Projections and Viewing

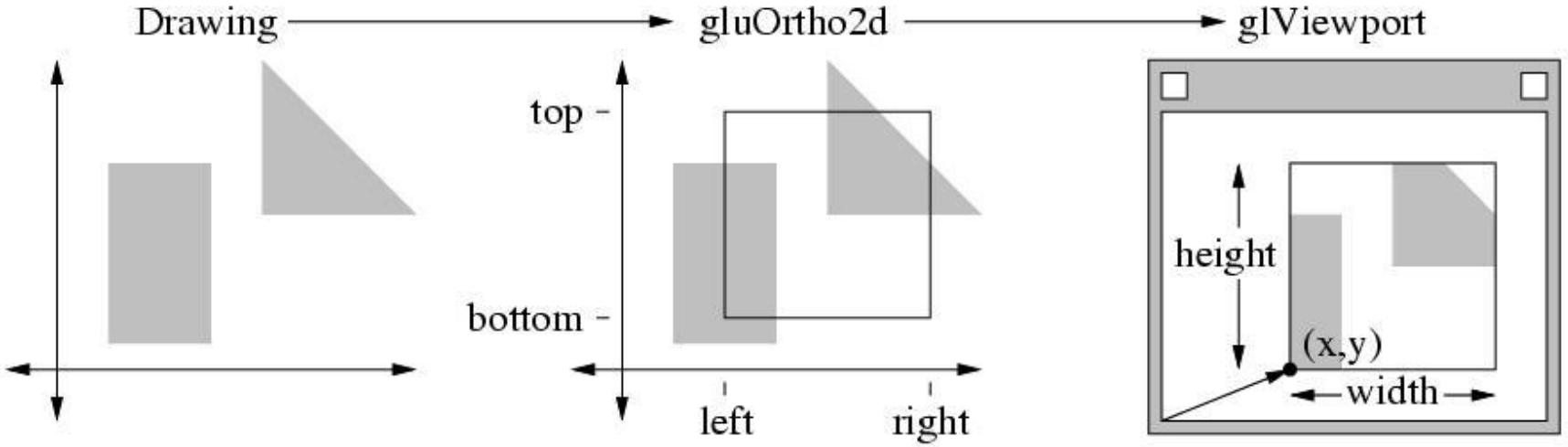
Computer Graphics
Lab 06

Contents

In this section we will learn how to render a 3D model.

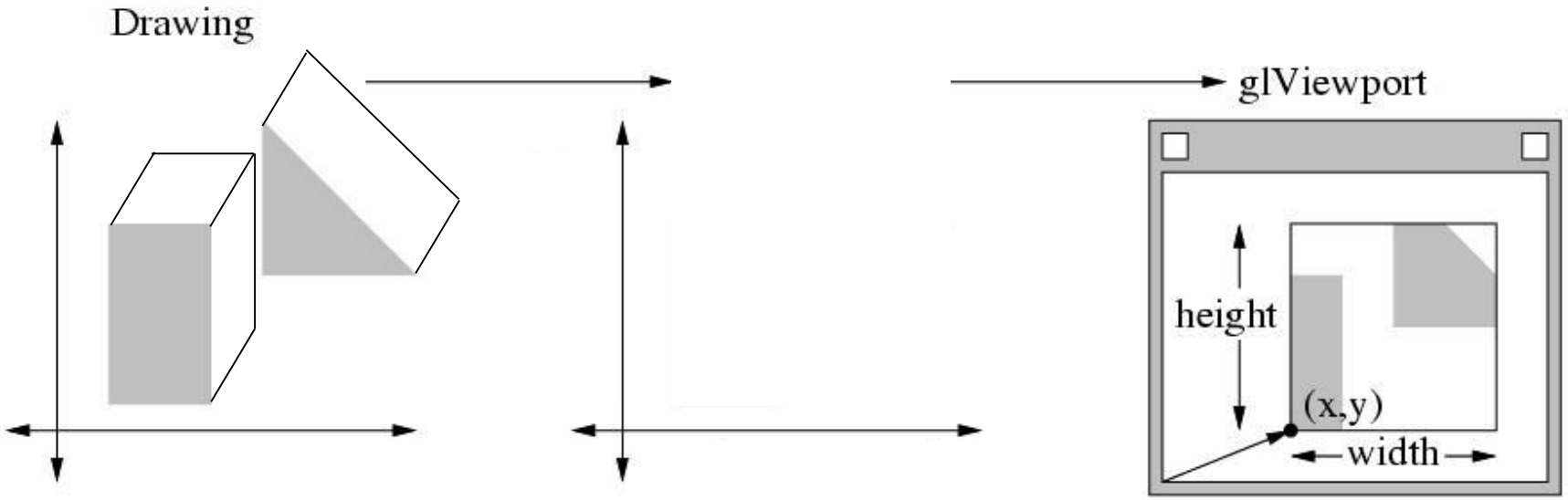
- Defining projection.
  - The 3D orthographic projection.
  - The 3D Perspective projection.
- Viewing transformations.
- Lab exercise.

2D world display process



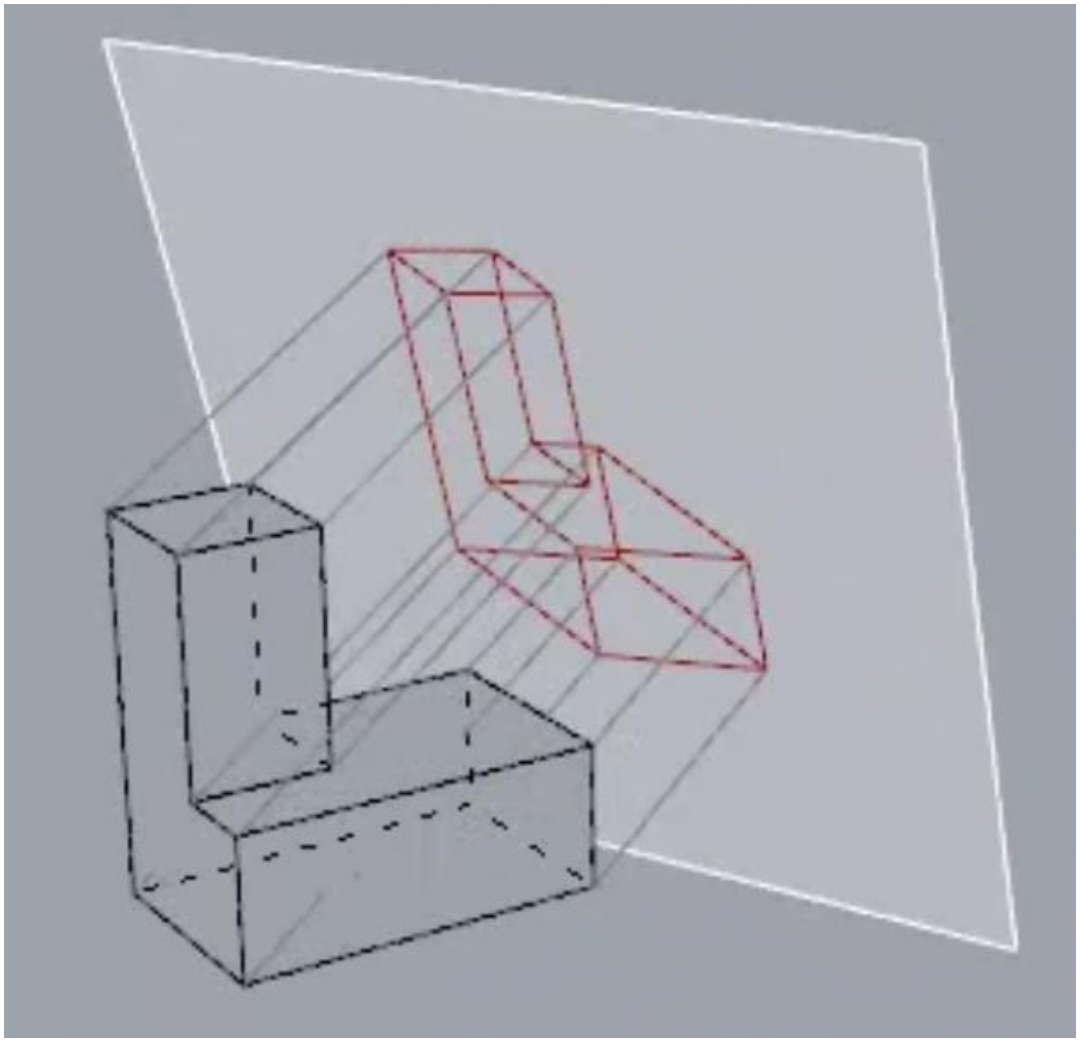
(As we allready saw)

