计算机图形学
Computer Graphics

陈中贵
chenzhonggui@xmu.edu.cn
http://graphics.xmu.edu.cn/~zgchen

Graphics@XMU

*课件来源于中科大黄章进副教授讲义

第二章 第二节

完整的程序

主要内容

- OpenGL程序的结构
- 控制函数
- 视图
- OpenGL的图元
- 属性

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()**：
    - 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()**：设置状态变量
    - 视图
    - 属性
  - 回调
    - 显示函数 **display**()
    - 输入和窗口函数

对simple.c进行修改

- 在新版本中，会得到同样的输出，但是常用的具有默认值的相应状态值都通过函数调用显式地指定
- 特别地，设置了
  - 颜色
  - 视图条件
  - 窗口属性

main.c

```
#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();
}
```

这自动包含了gl.h

定义窗口属性

显示回调函数

设置OpenGL状态

进入事件循环

控制函数

- 控制函数
  - 与窗口系统通信
  - 初始化程序
  - 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

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，窗口不会被显示

```
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**被显式地调用

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()**

init()函数

```
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);
}
```

清除色（背景色）为黑色

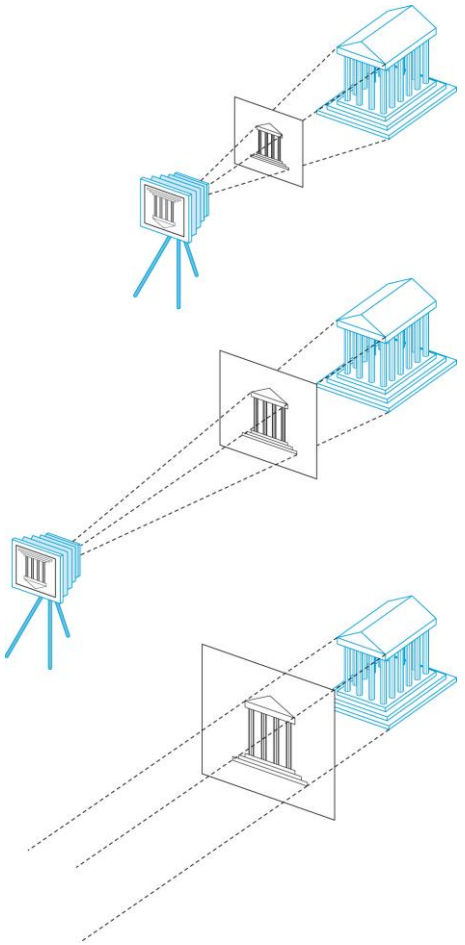
不透明窗口

矩形填充以白色

视景体

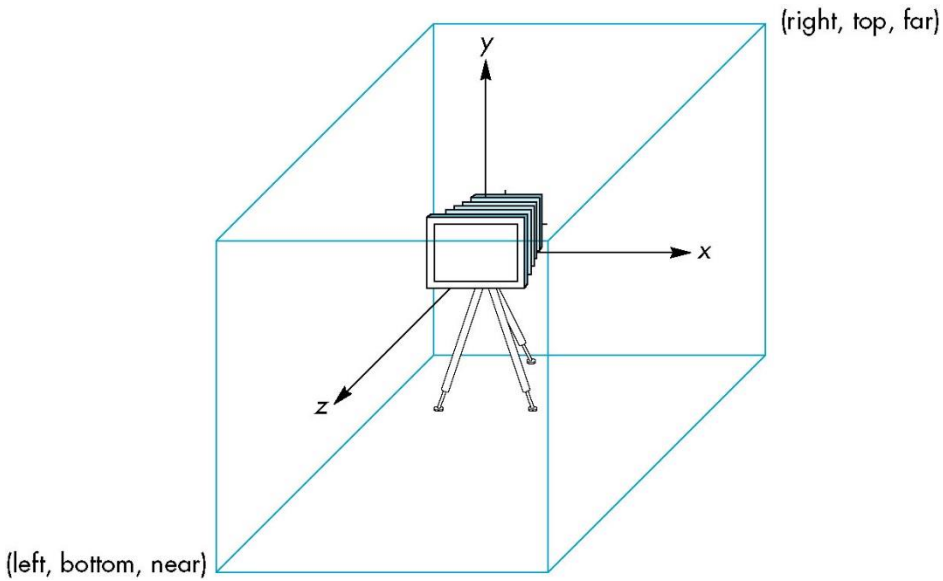
视图(Viewing)

- 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



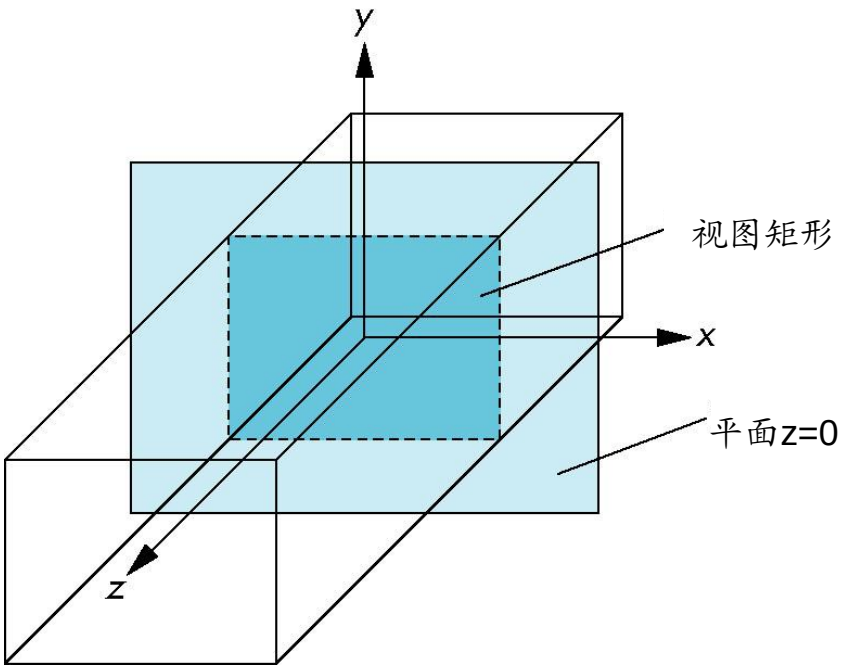
OpenGL中的照相机

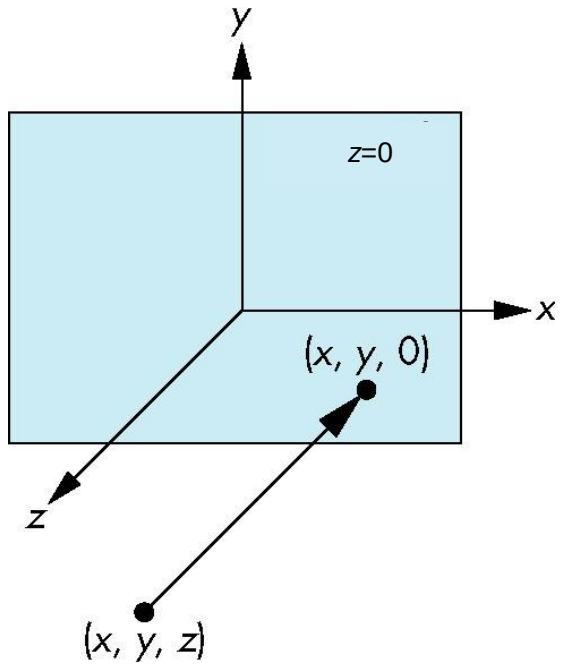
- 照相机被放置在对象坐标系的原点，指向z轴的负方向
- 默认的视景体是一个中心在原点，边长为2的立方体
- 正交投影可以对位于照相机后面的对象成像



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
- 由于只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);
```

