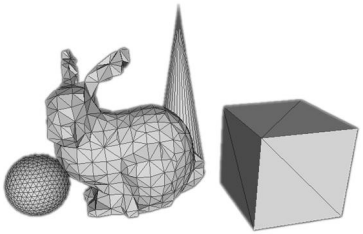
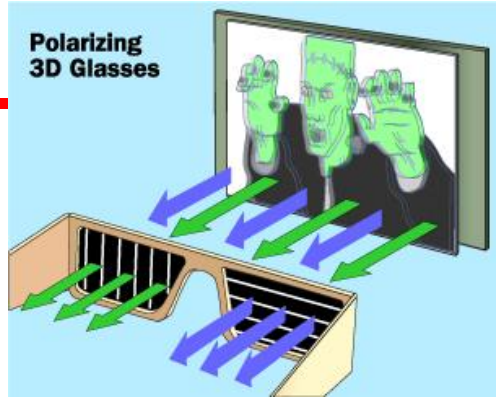
Introduction to Computer Graphics



Pipeline Architecture and Programming with OpenGL

图形学小知识



- 3D电影的原理？
- 双目视觉！

不戴3D眼镜直接观看

佩戴3D眼镜观影

上节内容回顾

- 图像生成模型
  - 四要素：几何、材质、光照、观察者
  - 视觉系统
    - 光的性质
    - 人眼构造
    - 加色/减色系统
  - 全局光照明模型（Global Illumination）与局部光照明模型（Local Illumination）
  - 针孔相机成像模型（Pinhole Camera）
    - 简单的投影计算
    - 裁剪的概念

本节内容

- 图形流水线介绍
- OpenGL简介
- OpenGL完整程序

本节内容

- 图形流水线介绍
  - The pipeline architecture of the graphics rendering procedures (图形渲染过程的体系结构）
  - Application Programming Interface （API）应用程序编程接口
- OpenGL简介
  - OpenGL是什么
  - OpenGL库的使用
- OpenGL完整程序

Image Formation Revisited
（成像模型回顾）

- Can we mimic the synthetic camera model to design graphics hardware and software? （可否模拟虚拟相机）
- Application Programming Interface (API)
  - Need only specify
    - Objects （物体）
    - Materials （材质）
    - Viewer （观察者）
    - Lights （光源）
- But how is the API implemented?

Two tasks of our course

- Learn how to use a specific API (OpenGL)

  如何用API来绘制图像

- Learn the implementation details behind the API

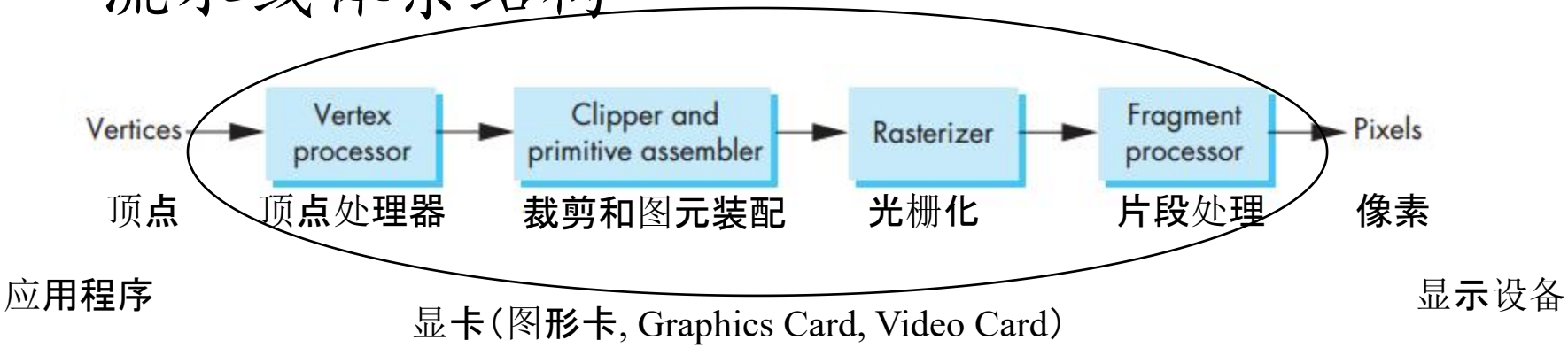
  这些API又是如何实现的呢？你自己可否写一个这样的API

Global Illumination or Local Illumination
（全局光照明 还是 局部光照明）

- Global Illumination
  - Ray Tracing光线跟踪方法
  - Radiosity 辐射度方法
  - 计算量非常大，目前没有硬件加速支持，不适合实时或交互式系统
- Local Illumination局部光照模型
  - Phong光照模型
  - 计算量适中，有硬件加速支持
  - 当前主流API：OpenGL, Direct3D

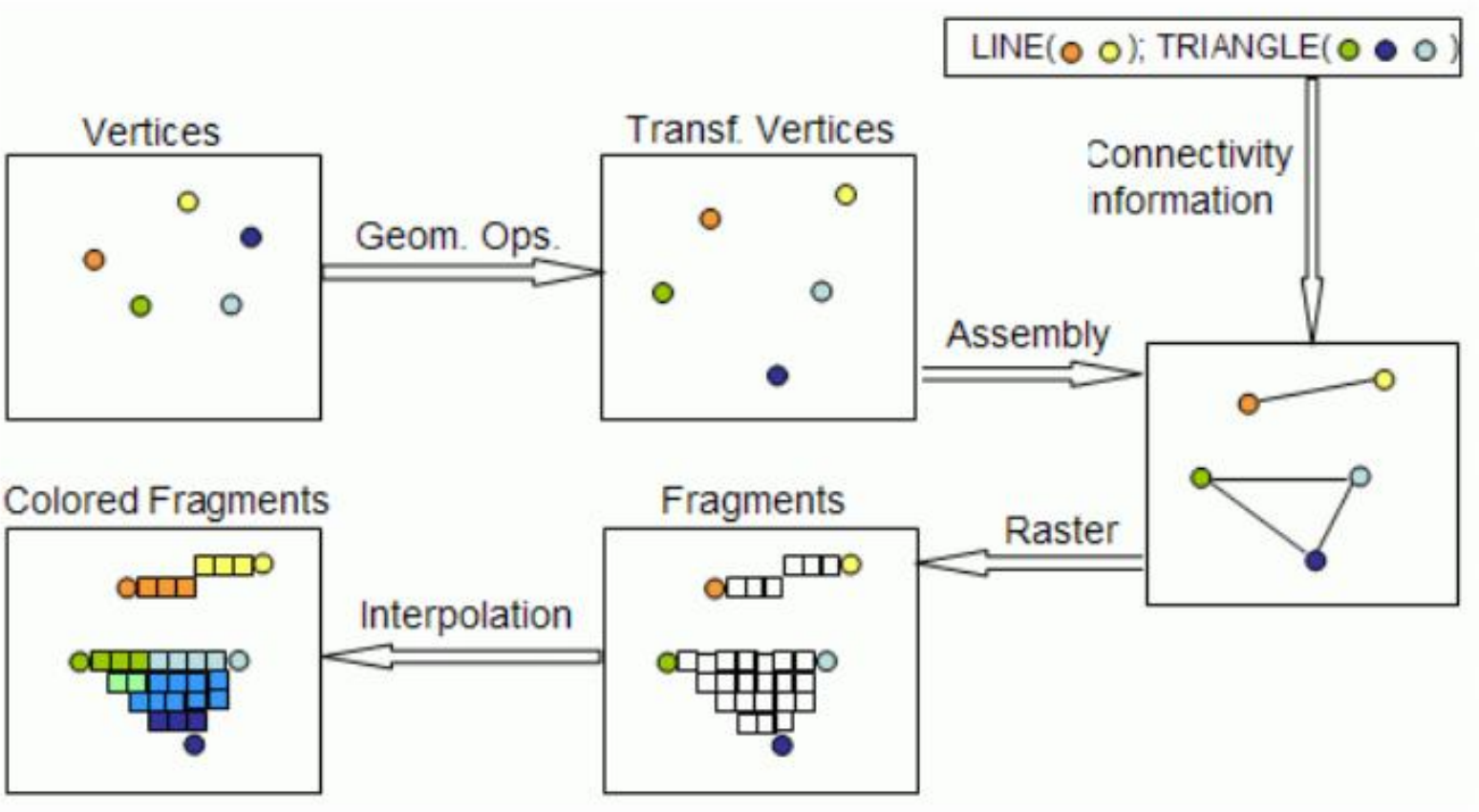
Pipeline architecture
流水线体系

- 按照应用程序定义对象的先后顺序，依次处理每个对象
  - 只考虑局部光照
- 流水线体系结构



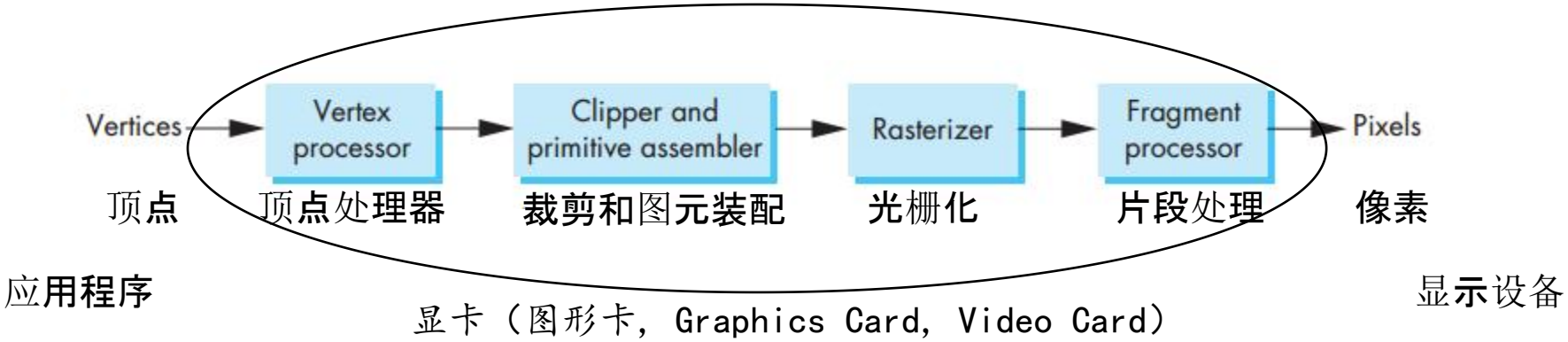
- 所有步骤都可以通过显示卡的硬件实现

Graphics Pipeline
(图形渲染流水线)



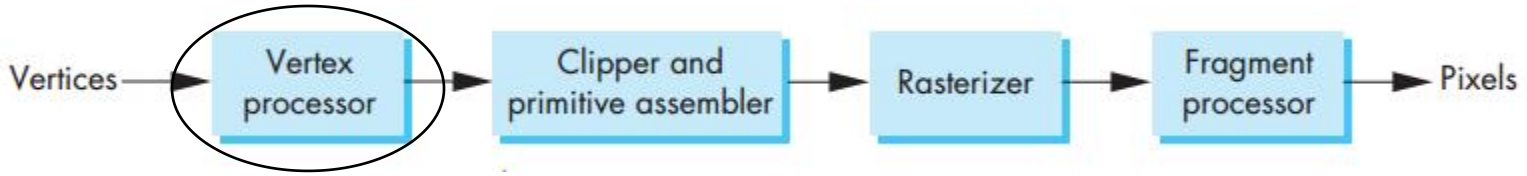
主要步骤

- 四个主要步骤：
  - 顶点处理
  - 裁剪和图元装配
  - 光栅化
  - 片段处理



顶点处理

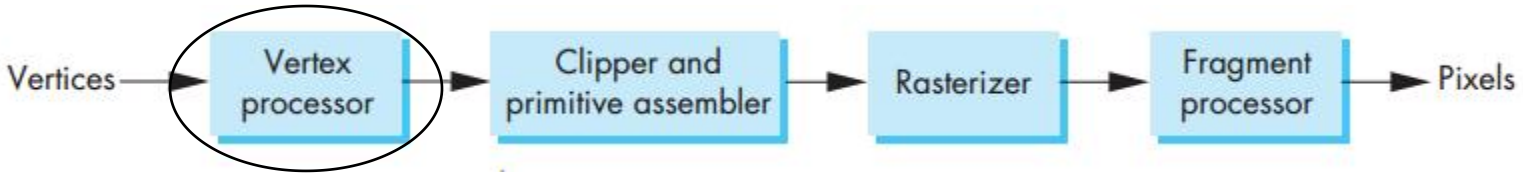
- 图元的类型和顶点集定义了场景的几何
  - 对象由一组图元组成，而每个图元又包含一组顶点
- 流水线中大部分工作是把对象在一个坐标系中表示转化为另一坐标系中的表示：
  - 世界坐标系
  - 照相机(眼睛)坐标系
  - 屏幕坐标系
- 坐标的每个变换相当于一次矩阵乘法
- 顶点处理器也计算顶点的颜色



