



1
XIAMEN
UNIVERSITY

COMPUTER GRAPHICS

Viewing Transformation

Dr. Zhonggui Chen

School of Informatics, Xiamen University

<http://graphics.xmu.edu.cn>

Modeling Transformations

- Translation

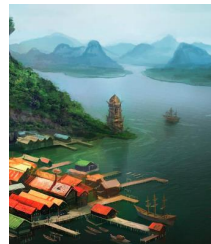
 - `glTranslatef(p , q , r)`

- Scaling

 - `glScalef(u , v , w)`

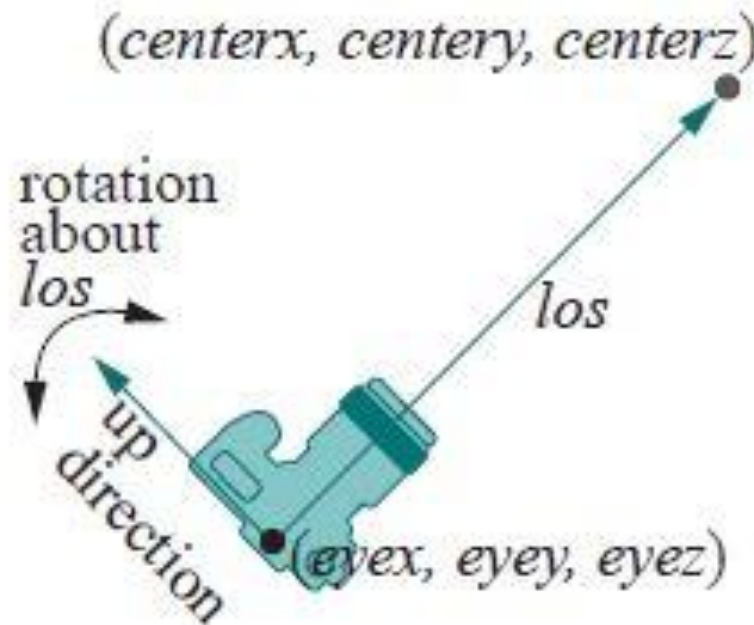
- Rotation

 - `glRotatef(A , p , q , r)`



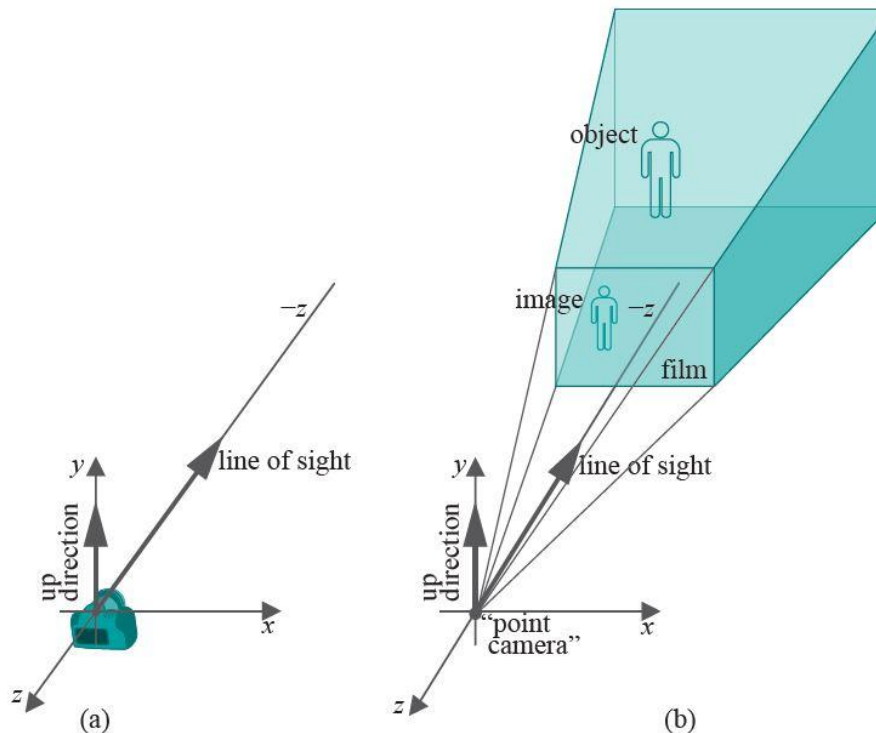
Viewing Transformaiton

- `gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)`
 - ▣ camera location = $(eyex, eyey, eyez)$
 - ▣ line of sight (*los*) = $(centerx, centery, centerz) - (eyex, eyey, eyez)$
 - ▣ up direction = (upx, upy, upz)



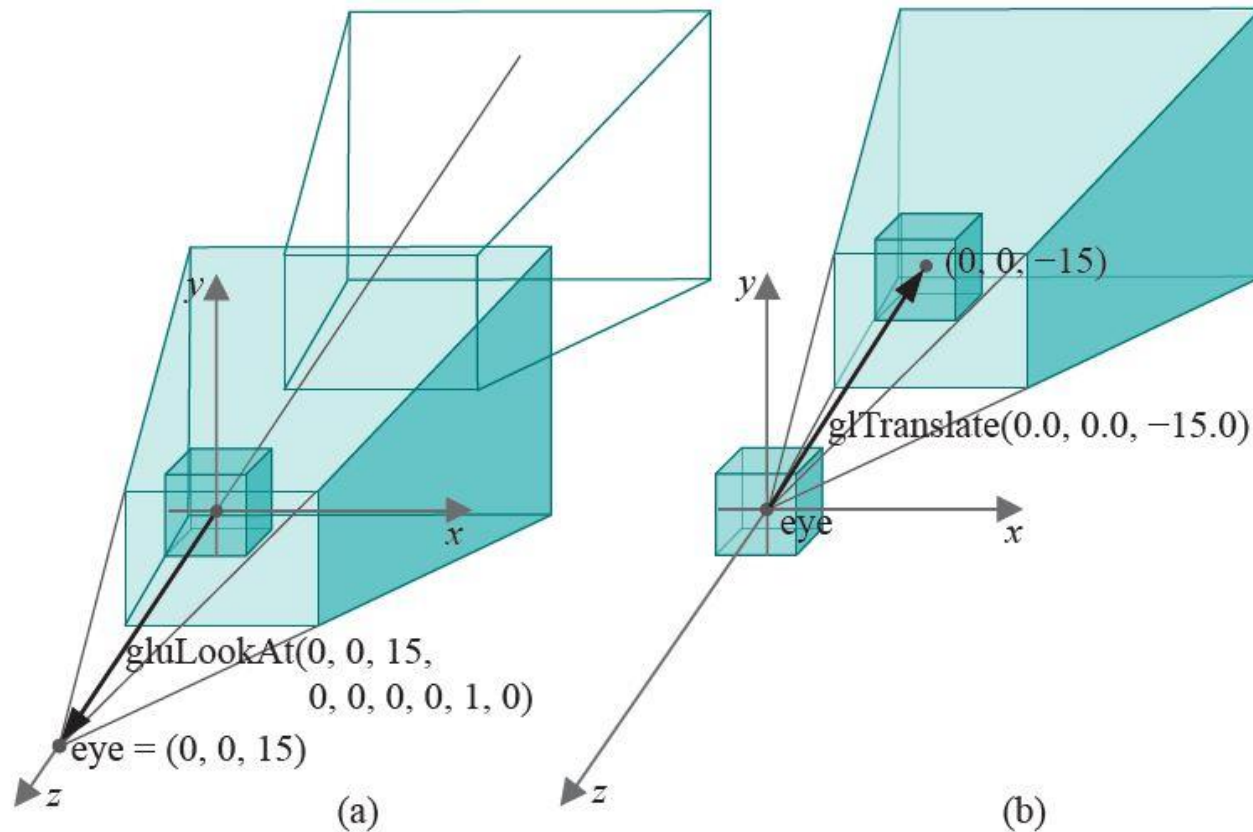
Camera's Default Pose

- it is located at the origin
- its lens points down the $-z$ direction (the line of sight)
- its top aligned along the $+y$ direction (the up direction)
- `gluLookAt(0.0, 0.0, 0.0, 0.0, 0.0, -1.0, 0.0, 1.0, 0.0)`



