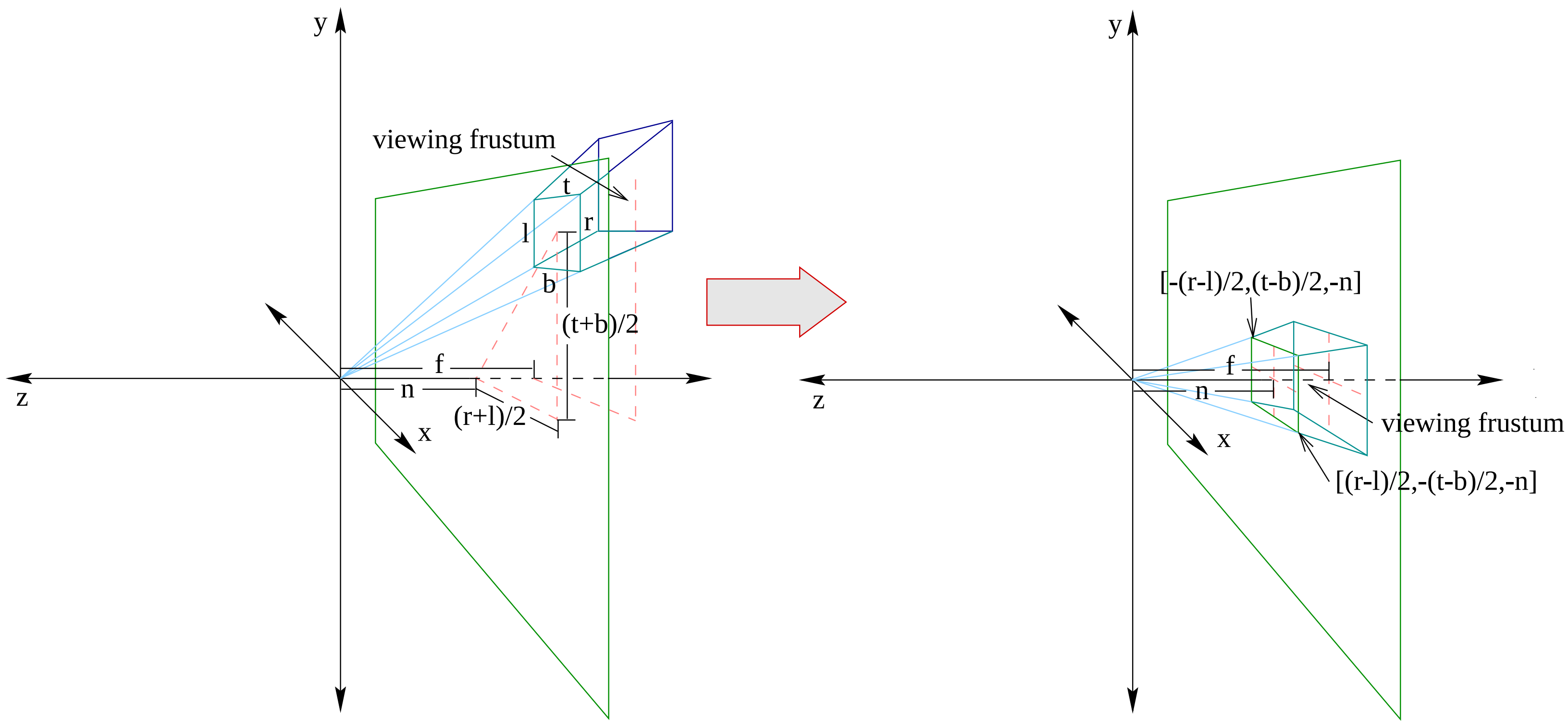
- Due to similar triangles $\mathcal{P}(\mathbf{p}) = (nx/z, ny/z, d)$
- For any other point $\mathbf{q} = (kx, ky, kz), k \neq 0$ on same projection line $\mathcal{P}(\mathbf{q}) = (nx/z, ny/z, n)$
- If we have surfaces, we need to know which ones occlude others from the eye position
- This projection loses all z information, so we cannot do occlusion testing after projection