投影（Projection）

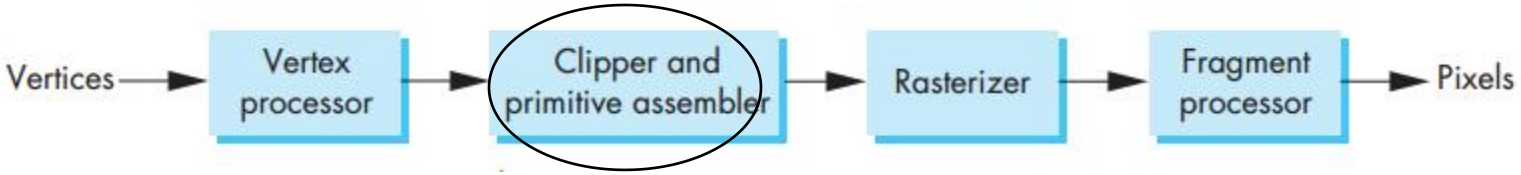
- 把三维观察者位置与三维对象结合在一起确定二维图像的构成
  - 透视投影：所有投影线交于投影中心
  - 平行投影：投影线平行，投影中心在无穷远，用投影方向表示

- 在顶点处理步中，对各个顶点的处理是相互独立的



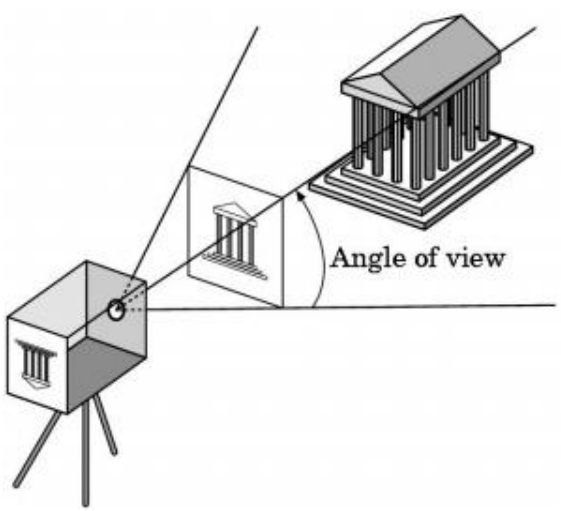
图元装配
（primitive assemble）

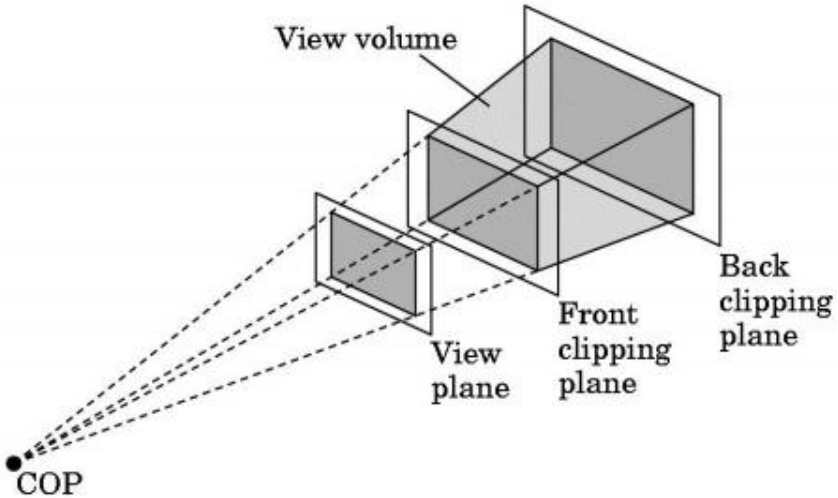
- 在进行裁剪和光栅化处理之前，顶点必须组装成几何对象
  - 线段
  - 多边形
  - 曲线和曲面



裁剪（Clipping）

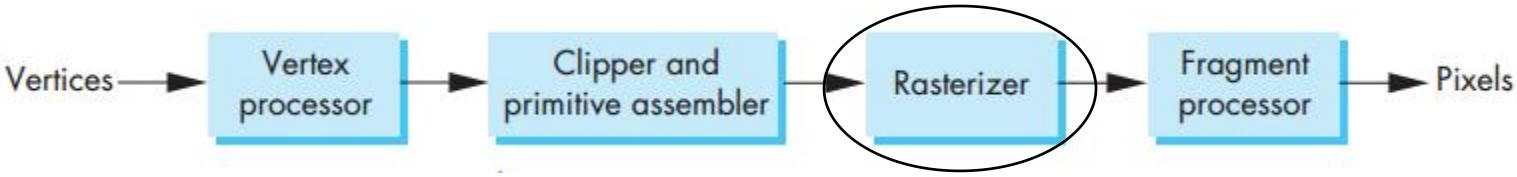
- 真正的照相机不能“看到”整个世界，图形学中的虚拟照相机也只能看到世界的一部分
  - 不在视景体中的对象要从场景中裁剪掉
- 裁剪必须针对逐个图元进行





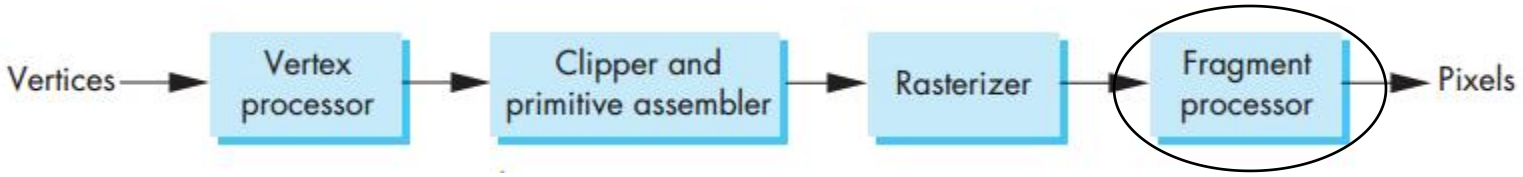
光栅化（Rasterization）

- 如果一个对象不被裁掉，那么在帧缓冲区中相应的像素就必须被赋予颜色
- 光栅化程序为每个图元生成一组片段
- 片段是“潜在的像素”
  - 在帧缓冲区中有一个位置
  - 具有颜色和深度属性
- 光栅化程序在对象上对顶点的属性进行插值得到片段的属性

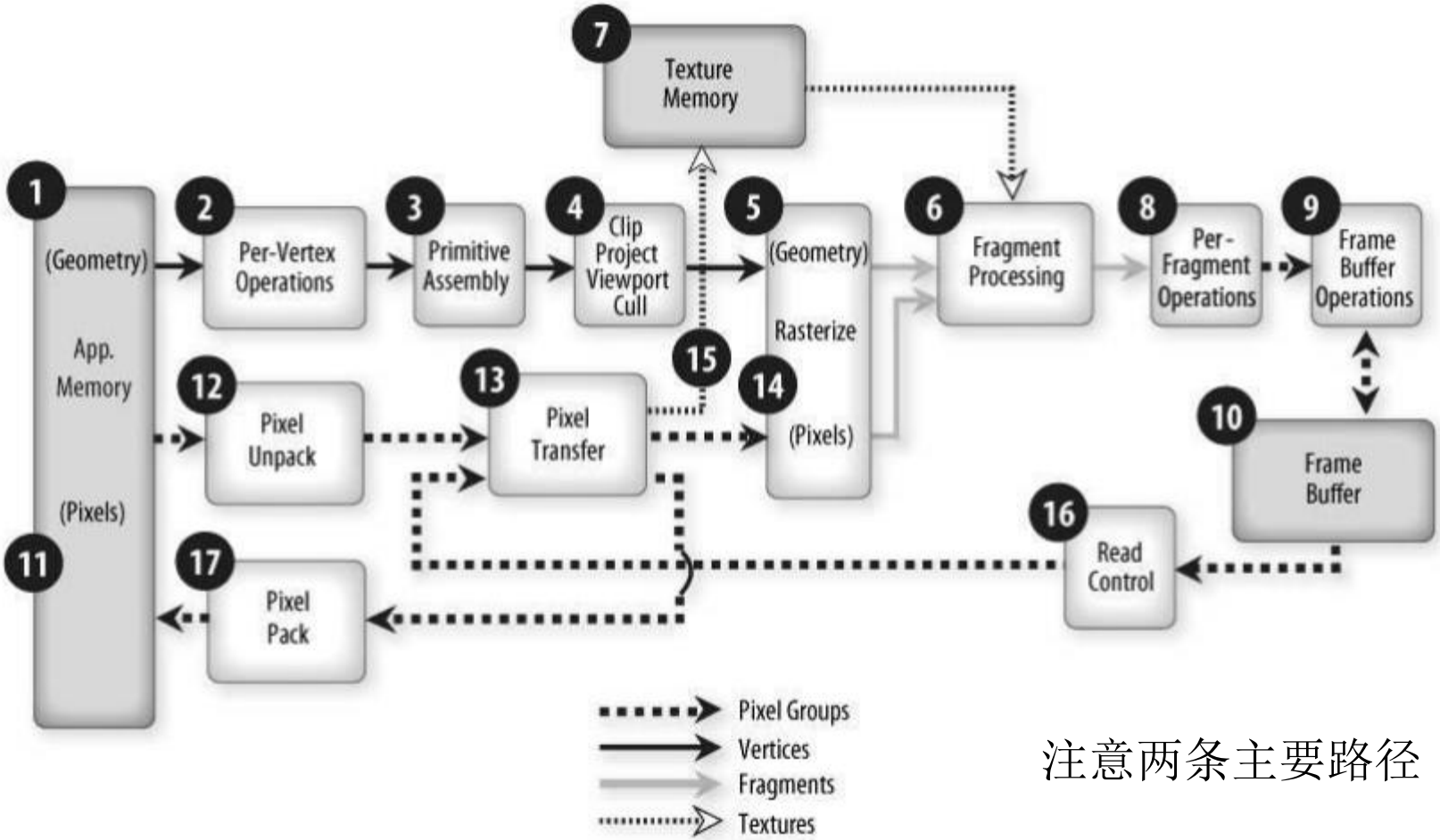


片段处理(Fragment Processing)

- 对片段进行处理，以确定帧缓冲区中相应像素的颜色
- 颜色可以由纹理映射确定，也可以由顶点颜色插值得到
- 片段可能被离照相机更近的其它片段挡住
  - 隐藏面消除

