Announcement

Project 1 has been posted online and in dropbox

Due: 11:59:59 pm, Friday, October 14

Project 1: Interactive Viewing of Two Teapots

How to create a teapot?

Before OpenGL 3.0, *glutSolidTeapot*

However, the function *glutSolidTeapot* is deprecated

Create a model or use an existing model

An Obj Model

An obj file stores an existing model

An obj file is structured in lines.

- The lines starting with # are comments
- “o” introduces a new object
- For each following line,
  - v introduces a vertex
  - vn introduces a normal
  - f introduces a face, using vertex indices, starting at 1

An Example of a Teapot Model

A teapot.obj downloaded from http://graphics.stanford.edu/courses/cs148-10-summer/as3/code/as3/teapot.obj

v -3.000000 1.800000 0.000000

v -2.991600 1.800000 -0.081000

v -2.991600 1.800000 0.081000

…

f 2968 2970 3004

f 3022 3021 3001

f 3001 3004 3022

An Obj Model

Load a model

```
void load_obj(const char* filename, vector<vec4>& vertices,
vector<GLushort>& elements,vector<vec3>& normals)
```

Use the model

```
glBufferData( GL_ARRAY_BUFFER, vertices.size()*sizeof(vec4),
&vertices[0], GL_STATIC_DRAW );
```

Display

```
glDrawElements( GL_TRIANGLES,
elements.size()*sizeof(GLushort), GL_UNSIGNED_SHORT, 0);
```

The indices in the element array buffer provide the topology of the model.

https://en.wikibooks.org/wiki/OpenGL_Programming/Modern_OpenGL_Tutorial_Load_OBJ

More Obj Models

http://goanna.cs.rmit.edu.au/~pknowles/

Topic

Chapter 4. Angel and Shreiner

Model view matrix and projection matrix

Three Basic Elements in Viewing

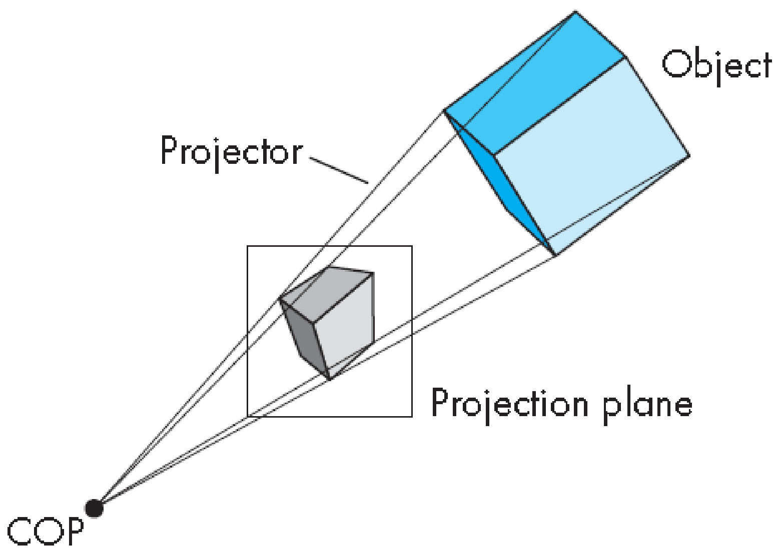
One or more objects

A viewer with a projection surface

- Planar geometric projections
  - standard projections project onto a plane
  - preserve lines but not necessarily angles
- Nonplanar projections are needed for applications such as map construction

