



中山大學
SUN YAT-SEN UNIVERSITY



国家超级计算广州中心
NATIONAL SUPERCOMPUTER CENTER IN GUANGZHOU

计算机图形学

OpenGL 编程介绍

陶钧

taoj23@mail.sysu.edu.cn

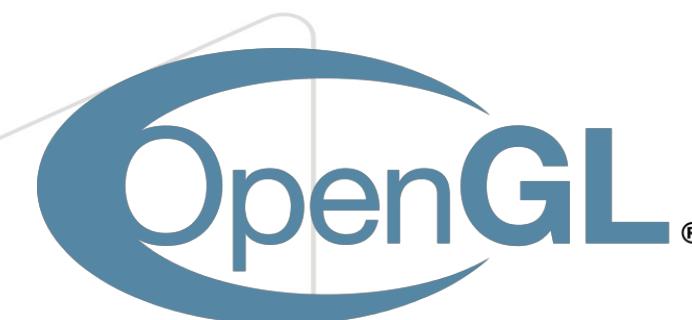
中山大学 数据科学与计算机学院
国家超级计算广州中心

- OpenGL是什么？
- OpenGL如何工作？
- OpenGL程序结构简介



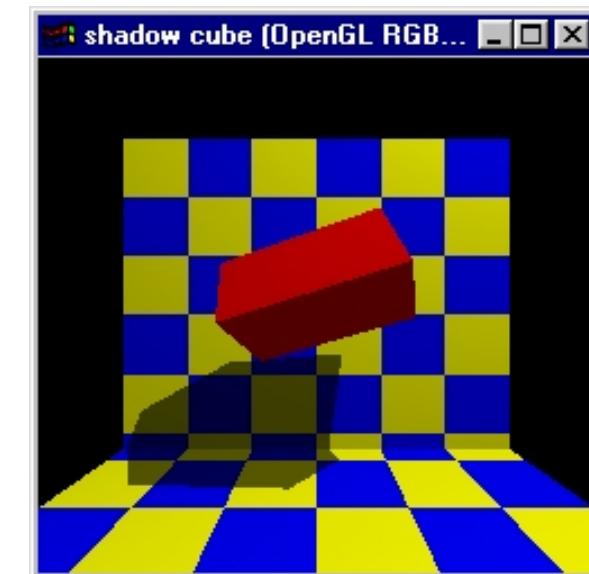
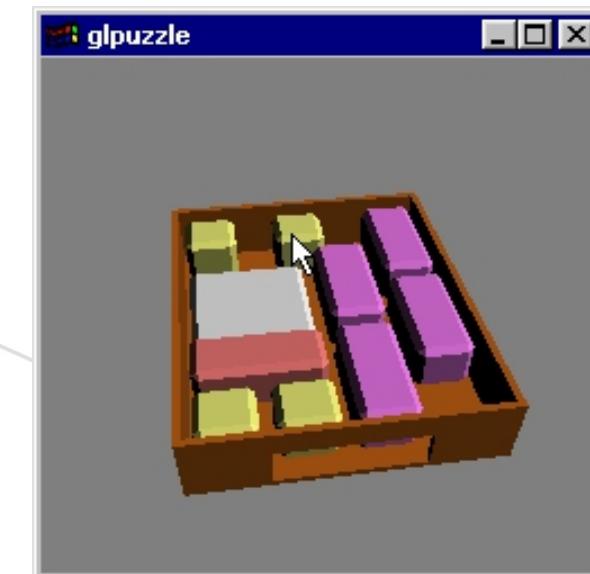
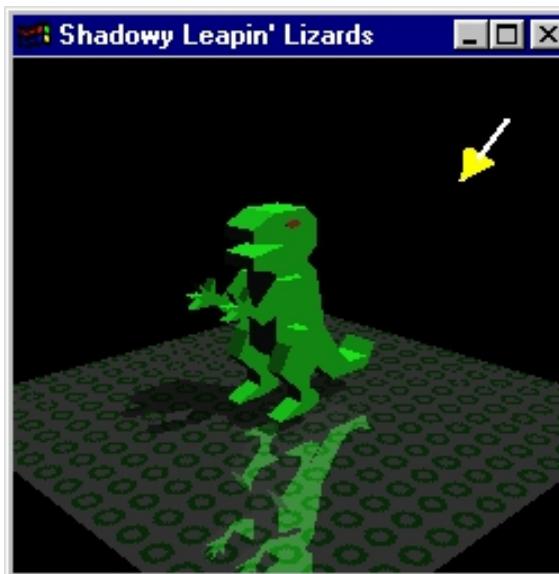
● Open Graphics Library (OpenGL)

- 用于渲染2D、3D矢量图形的跨语言、跨平台应用程序接口（API）
 - OpenGL的高效实现由显示装置产商提供，非常依赖于具体的硬件
 - 但API的内容不依赖于硬件或产商
- 1992年，由Silicon Graphics（SGI）领导成立由若干公司组成的OpenGL架构审查委员会（Architecture Review Board, ARB）
 - 源自当时图形工作站的领头羊SGI的IRIS GL API
 - OpenGL ARB负责OpenGL规范的维护和扩充
- 2006年，OpenGL ARB投票决定将控制权转移至Khronos Group
 - OpenGL API依然在不断进化中



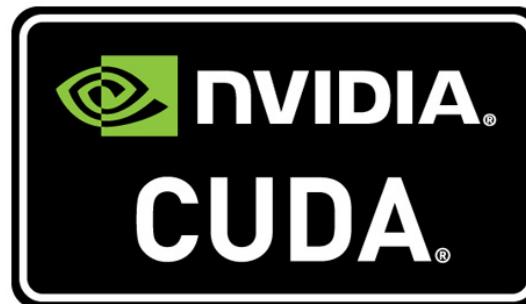
• Open Graphics Library (OpenGL)

- OpenGL是用于渲染的API
- 应用程序通过调用OpenGL API控制显卡，以简单的图形位元绘制2D/3D景象，最终产生屏幕空间中的图像
- API由近350个不同的函数调用组成
- OpenGL API只提供渲染绘制功能，没有音频、窗口等相关函数



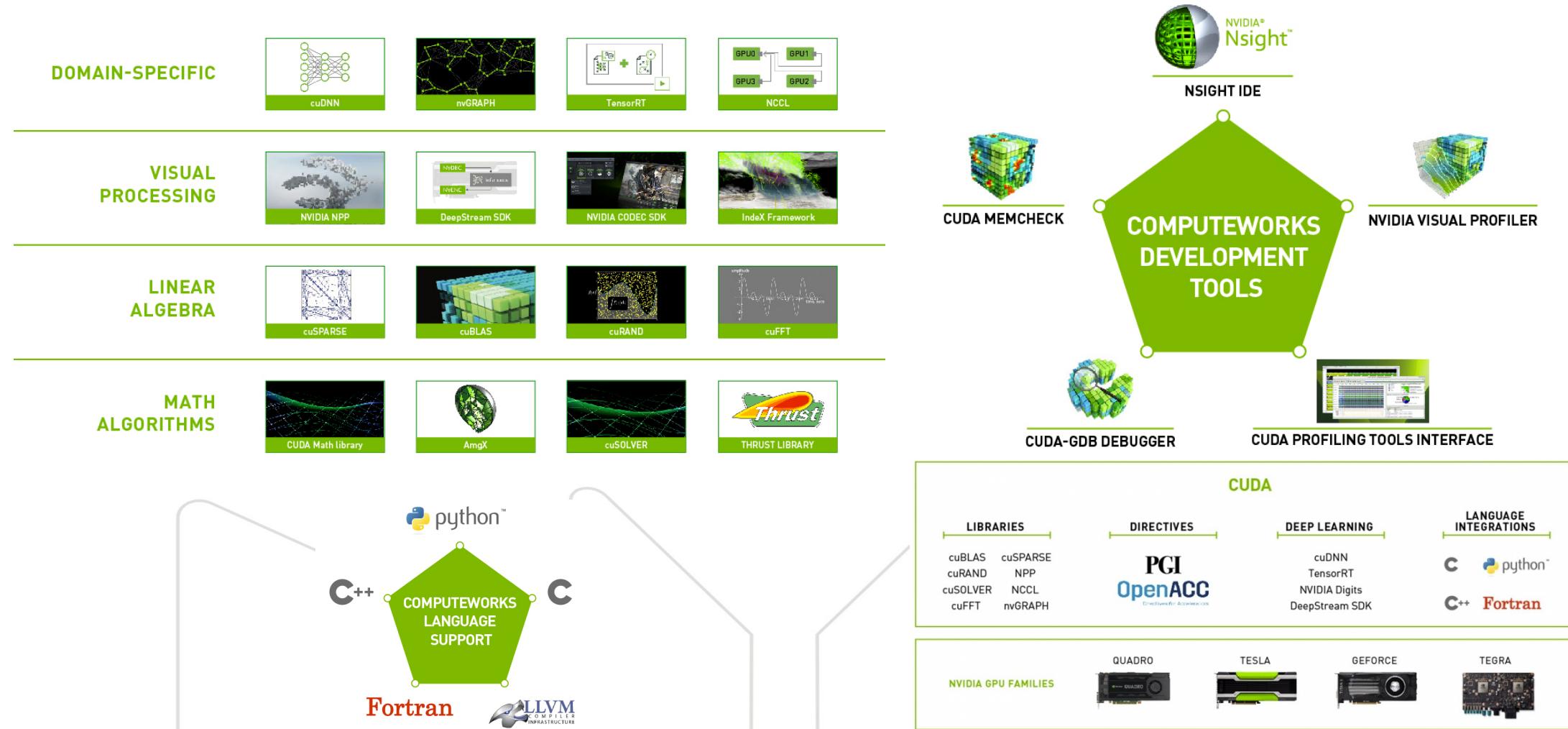
- OpenGL并非唯一的选择

- 其他选择还包括：NVIDIA CUDA, DirectX™, Windows Presentation Foundation™ (WPF) , RenderMan™, HTML5+WebGL™, JAVA 3D™



OpenGL并非唯一的选择

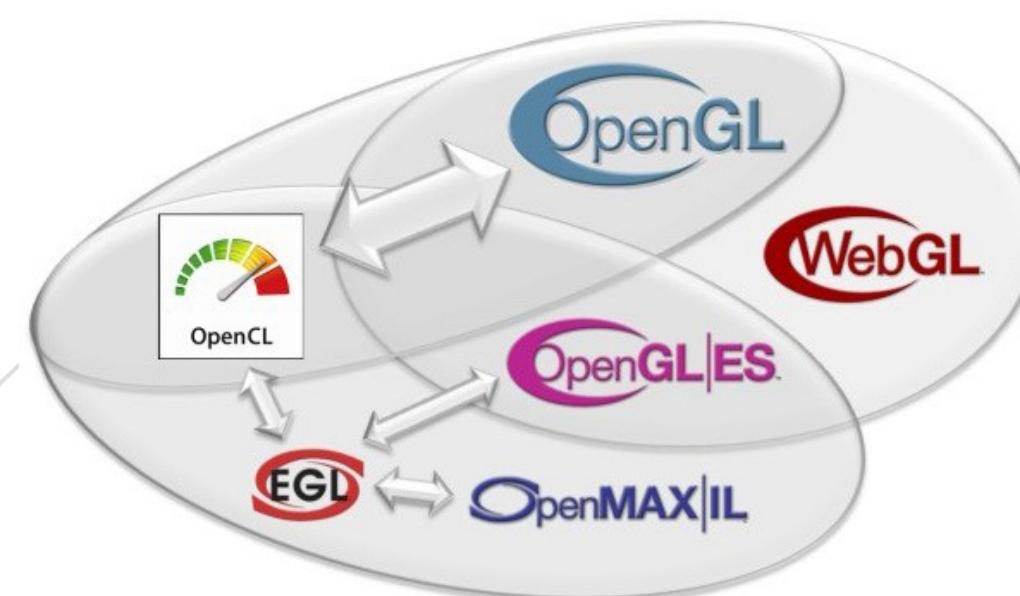
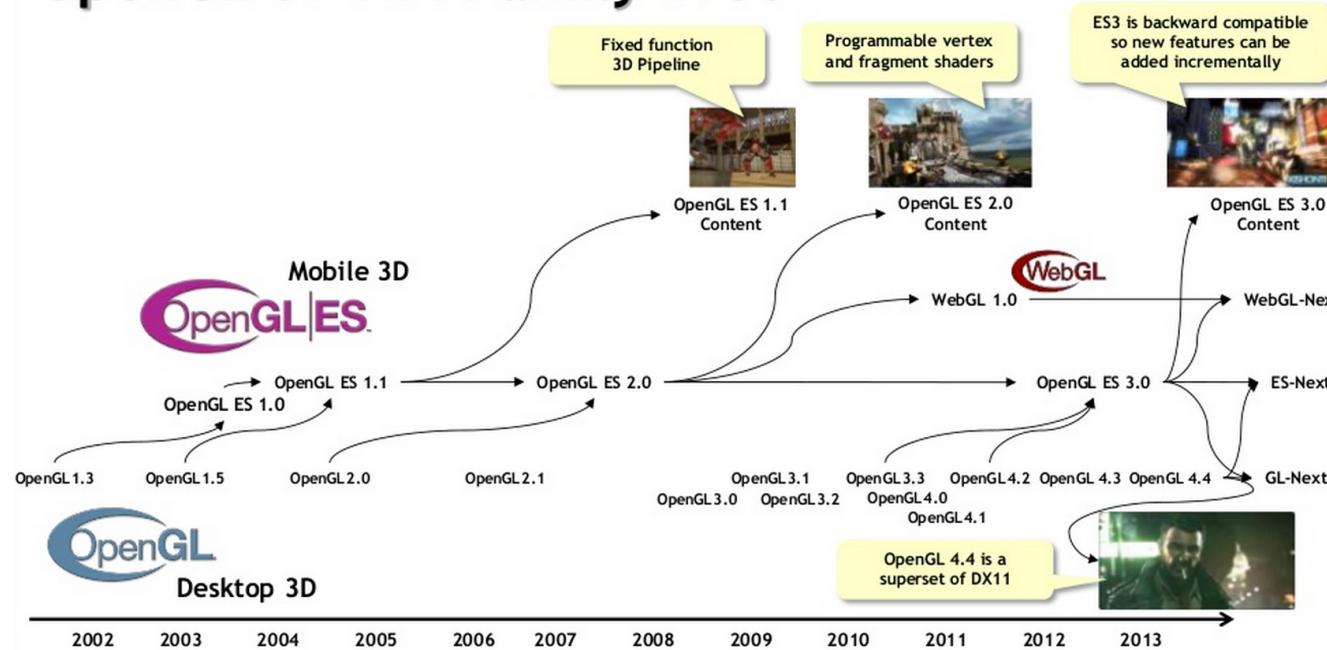
- NVIDIA CUDA Toolkit提供了使用显卡加速的高性能应用开发环境
 - 严格地说并不是一个图形库



● OpenGL发展

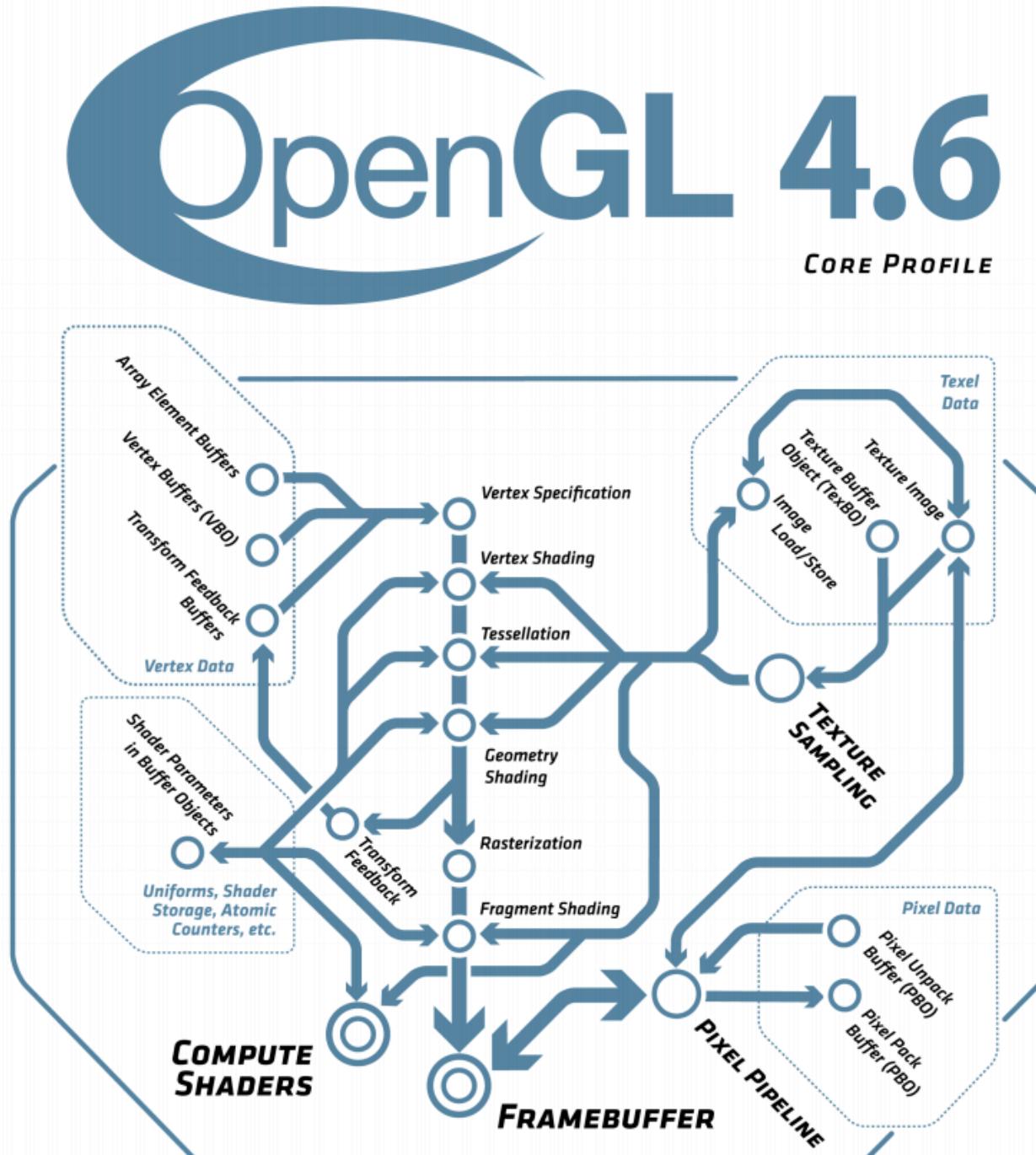
- 当前由Khronos Group发布OpenGL新版本，持续扩展原有API提供新特性
- 当前最新版本为2017年7月发布的OpenGL 4.6
- Vulkan (“Next Generation OpenGL Initiative (glNext) ”)
 - 整合OpenGL与OpenGL ES，不再兼容现有OpenGL版本
 - 初代Vulkan API与2016年2月发布

OpenGL 3D API Family Tree

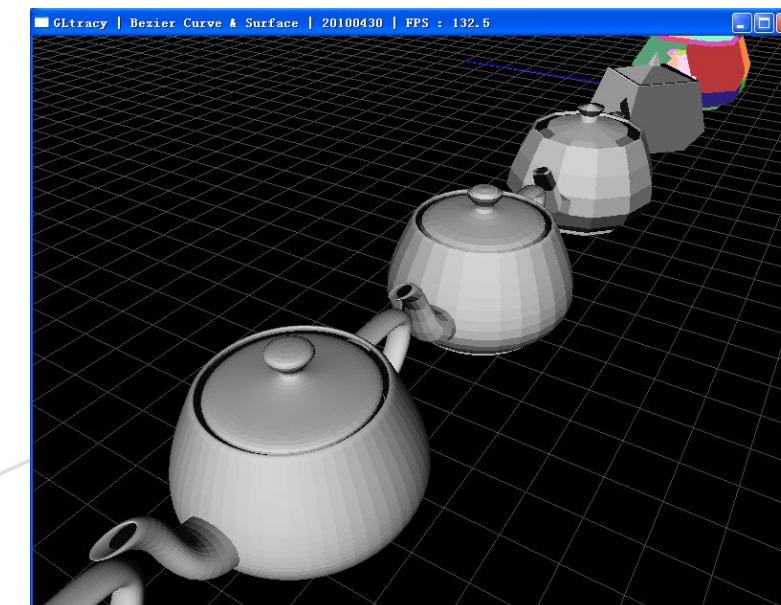


- OpenGL能做什么？

- 定义物体形状、材料属性、光照
- 排列物体，根据三维空间中的摄像机进行合成
- 将物体从数学表达形式转换为 fragment
 - 光栅化，rasterization
- 计算每个fragment的颜色，混合得到最终图像上每个像素的颜色



- OpenGL不包含复杂物体的渲染函数
 - 使用基本几何形状（点、线、及多边形）构建复杂物体
- OpenGL Utility Toolkit (GLUT)
 - OpenGL实用函数工具包，非官方OpenGL的组成部分
 - 提供系统级别的输入输出
 - 窗口定义、窗口控制、监控键盘与鼠标输入、等
 - 提供一系列基本几何体绘制函数
 - 正方体、球体、Utah teapot
 - 实心(solid) 及线框(wireframe) 模式
 - 提供一定的菜单功能



● 安装（以windows为例）

- OpenGL官网下载

- https://www.opengl.org/resources/libraries/glut/glut_downloads.php
- “We direct you to use **FreeGLUT** found on SourceForge:
<http://freeglut.sourceforge.net/>. The original GLUT has been unsupported for 20 years.”