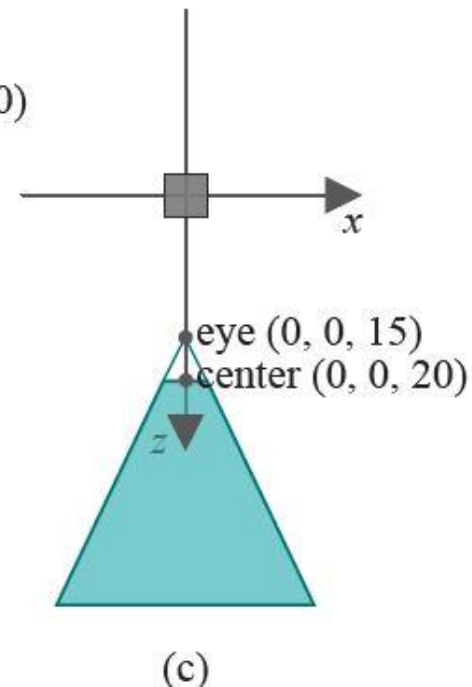
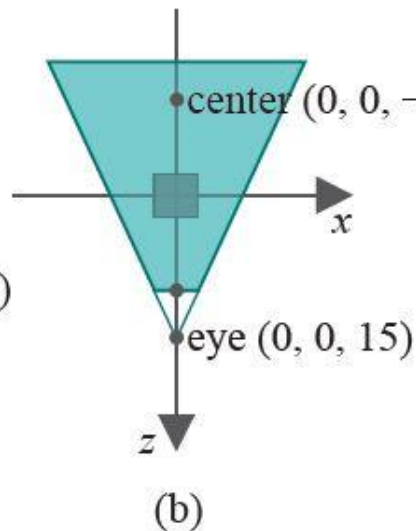
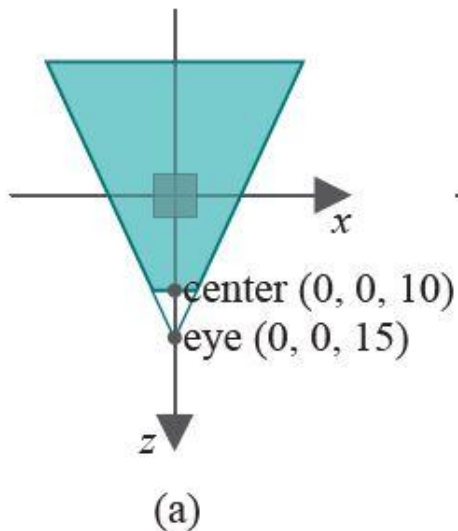
Comparing Modeling Transformation and Viewing Transformation

- `gluLookAt(0.0, 0.0, 15.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)`
- `glTranslatef(0.0, 0.0, -15.0)`



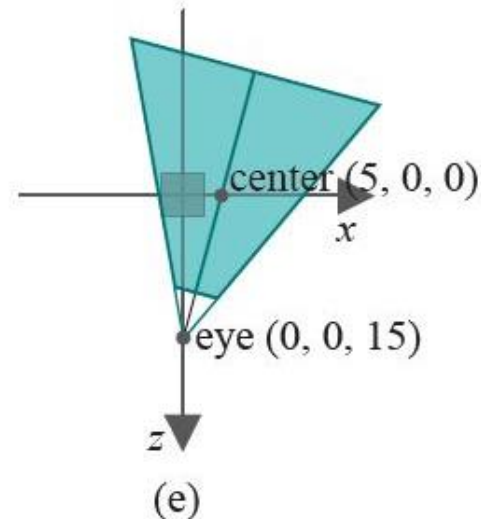
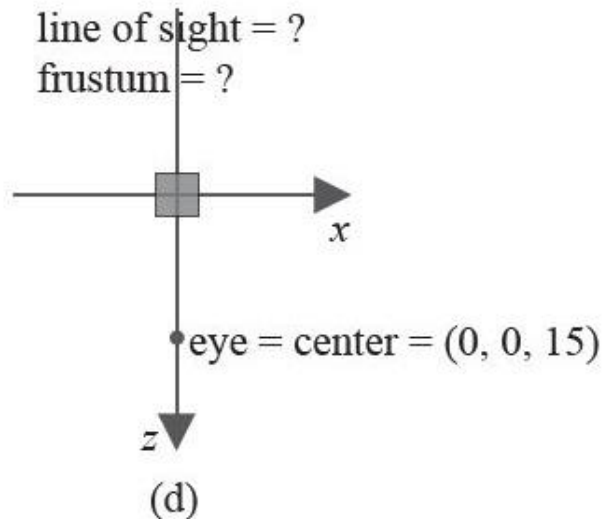
Understanding the Viewing Transformation

- `gluLookAt(0.0,0.0, 15.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)`
- successively change the parameters (`centerx`, `centery`, `centerz`) to the following sets:
 - ▣ (a) `(0.0,0.0,10.0)` (b) `(0.0,0.0,-10.0)` (c) `(0.0,0.0,20.0)`



Understanding the Viewing Transformation

- `gluLookAt(0.0,0.0, 15.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)`
- successively change the parameters (`centerx`, `centery`, `centerz`) to the following sets:
 - ▣ (d) `(0.0,0.0,15.0)` (e) `(5.0,0.0,0.0)`

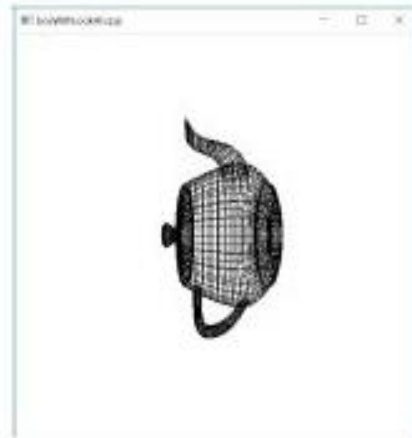


UP Direction

- `gluLookAt(0.0, 0.0, 15.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)`
- successively change the last three parameters (`upx`, `upy`, `upz`) to the following:
 - ▣ (b) `(1.0, 0.0, 0.0)` (c) `(0.0, -1.0, 0.0)` (d) `(1.0, 1.0, 0.0)`



(a)



(b)



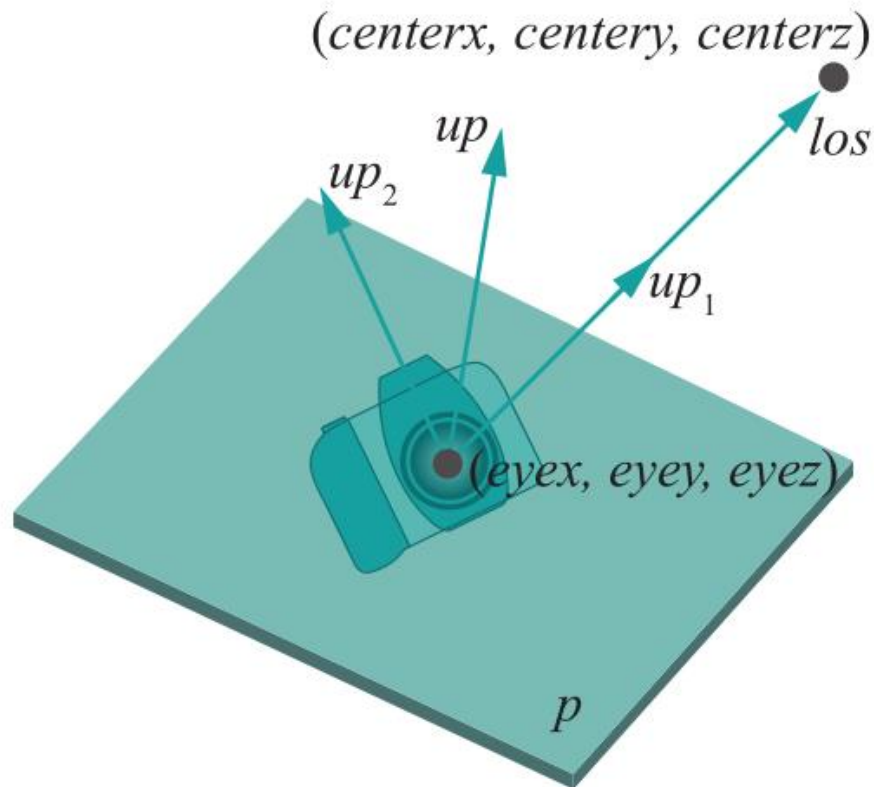
(c)



(d)

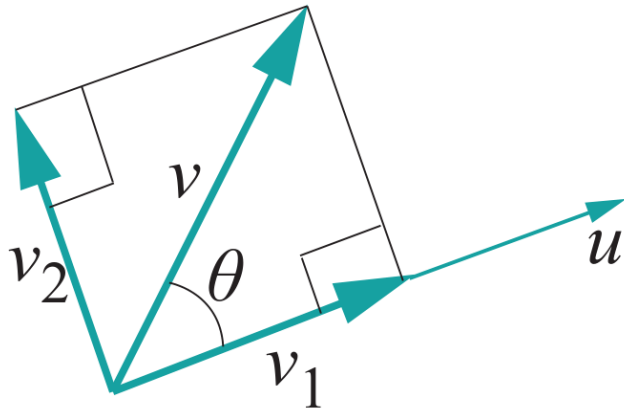
UP Direction

- `gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)`



UP Direction

- Split a given vector v as $v = v_1 + v_2$,
 - ▣ where the components v_1 and v_2 are, respectively, parallel and perpendicular to the given vector u

