

HW1 Tutorial

Outline

Section 1. BASIC GLUT - About `main()` function in `basicDraw.cpp`

1-1 Document

1-2 OpenGL Architecture

1-3 Initialization and window

1-4 Callback Registration

1-5 Geometric Object Rendering

1-6 Beginning Event Processing

Outline

Section 2. BASIC DRAW - About `display()` function in `basicDraw.cpp`

2-1 Overview – Code explanation

2-2 Clear color buffer & Clear depth buffer

2-3 Enable depth test

2-4 Draw the triangles

2-5 Complete drawing

2-6 Face culling

2-7 More Information about OpenGL ...

Outline

Section 3. BASIC DRAW - How to rotate the planet in OpenGL?

3-1 Overview

3-2 ModelView Matrix

3-3 Modeling Transformation

3-4 Projection Matrix

3-5 Viewport Matrix

3-6 Matrix Stack Mechanism

3-7 Example

Reference

1.BASIC GLUT

About `main()` function in basicDraw.cpp

1-1 Document

➤ <https://www.opengl.org/resources/libraries/glut/spec3/spec3.html>

The OpenGL Utility Toolkit (GLUT) Programming Interface API Version 3

Mark J. Kilgard
Silicon Graphics, Inc.

OpenGL is a trademark of Silicon Graphics, Inc. X Window System is a trademark of X Consortium, Inc. Spaceball is a registered trademark of Spatial Systems Inc.

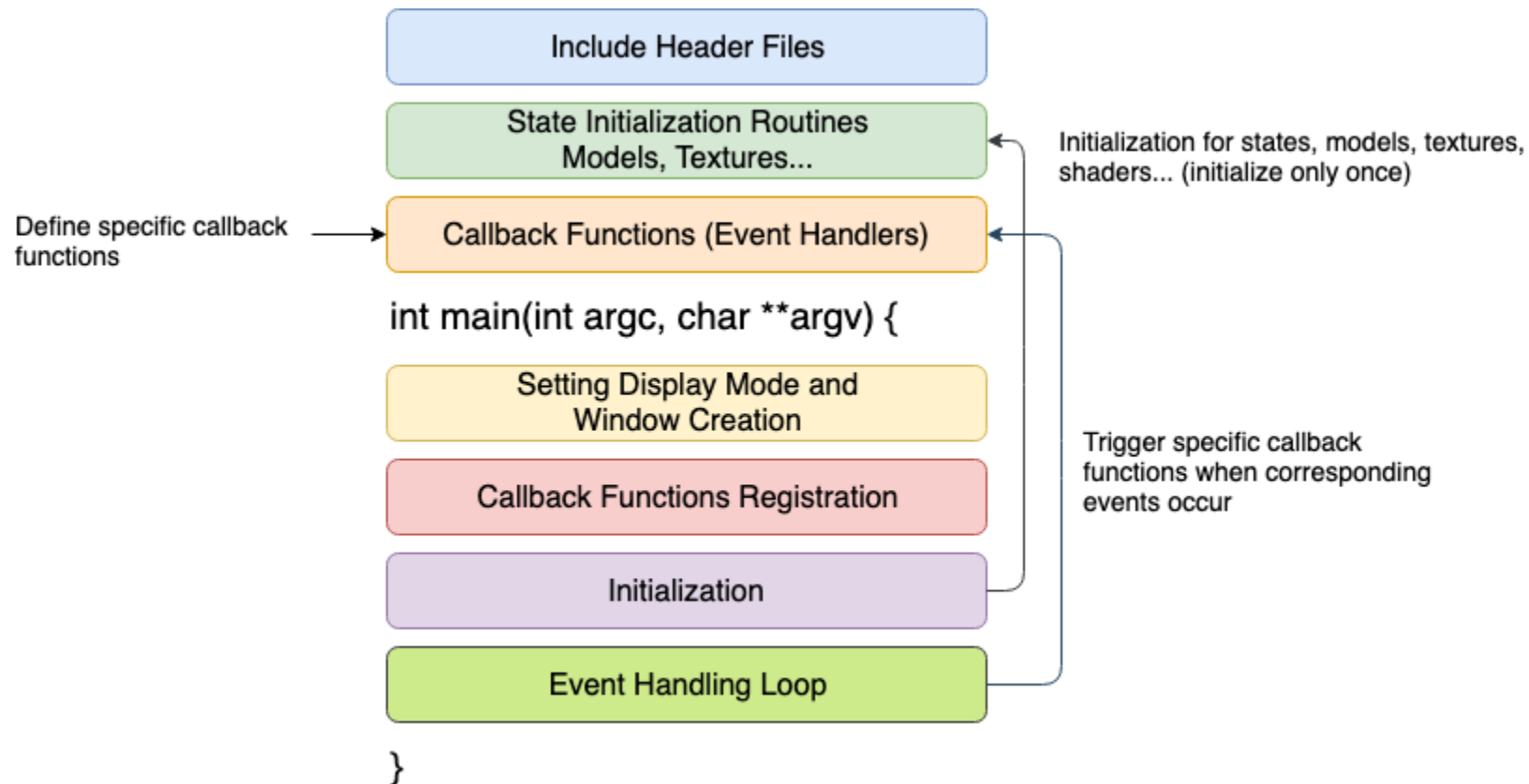
The author has taken care in preparation of this documentation but makes no expressed or implied warranty of any kind and assumes no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising from the use of information or programs contained herein.

Copyright © 1994, 1995, 1996. Mark J. Kilgard. All rights reserved.

All rights reserved. No part of this documentation may be reproduced, in any form or by any means, without permission in writing from the author.

-
- [Contents](#)
 - [1 Introduction](#)
 - [1.1 Background](#)
 - [1.2 Design Philosophy](#)
 - [1.3 API Version 2](#)
 - [1.4 API Version 3](#)
 - [1.5 Conventions](#)
 - [1.6 Terminology](#)
 - [2 Initialization](#)
 - [2.1 glutInit](#)
 - [2.2 glutInitWindowPosition, glutInitWindowSize](#)
 - [2.3 glutInitDisplayMode](#)
 - [3 Beginning Event Processing](#)
 - [3.1 glutMainLoop](#)
 - [4 Window Management](#)
 - [4.1 glutCreateWindow](#)
 - [4.2 glutCreateSubWindow](#)
 - [4.3 glutSetWindow, glutGetWindow](#)
 - [4.4 glutDestroyWindow](#)
 - [4.5 glutPostRedisplay](#)
 - [4.6 glutSwapBuffers](#)

1-2 OpenGL Architecture



1-3 Initialization and window

- void **glutInit**(int* argc, char **argv);
 - Initialize the GLUT library
 - Should be called before any GLUT functions
- void **glutInitDisplayMode**(unsigned int mode); (Red are commonly used parameters)
 - Specify a display mode for windows created
 - Color: **GLUT_RGBA**, GLUT_RGB or GLUT_INDEX
 - Framebuffer: GLUT_SINGLE or **GLUT_DOUBLE**
 - Buffer: GLUT_DEPTH, GLUT_STENCIL and GLUT_ACCUM

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("WindowName");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(mouseMotion);
    glutPassiveMotionFunc(passiveMouseMotion);
    glutIdleFunc(idle);

    glutMainLoop();

    return 0;
}
```


1-3 Initialization and window

- void **glutInitWindowSize**(int width, int height);
 - Set the initial window size
- void **glutInitWindowPosition**(int x, int y);
 - Set the initial window position
 - The actual position is left to the window system to determine
- int **glutCreateWindow**(char *name);
 - Create and open a window with previous settings

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("WindowName");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(mouseMotion);
    glutPassiveMotionFunc(passiveMouseMotion);
    glutIdleFunc(idle);

    glutMainLoop();

    return 0;
}
```

1-3 Initialization and window

- void **glutPostRedisplay()**;
 - Mark the current window as needing to be redisplayed
 - The window's display callback will be called
- void **glutSwapBuffers()**;
 - Swap the buffers of the current window
 - An implicit **glFlush()** is done by **glutSwapBuffers()**

```
void idle() {  
    glutPostRedisplay();  
}
```

```
glBegin(GL_TRIANGLES);  
//red triangle (z = 0)  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex3f(1.0f, 0.0f, 0.0f);  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex3f(0.0f, 1.0f, 0.0f);  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex3f(0.0f, 0.0f, 0.0f);  
//blue triangle (z = -1)  
glColor3f(0.0f, 0.0f, 1.0f);  
glVertex3f(1.0f, 0.0f, -1.0f);  
glColor3f(0.0f, 0.0f, 1.0f);  
glVertex3f(0.0f, 1.0f, -1.0f);  
glColor3f(0.0f, 0.0f, 1.0f);  
glVertex3f(0.0f, 0.0f, -1.0f);  
  
glEnd();  
// In display function  
glutSwapBuffers();
```

1-4 Callback Registration

- void **glutDisplayFunc**(void (*func)(void));
 - Put whatever you want to render in the callback
 - The callback is called when the window need to be redisplayed
 - Call **glutPostRedisplay()** to trigger the callback
- void **glutReshapeFunc**(void (*func)(int width, int height));
 - The callback is called when a window is created, resized or moved
 - Always call **glViewport()** to resize your viewport
- void **glutIdleFunc**(void (*func)(void));
 - Perform background processing tasks or continuous animation **when window system events are not being received**
 - The idle callback is continuously called when events are not being received