- 安装便捷：



- 将glut.dll拷贝至{Windows DLL dir}/glut32.dll
- 将glut.lib拷贝至{VC++ lib path}/glut32.lib
- 将glut.h拷贝至{VC++ include path}/GL/glut.h
- 使用时只需 `#include <GL/glut>` 即可

- 实际使用中建议Qt

- 在课程网页中将提供Qt程序模板

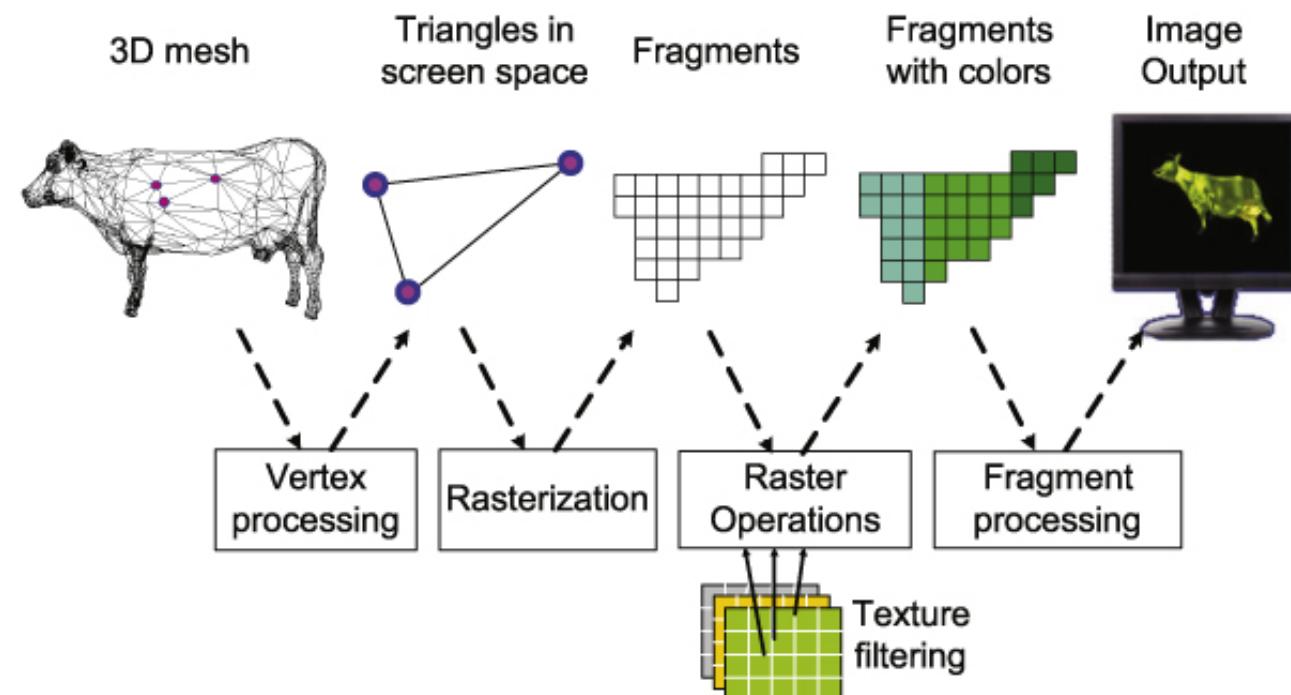
- OpenGL是什么？
- OpenGL如何工作？
- OpenGL程序结构简介



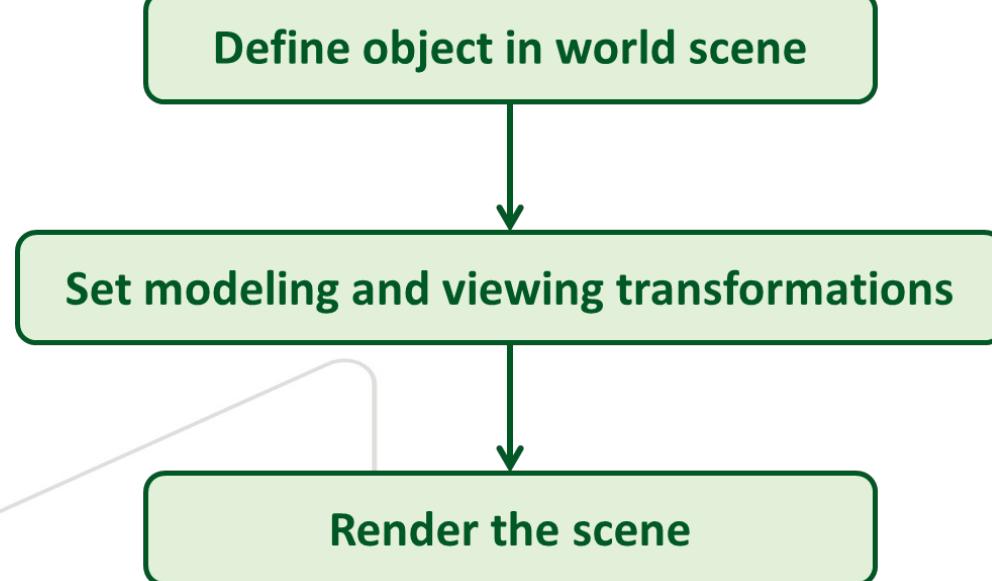
- OpenGL是一个状态机

- 通过函数调用设置其内部变量的状态，或查询当前状态
- 在设置新的状态前，当前状态不会发生改变
- 每个系统状态变量都有缺省值

- OpenGL简易pipeline



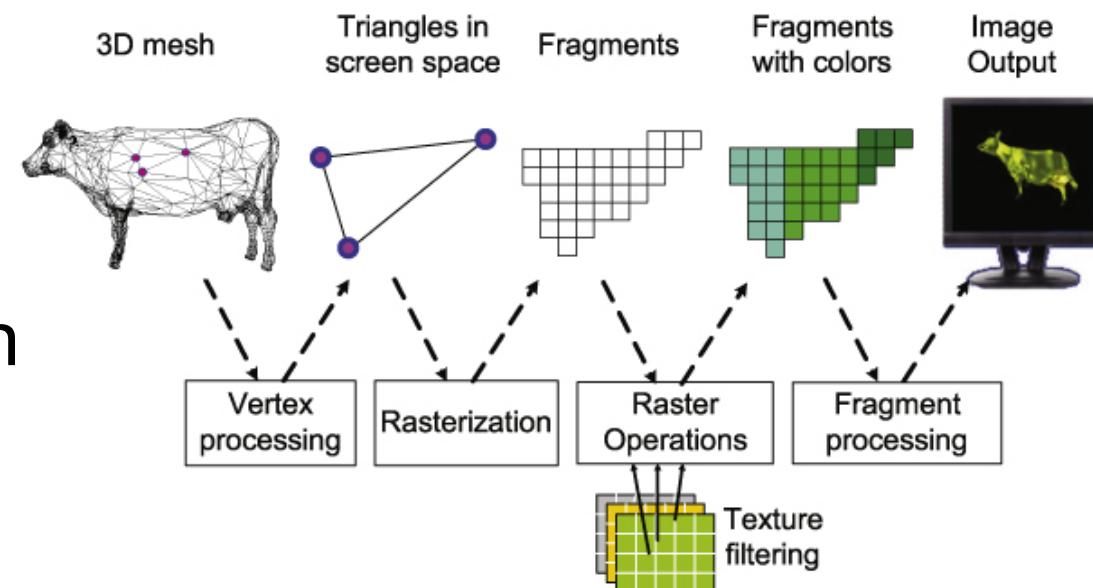
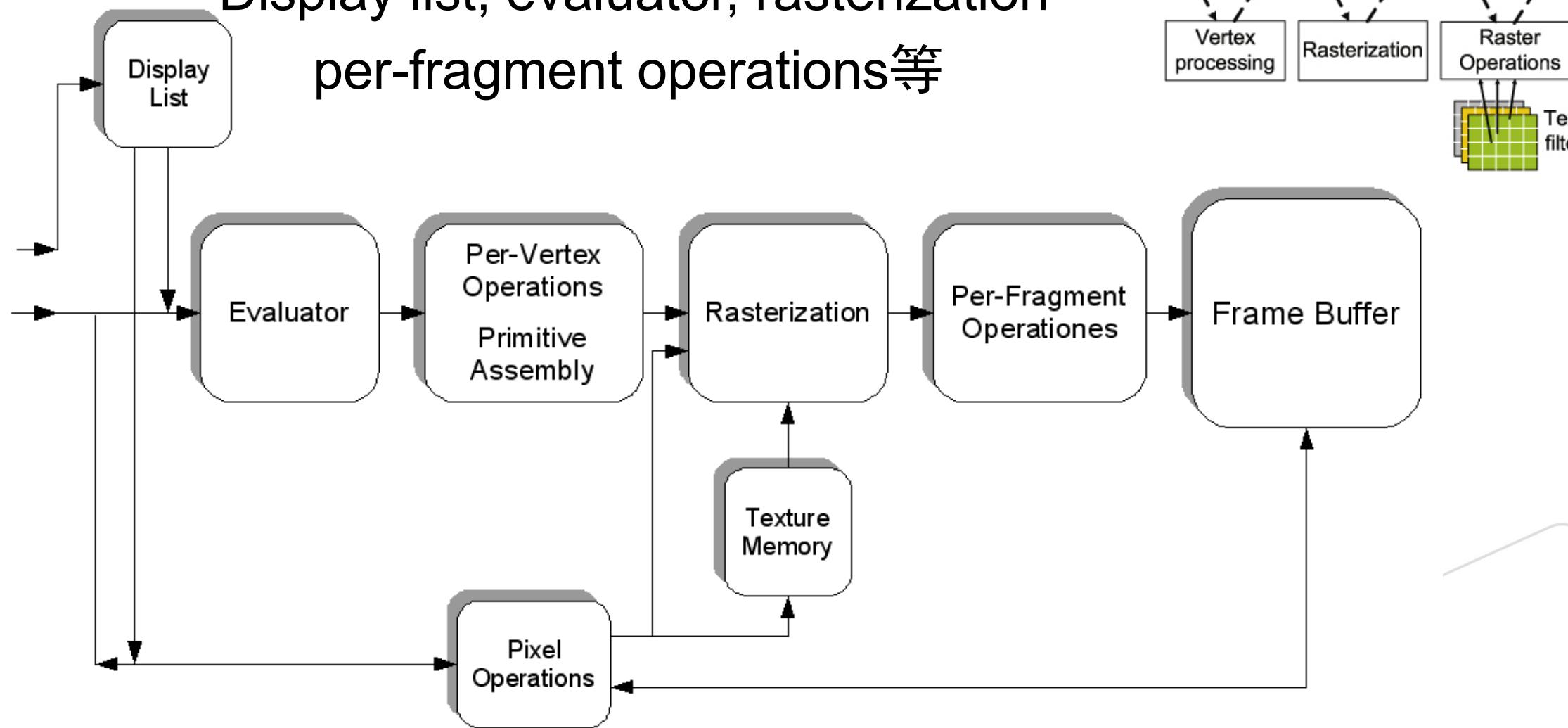
OpenGL工作流程



• OpenGL 简易 pipeline

– 通过以下部分实现

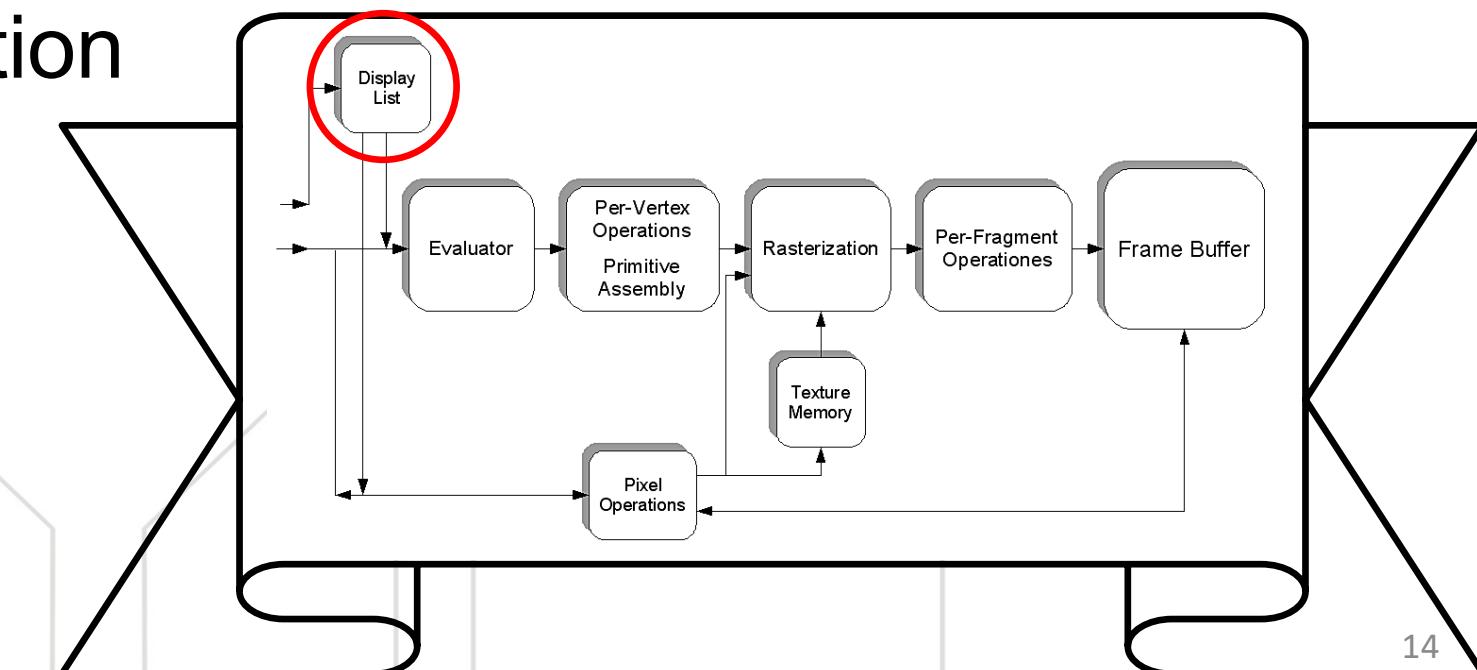
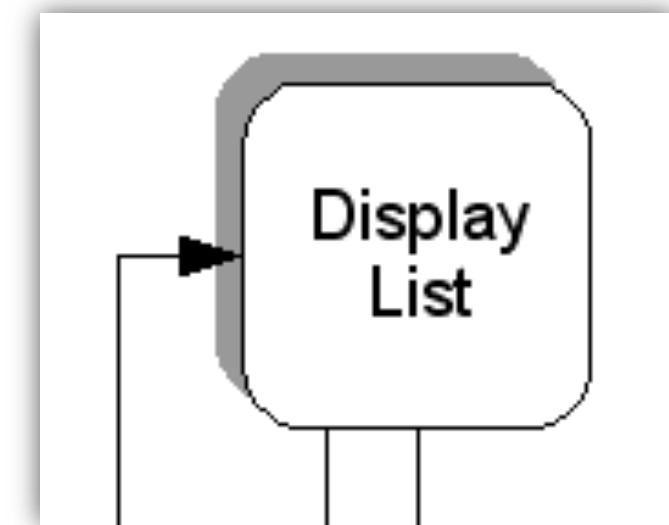
- Display list, evaluator, rasterization
per-fragment operations等



- OpenGL 简易 pipeline

- Display list

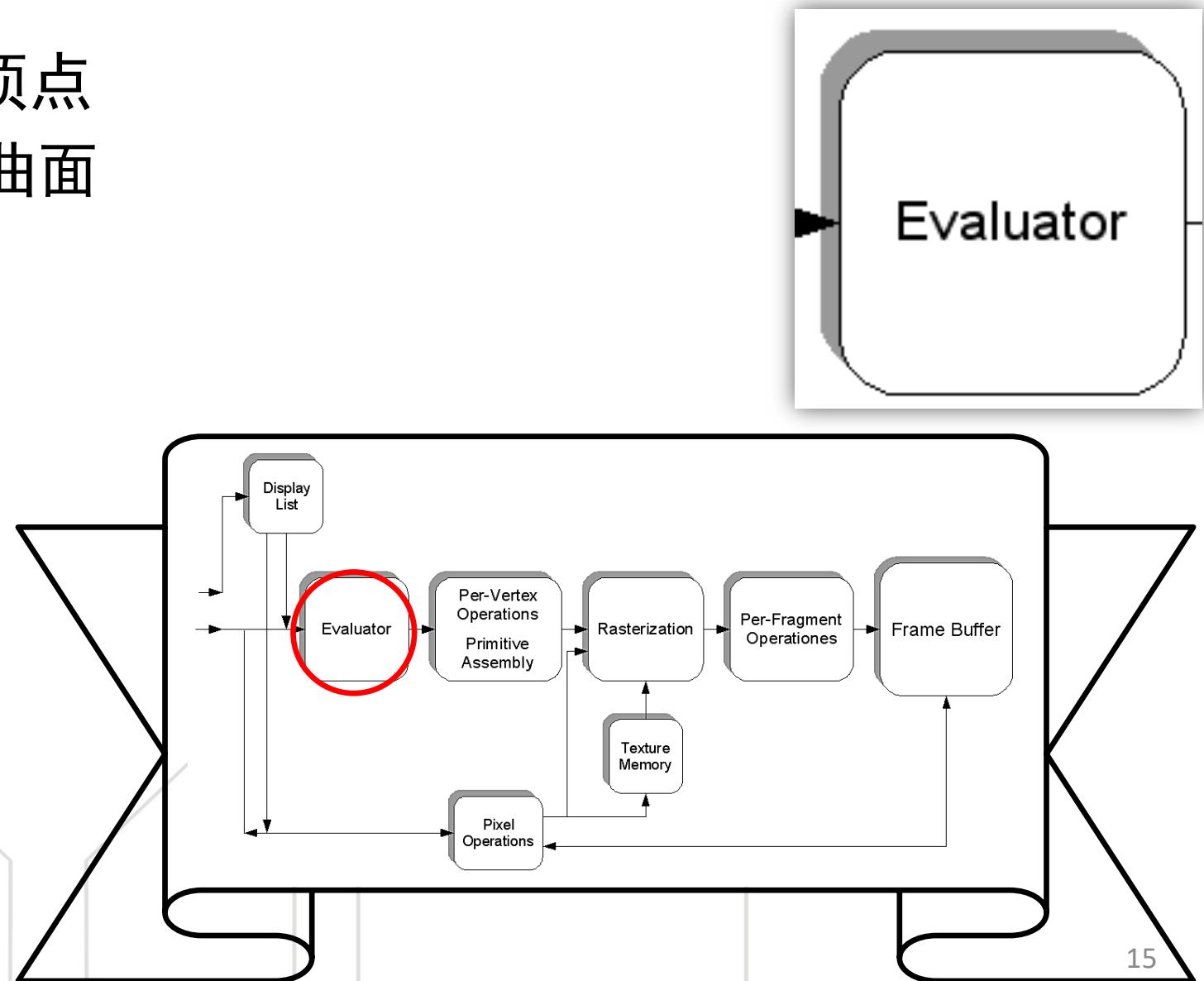
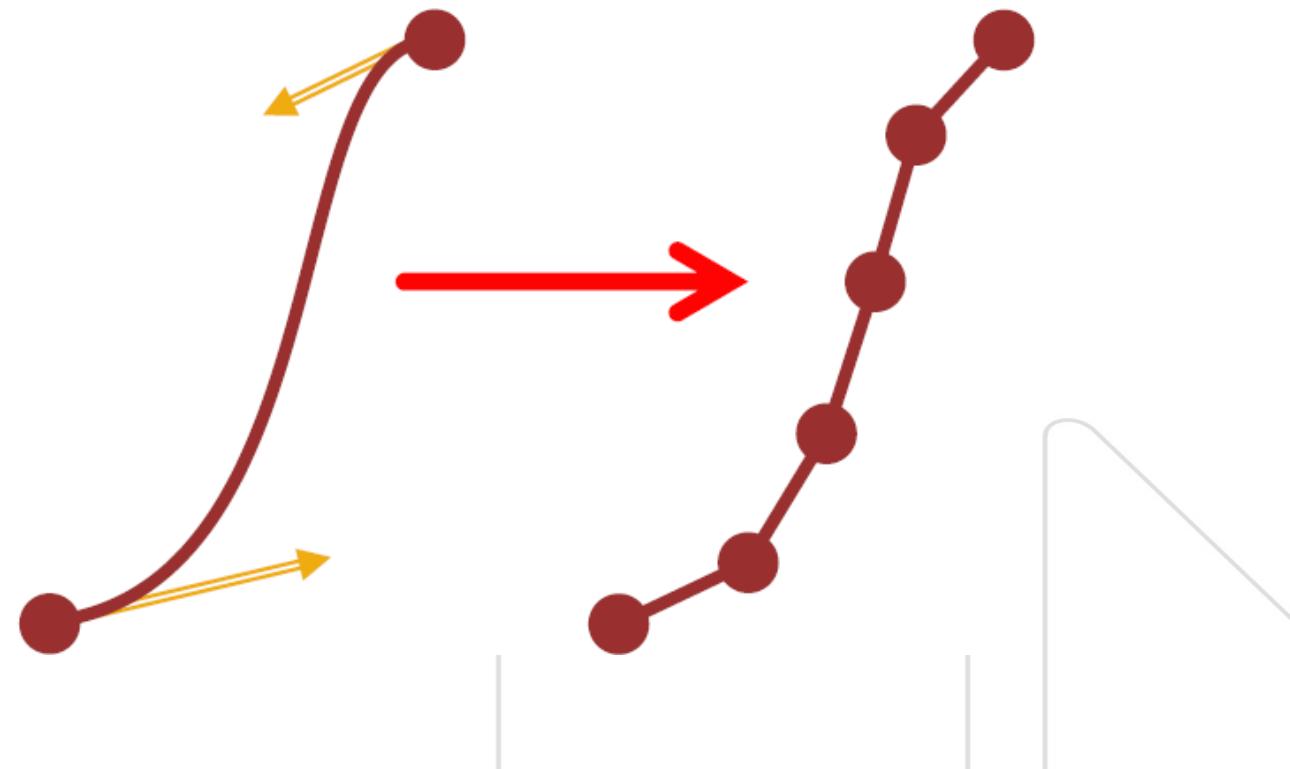
- 存储在GPU上的OpenGL command列表
 - 称为“子程序（subroutines）”
 - 可包含任意OpenGL command
 - pre-compiled
 - pre-computed transformation
 - 快！