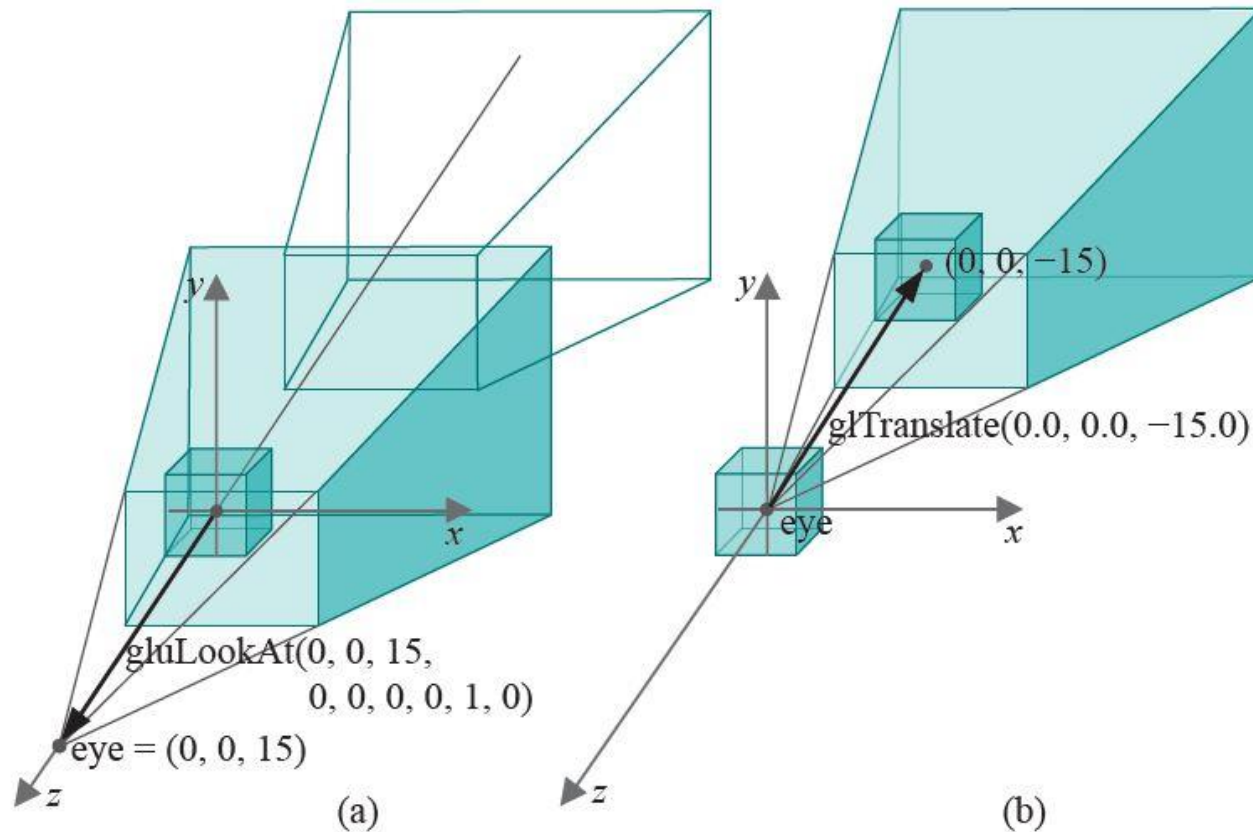


$$v_1 = \frac{u \cdot v}{|u|^2} u$$

$$v_2 = v - v_1 = v - \frac{u \cdot v}{|u|^2} u$$

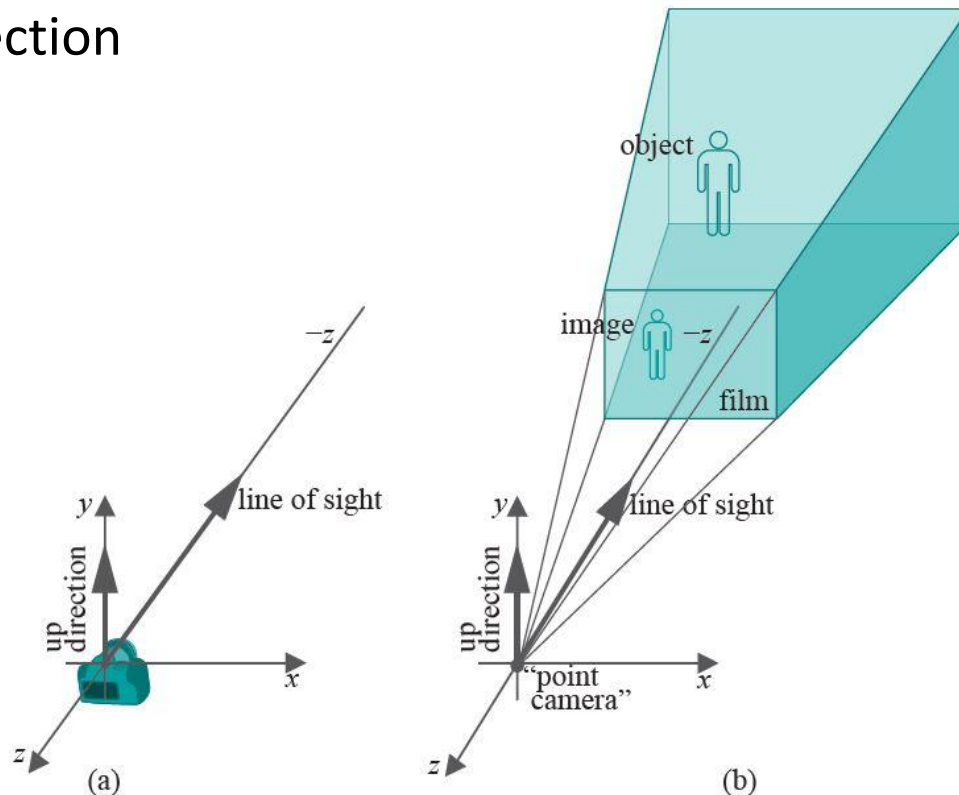
Simulating a Viewing Transformation with Modeling Transformations

- `gluLookAt(0.0, 0.0, 15.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)`
- `glTranslatef(0.0, 0.0, -15.0)`



Simulating a Viewing Transformation with Modeling Transformations

- The viewing transformation is simulated by replacing it with an equivalent sequence of modeling transformations
- The OpenGL camera *never* leaves its default pose at the origin with its lens pointing down the $-z$ direction and with its top aligned along the $+y$ direction

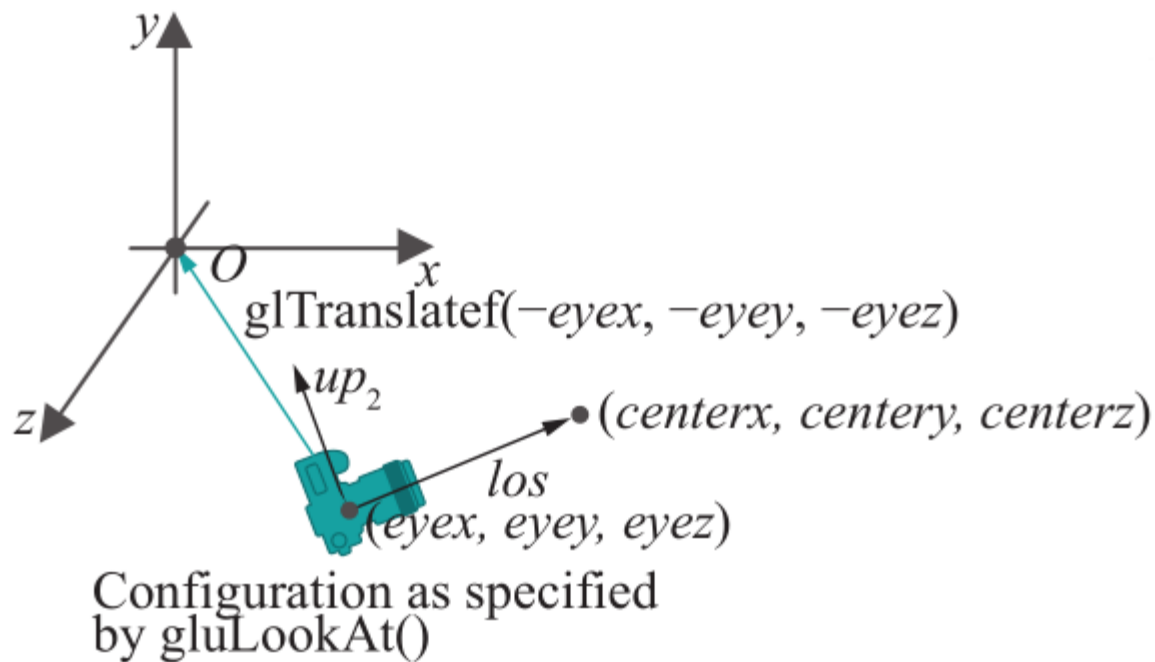


Simulating a Viewing Transformation with Modeling Transformations

- `gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)`
- Question: how to find a sequence of modeling transformations that are equivalent to `gluLookAt()` ?
- Simple example:
 - ▣ `gluLookAt(0.0, 0.0, 15.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0)`
 - ▣ `glTranslatef(0.0, 0.0, -15.0)`

