COMP 3271 Tutorial

Programming Assignment II

Tutorial on 3D Modeling

Submission

- Deadline:

  11:59pm, Nov. 9, 2018HKT

- Submission:

  rayView.cpp

If you want to change any other files or implement the program without the template, please email me(wenhua00@hku.hk) before submission.

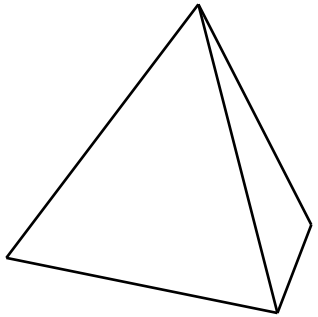
Outline

- 3D Modeling & 3D Rendering in OpenGL
- 3D Modeling & Interface Design – Programming Assignment II
  - About the Template
  - About the Task
    - Mouse controlled model editing
    - 3D solid handles

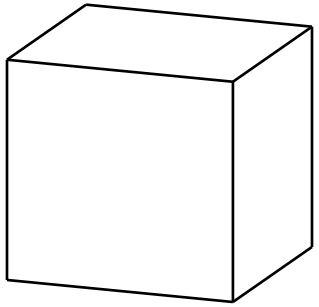
Outline

- 3D Modeling & 3D Rendering in OpenGL
- 3D Modeling & Interface Design – Programming Assignment
  - About the Template
  - About the Task
    - Mouse controlled model editing
    - 3D solid handles

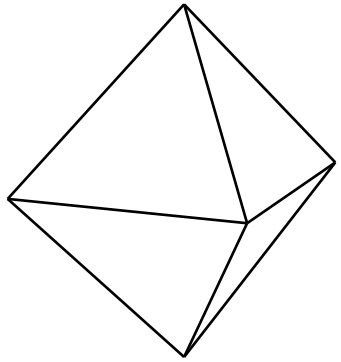
Some 3D Primitives



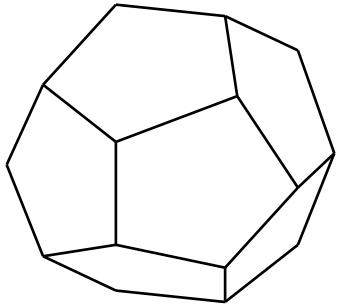
Tetrahedron



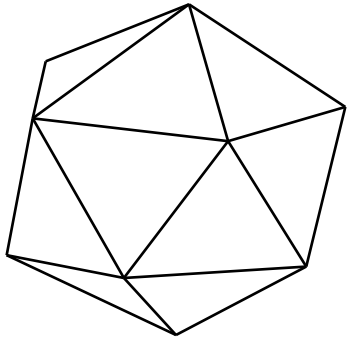
Cube



Octahedron



Dodecahedron (12 faces)



Icosahedron (20 faces)

Draw Geometric Objects – 3D

void glVertex3{b,s,i,f,ub,us,ui}(TYPE x, TYPEy, TYPEz);

void glVertex3{b,s,i,f,ub,us,ui}v(const TYPE*v);

- Example1: draw a triangle in 3D space

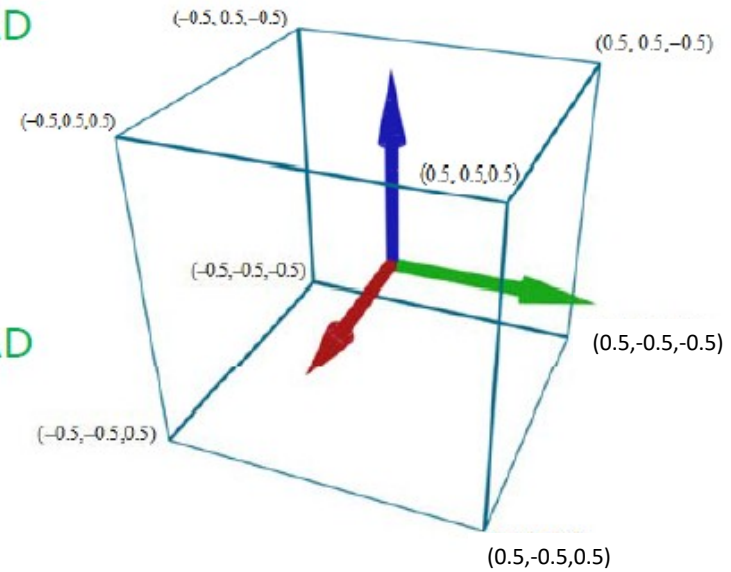
```
glBegin(GL_TRIANGLES); //Specify object type

    glVertex3f(-0.5, 0.5,0.5);      // Object's coordinates
    glVertex3f(0.5, 0.5, 0.6);
    glVertex3f(0.5, -0.5, 1.0);

glEnd();
```

Draw Geometric Objects – 3D

- Example2: draw a cube

Solution 1: Use GL_QUADS to draw a box.

```
glBegin(GL_QUADS);
glVertex3f(-0.5f, -0.5f, -0.5f); //First QUAD
glVertex3f(-0.5f, 0.5f, -0.5f);
glVertex3f(0.5f, 0.5f, -0.5f);
glVertex3f(0.5f, -0.5f, -0.5f);
... ...
glVertex3f(-0.5f, -0.5f, 0.5f); //Sixth QUAD
glVertex3f(-0.5f, 0.5f, 0.5f);
glVertex3f(0.5f, 0.5f, 0.5f);
glVertex3f(0.5f, -0.5f, 0.5f);
glEnd();
```

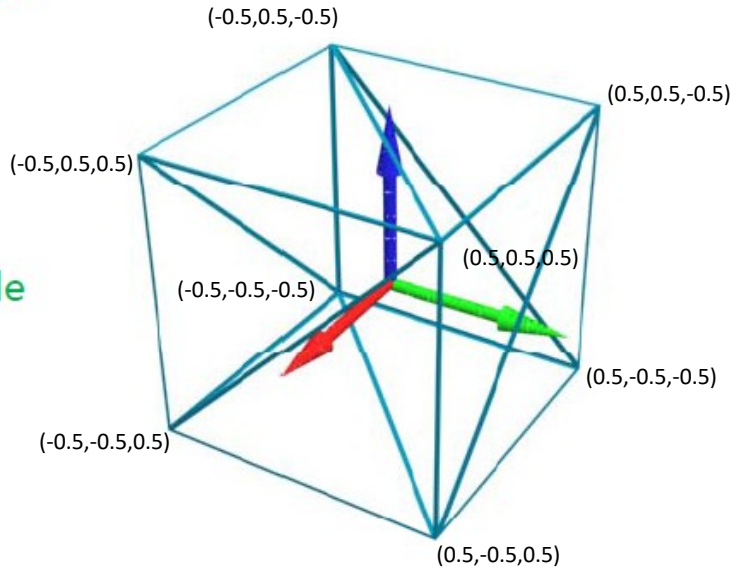


Draw Geometric Objects – 3D

- Example2: draw a cube

Solution 2: Use GL_TRIANGLES to draw a box.

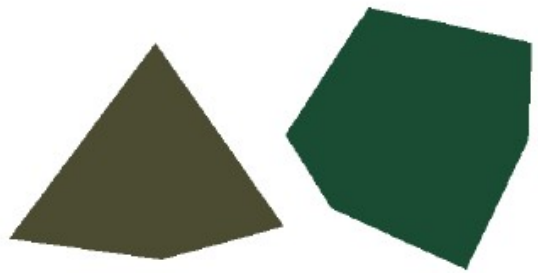
```
glBegin(GL_TRIANGLES);
glVertex3f(-0.5f, -0.5f, -0.5f); //1st triangle
glVertex3f(-0.5f, 0.5f, -0.5f);
glVertex3f(0.5f, 0.5f, -0.5f);
... ...

glVertex3f(-0.5f, 0.5f, 0.5f); //12th triangle
glVertex3f(0.5f, 0.5f, 0.5f);
glVertex3f(0.5f, -0.5f, 0.5f);
glEnd();
```



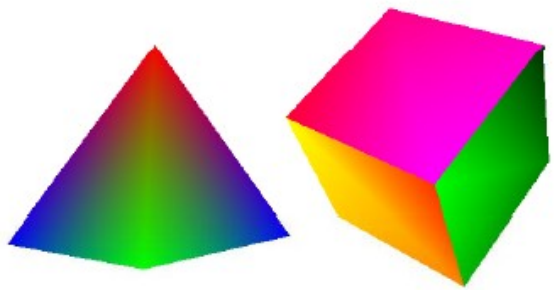
Draw Geometric Objects – 3D

Sample: use color to show 3D effect.

```
glBegin(GL_TRIANGLES);
glColor3f(1.0f, 0.0f, 0.0f); //First triangle
glVertex3f(-0.5f, -0.5f, -0.5f);
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3f(-0.5f, 0.5f, -0.5f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(0.5f, 0.5f, -0.5f);
    ... ...
glEnd();
```