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("WindowName");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(mouseMotion);
    glutPassiveMotionFunc(passiveMouseMotion);
    glutIdleFunc(idle);

    glutMainLoop();

    return 0;
}
```

1-4 Callback Registration

- void **glutKeyboardFunc**(void (*func)(unsigned char key, int x, int y));
 - Each key press generates a keyboard callback
 - **key**: The ASCII character generated by the pressed key
 - x and y: The mouse location in window relative coordinates when the key was pressed
- void **glutMouseFunc**(void (*func)(int button, int state, int x, int y));
 - Each press and each release mouse button in a window generates a mouse callback
 - button: GLUT_LEFT_BUTTON, GLUT_MIDDLE_BUTTON or GLUT_RIGHT_BUTTON
 - state: GLUT_UP or GLUT_DOWN
 - x and y: The mouse location in window relative coordinates when the mouse was pressed

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("WindowName");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(mouseMotion);
    glutPassiveMotionFunc(passiveMouseMotion);
    glutIdleFunc(idle);

    glutMainLoop();

    return 0;
}
```

1-4 Callback Registration

- void **glutMotionFunc**(void (*func)(int x, int y));
 - The callback is called when the mouse moves within the window **while any mouse buttons are pressed**
 - x and y: the mouse location in window relative coordinates
- void **glutPassiveMotionFunc**(void (*func)(int x, int y));
 - The callback is called when the mouse moves within the window **while no mouse buttons are pressed**
 - x and y: the mouse location in window relative coordinates

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("WindowName");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(mouseMotion);
    glutPassiveMotionFunc(passiveMouseMotion);
    glutIdleFunc(idle);

    glutMainLoop();

    return 0;
}
```

1-5 Geometric Object Rendering

- void **glutSolidSphere**(Gldouble size, GLint slices, GLint stacks);
- void **glutWireSphere**(Gldouble size, GLint slices, GLint stacks);
- void **gluCylinder**(GLUquadric* quad, GLdouble base, GLdouble top, GLdouble height, GLint slices, GLint stacks);
- void **glutSolidCube**(Gldouble size); void **glutWireCube**(Gldouble size);
- void **glutSolidCone**(Gldouble size); void **glutWireCone**(Gldouble size);
- void **glutSolidTorus**(Gldouble size); void **glutWireTorus**(Gldouble size);
- void **glutSolidTeapot**(Gldouble size); void **glutWireTeapot**(Gldouble size);

1-6 Beginning Event Processing

- `void glutMainLoop();`
 - Enter the GLUT event processing loop
 - **Once called, this routine will never return**

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("WindowName");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMouseFunc(mouse);
    glutMotionFunc(mouseMotion);
    glutPassiveMotionFunc(passiveMouseMotion);
    glutIdleFunc(idle);

    glutMainLoop();

    return 0;
}
```

2.BASIC DRAW

About `display()` function in `basicDraw.cpp`

2-1. In the function **display()** of basicDraw.cpp,

We do the following things,

1. Set up the MVP matrix (Mentioned later in the Section. 3 OpenGL-Transform)
2. Before drawing, we need to clear the color buffer and depth buffer by **glClear()** with the values set by **glClearColor()** and **glClearDepth()**
3. Enable the option for depth test so the face behind other faces would not render in the front
4. Draw two triangles by **glColor3f()** and **glVertex3f()** between **glBegin()**, **glEnd()**
5. In the end, swap the buffers by **glutSwapBuffers()**

In the next pages, we will introduce the openGL functions used in these steps since step2. And if you want to know more detailed information about the functions, please go to the link:

<https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/>

```
17
//ModelView Matrix
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(0.0f, 0.0f, 10.0f, 0.0f, 0.0f, 0.0f, 0.0f, 1.0f, 0.0f);
//Projection Matrix
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45, width / (GLfloat)height, 0.1, 1000);
//Viewport Matrix
glViewport(0, 0, width, height);

//
glMatrixMode(GL_MODELVIEW);
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
glClear(GL_COLOR_BUFFER_BIT);
glClearDepth(1.0);
glEnable(GL_DEPTH_TEST); //depth test
glDepthFunc(GL_LEQUAL);
glClear(GL_DEPTH_BUFFER_BIT);

glBegin(GL_TRIANGLES);
//red triangle (z = 0)
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(1.0f, 0.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 1.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 0.0f, 0.0f);
//blue triangle (z = -1)
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(1.0f, 0.0f, -1.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(0.0f, 1.0f, -1.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(0.0f, 0.0f, -1.0f);

glEnd();

glutSwapBuffers();
```

STEP1

STEP2

STEP3

STEP4

STEP5

2-2. Clear color buffer & Clear depth buffer

Before we start a new render iteration, we need to clear our color buffer and depth buffer, otherwise you're stuck with the written color values and depth values from the last render iteration.

Here are the functions used for buffer cleaning:

- void **glClearColor**(GLfloat red,
GLfloat green,
GLfloat blue,
GLfloat alpha);

Specify the red, green, blue, and alpha values used when the color buffers are cleared. The initial values are all 0. (black)

- void **glClearDepth**(GLdouble depth);

Specify the depth value used when the depth buffer is cleared. The initial value is 1. The depth is in the range [0, 1].

```
//
glMatrixMode(GL_MODELVIEW);
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
glClear(GL_COLOR_BUFFER_BIT);
glClearDepth(1.0);
glEnable(GL_DEPTH_TEST);    //depth test
glDepthFunc(GL_LEQUAL);
glClear(GL_DEPTH_BUFFER_BIT);
```

- void **glClear**(GLbitfield mask);

Clear the specified buffers to their current clearing values selected by `glClearColor()`, `glClearDepth()`, and `glClearStencil()`.

mask: **GL_COLOR_BUFFER_BIT, GL_DEPTH_BUFFER_BIT,**
GL_STENCIL_BUFFER_BIT

EX:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

2-3. Enable depth test

In order to prevent faces rendering to the front while they're behind other faces, we need to enable depth test before drawing:

Here are the functions for enabling depth test:

➤ `glEnable(GL_DEPTH_TEST);`

When depth test is enabled, OpenGL tests the depth value of a fragment against the content of the depth buffer. If this test passes, the depth buffer is updated with the new depth value. If the test fails, the fragment is discarded.

➤ `void glDepthFunc(GLenum func);`

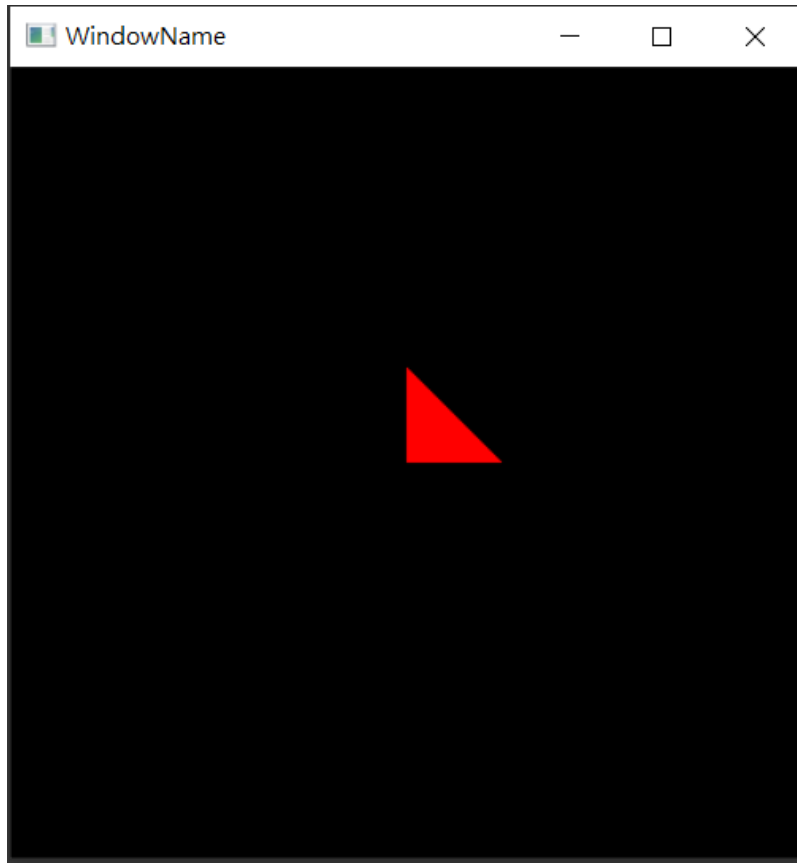
Specify the depth comparison function for the depth test.

func: GL_NEVER, GL_LESS, GL_EQUAL, GL_LEQUAL, GL_GREATER... The initial value is GL_LESS

Ex. `glDepthFunc(GL_LEQUAL);` =>

if (fragment's depth value <= stored depth value) pass.

```
//  
glMatrixMode(GL_MODELVIEW);  
glClearColor(0.0f, 0.0f, 0.0f, 0.0f);  
glClear(GL_COLOR_BUFFER_BIT);  
glClearDepth(1.0);  
glEnable(GL_DEPTH_TEST);      //depth test  
glDepthFunc(GL_LEQUAL);  
glClear(GL_DEPTH_BUFFER_BIT);
```



This is the output of the BasicDraw.cpp, what will happen if we don't enable the depth test?
(Try to delete the glEnable() and run the code basicDraw.cpp by yourself.)

If you are interested in more information about depth test:

<https://learnopengl.com/Advanced-OpenGL/Depth-testing>

2-4. Draw the triangles

By the help of OpenGL, we can draw our triangles by simply setting the vertex data **between glBegin() and glEnd()**

➤ void **glBegin**(GLenum mode);

Marks the beginning of a vertex-data list.

Vertex-data include vertex's color, normal, position, etc.

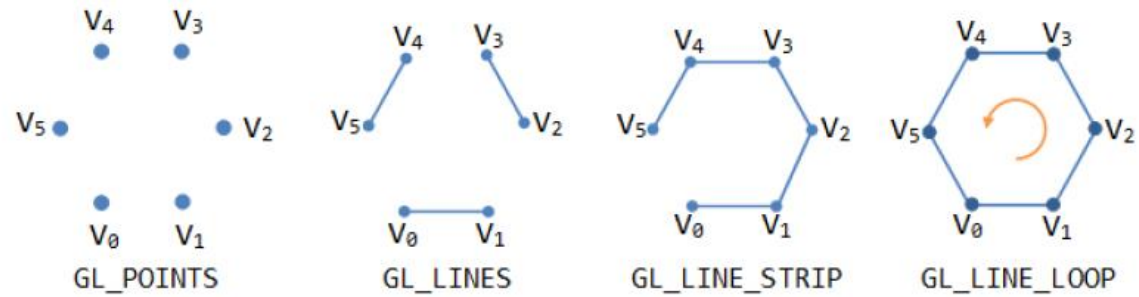
The mode specifies the primitive or primitives that will be created from vertices presented between glBegin() and glEnd().