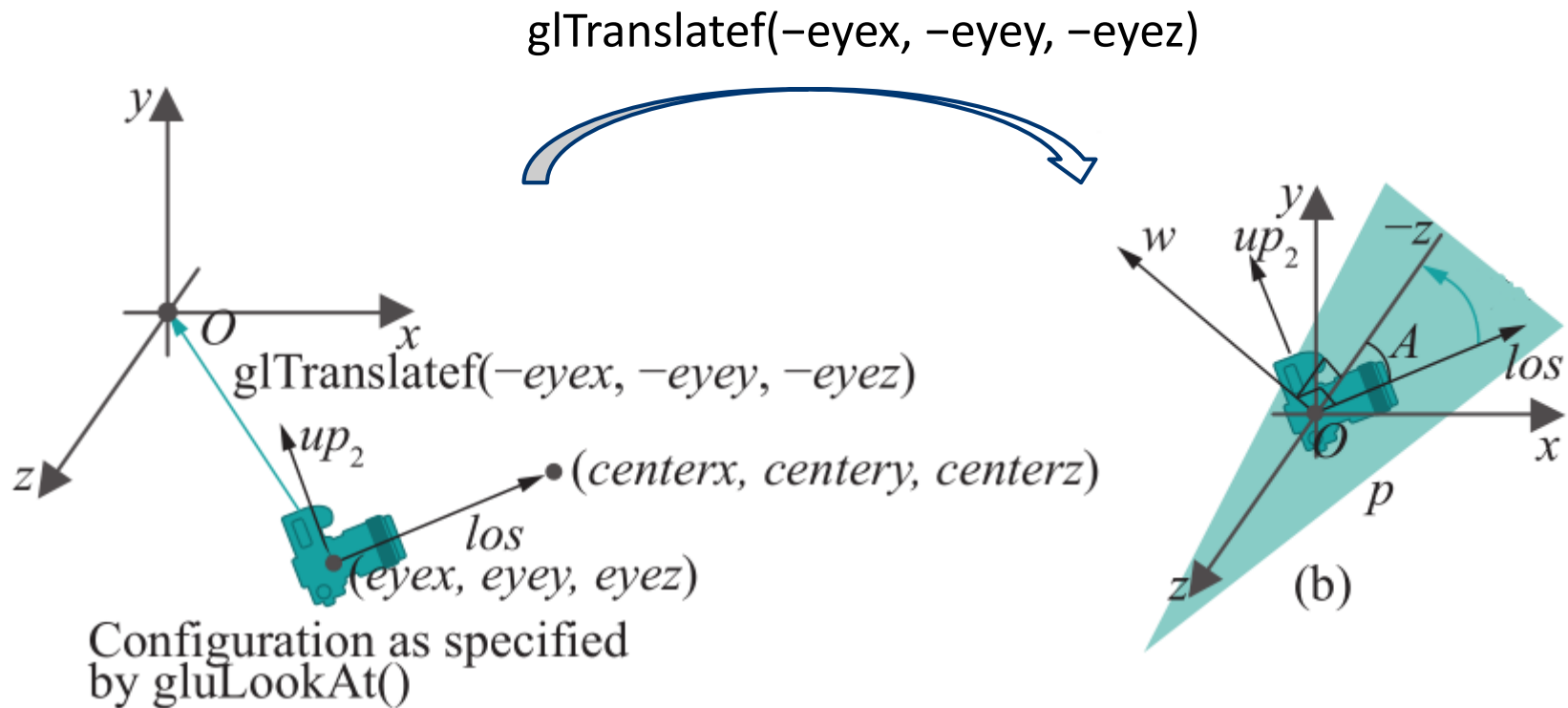
Simulating a Viewing Transformation with Modeling Transformations

- Restore the camera to its default pose



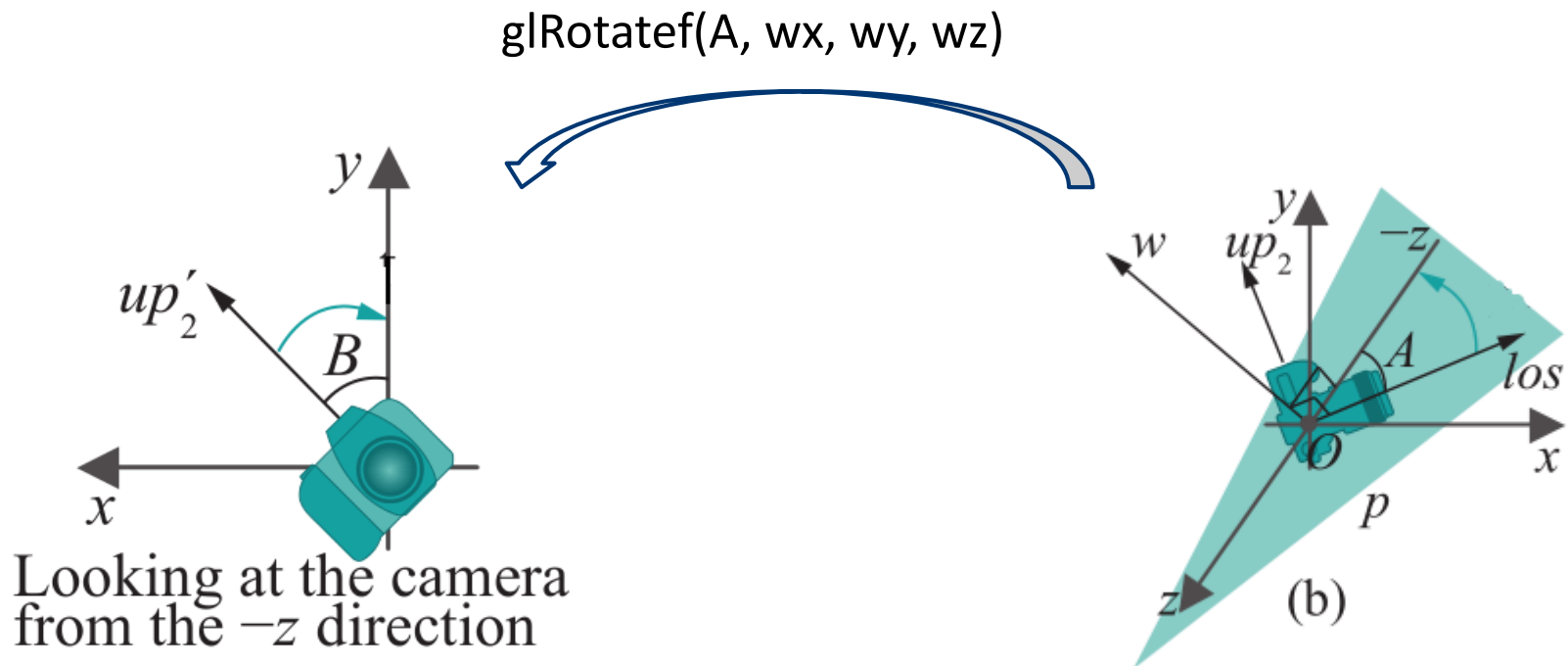
Simulating a Viewing Transformation with Modeling Transformations

