Computer Graphics

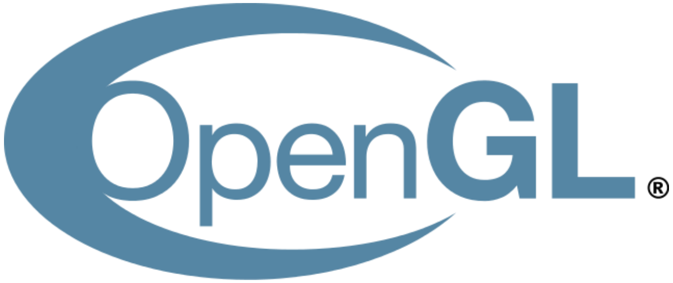
OpenGL Programming

Teacher: Dr. Zhuo SU (苏卓)

E-mail: suzhuo3@mail.sysu.edu.cn

School of Data and Computer Science

Introduction



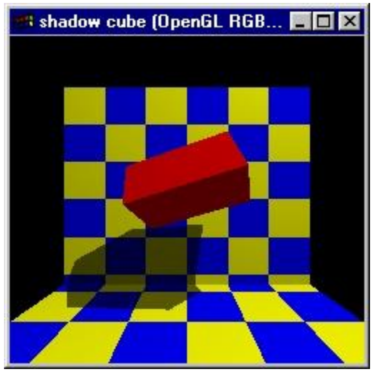
Industry Standard API for Computer Graphics

What is OpenGL?

- The standard specification defining an API that interfaces with the computer's graphics system
  - Cross-language
  - Cross-platform
  - Vendor-independent
- Introduced in 1992 by Silicon Graphics Inc.

OpenGL (Open Graphics Library)

- OpenGL is a cross-language, multi-platform application programming interface (API) for rendering 2D and 3D computer graphics.
- Applications make calls to OpenGL , which then renders an image (by handling the graphics hardware) and displays it
- The API contains about 150 commands.
- is purely concerned with rendering, providing no APIs related to input, audio, or windowing.

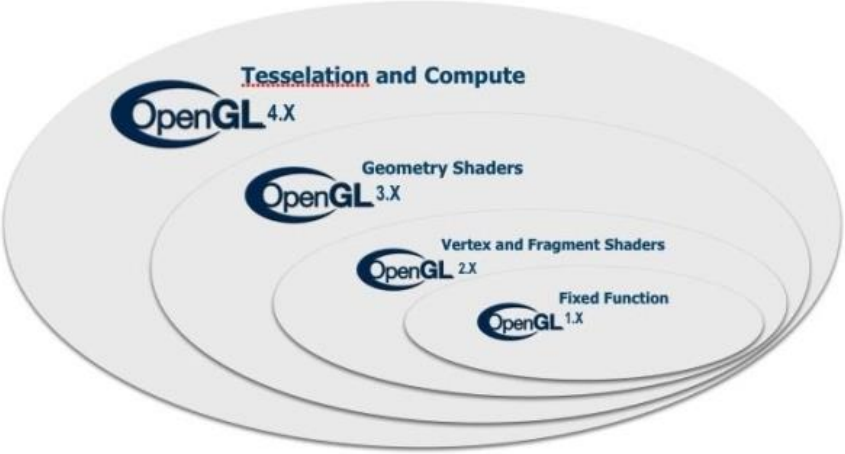


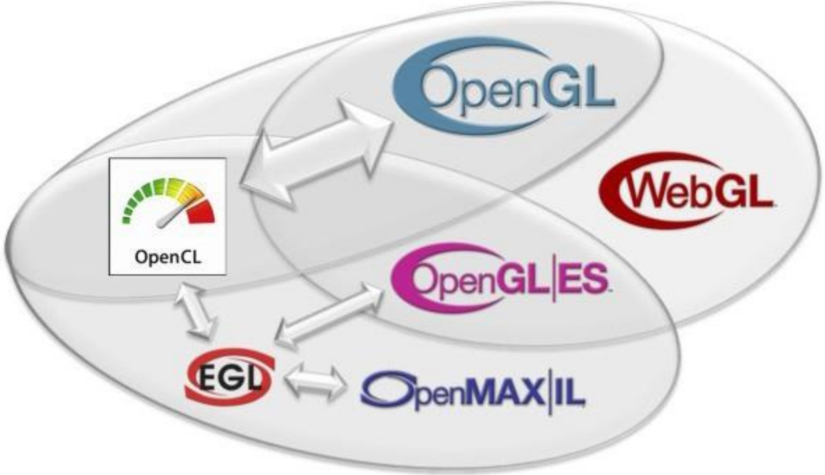
Not the Only One Choice

- Examples: NVIDIA CUDA, DirectX™, Windows Presentation Foundation™ (WPF), RenderMan™, HTML5 + WebGL™, JAVA 3D

Development of OpenGL

- OpenGL is an evolving API.
- New versions of the OpenGL specification are regularly released by the Khronos Group, each of which extends the API to support various new features.
- OpenGL 4.5 Release Date: August, 2014



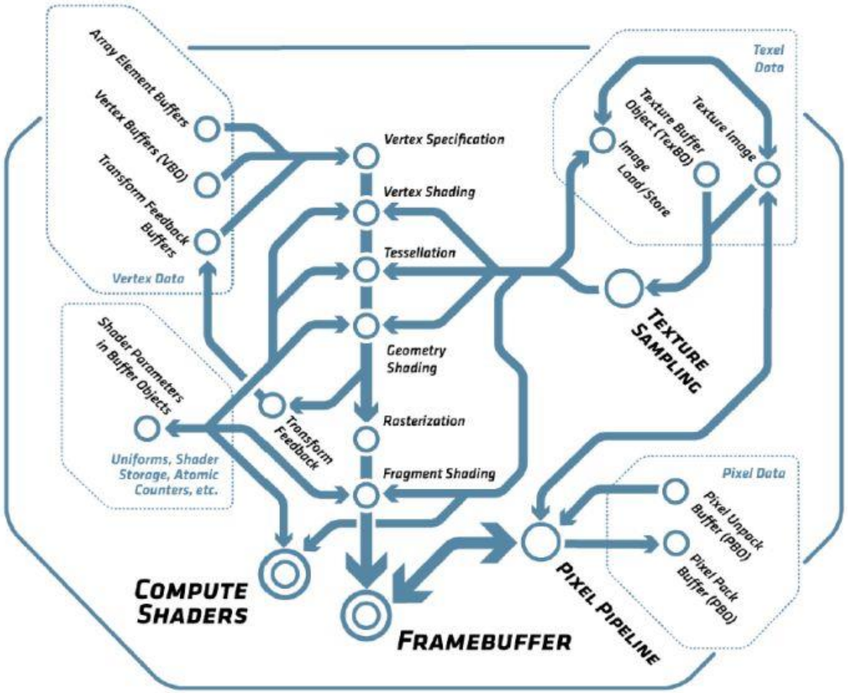


What OpenGL Does

- Allow definition of object shapes, material properties and lighting
- Arrange objects and interprets synthetic camera in 3D space
- Coverts mathematical representations of objects into pixels (rasterization)
- Calculates the color of every object

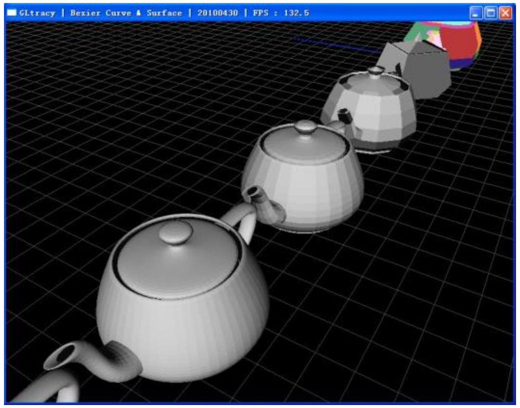
OpenGL 4.5

Core Profile

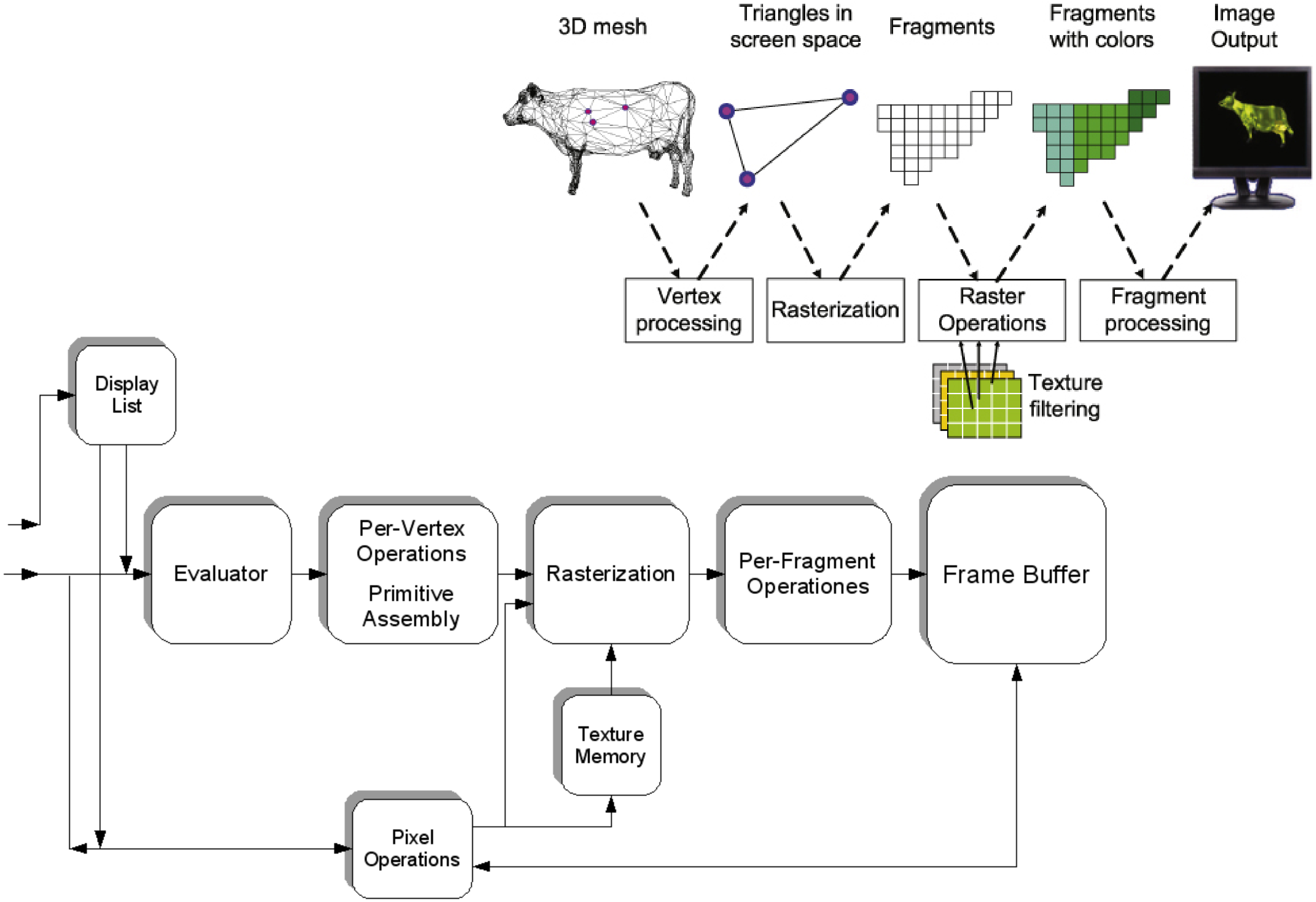


OpenGL and OpenGL Utility Toolkit

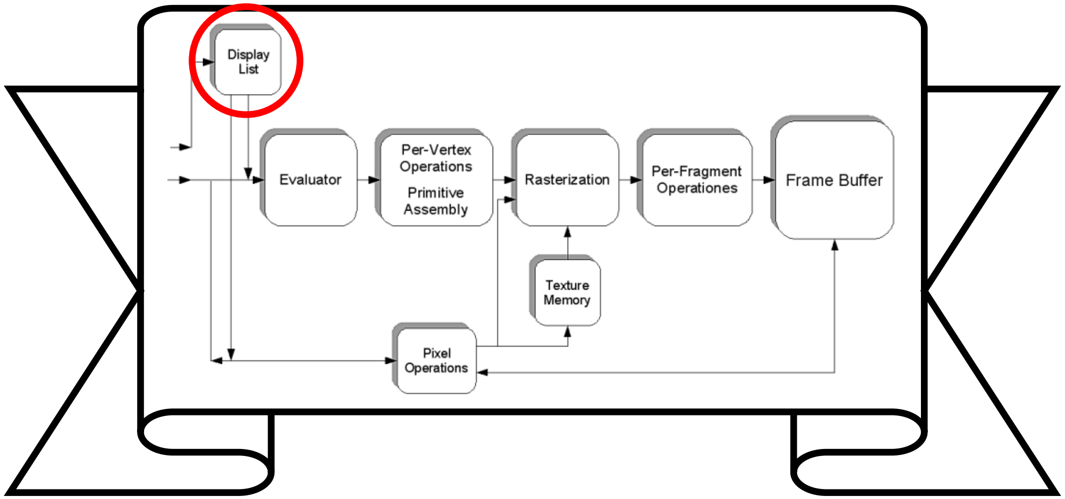
- No high-level rendering functions for complex objects
  - Build your shapes from primitives, points, lines, polygons, etc.
- The utility library GLUT provides additional support
  - (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system.
  - Functions performed include window definition, window control, and monitoring of keyboard and mouse input.
  - Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres and the Utah teapot.
  - GLUT also has some limited support for creating pop-up menus.



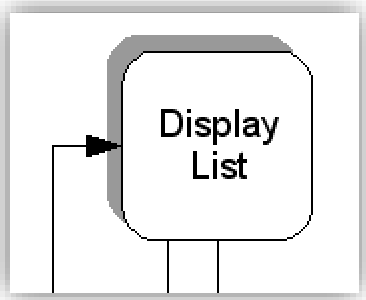
Simplified OpenGL Pipeline



Pieces of OpenGL Pipeline



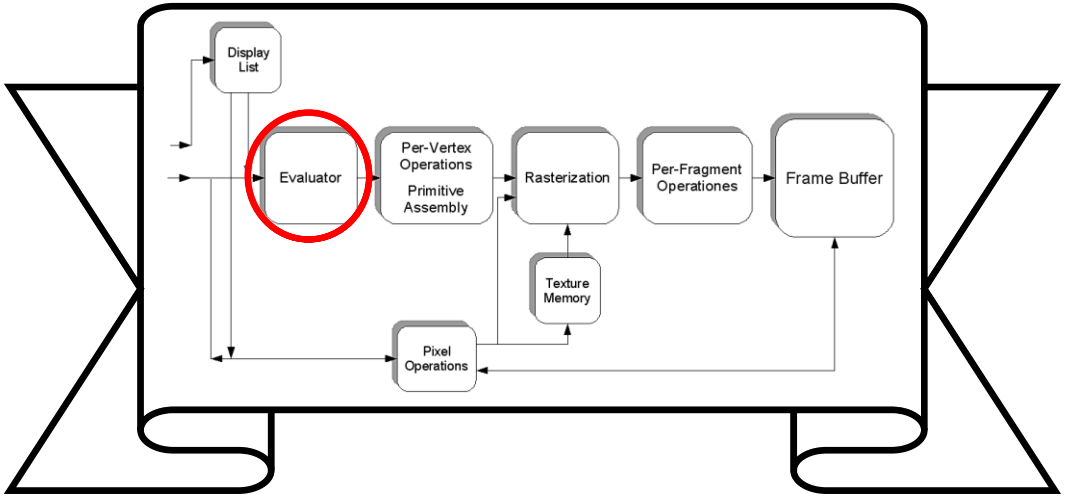
Stores “Subroutines (子程序)”



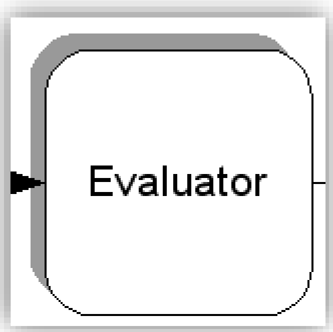
Faster!