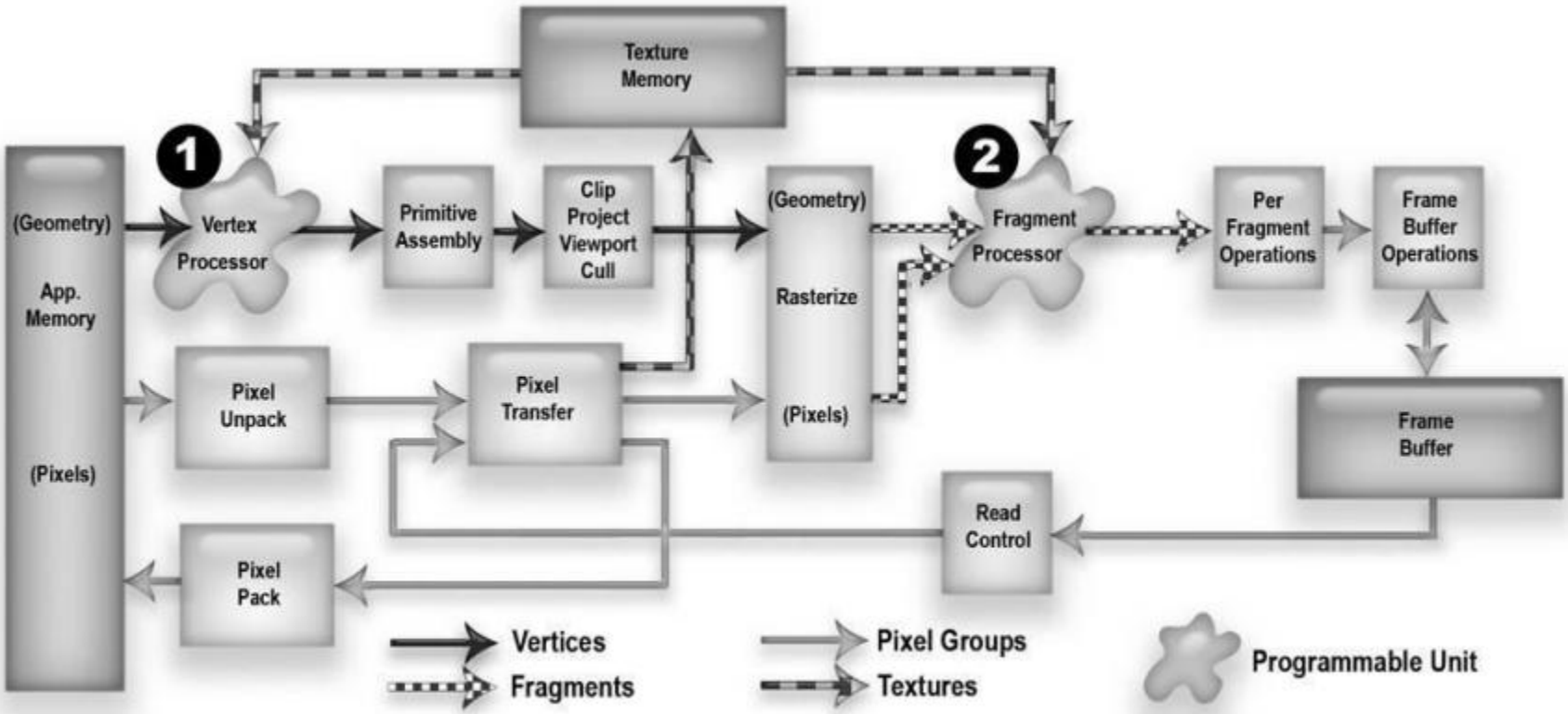


固定功能流水线(fixed function pipeline)



注意两条主要路径

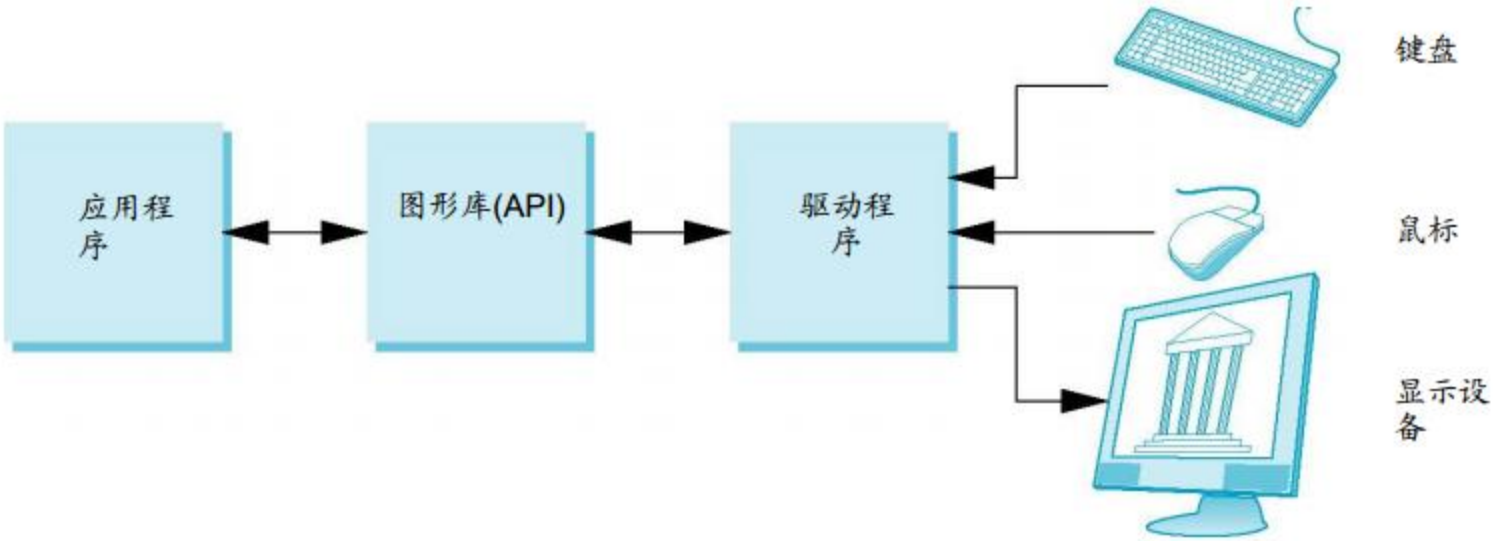
可编程流水线
(Programmable Pipeline)



注意两条主要路径

编程接口（Application Programming Interface, API)

- 应用程序设计人员是通过软件接口接触图形系统，这个接口就是应用编程接口(API)



图形API的构成

- 函数：定义生成一幅图像所需要的内容
  - 对象
  - 观察者
  - 光源
  - 材料属性

- 其它信息
  - 从鼠标和键盘等设备获取输入系统的能力

对象的定义

- 绝大多数API支持有限的基本几何对象，例如：
  - 点 points（零维对象）
  - 线段 line segments（一维对象）
  - 多边形 polygons（二维对象）
  - 某些曲线和曲面
  - 二次曲面 quadrics
  - 多项式参数曲面
- 所有基本形状都是通过空间中的位置或顶点(vertices)定义的。

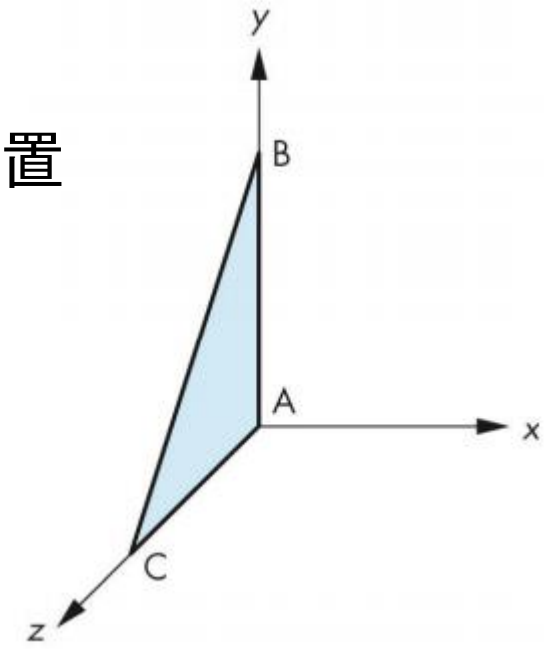
例如 （OpenGL）

对象类型

```
glBegin(GL_POLYGON)
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(0.0, 1.0, 0.0);
    glVertex3f(0.0, 0.0, 1.0);
glEnd( );
```

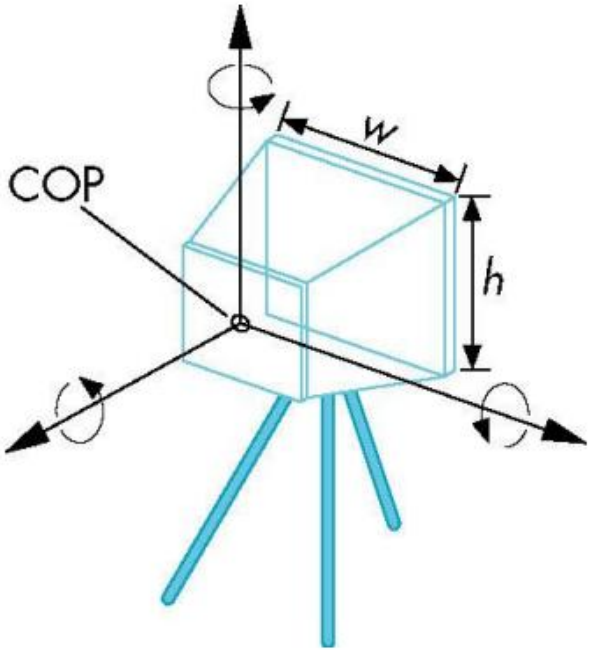
顶点位置

定义结束



照相机的指定

- 六个自由度 （6 degrees of freedom, DOF)
  - 镜头中心的位置，即投影中心(COP)
  - 方向
- 镜头、焦距
- 胶卷尺寸



光源与材质（light sources and materials)

- 光源类型 (light type)
  - 点光源与分布式光源
  - 聚光灯
  - 远光源与近光源
  - 光源的颜色属性
- 材料属性 (materials)
  - 吸收性：颜色属性
  - 反射性：漫反射、镜面

本节小结

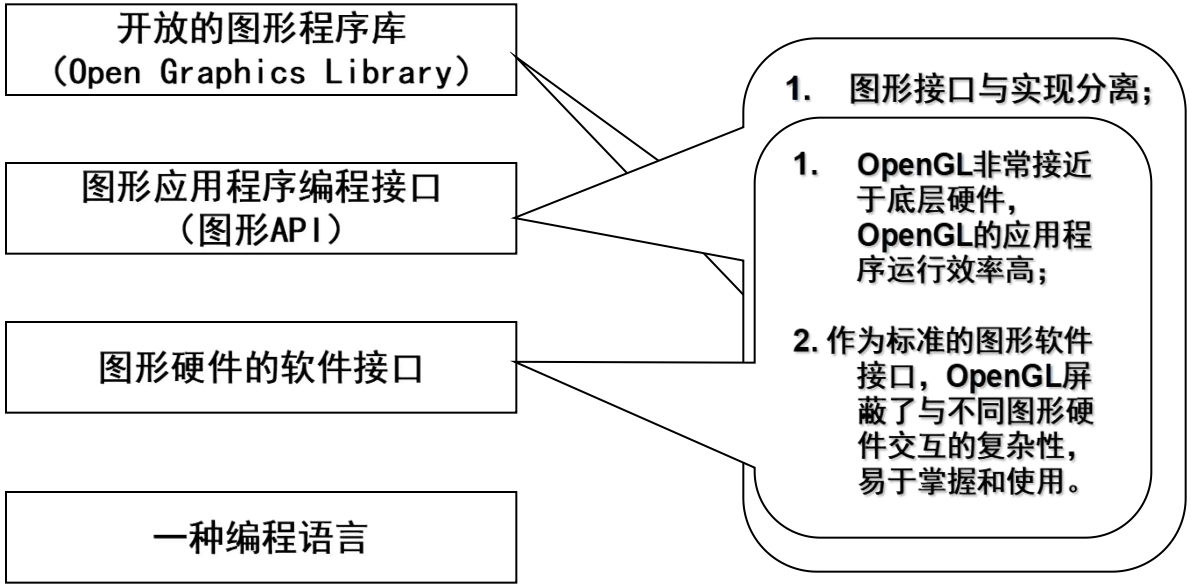
- 图形处理流水线体系结构
- 图形API的构成

本节内容

- 图形流水线介绍
  - The pipeline architecture of the graphics rendering procedures (图形渲染过程的体系结
  - Application Programming Interface （API）应用程序编程接口
- OpenGL简介
  - OpenGL是什么
  - OpenGL库的使用
- OpenGL完整程序

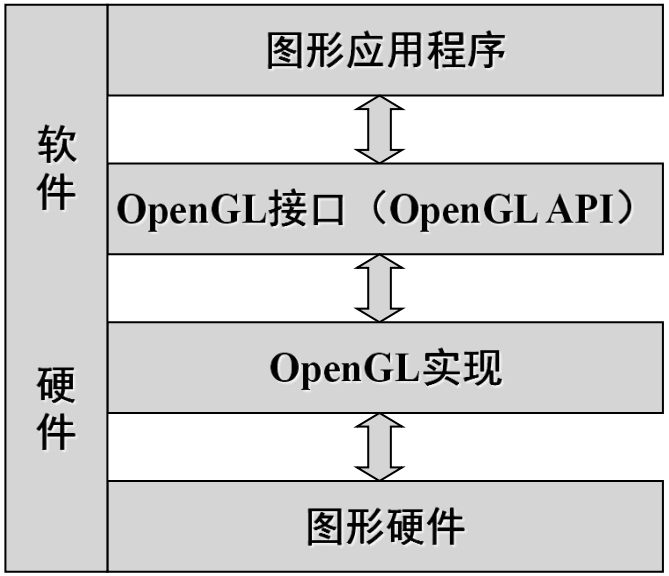
什么是OpenGL？——多种理解

- 如何理解OpenGL？



什么是OpenGL？——准确定义

- OpenGL——图形硬件的一种软件接口



什么是OpenGL？
——OpenGL接口

OpenGL核心API的功能

+

- 构建二维和三维几何物体，并控制这些物体按某种方式绘制到显示缓冲区中
- 提供了物体平移、旋转、缩放、光照、纹理、材质、像素、位图、文字、交互以及提高显示性能等方面的功能