Vertices with the same color



Vertices with different colors

GLU Quadrics

- **Function Introduction**

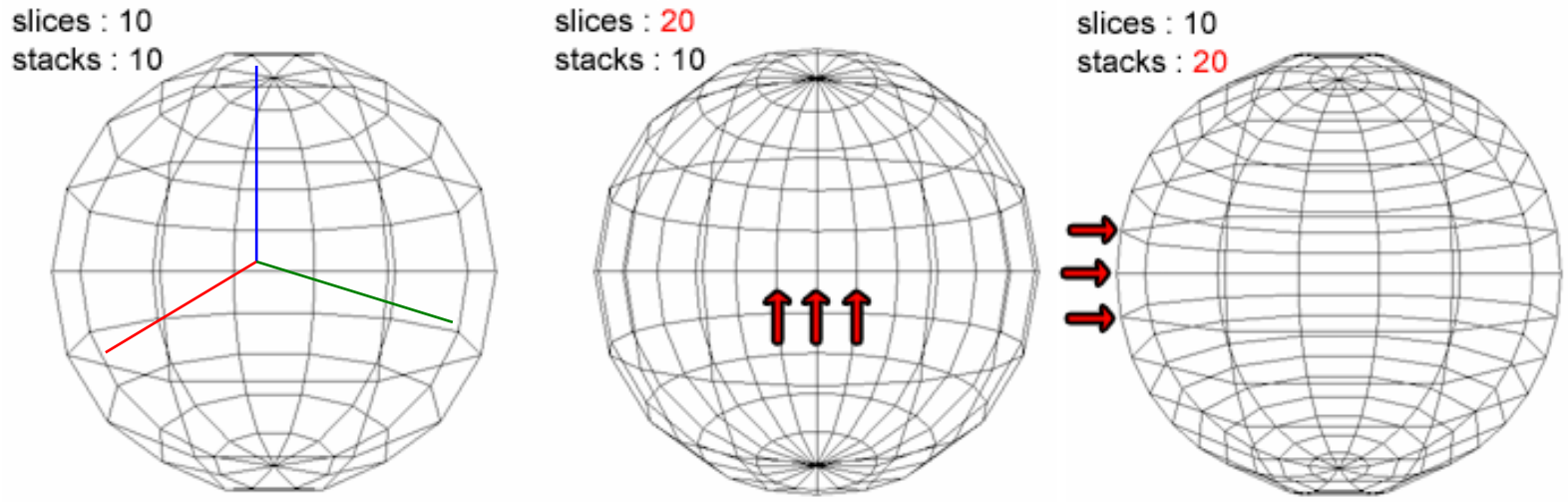
```
void  gluSphere(GLUquadric* quadric,
                GLdouble radius,
                GLint  slices,
                Glint  stacks)
```

Purpose : Draw a sphere symmetric around the origin.
quadric : Object pointer to a created quadric object.
radius: The radius of sphere .
slices: The number of segments in the vertical direction.
stacks: The number of segments in the horizon direction.

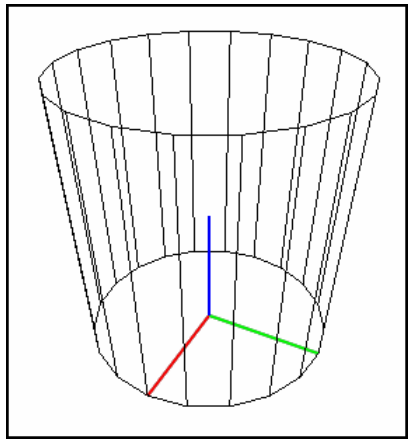
GLU Quadrics

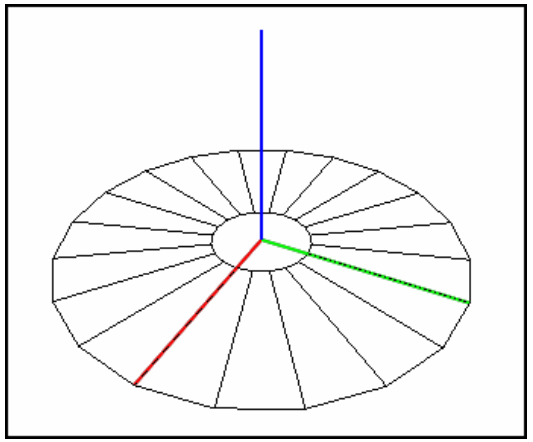
- Sample Code

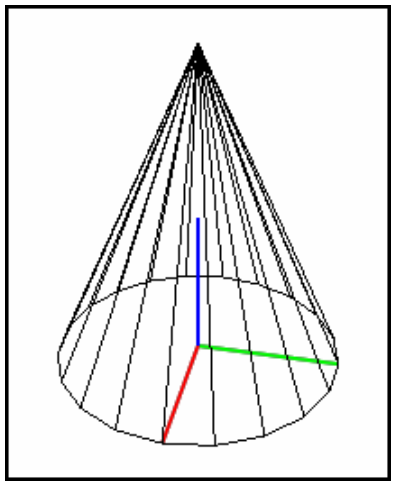
```
…
GLUquadricObj *qu;
qu=gluNewQuadric();
gluSphere(qu, 2.0f, 10, 10);
gluDeleteQuadric(qu);
…
```



Draw a cone



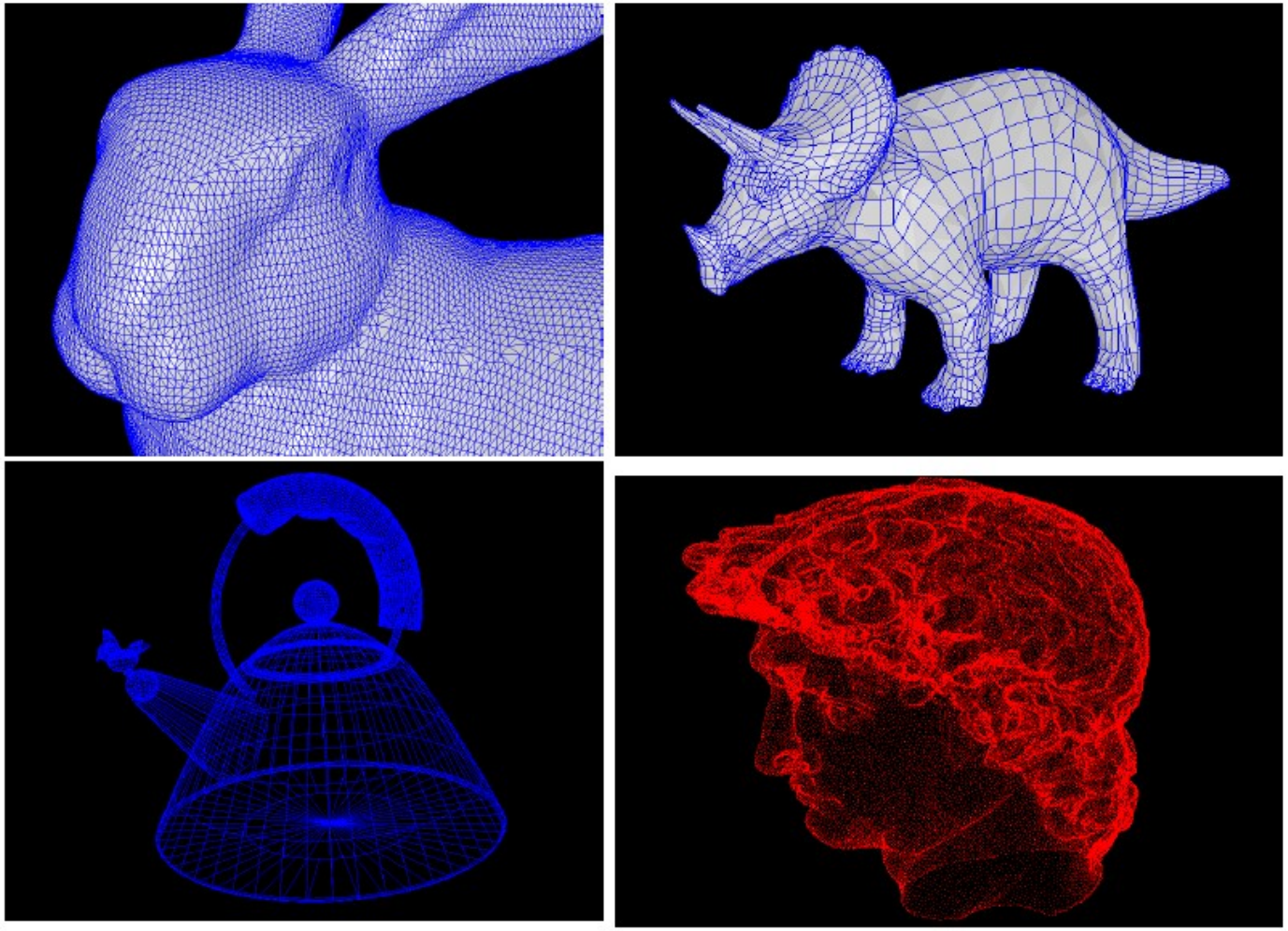




- **Cylinder** gluCylinder(quadric, baseRadius, topRadius, height, slices, stacks)
- (a **cone** is a cylinder with one radius = 0)
- **Disk** gluDisk(quadric, innerRadius, outerRadius, slices, rings)

```
/* draw a cone */
GLUquadricObj *qu;
qu=gluNewQuadric();
gluCylinder(qu, 1.0f, .0f,
2.0f,30,20);
gluDisk(qu,.0f,1.0f,30,5);
gluDeleteQuadric(qu);
```