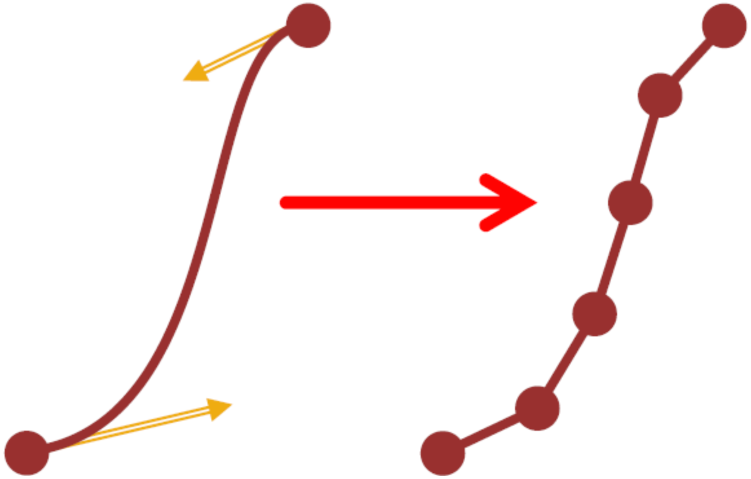
- Pre-compiled
- Store on GPU
- Pre-compute transformations

Pieces of OpenGL Pipeline

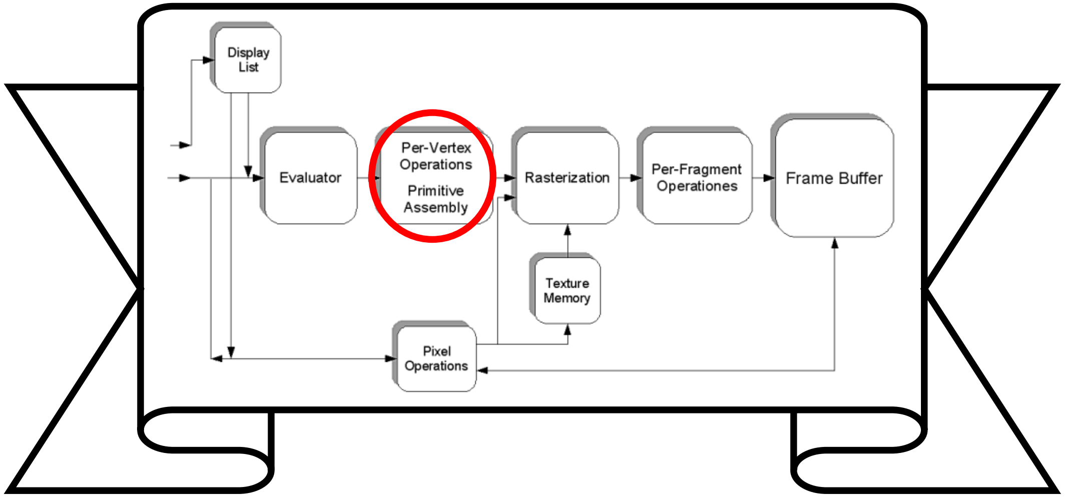


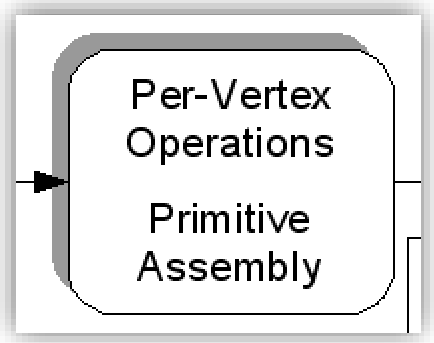
Construct geometric objects





Pieces of OpenGL Pipeline



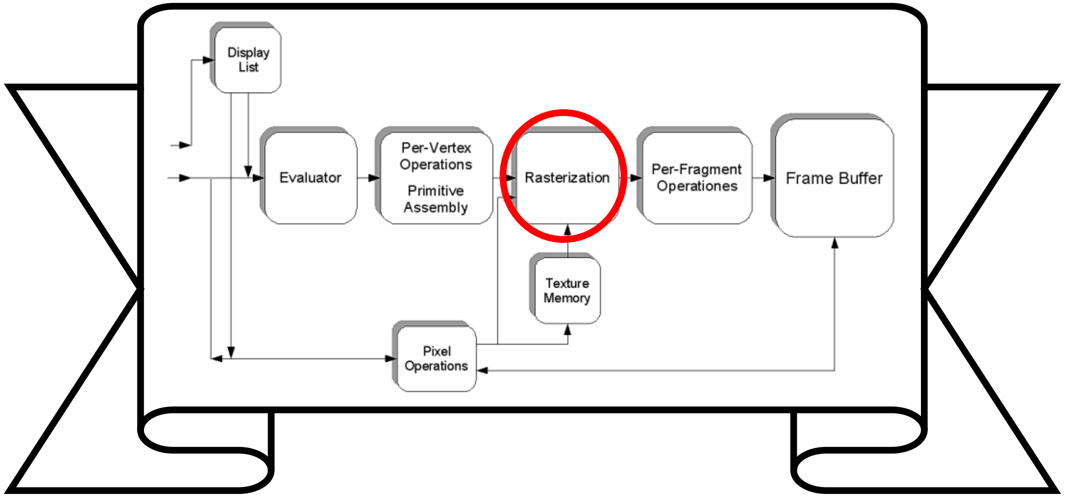


Change meshed geometry

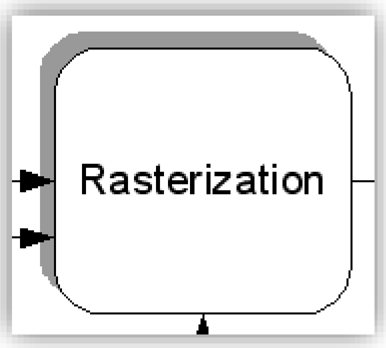
Store primitive shapes

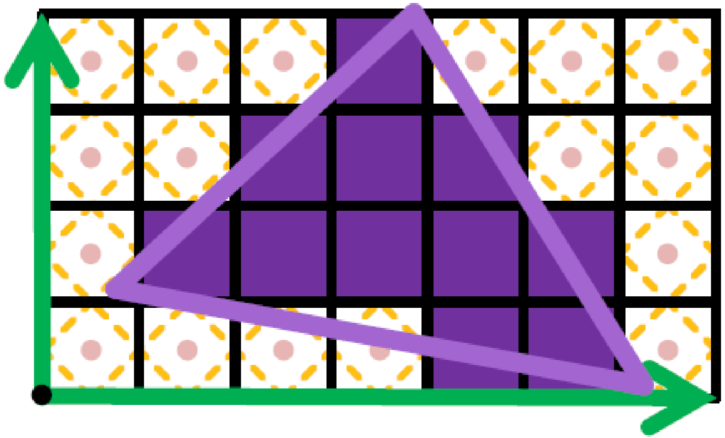
Includes clipping!

Pieces of OpenGL Pipeline

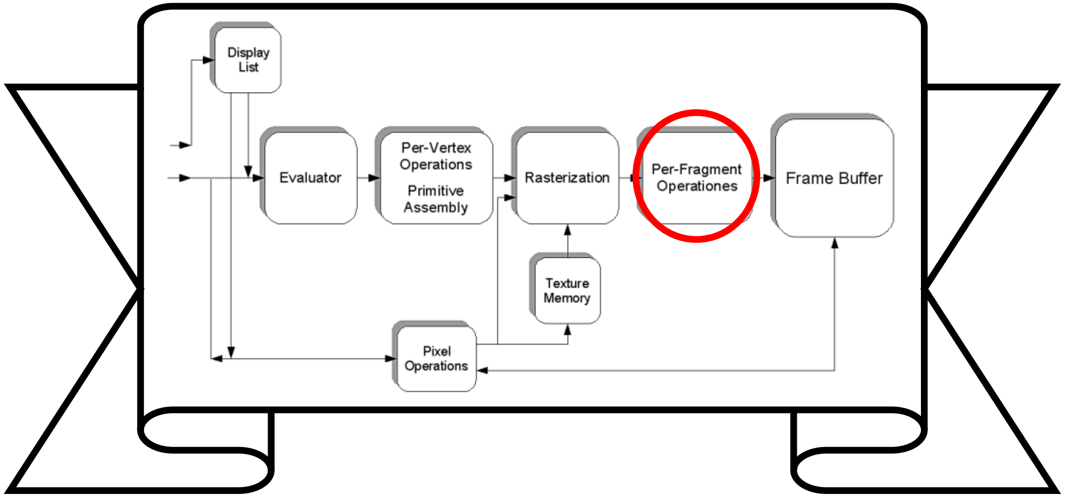


Rasterization

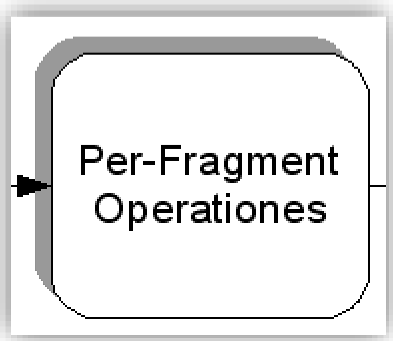


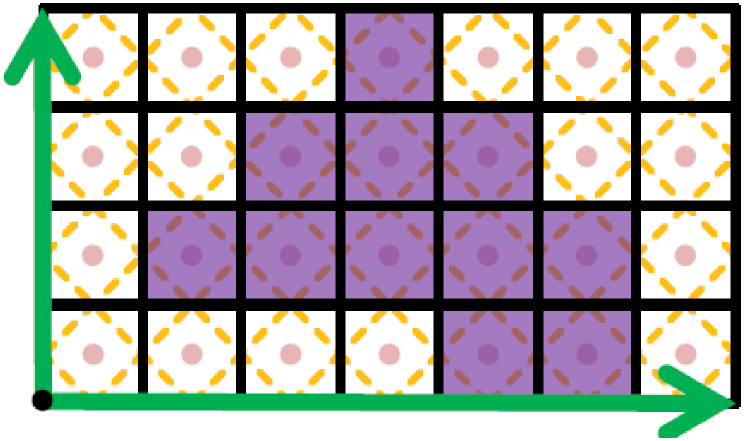


Pieces of OpenGL Pipeline

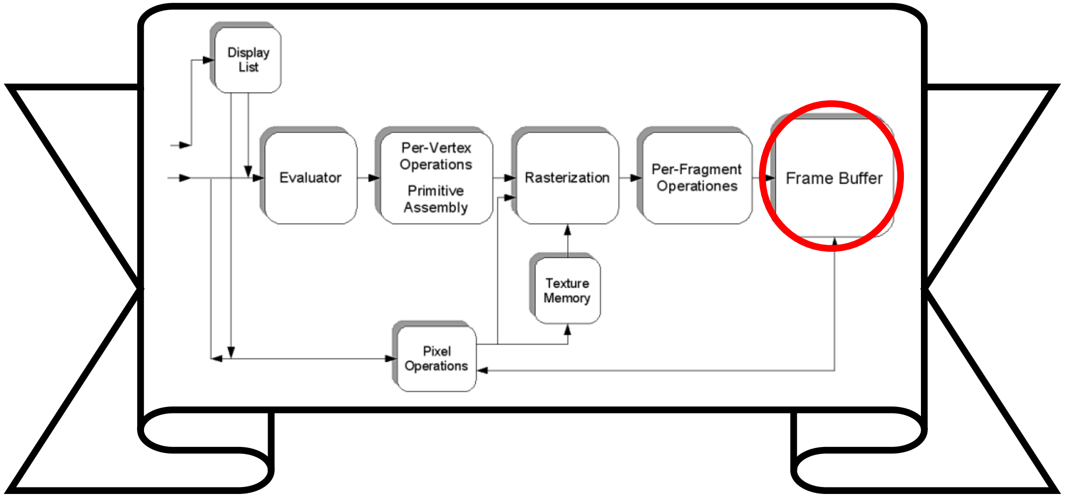


Modify and combine per-pixel information

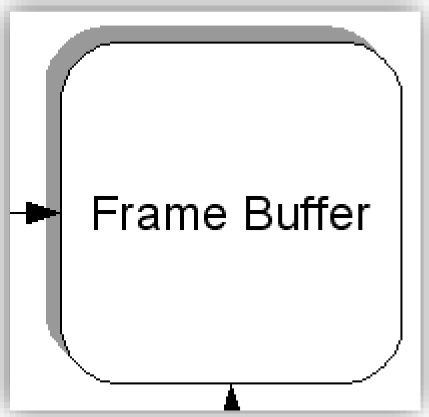


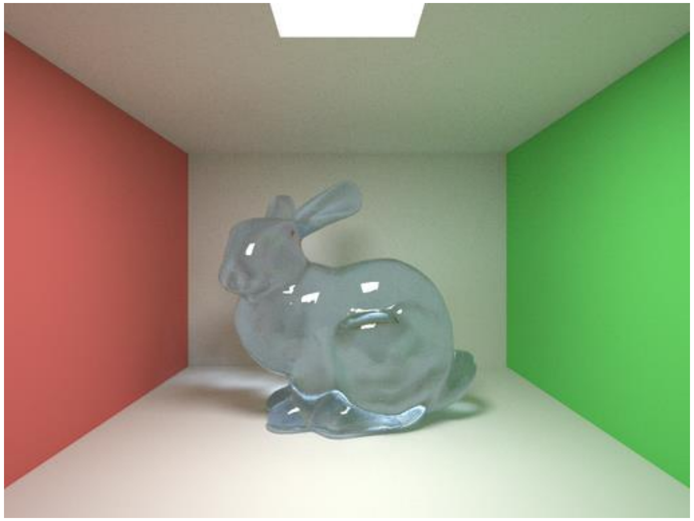


Pieces of OpenGL Pipeline



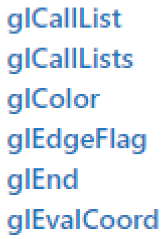
Prepare image to be displayed



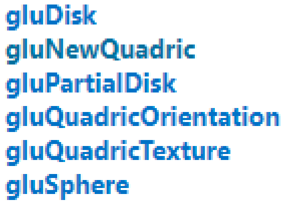


Related API

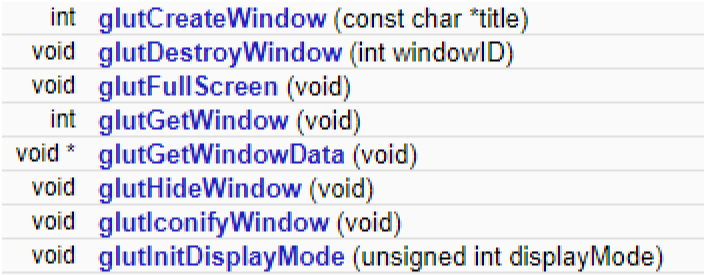
- **opengl32.lib (OpenGL Kernel Library)**
  - Part of OpenGL
  - Use the prefix of gl (ex: glBegin())
- **GLU (OpenGL Utility Library)**
  - Part of OpenGL
  - Use the prefix of glu (ex: gluLookAt())
- **GLUT (OpenGL Utility Toolkit)**
  - Not officially part of OpenGL
  - Provide common features for window system
  - create window, mouse and keyboard, menu, event-driven
  - Lack of modern GUI support (e.g. scroller)
  - Use the prefix of glut (ex: glutDisplayFunc())



glCallList
glCallLists
glColor
glEdgeFlag
glEnd
glEvalCoord



gluDisk
gluNewQuadric
gluPartialDisk
gluQuadricOrientation
gluQuadricTexture
gluSphere



int	glutCreateWindow (const char *title)
void	glutDestroyWindow (int windowID)
void	glutFullScreen (void)
int	glutGetWindow (void)
void *	glutGetWindowData (void)
void	glutHideWindow (void)
void	glutIconifyWindow (void)
void	glutInitDisplayMode (unsigned int displayMode)