−

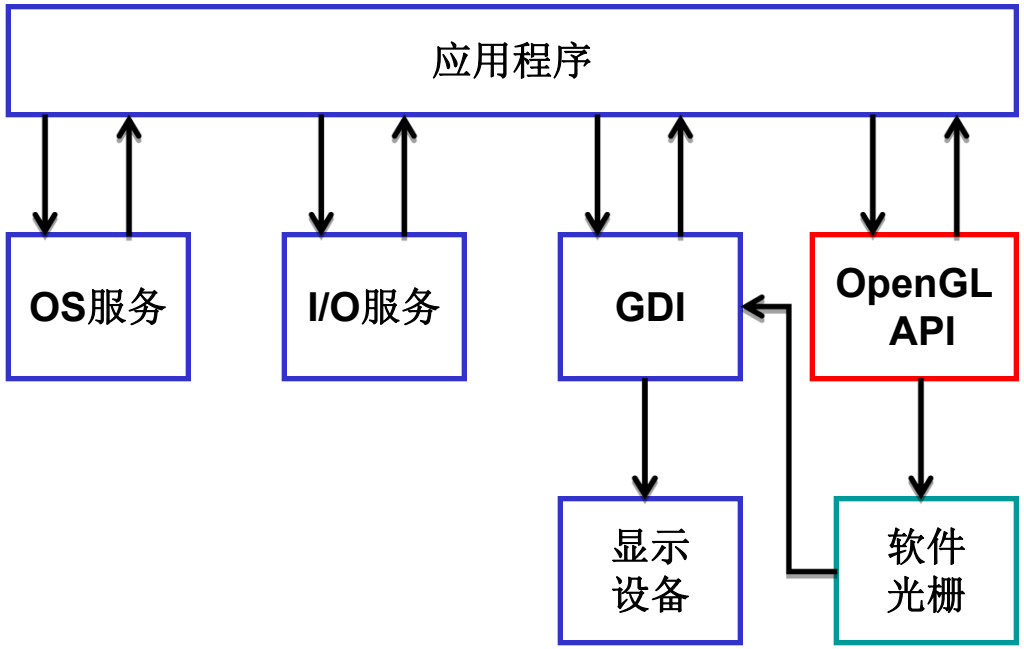
- 任何用于窗口管理、用户交互或文件I/O
- 直接描述复杂三维物体模型

+ 包括

− 不包括

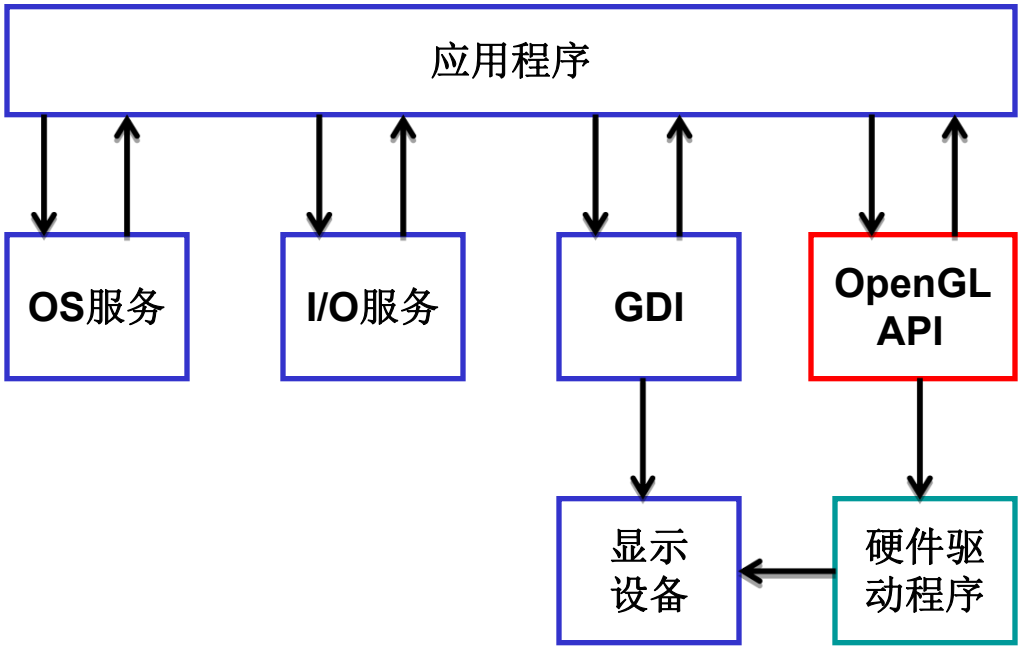
什么是OpenGL？
——OpenGL实现

- OpenGL实现方式——软件实现



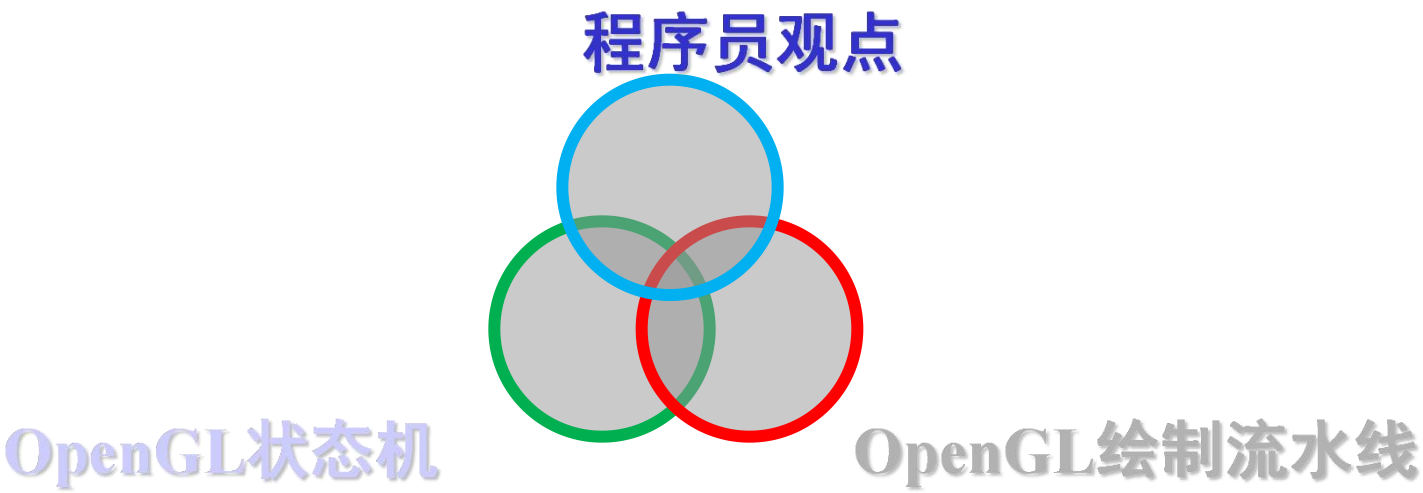
什么是OpenGL？
——OpenGL实现

- OpenGL实现方式——硬件实现



OpenGL的工作方式
——完整认识

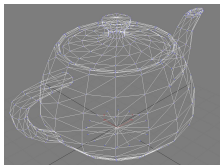
- 从三个视点出发揭示OpenGL的工作过程



OpenGL的工作方式
——程序员观点

- 图形应用程序的基本要素

指定绘制对象



描述对象属性

定义观察方式

OpenGL的工作方式
——状态机

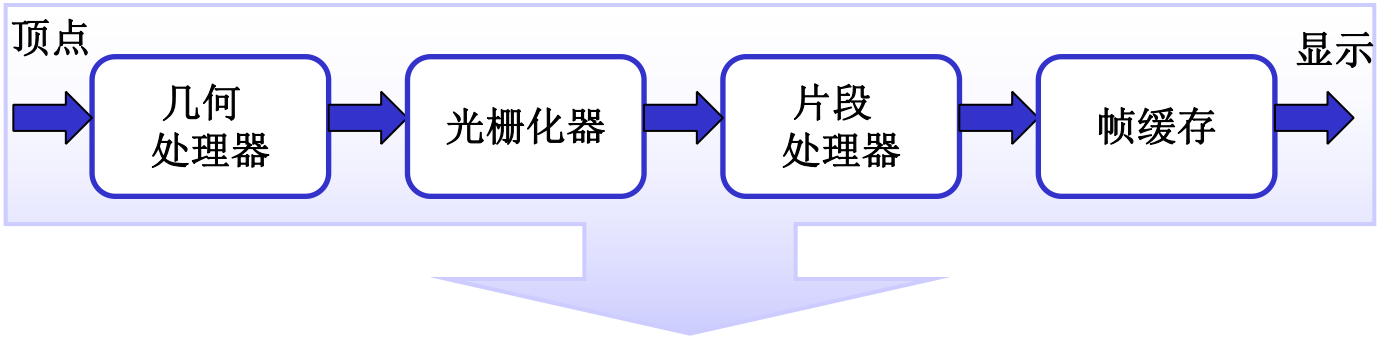
- OpenGL是一个具有输入和输出的状态机(State Machine)

OpenGL函数主要分为两类

- 指定输入对象的函数（几何对象的描述信息、离散对象）
- 修改OpenGL状态机的函数（修改颜色、观察条件、材质属性等状态）

OpenGL的工作方式
——流水线模型

- OpenGL内部基于流水线模型实现



流水线模型

- 与绝大多数图形硬件系统的底层工作方式接近
- 强调每个图元可独立确定自身的颜色和可见性
- 实现局部光照效果

认识OpenGL API——介绍

- OpenGL是API，而不是编程语言！
  - 标准的编程函数库
  - 遵循C语言调用约定
  - 官方绑定语言C/C++、Fortran，非官方绑定有Java、C#、Perl、Python等
- 使用OpenGL编写图形应用程序
  - 宿主语言编写基本程序
  - 调用OpenGL 函数库进行图形绘制
  - 调用其它图形程序包（Windows GDI等）

认识OpenGL API——函数库

OpenGL实用工具包

- 函数以“glut”为前缀
- 主要功能包括窗口创建与管理、用户交互（鼠标、键盘等），并提供一种基于事件驱动的窗口程序运行架构；
- GLUT库独立于操作系统和窗口系统。

OpenGL实用库

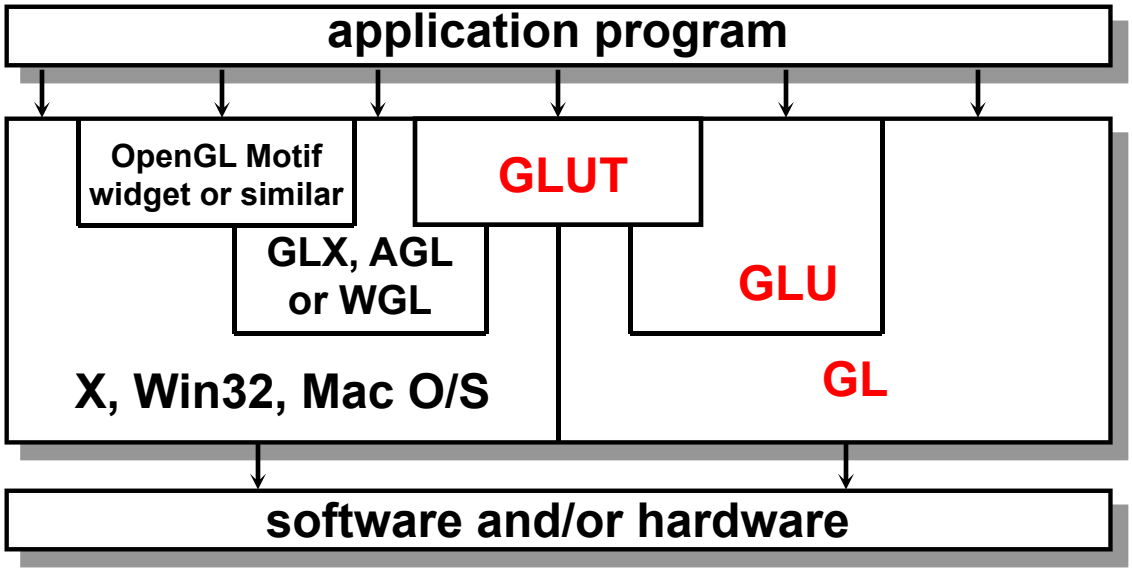
- 函数以“glu”为前缀；
- 主要功能包括绘制二次曲面、NURBS曲线曲面、复杂多边形以及纹理、矩阵管理等；
- 实用库中的所有函数全都使用OpenGL核心库函数来编写的，因此在使用上和OpenGL核心库的使用是完全相同的。