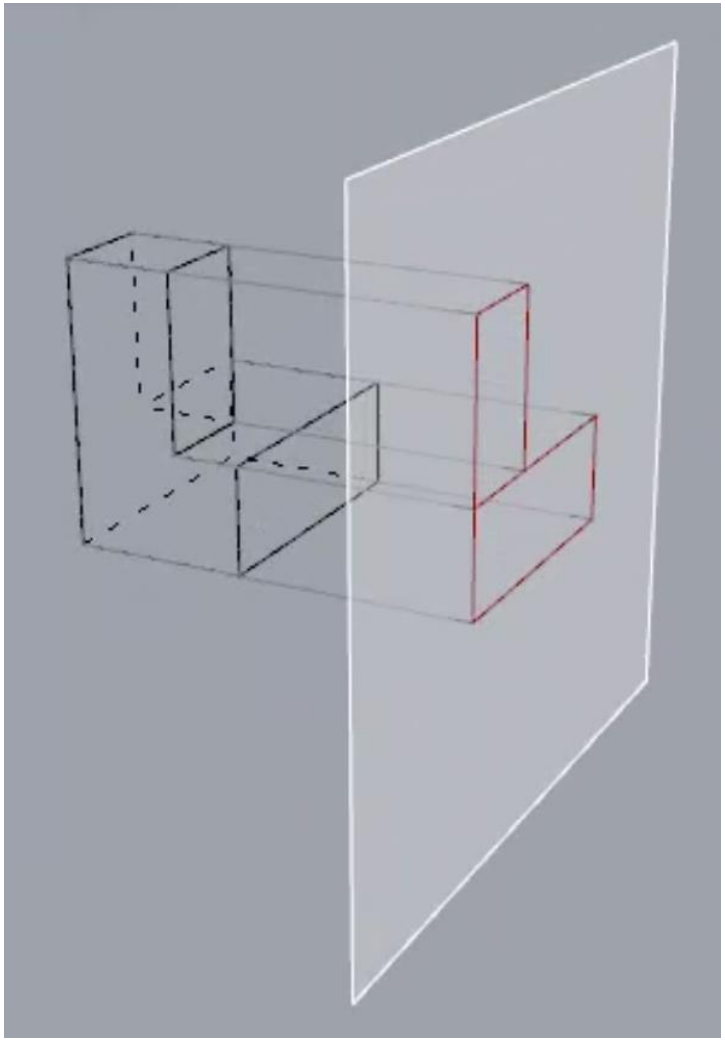
How to project 3D model on 2D?



We need to project the 3D world to 2D and bound it.

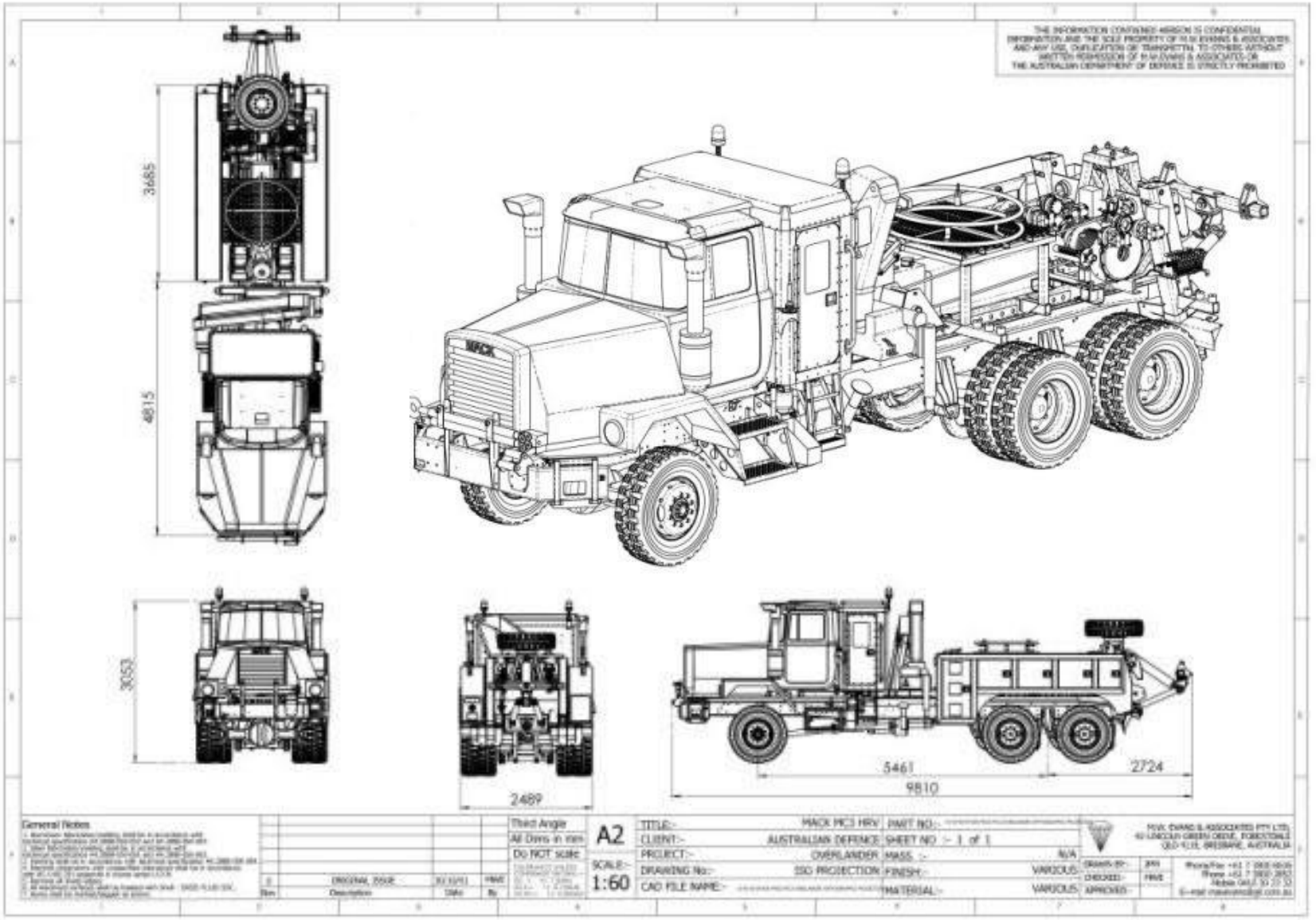
Orthographic Projection





Source: http://studiomaven.org/index.php?title=Context:Overview_of_Architectural_Drawings

