COMP3271 Tutorial

Programming Assignment II

Tutorial on 3D Modeling

Submission

- Deadline:

  11:59 PM Nov 04, 2022

- Submission:

MainFrame.cpp

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

Outline

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

Outline

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

Draw Geometric Objects – 3D

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

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

- Example1: draw a triangle in 3D space

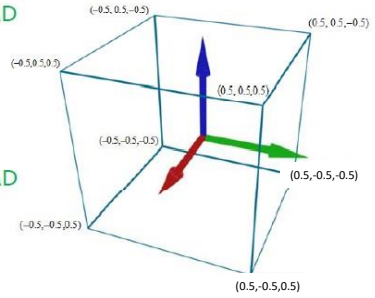
```
glBegin(GL_TRIANGLES);     // Specify object type

    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();
```

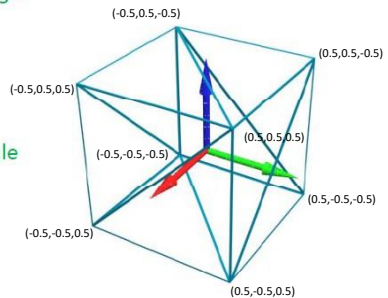


Draw Geometric Objects – 3D

- Example2: draw a cube

Solution 2: Use GL_TRIANGLES to draw a box.

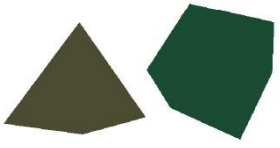
```
glBegin(GL_TRIANGLES);
glVertex3f(-0.5f, -0.5f, -0.5f); //1st triangle
glVertex3f(-0.5f, 0.5f, -0.5f);
glVertex3f(0.5f, 0.5f, -0.5f);
... ...

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



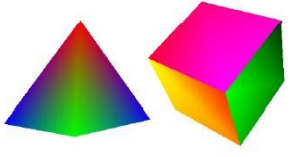
Draw Geometric Objects – 3D

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();
```