- OpenGL 简易 pipeline

- Evaluator

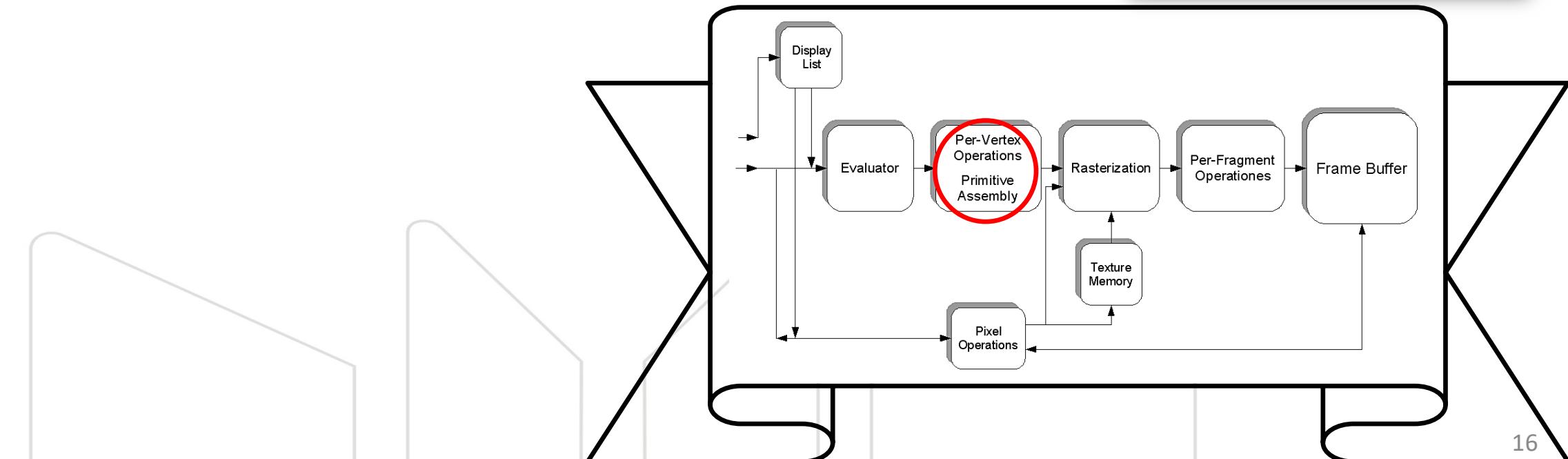
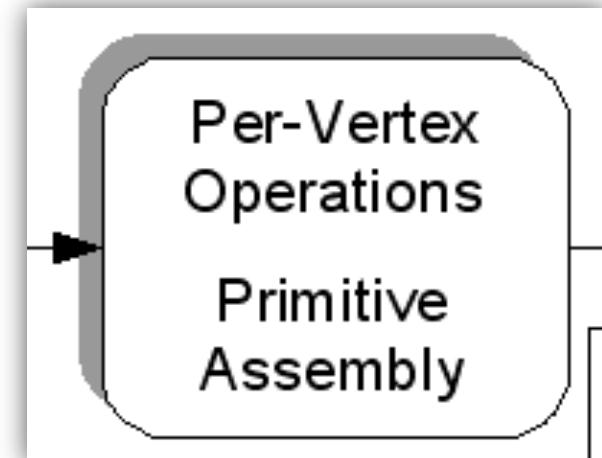
- 计算机的几何原语是离散化的顶点
 - Evaluator 使用控制点描述曲线/曲面



• OpenGL简易pipeline

– Per-Vertex Operations与primitive assembly

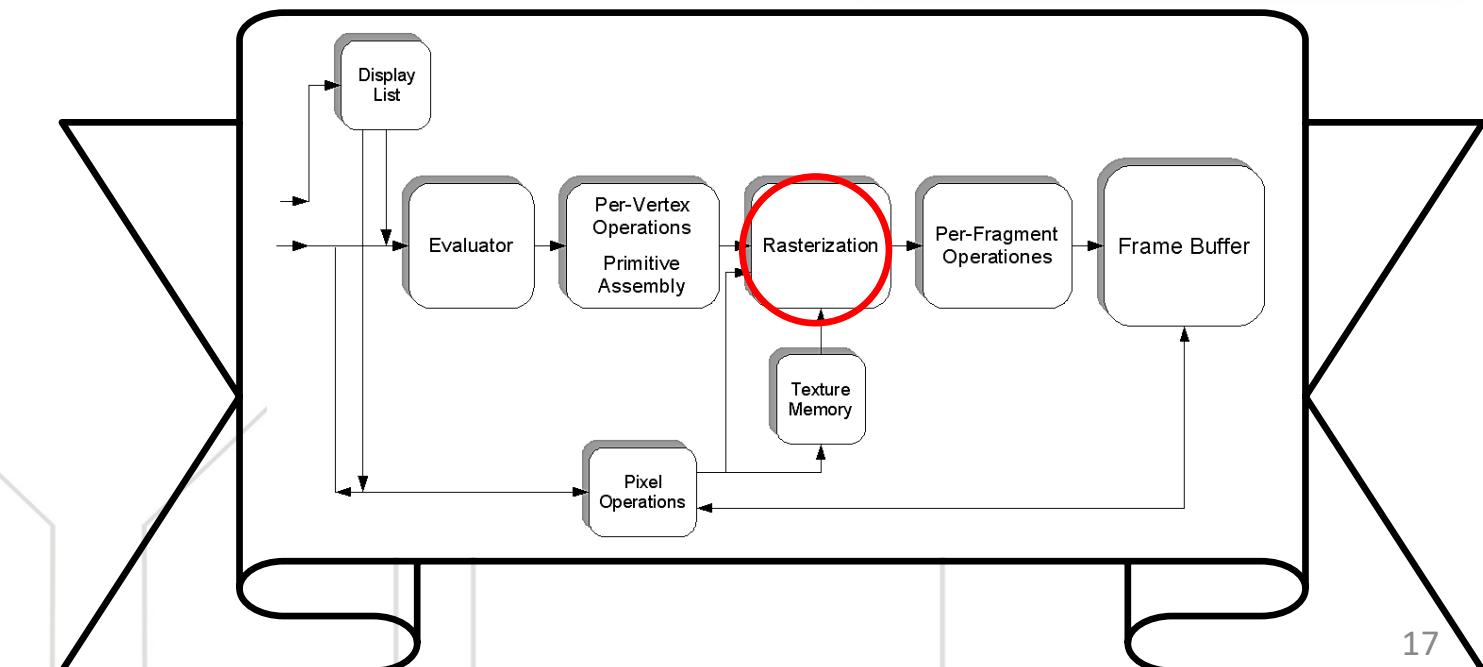
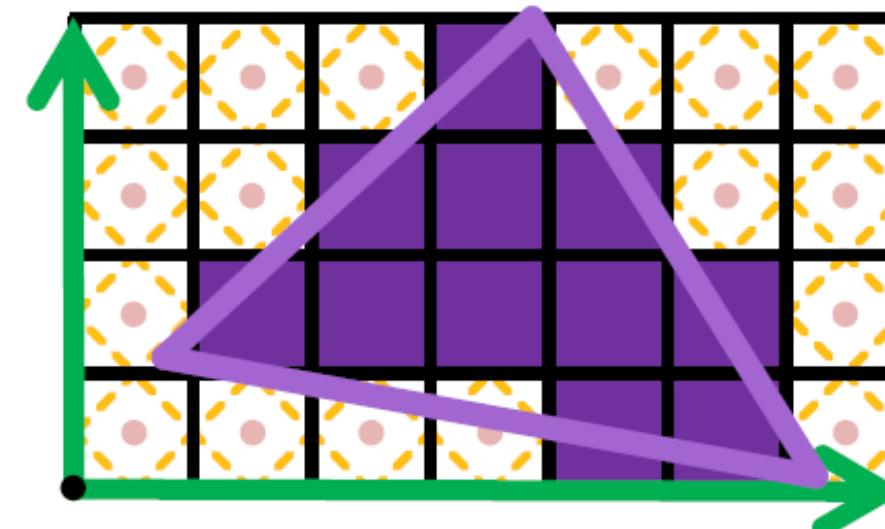
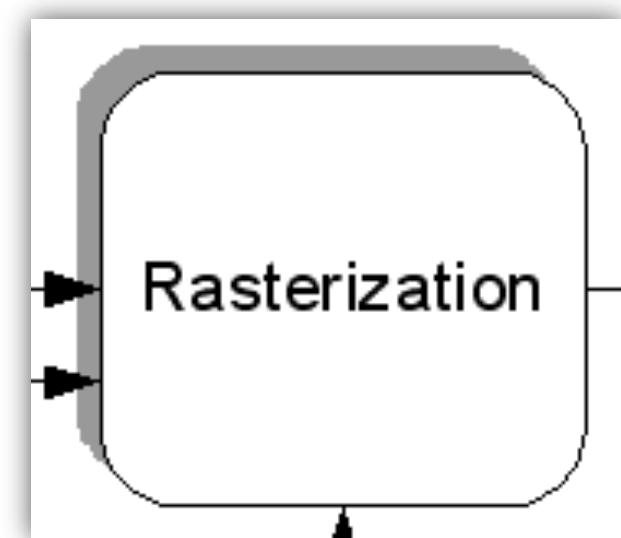
- 基于顶点进行操作
 - 空间变换等
- 将顶点组装成图元



- OpenGL 简易 pipeline

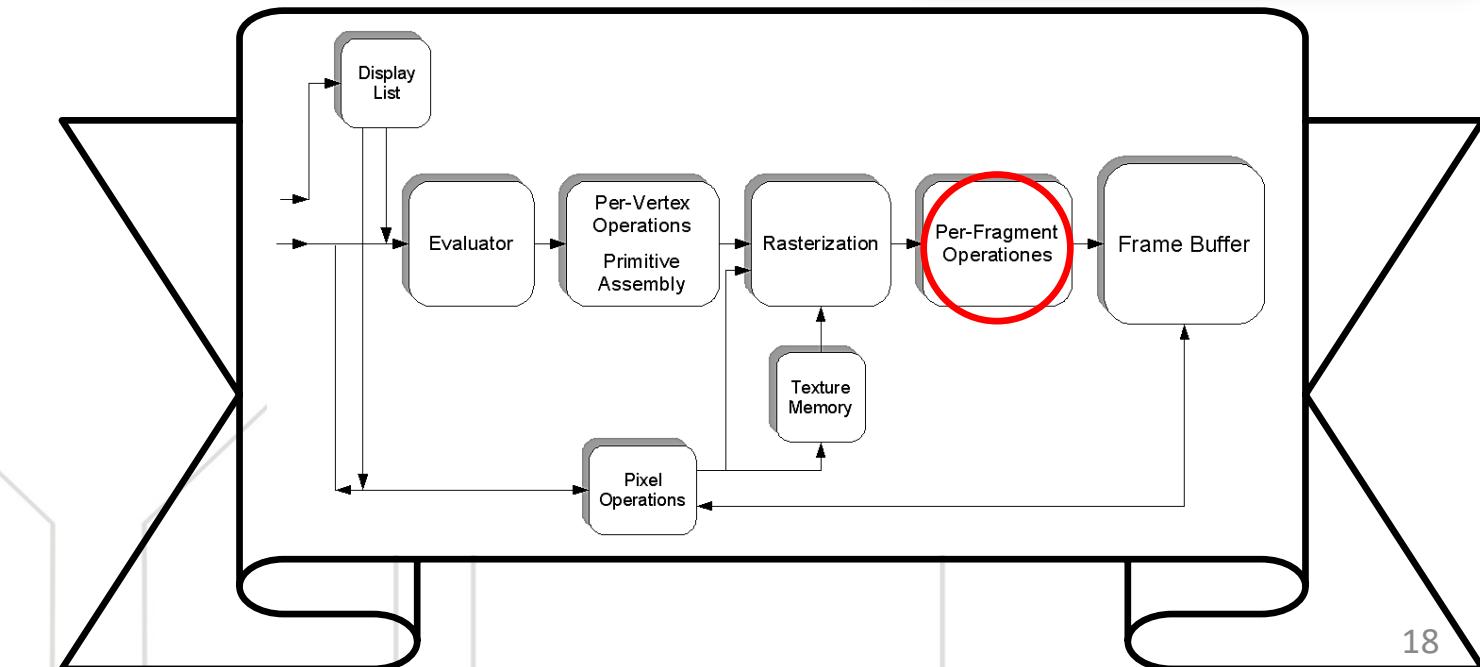
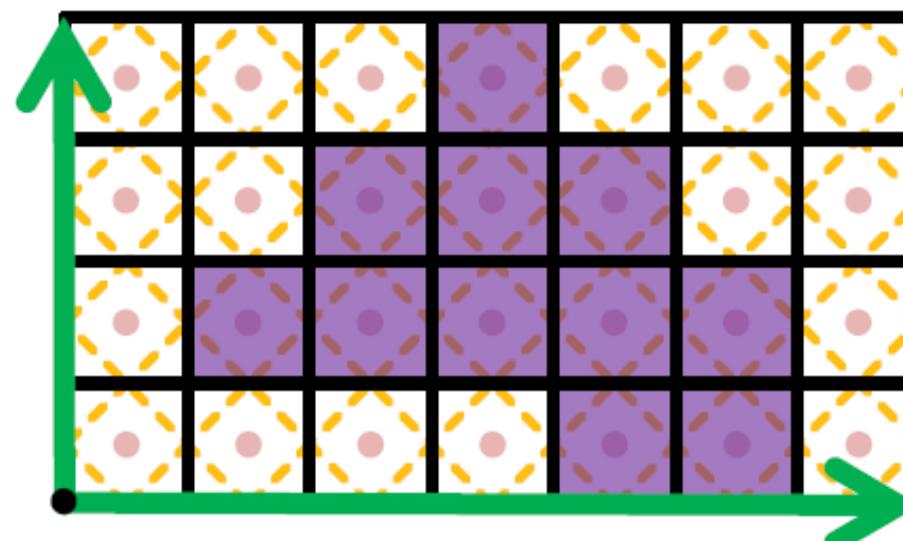
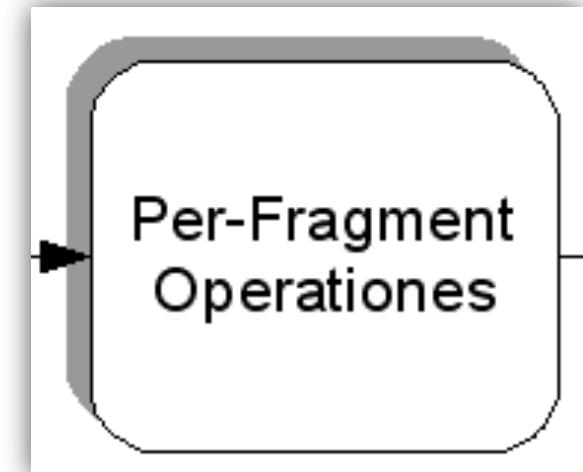
- Rasterization (光栅化)

- 将图元转换为片段 (fragment)
 - 转换过程中，直线的宽度，样式，点的大小支持抗锯齿等带来的覆盖计算都被考虑在内
 - 每个方形片段对应frame buffer中的一个像素



- OpenGL简易pipeline
 - Per-fragment operations

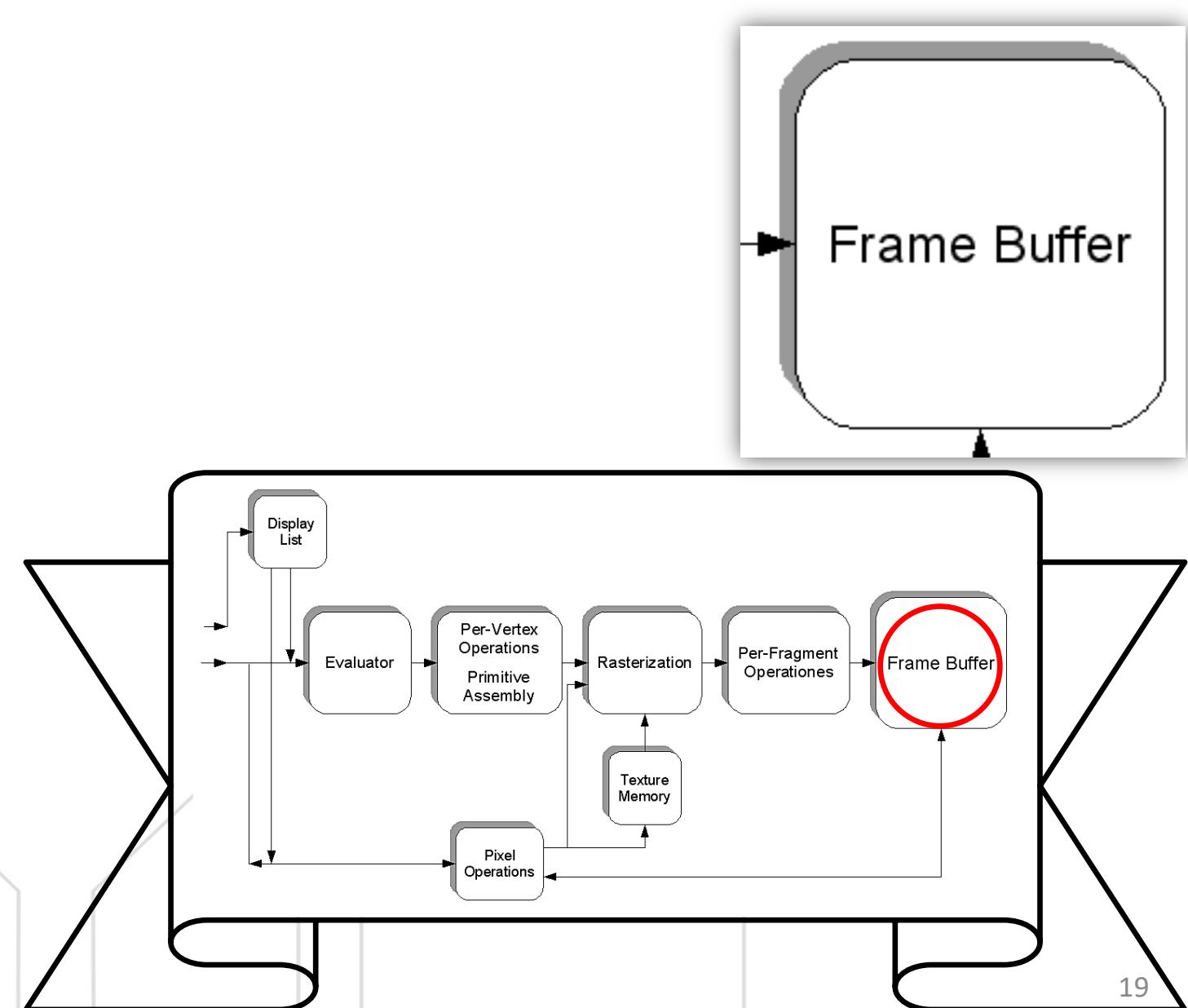
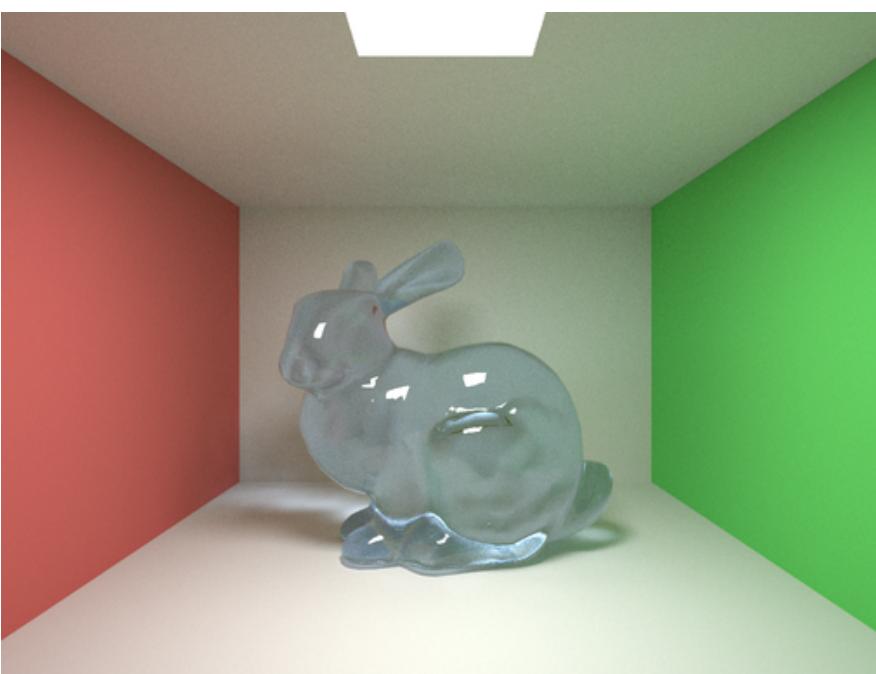
- 将frame buffer中同一像素对应的所有fragment混合，得到像素颜色
- 混合过程中考虑fragment的颜色、深度等信息