Orthographic Projection



Orthographic Projection

Defining projection

We are already familiar with a 2D world definition. However, we live in a 3D world but the screen where the rendered objects appear is a 2D space. Thus, we need to convert somehow our 3D world to 2D.

The operation which does this conversion is called projection.

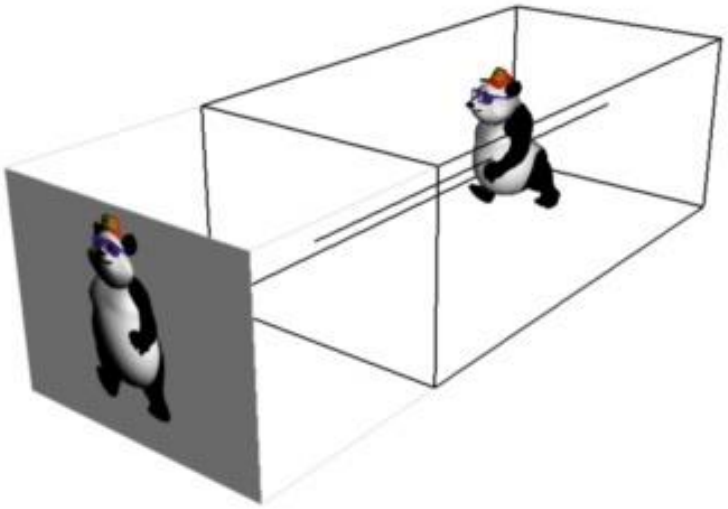
The GLU command that defines a 2D projection is **gluOrtho2D(…).** Using this command we define a bounding rectangle.

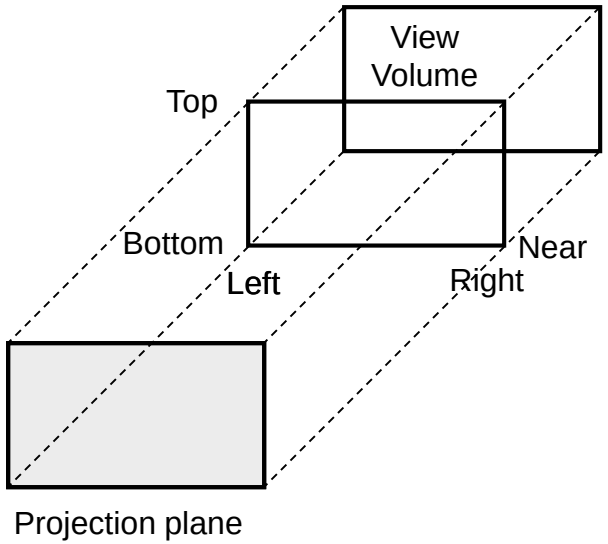
The extension to 3D would be a bounding box, which is defined with the more general GL command **glOrtho(…)**.

Projection Transformations - glOrtho

The following command multiplies the current matrix by an orthographic matrix:

```
void glOrtho(  double left, double right,
               double bottom, double top,
               double zNear, double zFar )
```





Projection Transformations – glOrtho

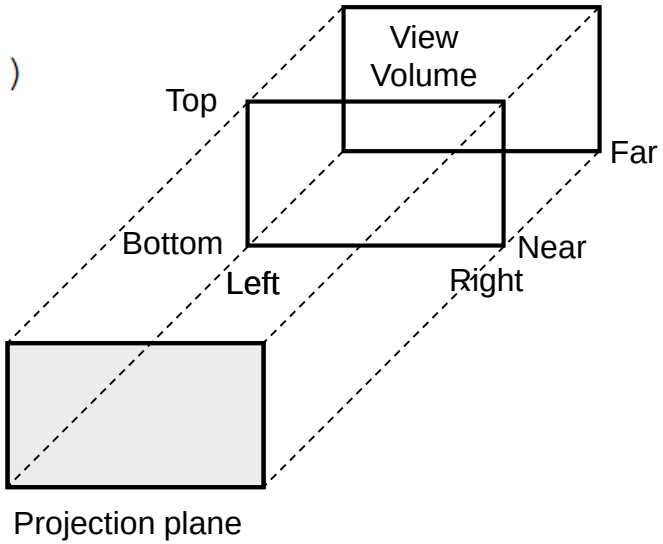
The following command multiplies the current matrix by an orthographic matrix:

```
void glOrtho( double left, double right,
              double bottom, double top,
              double zNear, double zFar )
```

left, *right* – The coordinates for the left-and right-vertical clipping planes.

bottom, *top* – The coordinates for the bottom- and top-horizontal clipping planes.

zNear, *zFar* – The distances to the nearer and farther depth clipping planes



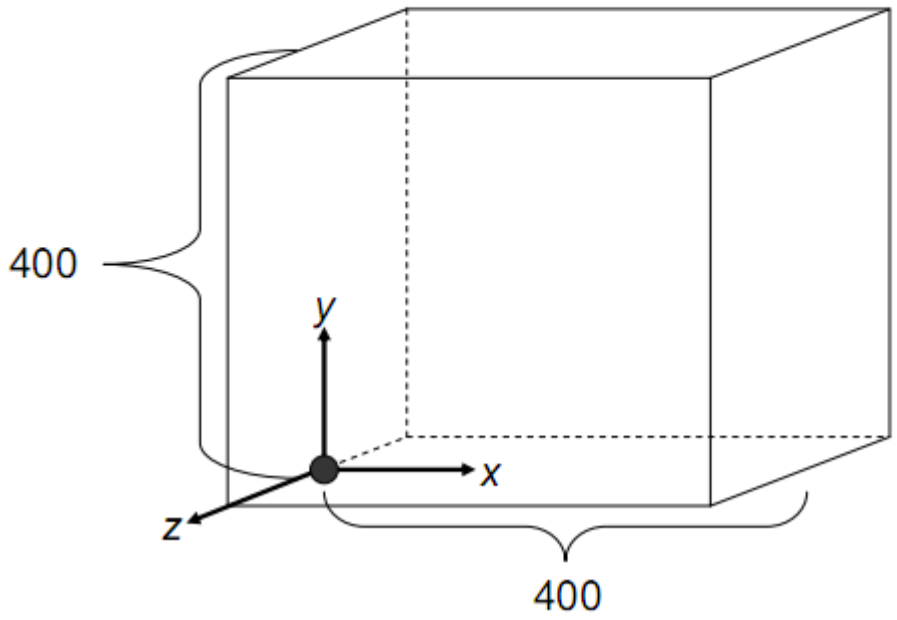
Note:

The distances to the *near* or *far* plane is negative if the plane is behind the viewer.