Installing GLUT - The OpenGL Utility Toolkit

- On Windows:
  - Download from OpenGL website:
  - https://www.opengl.org/resources/libraries/glut/glut_downloads.php
  - glut-3.7.6-bin has the dll/lib/header that are required
  - Copy glut.dll to {Windows DLL dir}\glut32.dll
  - Copy glut.lib to {VC++ lib path}\glut32.lib
  - Copy glut.h to {VC++ include path}\GL\glut.h

- freeglut：
  - http://freeglut.sourceforge.net/

Using GLUT

- Only need to include glut.h
  - #include <GL\glut.h>
  - Automatically includes gl.h and glu.h

- LearnOpenGL CN
  - https://learnopengl-cn.github.io/

欢迎来到OpenGL的世界

欢迎来到OpenGL的世界。这个工程只是我(Joey de Vries)的一次小小的尝试，希望能够建立起一个完善的OpenGL教学平台。无论你学习OpenGL是为了学业，找工作，或仅仅是因为兴趣，这个网站都将能够教会你**现代**(Core-profile) OpenGL从基础，中级，到高级的知识。LearnOpenGL的目标是使用易于理解的形式，使用清晰的例子，展现现代OpenGL的所有知识点，并与此同时为你以后的学习提供有用的参考。

如果您喜欢这个系列教程的话，不妨向Joey de Vries的Paypal进行捐赠，支持一下作者，让这个教程能够持续完善并更新。

为什么要阅读这些教程呢？

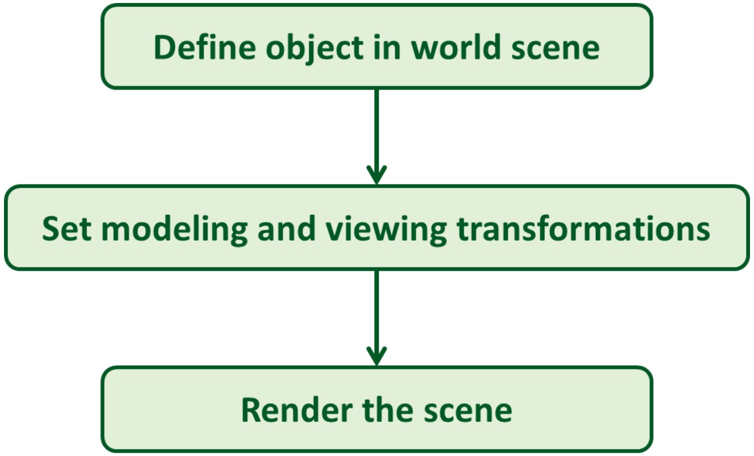
在互联网上，有关学习OpenGL的有成千上万的文档与资源，然而其中大部分的资源仅仅讨论了OpenGL的立即渲染模式（Immediate Mode，通常会说旧OpenGL），亦或是不完整，缺少适当的文档，甚至是仅仅不适合你的口味。所以，我的目标是提供一个既完整，又易懂的平台供人们学习。

如果你很享受那些提供手把手指导的教程，那些提供清晰例子的教程，以及那些不会一下将你淹没在细节中的教程，那么我的这些教程很可能就很适合你。我的教程旨在让那些没有图形编程经验的人们能够理解，又让那些有经验的读者有阅读下去的兴趣。我的教程同样也讨论了一些常用的概念，只需要你再有一点创造力，就能将你的想法变成真正的3D程序。如果如果你觉得前面这些讲的都是你，欢迎继续阅读我的教程。

How OpenGL Works

- OpenGL is a state machine
  - You give it orders to set the current state of any one of its internal variables, or to query for its current status
  - The current state won't change until you specify otherwise
  - Each of the system's state variables has a default value

Stages in OpenGL



Define object in world scene

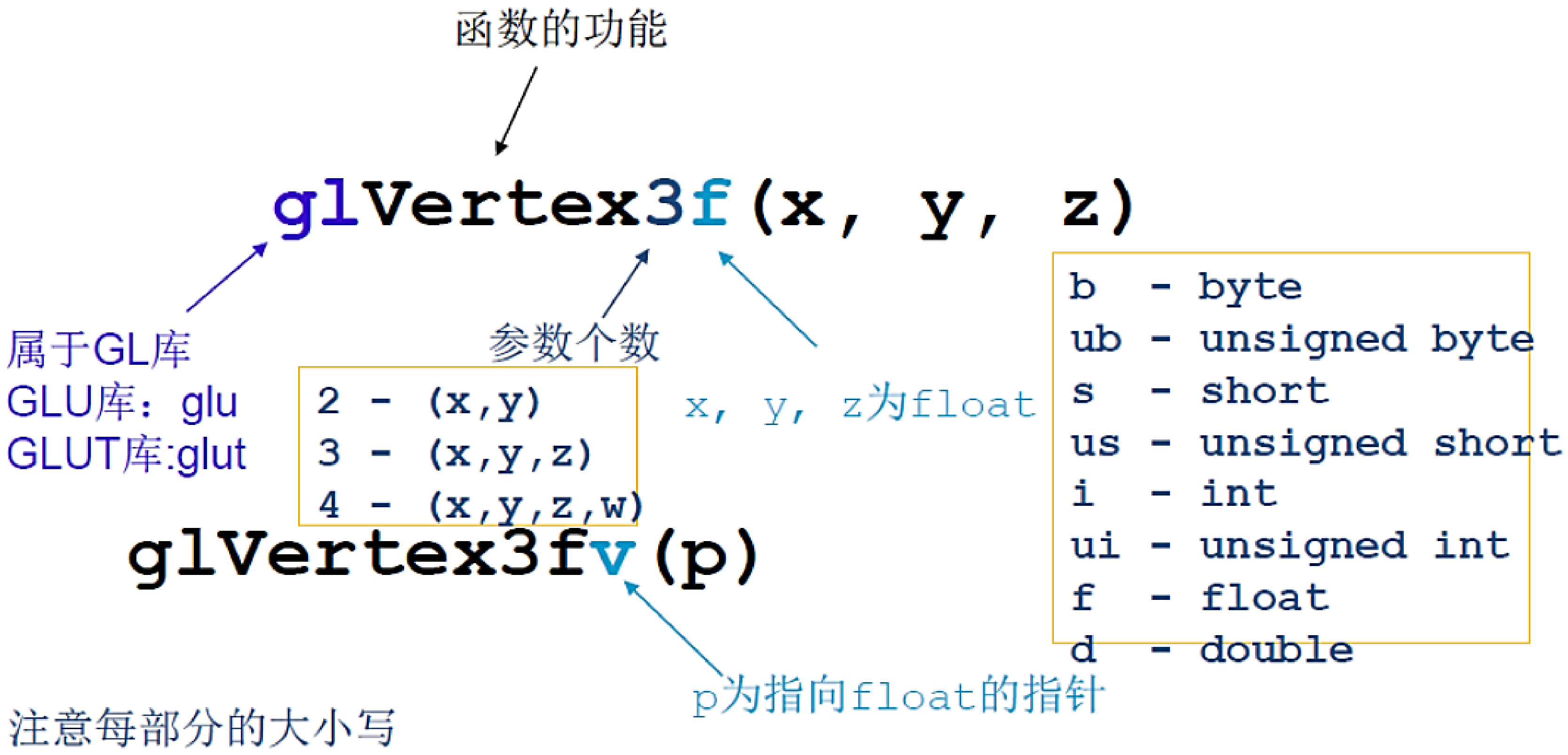
Set modeling and viewing transformations

Render the scene

Functions of OpenGL

- Primitive - WHAT - Point, Edge, Polygon
- Attribute - HOW
- Transformation - Viewing & Modeling
- Input - provided by GLUT
- Control - provided by GLUT
- Query

Function Format of OpenGL



函数的功能

glVertex3f(x, y, z)

属于GL库
GLU库：glu
GLUT库:glut

参数个数

2	-	(x,y)
3	-	(x,y,z)
4	-	(x,y,z,w)

x, y, z为float

b	-	byte
ub	-	unsigned byte
s	-	short
us	-	unsigned short
i	-	int
ui	-	unsigned int
f	-	float
d	-	double

glVertex3fv(p)

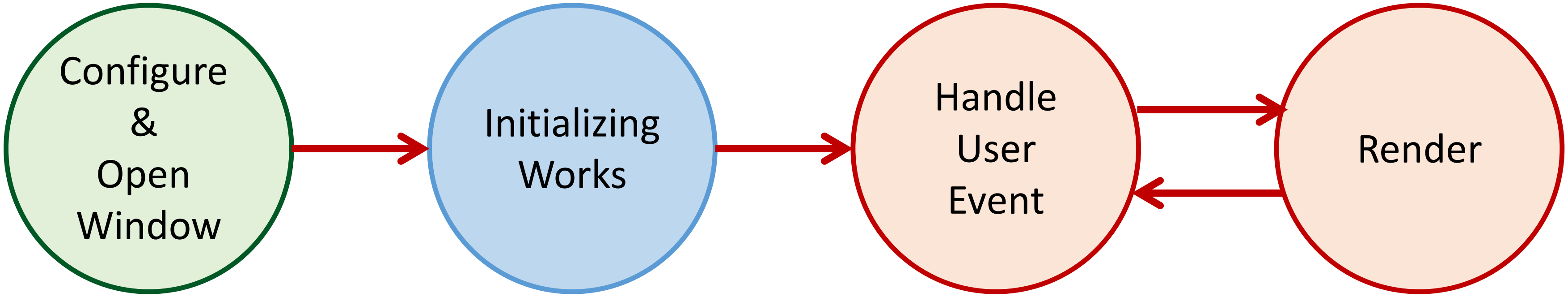
p为指向float的指针

注意每部分的大小写

OpenGL Hello World

- Prerequisite
- Head Files:
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GL/glut.h>
- Library Files:
  - Compiled files folder\opengl32.lib glu32.lib glut32.lib
  - C:\Windows\System32\opengl32.dll glu32.dll glut32.dll