- OpenGL 简易 pipeline

- Frame buffer

- 保存将要显示的帧的像素内容
 - 传至屏幕进行显示



● OpenGL函数

- 图元 (primitives) : 定义画什么: 点、边、多边形
- 属性 (attribute) : 怎么画
- 变换 (transformation) : 怎么看与怎么组合
- 输入与控制: 由工具库提供
 - GLUT, FLTK, Qt
- 查询



● OpenGL相关API

– OpenGL核心库

- OpenGL的一部分
- 以`gl`开头，如`glBegin()`

`glCallList`
`glCallLists`
`glColor`
`glEdgeFlag`
`glEnd`
`glEvalCoord`

OpenGL核心库，包含于Microsoft SDK，
用于常规的、核心的图形处理

– GLU (OpenGL Utility Library)

- OpenGL的一部分
- 以`glu`开头，如`gluLookAt()`

`gluDisk`
`gluNewQuadric`
`gluPartialDisk`
`gluQuadricOrientation`
`gluQuadricTexture`
`gluSphere`

OpenGL实用库，包括纹理映射、坐标变
换、多边形分化、绘制简单多边形实体

– GLUT (OpenGL Utility Toolkit)

- 非OpenGL的一部分
- 提供窗口管理、鼠标、键盘、菜单功能
- 事件驱动 (event-driven)
- 缺乏现代GUI支持 (如，滑动条)
- 以`glu`开头，如`gluLookAt()`

OpenGL工具库，以`glut`开头，是不依赖于窗口平台
的OpenGL工具包，提供窗口相关的复杂的绘制功能

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

● OpenGL函数命名方式

数据类型

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

函数功能

Vertex为指
明顶点位置

glVertex3f(x, y, z)

表明所属的库

gl: OpenGL核心库

glu: GLU

glut: GLUT

参数个数

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

glVertex3fv(p)

v - 表明参数使用
指针表示向量

- OpenGL是什么？
- OpenGL如何工作？
- OpenGL程序结构简介



● 需要文件

- 头文件

- `#include <GL/gl.h>`, `#include <GL/glu.h>`, `#include <GL/glut.h>`

- 库文件

- 静态库: `opengl32.lib` `glu32.lib` `glut32.lib`

– 放置于编译环境lib目录下

- 动态库: `opengl32.dll` `glu32.dll` `glut32.dll`

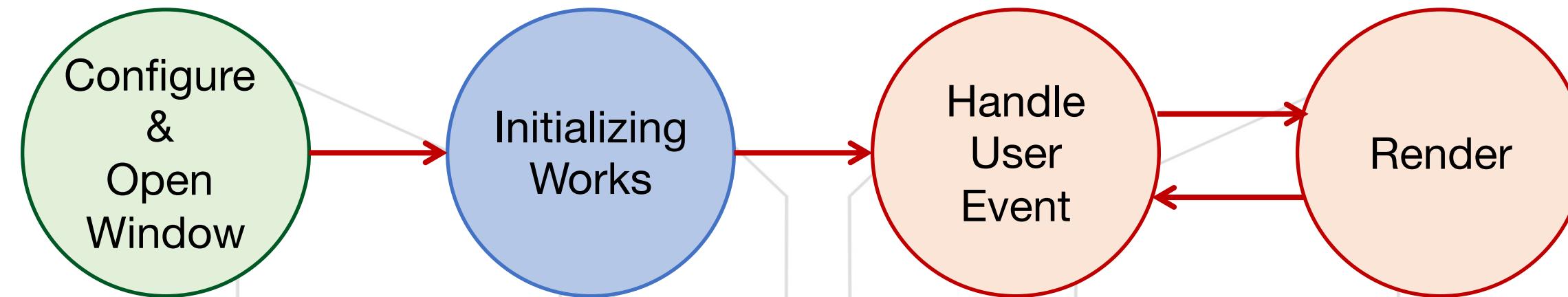
– 放置于系统目录 (`c:/Windows/System32`; `c:/Windows/SysWOW64`)

– 或 `dll` 目录 (`/usr/bin/`) 下



● 基本程序结构

- 非面向对象
- 使用状态进行控制
- 无限循环



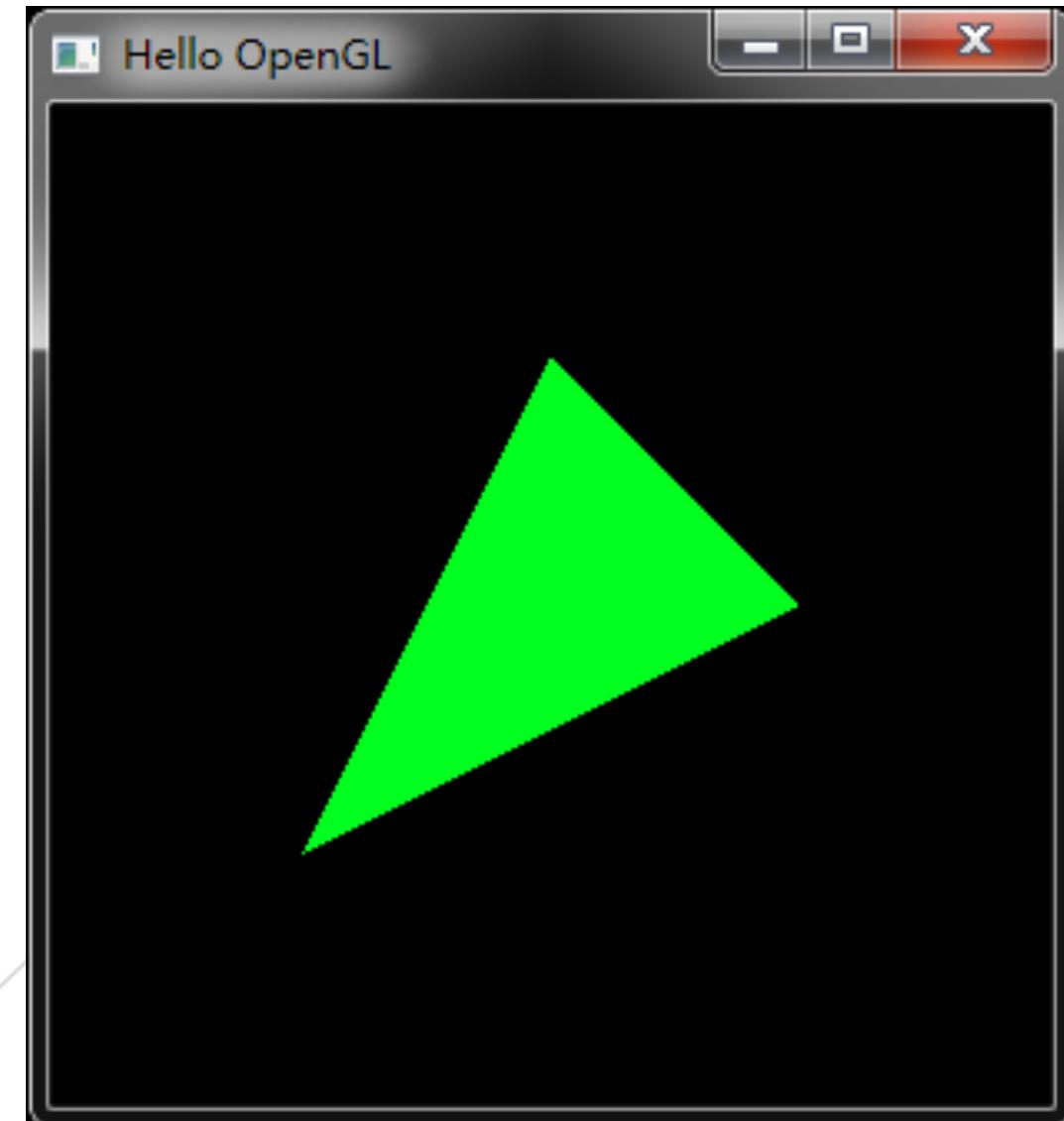
• OpenGL Hello World

– 不到20行代码！

```
#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;
}
```



● OpenGL Hello World

```
#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

● OpenGL Hello World

```
#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

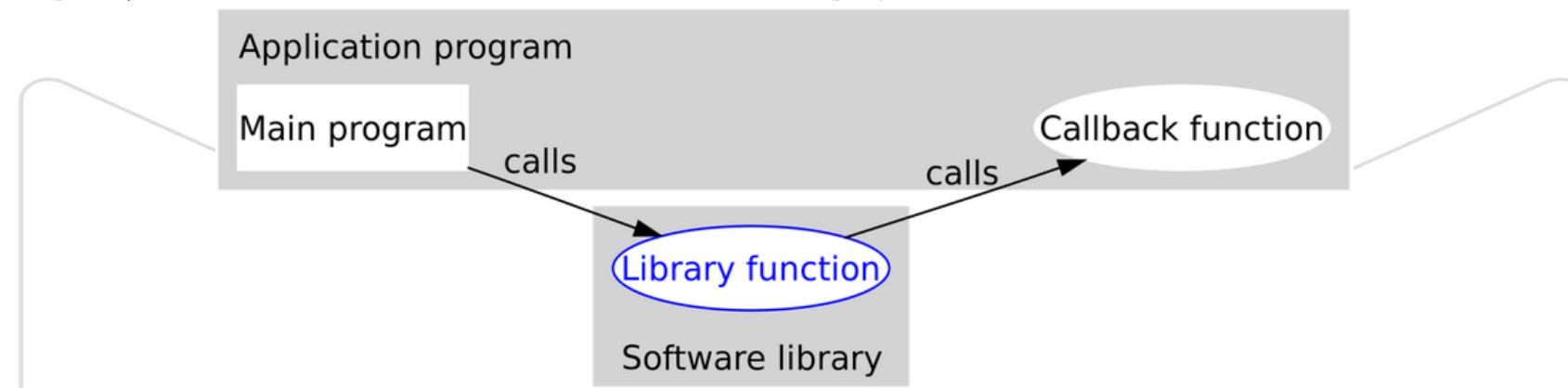
this is the 3 points of
the triangle

end of drawing

Do it!

● 回调函数 (callback)

- Wiki定义：在计算机程序设计中，回调函数，或简称回调（callback 即call then back 被主函数调用运算后会返回主函数），是指通过函数参数传递到其它代码的，某一块可执行代码的引用。这一设计允许了底层代码调用在高层定义的子程序。
- 用途：
 - 允许函数调用者在运行时调整原始函数的行为
 - 处理错误信号 (error signals)
- C/C++中常以函数指针的形式出现



● OpenGL回调函数工作流程

- 主线程运行循环**等待事件发生**
 - 鼠标、键盘操作等
- GUI框架提供**指明回调函数的机制**
 - 常使用函数指针作为参数传递
 - 将函数指针与特定事件联系
 - 当事件发生时调用对应的回调函数
- 回调函数**常带有参数**
 - 事件调度器通过参数传递额外信息
 - 如，鼠标操作的x与y坐标，键盘的键值，等

● GLUT回调函数举例

Display callback
当窗口redraw时调用

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

    glBegin(GL_QUADS);
    glColor3f(1.0f, 0.0f, 0.0);
    glVertex2f(-0.5f, 0.5f);
    glVertex2f(-0.5f, -0.5f);
    glVertex2f(0.5f, -0.5f);
    glVertex2f(-0.5f, -0.5f);
    glEnd();

    glutSwapBuffers();
}
```

Reshape callback
当窗口resize时调用

```
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();
}
```

● GLUT回调函数举例

Keyboard callback

当常规按键被按下时调用

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

当特殊按键被按下时调用

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

Mouse callback

当鼠标按键被按下时调用

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

当鼠标按键被按下时调用

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

● GLUT回调函数举例

– 用户控制结束程序

- OpenGL程序通常处于一个无限循环中，系统难以自动决定何时结束程序
- 使用回调函数由用户控制结束

```
void mousebutton(int button, int state, int x, int y)
{
    if (button==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
    {
        exit(0);
    }
}
```

● GLUT回调函数使用

- 将函数指针通过特定函数与事件关联
 - 在**init()**与**glutMainLoop()**之间通过相应的函数调用完成