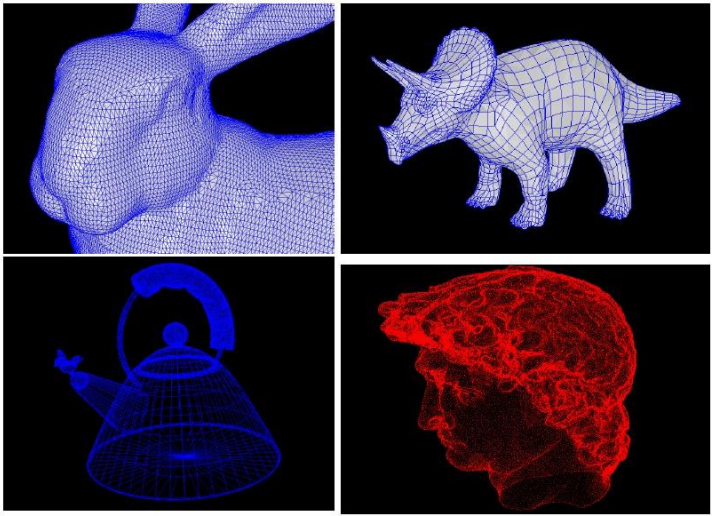


Vertices with the same color



Vertices with different colors

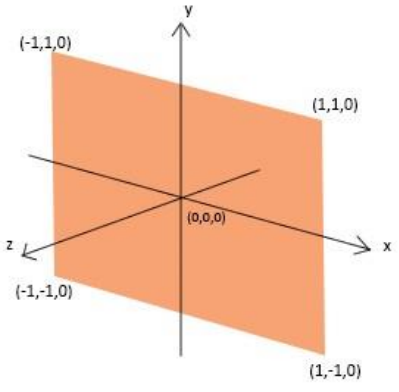
Draw Geometric Objects – 3D



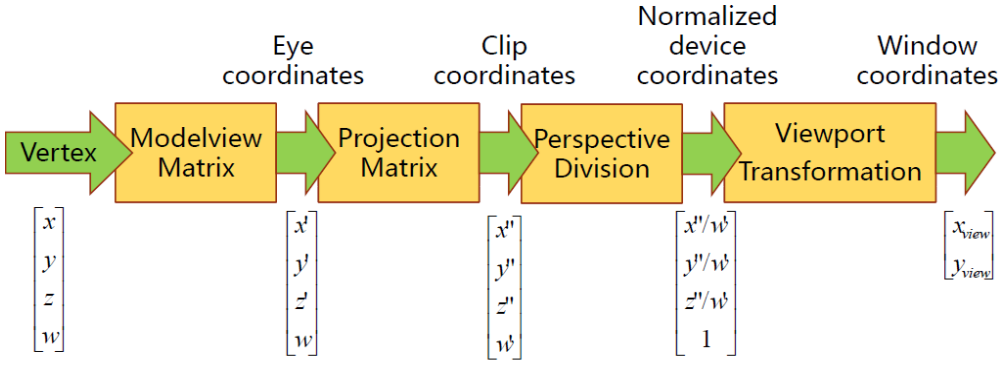
3D Rendering

- Coordinate system

1. Object or model coordinates
2. World coordinates
3. Eye (or Camera) coordinates
4. Clip coordinates
5. Normalized device coordinates
6. Window (or screen) coordinates



3D Rendering



3D Rendering

- Choose the transformation matrix

Projection matrix

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

glOrtho, glm::perspective, ...

Modelview matrix

Define the transformation of the camera and objects.

glTranslate, glRotate, glScale

glm::lookAt, glm::translate, glm::rotate, glm::scale

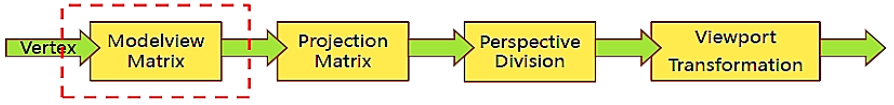
Choose the matrix to operate with:

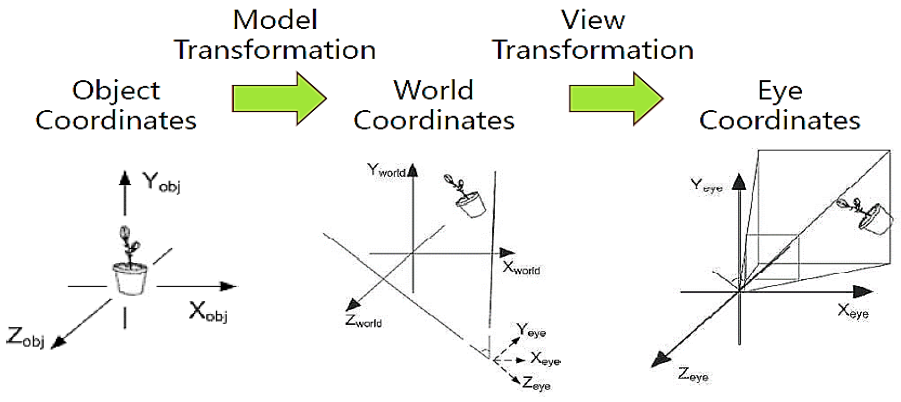
void glMatrixMode(MatrixType);