➤ void **glEnd**();

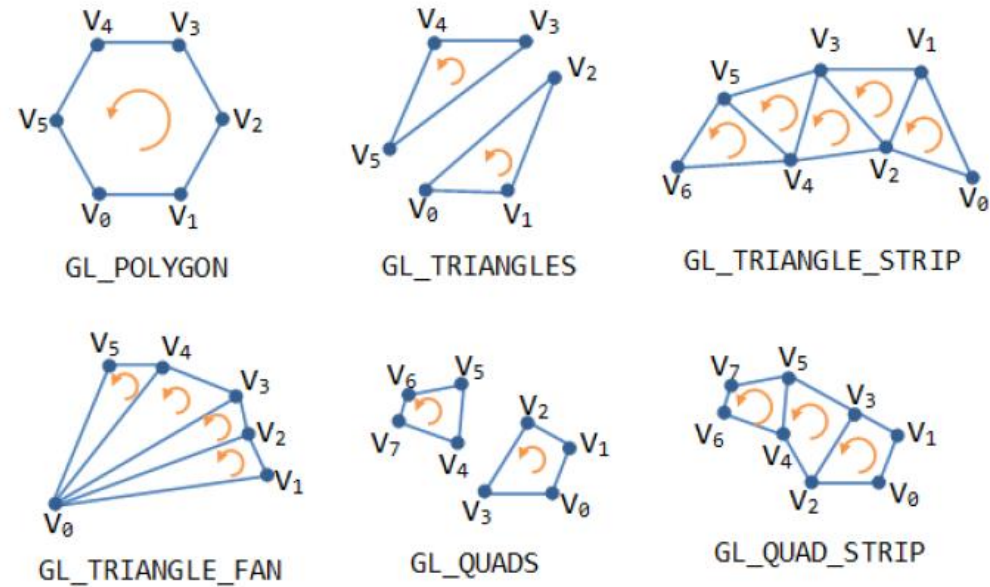
Marks the end of a vertex-data list

```
glBegin(GL_TRIANGLES);  
//red triangle (z = 0)  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex3f(1.0f, 0.0f, 0.0f);  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex3f(0.0f, 1.0f, 0.0f);  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex3f(0.0f, 0.0f, 0.0f);  
//blue triangle (z = -1)  
glColor3f(0.0f, 0.0f, 1.0f);  
glVertex3f(1.0f, 0.0f, -1.0f);  
glColor3f(0.0f, 0.0f, 1.0f);  
glVertex3f(0.0f, 1.0f, -1.0f);  
glColor3f(0.0f, 0.0f, 1.0f);  
glVertex3f(0.0f, 0.0f, -1.0f);  
  
glEnd();  
  
glutSwapBuffers();
```

2-4. Draw the triangles



mode of glBegin() :



OpenGL Primitives

2-4. Draw the triangles

Can only be effective between glBegin() and glEnd() pair

- void **glColor**{34}{sifd}[v](...);
Set the current color
- void **glMaterialfv**(GLenum face,
GLenum pname,
const GLfloat *params);
Specify material parameters for the **lighting model**
(If HW1 use lighting, use this to set the color!)
- void **glNormal3**{bsifd}[v](...)
Set the current normal vector
- void **glVertex**{234}{sifd}[v](...)
Specify a vertex for use in describing a geometric object

You have to set vertex's attributes before glVertex

Ex: Use **glColor()** and **glNormal()** before **glVertex()** to set the vertex's color and normal.

```
glBegin(GL_TRIANGLES);
//red triangle (z = 0)
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(1.0f, 0.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 1.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 0.0f, 0.0f);
//blue triangle (z = -1)
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(1.0f, 0.0f, -1.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(0.0f, 1.0f, -1.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(0.0f, 0.0f, -1.0f);

glEnd();

glutSwapBuffers();
```

2-5. Complete drawing

Don't forget to swap the buffers when you complete drawing

- void **glutSwapBuffers();**
Swap the front and back buffers.

```
glBegin(GL_TRIANGLES);  
//red triangle (z = 0)  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex3f(1.0f, 0.0f, 0.0f);  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex3f(0.0f, 1.0f, 0.0f);  
glColor3f(1.0f, 0.0f, 0.0f);  
glVertex3f(0.0f, 0.0f, 0.0f);  
//blue triangle (z = -1)  
glColor3f(0.0f, 0.0f, 1.0f);  
glVertex3f(1.0f, 0.0f, -1.0f);  
glColor3f(0.0f, 0.0f, 1.0f);  
glVertex3f(0.0f, 1.0f, -1.0f);  
glColor3f(0.0f, 0.0f, 1.0f);  
glVertex3f(0.0f, 0.0f, -1.0f);  
  
glEnd();  
  
glutSwapBuffers();
```


2-6 Face culling

OpenGL checks and renders all the faces that are front facing towards the viewer while discarding all the back face.

➤ `glEnable(GL_CULL_FACE);`

Enable OpenGL's `GL_CULL_FACE` option.

➤ `void glCullFace (GLenum mode)`

Specify whether front or back facing faces can be culled.

mode: `GL_BACK`, `GL_FRONT`, `GL_FRONT_AND_BACK`

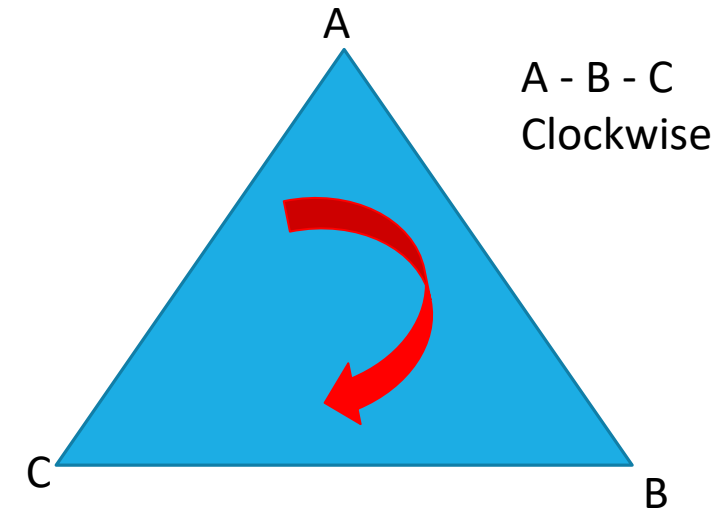
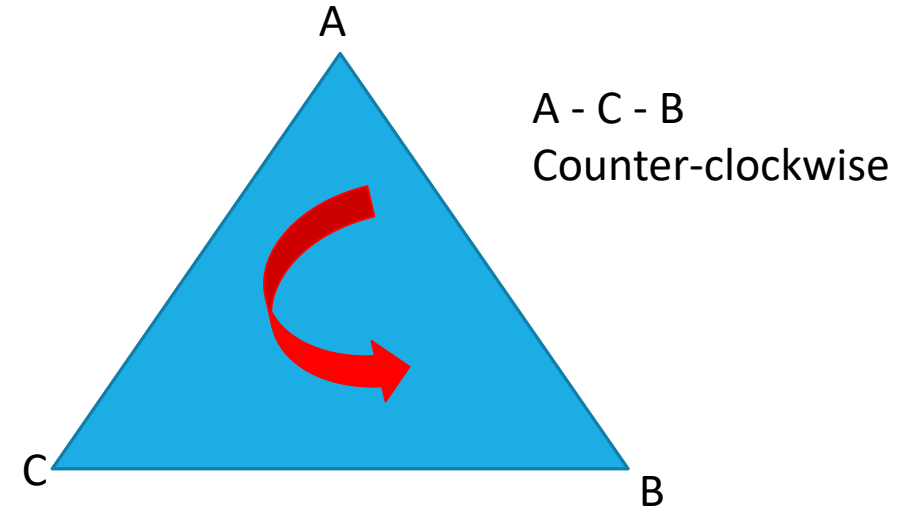
Ex. `glCullFace(GL_BACK)` culls only the back faces. OpenGL allows us to change the type of face that we want to cull as well.

➤ `void glFrontFace (GLenum mode);`

Define front and back facing polygons

mode: `GL_CW`, `GL_CCW`. The default value is `GL_CCW`.

Ex. `glFrontFace(GL_CCW)` => counter-clockwise ordering



2-7 More Information about OpenGL ...