Draw Geometric Objects – 3D



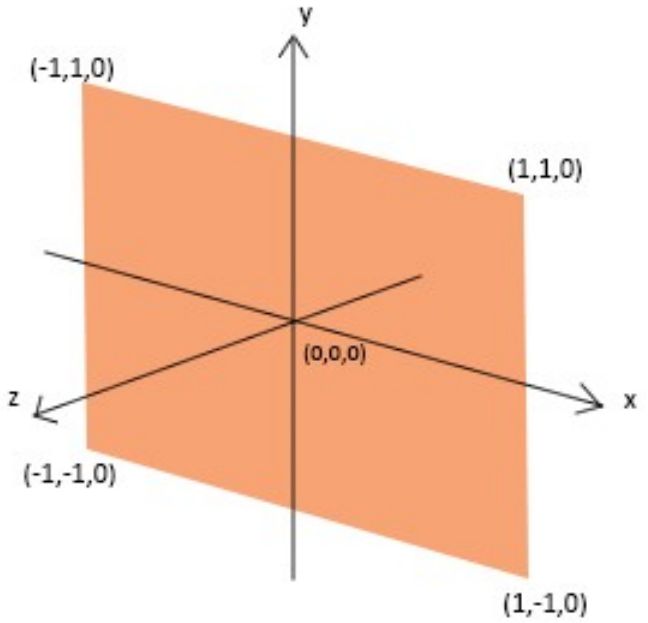
3D Rendering

- Coordinate system

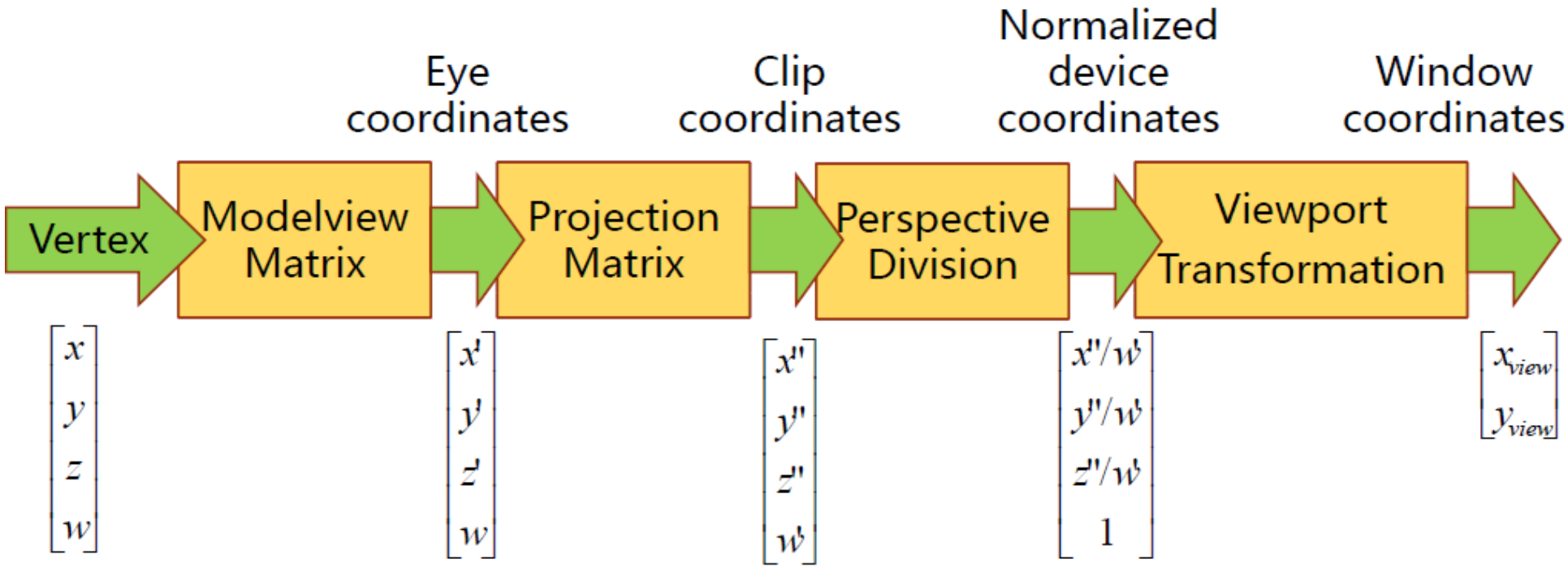
1. Object or model coordinates
2. World coordinates
3. Eye (or Camera) coordinates

} Initial state

4. Clip coordinates
5. Normalized device coordinates
6. Window (or screen) coordinates



3D Rendering



3D Rendering

- Choose the current matrix

Projection matrix

Define the canvas in 2D case and the viewing volume in 3D case.

gluOrtho2D, glOrtho, gluPerspective, glFrustum

Modelview matrix

Define the transformation of the camera and objects.

gluLookAt, glTranslate, glRotate, glScale

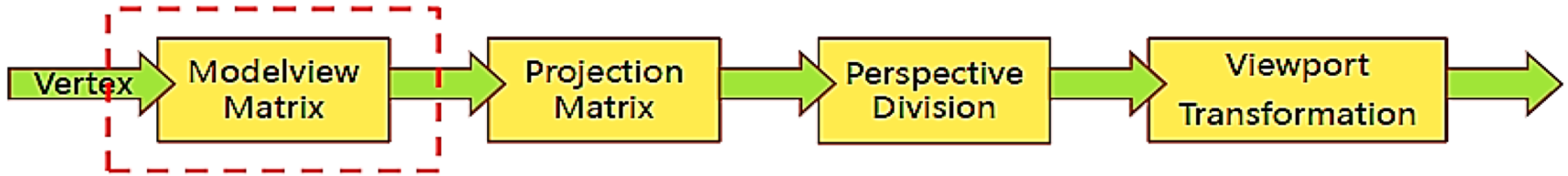
Choose the matrix to operate with:

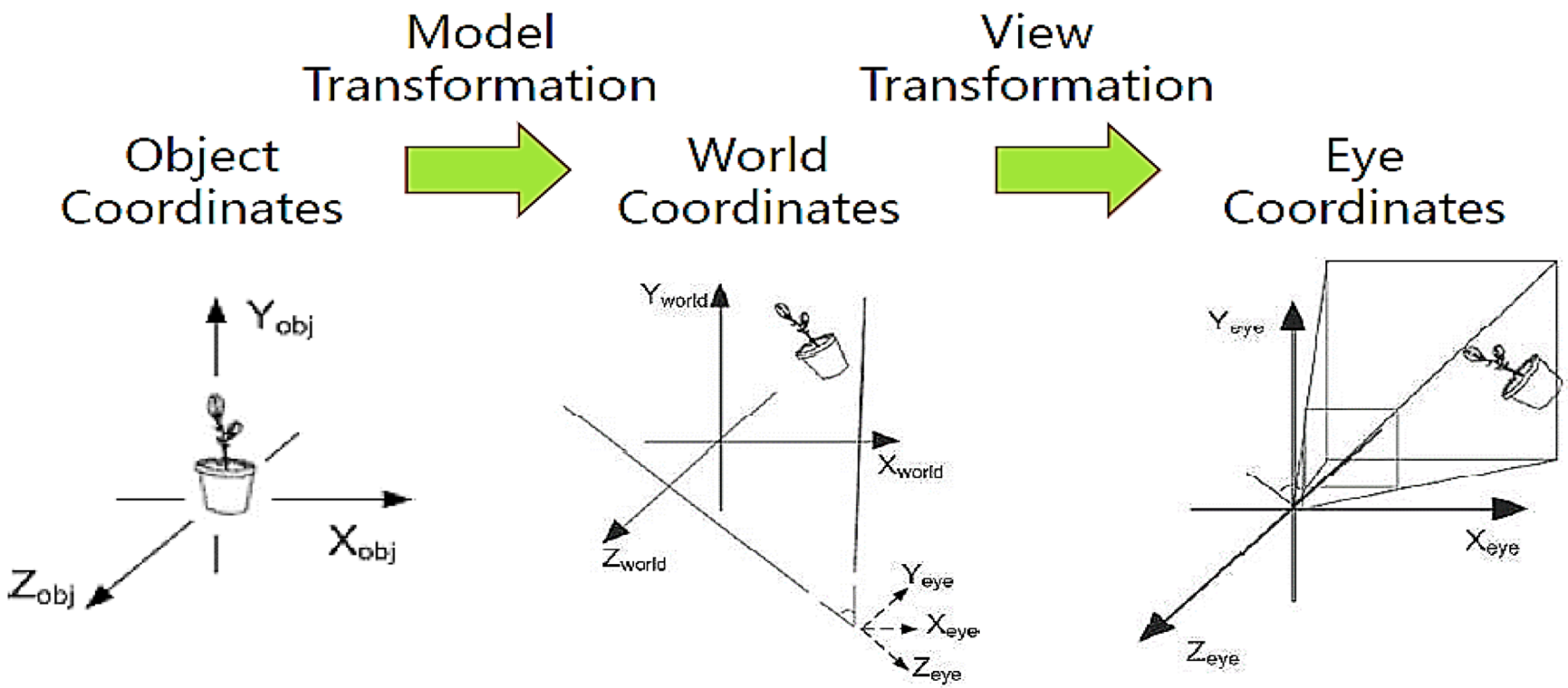
```
void glMatrixMode(MatrixType);
```

GL_PROJECTION: Choose projection matrix

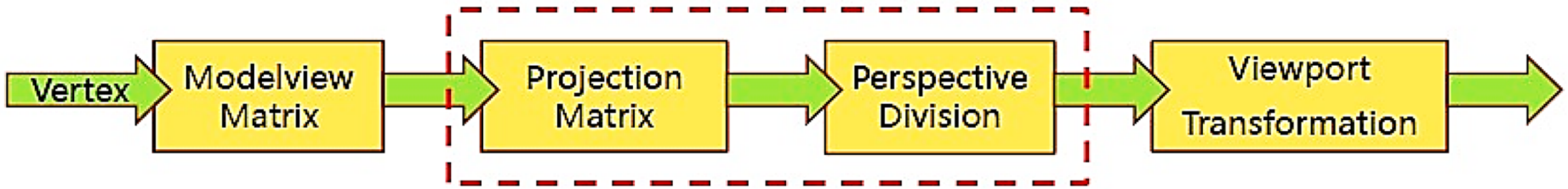
GL_MODELVIEW: Choose model view matrix

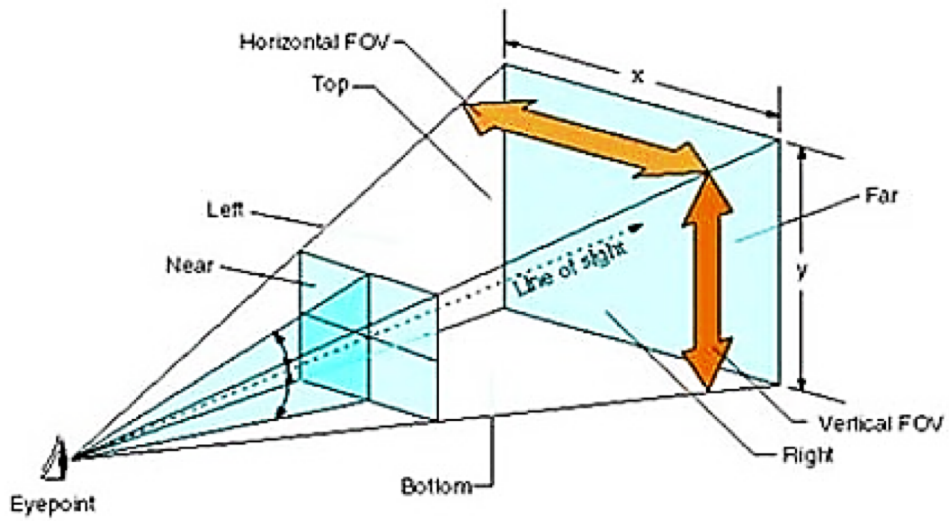
3D Rendering

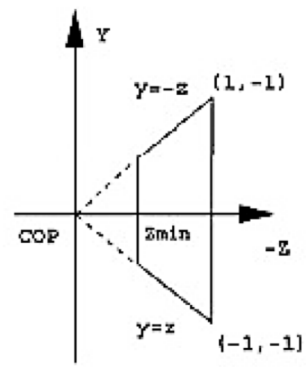


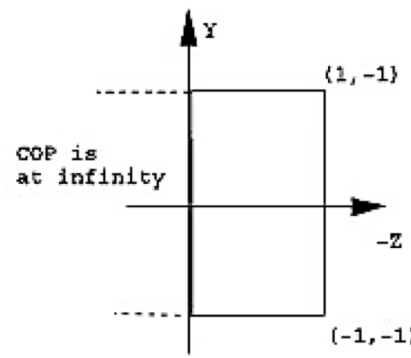


3D Rendering

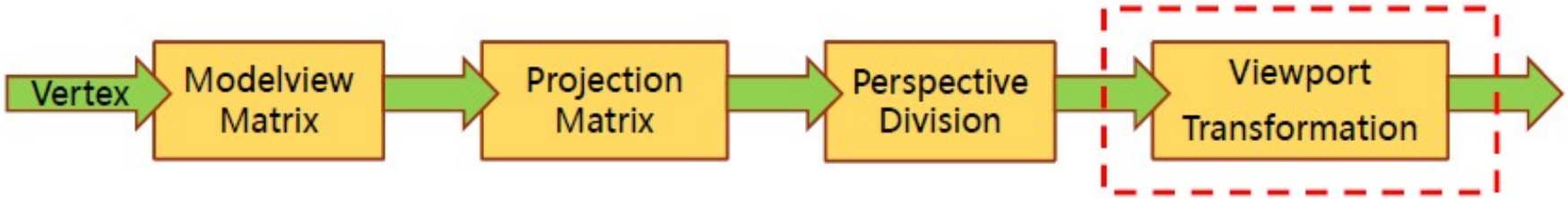


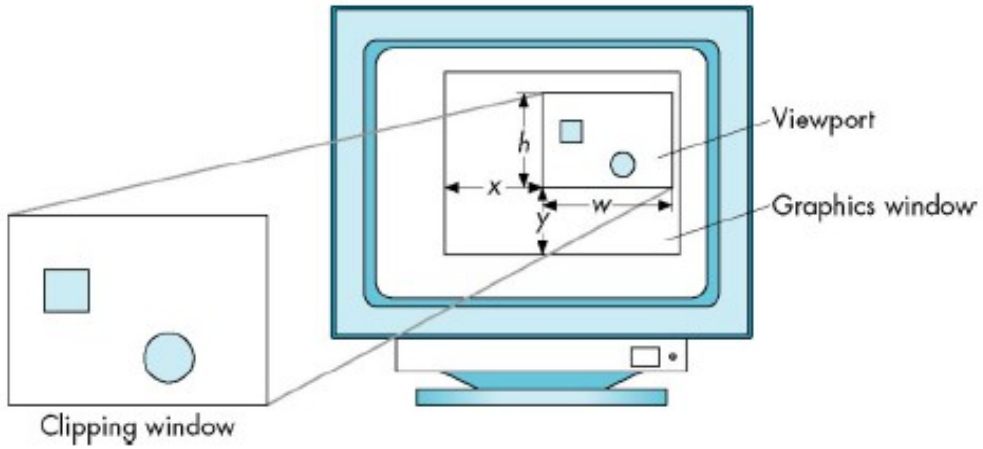






3D Rendering





3D Rendering

As a programmer, you need to do the following things:

- Specify the location/parameters of camera.
- Specify the geometry of the objects.
- Specify the lights (optional).

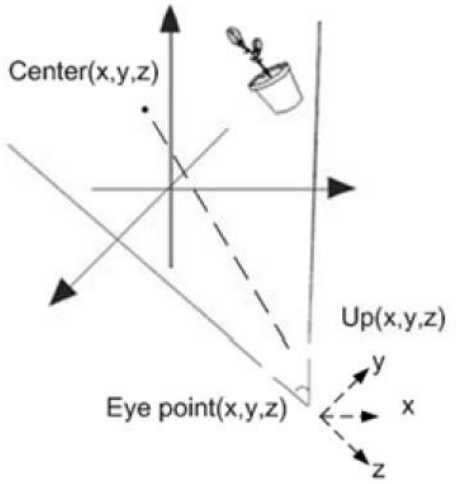
OpenGL will compute the result 2D image!

3D Rendering

- **Specify the location of camera**

gluLookAt(eyeX, eyeY, eyeZ, centerX, centerY, centerZ, upX, upY, upZ);

(eyeX, eyeY, eyeZ): Eye point position.
(centerX, centerY, centerZ) : Reference point.
(upX, upY, upZ) : Direction of up vector.

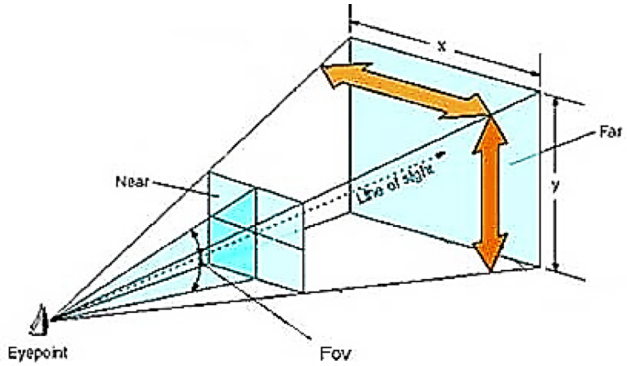


3D Rendering

- **Define the viewing volume**

Perspective projection

gluPerspective(fov, aspect, near, far);



Purpose: Define the perspective projection matrix

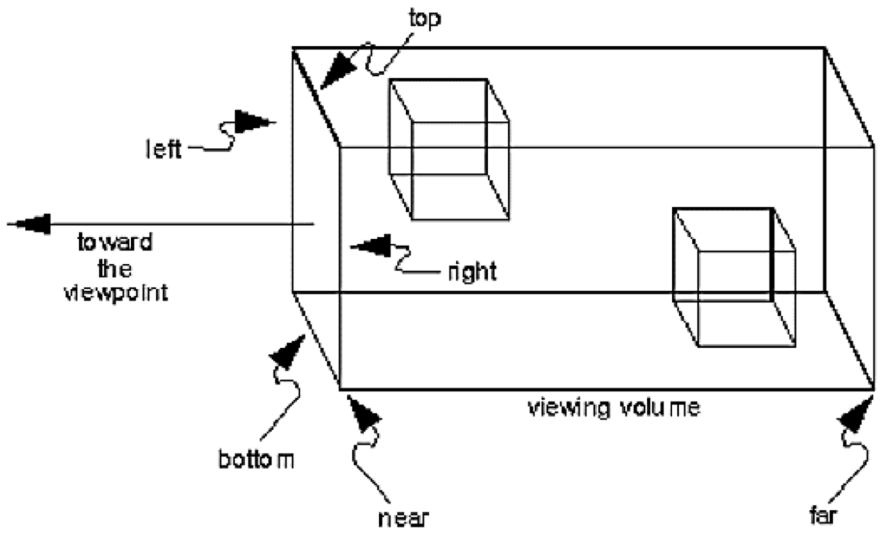
fov: specify the angle of scene view in model space.
aspect: specify the scene view height/width ratio;
near : distance of near plane from eye point(>0).
far : distance of far plane from eyepoint(>near).

3D Rendering

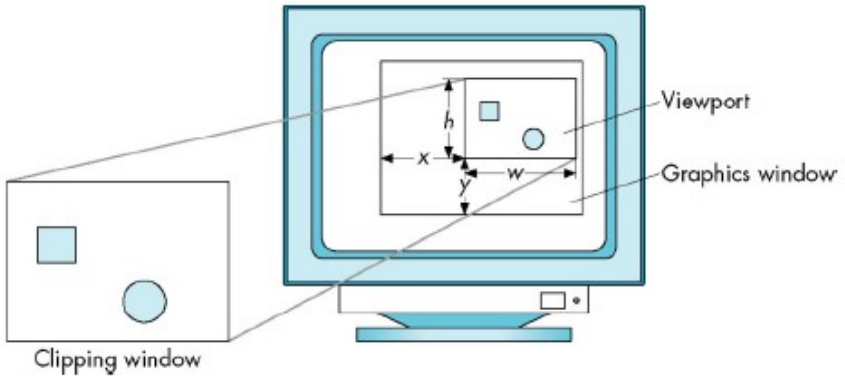
- **Define the viewing volume**

  Orthogonal projection

  glOrtho(Xmin, Xmax, Ymin, Ymax, near, far)



3D Rendering



void glViewport(GLint x, GLint y, GLsizei w, GLsizei h);

Purpose: Set viewport for model scene rendering.

(x , y) : define the left down point of a viewport.

(w, h) : define the height (h) and width (w) of the viewport.