- Restore the camera to its default pose



Simulating a Viewing Transformation with Modeling Transformations

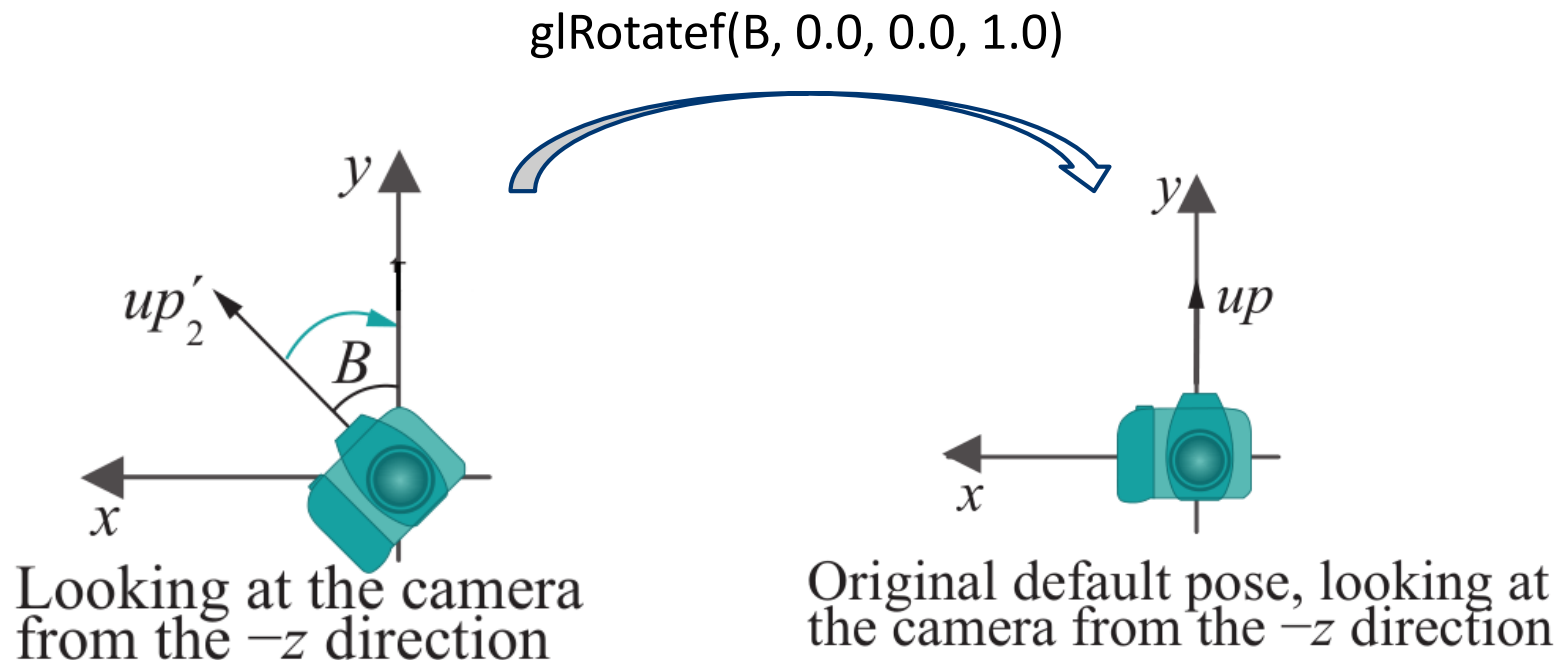
- Restore the camera to its default pose

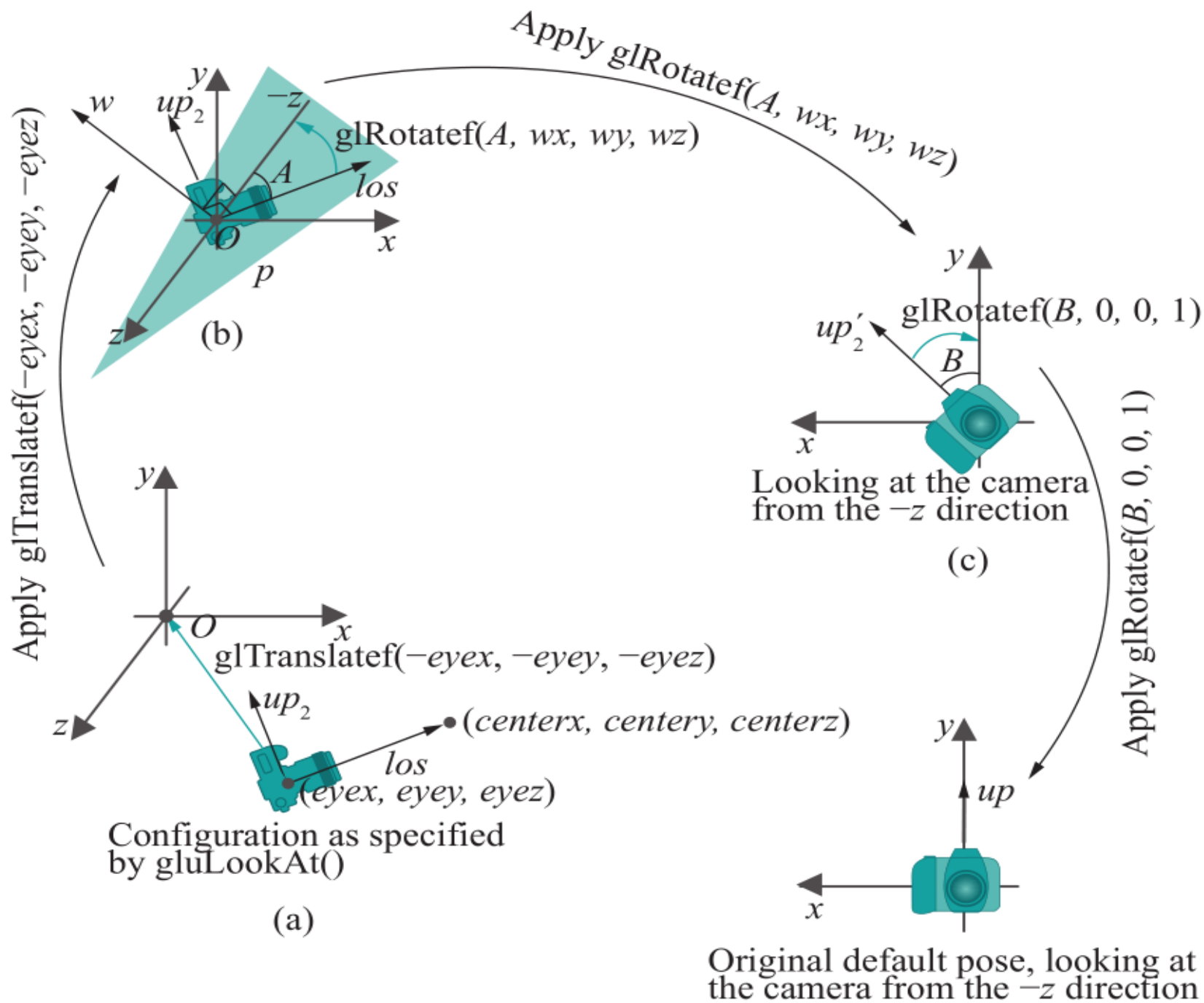


$$w = \text{cross_product}(los, -z)$$

Simulating a Viewing Transformation with Modeling Transformations

- Restore the camera to its default pose





Simulating a Viewing Transformation with Modeling Transformations

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



```
glRotatef(B, 0.0, 0.0, 1.0);
```

```
glRotatef(A, wx, wy, wz);
```

```
glTranslatef(-eyex, -eyey, -eyez);
```

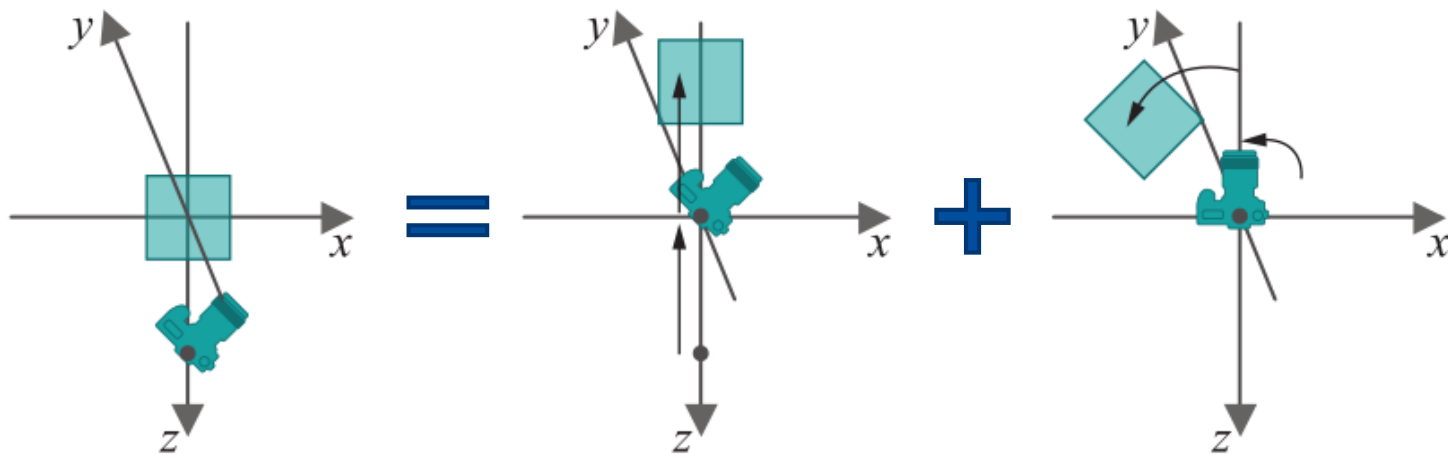
Simulating a Viewing Transformation with Modeling Transformations

```
gluLookAt(0.0, 0.0, 15.0, 15.0, 0.0, 0.0, 0.0, 1.0, 0.0);
```



```
glRotatef(45.0, 0.0, 1.0, 0.0);
```

```
glTranslatef(0.0, 0.0, -15.0);
```



```
gluLookAt(0.0, 0.0, 15.0, 15.0, 0.0, 0.0, 0.0, 1.0, 0.0);  
glTranslatef(0.0, 0.0, -15.0);  
glRotatef(45.0, 0.0, 1.0, 0.0);  
glTranslatef(0.0, 0.0, -15.0);
```