```
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();  
}
```

- OpenGL是什么？
- OpenGL如何工作
- OpenGL程序结构简介
- OpenGL基本概念与语句



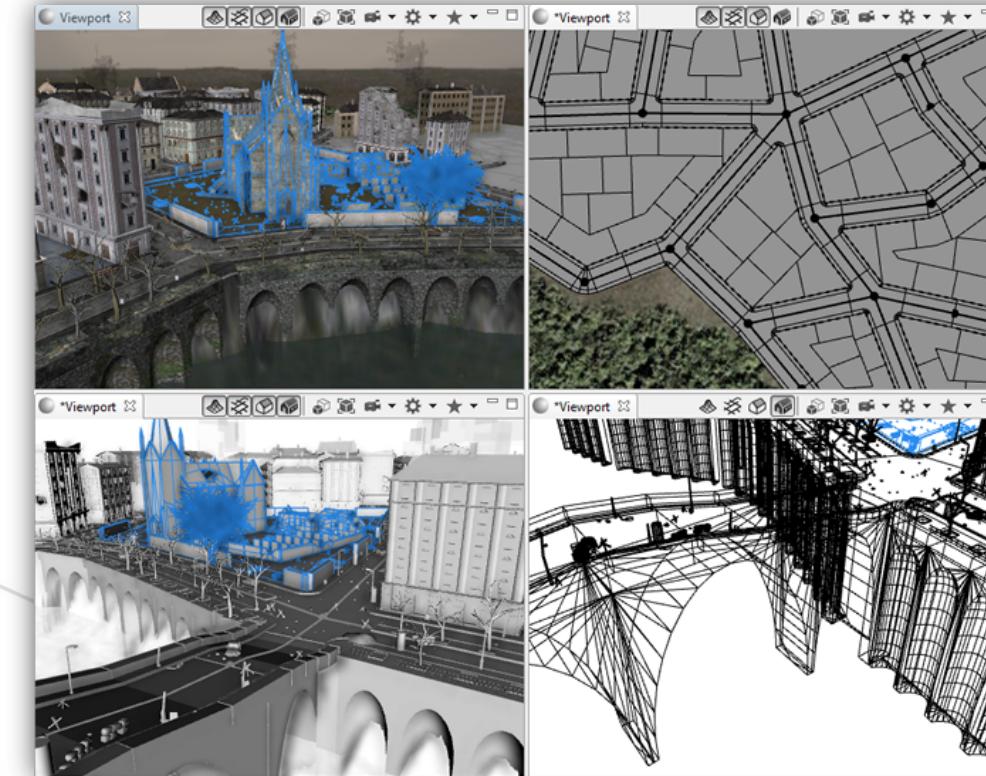
- 上下文 (context) 与视口 (Viewport)

- OpenGL **context** 存储关于当前OpenGL实例相关的所有信息
 - 当前绘制所使用的颜色、光照、光源信息、OpenGL函数调用所设置的状态和状态属性等
 - 一个OpenGL进程可以有多个context
 - 每个context对应一个可独立的绘制行为
- **Viewport** 是屏幕中依据context进行绘制的一个矩形区域
 - 可以是窗口中的一部分或整个窗口
 - 缺省使用整个窗口



● 视口 (Viewport)

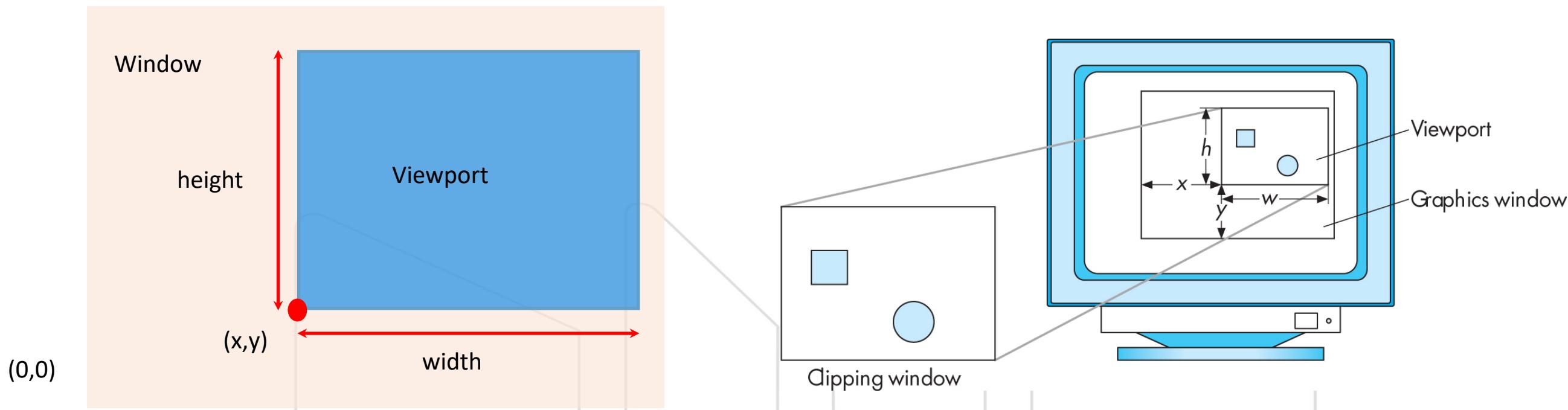
- 视口指明当前绘制区域
 - 从摄像机 (camera) 的角度看，视口为产生图片的大小
 - 将所有三维几何物体投影至视口所对应的像素区域
- 同一个窗口中可以通过多个视口进行绘制从而达到分屏效果



● 视口 (Viewport)

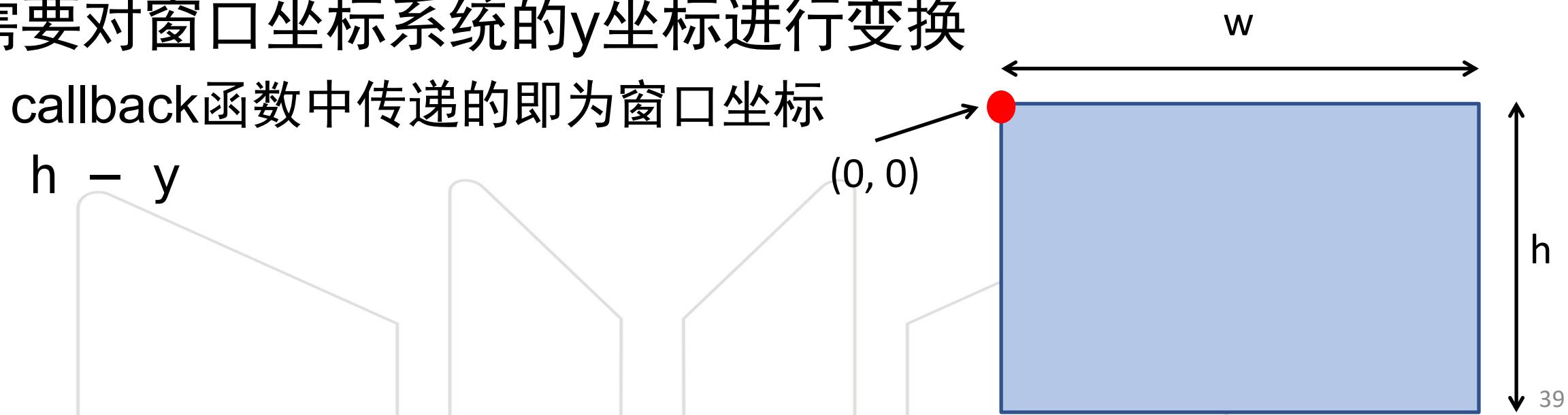
- 设置视口

- **glViewport(int x, int y, int width, int height)**
- **x**与**y**分别为视口左下角在当前窗口中的位置
- **width**与**height**指明视口的大小
- 视口的纵横比应当与当前观看的空间一致



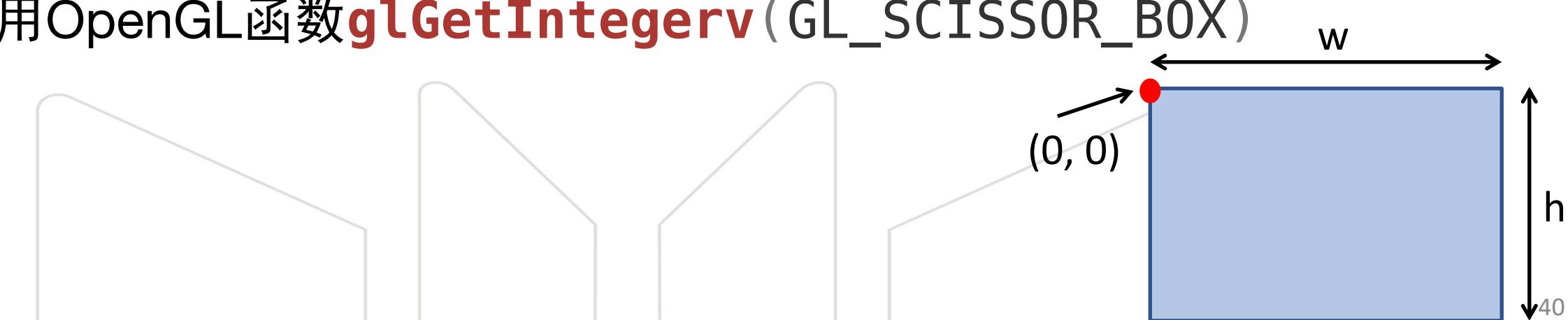
● 坐标系统

- OpenGL世界坐标系（world coordinate）与窗口的坐标系统**不同**
 - OpenGL世界坐标系原点在左下角（y轴自底向上）
 - 不以像素为单位
 - 窗口坐标系统远点在左上角（y轴自顶向下）
 - 以像素为单位
 - 原因在于显示器以自顶向下的方式刷新显示内容
- 通常需要对窗口坐标系统的y坐标进行变换
 - 如，callback函数中传递的即为窗口坐标
 - $y = h - y$



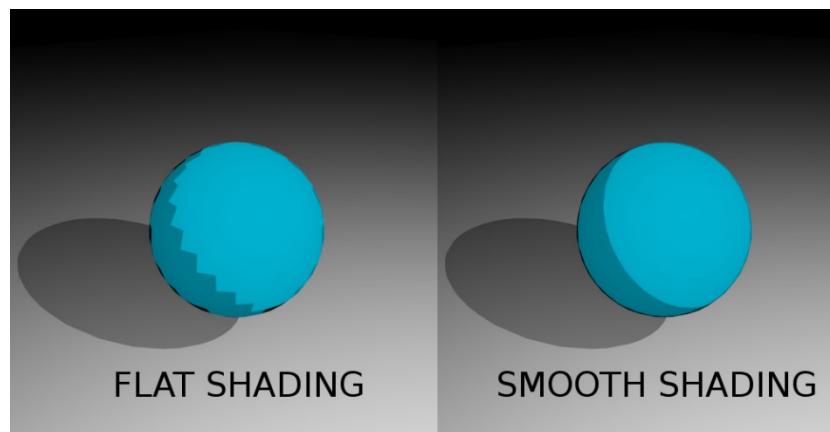
● 坐标系统

- 通常需要对窗口坐标系统的y坐标进行变换
 - 如, callback函数中传递的即为窗口坐标
 - $y = h - y$
 - 变换中需要知道窗口高度h
 - 在程序运行过程中, 窗口高度可能发生改变 (resize)
 - 可使用resize事件的callback函数在窗口大小发生变化时记录高度
 - 使用GLUT函数 **glutGet(GLUT_WINDOW_HEIGHT)**
 - 使用OpenGL函数**glGetIntegerv(GL_SCISSOR_BOX)**

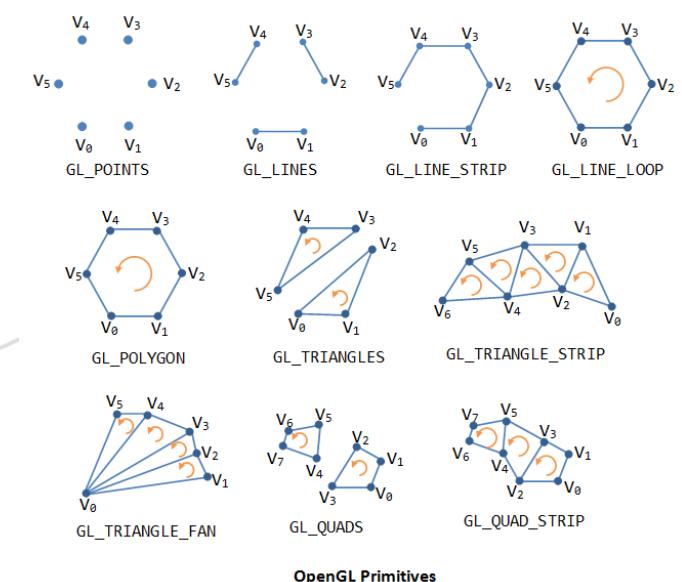
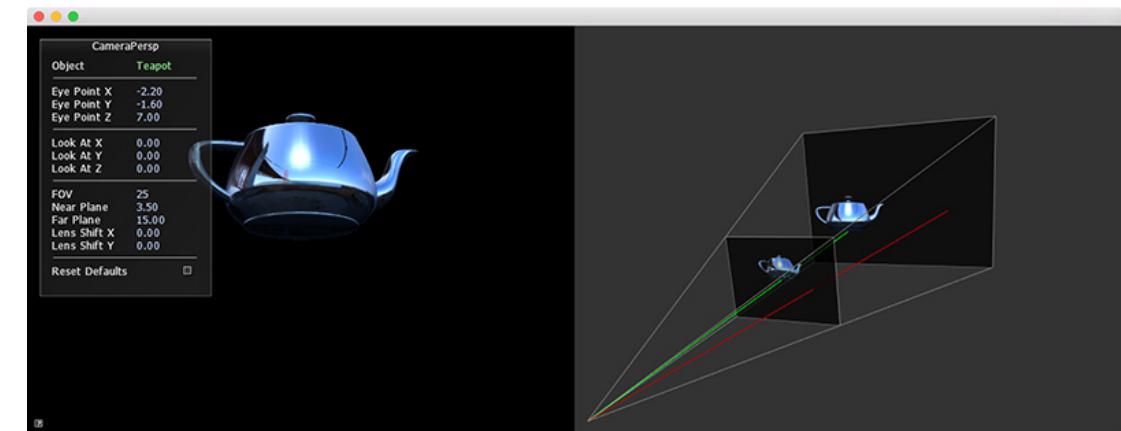


- OpenGL具有丰富的状态

- current color
- camera properties (location, orientation, field of view, etc.)
- lighting model (flat, smooth, etc.)
- type of primitive
- line width, line style, ...
- 及其他...



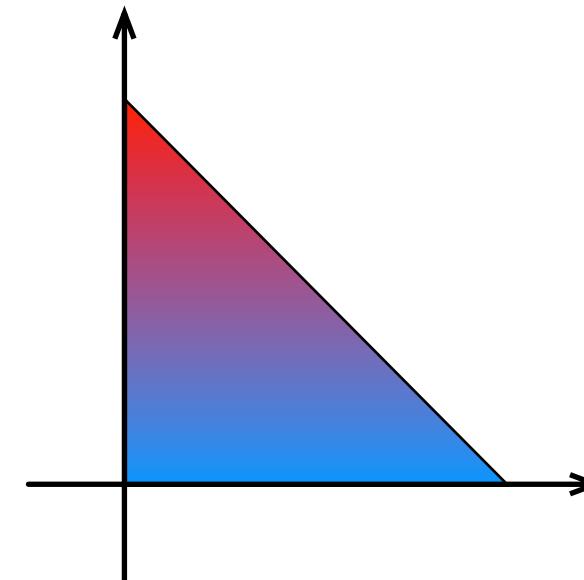
PATTERN	FACTOR			
0x00FF	1	——	——	——
0x00FF	2	—	—	—
0x0C0F	1	—	—	—
0x0C0F	3	—	—	—
0xAAAA	1	---	---	---
0xAAAA	2	—	—	—
0xAAAA	3	—	—	—
0xAAAA	4	—	—	—



- OpenGL状态设置

- 状态一经设置，在下次设置前将保持不变

```
glColor3f(0.0f, 0.0f, 1.0f);
glVertex2f(0.0f, 0.0f);
glVertex2f(1.0f, 0.0f);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex2f(0.0f, 1.0f);
```



- OpenGL有大量状态变量通过**glEnable()**与**glDisable()**进行设置
 - 如，**glEnable(GL_LIGHT0)**, **glEnable(GL_DEPTH_TEST)**, 等

● OpenGL输入

– 几何体以vertex list的形式输入OpenGL

- **glVertex***(), * = nt OR ntv, n – **number** (2, 3, 4),
t – **type** (i = integer, f = float, etc.), v – **vector**

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

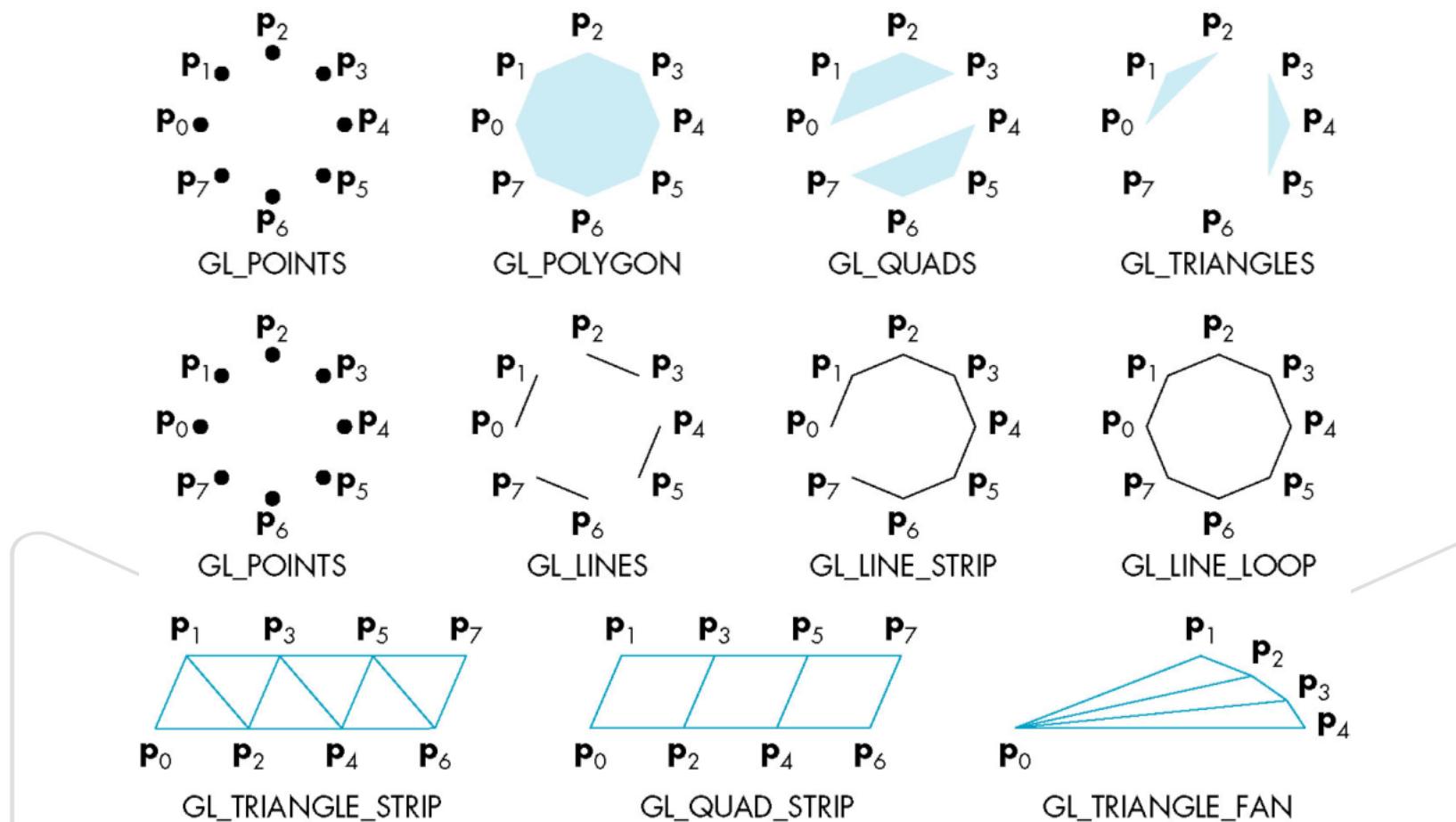
● OpenGL输入

– 几何体以vertex list的形式输入OpenGL

- `glVertex*`(), * = nt OR ntv, n – **number** (2, 3, 4), t – **type** (i = integer, f = float, etc.), v – **vector**
- 如, `glVertex2i(5, 4);`
 - 指明坐标为(5, 4)的顶点, 默认位于平面 $z=0$
 - 2 指明顶点位置为2维向量
 - i 指明向量中每一个分量为整型
- `glVertex3f(.25f, .25f, .5f);`
 - 指明坐标为(.25f, .25f, .5f)的顶点
- `double vertex[3] = {1.0, .33, 3.14159}; glVertex3dv(vertex);`
 - 指明坐标为(1.0, .33, 3.14159)的顶点
 - v 指明输入为单个数组而非多个数字

• OpenGL primitive types

- 几何体都以顶点的方式输入，但可以构成不同基本类型
 - 这些基本类型称为primitives（图元）
 - 点、线、三角形、四边形、多边形，等



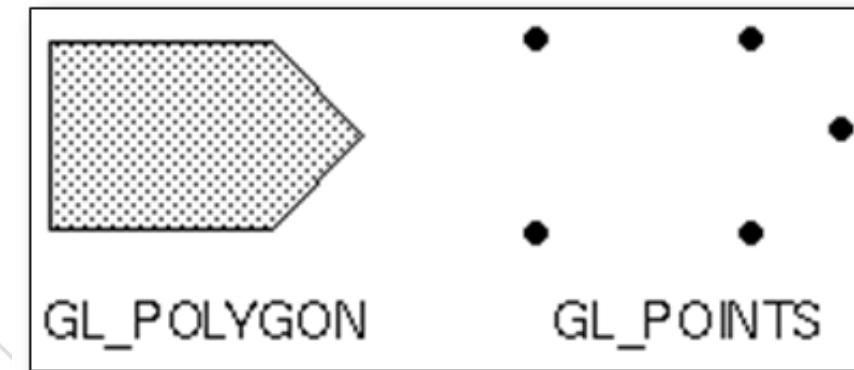
- OpenGL primitive types

- 指明primitive type

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

- 不同primitive type的效果

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



● 多边形显示模式

– **glPolygonMode(GLenum face, GLenum mode);**

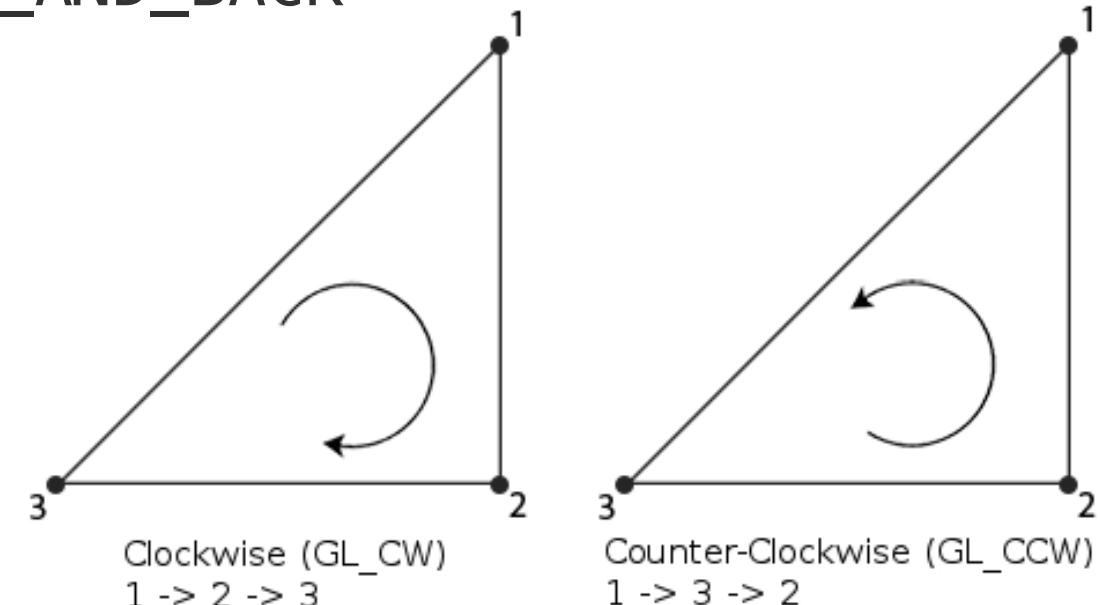
- 指明多边形如何显示
- face: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
- mode: GL_FILL, GL_LINE, GL_POINT

– **glFrontFace(GLenum mode);**

- 指明如何判断前向面
- mode: GL_CCW, GL_CW

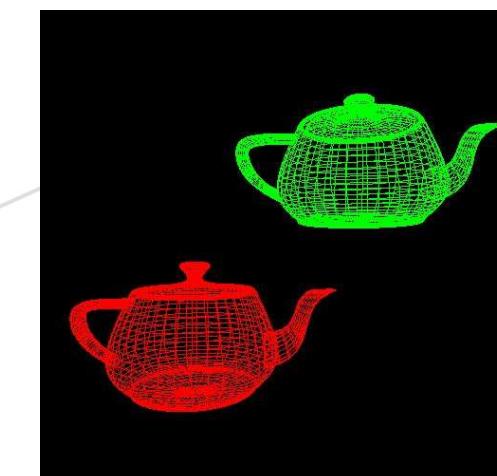
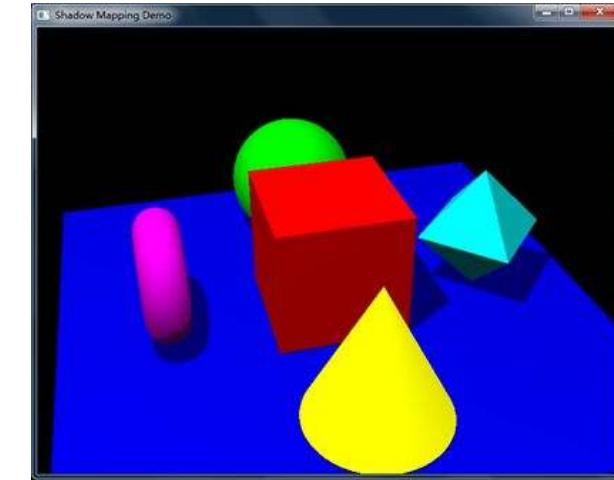
– **glCullFace(GLenum mode);**

- 指明剔除前向或（及）后向面
- mode: GL_FRONT, GL_BACK, GL_FRONT_AND_BACK
- 需要打开或关闭GL_CULL_FACE
- **glEnable(GL_CULL_FACE), glDisable(GL_CULL_FACE)**



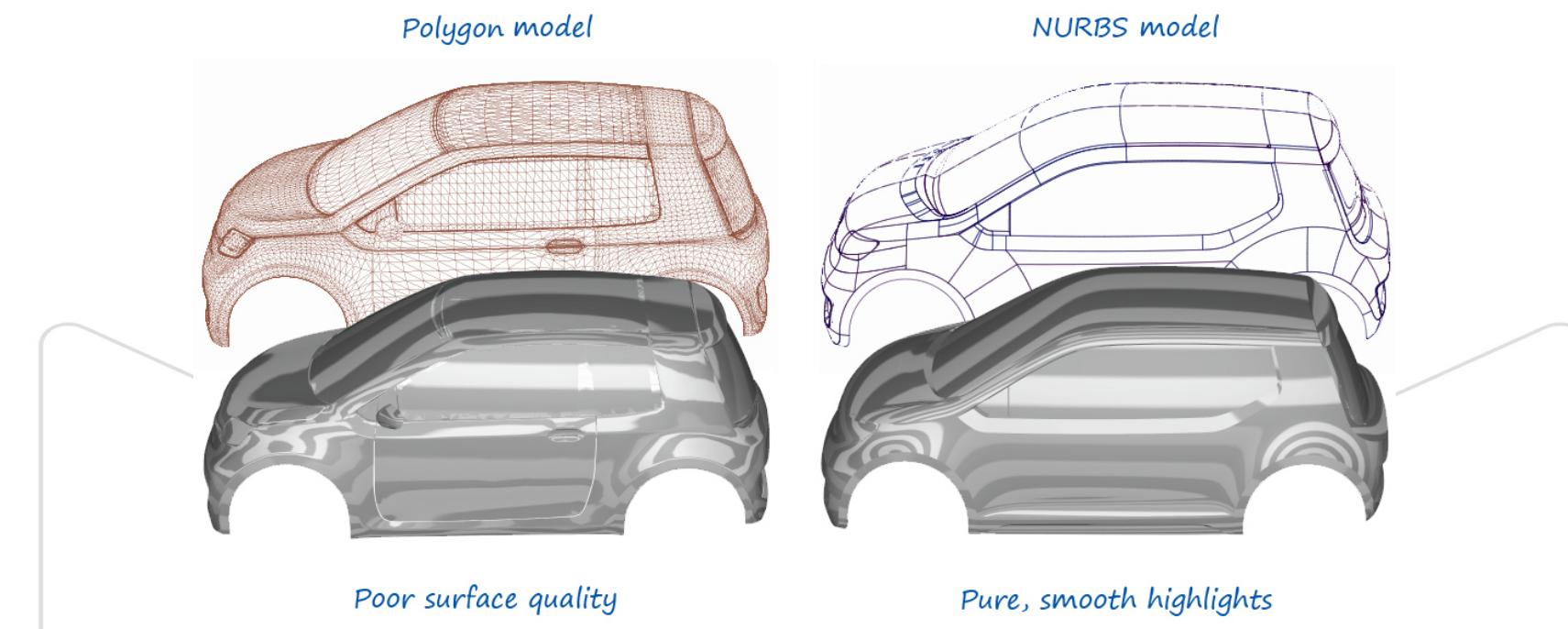
- OpenGL颜色

- OpenGL语句使用RGBA指定颜色
 - 浮点型每个分量的范围为[0.0, 1.0], 而整型为[0, 255]
- 指定及应用背景颜色
 - `glClearColor(0.0f, 0.0f, 0.0f, 0.0f);`
 - `glClear(GL_COLOR_BUFFER_BIT);`
- 指定物体颜色
 - `glColor3f(1.0f, 1.0f, 1.0f);`
 - `glColor3i(255, 255, 255);`
 - `glColor3fv(color_array);`
 - `glColor4f(1.0f, 1.0f, 1.0f, 1.0f);`



● 绘制其他物体

- GLU中有绘制圆柱体，圆锥体，及更复杂的NURBS曲面的函数
 - NURBS：非均匀有理B样条（Non-Uniform Rational B-Spline）
 - 1991年，ISO将NURBS作为定义工业产品几何形状的唯一数学方法
 - 贝塞尔曲线的一般形式
- GLUT中有绘制球体，正方体等的函数



- 完成OpenGL绘制

- OpenGL命令并非立即执行
 - 所有命令被置于一个command buffer中
- 当绘制命令结束时，需要将这些命令发送至显卡
 - 强制所有命令开始执行
 - **glFlush()**: 异步执行（函数调用后立即返回）
 - **glFinish()**: 同步执行（绘制结束后返回）



- OpenGL中的矩阵（两类）

- 建模观察矩阵（ModelView）

- 三维到三维
 - 将顶点从object coordinate转至eye coordinate

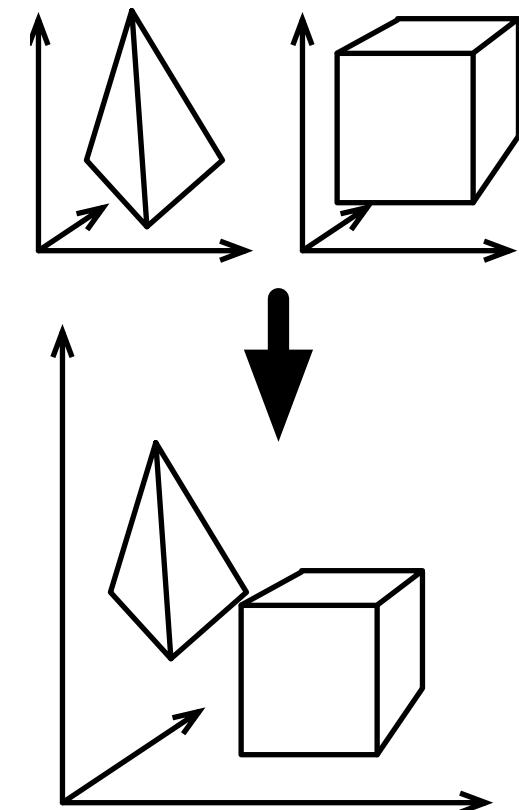
- 投影矩阵（Projection）

- 三维到二维（某种意义上说）
 - 将顶点从eye coordinate转至clip coordinate

- 所有矩阵被置于两个矩阵堆栈之中

- ModelView matrix (GL_MODELVIEW)
 - Projection matrix (GL_PROJECTION)

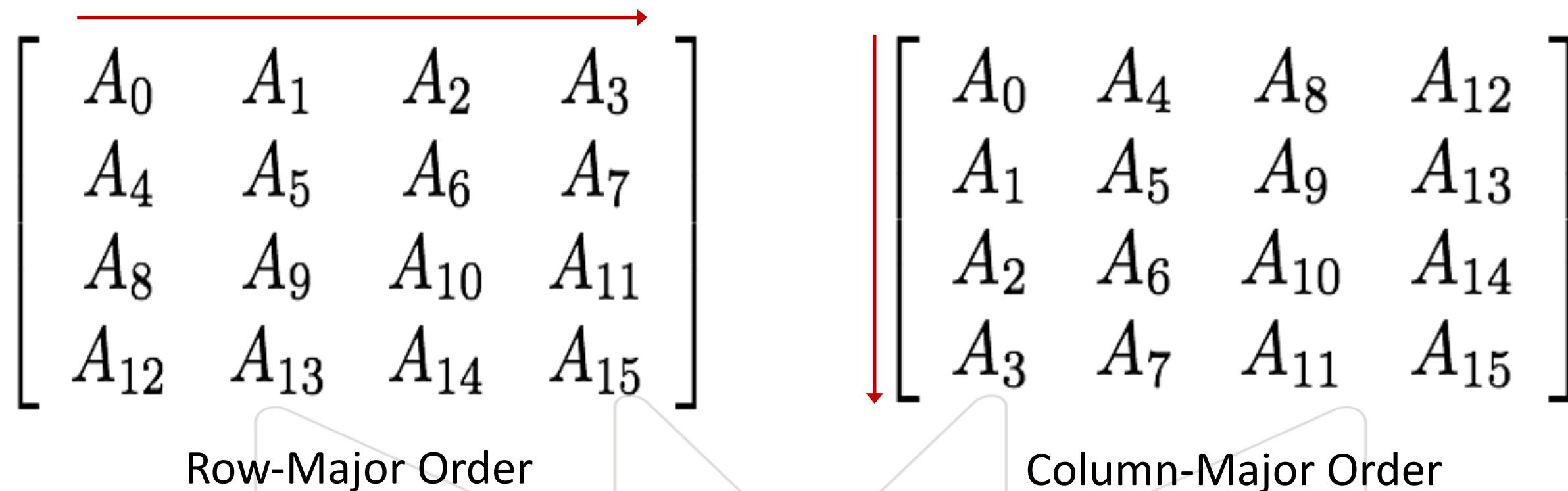
- 在使用矩阵进行变化前，
应先切换至相应的堆栈