The OpenGL Perspective Matrix

- The visible volume in world space is known as the *viewing pyramid* or *frustum*.
- Specify with the call **glFrustum**(l, r, b, t, n, f)
- In OpenGL, the window is in the *near* plane
- l and r are u-coordinates of left and right window boundaries in the near plane
- b and t are v-coordinates of bottom and top window boundaries in the near plane
- n and f are *positive distances* from the eye along the viewing ray to the near and far planes
- Maps the left and right clipping planes to $x = -1$ and $x = 1$
- Maps the bottom and top clipping planes to $y = -1$ and $y = 1$
- Maps the near and far clipping planes to $z = -1$ and $z = 1$



Manipulating the Camera



- After applying the modelview matrix, we are looking down the $-z$ axis.
- We need to move the ray from the origin through the window center onto the $-z$ axis.
- Rotation won't do since the window wouldn't be orthogonal to the z axis.
- Translation won't do since we need to keep the eye at the origin.

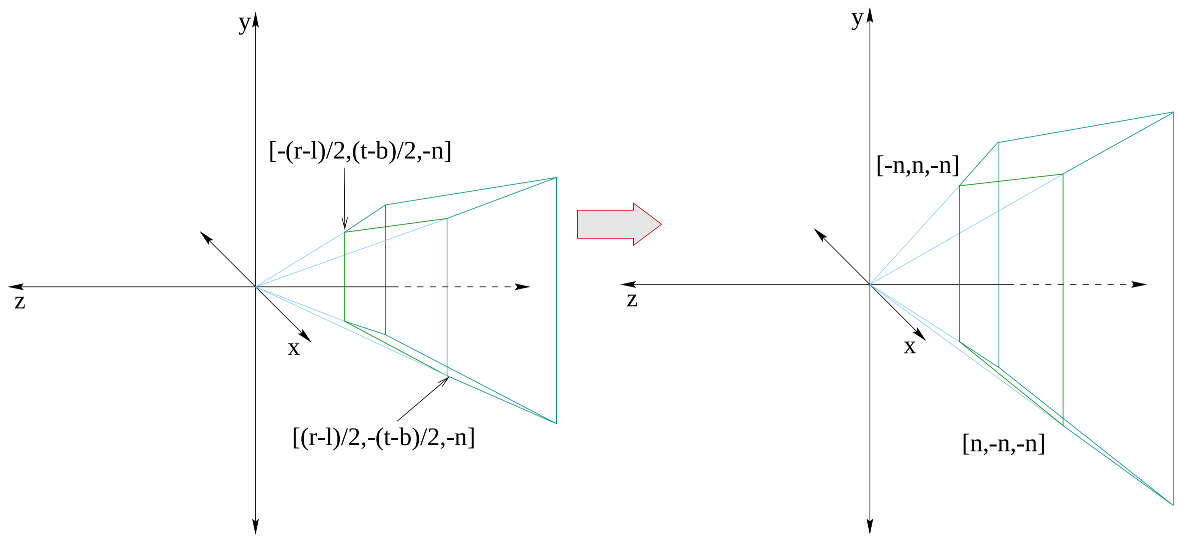
- We need differential translation as a function of z, i.e. shear.
- When $z = -n$, δx should be $-\frac{r+l}{2n}$ and δy should be $-\frac{t+b}{2n}$, so we get

$$
\begin{aligned}
x' &= x + \frac{r+l}{2n} z \\
y' &= y + \frac{t+b}{2n} z \\
z' &= z
\end{aligned}
$$

$$
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & \frac{r+l}{2n} & 0 \\ 0 & 1 & \frac{t+b}{2n} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$

Adjusting the Clipping Boundaries

- For ease of clipping, we want the oblique clipping planes to have equations $x = \pm z$ and $y = \pm z$.
- This will make the window square, with boundaries $l = b = -n$ and $r = t = n$.
- This requires a scale to make the window this size.