GL_PROJECTION: Choose projection matrix

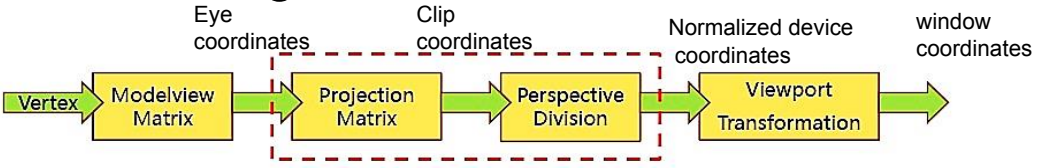
GL_MODELVIEW: Choose model view matrix

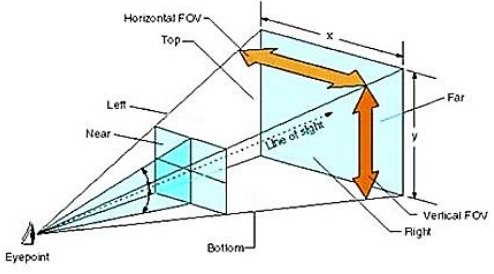
3D Rendering

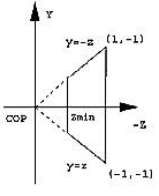


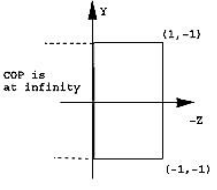


3D Rendering

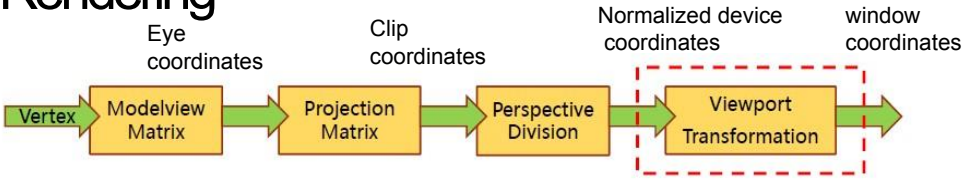


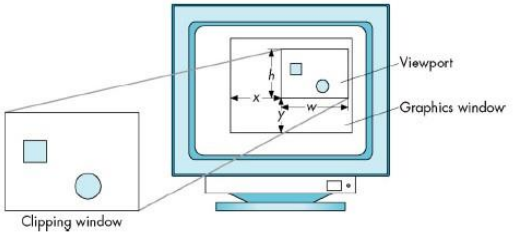






3D Rendering





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).

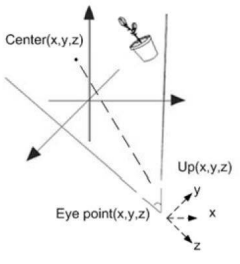
OpenGL will compute the result 2D image

3D Rendering

Specify the extrinsic parameters of camera

glm::lookAt(glm::vec3 eye, glm::vec3 center, glm::vec3 up);

Returns a 4x4 transformation matrix.
eye: Position of the eye
center: Position where the camera is looking at.
up: Direction of upvector.



3D Rendering

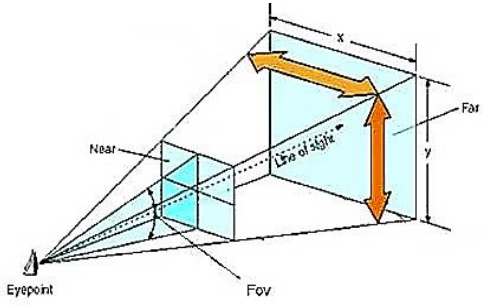
- **Define the viewing volume**

Perspective projection

glm::perspective(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).

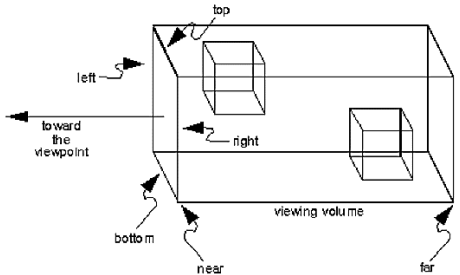


3D Rendering

- **Define the viewing volume**

  Orthogonal projection

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



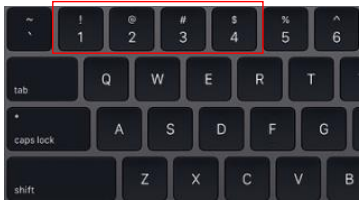
Outline

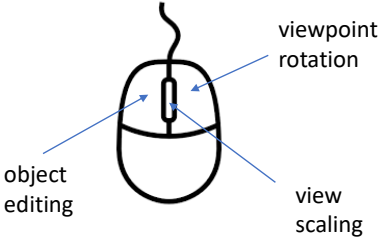
- 3D Rendering in OpenGL
- 3D Modeling & Interface Design – Programming Assignment 2
  - About the Template
  - About the Task
    - Mouse controlled model editing
    - Word space visualization