COMPUTER GRAPHICS

Surfaces

Dr. Zhonggui Chen
School of Informatics, Xiamen University
<http://graphics.xmu.edu.cn>

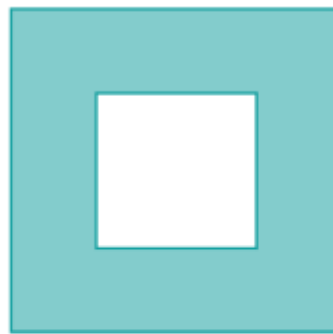
Polygon

- A simple planar polygon
 - ▣ It lies on a plane and its boundary consists of a single component which is a non-self-intersecting loop of straight line segments
 - ▣ `glBegin(GL_POLYGON)-glEnd()` (*convex polygons only*)



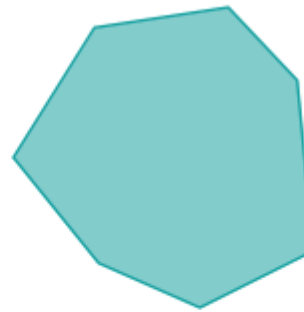
Self-intersecting
boundary

(a)



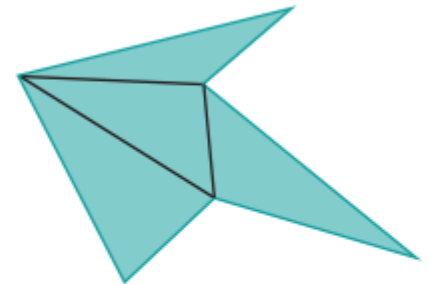
Multiple boundary
components

(b)



Convex

(c)

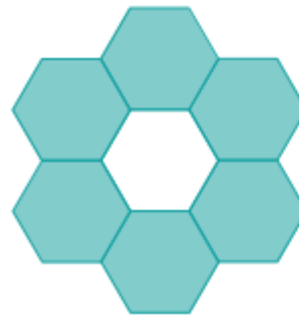


Non-convex (with a
triangulation indicated)

(d)

Meshes

- Mesh (polygonal mesh, polyhedral surface) is a union of convex polygons satisfying the following two conditions:
 - ▣ Any two polygons in the union are either disjoint or intersect in a vertex of both or intersect in an edge of both.
 - ▣ The neighborhood around each vertex is “sheet-like” (homeomorphic to a disk).



(a)



(b)



(c)

(a), (b), (c), and (f)
Meshes (d) and (e) Not
meshes: parts around
vertices U and W are
not sheet-like.



(d)



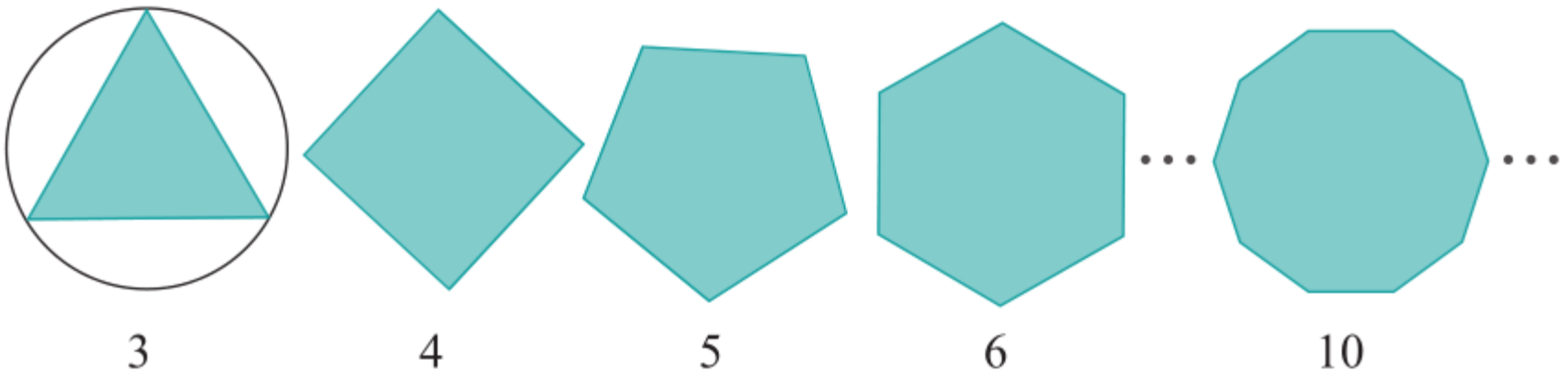
(e)



(f)

Regular polygons

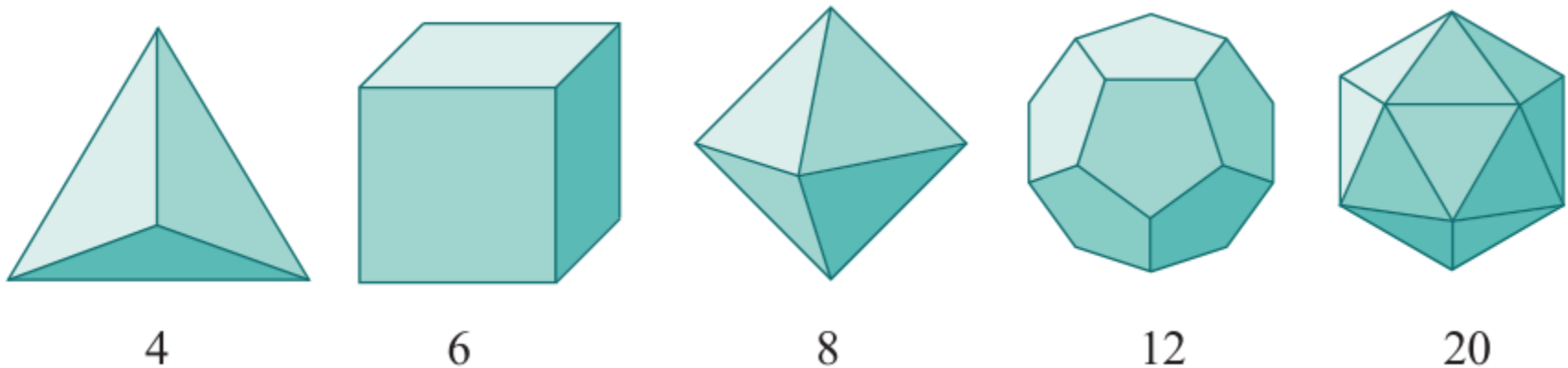
- A regular polygon is a simple planar polygon whose sides are of equal length and which has equal interior angles at its vertices



Regular polygons with number of sides indicated.

Regular polyhedra

- A polyhedron is a solid object whose boundary is a polygonal mesh.
- A regular polyhedron is a polyhedron all of whose faces are identical regular polygons.



The five regular polyhedra with the number of faces indicated.

Modeling Regular Polyhedra

- The coordinates of the vertices of the Cube, as well as the indices of the vertices comprising each of its faces, are listed in the following two tables

Cube

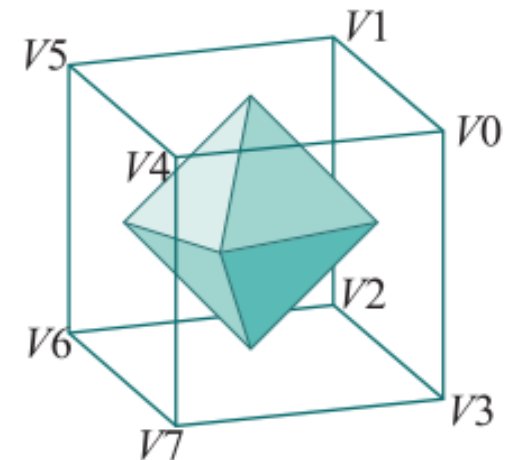
Vertex	Coordinates	Face	Vertices
V0	(1, 1, 1)	F0	(V3, V0, V1, V2)
V1	(1, 1, -1)	F1	(V2, V1, V5, V6)
V2	(1, -1, -1)	F2	(V6, V5, V4, V7)
V3	(1, -1, 1)	F3	(V7, V4, V0, V3)
V4	(-1, 1, 1)	F4	(V1, V0, V5, V4)
V5	(-1, 1, -1)	F5	(V3, V2, V6, V7)
V6	(-1, -1, -1)		
V7	(-1, -1, 1)		