Nate Robins computer graphics demos: https://user.xmission.com/~nate/tutors.html

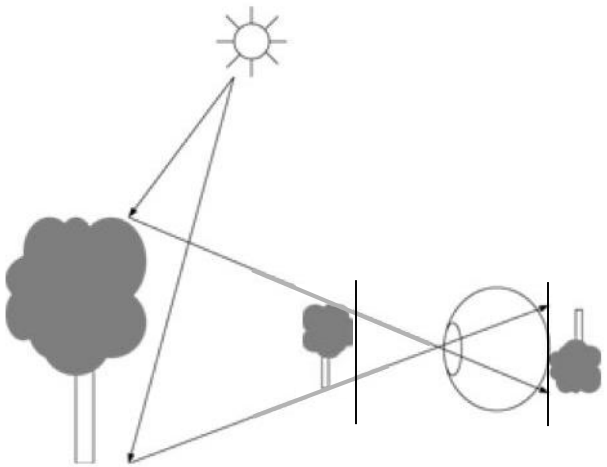
glOrtho example

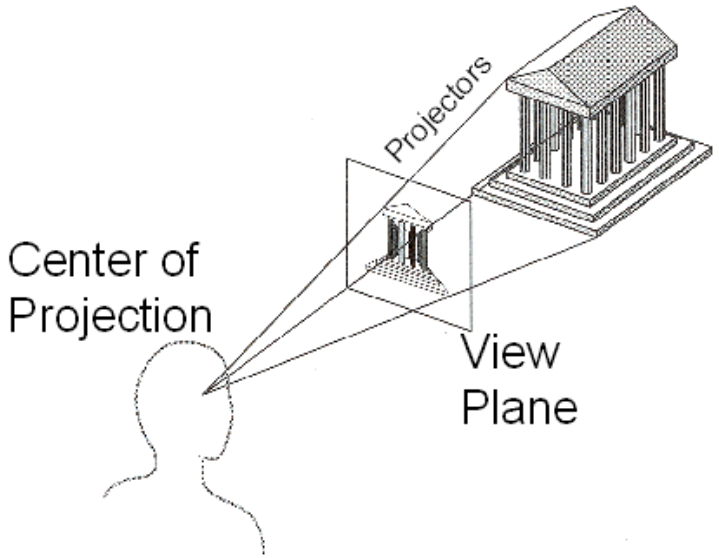
```
glOrtho(0.0, 400.0, 0.0, 400.0, -100.0, 100.0);
```



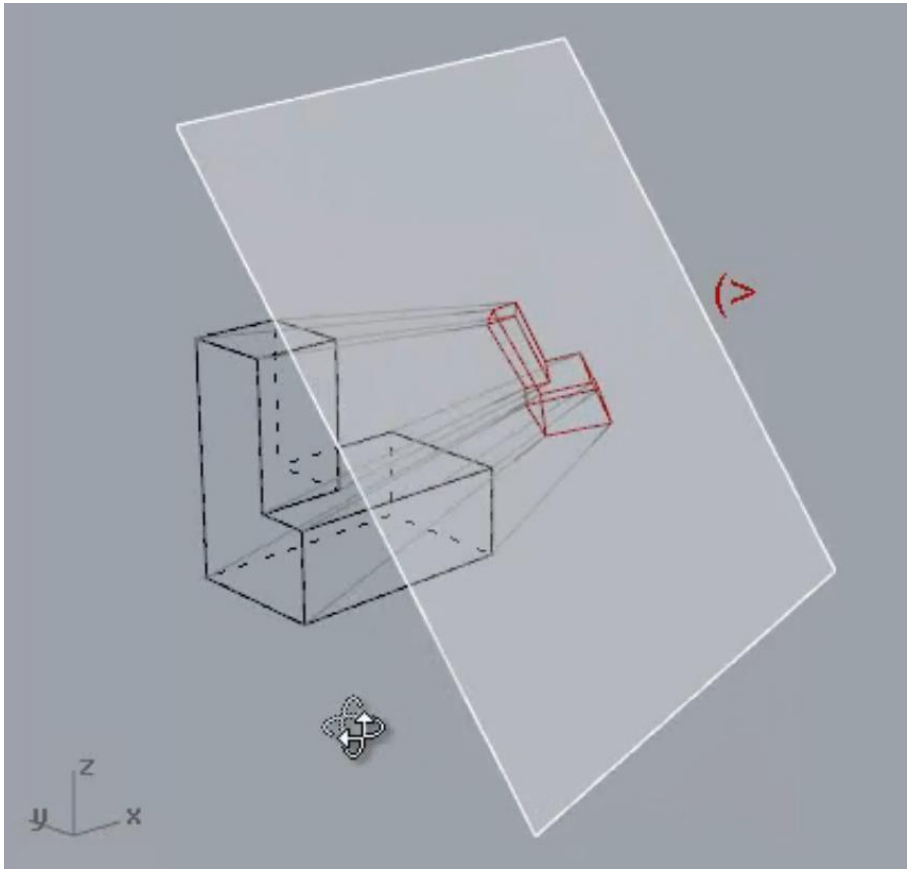
Perspective Projection

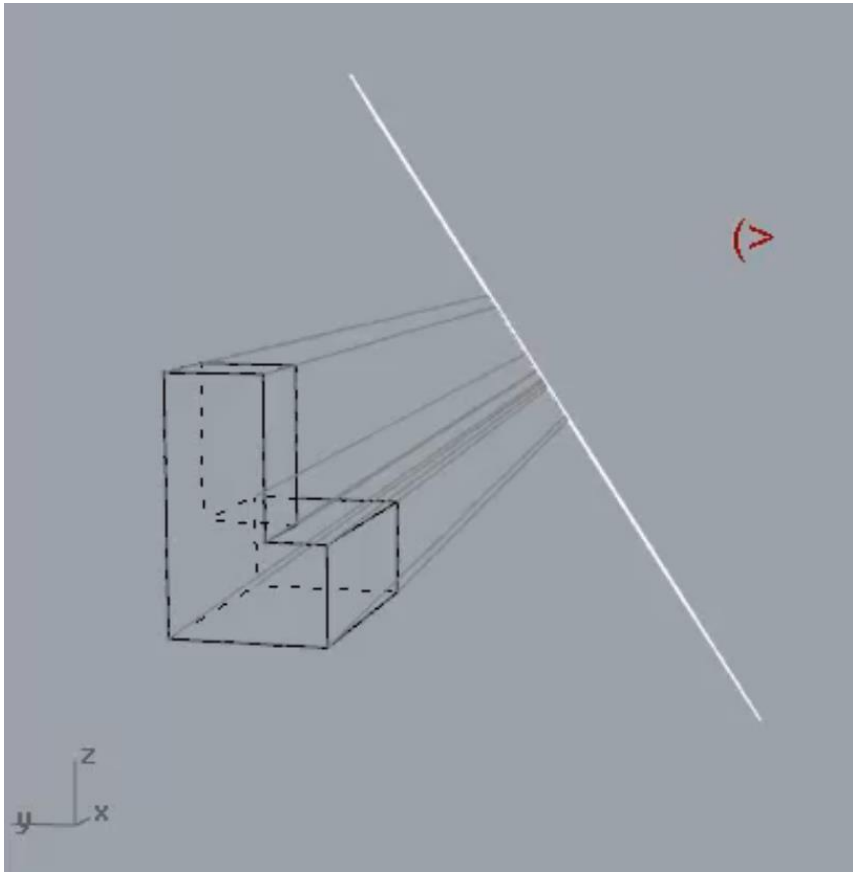
Perspective is the image as it is seen by the eye. Perspective works by representing the light that passes from a scene through an imaginary rectangle (the painting), to the viewer's eye