```
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...
```

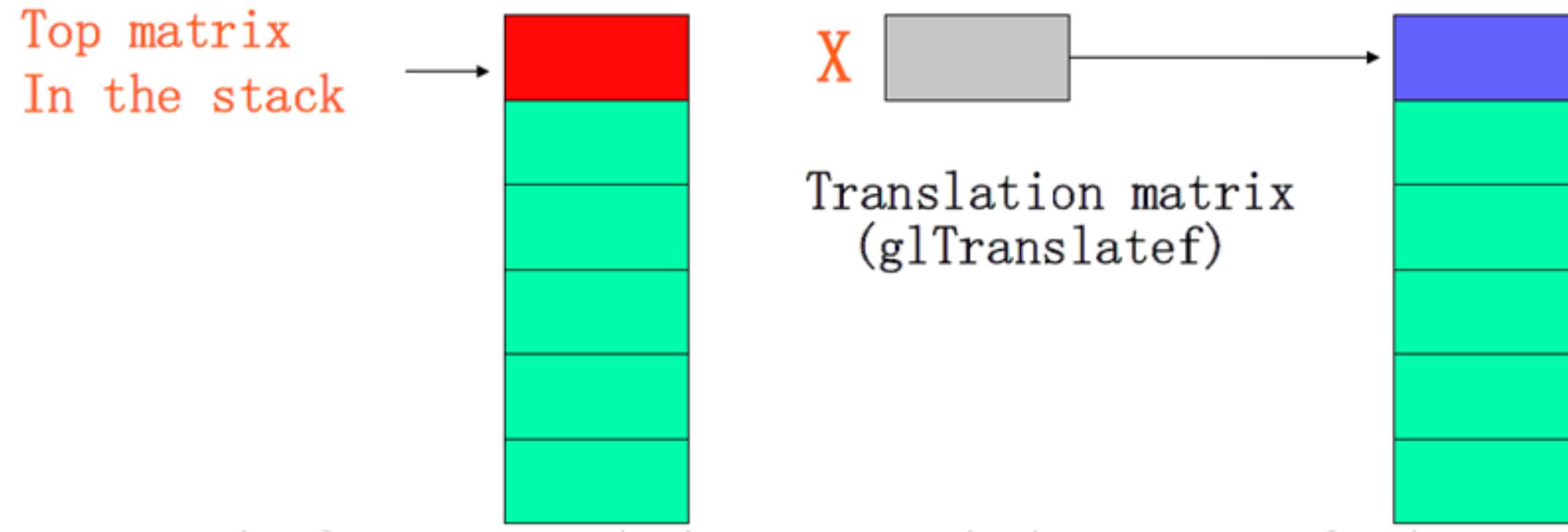
● OpenGL中的矩阵

- C/C++中矩阵是以row-major的方式存储于内存中的
- OpenGL中矩阵是以column-major的方式存储的



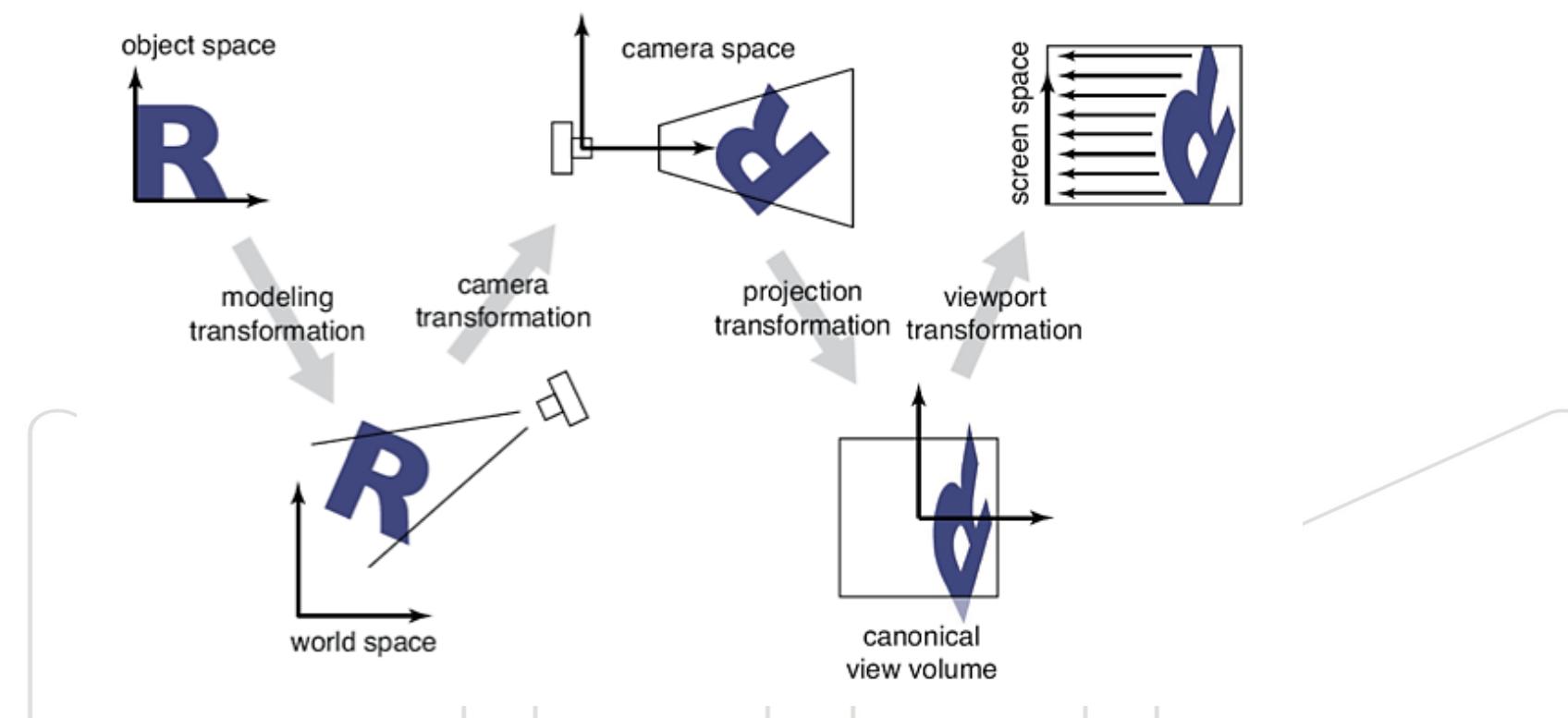
- OpenGL中的矩阵

- 矩阵操作永远作用于栈顶矩阵



● OpenGL中的矩阵

- 建模变换 (modeling transformation)
 - 对物体进行旋转、平移、缩放、及其组合而成的变换
- 观察变换 (viewing transformation)
 - 放置摄像头并调整其朝向



● OpenGL中的矩阵

- 在OpenGL中modeling与viewing transformation被合并为一个矩阵，即modelview矩阵
 - Modeling transformation: 放置物体
 - Viewing transformation: 放置摄像头
- 其原因可想象在现实世界中如何在镜头中平移一个物体
 - 移动物体或移动摄像头



● OpenGL中的矩阵

- OpenGL中所有变换都通过 4×4 矩阵完成
 - 无需手动构建矩阵，而是使用如下函数
- **glRotate{fd}(angle, x, y, z);**
 - 将场景以向量(x, y, z)为旋转轴，旋转angle度
- **glTranslate{fd}(x, y, z);**
 - 将场景平移(x, y, z)
- **glScale{fd}(x, y, z);**
 - 将场景沿三个坐标轴方向分别缩放(x, y, z)

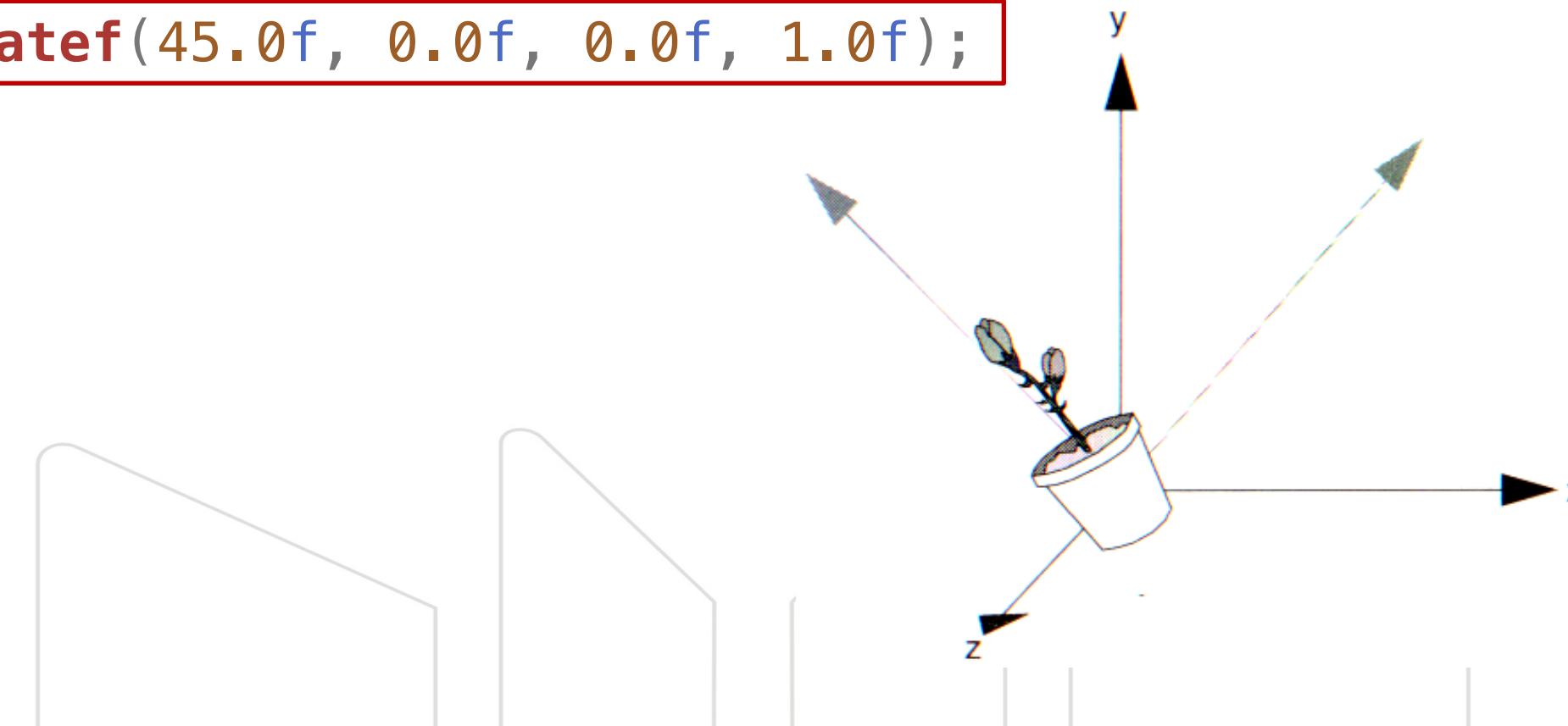


- OpenGL中的矩阵

- **glRotate{fd}(angle, x, y, z);**

- 将current matrix乘以旋转矩阵，从而将场景中所有物体以从原点到的射线为旋转轴，逆时针旋转angle度（degree）

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

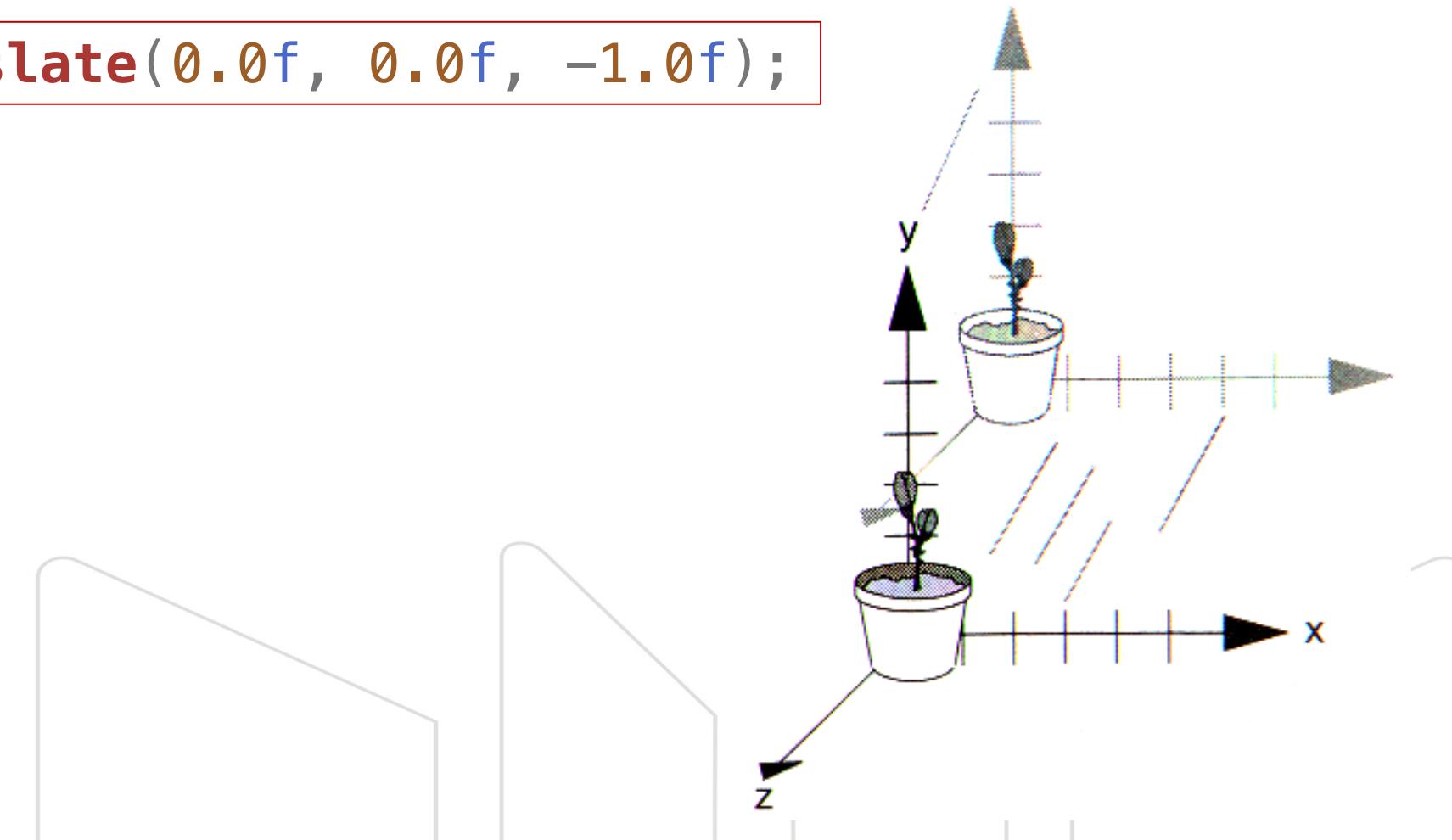


- OpenGL中的矩阵

- **glTranslate{fd}(x, y, z);**

- 将current matrix乘以平移矩阵，从而将场景中所有物体平移(x, y, z)

glTranslate(0.0f, 0.0f, -1.0f);

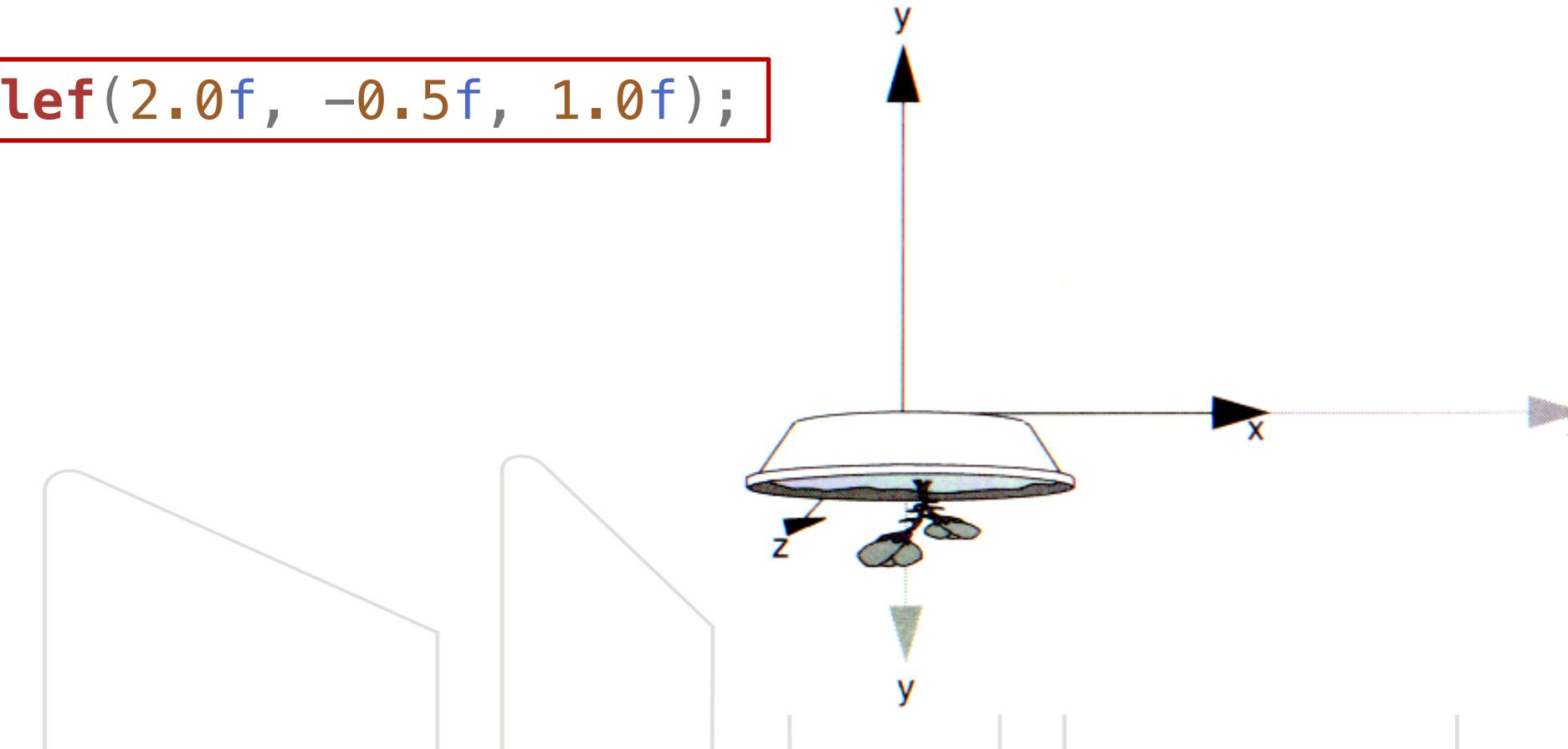


- OpenGL中的矩阵

- **glScale{fd}(x, y, z);**

- 将current matrix乘以缩放矩阵，从而使场景中所有物体沿三个坐标轴方向分别缩放(x, y, z)

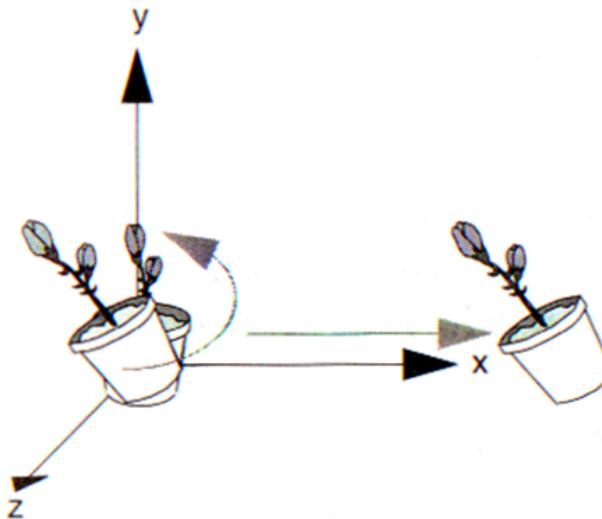
```
glScalef(2.0f, -0.5f, 1.0f);
```



● OpenGL中的矩阵

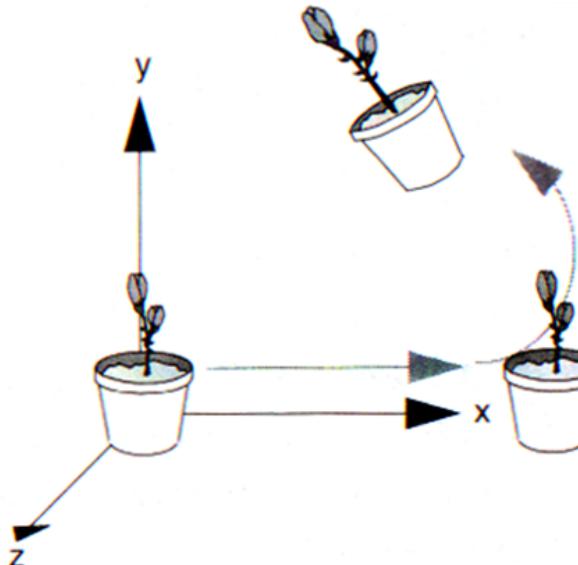
– OpenGL中，**最后调用的变换最先应用**

- 一系列变换是通过不断对栈顶矩阵（current matrix）右乘得到
- 可以从变换顶点坐标与变换坐标系统两个角度去考虑



Rotate then Translate