Modeling Regular Polyhedra

□ Duality

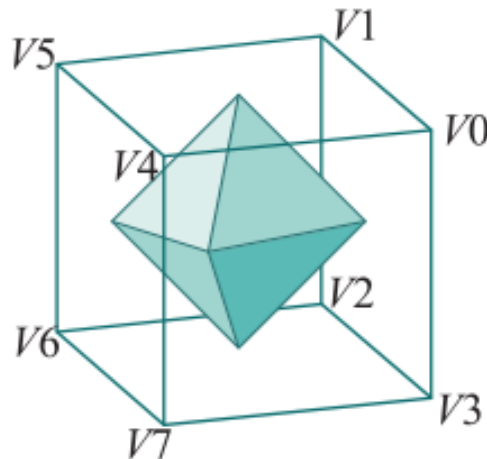
Octahedron

Vertex	Coordinates	Face	Vertices
$V'0$	$(1, 0, 0)$	$F'0$	$(V'0, V'4, V'3)$
$V'1$	$(0, 0, -1)$	$F'1$	$(V'4, V'0, V'1)$
$V'2$	$(-1, 0, 0)$	$F'2$	$(V'5, V'1, V'0)$
$V'3$	$(0, 0, 1)$	$F'3$	$(V'5, V'0, V'3)$
$V'4$	$(0, 1, 0)$	$F'4$	$(V'2, V'3, V'4)$
$V'5$	$(0, -1, 0)$	$F'5$	$(V'1, V'2, V'4)$
		$F'6$	$(V'2, V'1, V'5)$
		$F'7$	$(V'5, V'3, V'2)$



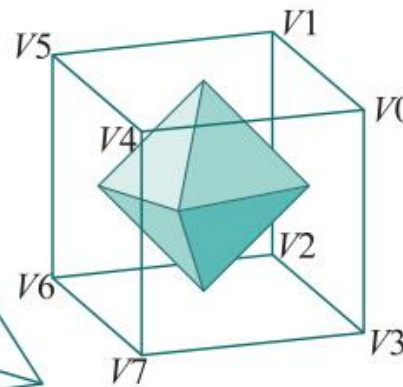
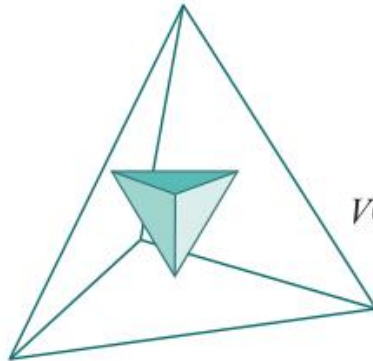
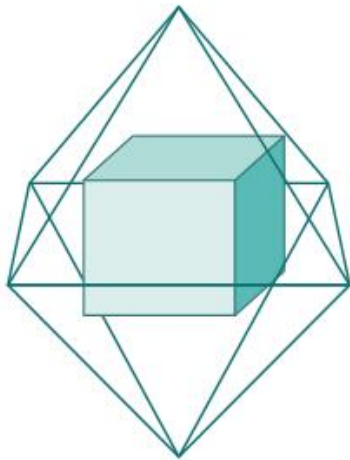
Modeling Regular Polyhedra

- The dual of a regular polyhedron P is the polyhedron P' inscribed in P as follows:
 - For each face f of P there is a vertex of P' , called f 's dual, located at the center of f
 - For each edge e of P there is an edge of P' , called e 's dual, joining the dual of the two faces of P adjacent to e
 - For each vertex v of P there is a face of P' , called v 's dual, with vertices at the duals of the faces of P that meet at v .



Duality

- The five regular polyhedra each contains its inscribed dual



Cubes and octahedrons are duals of one another.

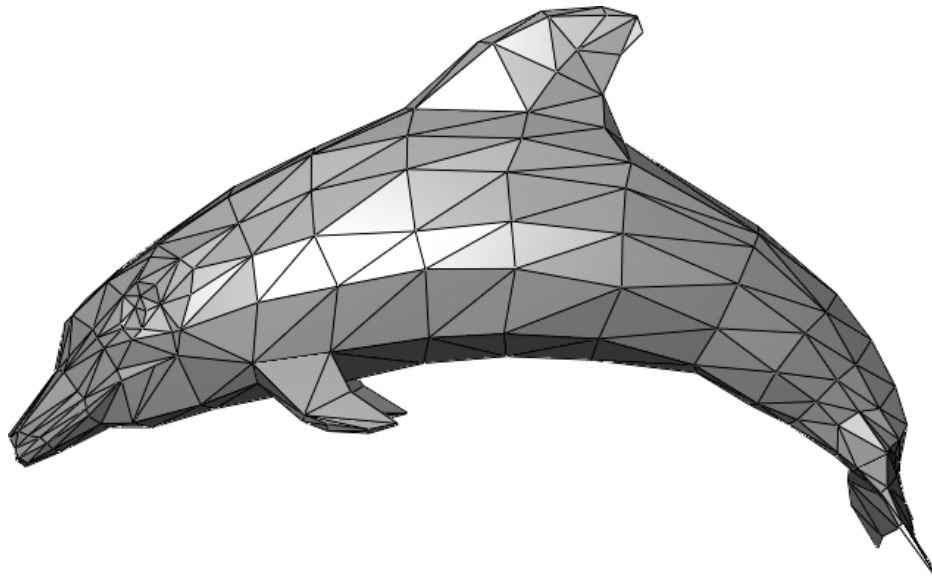
Tetrahedrons are self-dual.



Dodecahedrons and icosahedrons are duals of one another.

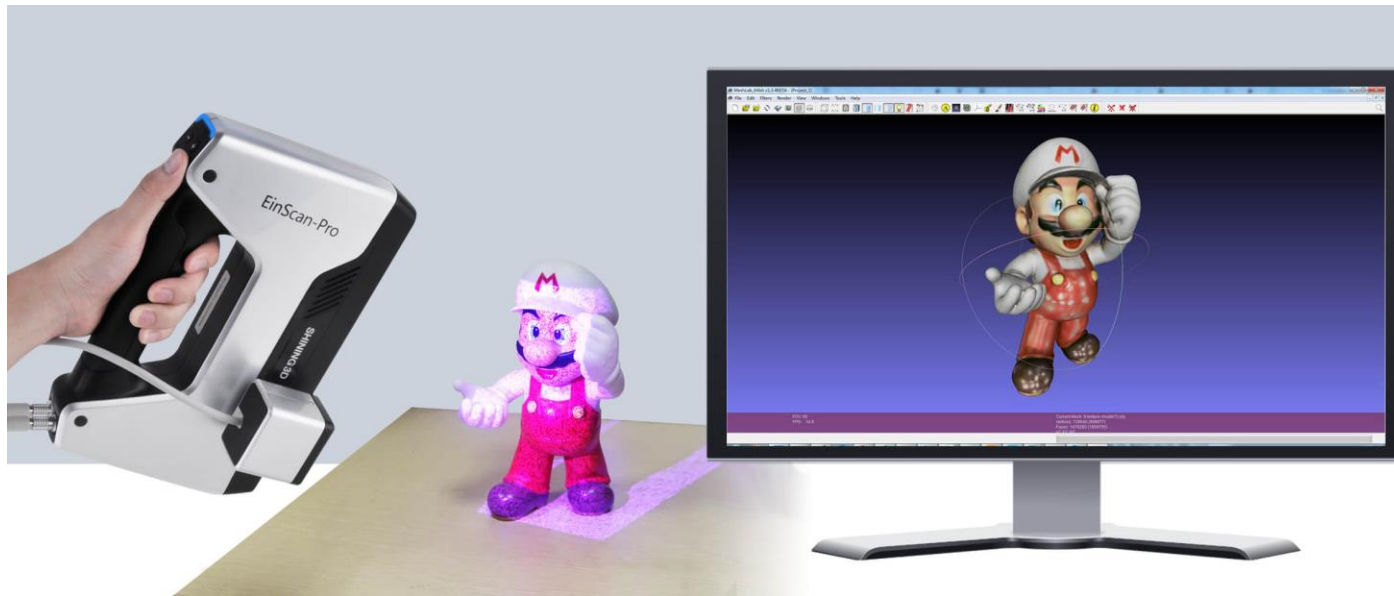
Modeling General Meshes

- How to model general mesh surfaces?
- 3D modeling software: Maya, Studio Max or Blender
- 3D scanning techniques