Orthographic Projection

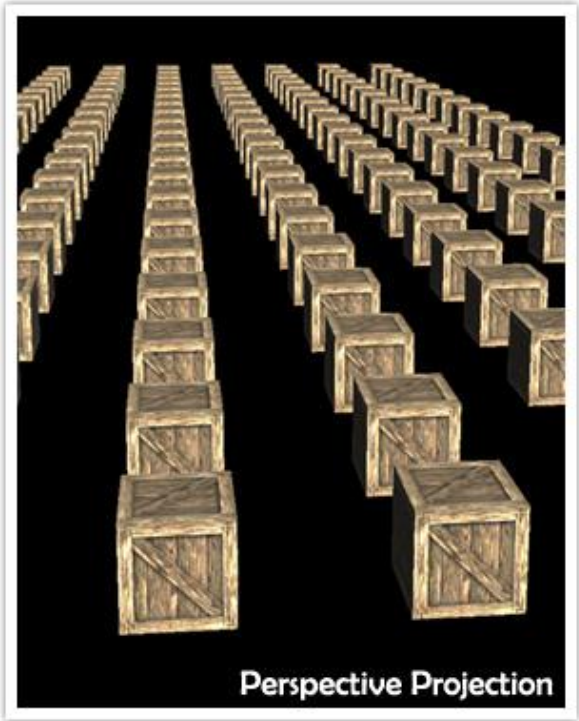


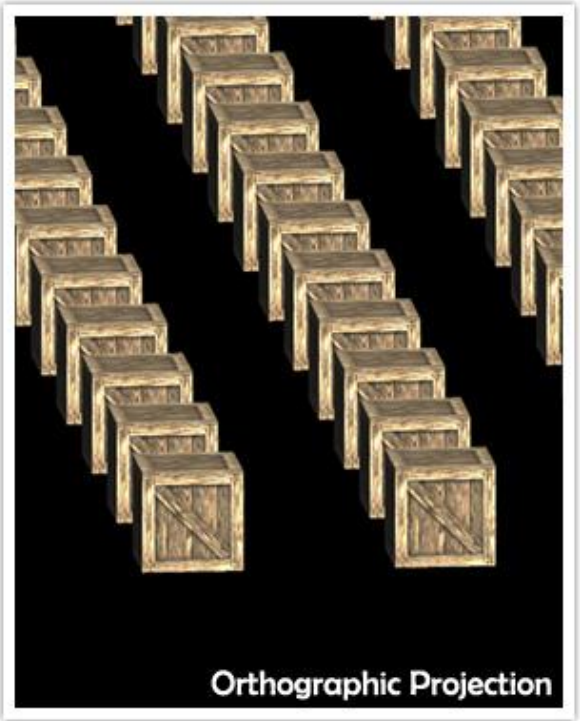


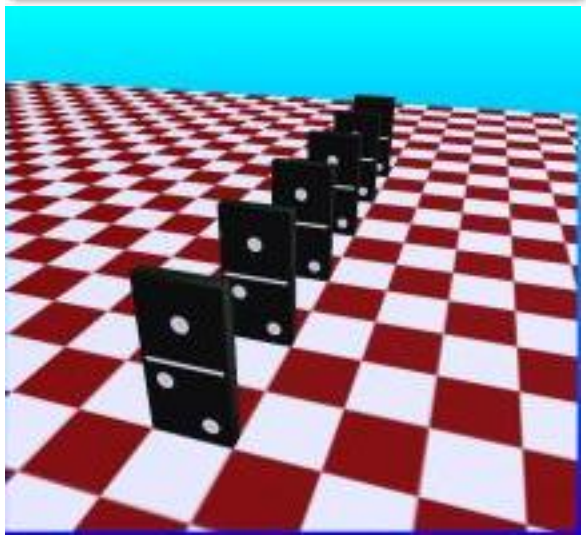
Source: http://studiomaven.org/index.php?title=Context:Overview_of_Architectural_Drawings

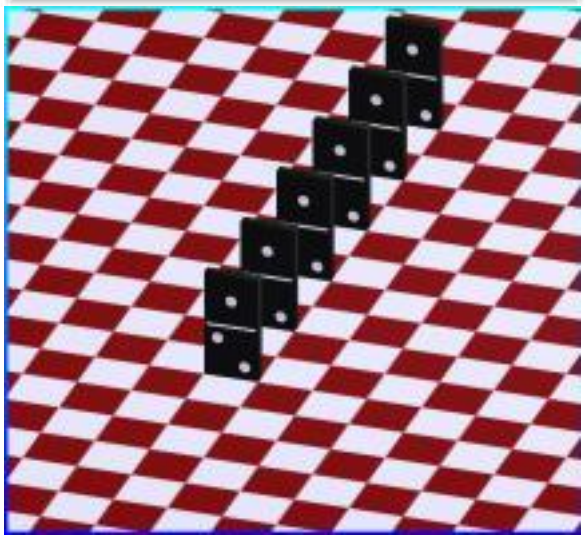
Perspective vs Orthographic

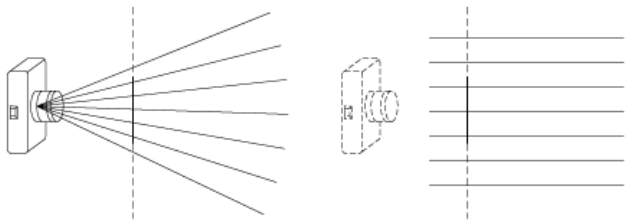
- In Perspective objects are smaller as their distance from the observer increases.
- In Perspective parallel lines seems to meet far away.
- In perspective projection lines meet in the center of projection. In Orthographic they are parallel.