OpenGL核心库

- 函数以“gl”为前缀
- 主要功能包括三维物体描述、平移、旋转、缩放、光照、纹理、材质、像素、位图、文字处理等；
- OpenGL 提供的所有功能都可以使用核心库函数来实现，并且对于不同的软件和硬件平台，这些函数的使用完全相同。

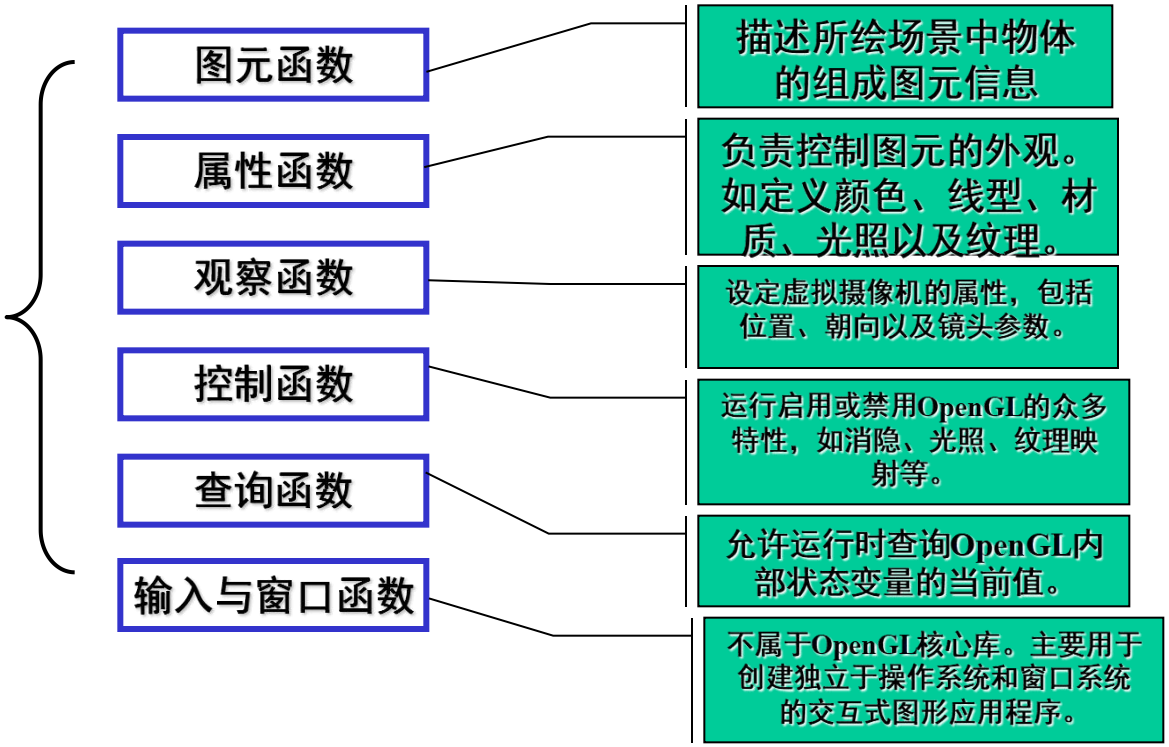
认识OpenGL API——函数库

- OpenGL核心库（GL库）、OpenGL实用库（GLU库）及OpenGL实用工具包（GLUT库）的关系



认识OpenGL API
——按功能分类

- OpenGL 函数库由200多个函数组成，依据功能可分类



认识OpenGL API
——编程规范

- 常量：GL_COLOR_BUFFER_BIT, GL_POLYGON, …
- 变量类型：GLint, GLshort, GLfloat, GLdouble, …

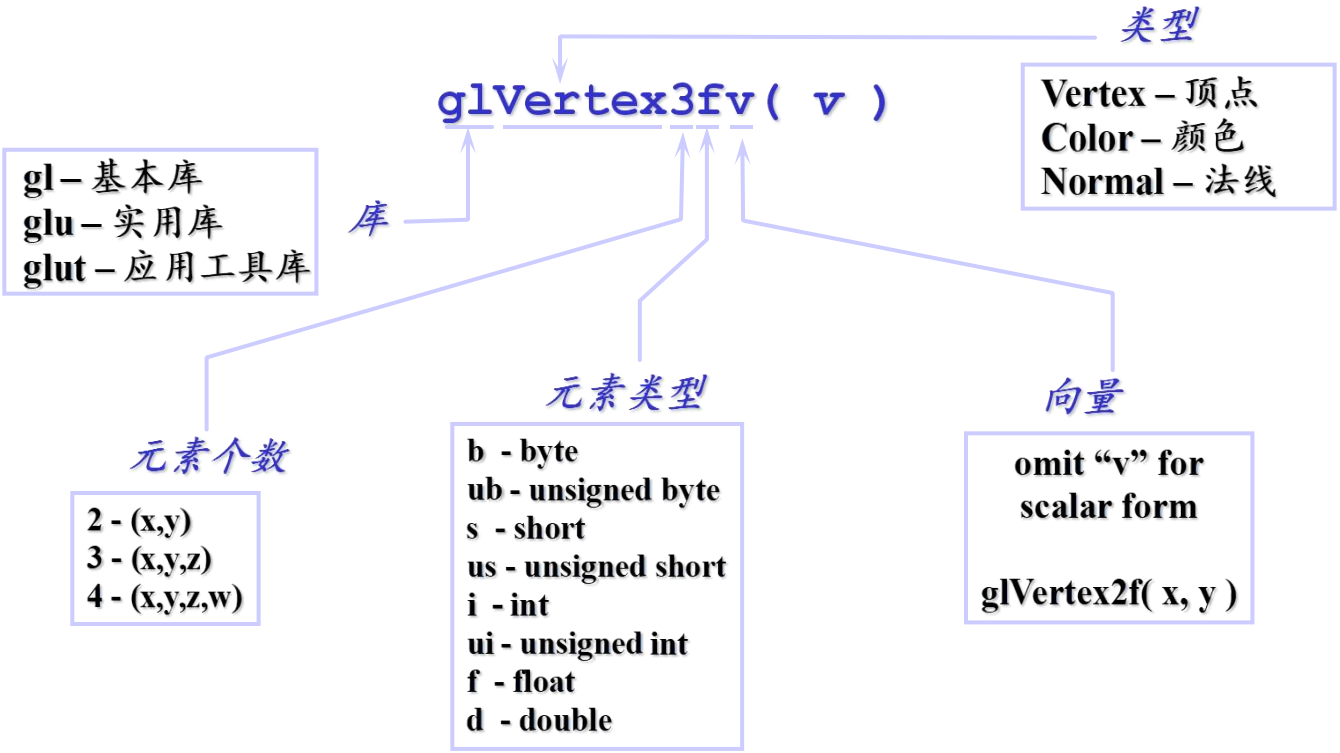
OpenGL类型定义	相应C语言类型	后缀定义
GLbyte	char	b
GLubyte, GLboolean	unsigned char	ub
GLshort	short	s
GLushort	unsigned short	us
GLint, GLsizei	int	i
GLuint, GLenum, GLbifield	unsigned int	ui
GLfloat, GLclampf	float	f
GLdouble, GLclampd	double	d

认识OpenGL API
——编程规范

- 函数名约定

```
glColor3f(1.0f,1.0f,1.0f)
glVertex2i(1,3)
glVertex3f(1.0f,3.0f,2.0f)
GLfloat v[]={1.0,3.0,2.0}
glVertex3fv( v )
```

认识OpenGL API
——编程规范



本节内容

- OpenGL简介
  - OpenGL是什么
  - OpenGL库的使用
- OpenGL完整程序
- OpenGL三维程序

OpenGL环境配置-头文件设置

- Windows+Visual Studio2010
- 从网上下载GLUT包，并解压
  - 下载地址为：GLUT下载地址：http://www.xmission.com/~nate/glut/glut-3.7.6-bin.zip
- 把GLUT头文件glut.h复制到Windows SDK的Include\gl子目录：
  - (对于Visual Studio 2010: "C:\Program Files\Microsoft SDKs\Windows\v7.0A\include\gl") (对于64位系统，该目录在"C:\Program Files (x86)"下)
- 说明：上述目录是Visual Studio查找头文件的默认目录。当然你也可以放在任意你感觉合适的位置，但是需要对Visual Studio进行设置，是它知道你存放的路径。

OpenGL环境配置—库设置

- 把GLUT库文件glut32.lib复制到Windows SDK的Lib子目录：
  - (对于Visual Studio 2010: "C:\Program Files\Microsoft SDKs\Windows\v7.0A\Lib") (该目录下有"OpenGL32"库文件)
  - (对于64位系统，该目录在"C:\Program Files (x86)"下)

- 说明：上述目录是Visual Studio查找库文件的默认目录。当然你也可以放在任意你感觉合适的位置，但是需要对Visual Studio进行设置，是它知道你存放的路径。

OpenGL环境配置-DLL设置

- 把GLUT的dll文件glut32.dll复制到系统目录：
- (对于32位系统: "C:\Windows\System32")
- (对于64位系统: "C:\Windows\SysWOW64")
- (该目录下有"opengl32.dll"文件)

OpenGL环境配置-IDE设置

- 经过上述设置之后，我们将OpenGL中需要用到的头文件、库文件、和动态链接库文件都放到了开发环境默认查找的目录，今后编写程序时，开发环境就能到上述位置找到所需的库文件。
- 此外，还需要告诉开发环境，应该使用什么库文件。（虽然放到了指定位置，但是目前尚未告诉开发环境应该使用什么库）

OpenGL环境配置-IDE设置

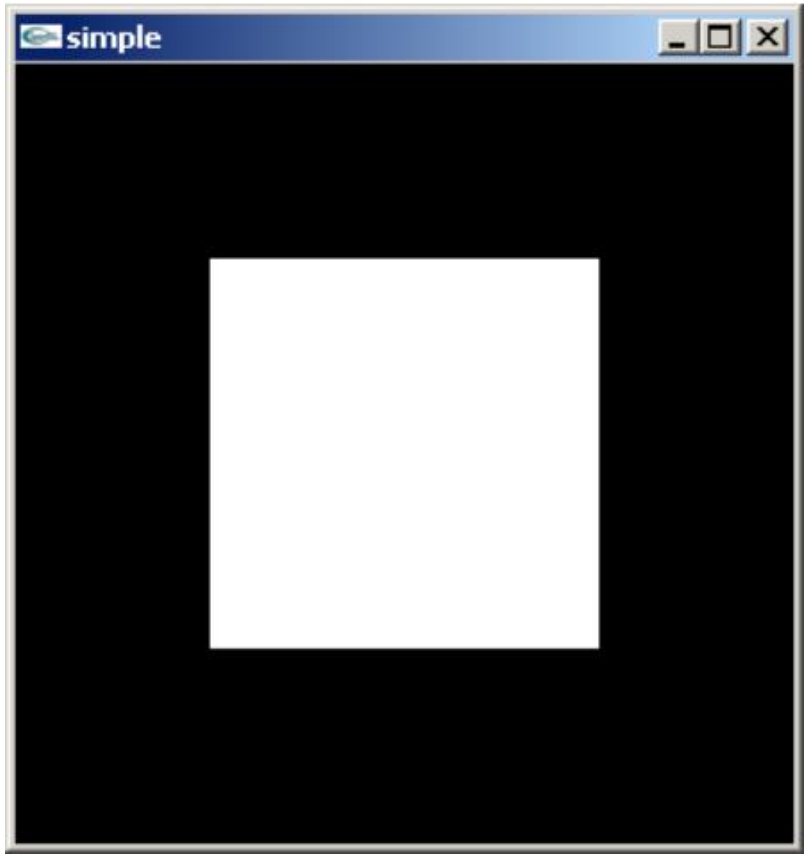
- 在 配置属性 -> 链接器 -> 输入 -> 附加依赖项 (Configuration Properties -> Linker -> Input -> Additional Dependencies)
  - 填入opengl32.lib glu32.lib glut32.lib（这里用分号或者回车分隔）