Projectors that go from the object(s) to the projection surface

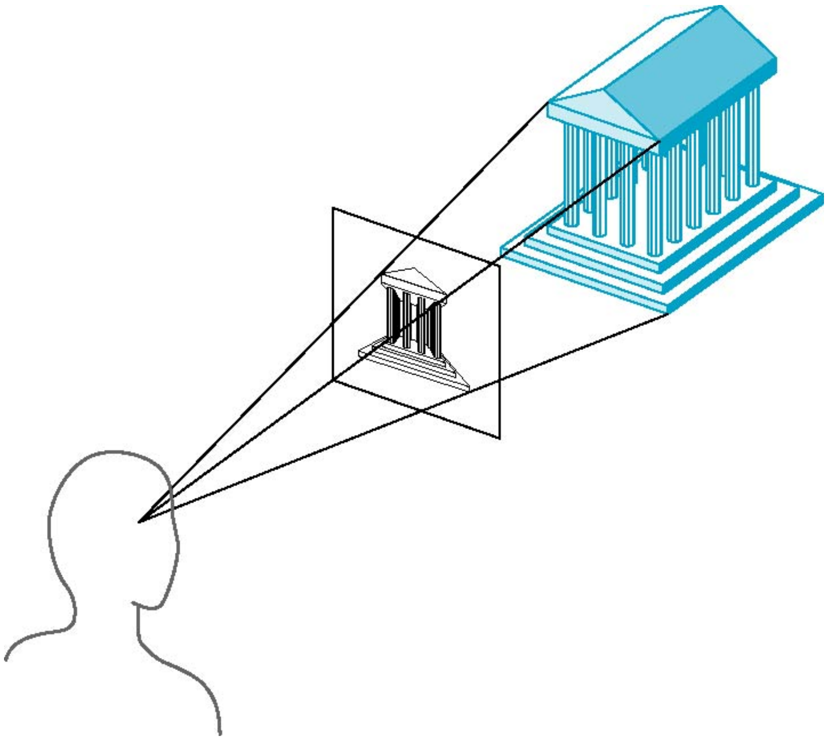
- Projectors are lines that either
  - converge at a center of projection
  - are parallel



E. Angel and D. Shreiner

Perspective Projection

Projectors coverge at center of projection



E. Angel and D. Shreiner

Perspective Projection

Objects further from viewer are projected smaller than the same sized objects closer to the viewer (*diminution*)

- Looks realistic

Equal distances along a line may be not projected into equal distances (*nonuniform foreshortening*)

Angles are preserved only in planes parallel to the projection plane

More difficult to construct by hand than parallel projections (but not more difficult by computer)

Computer Viewing

There are three aspects of the viewing process, all of which are implemented in the pipeline,

- Positioning the camera
  - Setting the model-view matrix
  - Transforming the coordinates in the object frame to the camera frame
- Selecting a lens
  - Setting the projection matrix
  - Attributes of the camera, e.g., focal length, etc
  - Transforming the coordinates in the camera frame to the clip coordinates frame
- Clipping
  - Setting the view volume