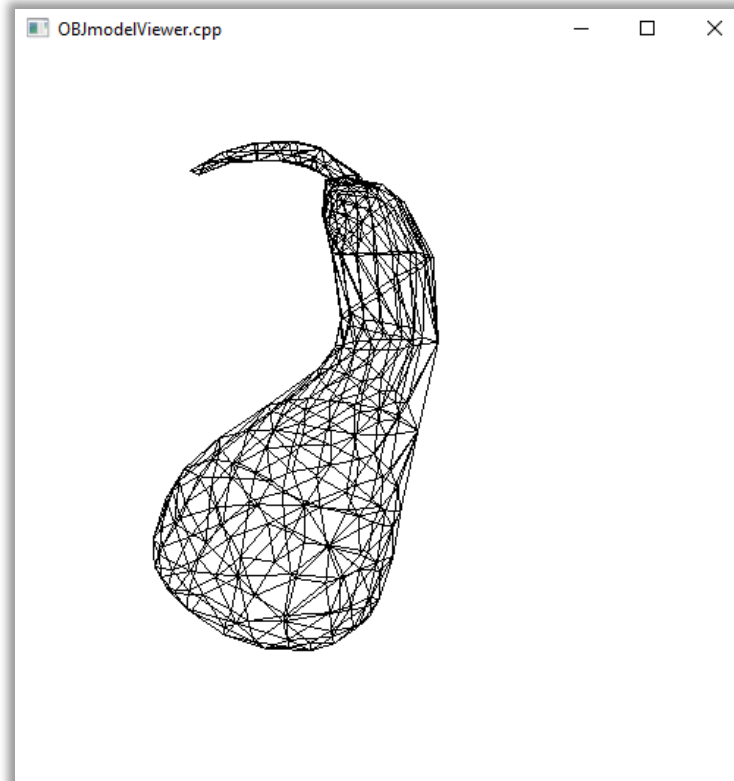
Modeling General Meshes

- How to model general mesh surfaces?
- 3D modeling software: Maya, Studio Max or Blender
- 3D scanning techniques



Importing Objects

- 3D file formats: OBJ, PLY, 3DS, STL, LWS, FBX, DXF, Collada, and IFC
- Example: Chapter10\OBJModelViewer\OBJmodelViewer.cpp



OBJ File Format

□ rectangle.obj

```
# rectangle.obj
# specified intentionally in an odd way;
# ...
v 0.0 0.0 0.0
v 1.0 0.0 0.0
f 1/1/2 2/2/1 3/2/1 4/3/2
v 1.0 1.0 0.0
vt 0.4 0.5
v 0.0 1.0 0.0
vt 0.2 0.9
vn 0.1 0.2 0.1
vn 0.4 0.1 0.3
vt 0.3 0.3
```

OBJ File Format


□ rectangle.obj

```
# rectangle.obj
# specified intentionally in an odd way;
# ...
v 0.0 0.0 0.0
v 1.0 0.0 0.0
f 1/1/2 2/2/1 3/2/1 4/3/2
v 1.0 1.0 0.0
vt 0.4 0.5
v 0.0 1.0 0.0
vt 0.2 0.9
vn 0.1 0.2 0.1
vn 0.4 0.1 0.3
vt 0.3 0.3
```

OBJ File Format

□ rectangle.obj

```
# rectangle.obj
# specified intentionally in an odd way;
# ...
v 0.0 0.0 0.0
v 1.0 0.0 0.0
f 1 2 3 4
v 1.0 1.0 0.0
vt 0.4 0.5
v 0.0 1.0 0.0
vt 0.2 0.9
vn 0.1 0.2 0.1
vn 0.4 0.1 0.3
vt 0.3 0.3
```



v/vt/vn:

v corresponds to the v-th in the list of v-lines in the OBJ file, starting with index 1 for the first v-line;

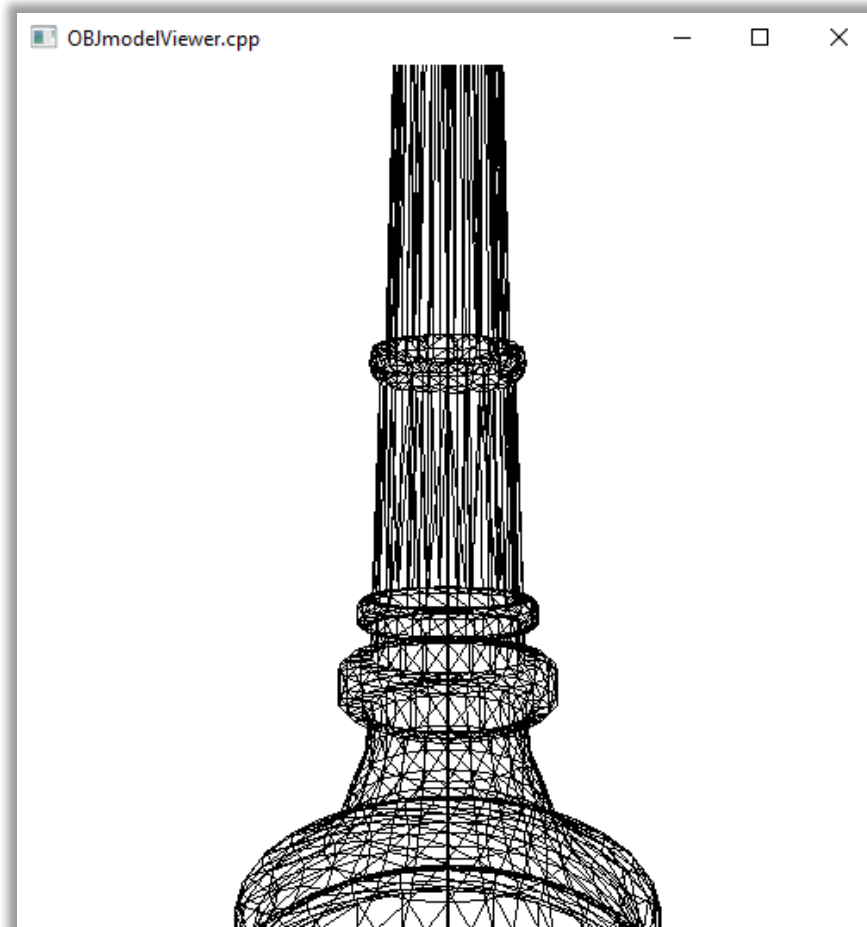
v-lines are not necessarily consecutive in the file.

Only the vertex index is mandatory for each vertex of a face; the texture coordinates and normal values indices are not.

Importing Objects

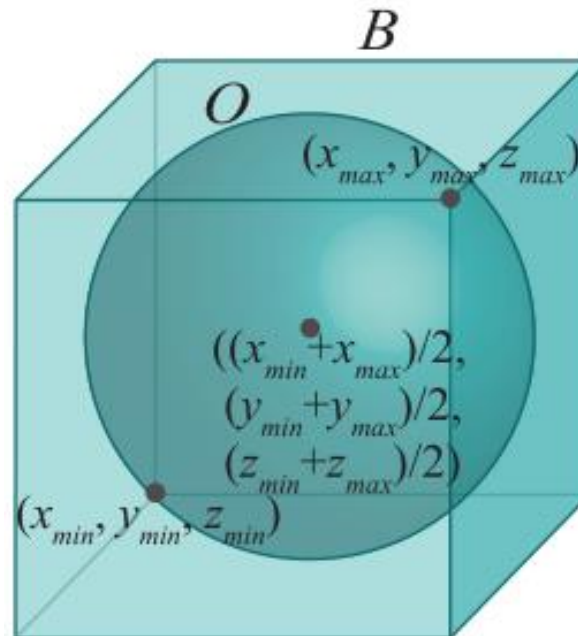
```
void loadOBJ(std::string fileName)
{
    std::string line;
    // Open the OBJ file.
    std::ifstream inFile(fileName.c_str(), std::ifstream::in);
    // Read successive lines.
    while (getline(inFile, line))
    {
        // Line has vertex data.
        if (line.substr(0, 2) == "v "){ ... }
        // Line has face data.
        else if (line.substr(0, 2) == "f "){ ... }
        // Nothing other than vertex and face data is processed.
        else {}
    }
    // Close the OBJ file.
    inFile.close();
}
```

Centralizing the Object



Bounding Box

- Suppose the smallest of the x-values of the points comprising an object O is x_{\min} , the largest of the x-values is x_{\max} , and that y_{\min} , y_{\max} , z_{\min} and z_{\max} are similarly defined.
- The bounding box of O is the axis-aligned box B with diagonally opposite corners at $(x_{\min}, y_{\min}, z_{\min})$ and $(x_{\max}, y_{\max}, z_{\max})$



Bounding box B
of a sphere O .

Centralizing the Object

- Centralize the bounding box

```
glScalef(4.0/diagonal, 4.0/diagonal, 4.0/diagonal);  
glTranslatef(-(Xmax+Xmin)/2, -(Ymax+Ymin)/2, -(Zmax + Zmin)/2);
```