本节内容

- 图形流水线介绍
- OpenGL简介
- OpenGL完整程序
  - 第一个OpenGL程序解析
  - OpenGL编程基础知识

第一个OpenGL程序解析

- 在黑色背景上画一个白色矩形



simple.c

```
#include <GL/glut.h>
void display(){
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_QUAD);
                glVertex2d(-0.5, -0.5);
                glVertex2d(-0.5, 0.5);
                glVertex2d(0.5, 0.5);
                glVertex2d(0.5, -0.5);
        glEnd();
        glFlush();
}

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutCreateWindow(“simple");
        glutDisplayFunc(display);
        glutMainLoop();
}
```

主要函数

- 初始化GLUT
  - **void glutInit(int * argc, char ** argv)**
- 创建窗口：窗口标题为title
  - **int glutCreateWindow(char * title)**
- 注册显示回调函数：窗口需要重绘时调用func
  - **void glutDisplayFunc(void (*func)(void))**
- 事件处理循环： main函数最后一条语句
  - **void glutMainLoop()**
- 清空帧缓冲区
  - **void glClear(GLbitfield mask)**
- 图元定义： mode可取GL_POINTS、 GL_LINES、 GL_POLYGON等
  - **void glBegin(Glenum mode)** // 开始**mode**型图元定义
  - **void glEnd()** // 结束顶点序列
- 强制执行OpenGL命令
  - **void flush()**

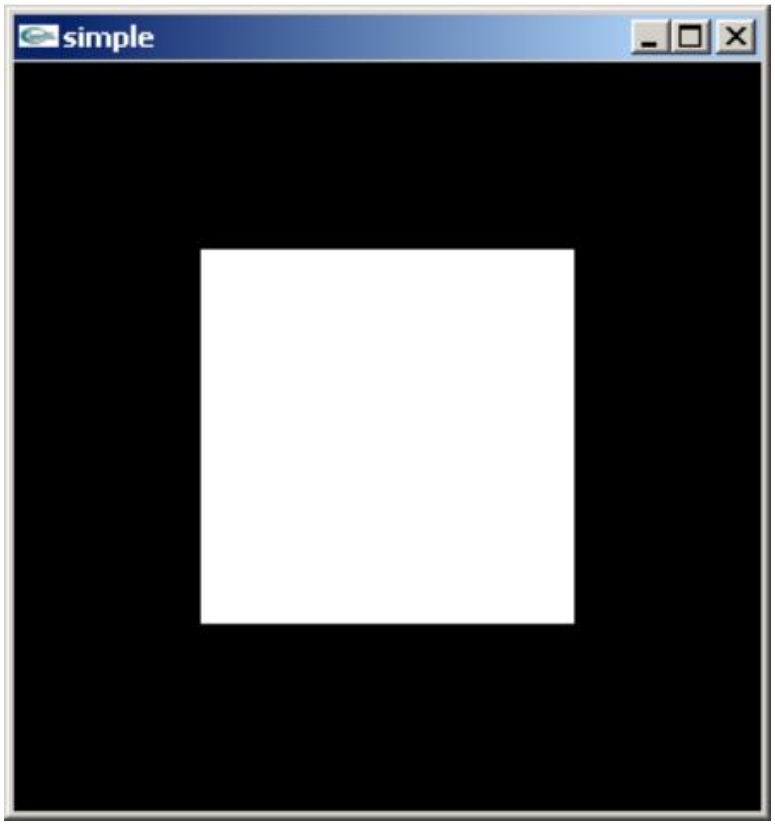
```
#include <GL/glut.h>
void display(){
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_QUAD);
                glVertex2d(-0.5, -0.5);
                glVertex2d(-0.5, 0.5);
                glVertex2d(0.5, 0.5);
                glVertex2d(0.5, -0.5);
        glEnd();
        glFlush();
}

int main(int argc, char* argv[]) {
        glutInit(&argc, argv);
        glutCreateWindow("simple");
        glutDisplayFunc(display);
        glutMainLoop();
}
```

事件循环的概念

- 注意在程序中定义了一个显示回调函数(display callback): display
  - 每个GLUT程序都必须有一个显示回调函数
  - 只要OpenGL确定显示内容要被刷新时，显示回调函数就会被调用：例如，当窗口被打开的时候
  - main函数是以程序进入事件循环做为结束

默认值

- simple.c非常简单
  - 大量使用状态变量的默认值
    - 视图
    - 颜色
    - 窗口参数
  - 以后的程序将直接改变一些默认值



本节内容

- OpenGL简介
  - OpenGL是什么
  - OpenGL库的使用
- OpenGL完整程序
  - 第一个OpenGL程序解析
  - OpenGL编程基础知识
    - OpenGL程序的结构
    - 控制函数
    - 视图
    - OpenGL的图元
    - 属性

本节内容

- OpenGL简介
  - OpenGL是什么
  - OpenGL库的使用
- OpenGL完整程序
  - 第一个OpenGL程序解析
  - OpenGL编程基础知识
    - OpenGL程序的结构
    - 控制函数
    - 视图
    - OpenGL的图元
    - 属性

OpenGL程序结构

- 绝大多数OpenGL程序具有类似的结构，包含下述函数

main函数

- 定义回调函数
- 打开一个或多个具有指定属性的窗口
- 进入事件循环（最后一条可执行语句）

init函数 设置初始状态变量

- 视图
- 属性

回调函数

- 显示函数 display()
- 输入和窗口函数

OpenGL程序结构

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

int main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(300,300);
        glutInitWindowPosition(0,0);
        glutCreateWindow("简单示例“);
        glutDisplayFunc(display);
        init();
        glutMainLoop();
}
```

glut.h包含了gl.h和glu.h

窗口属性

设置回调函数

设置初始状态

一切就绪，进入消息循环

本节内容

- 图形流水线介绍
- OpenGL简介
- OpenGL完整程序
  - 第一个OpenGL程序解析
  - OpenGL编程基础知识
    - OpenGL程序的结构
    - 控制函数
    - 视图
    - OpenGL的图元
    - 属性

OpenGL控制函数

- 控制函数
  - 与窗口系统通信,例如:glFlush等
  - 处理程序执行期间发生的错误
- GLUT库函数
  - 窗口管理
  - 事件处理循环
  - 回调函数机制

什么是窗口

- 窗口：显示器上的一块矩形区域
- 窗口内的位置用窗口坐标来指定，单位是像素
  - 科学和工程中，左下角是原点(0,0)
  - 光栅显示器按照从上到下，从左到右的顺序进行扫描，所以左上角是原点
  - OpenGL命令假定原点在左下角
  - 窗口系统返回的信息（例如鼠标位置）假定原点在左上角

GLUT窗口管理

- 初始化GLUT， 在调用其他GLUT函数前调用
  void glutInit(int * argc, char ** argv)
- 设置窗口的初始宽度和高度，单位为像素，缺省300x300
  void glutInitWindowSize(int width, int height)
- 窗口左上角相对于屏幕左上角的位置，单位为像素，缺省(0,0)
  void glutInitWindowPosition(int x, int y)
- 设置窗口的显示模式
  void glutInitDisplayMode(unsigned int mode)
  – 颜色模型： GL_RGB/GL_RGBA – RGB颜色(默认)； GL_INDEX – 索引颜色
  – 缓冲区类型： GL_SINGLE – 单缓冲区(默认)； GL_DOUBLE – 双缓冲
  – 属性选项按逻辑或组合在一起
- 创建窗口，标题为**title**。调用**glutMainLoop()**之前，窗口不会显示
  int glutCreateWindow(char * title)

GLUT事件处理

- GLUT事件处理循环
  void glutMainLoop()
  – 进入GLUT事件处理循环，当有事件发生时，调用相应的回调函数；否则，处于等待状态
  – **main()**函数是以程序进入事件循环做为结束

- 每个GLUT程序都必须有一个显式回调函数
  void glutDisplayFunc(void(*func)(void))
  – 窗口首次被打开
  – 窗口移动
  – **glutPostRedisplay**：显式地调用

GLUT回调函数

- GLUT使用回调函数机制来进行事件处理
  - 窗口重绘 glutDisplayFunc()
  - 窗口改变大小 glutReshapeFunc()
  - 键盘输入 glutKeyboardFunc()
  - 鼠标按键 glutMouseFunc()
  - 鼠标移动 glutMotionFunc()
  - 空闲函数 glutIdleFunc()

本节内容

- 图形流水线介绍
- OpenGL简介
- OpenGL完整程序
  - 第一个OpenGL程序解析
  - OpenGL编程基础知识
    - OpenGL程序的结构
    - 控制函数
    - 视图
    - OpenGL的图元
    - 属性

视图(Viewing)

- 首先来看一下一段典型的init代码