Important Transformations in OpenGL

Object (or model) coordinates

World coordinates

Eye (or camera) coordinates

Clip coordinates

Normalized device coordinates

Window (or screen) coordinates

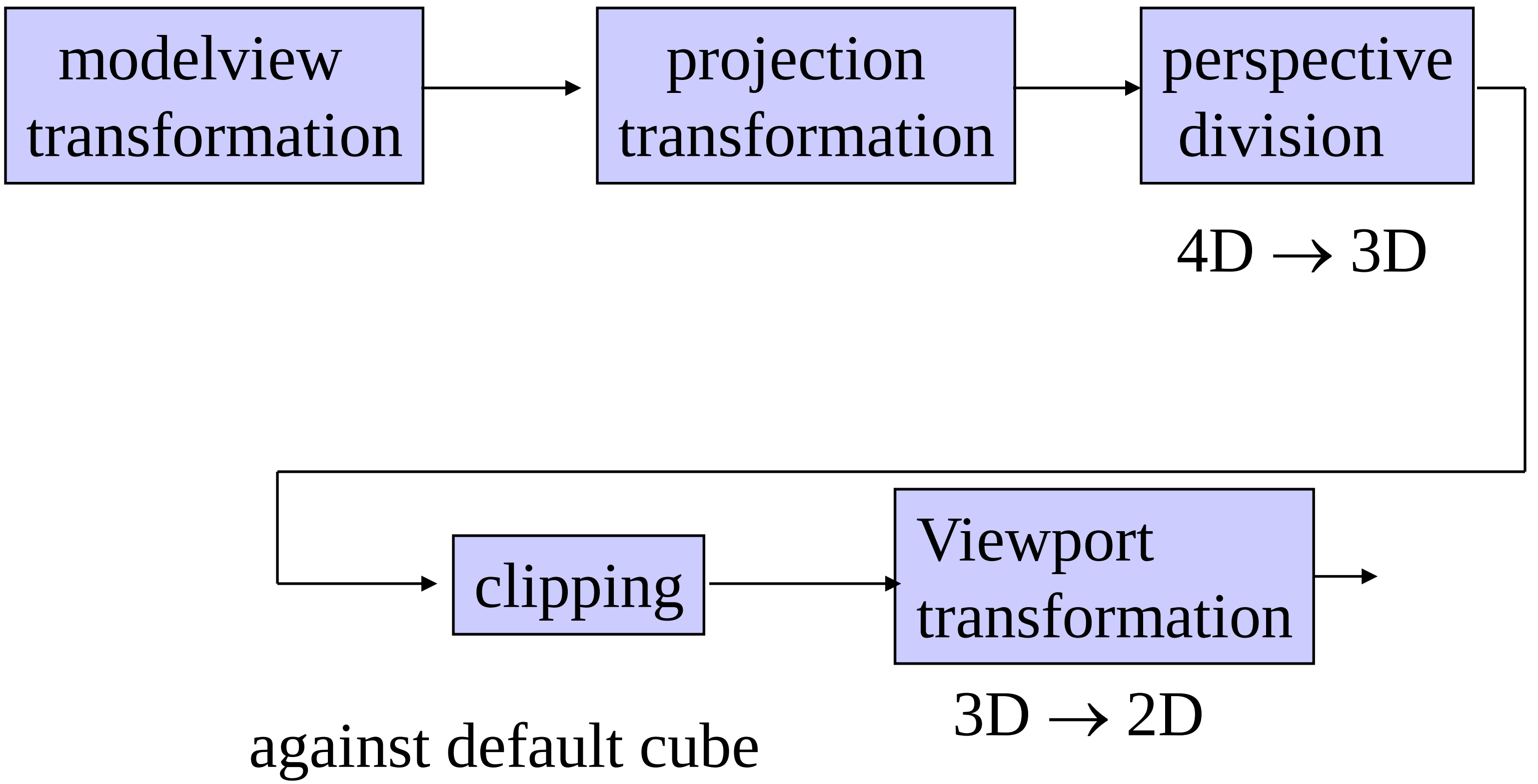
Model-view transformation **4D→4D**

Projection transformation **4D→4D**

Perspective division **4D → 3D**

Viewport transformation
3D→2D+depth

Pipeline View



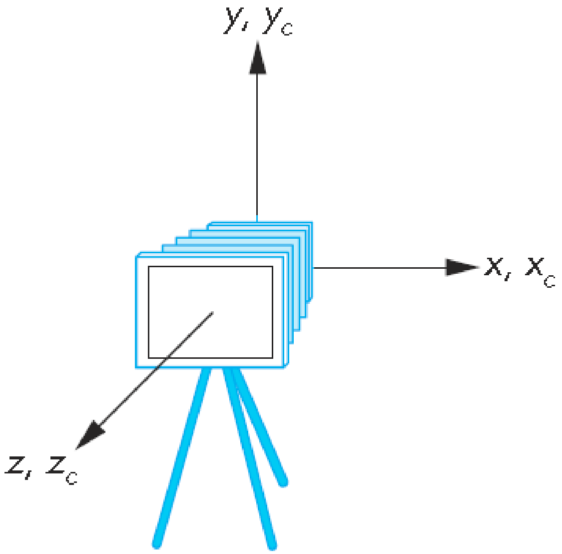
Step1: Derive the Model-view Matrix

In OpenGL, initially the object and camera frames are the same

- Default model-view matrix is an identity

The camera is located at origin and points in the negative z direction

Problem: Cannot see the objects behind the camera -- objects with positive z values



E. Angel and D. Shreiner

Moving the Camera Frame

we can either

- Move the objects along the negative z direction
  - Classical viewing: viewer is fixed
  - Translate the object/world frame
- Move the camera along the positive z direction
  - Camera viewing: objects are fixed
  - Translate the camera frame

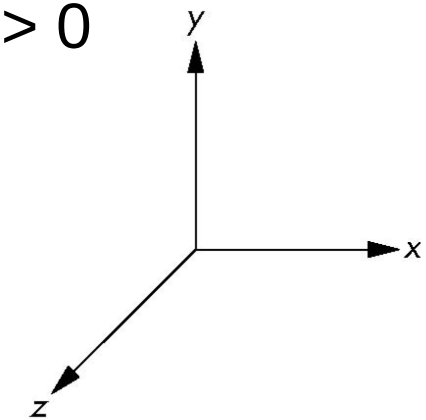
They are equivalent and are determined by the model-view matrix M