Template Provided Functions

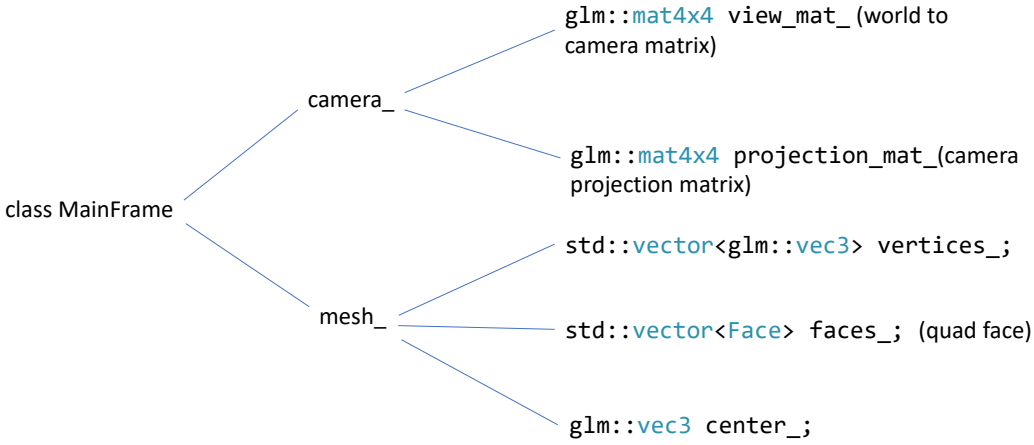
- Simple 3D primitive rendering
- Face subdivision
- Navigate editing mode by the number keys

  (1: rotation, 2: translation, 3: face subdivision, 4: extrusion)
- Viewpoint rotation / view scaling





Template Structure



*Refer to the headers for more information

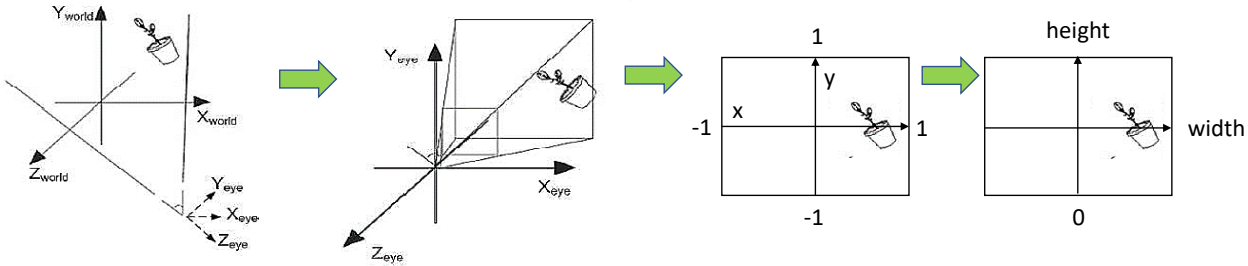
Transformation Conventions

View transformation

camera_.view_mat_

Projection transformation

camera_.projection_mat_



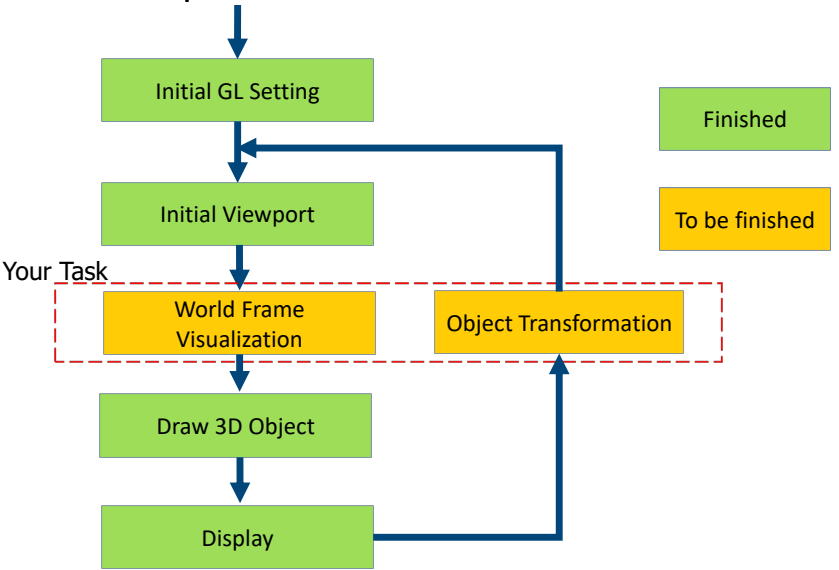
World coordinates

Eye coordinates / Camera coordinates

NDC coordinates

Screen coordinates

About the Template



Outline

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

Your Tasks

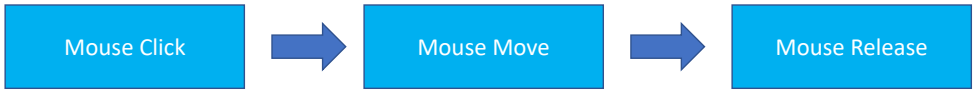
- Mouse controlled model editing (in '*LeftMouseMove*' function*)
  - Object Translation (View plane aligned)
  - Object Rotation (Trackball style)
  - Object Extrusion (Face normal aligned)
- 3D world space visualization (in 'VisualizeWorldSpace' function*)