Thus the mapping is

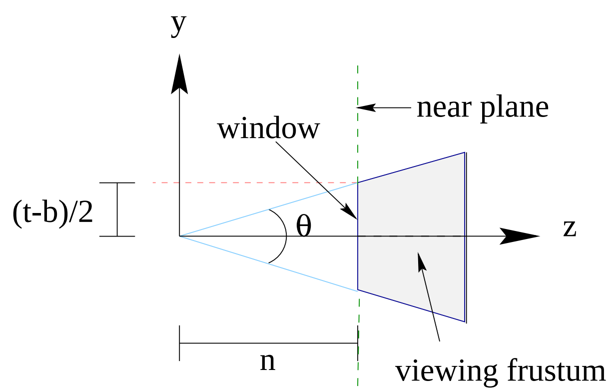
$$
\begin{aligned}
x' &= \frac{2nx}{r-l} \\
y' &= \frac{2ny}{t-b} \\
z' &= z
\end{aligned}
$$

or in matrix form:

$$
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$

Field of View Frustum Scaling

- After the frustum is centered on the $-z$ axis:



- Note that $\frac{n}{t-b} = \cot\left(\frac{\theta}{2}\right)$
- This gives the y mapping $y'' = y' \cot\left(\frac{\theta}{2}\right)$
- Since the window need not be square, we can define the x mapping using the aspect ratio aspect $= \frac{\Delta x}{\Delta y} = \frac{(r-l)}{(t-b)}$
- Then x maps as $x'' = x' \frac{\cot\left(\frac{\theta}{2}\right)}{\text{aspect}}$

- This gives us the alternative scaling formulation:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\cot\left(\frac{\theta}{2}\right)}{\textsf{aspect}} & 0 & 0 & 0 \\ 0 & \cot\left(\frac{\theta}{2}\right) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- This is used by **gluPerspective**(θ, aspect, n, f)

Complete OpenGL Perspective Matrix

- Combining the three steps given above, the complete OpenGL perspective matrix is

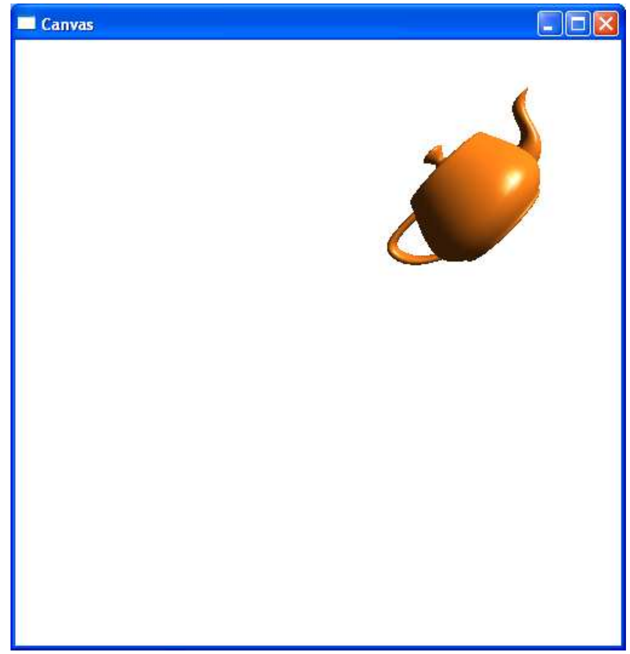
$$\begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & \frac{r+l}{2n} & 0 \\ 0 & 1 & \frac{t+b}{2n} & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Using **gluPerspective** the matrix becomes

$$\begin{bmatrix} \frac{\cot(\theta/2)}{aspect} & 0 & 0 & 0 \\ 0 & \cot(\theta/2) & 0 & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$


```
glMatrixMode (GL_MODELVIEW);
gluLookAt(0,0,0, 0,0,-1, 0,1,0);
glMatrixMode (GL_PROJECTION);
glLoadIdentity();
glFrustum(-1, 1, -1, 1, 2, 7);
glMatrixMode (GL_MODELVIEW);
glutSolidTeapot(1);
```




```
glMatrixMode (GL_MODELVIEW);
gluLookAt(0,0,0, 0,0,-1, 1,1,0);
glMatrixMode (GL_PROJECTION);
glLoadIdentity();
glFrustum(-3, 1, -3, 1, 2, 7);
glMatrixMode (GL_MODELVIEW);
glutSolidTeapot(1);
```

Reading Assignment and News

Chapter 5 pages 233 - 259, of Recommended Text.

Please also track the News section of the Course Web Pages for the most recent Announcements related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics25/cs354/)