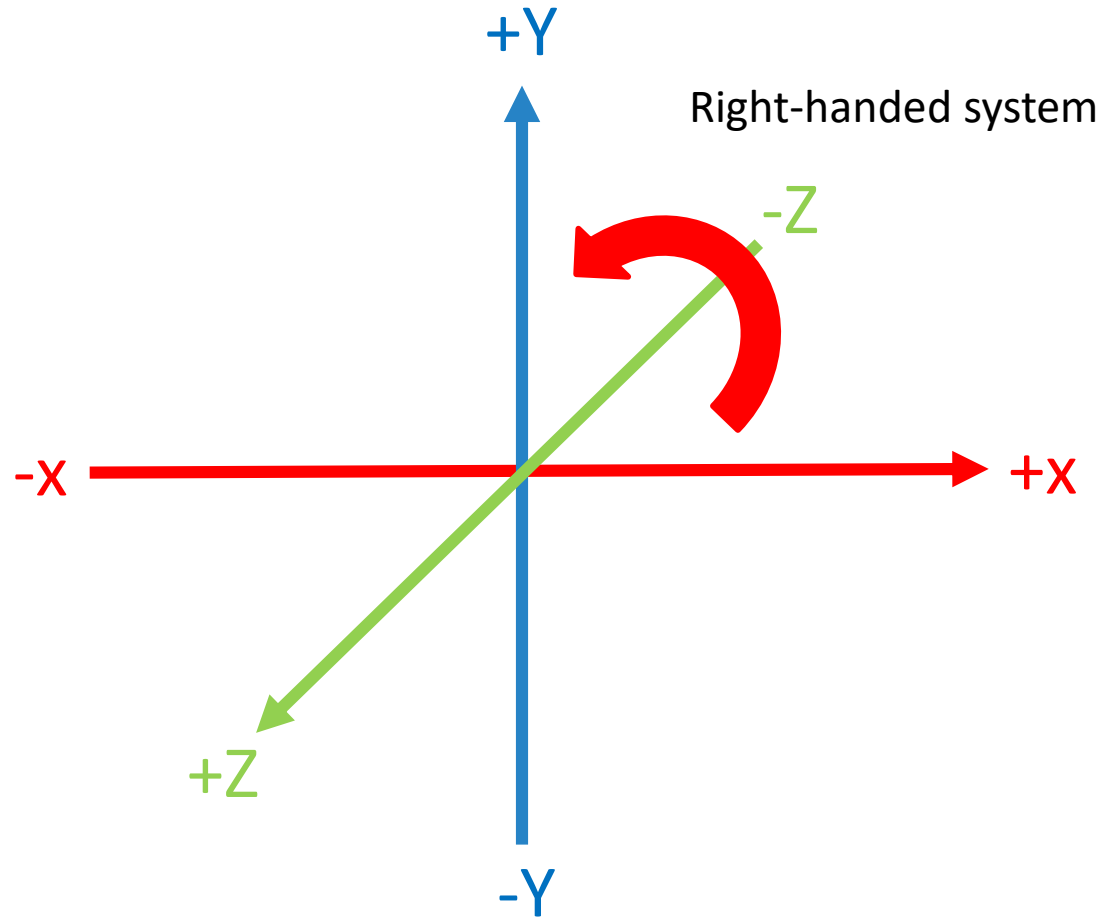
OpenGL data type

Table 3-2 OpenGL variable types and corresponding C data types

OpenGL Data Type	Internal Representation	Defined as C Type	C Literal Suffix
GLbyte	8-bit integer	Signed char	b
GLshort	16-bit integer	Short	s
GLint, GLsizei	32-bit integer	Long	I
GLfloat, GLclampf	32-bit floating point	Float	f
GLdouble, GLclampd	64-bit floating point	Double	d
GLubyte, GLboolean	8-bit unsigned integer	Unsigned char	ub
GLushort	16-bit unsigned integer	Unsigned short	us
GLuint, GLenum, GLbitfield	32-bit unsigned integer	Unsigned long	ui

2-7 More Information about OpenGL ...

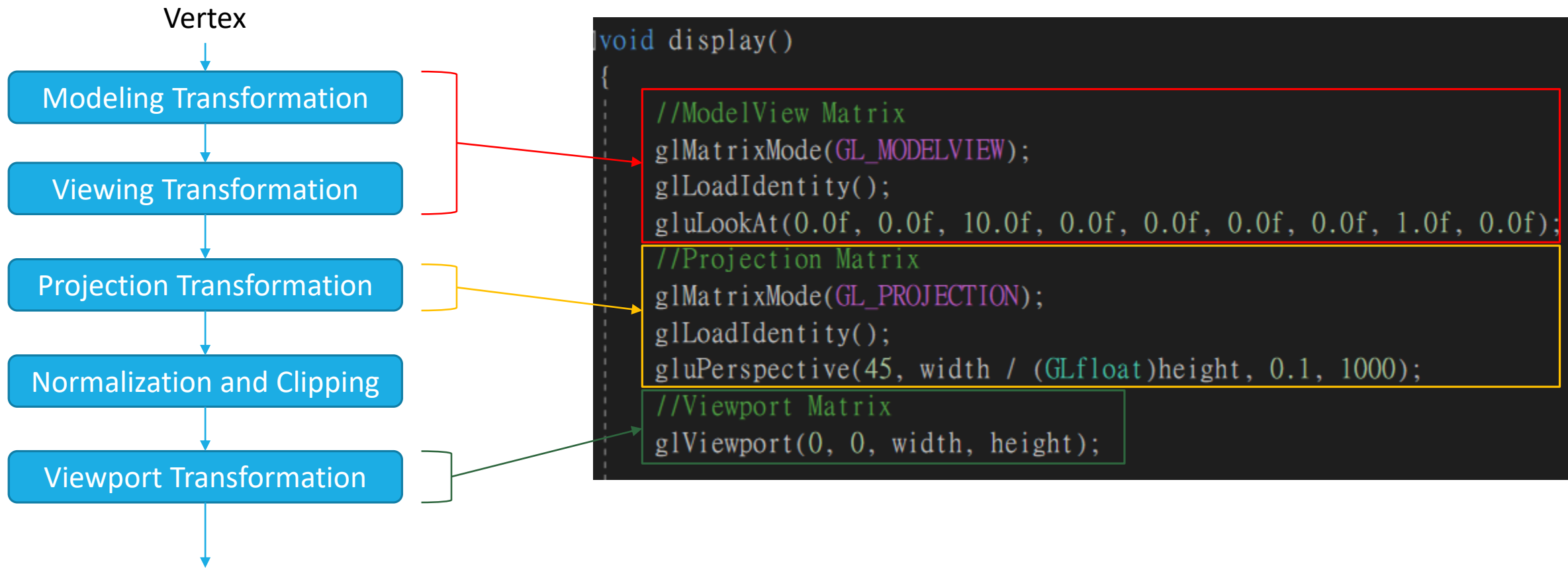
OpenGL coordinate system



OpenGL-Transform

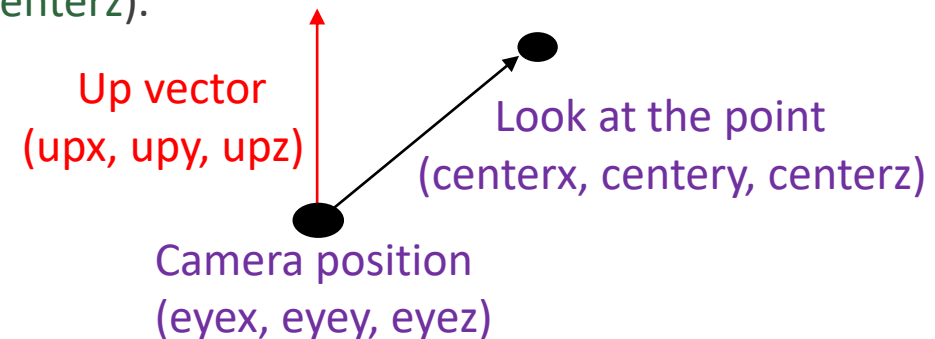
How to rotate the planet in OpenGL?

3-1 Overview



3-2 ModelView Matrix

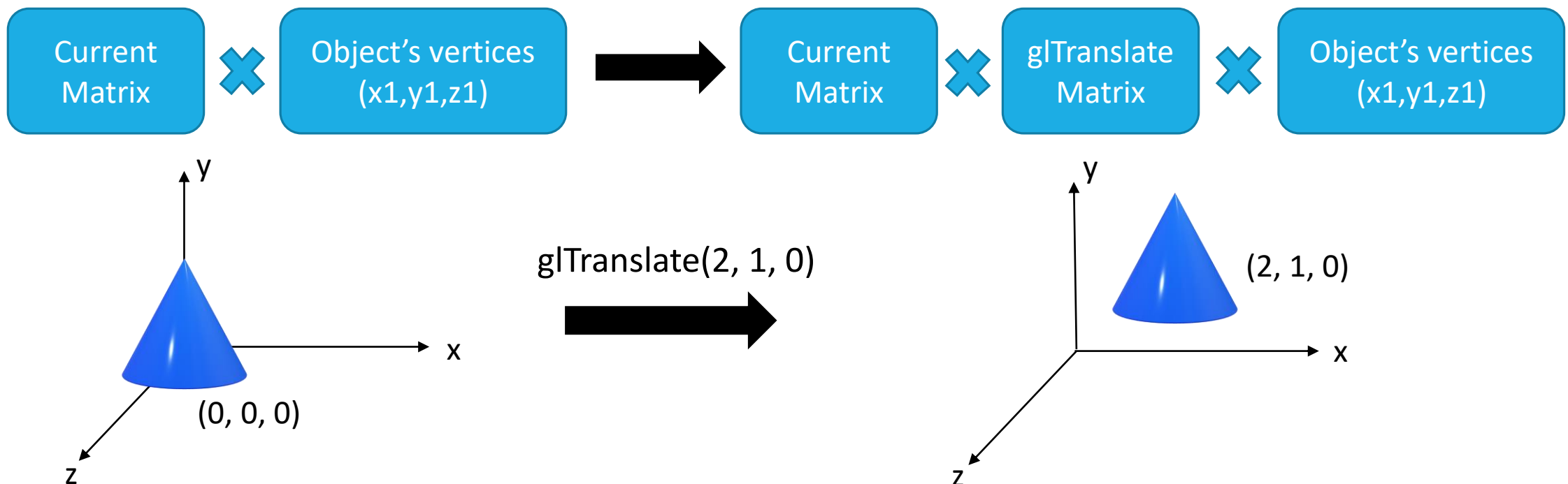
- `void glMatrixMode (GLenum mode);`
 - There are three different modes : `GL_MODELVIEW`, `GL_PROJECTION`, or `GL_TEXTURE`.
 - Each mode has its corresponding matrix stack, and only one matrix stack is active at a time.
- `void glLoadIdentity (void);`
 - Replace the current matrix with the identity matrix.
- `void gluLookAt (GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble centerx, GLdouble centery, GLdouble centerz, GLdouble upx, GLdouble upy, GLdouble upz);`
 - Viewing direction is from $(eyex, eyey, eyez)$ to $(centerx, centery, centerz)$.
 - Camera's up vector is (upx, upy, upz) .
 - Changing the parameters in `gluLookAt()` means changing the camera's position, not the object's position.