Basic Structure Of OpenGL Program



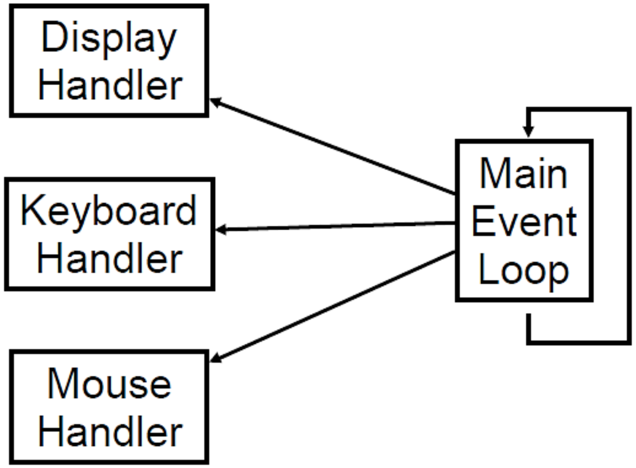
Configure & Open Window

Initializing Works

Handle User Event

Render

- NOT Object-Oriented!!
- Use states to control
- Infinite Loop



Display Handler

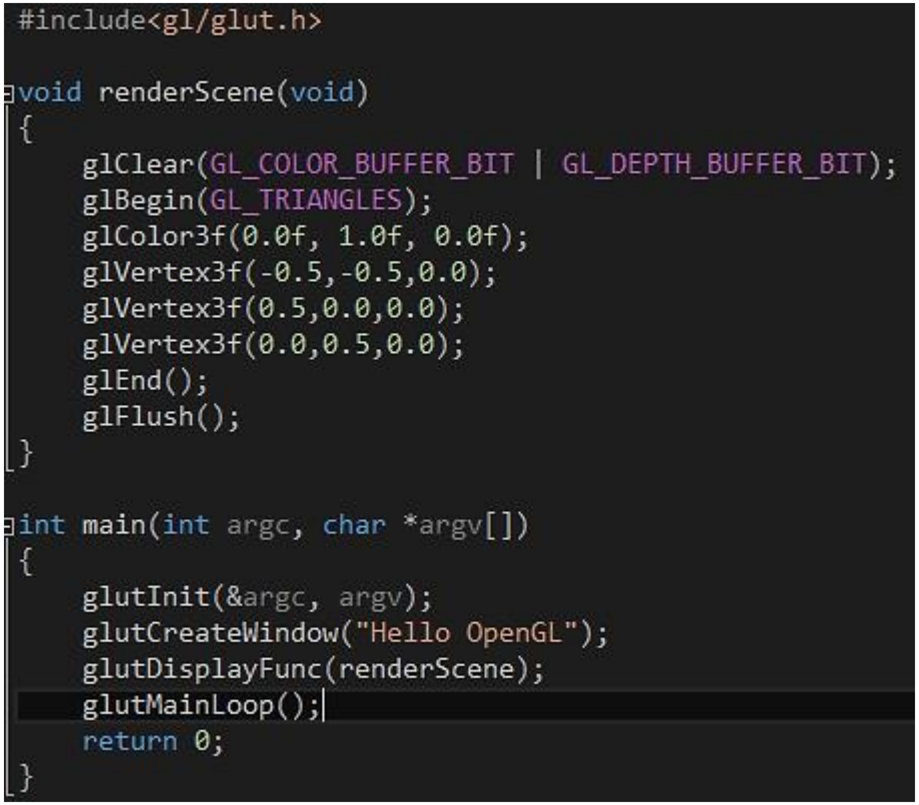
Keyboard Handler

Mouse Handler

Main Event Loop

Event Driven Programming

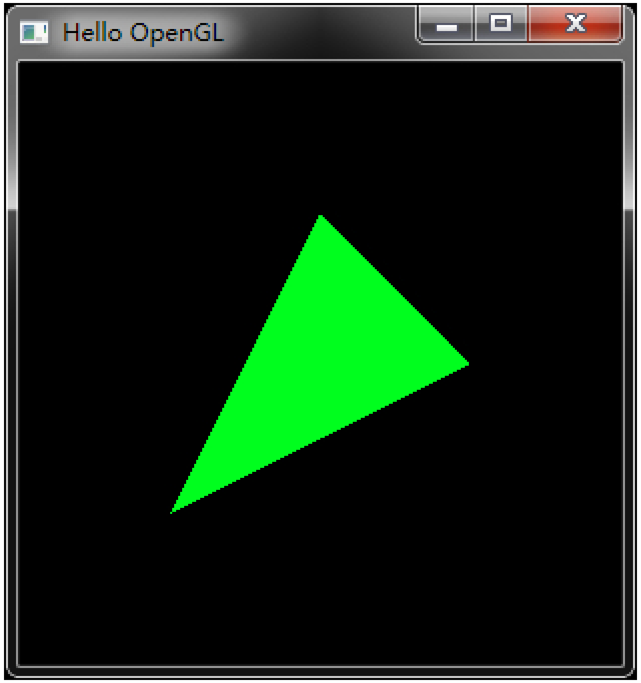
2D demo



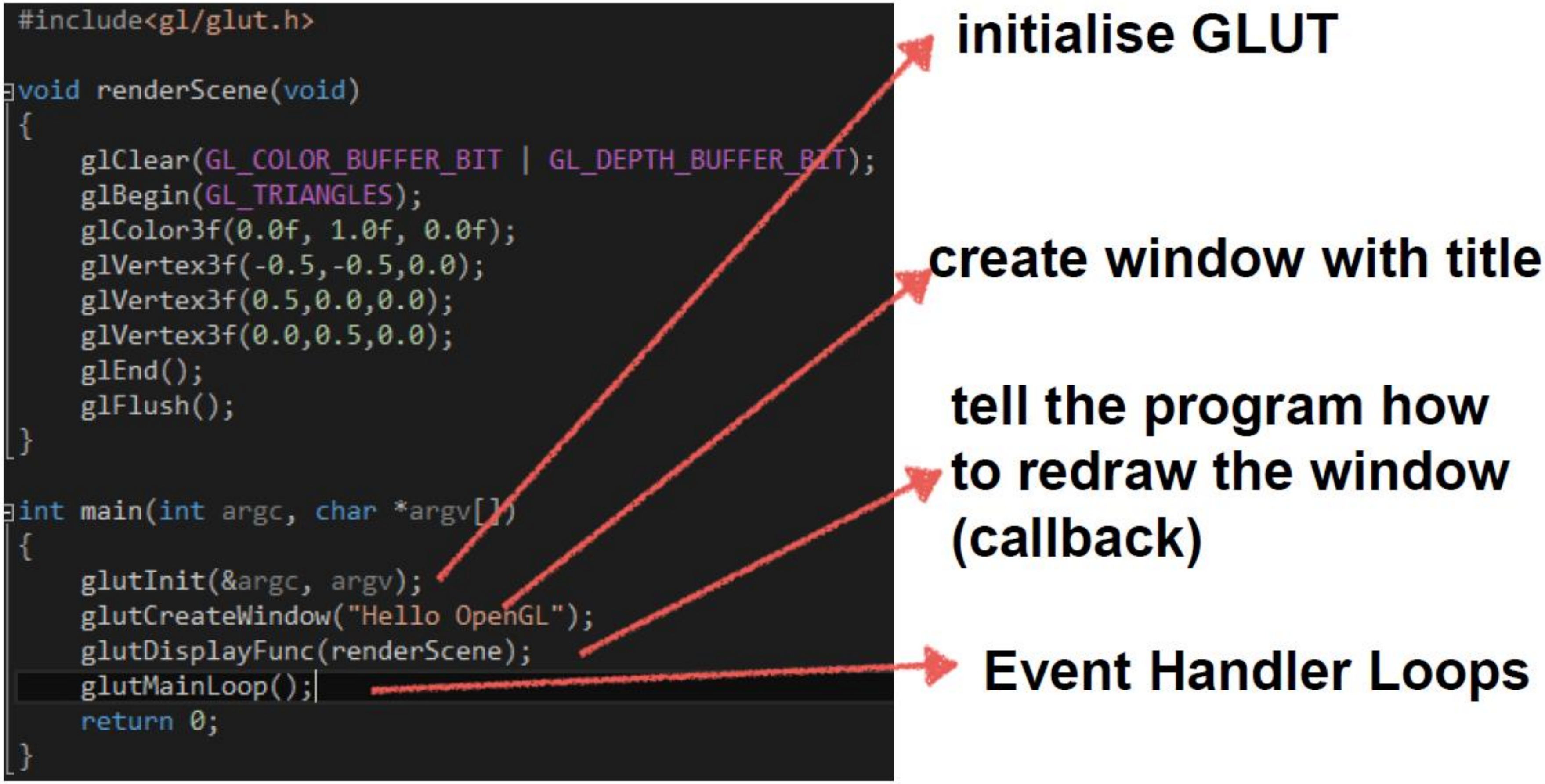
```
#include<gl/glut.h>

void renderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f(-0.5,-0.5,0.0);
    glVertex3f(0.5,0.0,0.0);
    glVertex3f(0.0,0.5,0.0);
    glEnd();
    glFlush();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutCreateWindow("Hello OpenGL");
    glutDisplayFunc(renderScene);
    glutMainLoop();
    return 0;
}
```

Less than 20 lines!

Not that HARD



2D demo



```
#include<gl/glut.h>

void renderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f(-0.5,-0.5,0.0);
    glVertex3f(0.5,0.0,0.0);
    glVertex3f(0.0,0.5,0.0);
    glEnd();
    glFlush();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutCreateWindow("Hello OpenGL");
    glutDisplayFunc(renderScene);
    glutMainLoop();
    return 0;
}
```

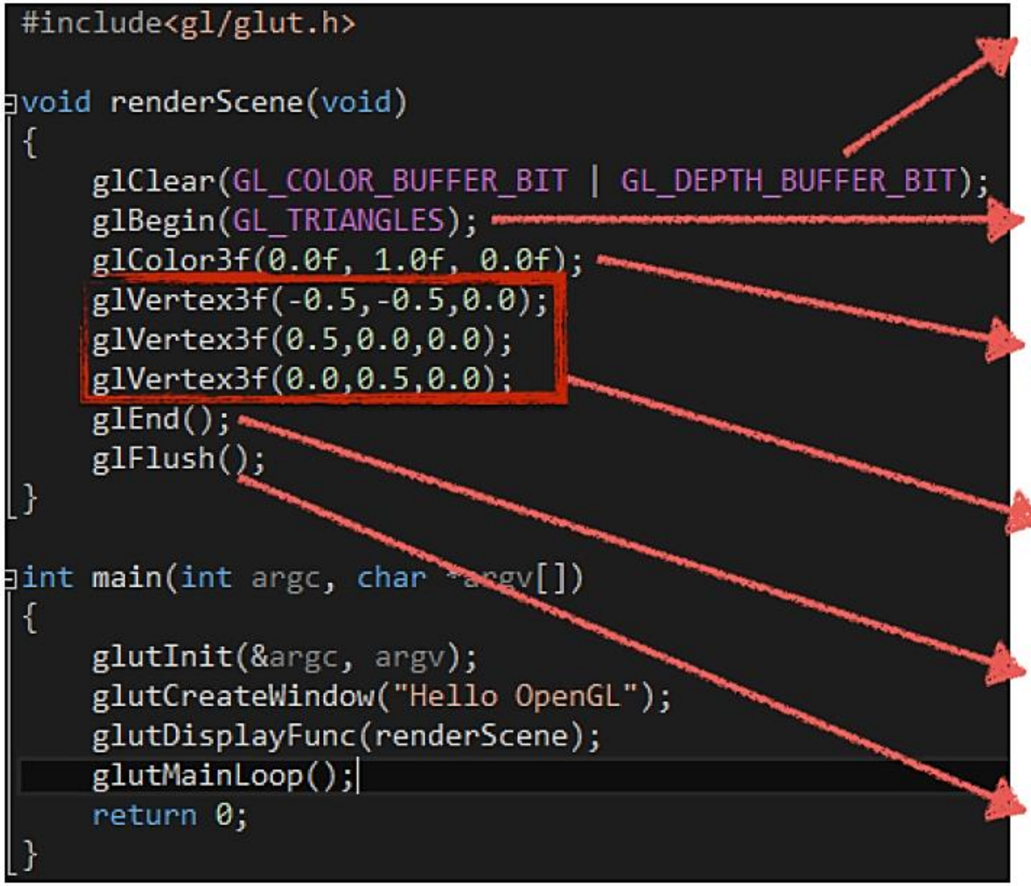
initialise GLUT

create window with title

tell the program how to redraw the window (callback)

Event Handler Loops

2D demo



```
#include<gl/glut.h>

void renderScene(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glBegin(GL_TRIANGLES);
    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f(-0.5,-0.5,0.0);
    glVertex3f(0.5,0.0,0.0);
    glVertex3f(0.0,0.5,0.0);
    glEnd();
    glFlush();
}

int main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutCreateWindow("Hello OpenGL");
    glutDisplayFunc(renderScene);
    glutMainLoop();
    return 0;
}
```

clear the buffer

let’s draw a triangle

using RGB color green

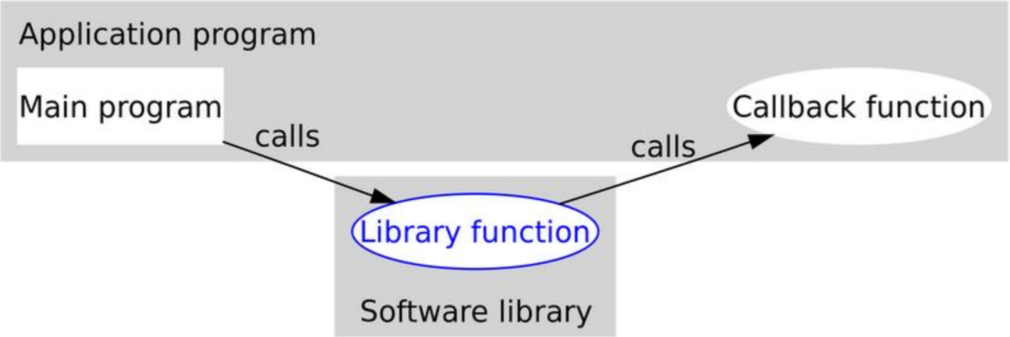
this is the 3 points of the triangle

end of drawing

Do it!

Callbacks

- Wiki: In computer programming, a callback is a reference to a piece of executable code, that is passed as an argument to other code. This allows a lower-level software layer to call a subroutine (or function) defined in a higher-level layer.
- Usage
  - Callbacks allow the user of a function to fine-tune it at runtime, another use is in error signaling.
  - Callbacks may also be used to control whether a function acts or not.
- In C/C++: function pointer

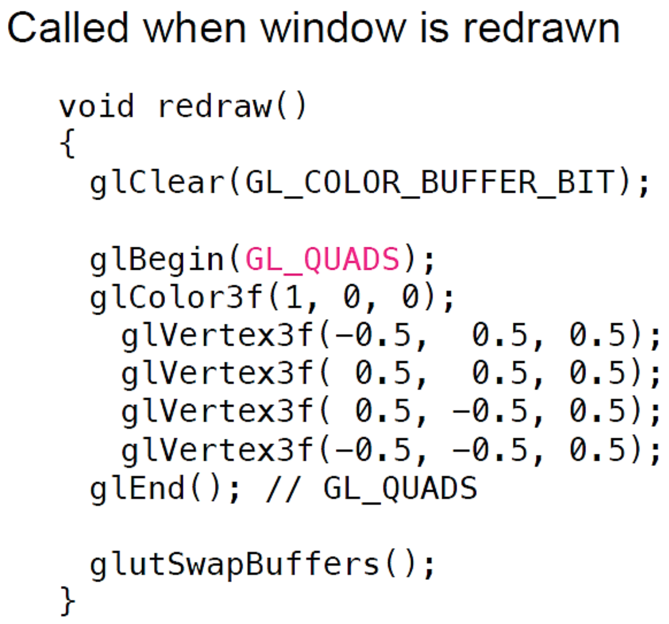


Callbacks

- Typically, the **main thread** will **just run in a loop**, **waiting for events to occur** - for example, for the user to move his mouse in your window, or click one of your buttons.
- The GUI framework will provide a mechanism for you **to pass it function pointers**, which it will then associate with certain events. When an event occurs, the event loop will invoke any callback functions you've provided for that event.
- Often, the callback function will **have parameters**, and the event dispatcher (事件调度器) will **provide you with extra information** about the event (perhaps the exact x,y coordinates of the mouse, for example) through the arguments it calls your callback function with.

Callback

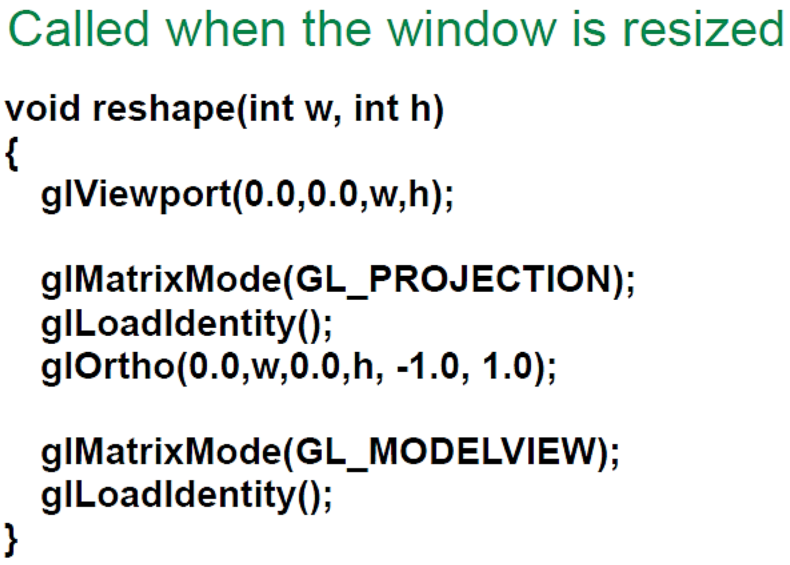
- Display Callback



Called when window is redrawn

```
void redraw()
{
  glClear(GL_COLOR_BUFFER_BIT);

  glBegin(GL_QUADS);
  glColor3f(1, 0, 0);
    glVertex3f(-0.5,  0.5, 0.5);
    glVertex3f( 0.5,  0.5, 0.5);
    glVertex3f( 0.5, -0.5, 0.5);
    glVertex3f(-0.5, -0.5, 0.5);
  glEnd(); // GL_QUADS

  glutSwapBuffers();
}
```

- Reshape Callback

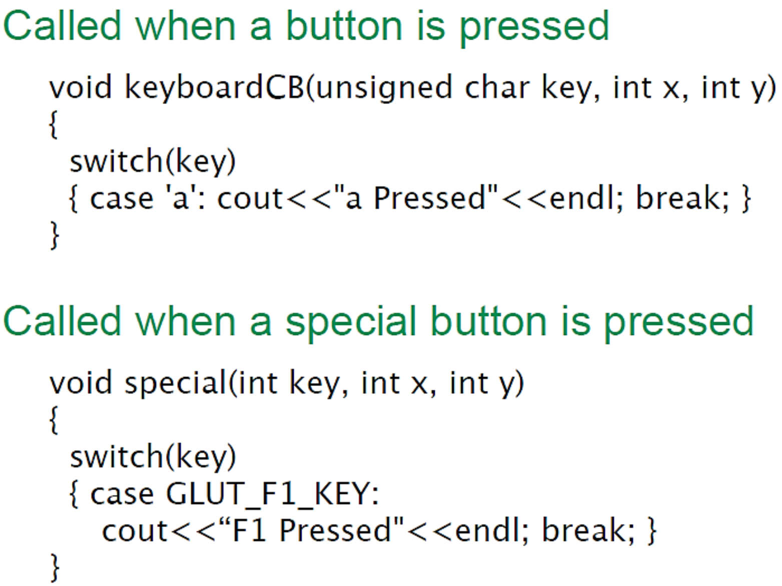


Called when the window is resized

```
void reshape(int w, int h)
{
   glViewport(0.0,0.0,w,h);

   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(0.0,w,0.0,h, -1.0, 1.0);

   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
}
```