*Both functions are in 'MainFrame.cpp'

Outline

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

Basic Mouse Control



LeftMouseClick(…)

Setup Editing Mode

LeftMouseMove(…)

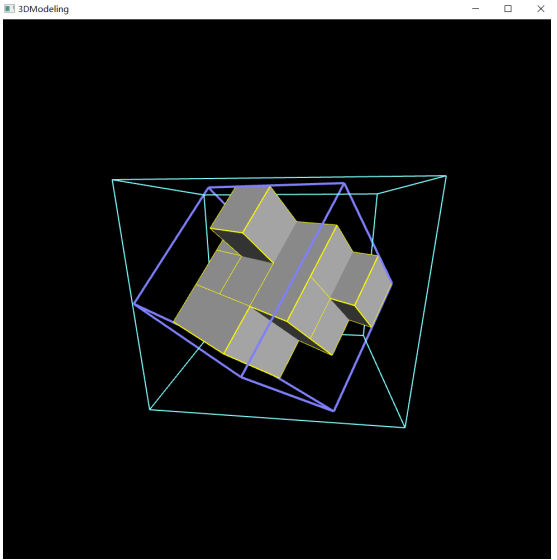
Apply Transformation

To be finished

LeftMouseRelease(…)

Commit Transformation *&*
Exit Editing Mode

Implementation Outline – 3D object modeling



Implementation Outline – Interface of LeftMouseMove()

- Input
  - Object information
    - mesh_ (class member)
  - View information
    - camera_ (class member)
  - 2D screen position
    - Start mouse position in screen coordinate (Sstart): start_x , start_y
    - Current mouse position in screen coordinate (Scurr): end_x, end_y

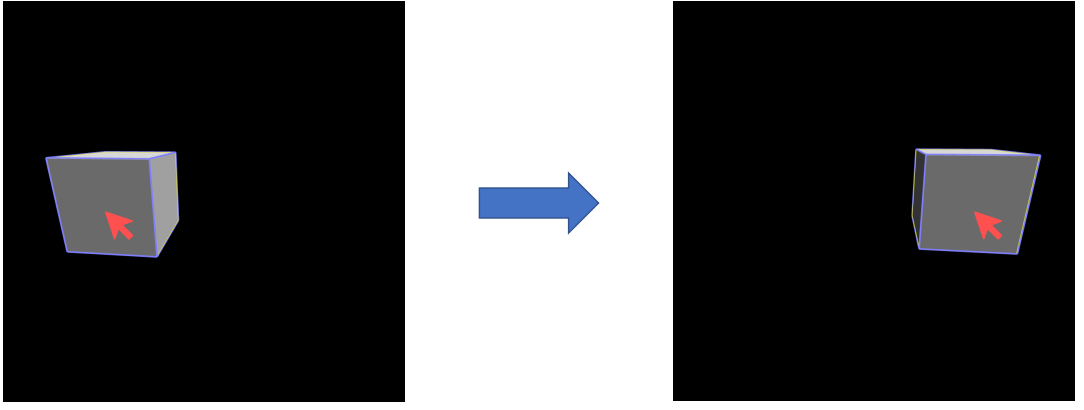
- To-do
  - Find the corresponding transformation matrices and multiply it to the object.