3-3 Modeling Transformation

➤ `void glTranslate{fd} (TYPE x, TYPE y, TYPE z);`

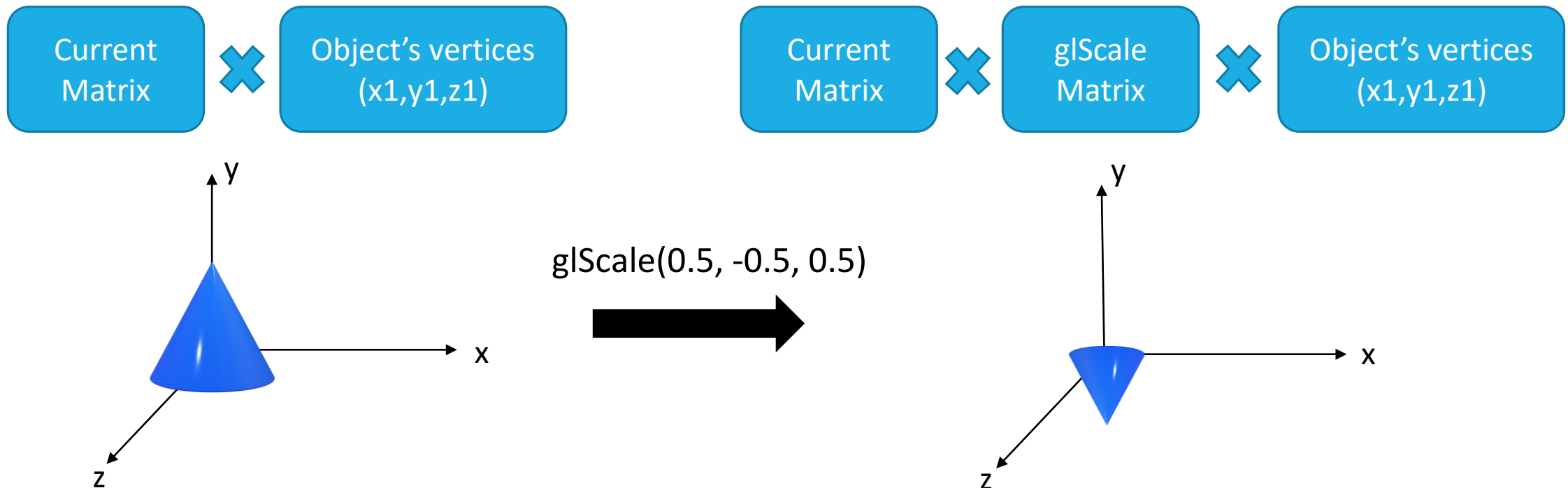
It will translate the object's vertices by the given `x`, `y`, `z` values.



3-3 Modeling Transformation

➤ `void glScale{fd}(TYPE x, TYPE y, TYPE z);`

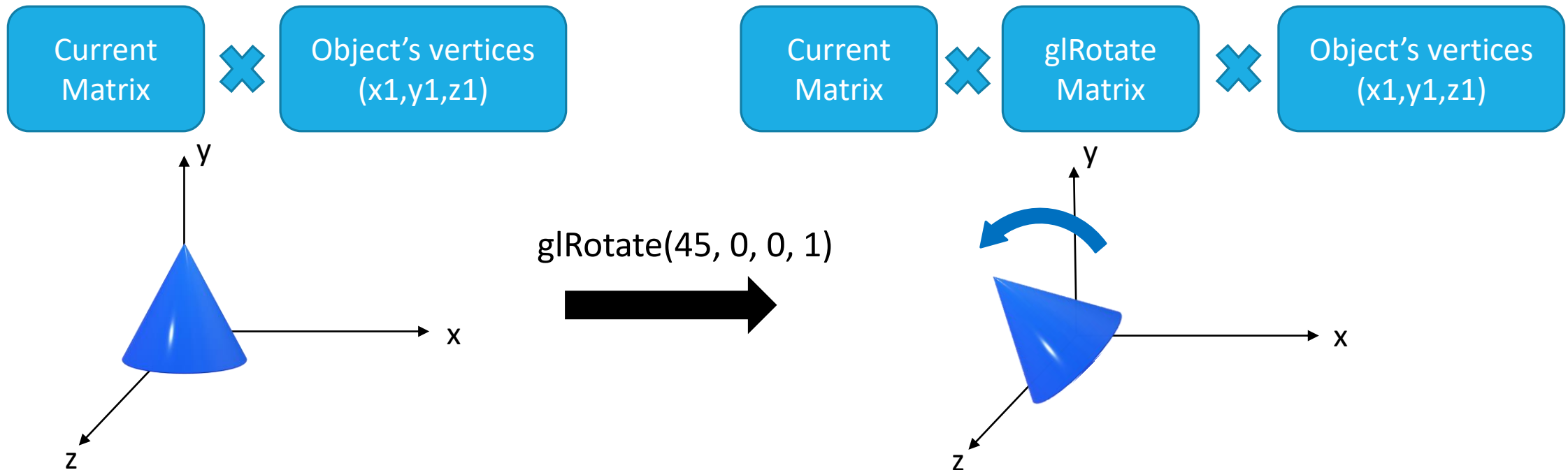
It will scale the object along the *x*, *y*, or *z* axes.



3-3 Modeling Transformation

➤ `void glRotate{fd}(TYPE angle, TYPE x, TYPE y, TYPE z);`

It will rotate the object in a counterclockwise direction. The `angle` parameter is the angle of rotation in degrees. The rotating axis is from the origin to the point `(x, y, z)`.



3-3 Modeling Transformation

In OpenGL, matrices multiplications are **in reverse order** when applied to the vertices.

.....

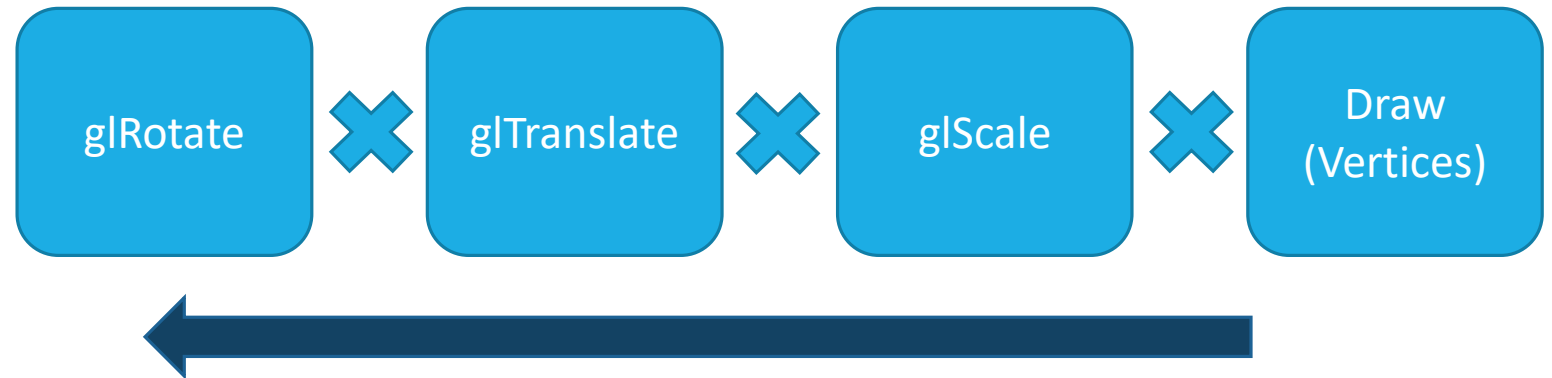
```
glRotate();
```

```
glTranslate();
```

```
glScale();
```

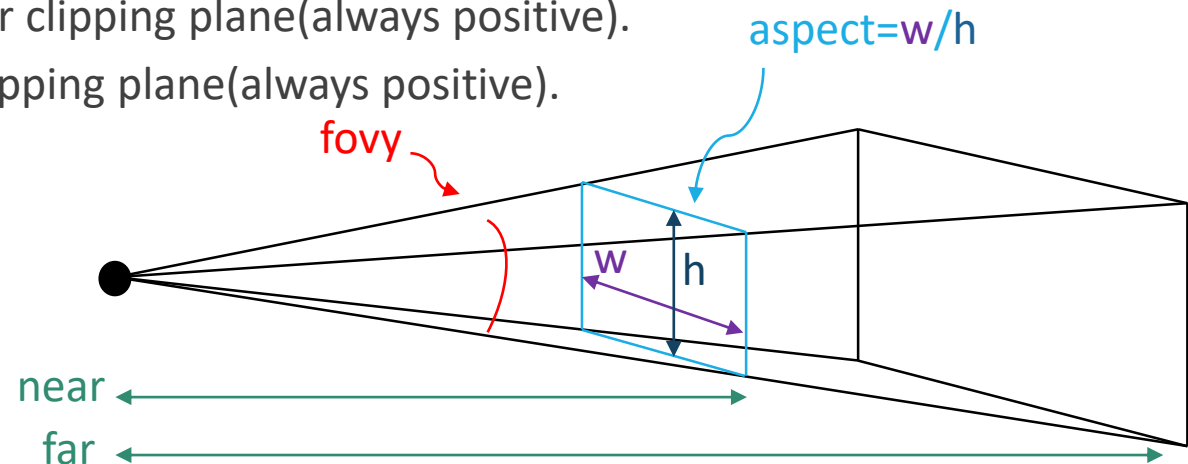
```
Draw();
```

.....



