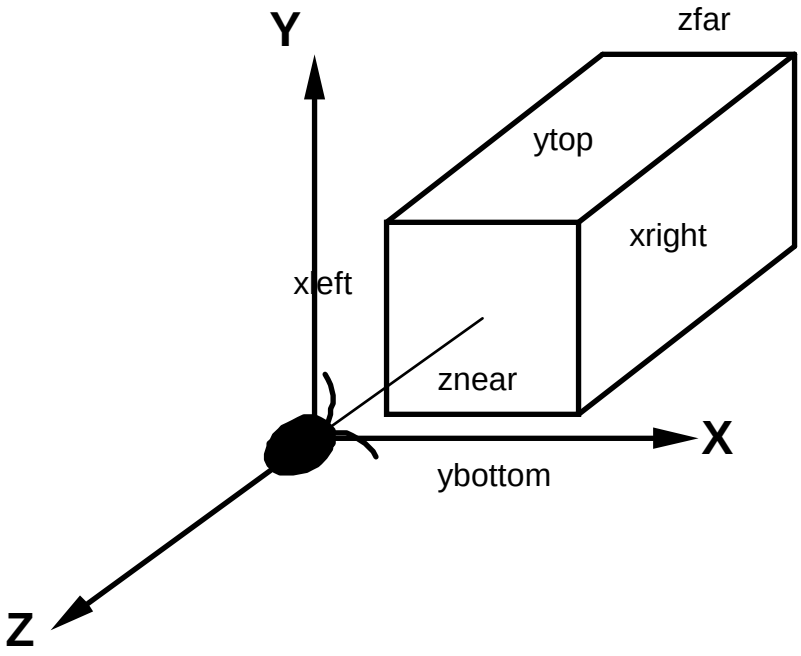
OpenGL 3D Viewing

Orthographic 3D Viewing

When viewing 3D scene data, OpenGL automatically places the eye at the origin, looking in the -Z direction, as shown below:



If this is not how you want to view your scene, you can "change" this situation by moving the scene with the `gluLookAt( )` OpenGL routine:

```
gluLookAt( ex,ey,ez,  cx,cy,cz,  ux,uy,uz )
```

`(ex,ey,ez)` is where you want to view your scene from (i.e., your eye point). `(cx,cy,cz)` is a point you want the eye to be looking at (i.e., the center point). `(ux,uy,uz)` is a vector indicating which direction you want to be considered "up" (i.e., how is your face twisted).

It is convenient to think of `gluLookAt( )` as a way to transform your eye to where you want it to be, but in fact what really happens is that `gluLookAt( )` transforms the scene backwards so that it is in the right place so an eye at the origin sees the correct view.

Once the eye is in this position, the extent of the scene that the program wants to be visible is defined by a call to `glOrtho( )`:

```
glOrtho( xleft, xright, ybottom, ytop, znear, zfar );
```

`xleft, xright, ybottom, ytop, znear,` and `zfar` are all floating point numbers in the units in which the scene is defined. Even though the z-clipping takes place in the -Z region, both `znear` and `zfar` are defined as *distances in front of the eye* and are thus positive numbers. `glOrtho()` defines a rectangular-solid-shaped area as shown above. (It is the rectangular shape of the viewing volume that makes this an *orthographic* view.) Any elements of the scene that exist in this box will be mapped to the viewport and, thus, be seen. Any elements that exist outside this box will be clipped.

It sometimes helps to understand what is happening internally. A 3D point in your scene is mapped into X and Y window coordinates by projecting it onto the `znear` plane. The Z value is left undisturbed. This being an orthographic projection, the 3D point is projected to the `znear` plane by a line parallel to the Z axis. Thus, the projection of the point (X,Y,Z) onto the `znear` plane produces window coordinates of:

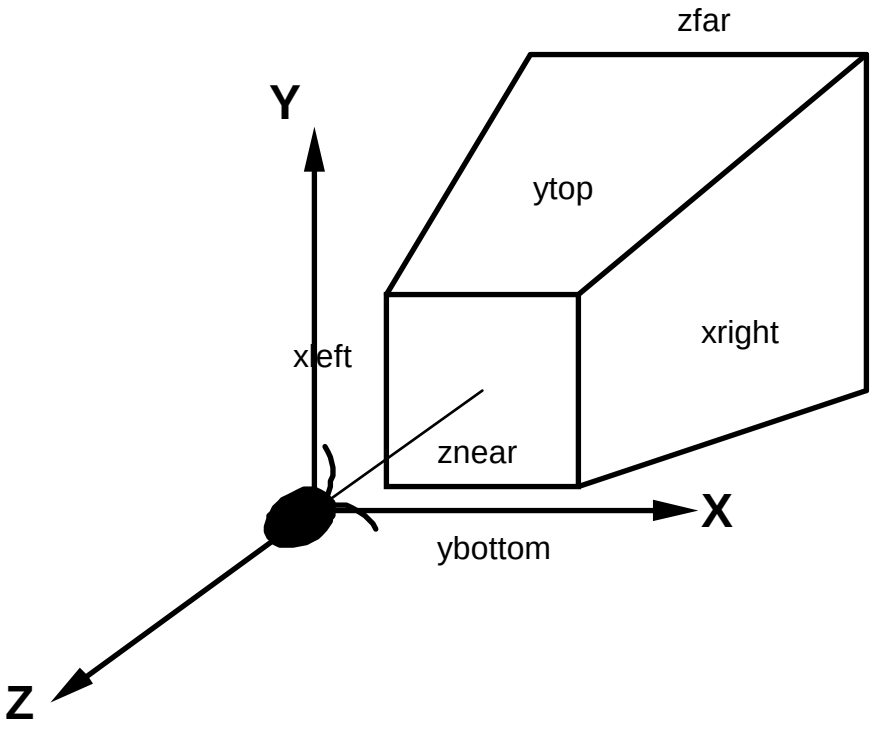
$$Xw = X$$
$$Yw = Y$$

It is tempting to make the `znear` and `zfar` planes enormously far apart so that nothing of the scene is missed. This will work OK, unless z-buffering is being used. The distance `(zfar-znear)` is mapped to the integer range of the z-buffer. Even though the range of values that can be stored in the z-buffer is typically either 2^{16} or 2^{24}, super large value of `(zfar-znear)` will result in a situation where many Z values in the scene map into the same integer Z value in the z-buffer. This is an ugly phenomenon called *z-fighting*.

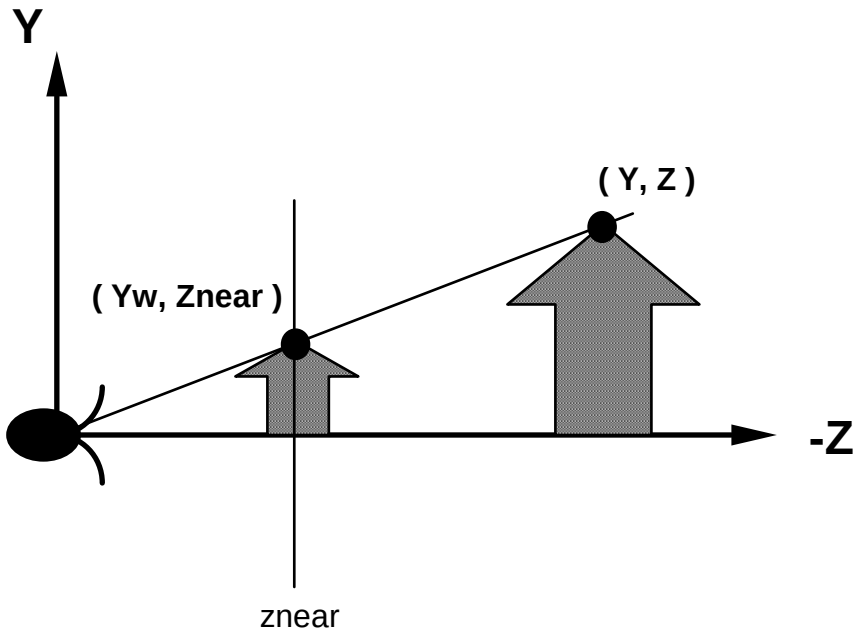
Perspective 3D Viewing

Orthographic 3D viewing works very well. It is straightforward and easy to ask for. But, it has one major deficiency: objects in the scene do not project realistically. In an orthographic projection, an object moving steadily away from us does not get smaller, as we see it happen in real life. We need to define another way of getting a 3D-to-2D projection that gives us the farther-gets-smaller effect that we have come to expect.