Callback

- Keyboard Callback



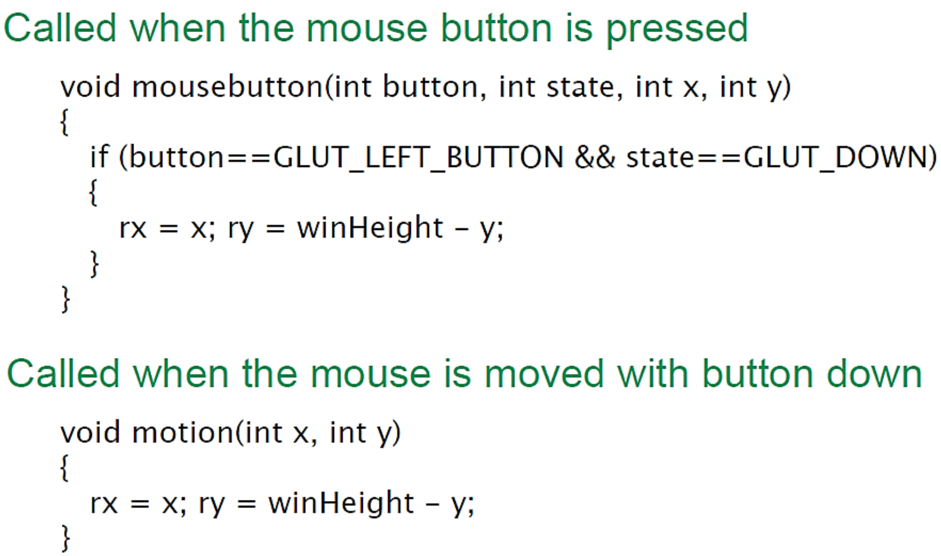
Called when a button is pressed

```
void keyboardCB(unsigned char key, int x, int y)
{
  switch(key)
  { case 'a': cout<<"a Pressed"<<endl; break; }
}
```

Called when a special button is pressed

```
void special(int key, int x, int y)
{
  switch(key)
  { case GLUT_F1_KEY:
      cout<<“F1 Pressed"<<endl; break; }
}
```

- Mouse Callback



Called when the mouse button is pressed

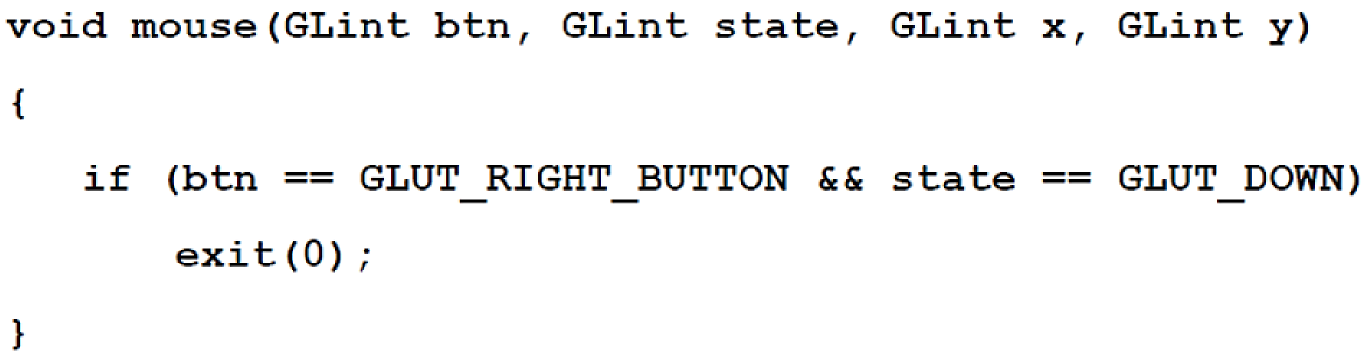
```
void mousebutton(int button, int state, int x, int y)
{
   if (button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
   {
      rx = x; ry = winHeight – y;
   }
}
```

Called when the mouse is moved with button down

```
void motion(int x, int y)
{
   rx = x; ry = winHeight – y;
}
```

Closing the program

- There is no idea to close the current program by OpenGL in previous programs.
- However, we can do the close operation by simple mouse callback.



```
void mouse(GLint btn, GLint state, GLint x, GLint y)
{
   if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
       exit(0);
}
```

OpenGL - GLUT Example

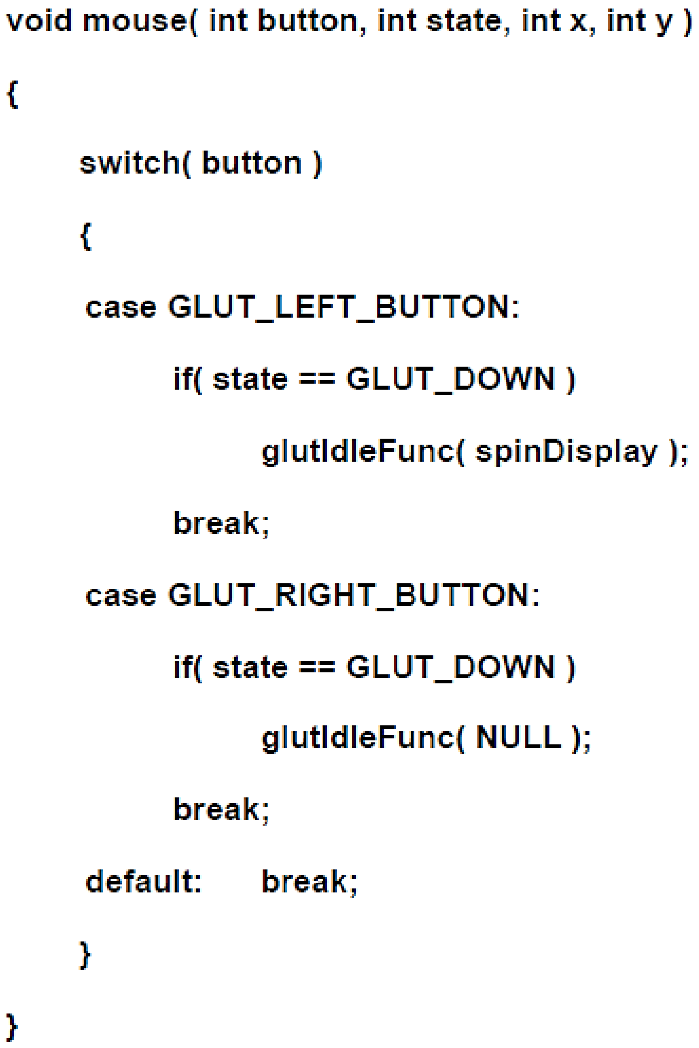
```
#include <gl/glut.h>
#include <stdlib.h>
static GLfloat spin = 0.0;
void init( void )
{
  glClearColor( 0.0, 0.0, 0.0, 0.0 );
  glShadeModel( GL_FLAT );
}
```

```
void reshape( int w, int h )
{
     glViewport( 0, 0, (GLsizei) w, (GLsizei)
     h );
     glMatrixMode( GL_PROJECTION );
     glLoadIdentity();
     glOrtho( -50.0, 50.0, -50.0, 50.0, -1.0, 1.0 );
     glMatrixMode( GL_MODELVIEW );
     glLoadIdentity();
}
```

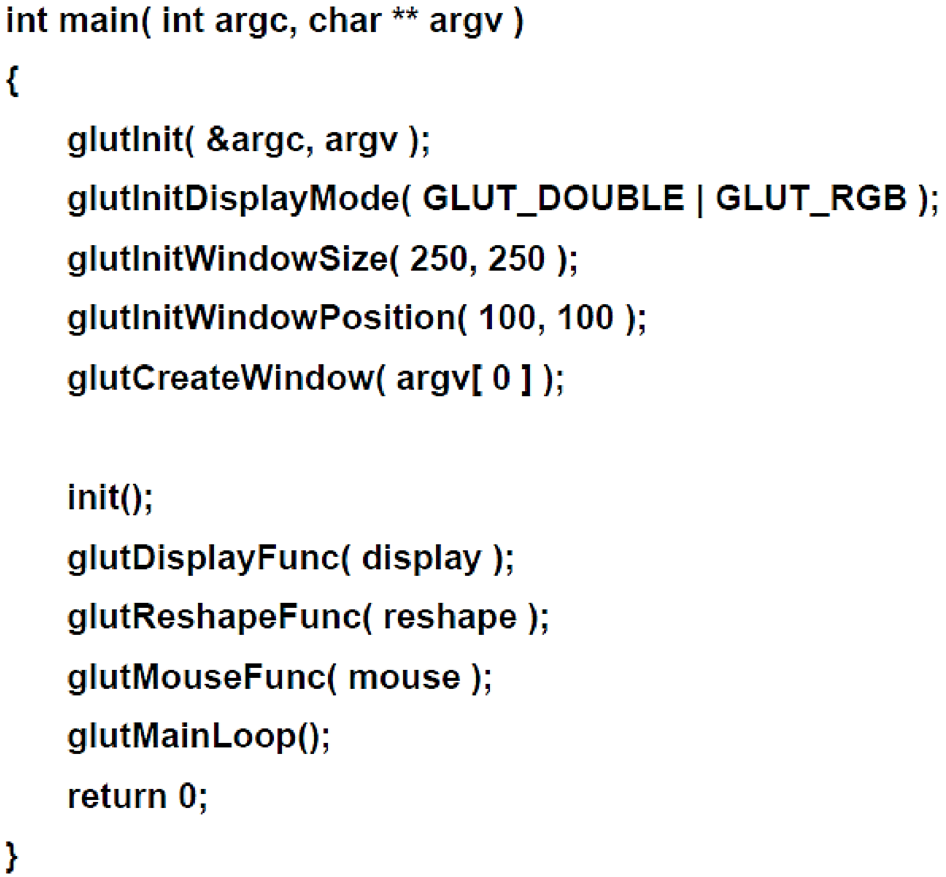
```
void spinDisplay( void )
{
     spin += 2.0;
     if( spin > 360.0 )
      spin -= 360.0;
     glutPostRedisplay();
}
```

```
void display( void )
{
glClear( GL_COLOR_BUFFER_BIT );
glPushMatrix();
glRotatef( spin, 0.0, 0.0, 1.0 );
glColor3f( 1.0, 1.0, 1.0 );
glRectf( -25.0, -25.0, 25.0, 25.0 );
glPopMatrix();
glutSwapBuffers();
}
```

OpenGL - GLUT Example



```
void mouse( int button, int state, int x, int y )
{
    switch( button )
    {
     case GLUT_LEFT_BUTTON:
          if( state == GLUT_DOWN )
               glutIdleFunc( spinDisplay );
          break;
     case GLUT_RIGHT_BUTTON:
          if( state == GLUT_DOWN )
               glutIdleFunc( NULL );
          break;
     default:     break;
    }
}
```

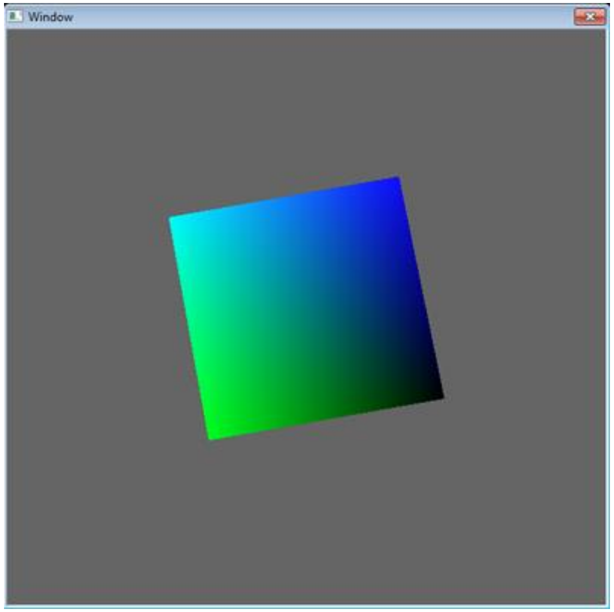


```
int main( int argc, char ** argv )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_DOUBLE | GLUT_RGB );
    glutInitWindowSize( 250, 250 );
    glutInitWindowPosition( 100, 100 );
    glutCreateWindow( argv[ 0 ] );

    init();
    glutDisplayFunc( display );
    glutReshapeFunc( reshape );
    glutMouseFunc( mouse );
    glutMainLoop();
    return 0;
}
```

Details of OpenGL Program

Contexts and Viewports?

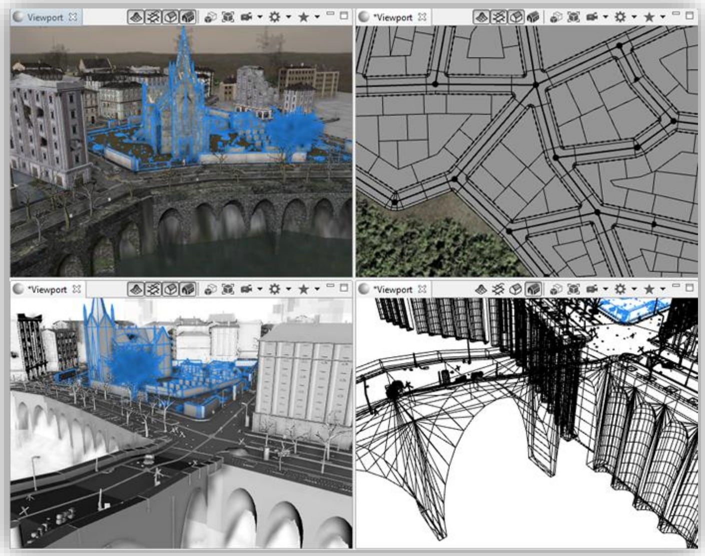
- Each OpenGL application creates a context to issue rendering commands to.
- The application must also define a viewport, a region of pixels on the screen that can see the context.
- Can be
  - Part of a window
  - An entire window
  - The whole screen



Viewport

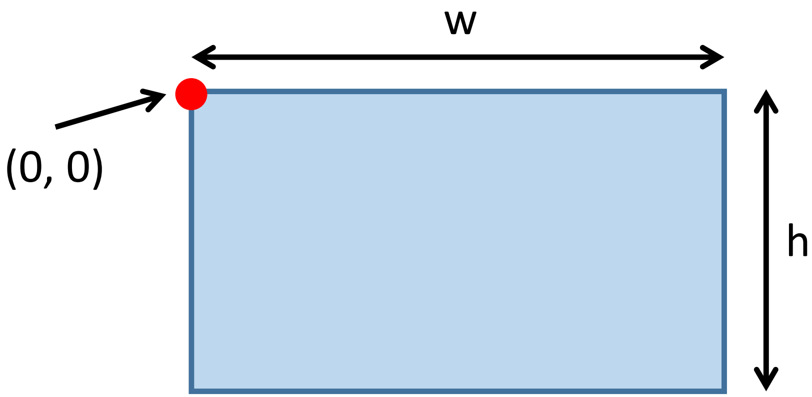
- The viewport is the part of the window your drawing is displayed to
  - By default, the viewport is the entire window
- Modifying the viewport is analogous to changing the size of the final picture
  - From the camera analogy
- Can have multiple viewports in the

  same window for a split-screen effect



Position (定位)

- 在屏幕上的位置通常是以pixel为单位，原点在左上角
  - 原因在于显示器是以自顶向下的方式刷新显示内容
- 在OpenGL中应用一个世界坐标系(World Coordinate)，其原点在左下角
- 在这个坐标系中的y坐标需要从窗口高度中减去Callback Function返回的y值：
  - **y := h - y**

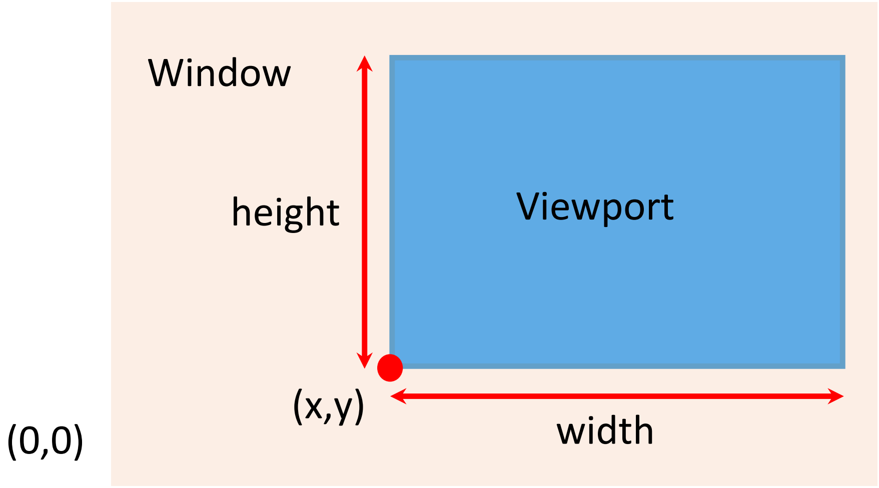


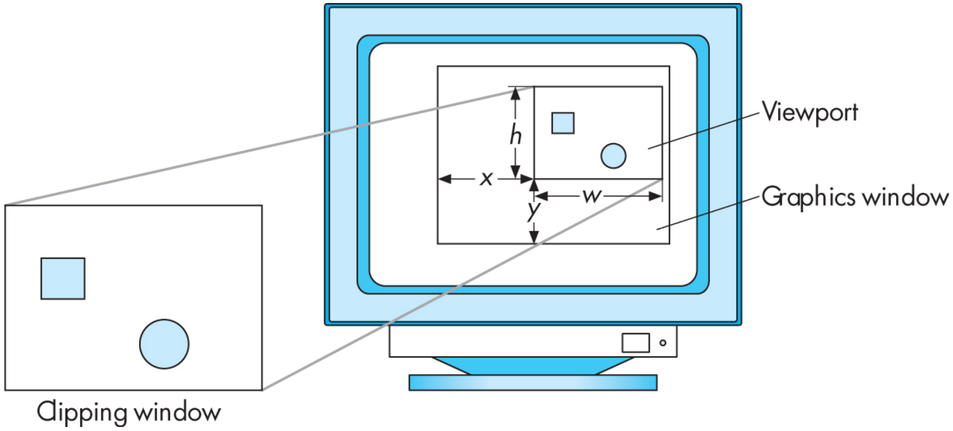
Get the height of window

- To finish the change of y coordinate, we need to know the window size.
  - The height would be changed in the procedure of the program running.
  - Need a global variant to track the changing.
  - The new height will return a callback function for shape changing.
  - Also use the glGetIntv() and glGetFloat() to obtain.