Projection Transformations - Perspective (1)

```
void glFrustum(double left, double right, double bottom,
               double top, double near, double far );
```

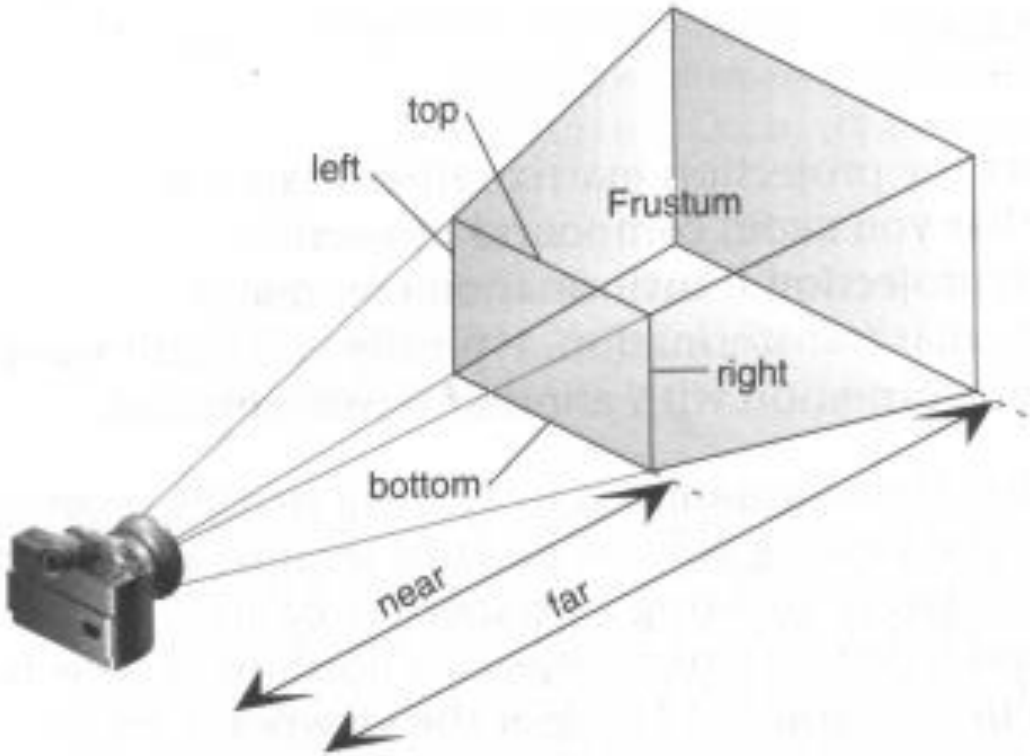
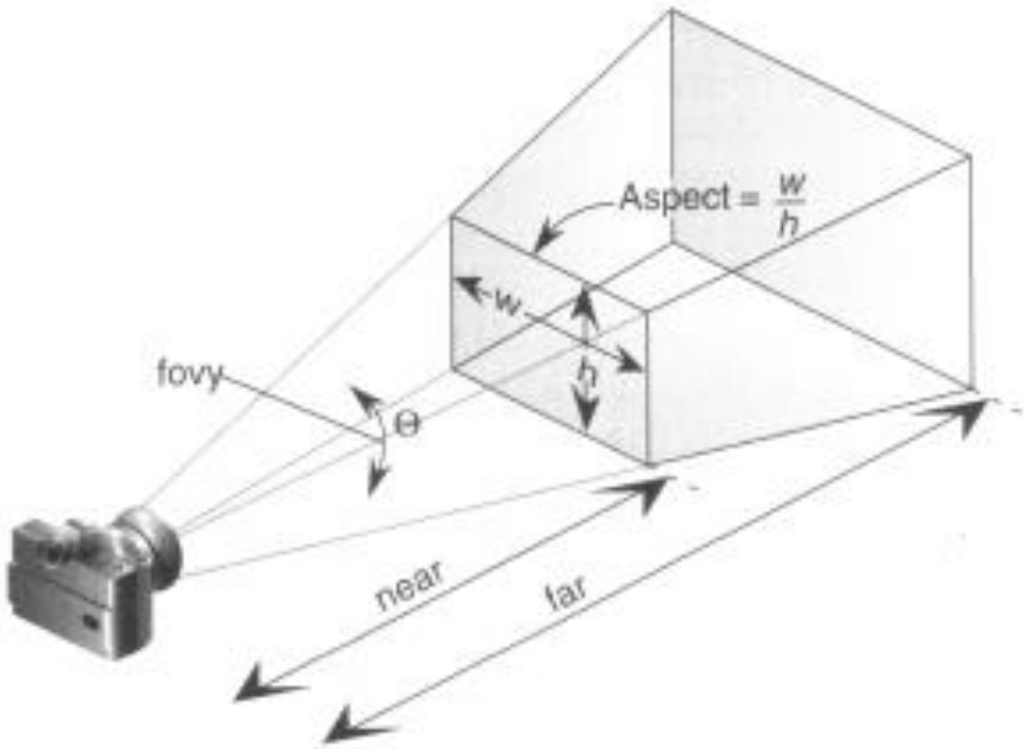


Figure 3-13 Perspective Viewing Volume Specified by glFrustum()

Projection Transformations – Perspective (2)

```
void gluPerspective( double fovy, double aspect,
                     double near, double far );
```



Nate Robins computer graphics demos: https://user.xmission.com/~nate/tutors.html

Projection Transformations – Perspective (contd.)

```
void glFrustum(double left, double right, double bottom,
               double top, double znear, double zfar );
```

left, right – The coordinates for the left- and right-vertical clipping planes.
bottom, top – The coordinates for the bottom- and top-horizontal clipping planes.
znear, zfar – The distances to the near- and far-depth clipping planes.

```
void gluPerspective( double fovy, double aspect,
                     double near, double far );
```

fovy – Specifies the field of view angle, in degrees, in the y direction.
aspect – Specifies the aspect ratio that determines the field of view in the x direction = x/y.
near – Specifies the distance from the viewer to the near clipping plane.
far – Specifies the distance from the viewer to the far clipping plane.

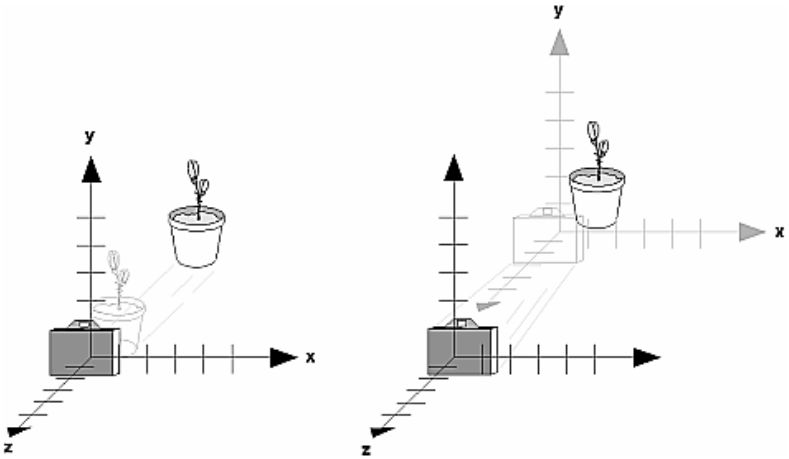
Note:

Distances from the viewer to near and far clipping planes must be always positive and non-zero!