- representing a translation (`Translate(0.0,0.0,-d), d > 0`

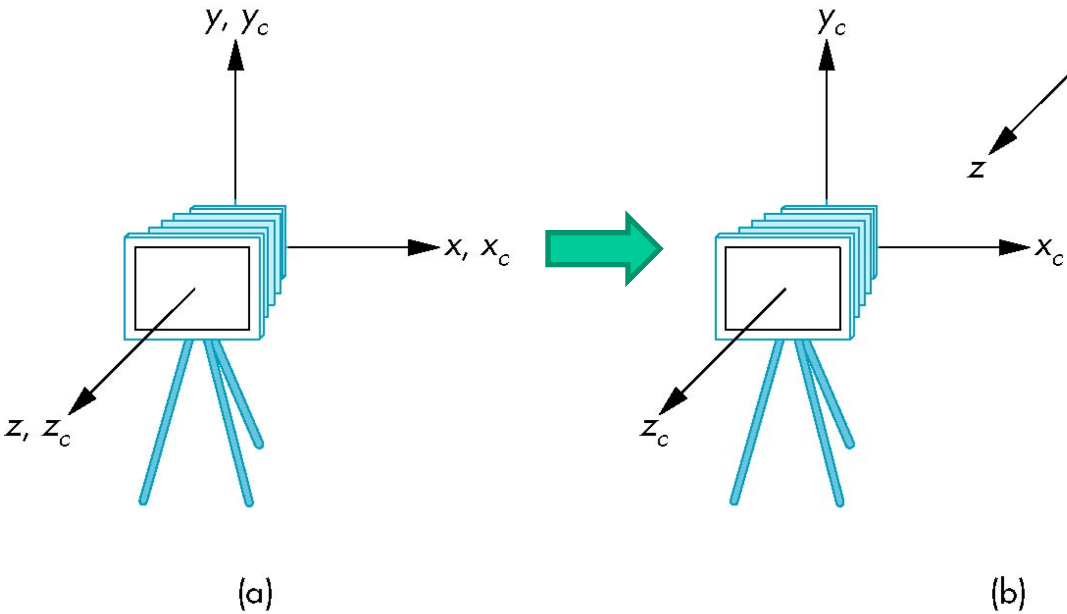
Moving the Camera Frame

frames after translation by –d

d > 0



default frames



$$P_c = \mathbf{M}P$$

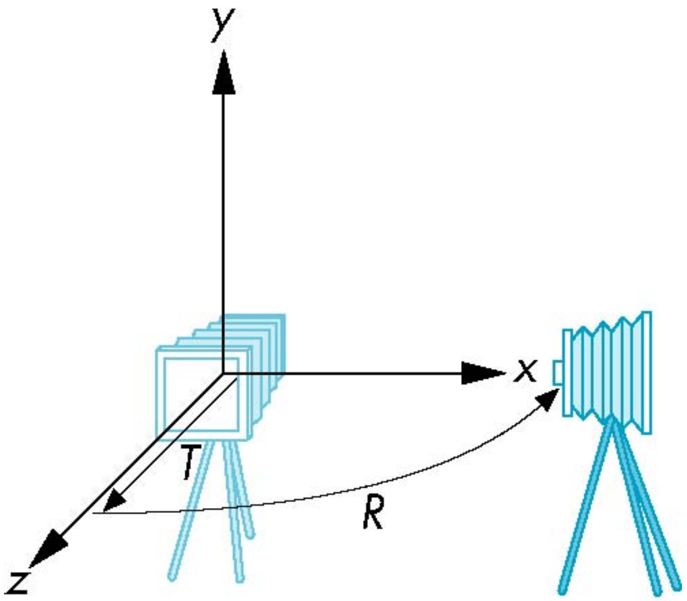
$$\mathbf{M} = \mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Moving the Camera

We can move the camera to any desired position by a sequence of rotations and translations