```
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glTranslate(0.0f, 0.0f, -1.0f);
drawObject();
```



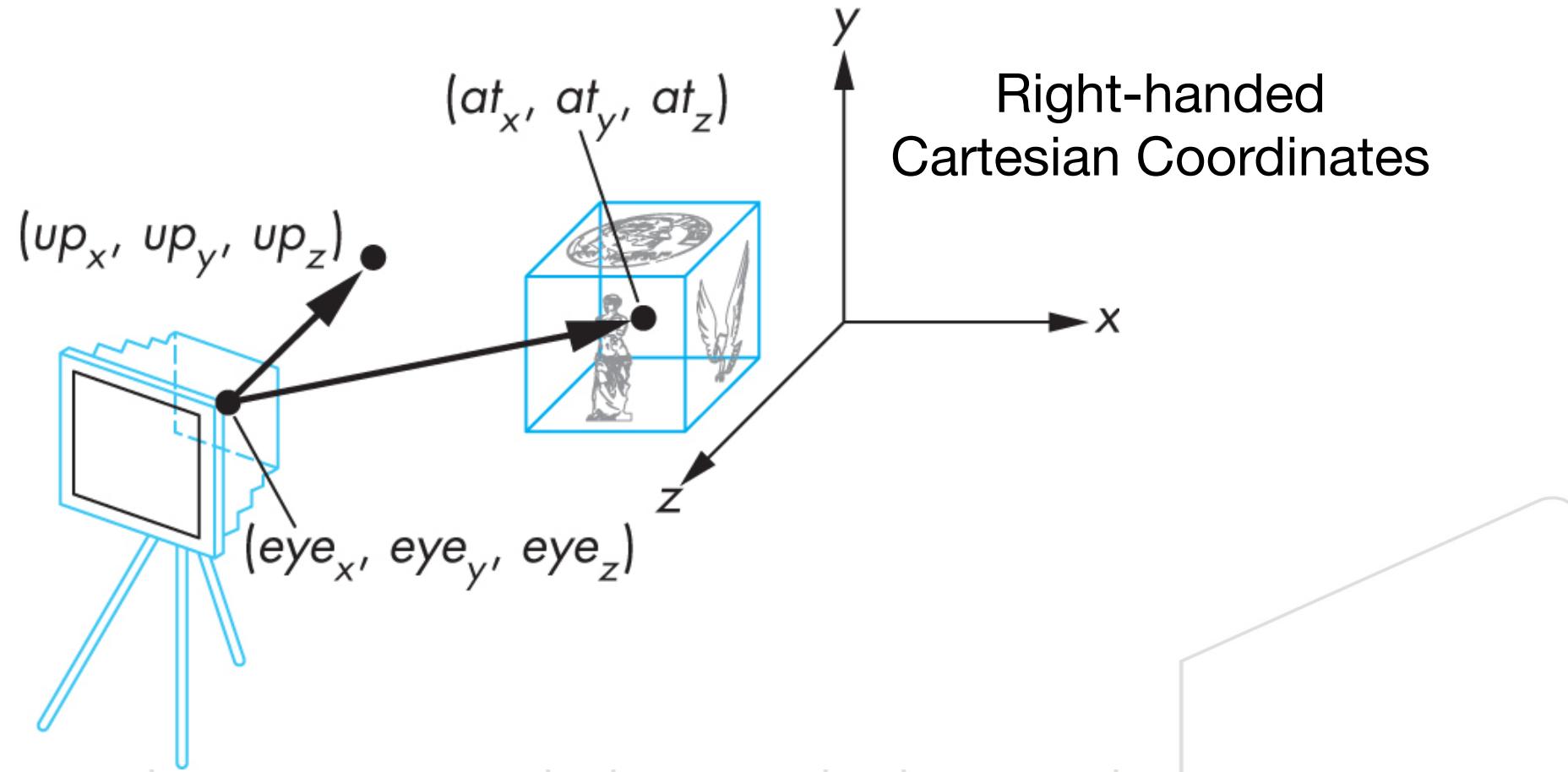
Translate then Rotate