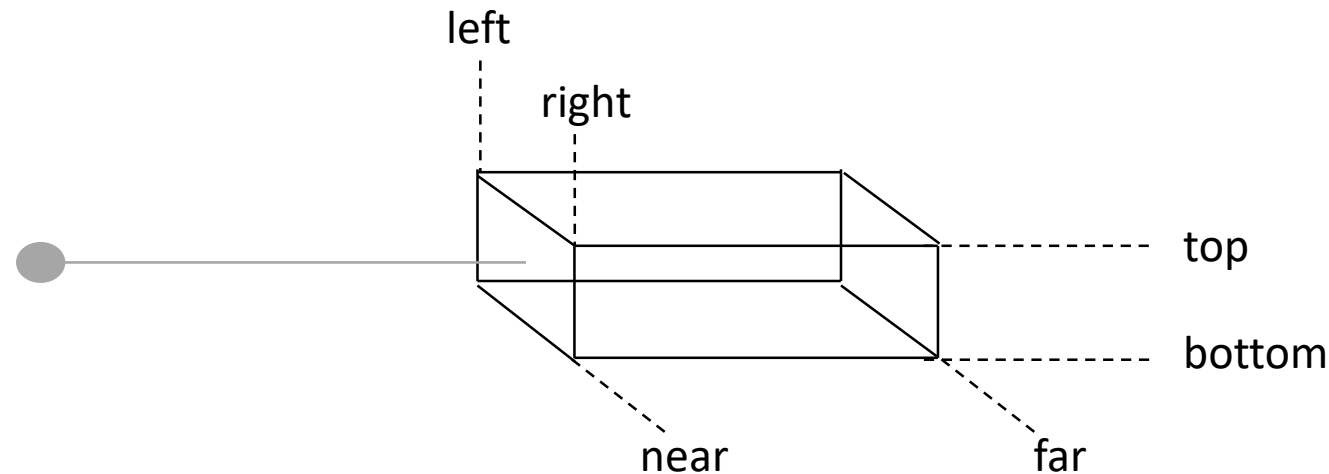
3-4 Projection Matrix --- Perspective Projection

- `glMatrixMode(GL_PROJECTION);`
- `glLoadIdentity();`
- `void gluPerspective (GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far);`
 - `fovy` : the angle of the field of view in the y direction.(in degrees)
 - `aspect` : the aspect ratio(w/h) that determines the field of view.
 - `near` : the distance between the viewer to the near clipping plane(always positive).
 - `far` : the distance between the viewer to the far clipping plane(always positive).



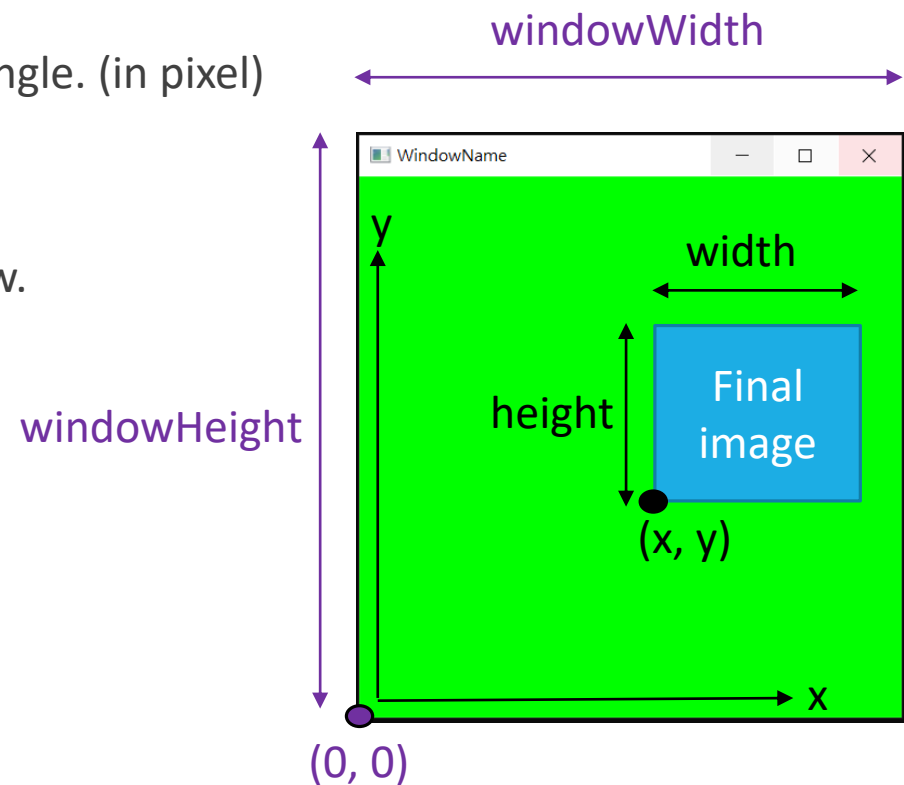
3-4 Projection Matrix --- Orthographic Projection

- void **glOrtho** (GLdouble **left**, GLdouble **right**, GLdouble **bottom**, GLdouble **top**, GLdouble **near**, GLdouble **far**);
 - **left, right** : the coordinates of the left and right vertical clipping planes.
 - **bottom, top** : the coordinates of the bottom and top horizontal clipping planes.
 - **near, far** : the coordinates of the near and far depth clipping planes.



3-5 Viewport Matrix

- void **glViewport** (GLint x , GLint y , GLsizei $width$, GLsizei $height$);
 - Map the final image to some region of the window.
 - The point (x, y) : the lower-left corner of the viewport rectangle. (in pixel)
 - $width, height$: the size of the viewport rectangle.
 - default value : (0, 0, $windowWidth$, $windowHeight$)
 - $windowWidth$ and $windowHeight$ are the size of the window.

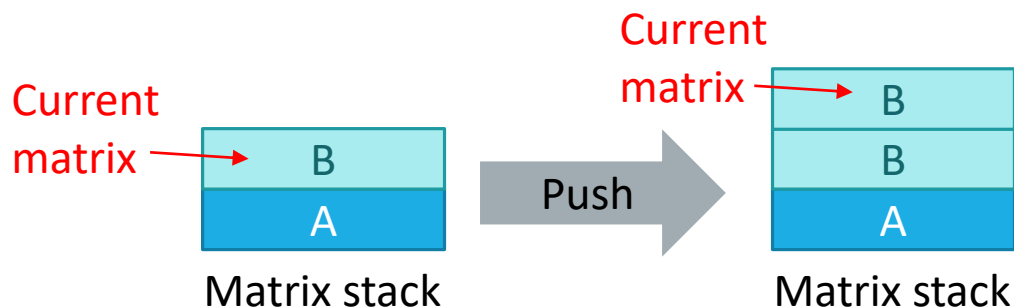


3-6 Matrix Stack Mechanism

- You can **store** certain transformation matrix on the matrix stack and easily get it when you want to reuse.
- The top of the matrix stack is the current matrix.
- Use **glPushMatrix()** and **glPopMatrix()** to manage the stack.

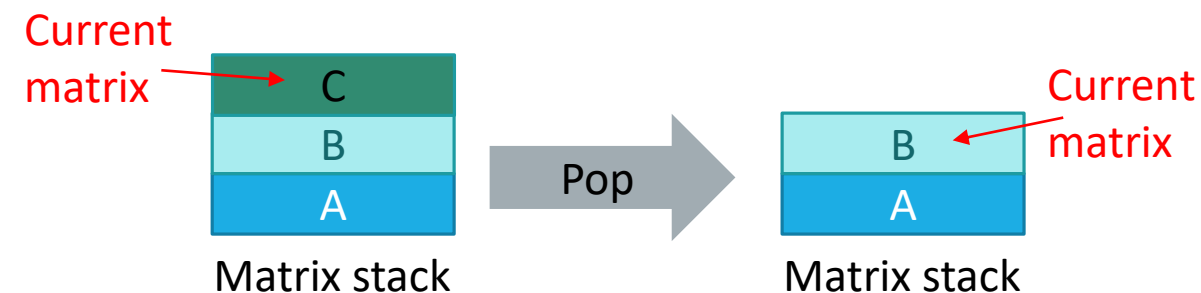
void **glPushMatrix()** :

Push the current matrix on the top of the stack.



void **glPopMatrix()** :

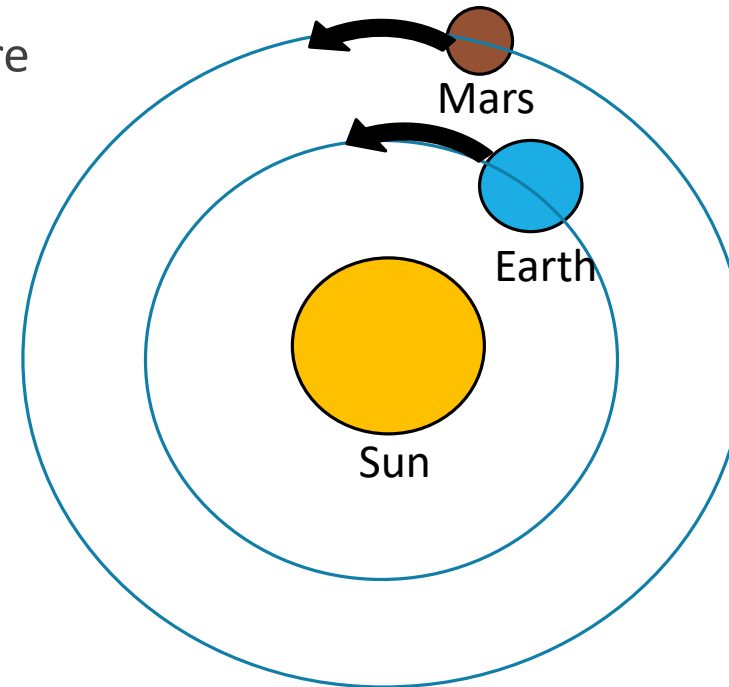
Pop the top matrix off the matrix stack.