```
void init()
{
        glClearColor(0.0, 0.0, 0.0, 1.0);
        glColor3f(1.0, 1.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
}
```

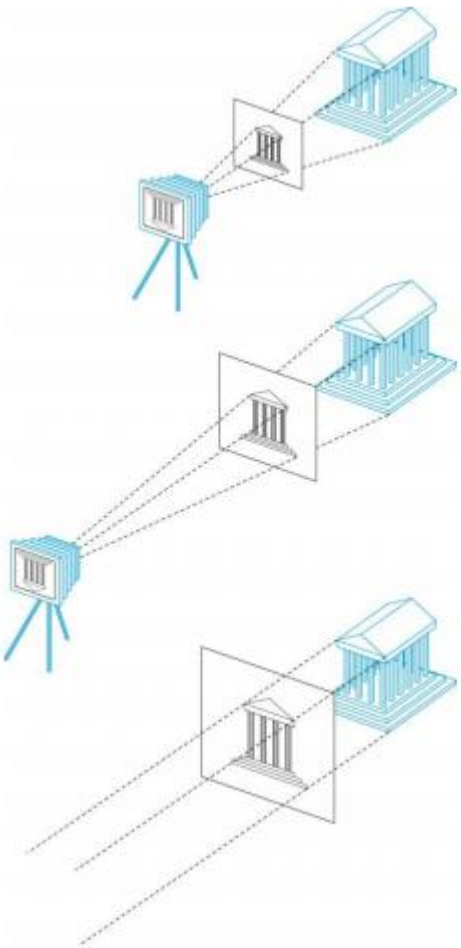
设置背景色为黑色

设置前景色（画笔）为白色

设置视景体
（相机的投影矩阵）

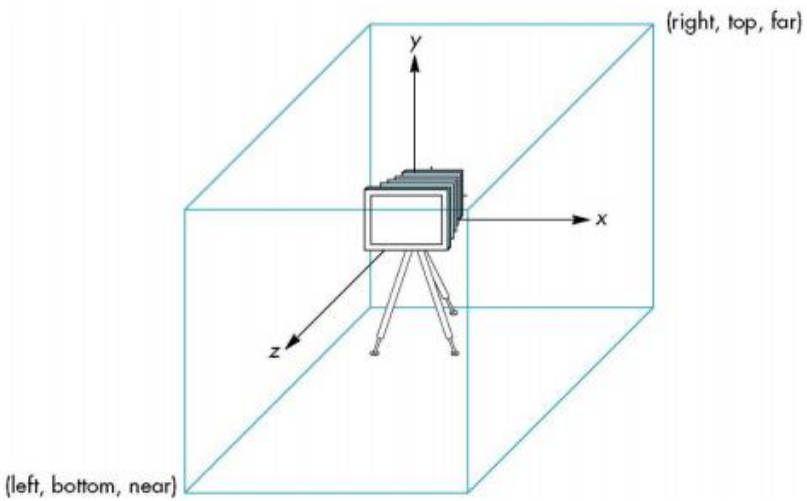
视图(Viewing)

- 虚拟照相机模型：对象与照相机独立
  - 应用程序只需关心对象和照相机的参数设置
- OpenGL默认视图：正交投影
  - 镜头焦距无限大，无限远离对象
  - 极限情形下，投影线彼此平行，投影中心变为投影方向



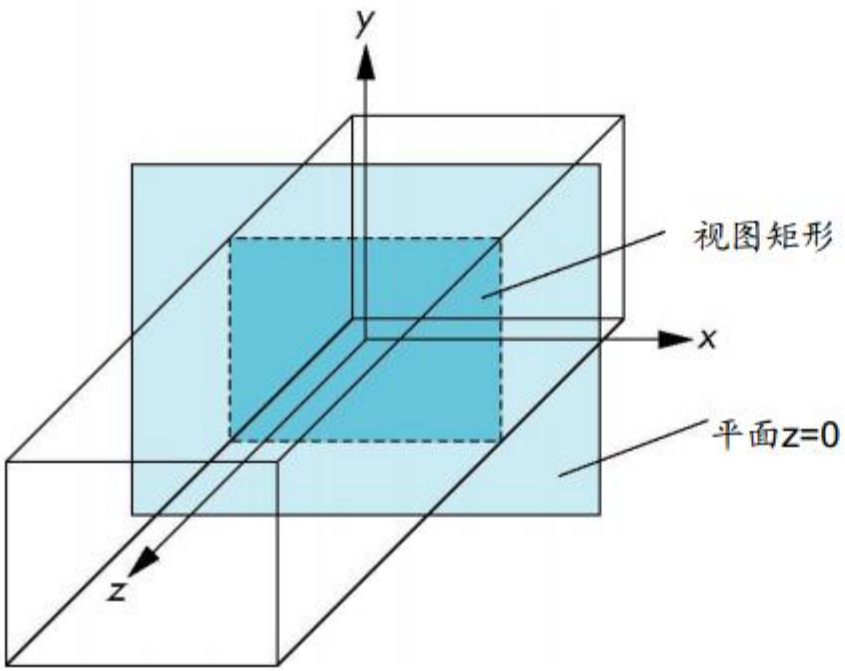
OpenGL中的照相机

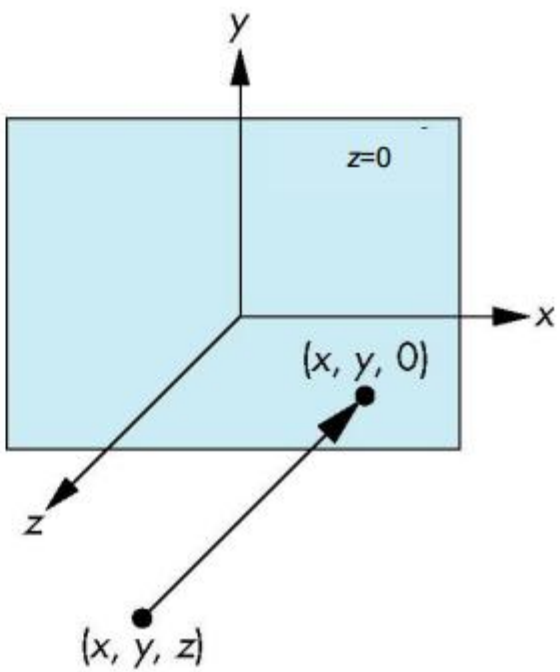
- 照相机被放置在对象坐标系的原点，指向z轴的负方向
  - 默认的视景体是一个中心在原点，边长为2的立方体
  - 正交投影可以对位于照相机后面的对象成像



正交视图

- 在正交视图（OpenGL默认视图）中，点沿着z轴投影到z=0的平面上





变换与视图

- 在OpenGL中投影是利用投影矩阵乘法(变换)进行的

- 由于存在一个变换函数系列，因此必须先设置矩阵类型

  glMatrixMode(GL_PROJECTION);

- 变换函数是累积在一起的，因此需要从单位阵开始，然后把它改变为一个定义正交投影视景体的投影矩阵

  glLoadIdentity();
  glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);

二维与三维视图

- 在glOrtho(左, 右, 底, 顶, 近, 远)中的近与远是相对于照相机的距离而言的
  void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,GLdouble near, GLdouble far)
- 二维顶点命令把所有的顶点放在z=0的平面上
- 如果应用程序处于二维状态，那么可以使用下述函数设置正交视景体：
  void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)
- 对于二维情形，视景体或裁剪体退化为裁剪窗口或观察矩形

矩阵模式

- 变换矩阵是系统的状态变量
  - 模型-视图矩阵(model-view)
  - 投影矩阵(projection)
  - 初始值都是单位矩阵
- 每次只能对一个矩阵进行操作
- 通过设置矩阵模式来选择对哪个矩阵进行操作
  - 矩阵模式也是系统状态变量

矩阵模式

- 默认矩阵模式是对模-视矩阵进行操作

- 设置二维观察矩形：

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,50.0,0.0,50.0);
glMatrixMode(GL_MODELVIEW);
```

是何作用？
哪些函数可以修改此矩阵

- 最好每次改变矩阵模式后都返回约定的矩阵模式，例如模-视模式

本节内容

- 图形流水线介绍
- OpenGL简介
- OpenGL完整程序
  - 第一个OpenGL程序解析
  - OpenGL编程基础知识
    - OpenGL程序的结构
    - 控制函数
    - 视图
    - OpenGL的图元
    - 属性

图元

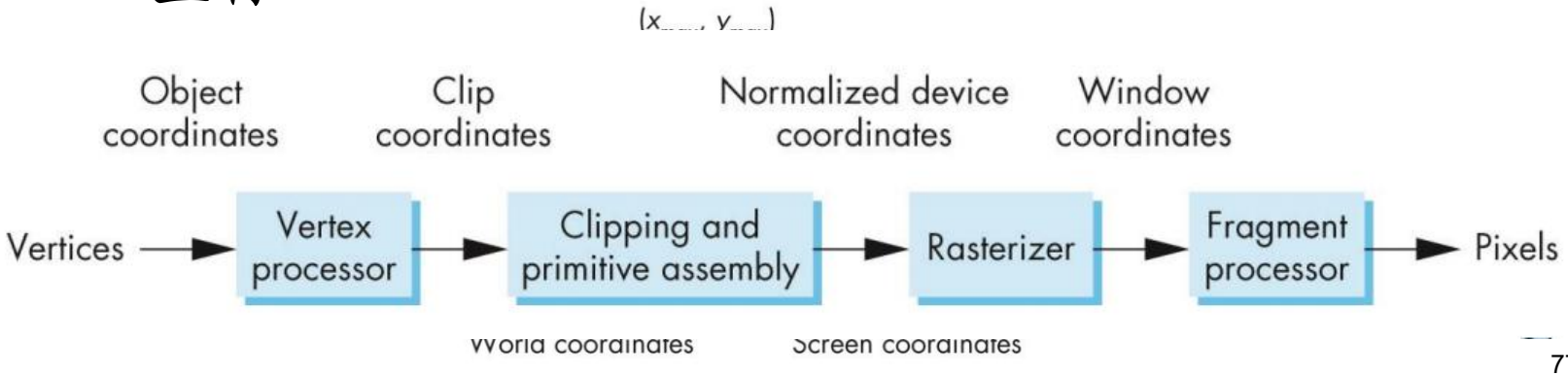
- 先看一个简单的display函数

```
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glBegin(GL_POLYGON);
                glVertex2d(-0.5,-0.5);
                glVertex2d(-0.5,0.5);
                glVertex2d(0.5,0.5);
                glVertex2d(0.5,-0.5);
        glEnd();
        glFlush();
}
```

图元函数

图元—坐标系

- 最初的图形系统要求用户直接按照显示设备的单位来确定所有的信息
- 设备无关图形学
  - 顶点坐标在对象坐标系或世界坐标系中定义
  - 顶点坐标最终映射成窗口坐标或屏幕坐标



图元

- 最小系统观点
  - 线段、多边形和文本(字符串)，可硬件高效生成
- 复杂图元观点
  - 圆、曲线、曲面和实体等复杂图
- OpenGL
  - 核心库：点、线、多边形
  - GLU：二次曲面、NURBS曲面
  - GLUT：笔划字符、点阵字符

定义几何图元

- 图元由下面语句定义:

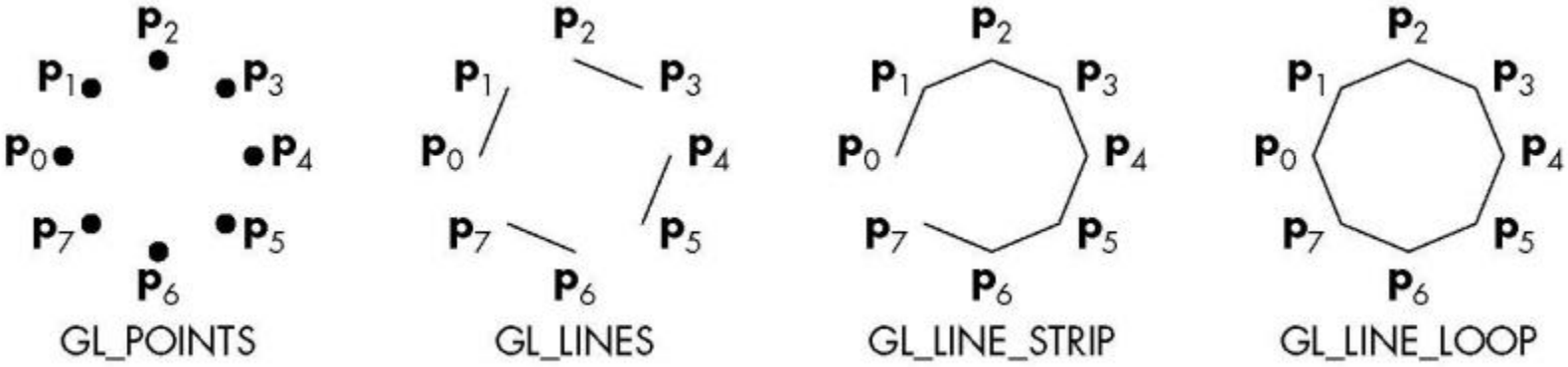
  glBegin(*primType*);
  glEnd();

  - *primType* 决定顶点如何组合成图元

```
glBegin( primType );
for ( i = 0; i < n; ++i ) {
        glColor3f( red[i], green[i], blue[i] );
        glVertex3fv( coords[i] );
}
glEnd();
```

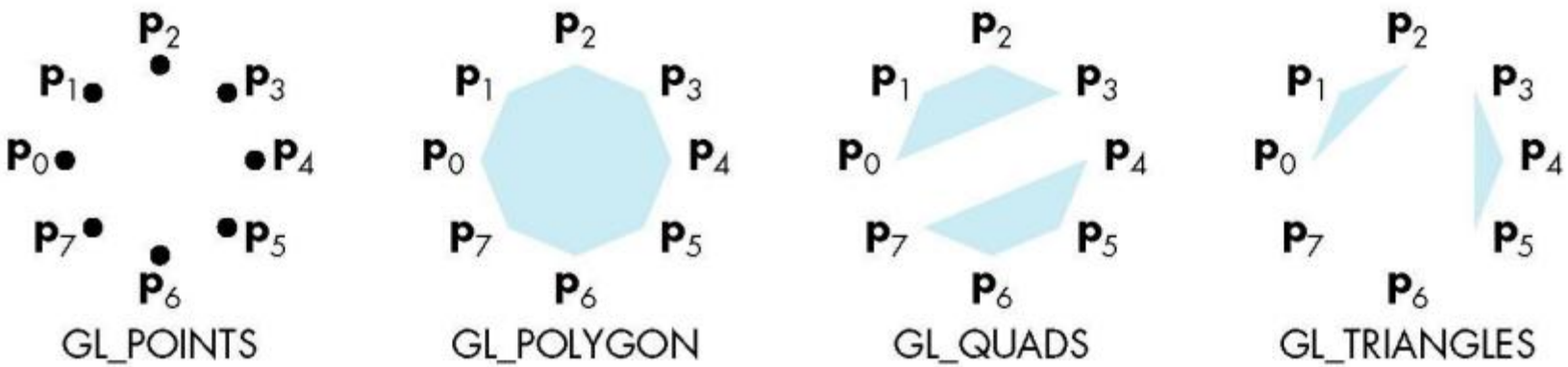
OpenGL的几何图元

- 点： GL_POINTS
- 线段： GL_LINES
- 开折线： GL_LINE_STRIP
- 封闭折线： GL_LINE_LOOP



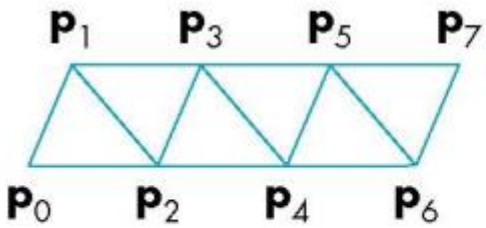
OpenGL的几何图元

- 多边形： GL_POLYGON
- 三角形： GL_TRIANGLES
- 四边形： GL_QUADS

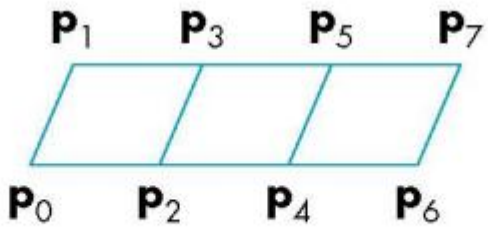


OpenGL的几何图元

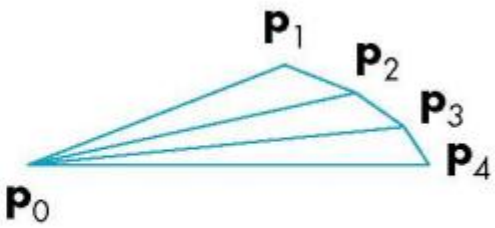
- 三角形带： GL_TRIANGLE_STRIP
- 四边形带： GL_QUAD_STRIP
- 三角形扇： GL_TRIANGLE_FAN