Setting the Viewport

- glViewport(int x, int y, int width, int height)
  - (x, y) is the location of the origin (lower-left) within the window
  - (width, height) is the size of the viewport
- The aspect ratio of the viewport should be the **same** as that of the viewing volume





OpenGL as a State Machine

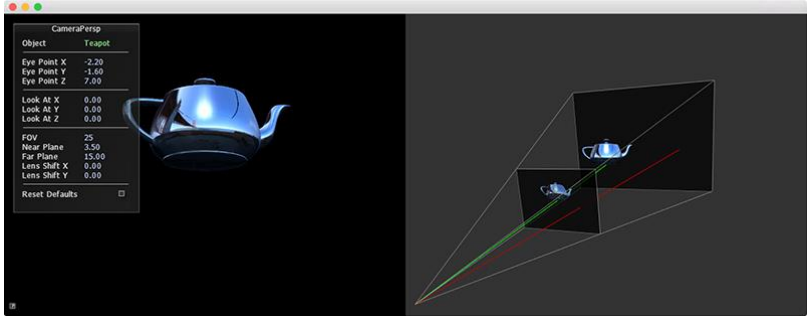
- Put a value into various states, then it will remain in effect until being changed.
    - e.g. glColor*()
- Many state variables are enabled or disabled with

  glEnable() or glDisable()

    - e.g. glEnable(GL_LIGHT0)

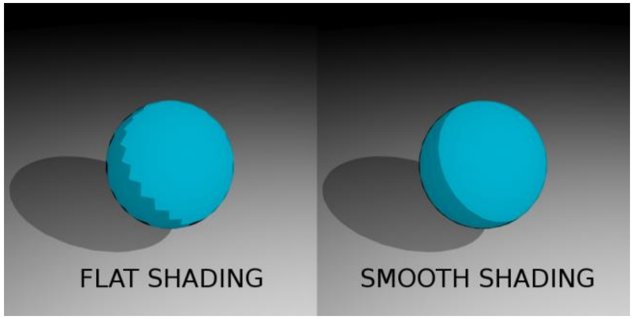
OpenGL State

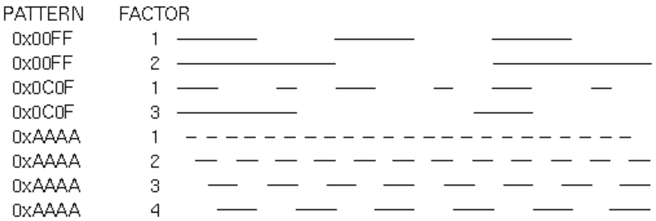
- Some attributes of the OpenGL state
  - Current color
  - Camera properties (location, orientation, field of view, etc.)
  - Lighting model (flat, smooth, etc.)
  - Type of primitive being drawn
  - Line width, dotted line or full line,…
  - And many more...



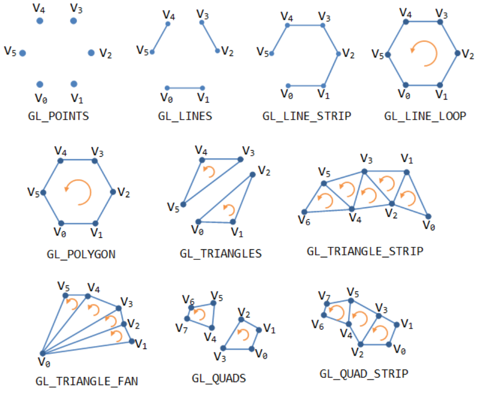
CameraPersp

Object	Teapot
Eye Point X	-2.20
Eye Point Y	-1.60
Eye Point Z	7.00
Look At X	0.00
Look At Y	0.00
Look At Z	0.00
FOV	25
Near Plane	3.50
Far Plane	15.00
Lens Shift X	0.00
Lens Shift Y	0.00
Reset Defaults	□





PATTERN	FACTOR
0x00FF	1
0x00FF	2
0x0C0F	1
0x0C0F	3
0xAAAA	1
0xAAAA	2
0xAAAA	3
0xAAAA	4



OpenGL Primitives

OpenGL Input

- All inputs (i.e. geometry) to an OpenGL context are defined as vertex lists
- glVertex (*)
  - * = nt OR ntv
  - n - number (2, 3, 4)
  - t - type (i = integer, f = float, etc.)
  - v - vector

OpenGL Types

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned long	GLuint, GLenum, GLbitfield

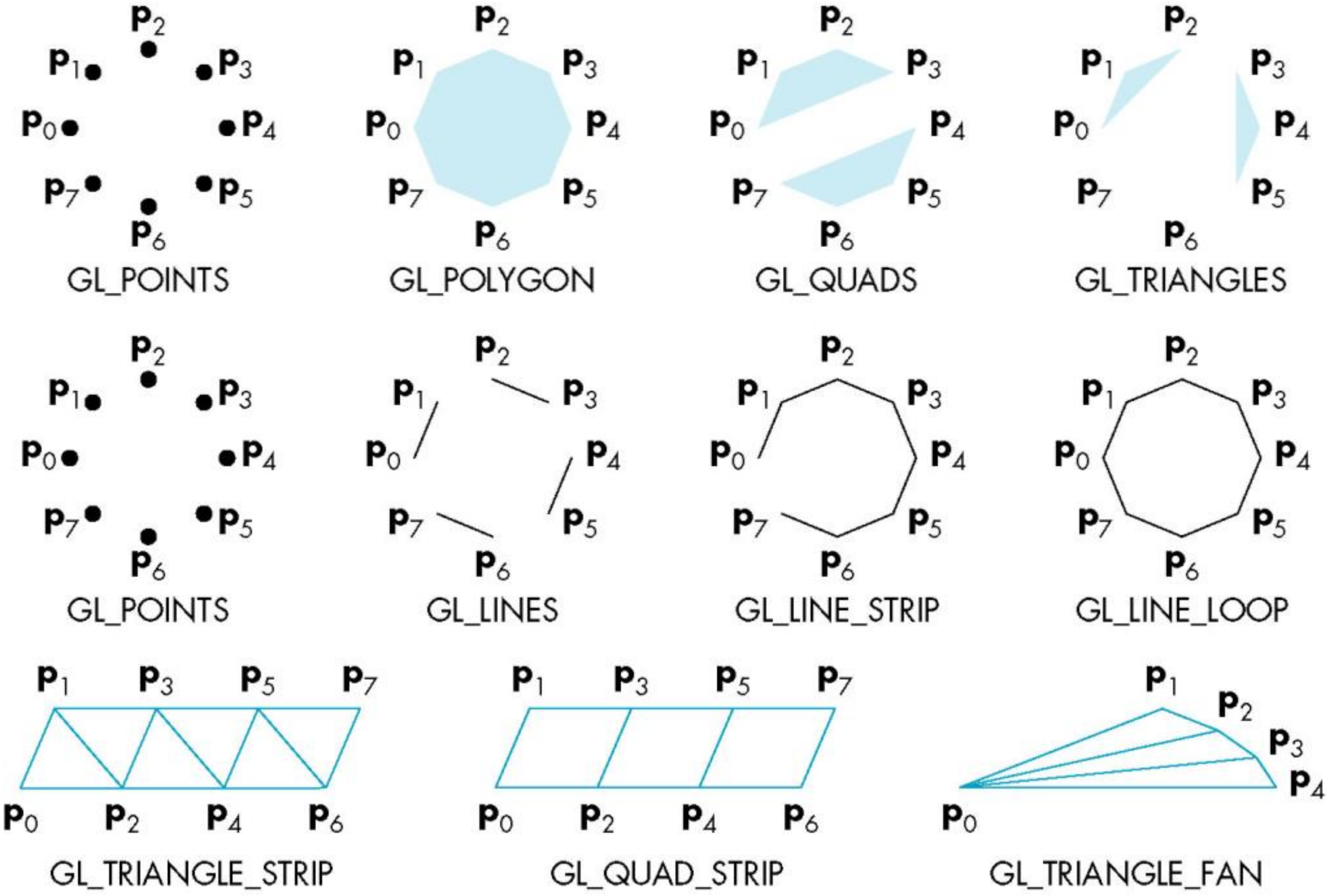
OpenGL Input

- Examples:
  - glVertex2i(5, 4);
    - Specifies a vertex at location (5, 4) on the z = 0 plane
    - “2” tells the system to expect a 2-vector (a vertex defined in 2D)
    - “i” tells the system that the vertex will have integer locations
  - glVertex3f(.25, .25, .5);
  - double vertex[3] = {1.0, .33, 3.14159};
    glVertex3dv(vertex);
    - “v” tells the system to expect the coordinate list in a single data structure, instead of a list of n numbers

OpenGL Primitive Types

- All geometry is specified by vertex lists
  - But can draw multiple types of things
    - Points
    - Lines
    - Triangles
    - etc.
- The different things the system knows how to draw are the system **primitives**

OpenGL Primitive Types



GL_POINTS

GL_POLYGON

GL_QUADS

GL_TRIANGLES

GL_POINTS

GL_LINES

GL_LINE_STRIP

GL_LINE_LOOP

GL_TRIANGLE_STRIP

GL_QUAD_STRIP

GL_TRIANGLE_FAN

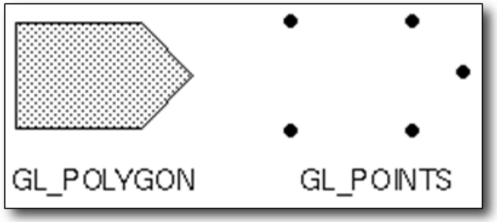
Specifying the OpenGL Primitive Type

- glBegin(primitiveType);

  ```
          // A list of glVertex* calls goes here
          // ...
  glEnd();
  ```

- primitiveType can be any of several things

```
glBegin(GL_POLYGON);
  glVertex2f(0.0, 0.0);
  glVertex2f(0.0, 3.0);
  glVertex2f(3.0, 3.0);
  glVertex2f(4.0, 1.5);
  glVertex2f(3.0, 0.0);
glEnd();
```



Color in OpenGL

- OpenGL colors are typically defined as RGB components
  - each of which is a float in the range [0.0, 1.0]

- For the screen's background:
  - glClearColor(0.0, 0.0, 0.0); // black color
  - glClear(GL_COLOR_BUFFER_BIT);
- For objects:
  - glColor3f(1.0, 1.0, 1.0); // white color

- GLUT_RGB and GLUT_RGBA
- alpha channel

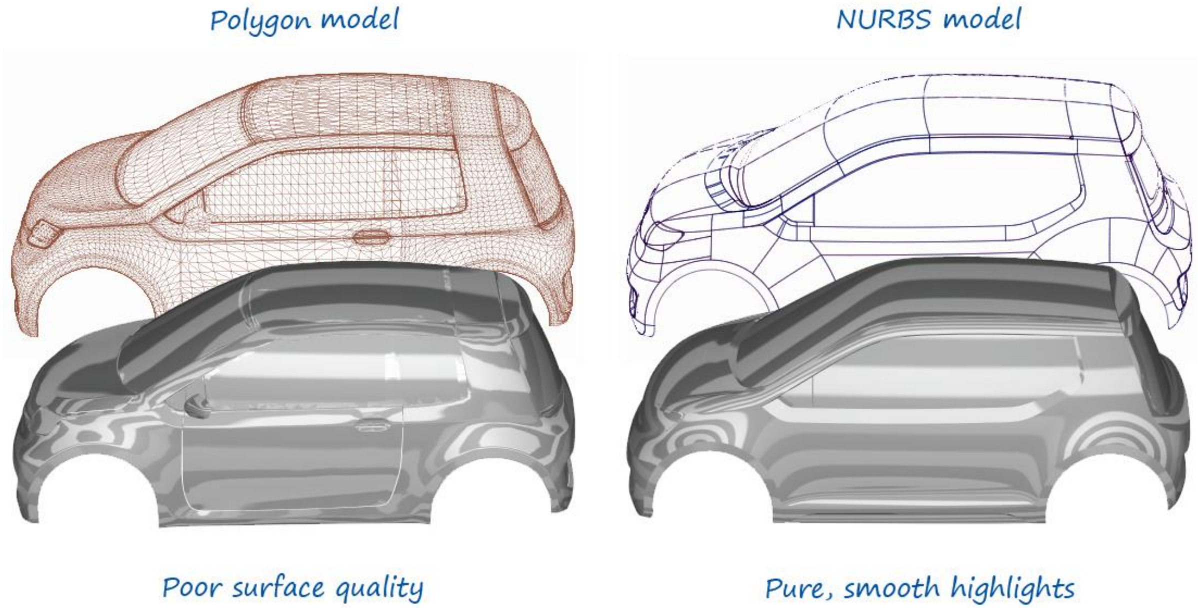
- glColor3f (1.0, 1.0, 1.0);
- glColor3i (0, 255, 255);
- glColor3fv (colorArray);

Polygon Display Modes

- glPolygonMode(GLenum face, GLenum mode);
  - Faces: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
  - Modes: GL_FILL, GL_LINE, GL_POINT
  - By default, both the front and back face are drawn filled
- glFrontFace(GLenum mode);
  - Mode is either GL_CCW (default) or GL_CW
- glCullFace(Glenum mode);
  - Mode is either GL_FRONT, GL_BACK, GL_FRONT_AND_BACK;
- You must enable and disable culling with
  - glEnable(GL_CULL_FACE) or glDisable(GL_CULL_FACE);

Drawing Other Objects

- GLU contains calls to draw cylinders, cons, and more complex surfaces called NURBS.
- GLUT contains calls to draw spheres and cubes.