3-7 Example

➤ How to draw a solar system?

The sun is in the center, and the Earth and the Mars are rotating around the sun.



Step 1



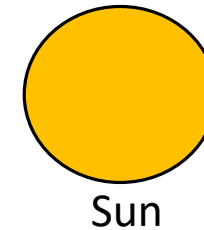
```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
gluLookAt(.....); //matrix M  
glMatrixMode(GL_PROJECTION);  
.....
```

Set up default transform matrices.

Step 2

Matrix stack  ← Current matrix

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
gluLookAt(.....); //matrix M  
glMatrixMode(GL_PROJECTION);  
.....  
glMatrixMode(GL_MODELVIEW);  
drawSun();
```



Activate the matrix stack of the MODELVIEW, and draw the Sun at the default position.

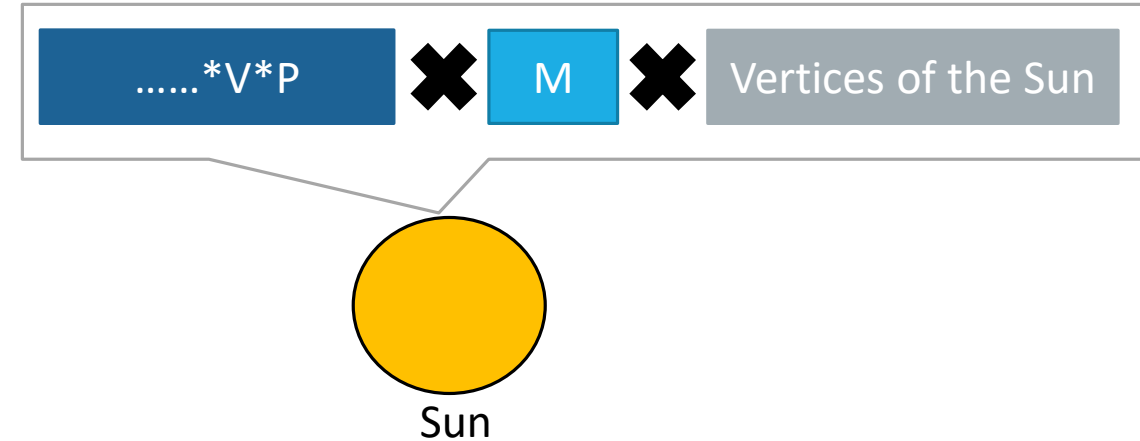
Step 2

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
gluLookAt(.....); //matrix M
glMatrixMode(GL_PROJECTION);
.....
```

```
glMatrixMode(GL_MODELVIEW);
drawSun();
```

```
glBegin();
..... //vertex data list
.....
glEnd();
```

Matrix stack M ← Current matrix



In the function `drawSun()`, it set up the vertex data list.

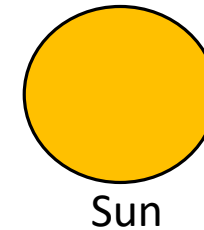
Step 3

Matrix stack



Current
matrix

```
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();  
gluLookAt(.....); //matrix M  
glMatrixMode(GL_PROJECTION);  
.....  
glMatrixMode(GL_MODELVIEW);  
drawSun();  
glPushMatrix(); //push the current matrix M to the  
                top of the matrix stack
```



Sun

Use glPushMatrix() to store the position of the Sun.

Step 4

Matrix stack



Current
matrix

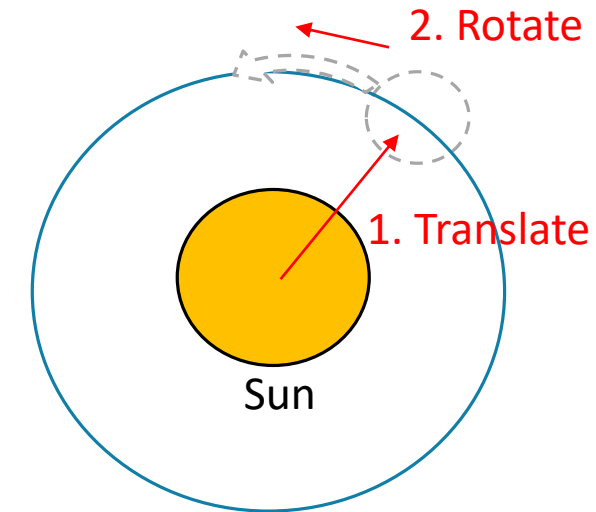
.....

```
drawSun();
```

```
glPushMatrix();//push the current matrix M to the  
top of the matrix stack
```

```
glRotate();
```

```
glTranslate(); //remember that matrices multiplications  
are in reverse order
```



To draw the Earth rotating around the Sun, we first translate it to certain position and then rotate it.

Step 5

.....

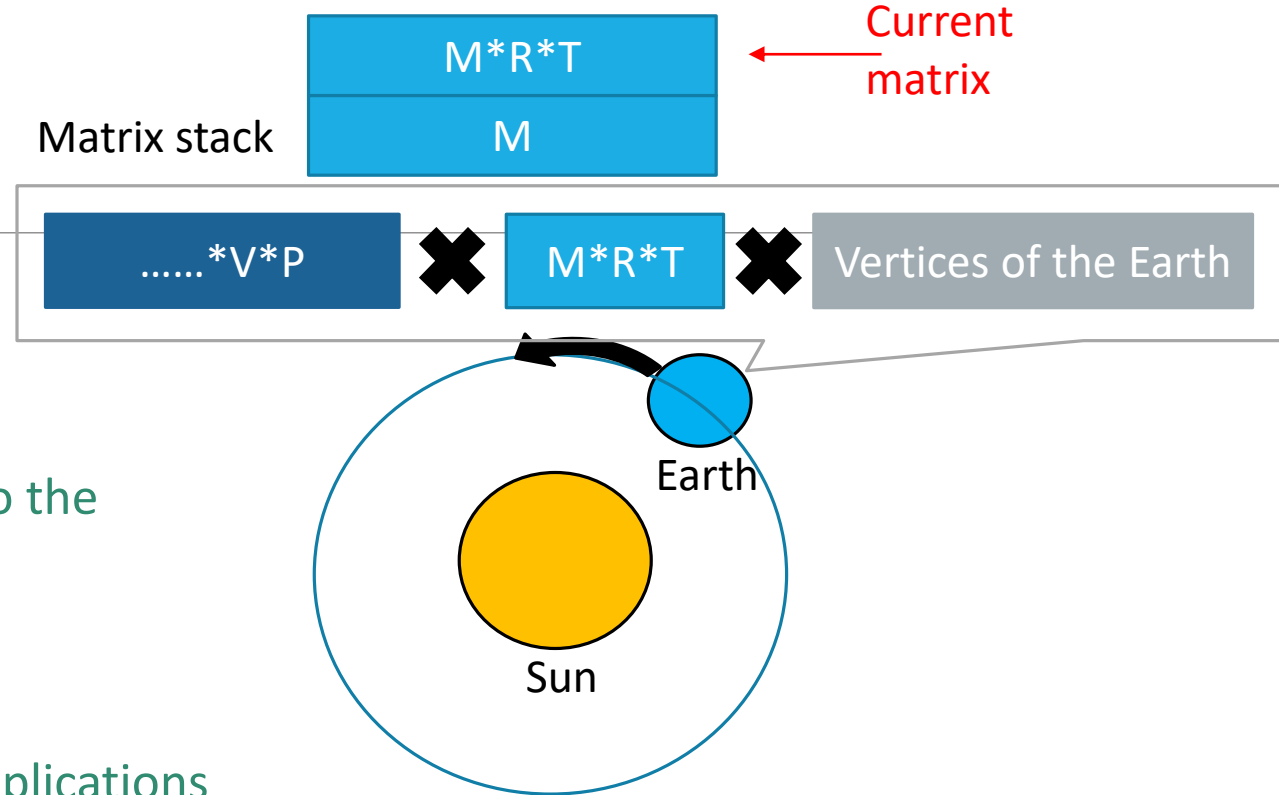
```
drawSun();
```

```
glPushMatrix();//push the current matrix M to the  
top of the matrix stack
```

```
glRotate();
```

```
glTranslate(); //remember that matrices multiplications  
are in reverse order
```

```
drawEarth();
```



Use drawEarth() to draw on the window.