Outline

- 3D Modeling & 3D Rendering in OpenGL
- 3D Modeling & Interface Design – Programming Assignment
  - About the Template
  - About the Task
    - Mouse controlled model editing
    - 3D solid handles

Template Provided Functions

- 3D Primitives Modeling
  - Triangular based models

    Cube Pyramid Tetrahedron Octahedron Icosahedron Dodecahedron
  - Quadratics models

    Sphere Cone Cylinder
- Object(s) Selection
- Navigate Control Mode
- Viewpoint rotation

Outline

- 3D Modeling & 3D Rendering in OpenGL
- 3D Modeling & Interface Design – Programming Assignment
  - About the Template
  - About the Task
    - Mouse controlled model editing
    - 3D solid handles

Your Tasks

- Mouse controlled model editing (in ‘UserMouseMove’ function*)
  - Translation (axis aligned)
  - Rotation (Trackball style)
  - Scaling (axis aligned)
- 3D solid handles (in ‘UserDrawControlHandle’ function*)

*Both functions are in ‘RayView.cpp’

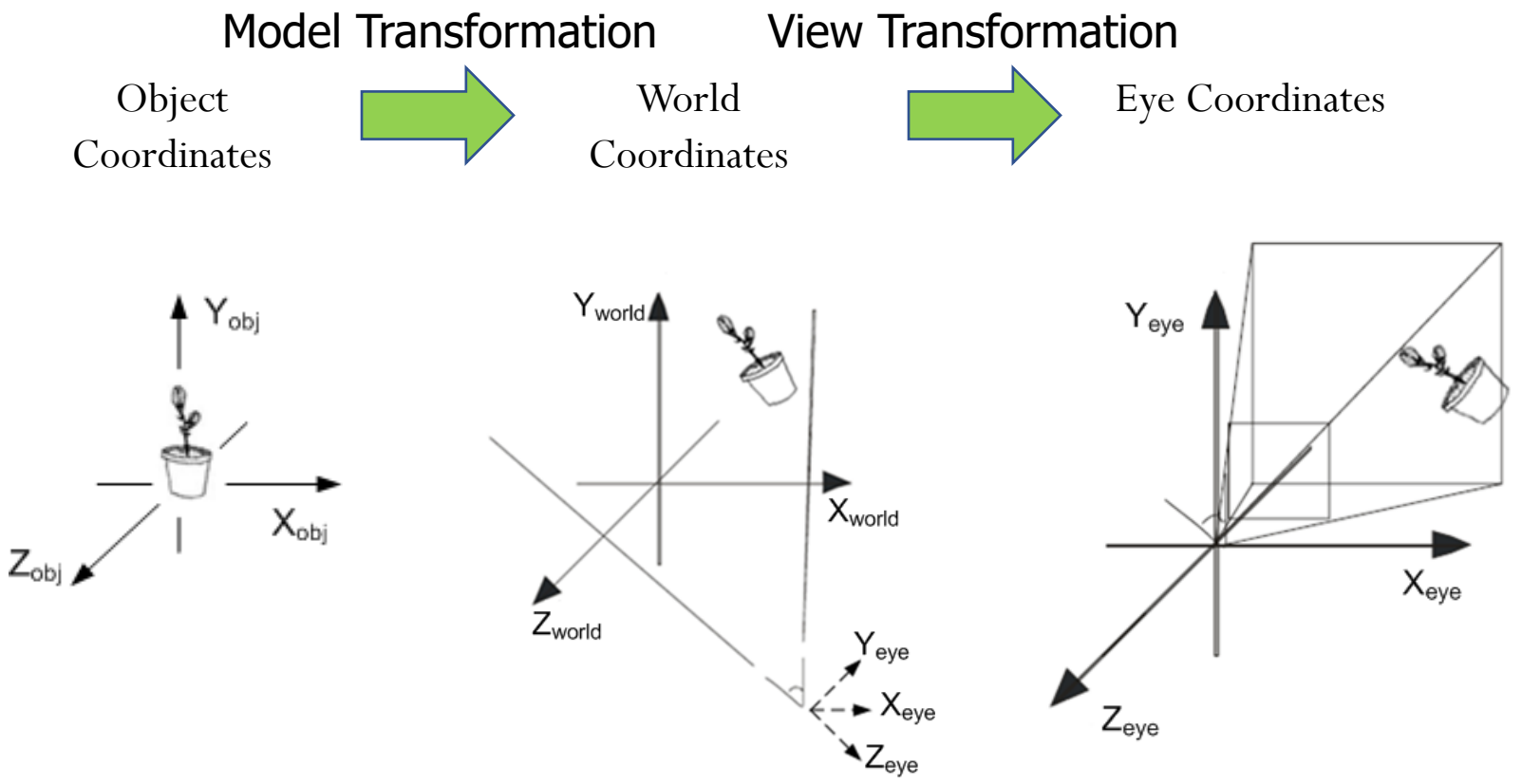
Outline

- 3D Modeling & 3D Rendering in OpenGL
- 3D Modeling & Interface Design – Programming Assignment
  - About the Template
  - About the Task
    - Mouse controlled model editing
    - 3D solid handles

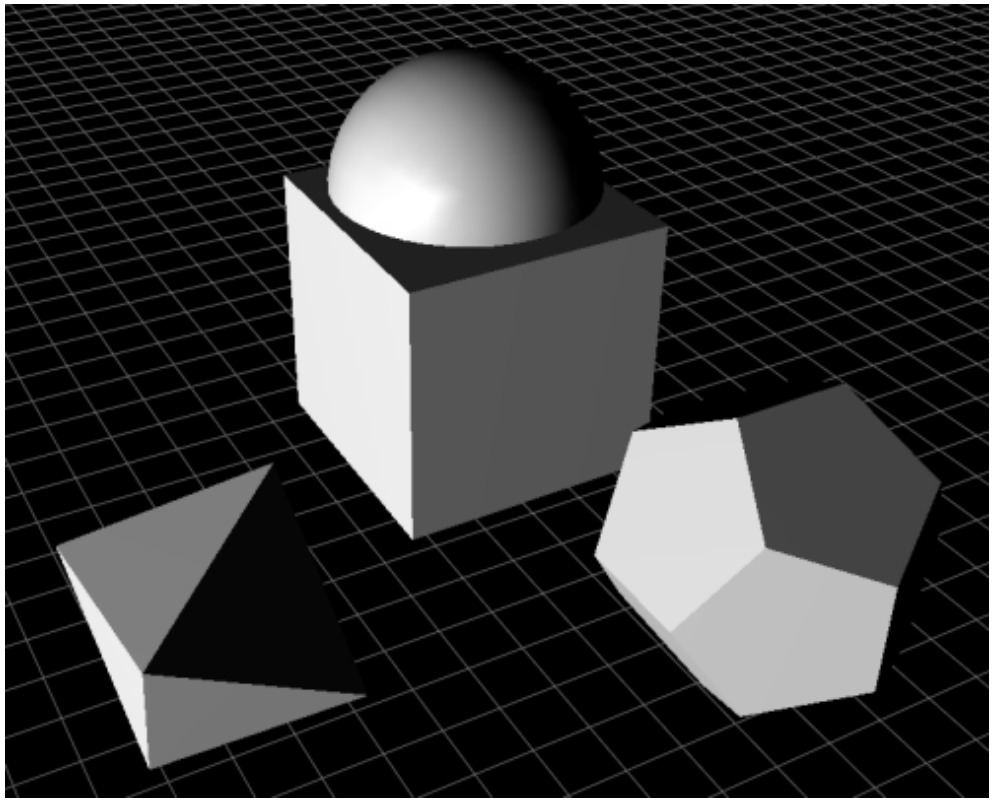
Basic Mouse Control

UserLButtonDown(…)	UserMouseMove(…)	UserLButtonUp(…)
UserRButtonDown(…)		UserRButtonUp(…)
Setup Editing Mode	*Do Transformation*	*Exit Editing Mode*

Implementation Outline – Object transformation



Implementation Outline – Object transformation



Implementation Outline – Interface of UserMouseMove()

- Input
  - View information
    - pDoc->m_camera.viewpoint
  - 2D screen position
    - Start mouse position (Sstart): m_nSx , m_nSy
    - Current mouse position(Scurr): Function parameters x,y

- To-do
  - Form the transformation matrix and multiply it to each selected object.

Implementation Outline – Conventions in this template

- We always use the start condition in *mousedown.*

- Model's transform matrix is saved on *mousedownevent.*

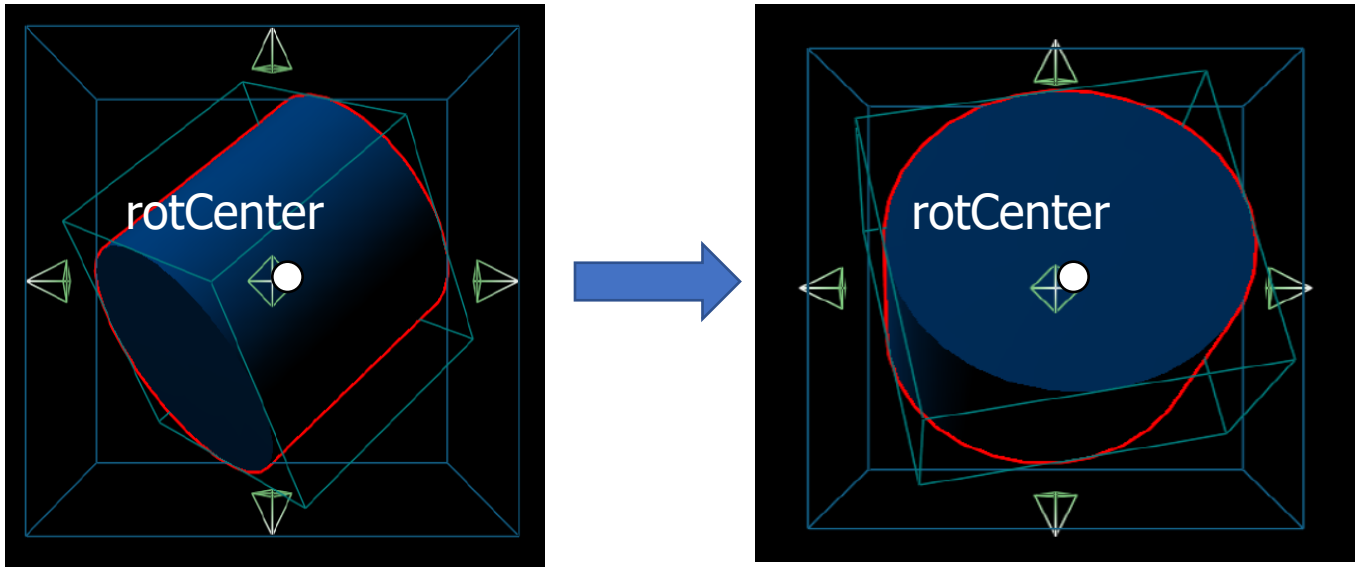
  ```
  pDoc->SelectedObjectsBackupM();
  //backup all the matrices of selected objects
  ```