Finishing Up Your OpenGL Program

- OpenGL commands are not executed immediately
  - They are put into a command buffer that gets fed to the hardware
- When you’re done drawing, need to send the commands to the graphics hardware
  - glFlush() or glFinish()
- glFlush();
  - Forces all issued commands to begin execution
  - Returns immediately (asynchronous)
- glFinish();
  - Forces all issued commands to begin execute
  - Does not return until execution is complete (synchronous)

Matrices in OpenGL

- Vertices are transformed by 2 matrices:
  - **ModelView**
    - Maps 3D to 3D
    - Transforms vertices from object coordinates to eye coordinates
  - **Projection**
    - Maps 3D to 2D (sort of)
    - Transforms vertices from eye coordinates to clip coordinates

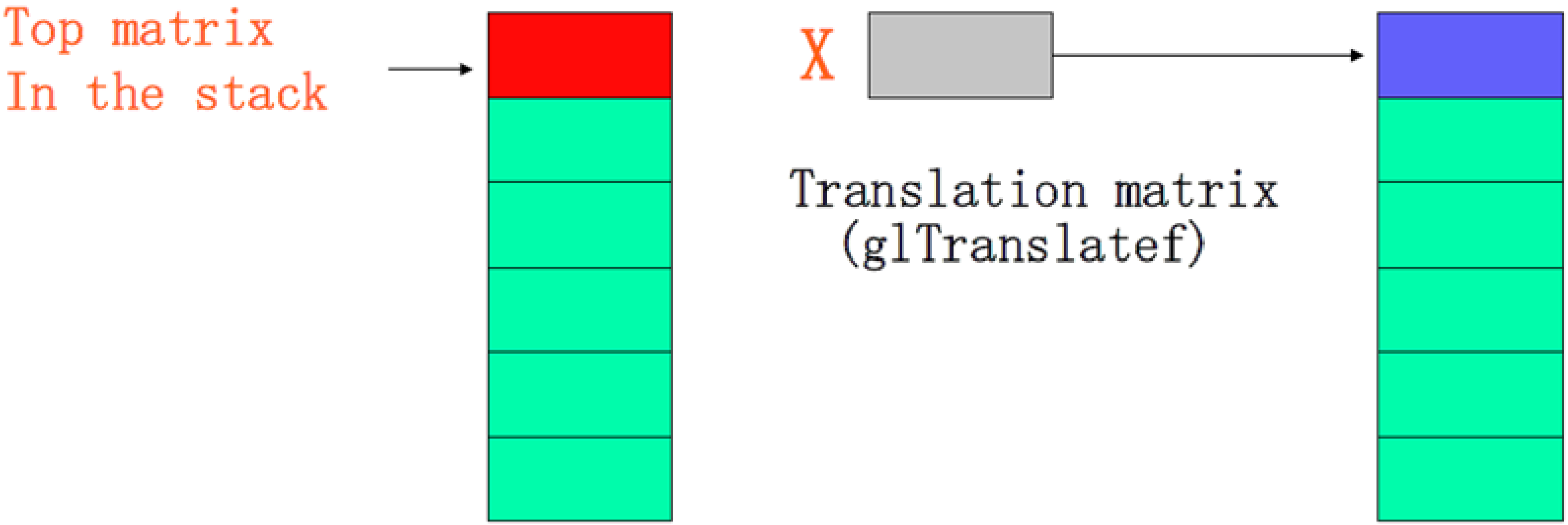
Matrix in OpenGL

- There are two matrix stacks.
  - ModelView matrix (GL_MODELVIEW)
  - Projection matrix (GL_PROJECTION)
- When we call functions of transformation, we should change to the appropriate matrix stack first.

```
glMatrixMode(GL_MODELVIEW);
//now we are in modelview matrix stack!
//do modelview transformation here…..
glMatrixMode(GL_PROJECTION);
//now we are in projection matrix stack!
//do projection transformation here….
```

Matrix in OpenGL

- Matrix multiplications always apply to the top of matrix stack.



WARNING! OpenGL Matrices

- In C/C++, we are used to row-major matrices
- In OpenGL, matrices are specified in column-major order

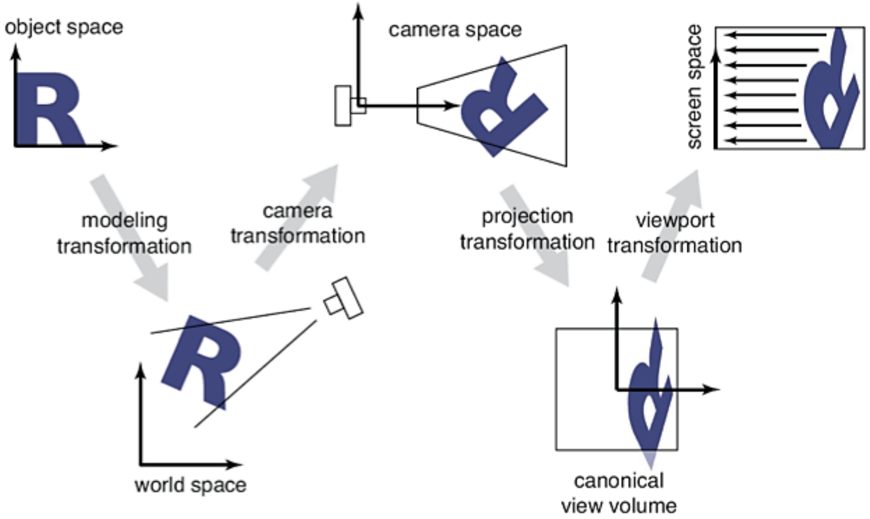
$$\overset{\xrightarrow{\hspace{6em}}}{\begin{bmatrix} A_0 & A_1 & A_2 & A_3 \\ A_4 & A_5 & A_6 & A_7 \\ A_8 & A_9 & A_{10} & A_{11} \\ A_{12} & A_{13} & A_{14} & A_{15} \end{bmatrix}} \qquad \Big\downarrow \begin{bmatrix} A_0 & A_4 & A_8 & A_{12} \\ A_1 & A_5 & A_9 & A_{13} \\ A_2 & A_6 & A_{10} & A_{14} \\ A_3 & A_7 & A_{11} & A_{15} \end{bmatrix}$$

Row-Major Order

Column-Major Order

The ModelView Matrix

- Modeling Transformation
  - Perform rotate, translate, scale and combinations of these transformations to the object.
- Viewing Transformation
  - To positioning and aiming the camera



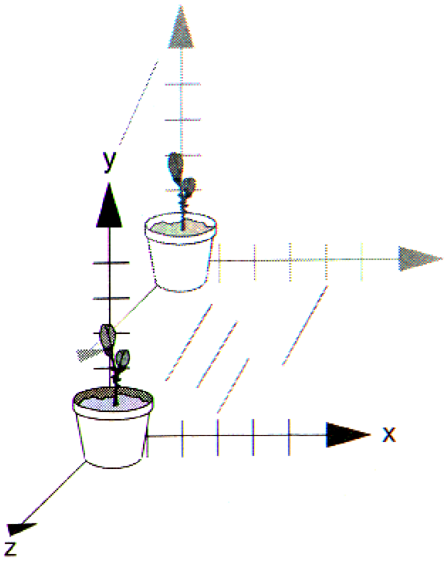
The ModelView Matrix

- In OpenGL, the viewing and modeling transforms are combined into a single matrix - the modelview matrix
  - Viewing Transform - positioning the camera
  - Modeling Transform - positioning the object
- Why?
  - Consider how you would “translate” a fixed object with a real camera

Modeling Transformations

- glTranslate{fd}(x, y, z)
  - Multiplies current matrix by a matrix that moves an object by x,y,z

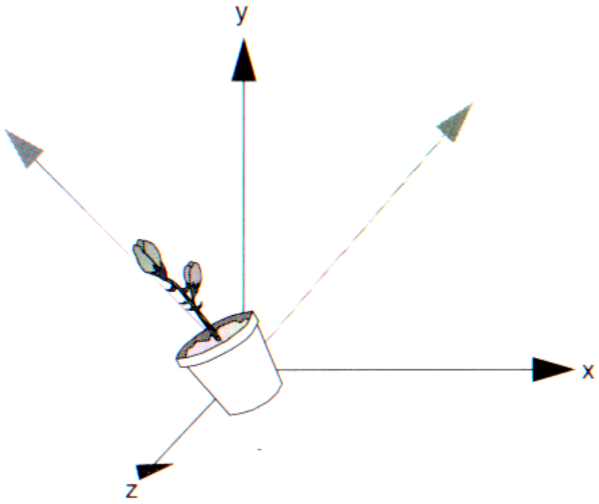
glTranslatef(0, 0, -1)



Modeling Transformations

- glRotate{fd}(angle, x, y, z)
  - Multiplies current matrix by a matrix that rotates an object in a counterclockwise direction about the ray from origin to (x,y,z) with angle as the degrees

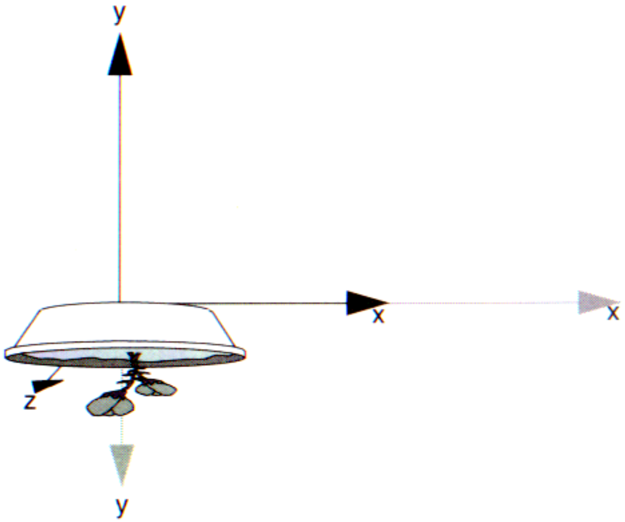
glRotatef(45.0, 0, 0, 1)



Modeling Transformations

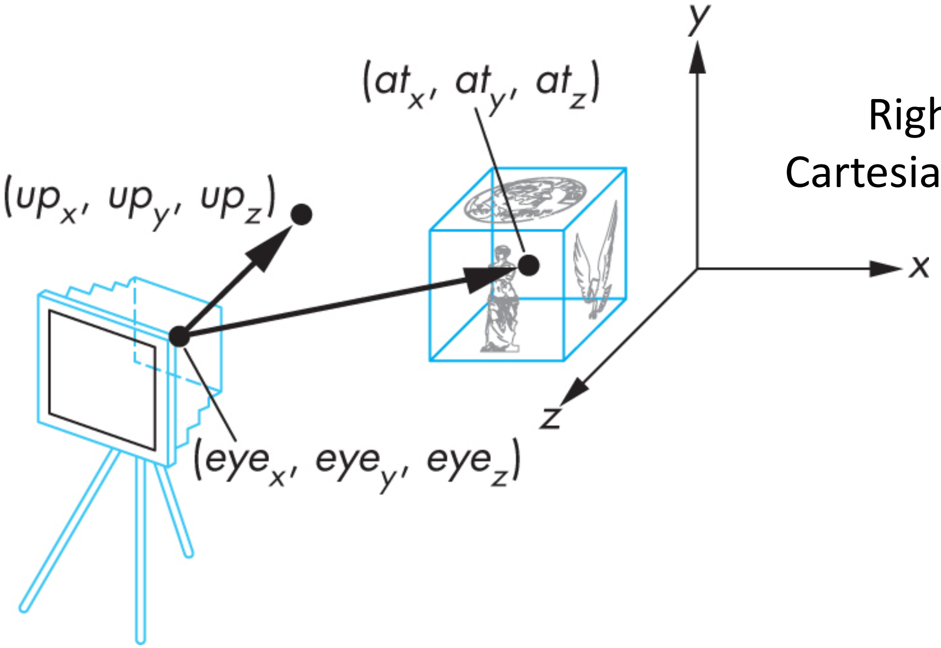
- glScale{fd} (x, y, z)
  - Multiplies current matrix by a matrix that scales an object along axes.

glScalef(2.0, -0.5, 1.0)



Viewing Transformations

- gluLookAt (eyex, eyey, eyez, atx, aty, atz, upx, upy, upz);
- By default the camera is at the origin, looking down negative z, and the up vector is the positive y axis



Right-handed
Cartesian Coordinates

Using OpenGL Matrices

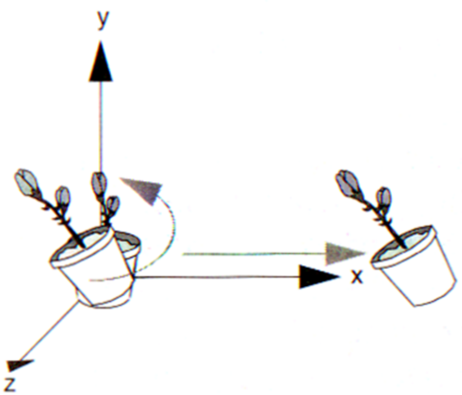
- **Use the following function to specify which matrix you are changing:**
  - glMatrixMode(whichMatrix): whichMatrix = GL_PROJECTION | GL_MODELVIEW
- **To guarantee a "fresh start", use glLoadIdentity():**
  - Loads the identity matrix into the active matrix
- **To load a user-defined matrix into the current matrix:**
  - glLoadMatrix{fd}(TYPE *m)
- **To multiply the current matrix by a user defined matrix:**
  - glMultMatrix{fd}(TYPE *m)
- SUGGESTION: To avoid row-/column-major confusion, specify matrices as m[16] instead of m[4][4]

Transforms in OpenGL

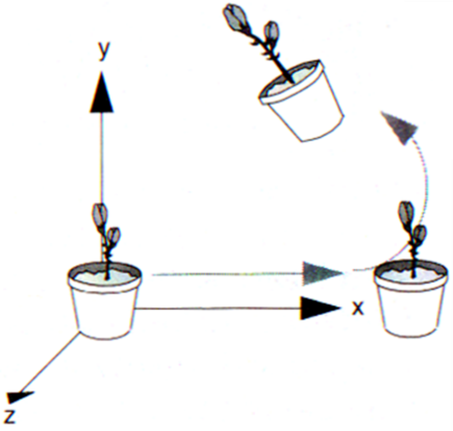
- OpenGL uses 4x4 matrices for all its transforms
  - But you don’t have to build them all by hand!
- glRotate{fd}(angle, x, y, z)
  - Rotates counter-clockwise by angle degrees about the vector (x, y, z)
- glTranslate{fd}(x, y, z)
- glScale{fd}(x, y, z)

Order of Transforms

- In OpenGL, **the last** transform in a list is applied **FIRST**
  - Think back to right-multiplication of transforms



Rotate then Translate



Translate then Rotate

```
glTranslatef( 1, 0, 0 );
glRotatef( 45.0, 0, 0, 1 );
drawObject();
```

```
glRotatef( 45.0, 0, 0, 1 );
glTranslatef( 1, 0, 0 );
drawObject();
```

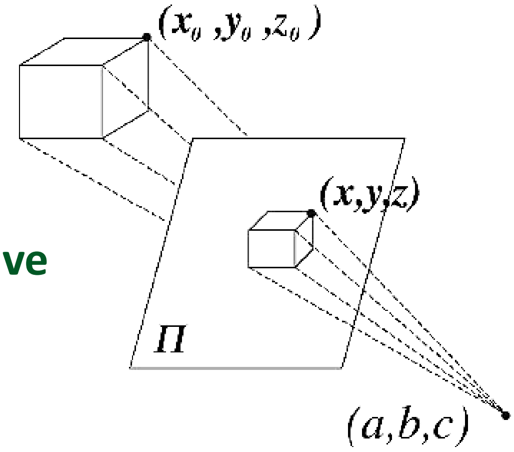
Projection Transforms

- The projection matrix defines the viewing volume
  - Used for 2 things:
    - Projects an object onto the screen
    - Determines how objects are clipped