Example: side view

- Rotate the camera
- Move it away from origin
- Model-view matrix M = TR



E. Angel and D. Shreiner

OpenGL code

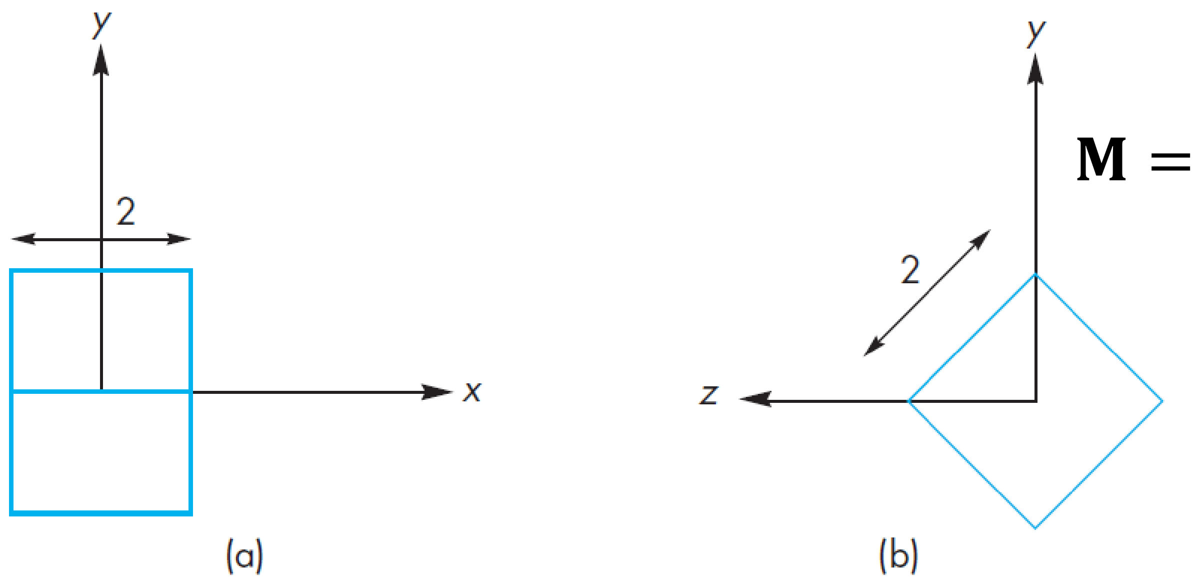
Last transformation specified is first to be applied

```
// Using mat.h

mat4 t = Translate (0.0, 0.0, -d);
mat4 ry = RotateY(90.0);
mat4 m = t*ry;
```

Example: Create an Isometric View of a Cube

Step 1: rotate the cube about the x-axis by 45 degrees – see the other two faces symmetrically



$$\mathbf{M} = \mathbf{T}\mathbf{R}_{\mathrm{y}}\mathbf{R}_{\mathrm{x}}$$

FIGURE 4.15 Cube after rotation about x-axis. (a) View from positive z-axis. (b) View from positive y-axis. E. Angel and D. Shreiner

Step 2: rotate the cube about the y-axis by -35.26 degrees

Step 3: move the camera away from the cube

The LookAt Function

The GLU library contained the function gluLookAt to form the required modelview matrix through a simple interface

Replaced by LookAt() in mat.h

- Can concatenate with modeling transformations

```
mat4 mv = LookAt(vec4 eye, vec4 at, vec4 up);
```