Your Tasks

- Mouse controlled model editing (in ‘LeftMouseMove’ function*)
  - Object Translation (View plane aligned)
  - Object Rotation (Trackball style)
  - Object Extrusion (Face normal aligned)
- World space visualization (in ‘DrawWorldFrame’ function*)

*Both functions are in ‘MainFrame.cpp’

Task 1: Object Translation

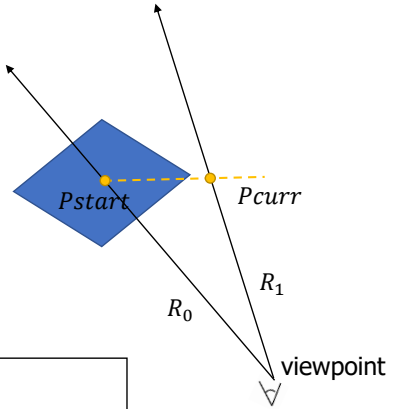


Object Translation – Find the start & current position

world coordinates screen coordinates

- Find Pstart and Pcurr corresponding to Sstart and Scurr respectively.
- Pstart is the intersected point of the object and the ray R_0 that goes through Sstart.
- Pcurr is on the ray R_1 that goes through Scurr.
- Pcurr and Pstart should have the same z-value in camera coordinates.

constraints



Some possible useful functions:

```
•glm::vec3 Screen2World(float scr_x, float scr_y, float camera_z);
•std::tuple<glm::vec3, glm::vec3> Screen2WorldRay(float scr_x, float scr_y);
•std::tuple<int, glm::vec3> Mesh::FaceIntersection(const glm::vec3& o, const glm::vec3& d)
```

Object Translation – Construct translation matrix

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

Or directly use glm::translate(glm::mat4x4(1.f), Pcurr - Pstart)

Identity matrix

Translation vector

Object Translation - Procedure

Step 1: For the start mouse position Sstart, find Pstart in world coordinate

Step 2: For the current mouse position Scurr, find Pcurr in world coordinate

Step 3: Create the translation matrix (Pstart -> Pcurr)

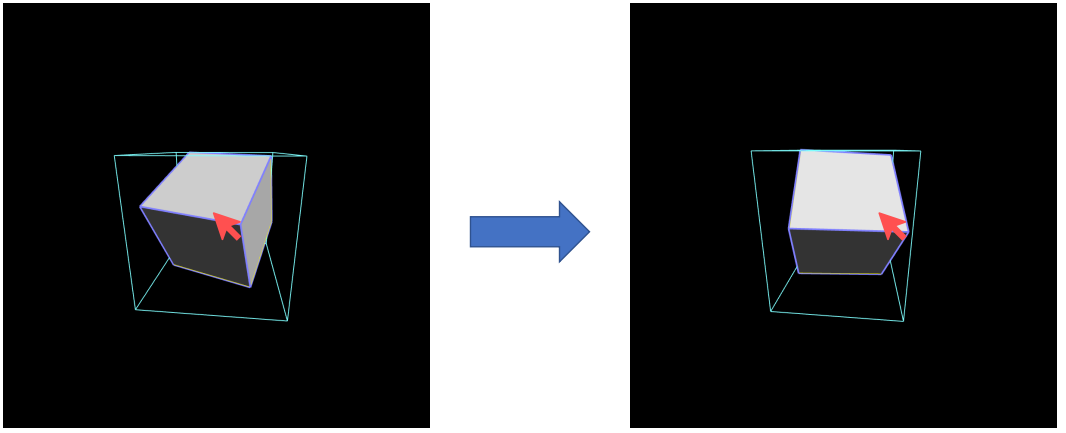
Step 4: Apply the transformation

Your Tasks

- Mouse controlled model editing (in ‘LeftMouseMove’ function*)
  - Object Translation (View plane aligned)
  - Object Rotation (Trackball style)
  - Object Extrusion (Face normal aligned)
- World space visualization (in ‘DrawWorldFrame’ function*)

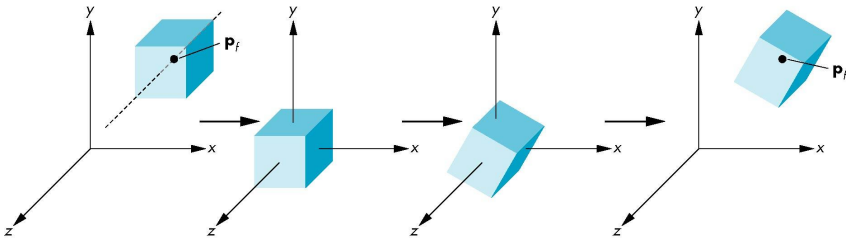
*Both functions are in ‘MainFrame.cpp’

Task 2: Object Rotation



Object Rotation – Rotation About a Fixed Point

- Move fixed point P to origin
- Rotate around the origin
- Move fixed point P back
- $M = T(P) * R * T(-P)$