GL_TRIANGLE_STRIP



GL_QUAD_STRIP



GL_TRIANGLE_FAN

绘制复杂物体

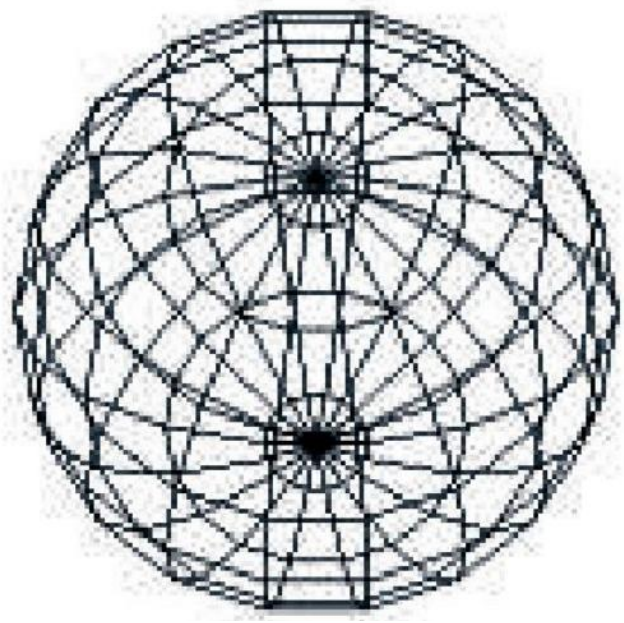
- 用多边形近似（往往三角形）
- 单位球面的参数表达：

  $x(\theta,\varphi)=\sin\theta\cos\varphi$

  $y(\theta,\varphi)=\cos\theta\cos\varphi$

  $z(\theta,\varphi)=\sin\varphi$

  其中 $0\leq\theta\leq 2\pi$, $0\leq\varphi\leq\pi$
- 中间四边形带->三角形带
- 两极处三角形扇



GLU/GLUT提供曲面函数

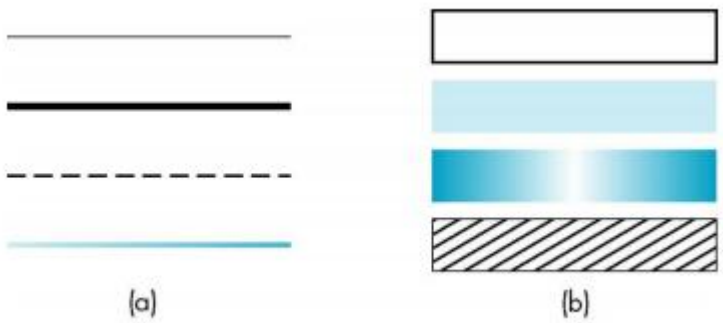
- 图元都是由顶点来定义的。
- 使用已有的图元来近似曲线和曲面
  - 正n边形来近似圆
  - 三角形和四边形来近似球面
  - 多边形网格来近似曲面
- 从数学定义出发，编写图形函数来生成
  - 二次曲面
  - NURBS曲线/曲面
- OpenGL中， GLU库和GLUT库提供了对常见曲面的一些近似

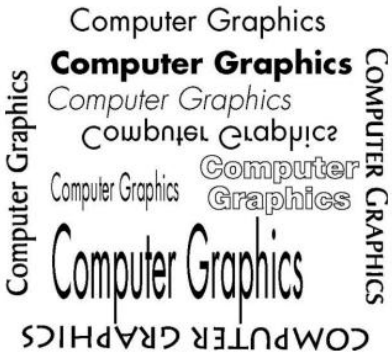
本节内容

- 图形流水线介绍
- OpenGL简介
- OpenGL完整程序
  - 第一个OpenGL程序解析
  - OpenGL编程基础知识
    - OpenGL程序的结构
    - 控制函数
    - 视图
    - OpenGL的图元
    - 属性

属性（状态）

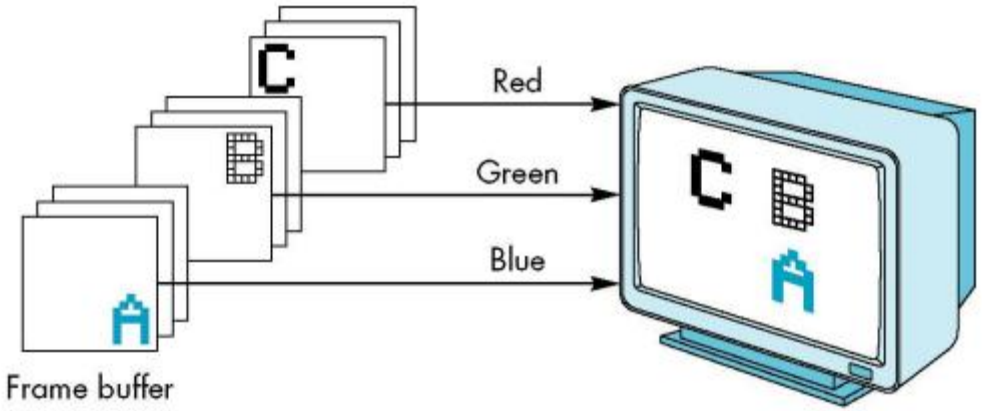
- 属性(attribute)决定图元将如何绘制，是OpenGL状态的一部分
  - 点：颜色、大小
  - 线段：颜色、宽度、模式（实线、虚线）
  - 多边形：绘制模式、填充模式
  - 笔划文本：字体、字号、方向





颜色

- 颜色的每个分量在帧缓冲区中是分开存储的
- 在缓冲区中通常每个分量占用8位字节
- 颜色值用float表示的变化范围是从0.0(无)到1.0(全部), 而用unsigned byte表示的变化范围是从0到255



RGBA颜色

- 第四个分量（A或者α）是不透明度或透明度
  - 完全透明： A=0.0
  - 完全不透明： A=1.0
- 和RGB值一样存储在帧缓冲区中
- 用于图像融合
- 例子：设置清屏颜色（背景色），默认黑色：
  glClearColor(0.0, 0.0, 0.0, 1.0);

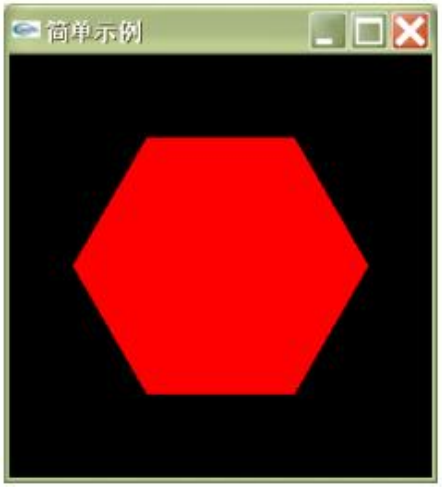
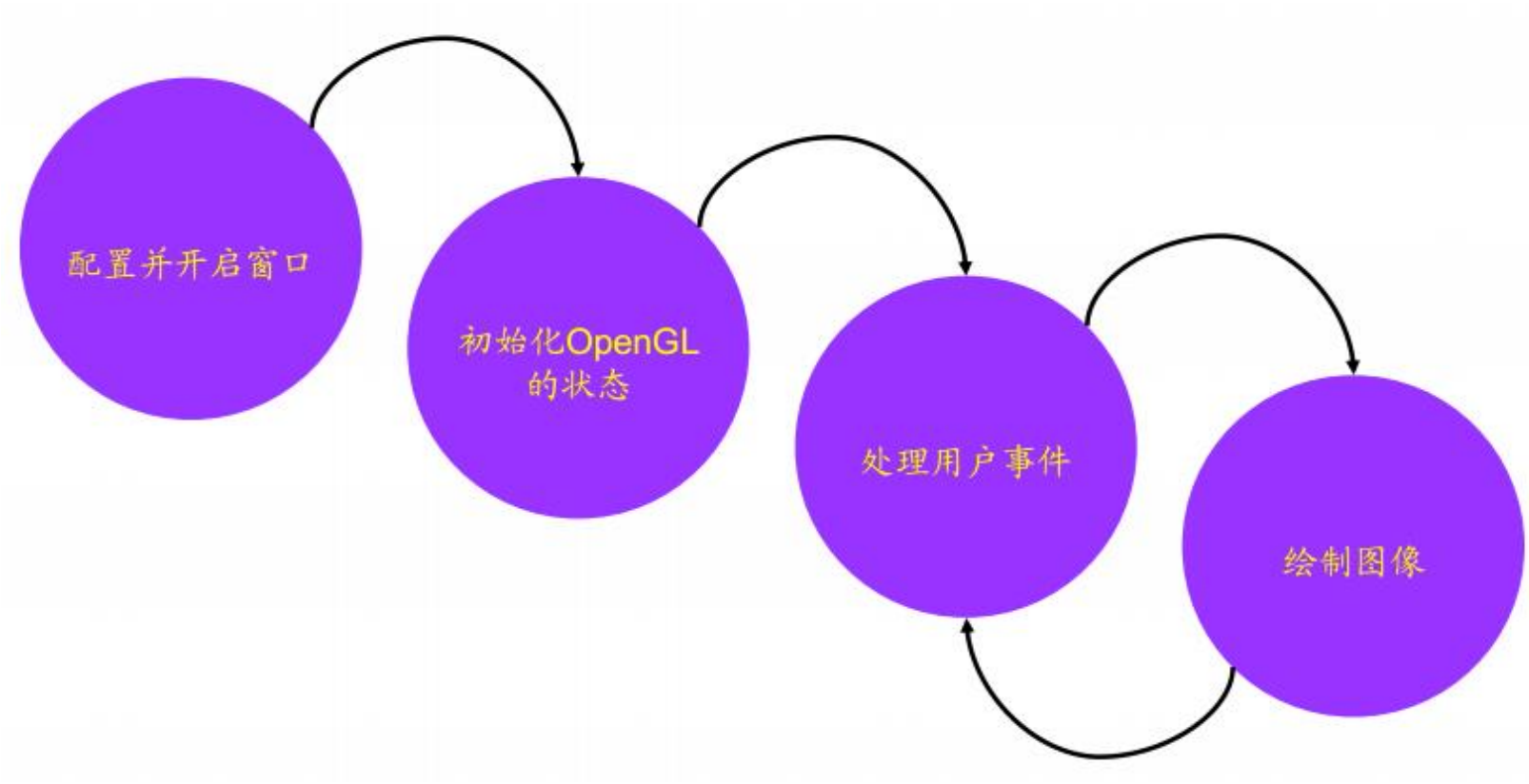
设置颜色

- 由glColor*设置的颜色成为状态的一部分，后续构造过程将使用这一颜色，直至它被修改为止
  - 颜色与其它属性不是对象的一部分，但是在渲染对象时要把这些属性赋给对象
- 可以按下述过程创建具有不同颜色的顶点

```
glColor(…);
glVertex(…);
glColor(…);
glVertex(…);
```

为多边形内部上色（着色模型）

- 默认状态是平滑着色
  – OpenGL根据多边形顶点的颜色插值出来内部的颜色
- 另外一种状态是平面着色
  – 第一个顶点的颜色确定填充颜色
- 设置着色模型：
  void glShadeModel(Glenum mode)
  – 平滑模式： GL_SMOOTH
  – 平面模式： GL_FLAT



再回顾：OpenGL程序的一般结构