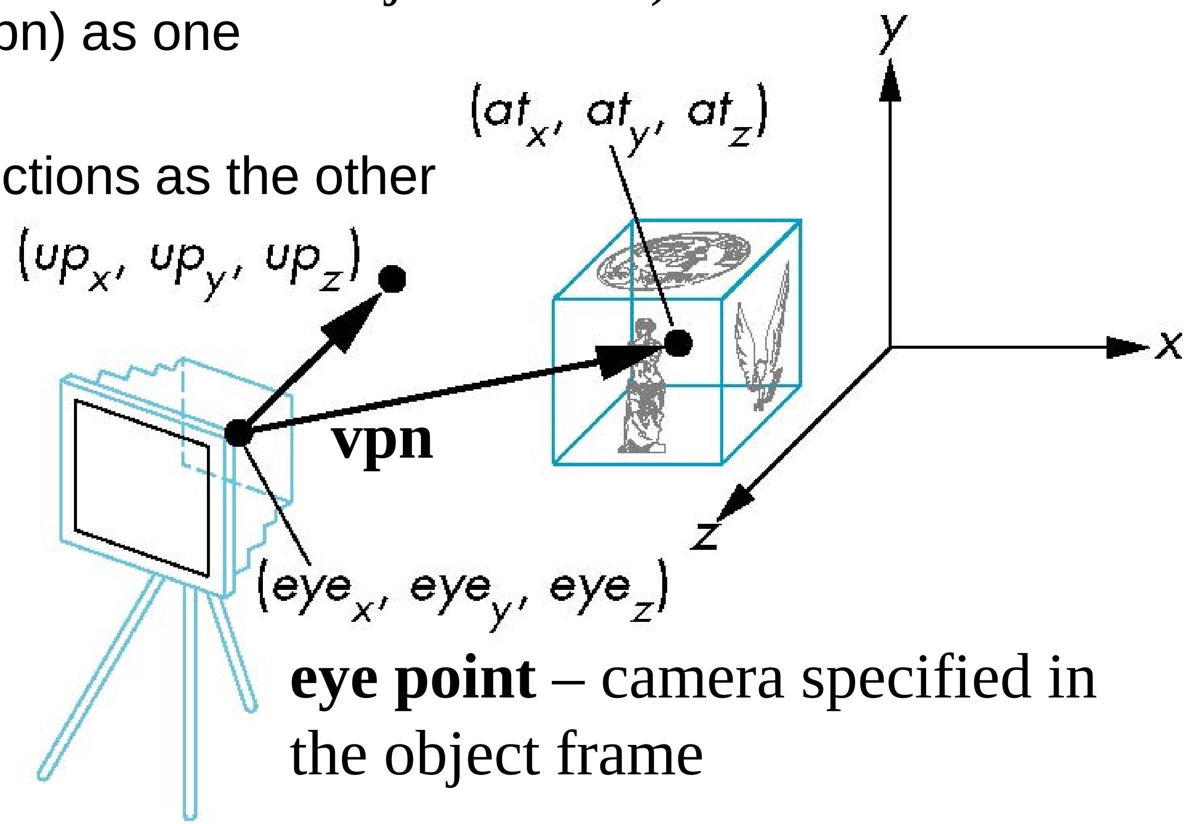
Need to set an up direction

LookAt(eye, at, up)

Objective: construct a new frame with

- the origin at the eye point**,**
- The view plane normal (vpn) as one coordinate direction
- Two other orthogonal directions as the other two coordinate directions

at point – the point (e.g., the object center) the camera looks at



eye point – camera specified in the object frame

LookAt(eye, at, up)

$$\mathbf{vpn} = \mathbf{a} - \mathbf{e},$$

$$\mathbf{n} = \frac{\mathbf{vpn}}{|\mathbf{vpn}|}$$

$$\mathbf{u} = \frac{\mathbf{v}_{\mathbf{up}} \times \mathbf{n}}{|\mathbf{v}_{\mathbf{up}} \times \mathbf{n}|}$$

$$\mathbf{v} = \frac{\mathbf{n} \times \mathbf{u}}{|\mathbf{n} \times \mathbf{u}|}$$

$$\Rightarrow$$

$$\mathbf{M} = \begin{bmatrix} -u_x & -u_y & -u_z & -\mathbf{u} \cdot \mathbf{vpn} \\ v_x & v_y & v_z & \mathbf{v} \cdot \mathbf{vpn} \\ -n_x & -n_y & -n_z & -\mathbf{n} \cdot \mathbf{vpn} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Other Viewing APIs

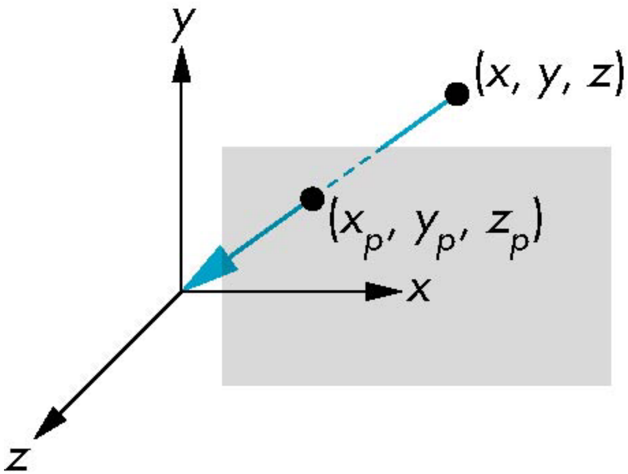
The LookAt function is only one possible API for positioning the camera