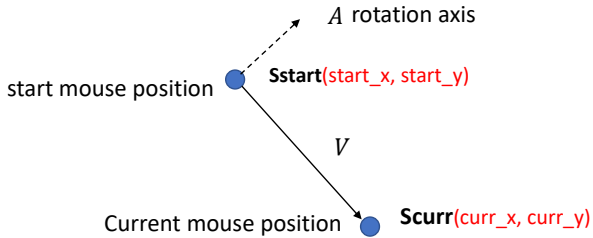
Object Rotation – Construct the transformation matrix

- *mesh_.center_* is the fixed point. The overall transformation is

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

- You may use *glm::mat4x4* to do the matrix multiplications.

Object Rotation – Find the rotation axis & rotation angle



(in 2D screen space)

- 2D vector V = Scurr – Sstart = (Vx, Vy)
- 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 'Screen2World'.
- Rotation axis *ra= normalize((Sstart +A)'-Sstart')* *(world space)*
- Rotation angle = $k \times |A| = k \times \sqrt{Ax^2 + Ay^2}$

  (assign proper constant k by yourself)

Object Rotation – Construct the Rotational Matrix

1. Use OpenGL library to compute R:

```
float R[16];
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();
```

2. Or use GLM to compute R:

```
glm::mat4x4 R;
R = glm::rotate(glm::mat4x4(1.f), angle, ra);
```

Object Rotation - Procedure

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

Step 2 : Get the rotation matrix about the rotation axis

Step 3 : For the current object:

(a) Translate the rotation center to the origin

(b) Rotate round the origin

(c) Translate the rotation center back

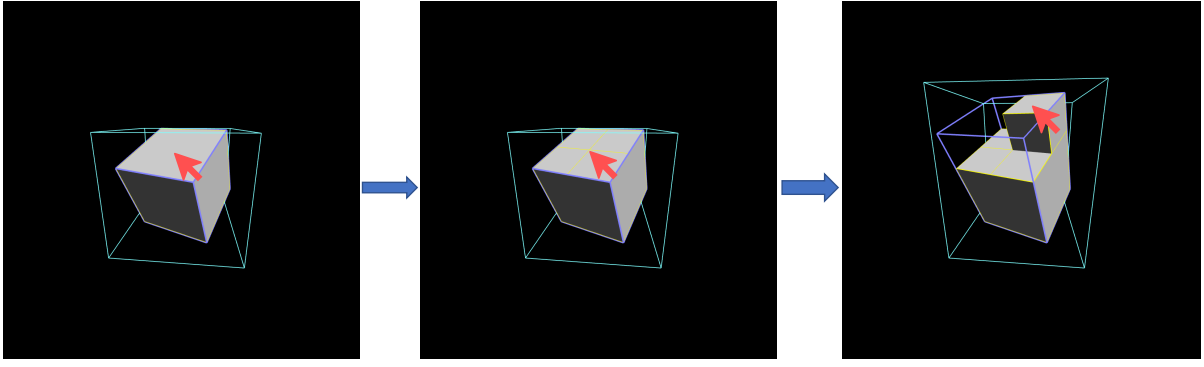
Step 4: Apply the transformation

Your Tasks

- Mouse controlled model editing (in ‘LeftMouseMove’ function*)
  - Object Translation (View plane aligned)
  - Object Rotation (Trackball style)
  - Object Extrusion (Face normal aligned)
- World space visualization (in ‘DrawWorldFrame’ function*)

*Both functions are in ‘MainFrame.cpp’

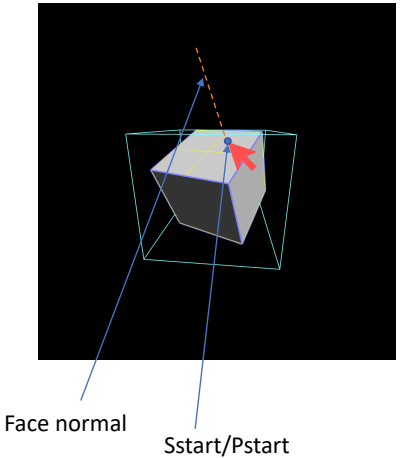
Task 3: Object Extrusion

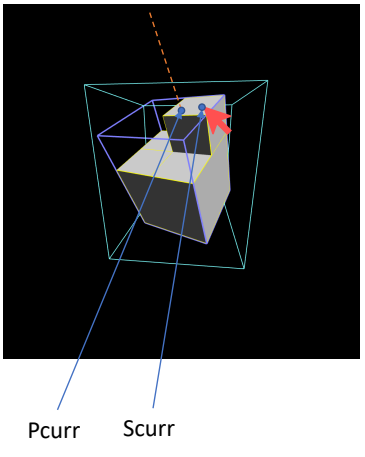


Face subdivision (provided)