- Model's transform matrix should be restored on *mousemoveevent.* This is to ensure the object is unchanged if mouse returns to its start condition.

  ```
  pDoc->SelectedObjectsRestoreM();
  //restore all the matrices of selected objects
  ```

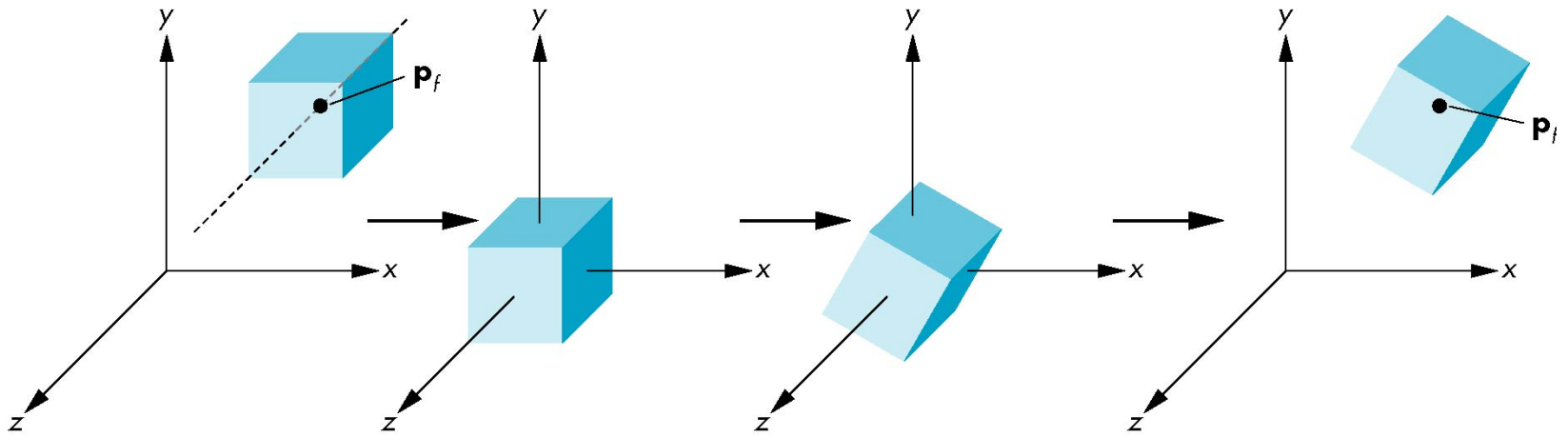
Object Rotation

- rotCenter denotes the rotation center.
- rotCenter is a class member variable. You may use it directly.
- Rotation angle is proportional to the amount of mouse motion.



Object Rotation – Rotation About a Fixed Point

- Move fixed point P_f to origin
- Rotate around the origin
- Move fixed point P_f back
- $\mathbf{M} = \mathbf{T}(p_f)\ \mathbf{R}\ \mathbf{T}(-p_f)$



Object Rotation – Form the transformation matrix

- *rotCenter* is the fixed point. The overall transformation is

$$\begin{pmatrix} 1, 0, 0, rotCenter[0] \\ 0, 1, 0, rotCenter[1] \\ 0, 0, 1, rotCenter[2] \\ 0, 0, 0, \quad 1 \end{pmatrix} \begin{pmatrix} \\ \quad R \\ \\ \end{pmatrix} \begin{pmatrix} 1, 0, 0, -rotCenter[0] \\ 0, 1, 0, -rotCenter[1] \\ 0, 0, 1, -rotCenter[2] \\ 0, 0, 0, \quad 1 \end{pmatrix}$$

- Use CPrimitive's *MultM(N)* function to do the matrix multiplications. Note that MultM is a left matrix multiplication, i.e. M = N*M

Object Rotation - Sample Code

```
float tmat[16]={.0f};   float itmat[16]={.0f};   float rmat[16]={.0f};
tmat[0]  = tmat[5]  =  tmat[10] =  tmat[15] = 1.0f;
itmat[0] = itmat[5] = itmat[10] = itmat[15] = 1.0f;
tmat[12] = rotCenter[0]; tmat[13] = rotCenter[1]; tmat[14] = rotCenter[2];
itmat[12]=-rotCenter[0]; itmat[13]=-rotCenter[1]; itmat[14]=-rotCenter[2];

//get rmat

for(list<LPPRIMITIVE>::iterator it=pDoc->m_selectedobjects.begin();
   it!=pDoc->m_selectedobjects.end(); it++)
   {        (*it)->MultM(itmat);
            (*it)->MultM(rmat);
            (*it)->MultM(tmat);
   }
```

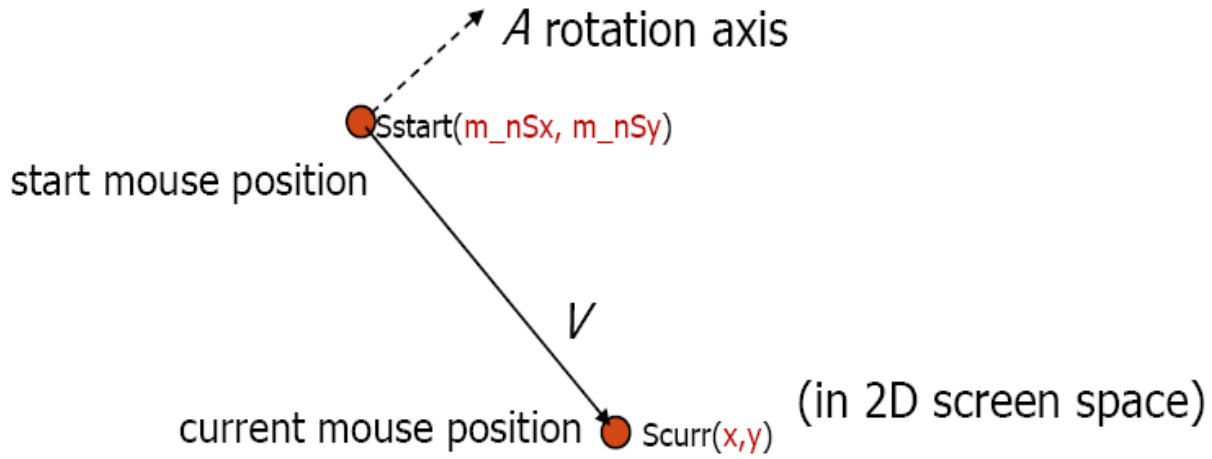
$$\begin{pmatrix} 1, 0, 0, rotCenter[0] \\ 0, 1, 0, rotCenter[1] \\ 0, 0, 1, rotCenter[2] \\ 0, 0, 0, 1 \end{pmatrix}$$

tmat

$$\begin{pmatrix} 1, 0, 0, -rotCenter[0] \\ 0, 1, 0, -rotCenter[1] \\ 0, 0, 1, -rotCenter[2] \\ 0, 0, 0, 1 \end{pmatrix}$$

itmat

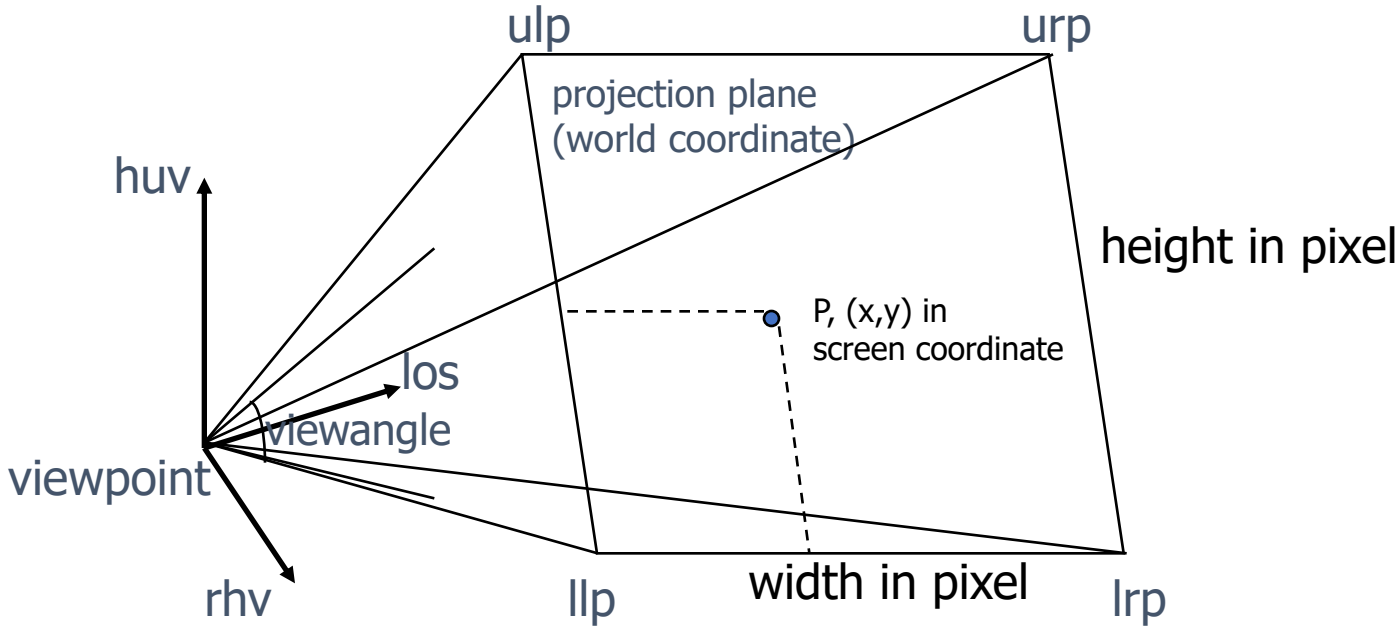
Object Rotation – Find the rotation axis & rotation angle



- 2D vector V=Scurr - Sstart
- Rotate V by 90 degree to get vector A(Ax, Ay), here Ax=-Vy, Ay=Vx.
- Project Sstart and Sstart +A to world coordinates to get Sstart'and (Sstart +A)' with function 'ProjectScreenPoint'.
- Rotation axis *ra= (Sstart +A)'-Sstart'*
- Rotation angle = $k \times |A| = k \times \sqrt{Ax^2 + Ay^2}$

  (assign k by yourself, suggested value: 0.5)

Project screen coordinates to world coordinates



huv -- head up vector (unit vector)
los -- line of sight (unit vector)
rhv -- right hand vector (unit vector)

Calculate the corresponding world coordinate P' of a screen position P

P'=llp + (lrp-llp)*x/width+(ulp-llp)*y/height

Using Function ProjectScreenPoint(x,y).