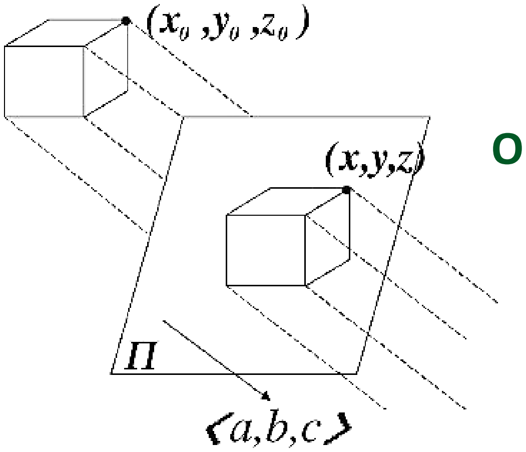
- The viewpoint (the location of the “camera”) that we’ve been talking about is at one end of the viewing volume

Projection Transform

- Perspective
  - Viewing volume is a truncated pyramid
    - aka frustum
- Orthographic
  - Viewing volume is a box



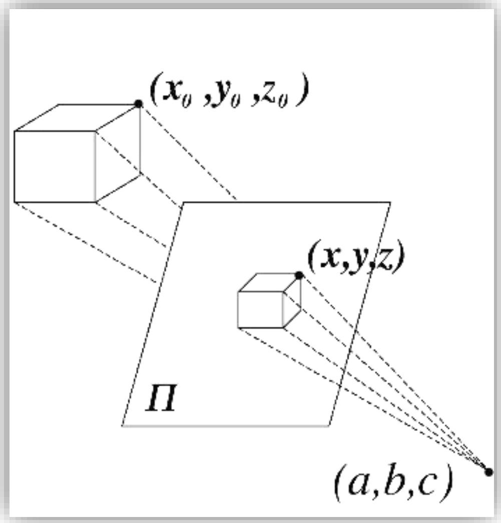
Perspective



Orthographic

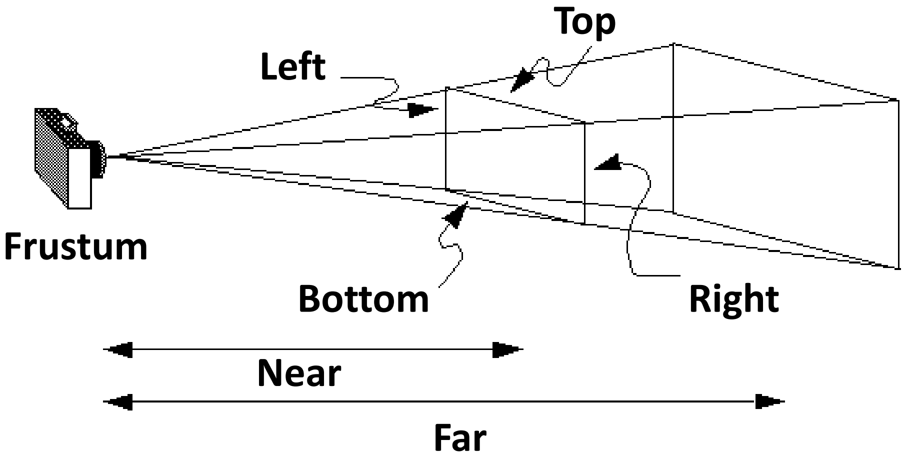
Perspective Projection

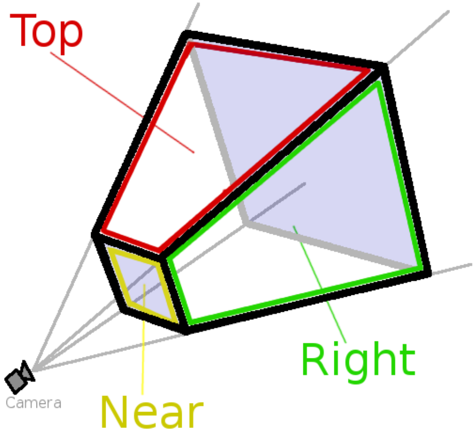
- The most noticeable effect of perspective projection is foreshortening
- OpenGL provides several functions to define a viewing frustum
  - glFrustum(...)
  - gluPerspective(...)



glFrustum (视锥体/视景体)

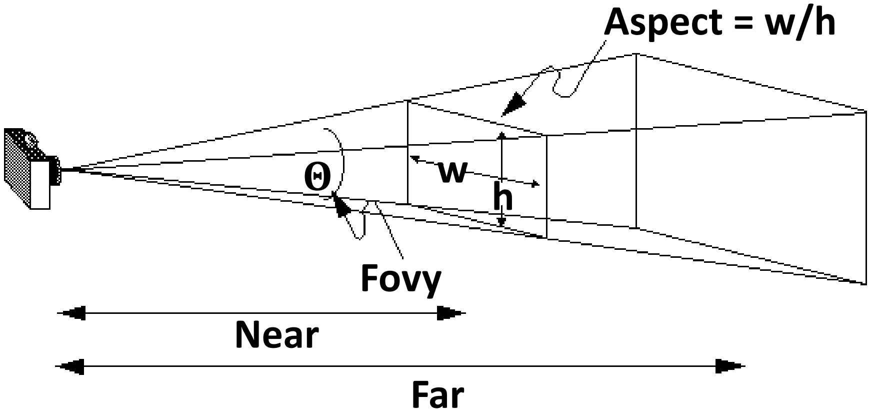
- glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)
  - (left, bottom, -near) and (right, top, -near) are the bottom-left and top-right corners of the near clip plane
  - far is the distance to the far clip plane
  - near and far should always be positive





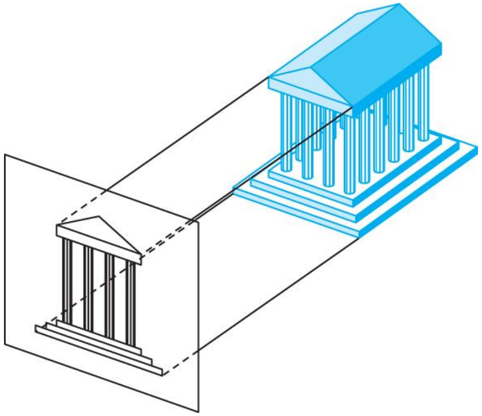
gluPerspective (透视图)

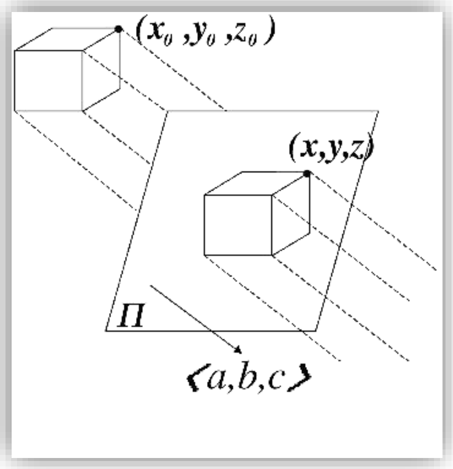
- This GL Utility Library function provides a more intuitive way (I think) to define a frustum
- gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble near, GLdouble far)
  - fovy - field of view in y (in degrees)
  - aspect - aspect ratio (width / height)
  - near and far - same as with glFrustum()



Orthographic Projection

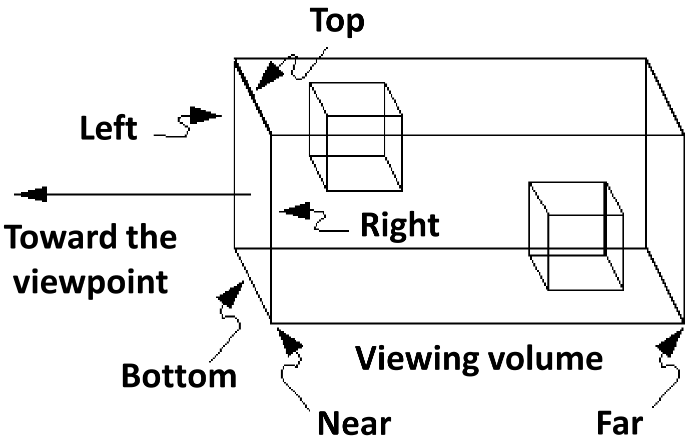
- With orthographic projection, there is no foreshortening (透视收缩)
  - Distance from the camera does not change apparent size
- Again, there are several functions that can define an orthographic projection
  - glOrtho()
  - gluOrtho2D()

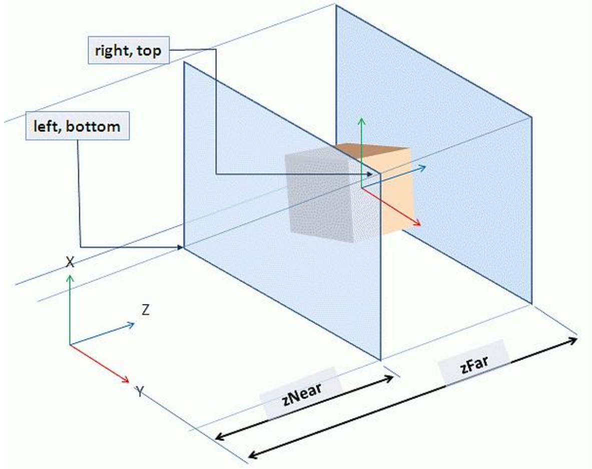




glOrtho

- glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)
  - Arguments are the same as glPerspective()
  - (left, bottom, -near) and (right, top, -near) are the bottom-left and top-right corners of the near clip plane
  - near and far can be any values, but they should not be the same

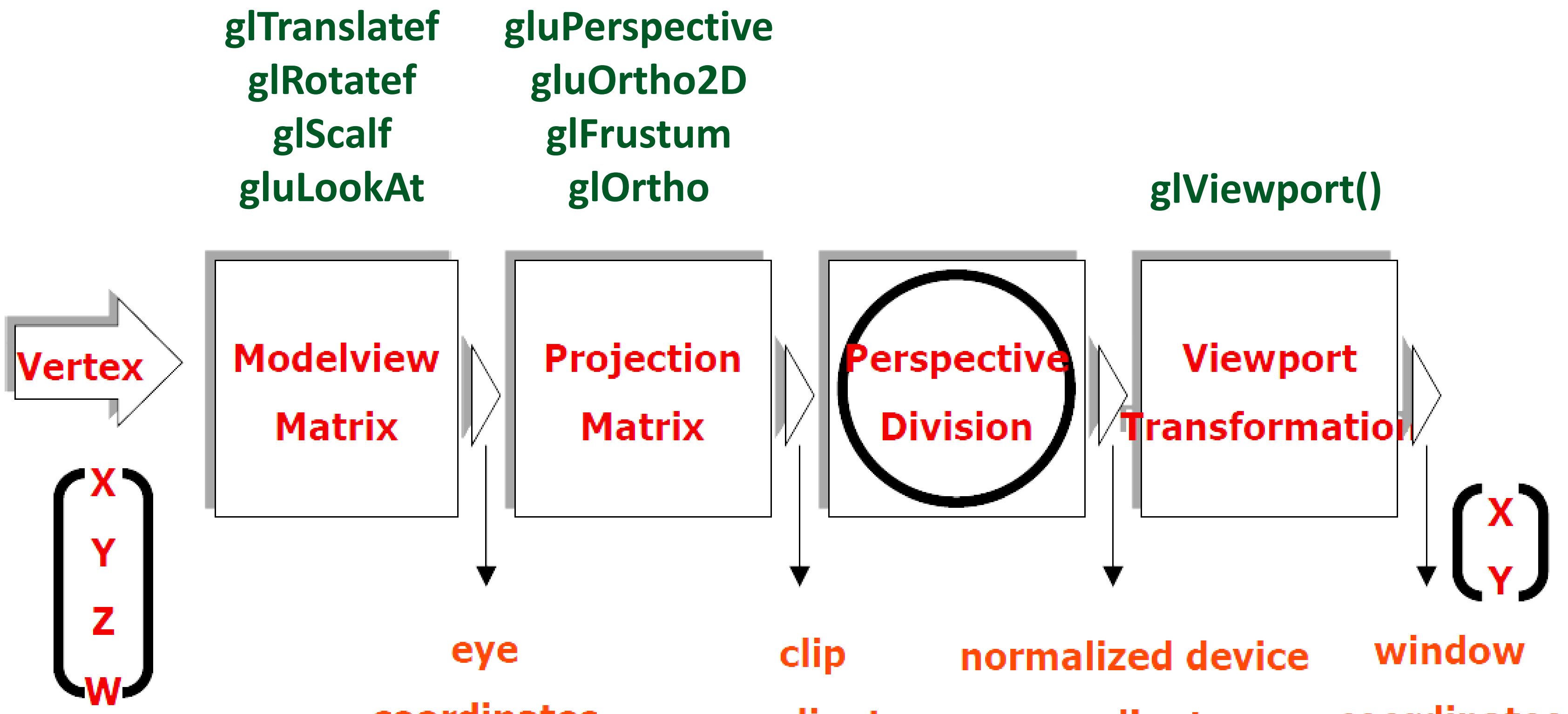




gluOrtho2D

- This GL Utility Library function provides a more intuitive way (I think) to define a frustum
- gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)
  - (left, bottom) and (right, top) define the (x, y) coordinates of the bottom-left and top-right corners of the clipping region
  - Automatically clips to between -1.0 and 1.0 in z
- In 2D mode, frustum is equal to viewport

OpenGL Transformations



glTranslatef
glRotatef
glScalf
gluLookAt

gluPerspective
gluOrtho2D
glFrustum
glOrtho

glViewport()

Vertex

X
Y
Z
W

Modelview Matrix

Projection Matrix

Perspective Division

Viewport Transformation

X
Y

eye coordinates

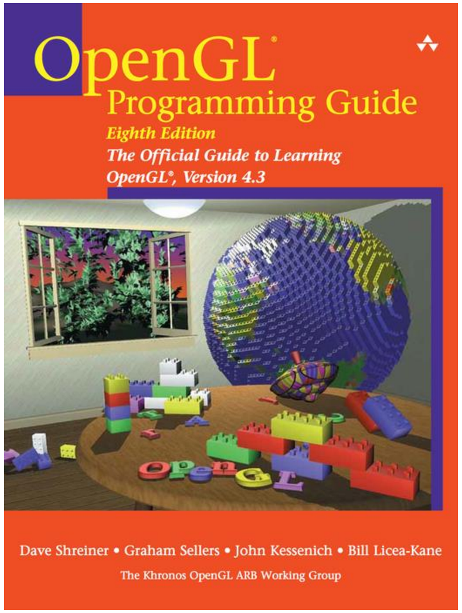
clip coordinates

normalized device coordinates

window coordinates

References

- OpenGL officially website:
  - http://www.opengl.org
- LearnOpenGL CN
  - https://learnopengl-cn.github.io/ (Chinese)
- The Red Book (OpenGL Programming Guide)


OpenGL
Programming Guide
Eighth Edition
The Official Guide to Learning OpenGL®, Version 4.3
Dave Shreiner • Graham Sellers • John Kessenich • Bill Licea-Kane
The Khronos OpenGL ARB Working Group

An PDF version is available online:

http://www.csc.villanova.edu/~mdamian/Textbooks/opengl_programming_guide_8th_edition.pdf

http://www.opengl-redbook.com/