Face extrusion (to be finished)

Object Extrusion – Find the start & current position

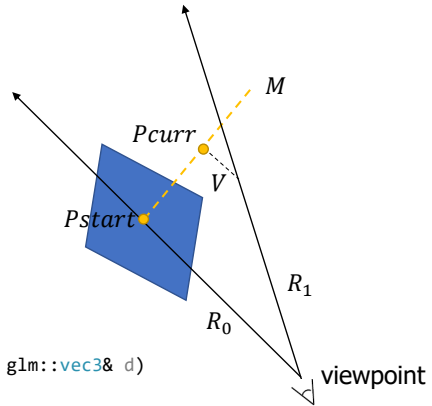




Object Extrusion – Find the start & current position

- Find Pstart and Pcurr corresponding to Sstart and Scurr respectively.
- Pstart is the intersected point of the object and the ray R_0 that goes through Sstart.
- Pcurr is on the ray $M = Pstart + t \cdot N$, where N is the normal vector of the intersected face.
- Pcurr is the closest point to ray R_1 that goes through Scurr.

$V = cross(M, R)$ (V means the shortest path between M and R_1)
Plane Q can be defined by V and ray R_1.
$Pcurr$ is the intersection point of Plane Q and line M.



Some possible useful functions:

```
•glm::vec3 Screen2World(float scr_x, float scr_y, float camera_z);
•std::tuple<glm::vec3, glm::vec3> Screen2WorldRay(float scr_x, float scr_y);
•std::tuple<int, glm::vec3> Mesh::FaceIntersection(const glm::vec3& o, const glm::vec3& d)
```

Object Extrusion - Procedure

Step 1: For the current intersected face by ray R_0, calculate the normal N

Step 3: Calculate $Pcurr$

Step 3: Create the translation matrix ($Pstart$ -> $Pcurr$)

Step 4: Apply the transformation

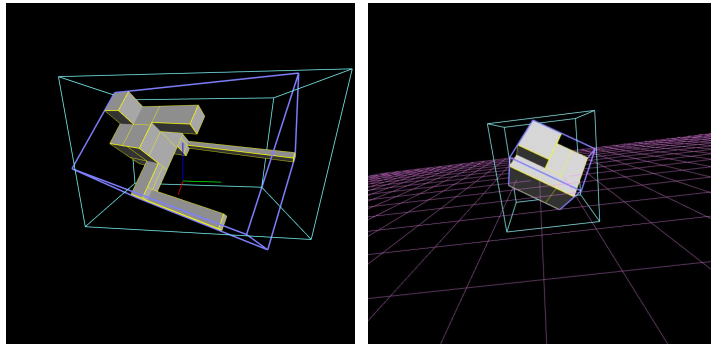
Outline

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

World Space Visualization

- VisualizeWorldSpace(…)
- Called in each time of rendering.

Add codes to function “VisualizeWorldSpace” in file “MainFrame.cpp”



Examples

GLM(OpenGL Mathematics) operations

- Vectors: glm::vec3, glm::vec4, …
- Normalize : glm::normalize(…)
- Length : glm::length(…);
- Dot product : glm::dot(…);
- Cross product : glm::cross(…);
- Matrices: glm::mat4x4, …
- Matrices multiplication: operator *
- Matrices inverse: glm::inverse(…);
- …

Some Notes

- Memory layout of matrices in OpenGL is column-major (GLM is the same):

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

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

Marking Scheme

- Model Transformation
  - Translation (view plane aligned)
  - Rotation (trackball style)
  - Extrusion (face normal aligned)
- World Space Visualization
- Programming Structure

Hand-in

- Submit your ***MainFrame.cpp*** file through Moodle.

- 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

- GLM reference
- http://glm.g-truc.net/0.9.8/api/index.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