Object Rotation – Finding the Rotational Matrix R

- Use OpenGL library to compute R

```
glPushMatrix();
glLoadIdentity();
//angle is the rotation angle, ra is the rotation axis
glRotatef(angle,ra[0],ra[1],ra[2]);
glGetFloatv(GL_MODELVIEW_MATRIX, R);
glPopMatrix();
```

Object Rotation - Procedure

Step 1 : Find the rotation axis and determine the rotation degree

Step 2 : Get the rotation matrix about the origin

Step 3 : Restore all the matrices of selected objects

Step 4 : For each selected object:

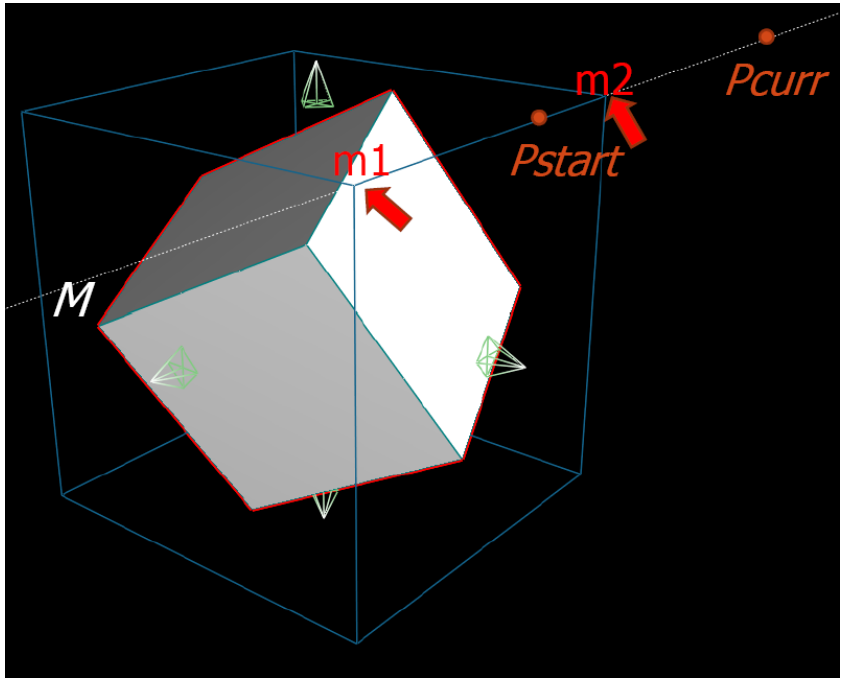
(a) Translate the rotation center to the origin

(b) Rotate round the origin

(c) Translate the rotation center back

Object Translation

- Translate along the user selected line M, with two endpoints m1, m2.

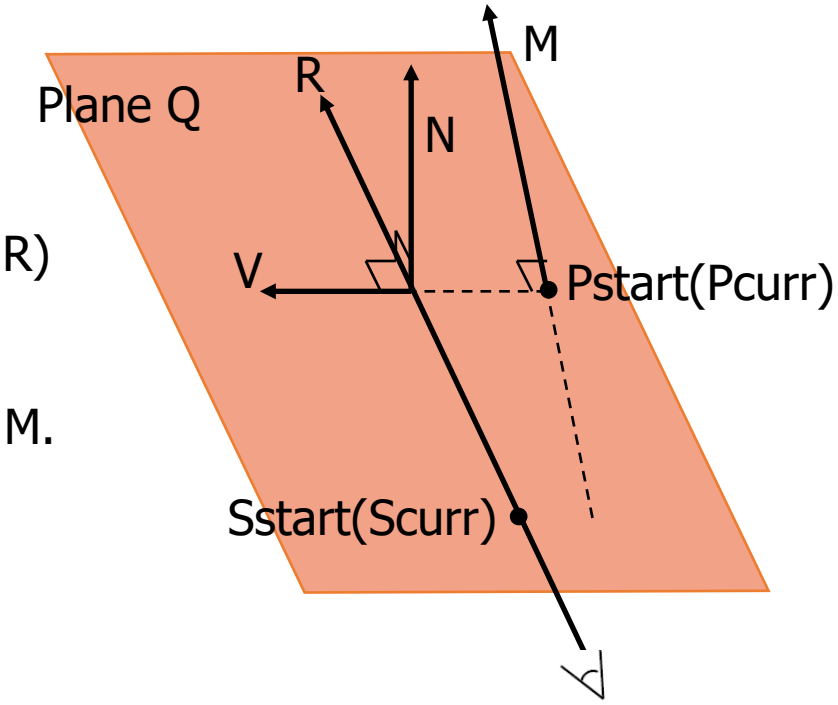


$$T = \begin{pmatrix} 1 & 0 & 0 & Pcurr[0] - Pstart[0] \\ 0 & 1 & 0 & Pcurr[1] - Pstart[1] \\ 0 & 0 & 1 & Pcurr[2] - Pstart[2] \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Object Translation – Find the start & current position on M

- Find Pstart and Pcurr corresponding to Sstart and Scurr respectively.
- Pstart and Pcurr are on the user selected line M.
- Pstart (Pcurr) is the closest point to ray R that goes through Sstart (Scurr).

V = cross(M, R) (V means the shortest path between M and R)
N = cross(R, V)
Plane Q can be defined by N and viewpoint.
Pstart (or Pcurr) is the intersection point of Plane Q and line M.



Object Translation - Procedure

Step 1: For the start mouse position Sstart, find Pstart on M

Step 2: For the current mouse position Scurr, find Pcurr on M

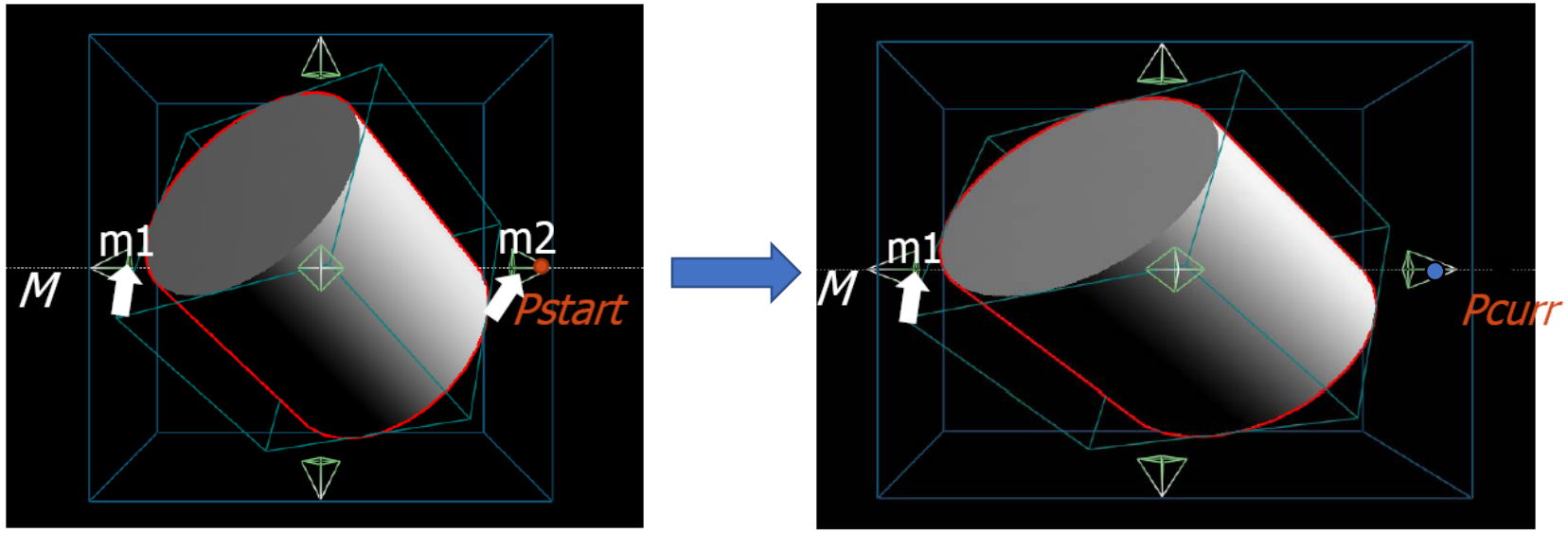
Step 3: Create the translation matrix

Step 4: Restore all the matrices of selected objects

Step 5: For each selected object,
do the translation.