A mistake many programmers make is to specify a *zNear* clipping plane value of 0.0 or a negative value which isn't allowed

Viewing transformation

- We have two options:
  - Moving the view point
  - Moving the object and keep the view point fixed



Viewing transformation

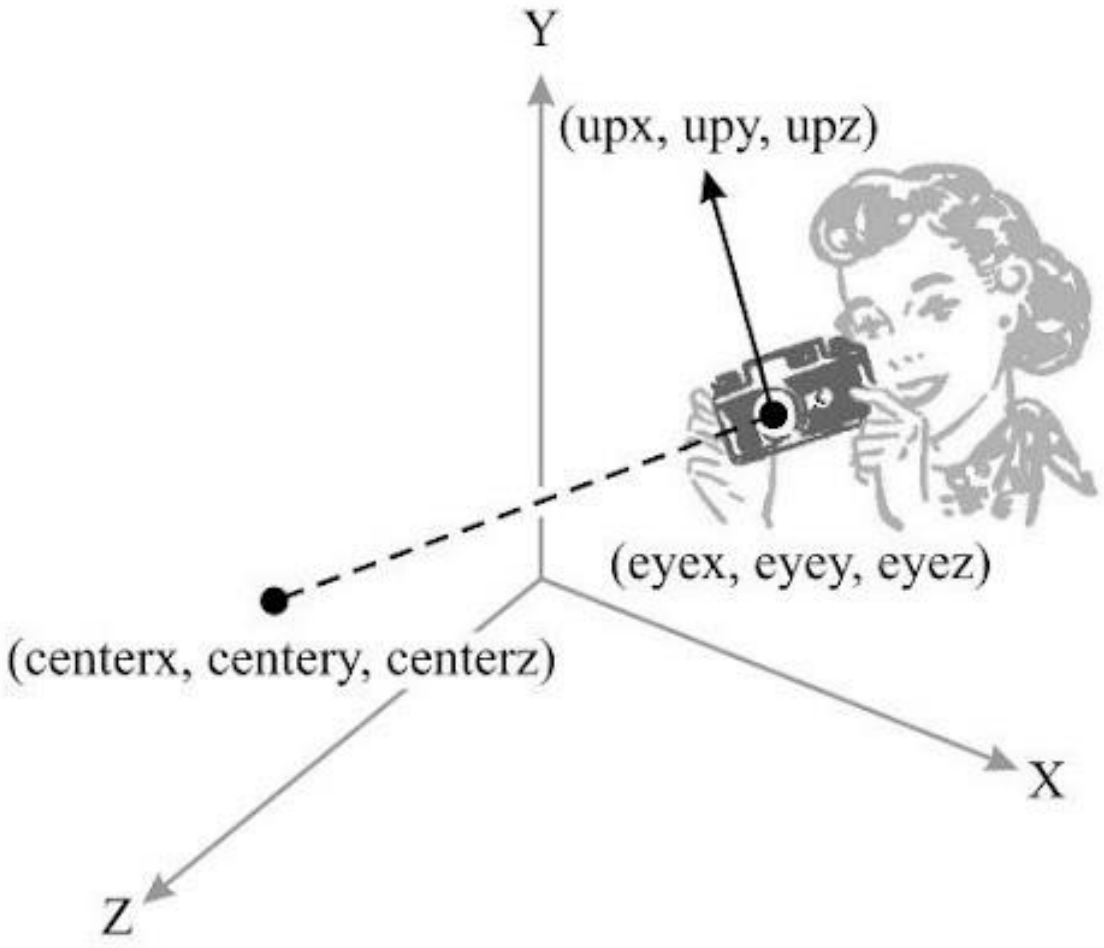
```
void gluLookAt(double eyex, double eyey, double eyez,
               double centerx, double centery, double centerz,
               double upx, double upy, double upz );
```

eyex, eyey, eyez – The position of the eye point.
centerx, centery, centerz – The position of the reference point.
upx, upy, upz – The direction of the up vector.

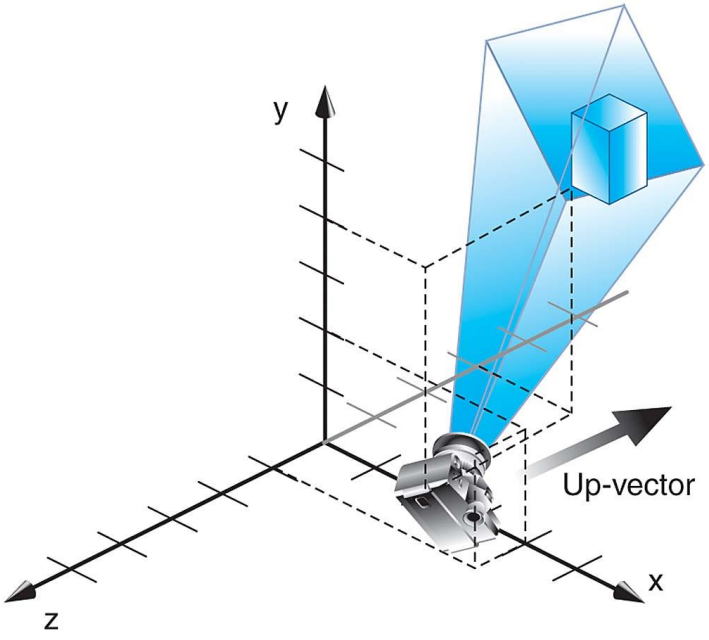
Notes:

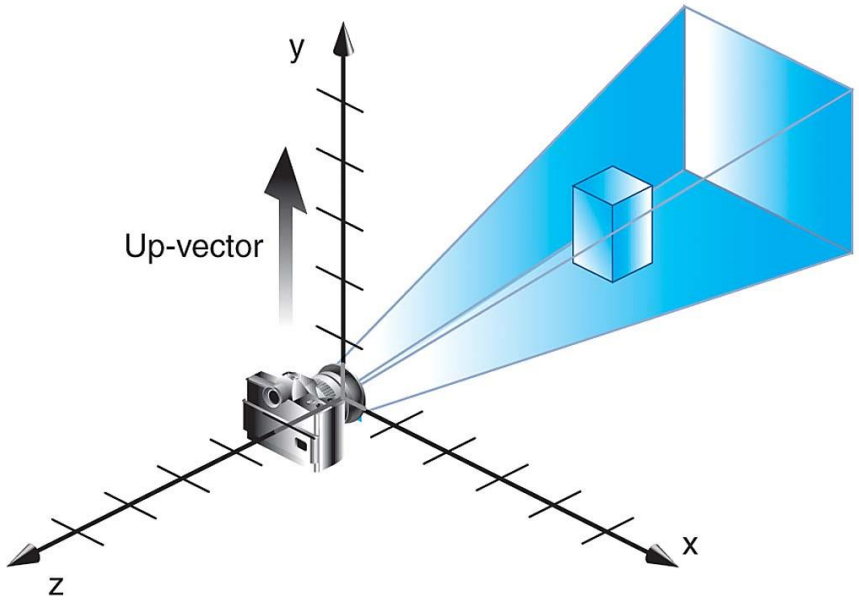
- The command **gluLookAt** (…) is actually a compound of basic transformations (translation, rotation, scaling). On which stack should it operate? (PROJECTIO or MODELVIEW)
- The default eye position is at the origin.
- The default eye's direction is down the negative Z axis.