Others include

- View reference point, view plane normal, view up (PHIGS, GKS-3D)
- Yaw, pitch, roll
- Elevation, azimuth, twist
- Direction angles

Step2: Projections and Normalization

The default projection is **orthogonal (orthographic) projection**



E. Angel and D. Shreiner

For points within the view volume

$$x_p = x$$
$$y_p = y$$
$$z_p = 0$$

In homogeneous coordinates

$$\mathbf{p}_p = \mathbf{M}_{orth}\mathbf{p}$$

$$\mathbf{M}_{orth} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

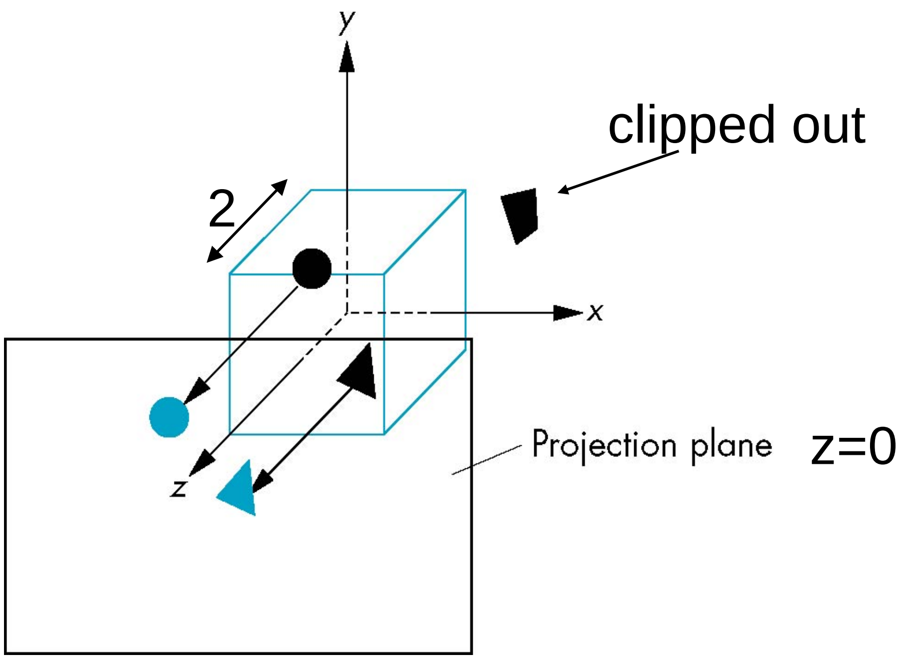
Orthogonal Normalization

Default projection:

- the orthographic camera is at origin
- the default volume is enclosed in

$$x = \pm 1, y = \pm 1, z = \pm 1$$

`Ortho(-1.0,1.0,-1.0,1.0,-1.0,1.0)`



A general orthogonal projection

`Ortho(left,right,bottom,top,near,far)`

The clipping volume is different than the default

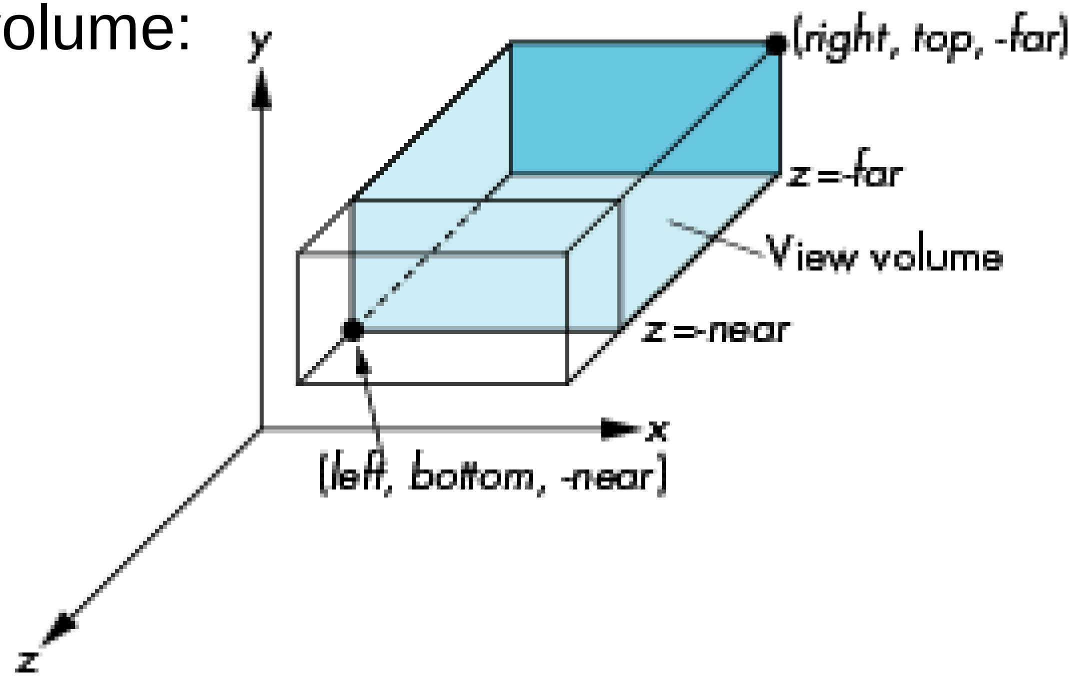
OpenGL Orthogonal Viewing

`Ortho(left,right,bottom,top,near,far)`

The 4 sides of clipping volume:

$$x = right$$
$$x = left$$
$$y = top$$
$$y = bottom$$

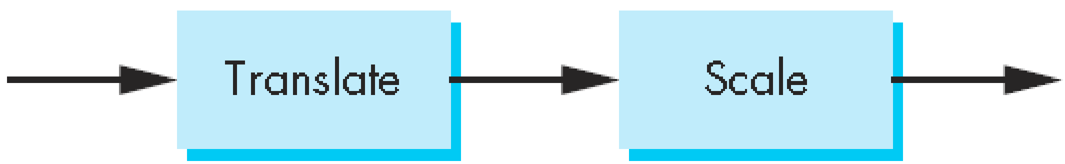
Form a projection matrix

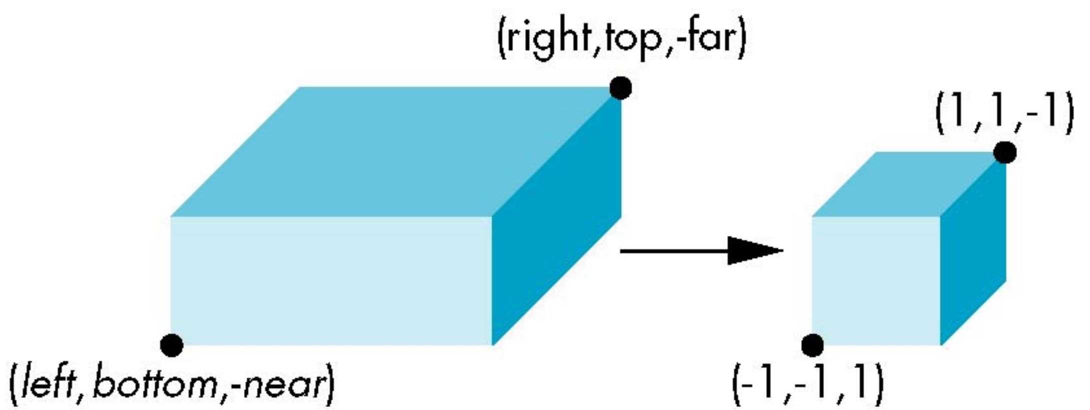


All are measured in the *camera frame*

Orthogonal Normalization

Find transformation to convert specified clipping volume to default





Affine Transformation for Orthogonal Normalization

Two steps:

- Move center to origin

$$\mathbf{T}(-(left + right)/2, -(bottom + top)/2, (near + far)/2))$$

- Scale to have sides of length 2

$$\boldsymbol{S}(2/(left - right), 2/(top - bottom), 2/(near - far))$$

$$\mathbf{N} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right - left} & 0 & 0 & -\frac{right - left}{right - left} \\ 0 & \frac{2}{top - bottom} & 0 & -\frac{top + bottom}{top - bottom} \\ 0 & 0 & \frac{2}{near - far} & \frac{far + near}{far - near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Final Orthographic Projection Matrix

General orthogonal projection in 4D is

$$\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{ST}$$