Object Scaling (Axis aligned)

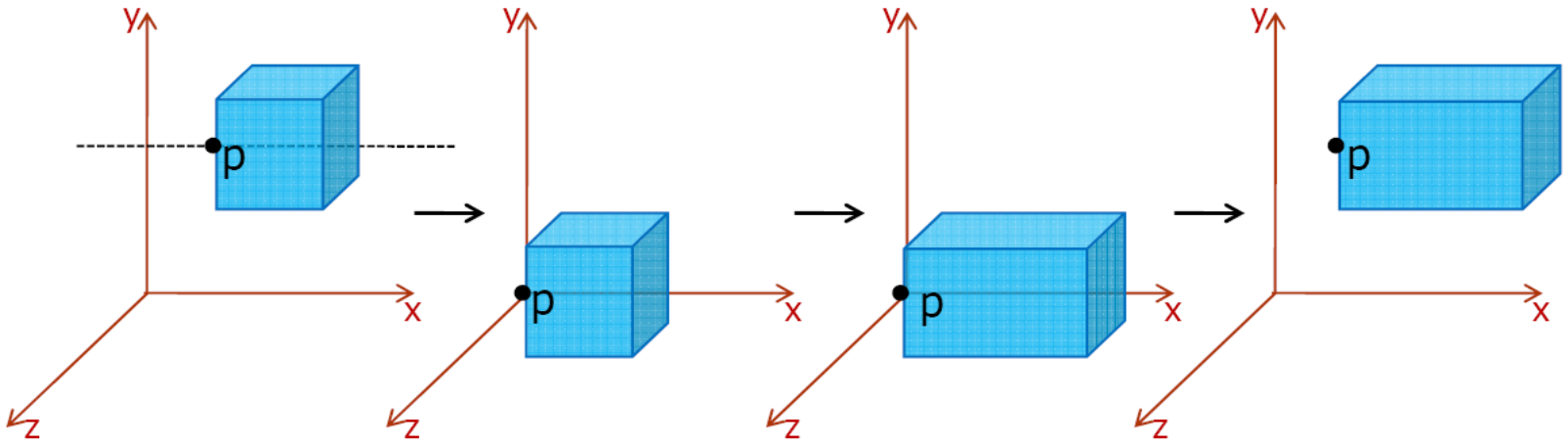
- Scale along the user selected line M, denoted by two points m1, m2. (here, M is one of the world axes.)
- Scale about the fixed point m1.



Object Scaling – Scaling About a Fixed Point P

- Move P to the origin
- Scale the object
- Move P back

Transformation Matrix = T(p) S T(-p)



Object Scaling – Form the transformation matrix

- m1 is the fixed point. The overall transformation is

$$\begin{bmatrix} 1 & 0 & 0 & m1[0] \\ 0 & 1 & 0 & m1[1] \\ 0 & 0 & 1 & m1[2] \\ 0 & 0 & 0 & 1 \end{bmatrix} \times S \times \begin{bmatrix} 1 & 0 & 0 & -m1[0] \\ 0 & 1 & 0 & -m1[1] \\ 0 & 0 & 1 & -m1[2] \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Use CPrimitive MultM(N) function to do the matrix multiplications. Note that MultM is a left matrix multiplication, i.e. M:=N*M.

Finding the Scaling Matrix S

- ScaleDir denotes the scaling direction.
- ScaleDir is a class member variable. You may use it directly.
- Compute the scaling value by:

$$svalue = \frac{(Pcurr - m1).length()}{(Pstart - m1).length()}$$

If scaleDir = 2:

$$\begin{bmatrix} Svalue & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If scaleDir = 0:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & Svalue & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

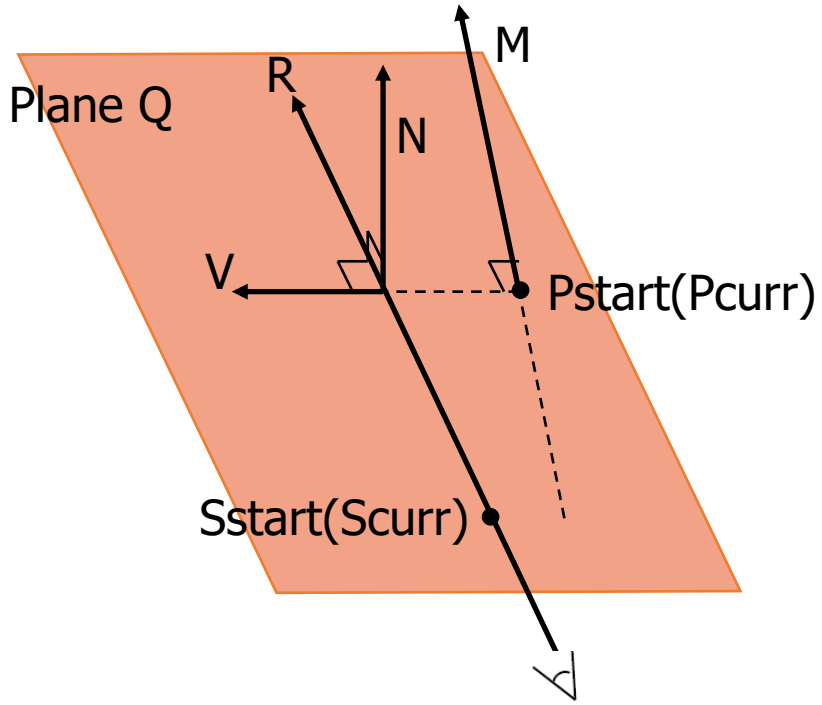
If scaleDir = 1:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & Svalue & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Object Scaling – Find the start & current position on M

- Find Pstart and Pcurr corresponding to Sstart and Scurr respectively.
- Pstart and Pcurr are on the user selected line M.
- Pstart (Pcurr) is the closest point to ray R that goes through Sstart (Scurr).

V = cross(M, R)
N = cross(R, V)
Plane Q can be defined by N and viewpoint.
Pstart (or Pcurr) is the intersection point of Plane Q and line M.



Object Scaling - Procedure

Step 1: For the start mouse position Sstart, find Pstart on M

Step 2: For the current mouse position Scurr, find Pcurr on M

Step 3: Determine the scaling value and create the scaling matrix

Step 4: Restore all the matrices of selected objects

Step 5: For each selected object:

(a) Translate the scaling fixed point to the origin.

(b) Do the scaling.

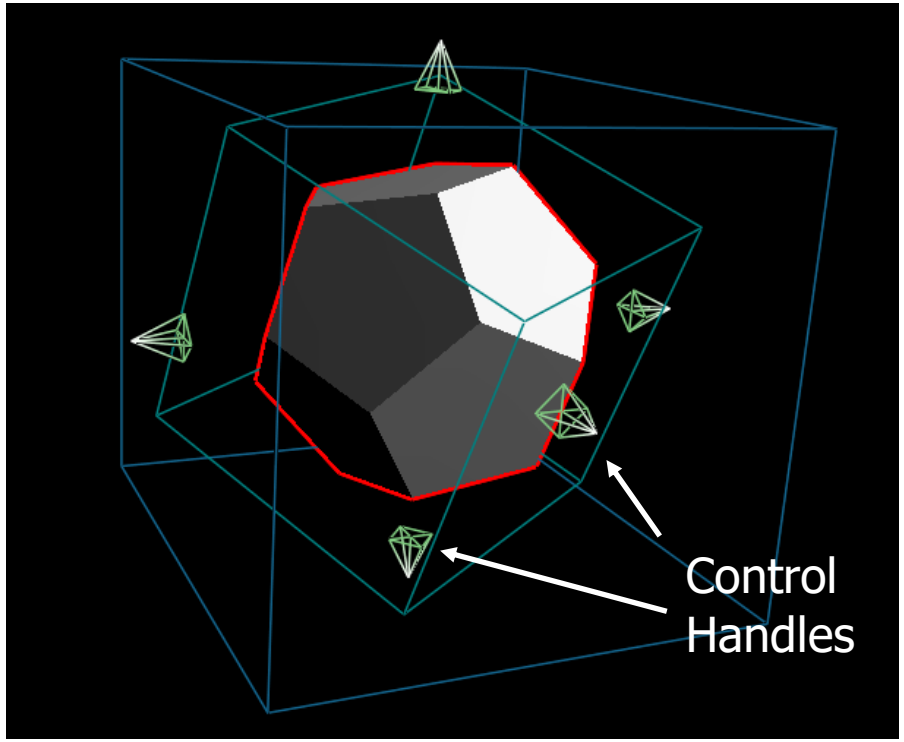
(c) Translate the scaling fixed point back.

Outline

- 3D Modeling & 3D Rendering in OpenGL
- 3D Modeling & Interface Design – Programming Assignment
  - About the Template
  - About the Task
    - Mouse controlled model editing
    - 3D solid handles

Control Handle

- Transformation modes are determined by clicking of different control handles



Drawing control handles

- UserDrawControlHandle(…)
- Called in each frame to draw handles.

Add codes to function “UserDrawControlHandle” in file “rayView.cpp”

```
void CRayView::UserDrawControlHandle(V3 origin, V3 x-axis, V3 y-axis, V3 z-axis)
{  //The parameters form a local coordinate system.

   //Calculate Point Coordinates for the designed handle object.

   //Draw Designed 3D Handle,
}
```

Provided Vector Operations

- Declaration: V3 v0,v1,v2;
- Normalize : v0.normalize();
- Length : v0.length();
- Dot product : float x=v0.dot(v1);
- Cross product : v2=v0.cross(v1);
- … See more in “v3.h”

Some Notes

- Matrix defined in OpenGL is like:

$$M=\begin{pmatrix} m_0, & m_4, & m_8, & m_{12} \\ m_1, & m_5, & m_9, & m_{13} \\ m_2, & m_6, & m_{10}, & m_{14} \\ m_3, & m_7, & m_{11}, & m_{15} \end{pmatrix}$$

- The origin of the screen coordinate you get is at the bottom left corner.

Marking Scheme

- Model Transformation
  - Translation (axis aligned)
  - Rotation (Trackball style)
  - Scaling (axis aligned)
- Handle Design
- Programming Structure

Hand-in

- Submit your rayView.cpp file through the Web-handin.

- Late Policy
  - 50% off for the delay of each day.
  - Re-submission after deadline is treated as late submission.
- NO PLAGIARISM!

References

- OpenGL Official Site
  http://www.opengl.org

- GLU reference
  http://pyopengl.sourceforge.net/documentation/manual/reference-GLU.html

- Edward Angel, Interactive Computer Graphics, a top-down approach with OpenGL(5th edition), Addison Wesley, 2009

- Jackie Neider, Tom Davis, Mason Woo, OpenGL Programming Guide, Addison Wesley, 1996

Q & A!