In contrast to the orthographic world, the perspective viewing volume is a truncated pyramid, or frustum:



As we did with the orthographic world, we want to project every (X,Y,Z) point in the scene onto the near z-clipping plane to determine X and Y window coordinates. In the perspective case, all projection lines go through the eye position, or origin. If we look at the situation from the +X axis, we see:



From similar triangles, we can compute the window coordinates of a point (X,Y,Z):

$$\frac{Yw}{Y} = \frac{Znear}{-Z}$$

$$\frac{Xw}{X} = \frac{Znear}{-Z}$$

or:

$$Xw = X \cdot \left(\frac{Znear}{-Z} \right)$$

$$Yw = Y \cdot \left(\frac{Znear}{-Z} \right)$$

$$Zw = Z$$

These equations show that:

- As objects get farther away (Z becomes larger negatively), the corresponding Xw and Yw values get smaller.
- Objects behind the observer (Z>0) will be inverted. Do not let this happen.
- Objects at the observer (Z=0) will project to infinity. Do not let this happen.

The OpenGL `glFrustum( )` procedure is the perspective equivalent to `glOrtho()`:

```
glFrustum( xleft, xright, ybottom, ytop, znear, zfar );
```

As before, the values of `xleft, xright, ybottom,` and `ytop` are the boundaries of the scene as projected onto the `znear` plane.

But, specifying boundaries is a little tricky. Recognize that specifying `znear` and the boundaries in `glFrustum( )` is really just the same as specifying the angle of vision that comes off the eye. This is called the *field-of-view* (FOV) angle. The OpenGL `gluPerspective( )` routine lets us specify the FOV and skip the X and Y boundaries:

```
float fov;
float aspect;
float znear, zfar;
   . . .
gluPerspective( fov, aspect, znear, zfar );
```

where `fov` is the field-of-view angle in the Y direction (in degrees), `aspect` is the ratio of width in the X direction to height in the Y direction, and `znear` and `zfar` are as specified before.

There is nothing magical about `gluPerspective( )`:

```
void
gluPerspective( float fovy, float aspect, float near, float far )
	// fovy is the full field of view angle in degrees
	// aspect is the x/y aspect ratio
	// near, far are z clipping distances in front of the eye
{
	float rad;		// half field of view angle in radians
	float left, right;	// x clipping boundaries
	float bottom, top;	// y clipping boundaries

	rad = (M_PI/180.) * (fovy/2.);
	top = near * tan( rad );
	bottom = -top;
	left = aspect * bottom;
	right = aspect * top;
	glFrustum( left, right, bottom, top, near, far );
}
```

Rule of Thumb: a field-of-view value of about 50-100˚ seems to work well.

As with the orthographic projection, it is tempting to make the `znear` and `zfar` planes enormously far apart so that nothing of the scene is missed. This will work OK, unless z-buffering is being used. The distance `(zfar-znear)` is mapped to the integer range of the z-buffer. A super large value of `(zfar-znear)` will result in a situation where many Z values in the scene map into the same integer Z value in the z-buffer. This is an ugly phenomenon called *z-fighting*.

Summary of the Use of Projections in 3D Viewing

Orthographic projections are at their best when:

- Items in the scene need to be checked to see if they line up or are the same size
- Lines need to be checked to see if they are parallel
- We do not care that z-distance is handled unrealistically
- We are not trying to move through the scene

Perspective projections are at their best when:

- Realism counts
- We want to move through the scene
- We do not care that it is difficult to measure or align things

Knowing All This, What Could You Manipulate?

Could Manipulate	Effect
The distance from the eye to the object	Camera “dolly” / zoom
Just the eye position	Walking while staring at something
Just the look-at position	Looking around from a standing position
Both	Walking while looking around
Set the look-at position to be tangent to the eye’s path	Walking while looking straight ahead, roller coaster ride
Up-direction	Banking
Field-of-view angle	Zoom / Wide angle / Fisheye
Z-far	Dense fog (with the OpenGL glFog feature)
Aspect ratio	Cockpit window size