```
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glTranslate(0.0f, 0.0f, -1.0f);
drawObject();
```

● OpenGL中的矩阵

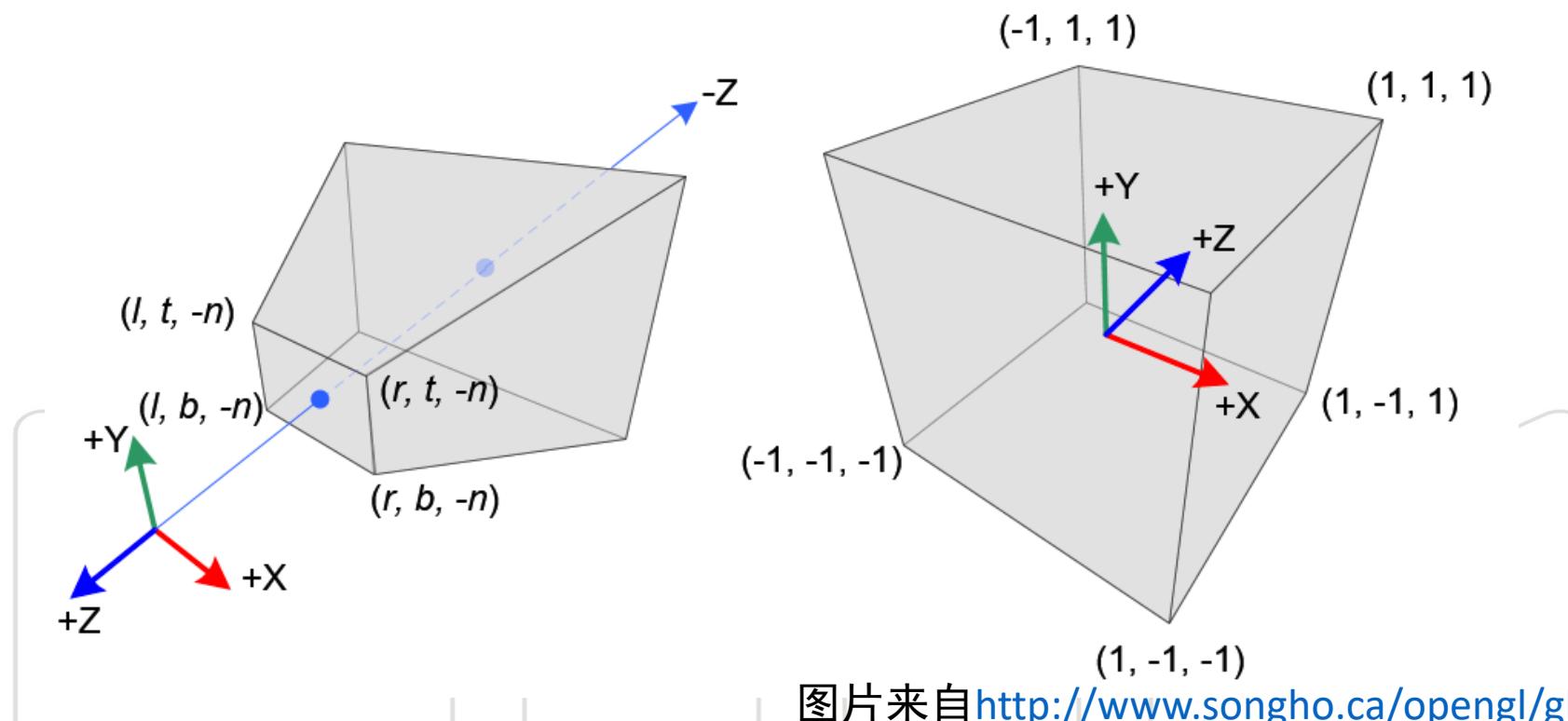
– Viewing transformation (观察变换)

- **gluLookAt** (eyex , eyey , eye_z , atx , aty , atz , upx , upy , upz) ;
- 缺省视角为，摄像头位于原点，视线朝向-z方向，以+y为上



● OpenGL中的矩阵

- 投影矩阵（projection matrix）定义了view volume
 - 决定物体如何被投影到屏幕上
 - 决定物体如何被裁剪（哪部分需要被显示）
- 视角（即摄像头的位置）就处于view volume的一端



图片来自http://www.songho.ca/opengl/gl_projectionmatrix.html 62

- OpenGL中的矩阵

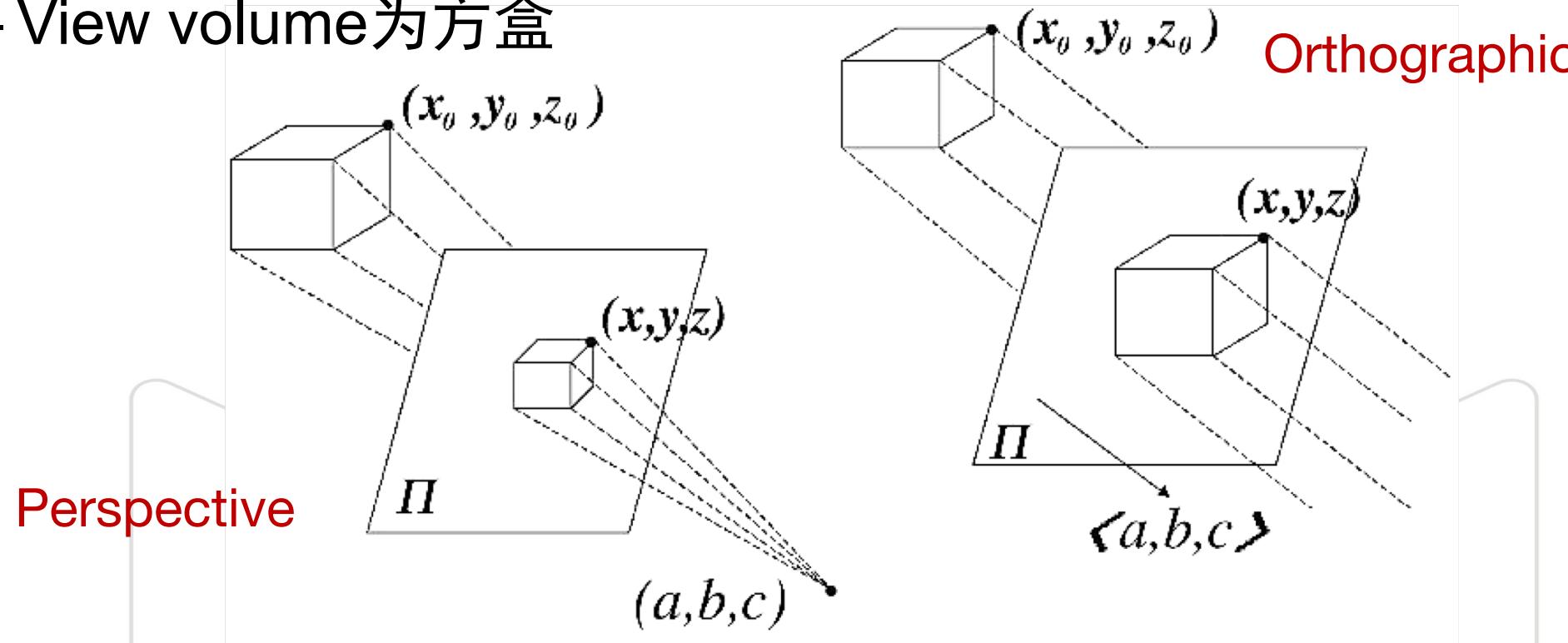
- 投影变换的两种模式

- Perspective

- View volume为截断的金字塔（即视锥，frustum）

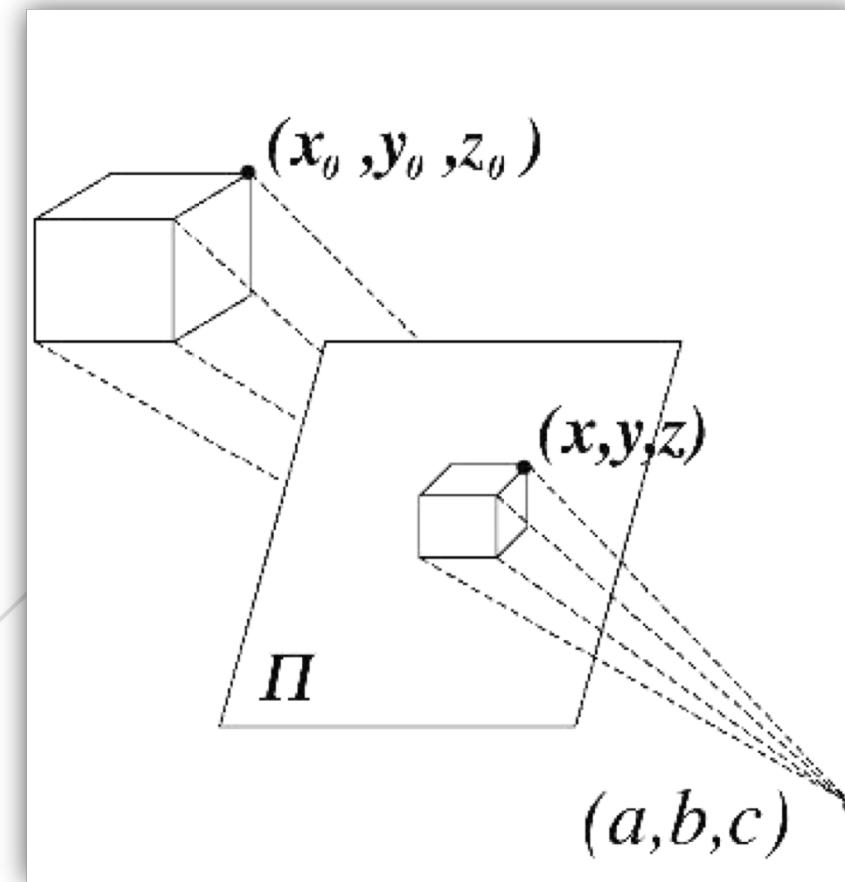
- Orthographic

- View volume为方盒



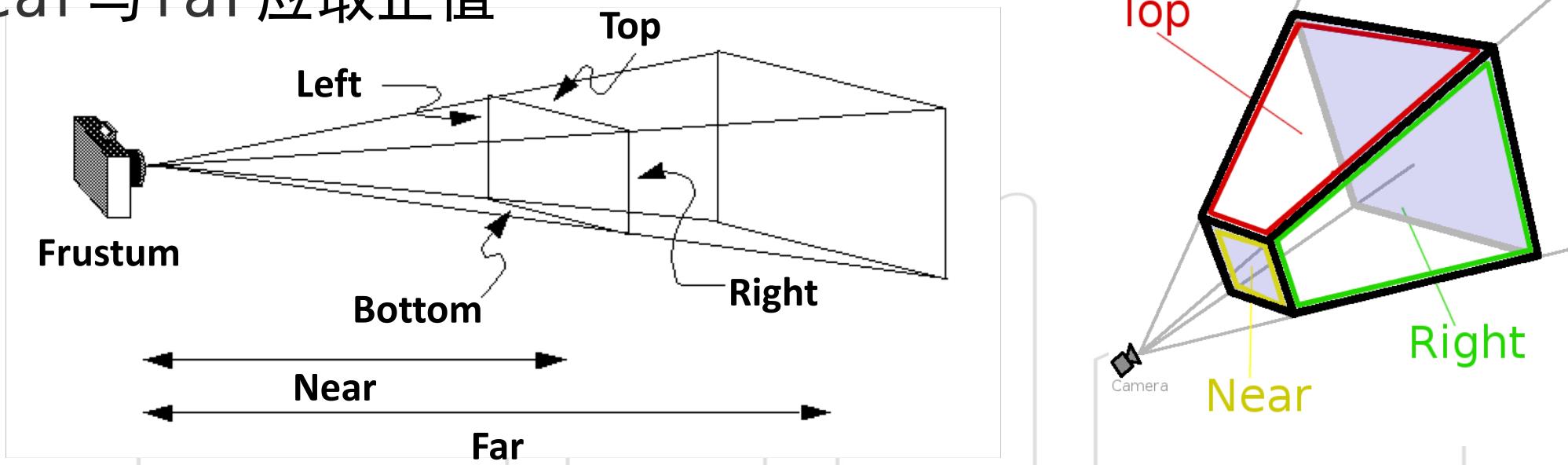
- OpenGL中的矩阵

- Perspective projection最为显著的特征为透视搜索（近大远小）
- OpenGL提供指明viewing frustum的多种方式
 - **glFrustum(...);**
 - **gluPerspective(...);**



● OpenGL中的矩阵

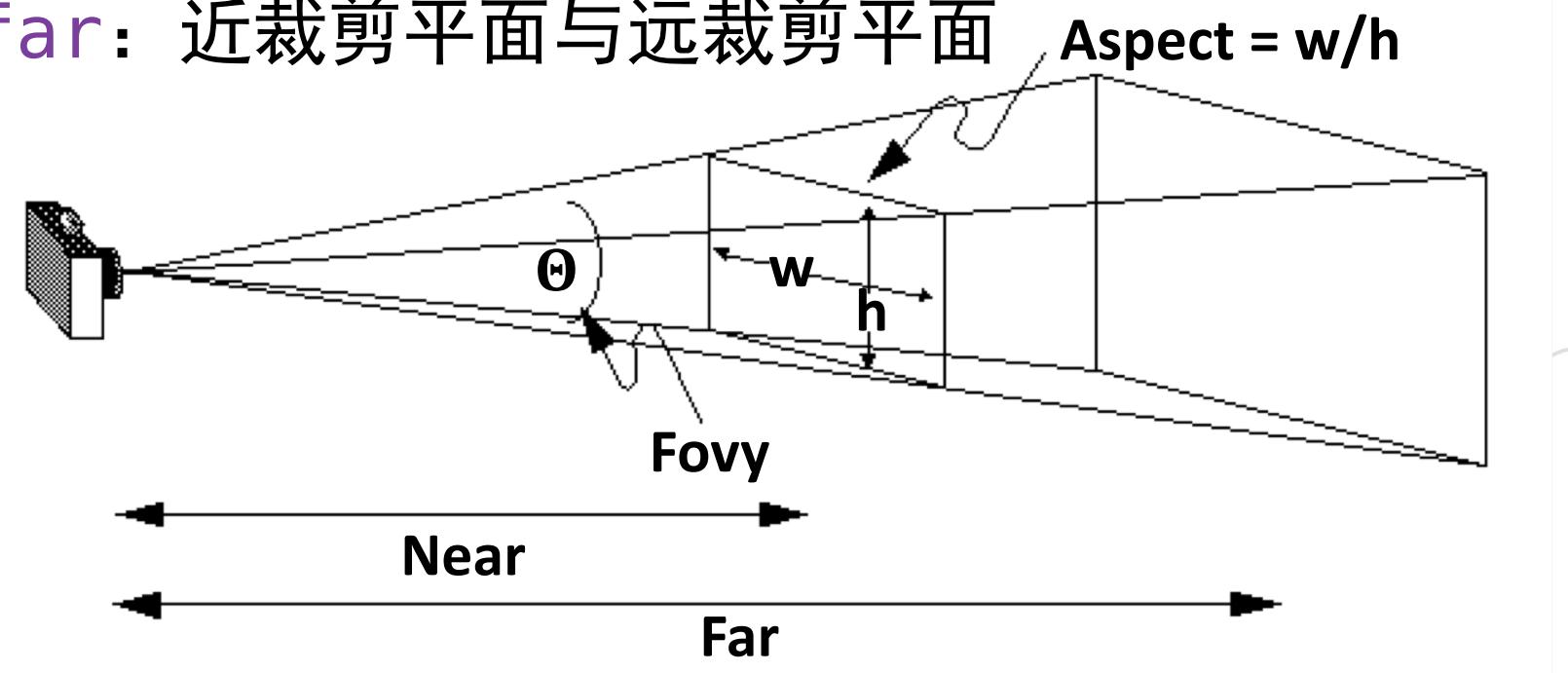
- **glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)**
 - **left, right, bottom, top**指明了近裁剪平面 (**near**) 上的边界
 - (**left, bottom, -near**) 与 (**right, top, -near**) 分别为近裁剪平面上的左下角与右上角顶点
 - **far**指明了远裁剪平面
 - **near**与**far**应取正值



● OpenGL中的矩阵

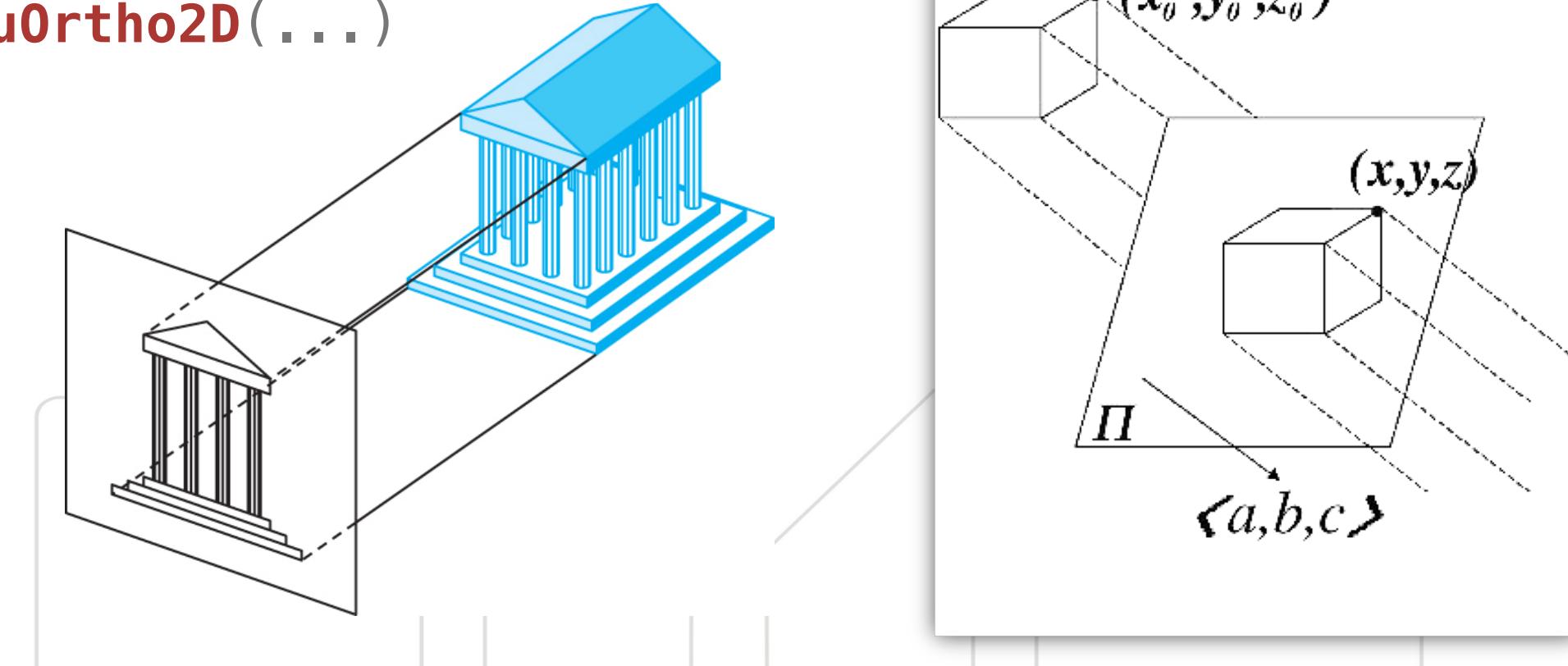
– **gluPerspective**(GLdouble fovy, GLdouble aspect,
GLdouble near, GLdouble far)

- 由GLU库提供的更简便的指定frustum的函数
- **fovy**: y方向上的视野 (field of view) 宽度, 单位为角度
- **aspect**: 纵横比 (width/height)
- **near**与**far**: 近裁剪平面与远裁剪平面



● OpenGL中的矩阵

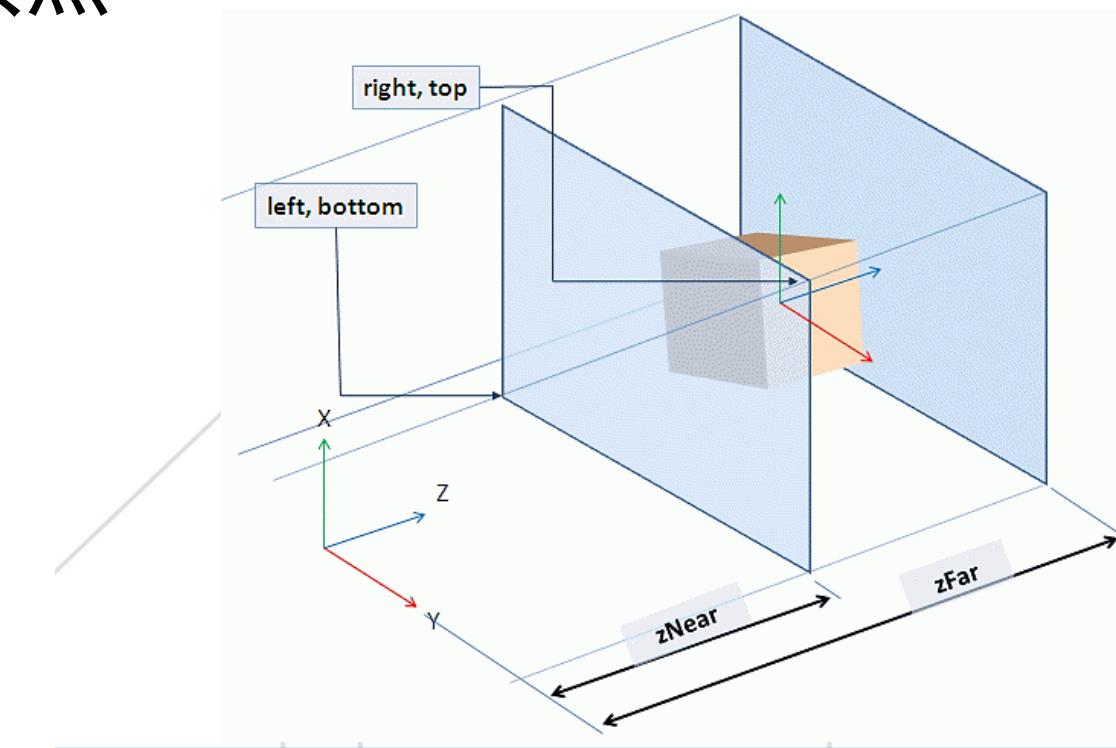
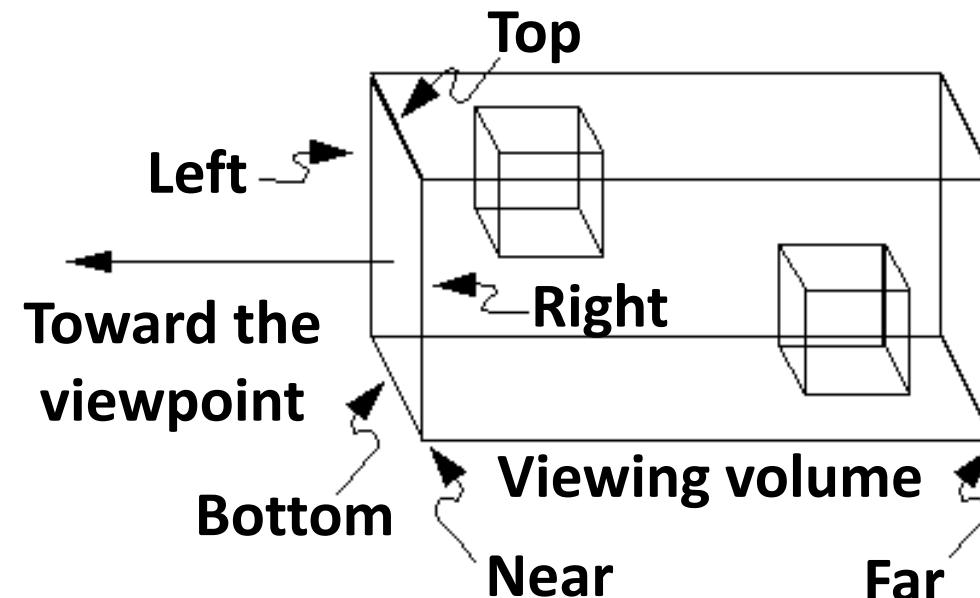
- Orthographic projection中不存在透视收缩 (foreshortening)
 - 物体与摄像头之间的距离并不改变其大小
 - 同样有多个定义orthographic projection的函数
 - **glOrtho(...)**
 - **gluOrtho2D(...)**



● OpenGL中的矩阵

– **glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)**

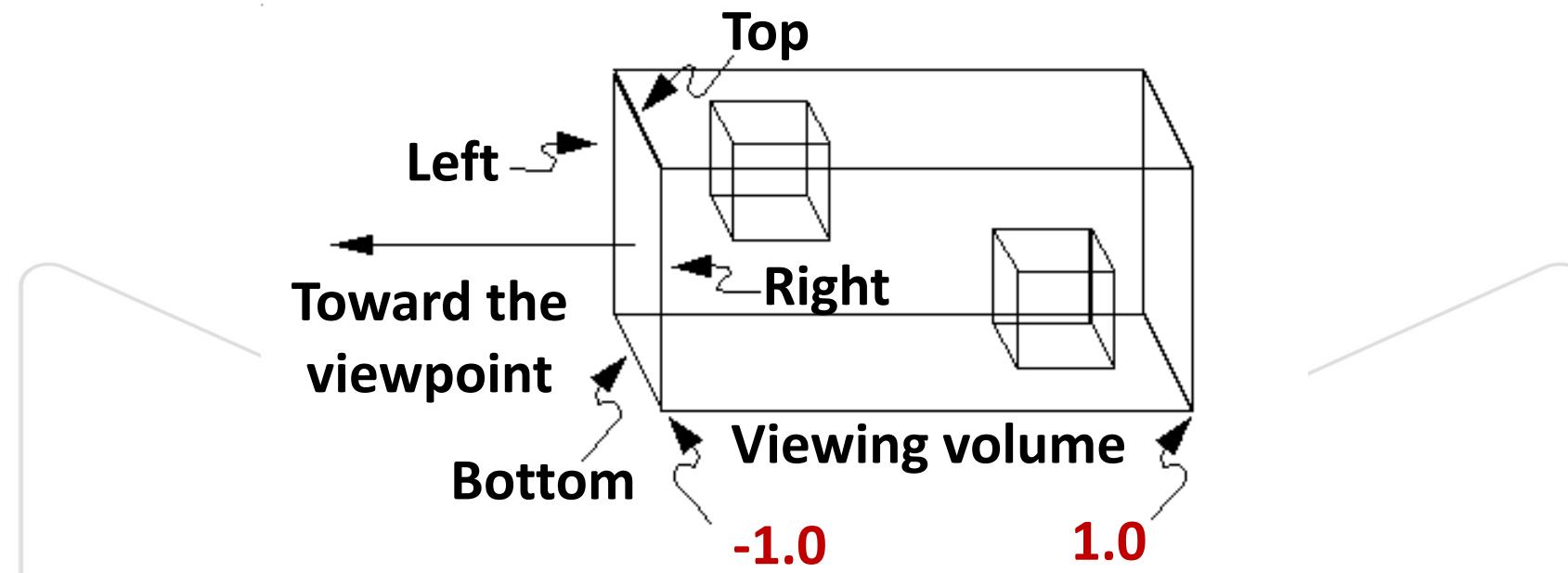
- 参数与**glPerspective()**很相似
- (*left*, *bottom*, *-near*)与(*right*, *top*, *-near*)分别为近裁剪平面上的左下角与右上角顶点
- 可以是任意不相等的两个值



● OpenGL中的矩阵

– **gluOrtho2D(GLdouble left, GLdouble right,
GLdouble bottom, GLdouble top)**

- 由GLU库提供的更简便的指定frustum的函数
- (left, bottom)与(right, top)分别为近裁区域中左下角与右上角
- 自动定义远近裁剪平面分别为-1.0及1.0
- 二维模式下, frustum与viewport等价



● OpenGL中的矩阵

`glTranslatef` `gluPerspective`

`glRotatef`

`glScalf`

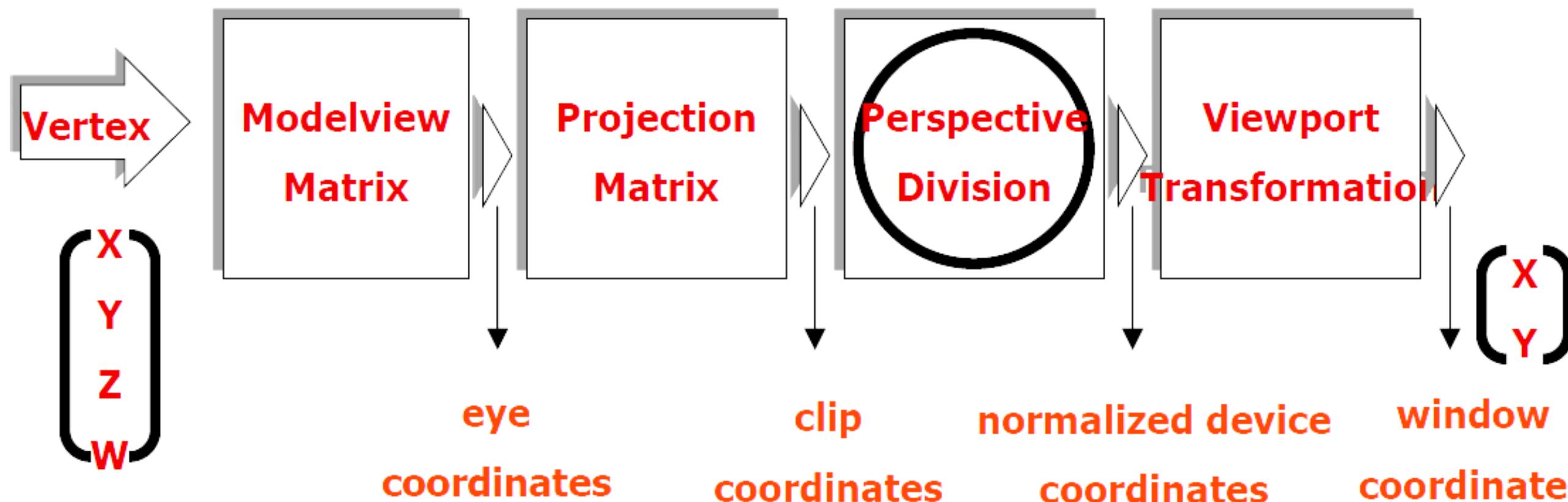
`gluLookAt`

`gluOrtho2D`

`glFrustum`

`glOrtho`

`glViewport()`



- OpenGL是什么?
 - 功能介绍、发展历程、相关库 (GLU, GLUT)
- OpenGL如何工作
 - 简易pipeline (vertex operations→rasterization→frame buffer)
 - OpenGL函数类型及其命名方式
- OpenGL程序结构简介
 - Hello world、GLUT回调函数
- OpenGL基本概念与语句
 - primitive类型
 - 矩阵 (matrix stack, modelview, projection)

Questions?