Step 6

Matrix stack



Current
matrix

.....

```
drawSun();
```

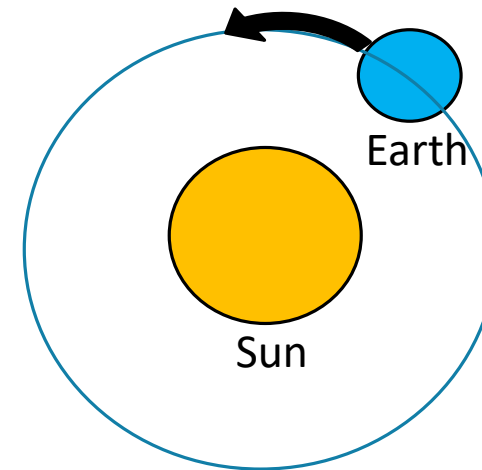
```
glPushMatrix();//push the current matrix M to the  
top of the matrix stack
```

```
glRotate();
```

```
glTranslate(); //remember that matrices multiplications  
are in reverse order
```

```
drawEarth();
```

```
glPopMatrix(); //pop off the current matrix  $M \cdot R \cdot T$ 
```



We want to draw the Mars which was also rotating around the Sun, so we need to pop off the current matrix(matrix $M \cdot R \cdot T$) to get the Sun position(matrix M)

Step 6

Matrix stack

$M * R * T * T * R$

Current
matrix

.....

```
drawSun();
```

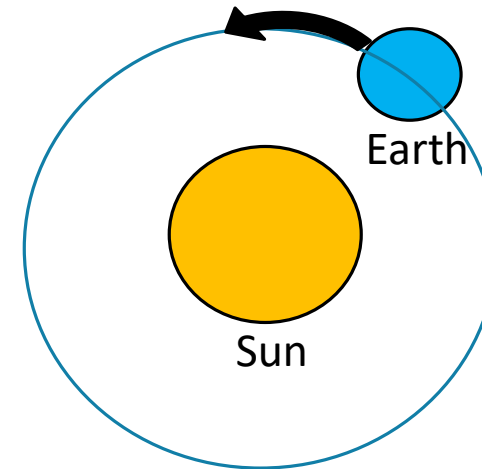
```
glPushMatrix(); //push the current matrix M to the  
top of the matrix stack
```

```
glRotate(a);
```

```
glTranslate(b); //remember that matrices multiplications  
are in reverse order
```

```
drawEarth();
```

```
glPopMatrix(); //pop off the current matrix M * R * T  
glTranslate(-b); glRotate(-a); //inverse
```



However, If you don't use `glPushMatrix()` and `glPopMatrix()` in your code, then you need to multiply the inverse matrix of Rotate and Translate to get the Sun position.

Step 7

Matrix stack



Current
matrix



.....

```
drawSun();
```

```
glPushMatrix(); //push the current matrix M to the  
top of the matrix stack
```

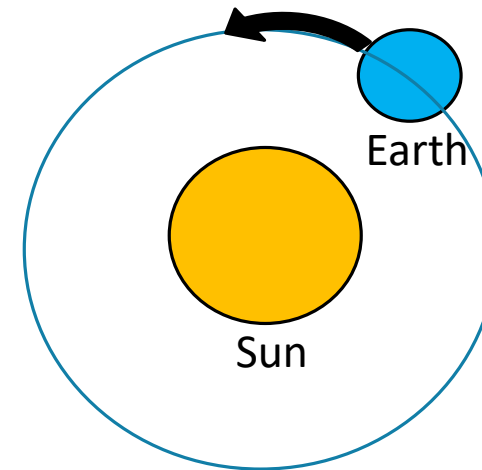
```
glRotate();
```

```
glTranslate(); //remember that matrices multiplications  
are in reverse order
```

```
drawEarth();
```

```
glPopMatrix(); //pop off the current matrix  $M \cdot R \cdot T$ 
```

```
glPushMatrix(); //store the Sun again
```



Same as before, we use `glPushMatrix()` to store the Sun again.

Step 8

.....

```
drawEarth();
```

```
glPopMatrix(); //pop off the current matrix M*R*T
```

```
glPushMatrix();//store the Sun again
```

```
glRotate();
```

```
glTranslate(); //remember that matrices multiplications  
are in reverse order
```

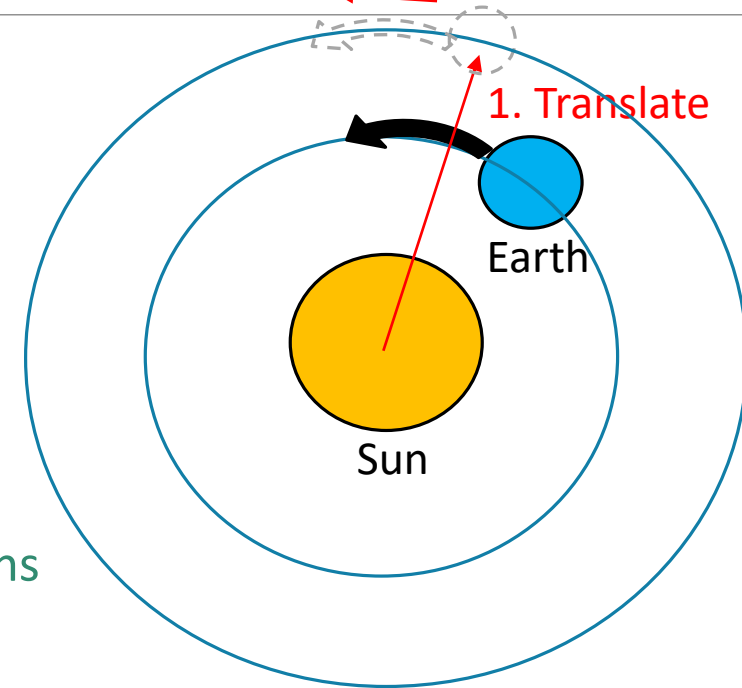
Matrix stack



Current
matrix

2. Rotate

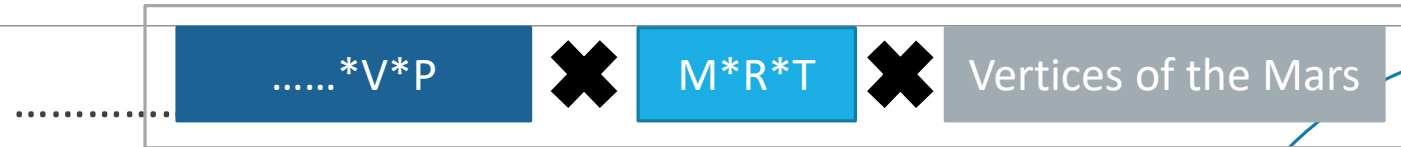
1. Translate



Same as before, we translate it to certain position and then rotate it.

Step 9

Matrix stack



`drawEarth();`

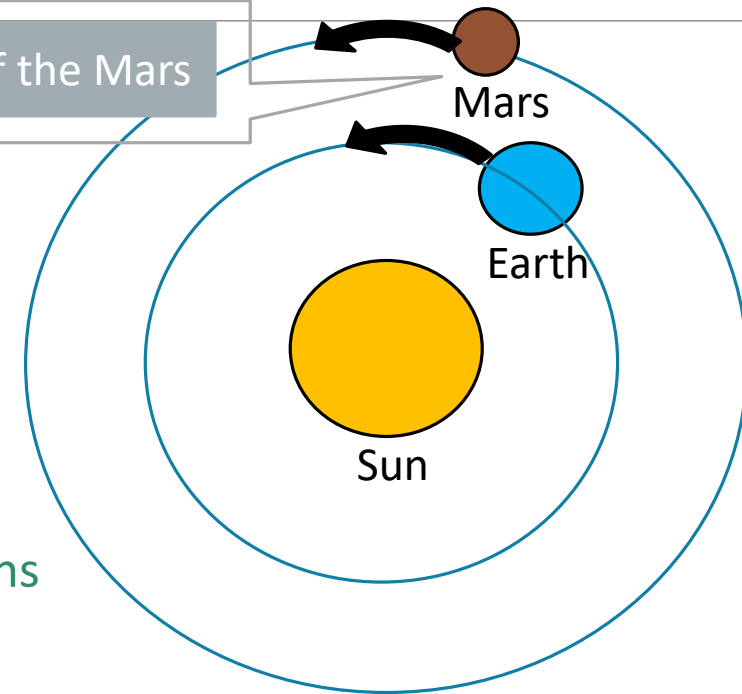
`glPopMatrix(); //pop off the current matrix $M * R * T$`

`glPushMatrix(); //store the Sun again`

`glRotate();`

`glTranslate(); //remember that matrices multiplications are in reverse order`

`drawMars();`



Use `drawMars()` to draw on the window.

Step 10

.....

```
drawEarth();
```

```
glPopMatrix(); //pop off the current matrix  $M \cdot R \cdot T$ 
```

```
glPushMatrix(); //store the Sun again
```

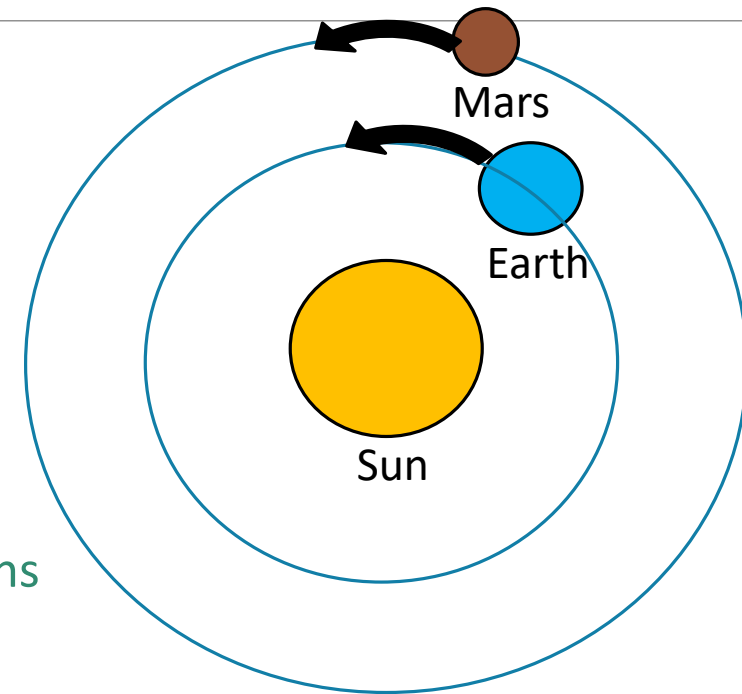
```
glRotate();
```

```
glTranslate(); //remember that matrices multiplications  
are in reverse order
```

```
drawMars();
```

```
glPopMatrix(); //pop off the current matrix  $M \cdot R \cdot T$ 
```

Matrix stack M ← Current matrix



Pop off the current matrix(matrix $M \cdot R \cdot T$) to get the Sun position(matrix M), if you want to draw other planets.

<https://www.khronos.org/registry/OpenGL-Refpages/gl2.1/>

<https://learnopengl.com/>

<http://opengl.czweb.org/ch03/031-034.html>