Viewing transformation



Viewing transformation



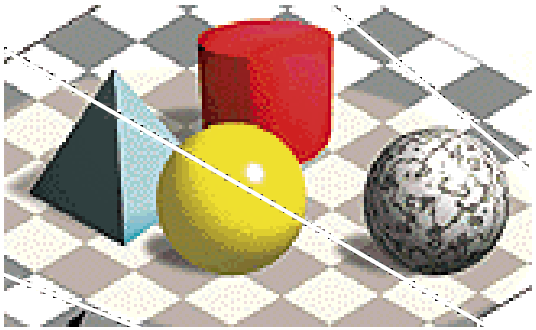


Viewing transformation

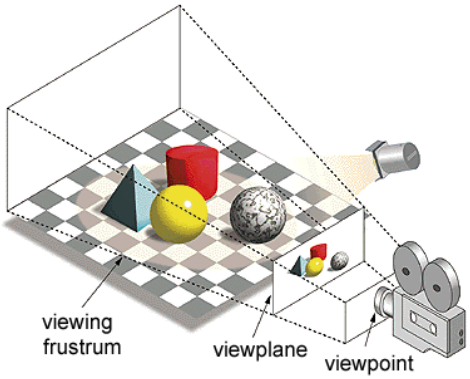
- Eye position is always at the origin, and eye's direction is down the negative Z axis.
- gluLookAt is equivalent to moving the axis to be in front of the eye, using transformations.
- Therefore you should call gluLookAt before drawing any object.
- Call gluLookAt after choosing ModelView matrix and loading identity.

Graphics pipeline 2D

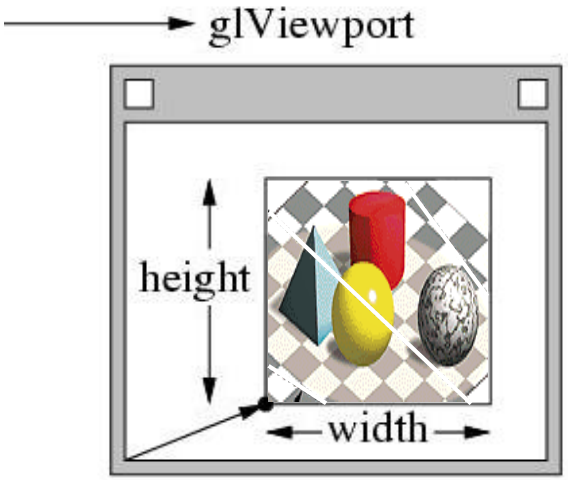
Model

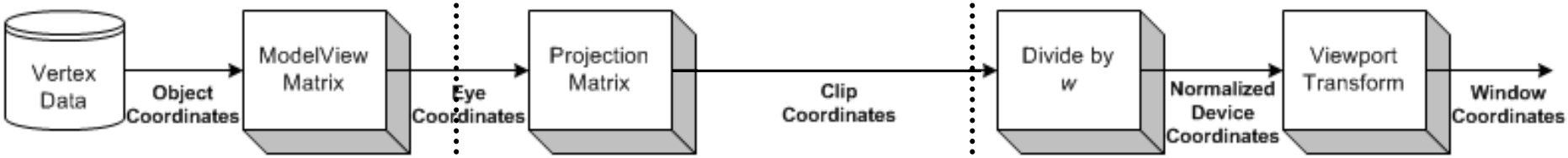


projection



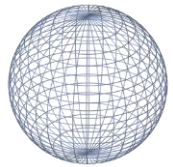
Location on window

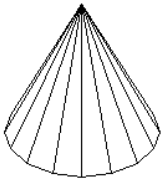


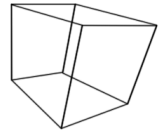


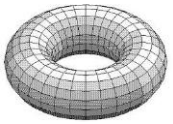
Some 3D built-in OpenGL functions

- void **glutWireSphere**(GLdouble radius, GLint slices, GLint stacks);
- void **glutSolidSphere**(GLdouble radius, GLint slices, GLint stacks);
- void **glutWireCone**(GLdouble base, GLdouble height, GLint slices, GLint stacks);
- void **glutSolidCone**(GLdouble base, GLdouble height, GLint slices, GLint stacks);
- void **glutWireCube**(GLdouble size);
- void **glutSolidCube**(GLdouble size);
- void **glutWireTorus**(Gldouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);
- void **glutSolidTorus**(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);



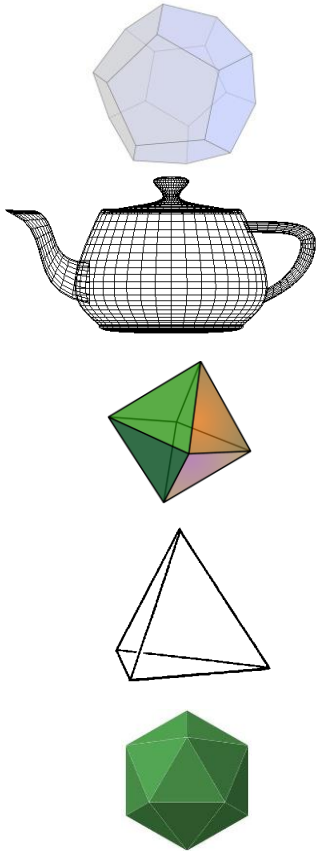






Some more 3D OpenGL functions

- void glutWireDodecahedron(void);
- void glutSolidDodecahedron(void);
- void glutWireTeapot(GLdouble size);
- void glutSolidTeapot(GLdouble size);
- void glutWireOctahedron(void);
- void glutSolidOctahedron(void);
- void glutWireTetrahedron(void);
- void glutSolidTetrahedron(void);
- void glutWireIcosahedron(void);
- void glutSolidIcosahedron(void);



תרגיל

1. ציירו צורה תלת מימדית פשוטה והציגו אותה:
   (הפעם בצעו את כל השלבים ביחד ורק אז בידקו את התוצאה)
   1. ציירו צורה אחת בראשית הצירים. (בחרו ב-wireframe)
   2. שנו את נקודת המבט, כך שהמבט יהיה מ-(0,0,5) אל עבר הראשית.
   3. בצעו הטלה באמצעות ortho. חישבו מה מיקום המצלמה, מה מיקום האוביקט ואיפה אתם רוצים למקם את התיבה של ההטלה ortho.
2. הוסיפו צורה תלת מימדית נוספת ליד הצורה שציירתם.
3. הוסיפו סיבוב של הגופים:
   1. השתמשו בכפתורים במקלדת על מנת לסובב את הצורות סביב ציר ה-x. כפתור לסיבוב עם כיוון השעון וכפתור לסיבוב נגד כיוון השעון. (סיבוב הגופים ולא שינוי נקודת המבט)
   2. הוסיפו סיבוב נוסף סביב ציר y (עם כפתורים נוספים).
   3. הוסיפו סיבוב נוסף סביב ציר z.