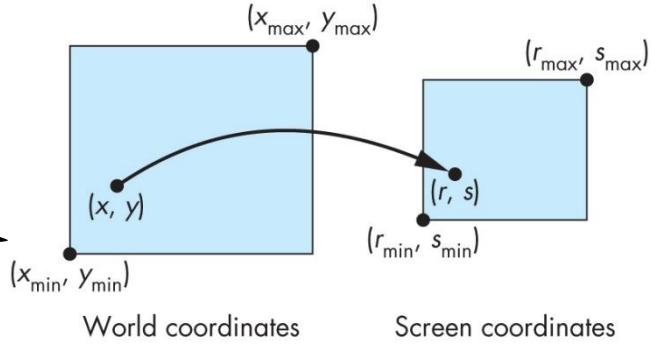
- 最好每次改变矩阵模式后都返回约定的矩阵模式，例如模-视模式

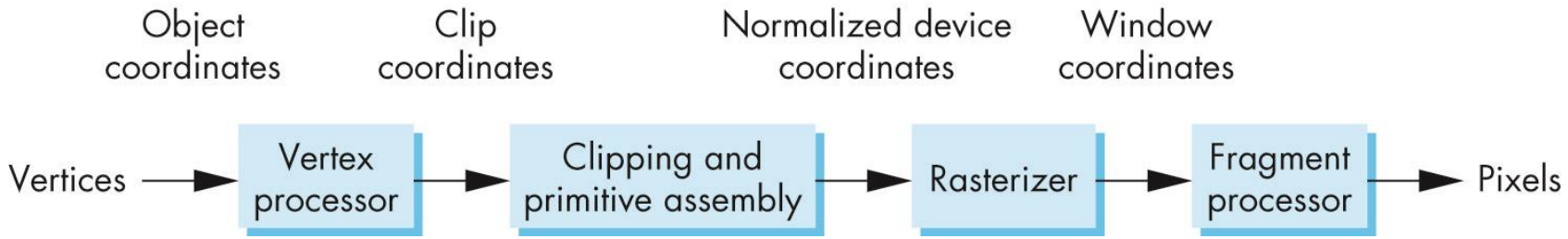
display()函数

```
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();
}
```

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



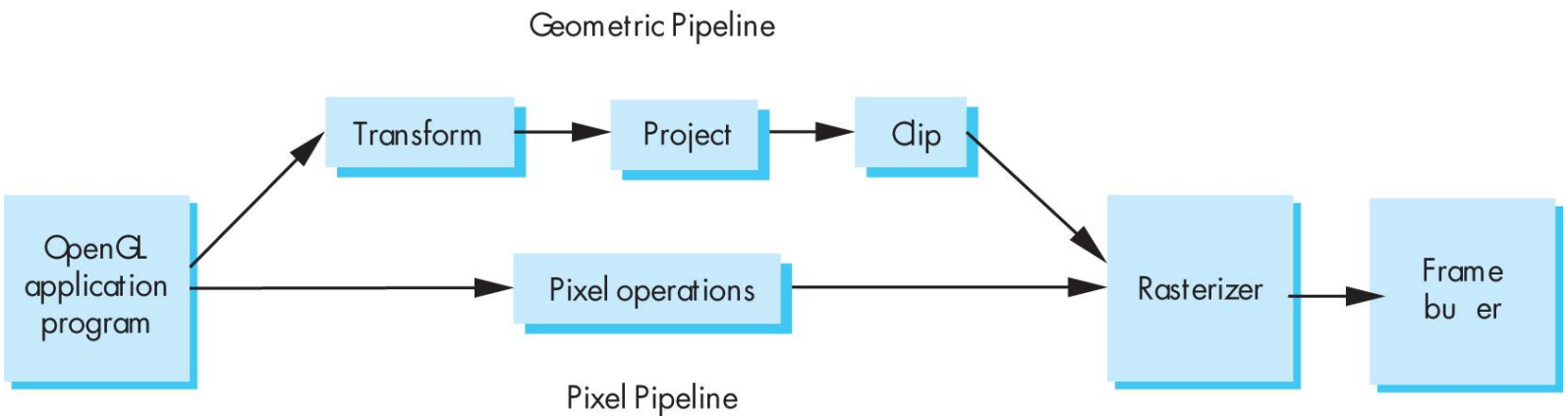


图元

- 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元
- 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

图元

- 几何图元：点、线段、多边形、曲线和曲面
  - 几何流水线：变换、裁剪、光栅化
- 图像图元/光栅图元：像素阵列
  - 像素流水线



定义几何图元

- 图元由下面语句定义:

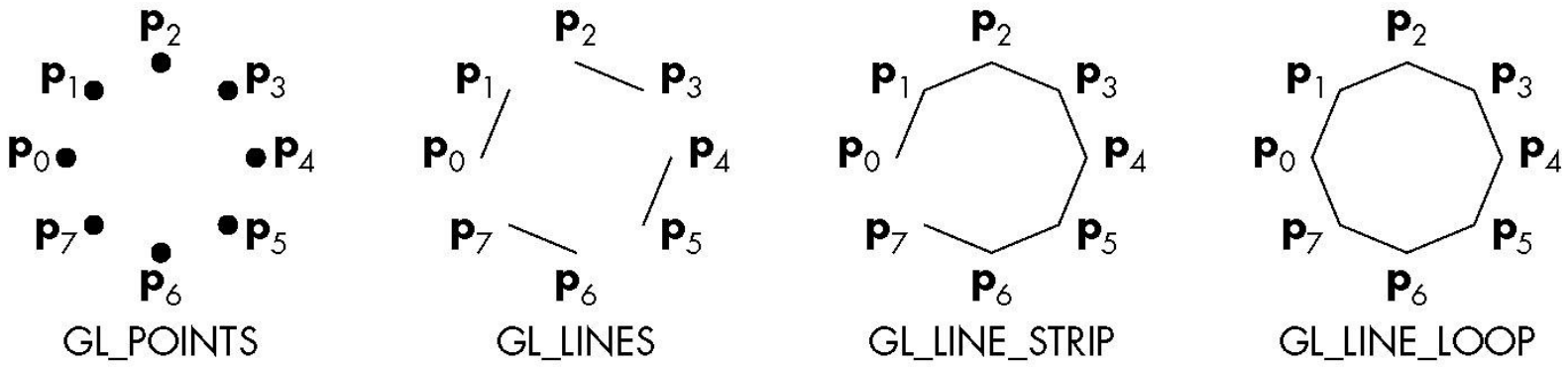
```
glBegin( primType );
glEnd();
```

  - *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();
```

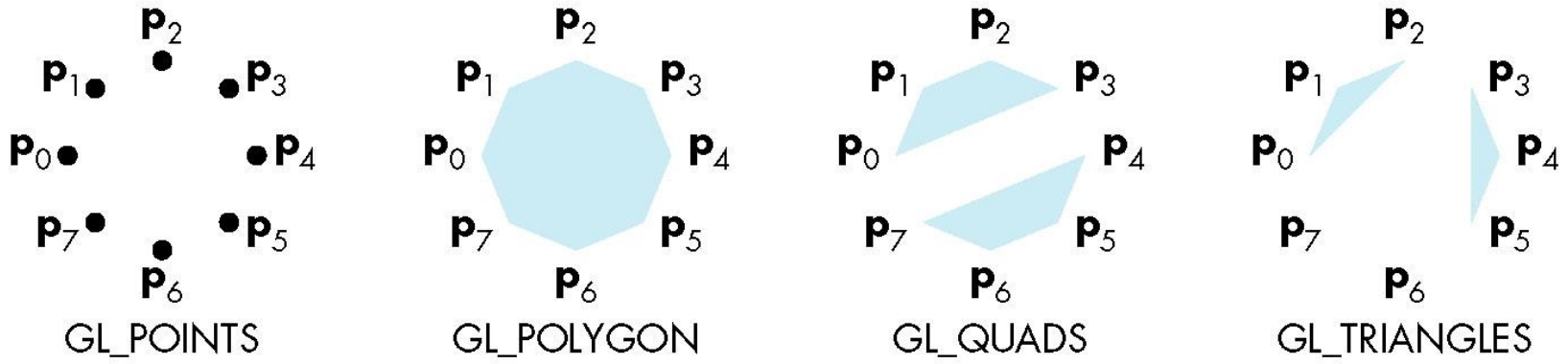
OpenGL的几何图元

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



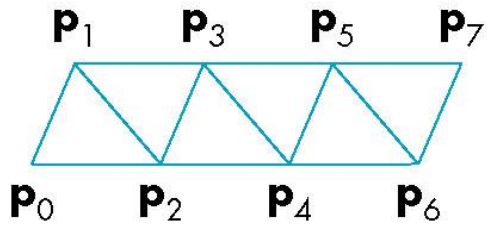
OpenGL的几何图元

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

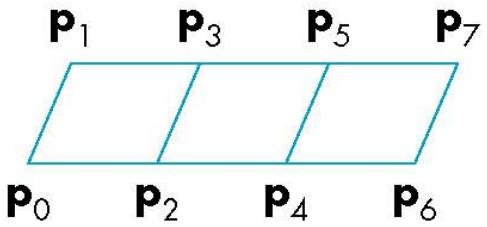


OpenGL的几何图元

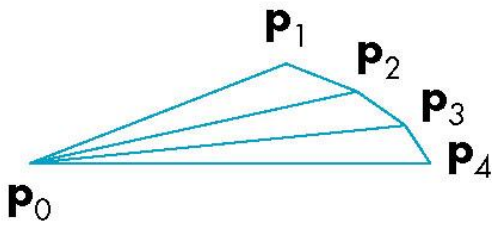
- 三角形带：GL_TRIANGLE_STRIP
- 四边形带：GL_QUAD_STRIP
- 三角形扇：GL_TRIANGLE_FAN



GL_TRIANGLE_STRIP



GL_QUAD_STRIP

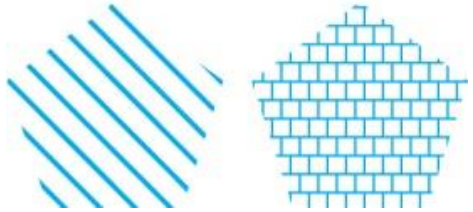


GL_TRIANGLE_FAN

多边形

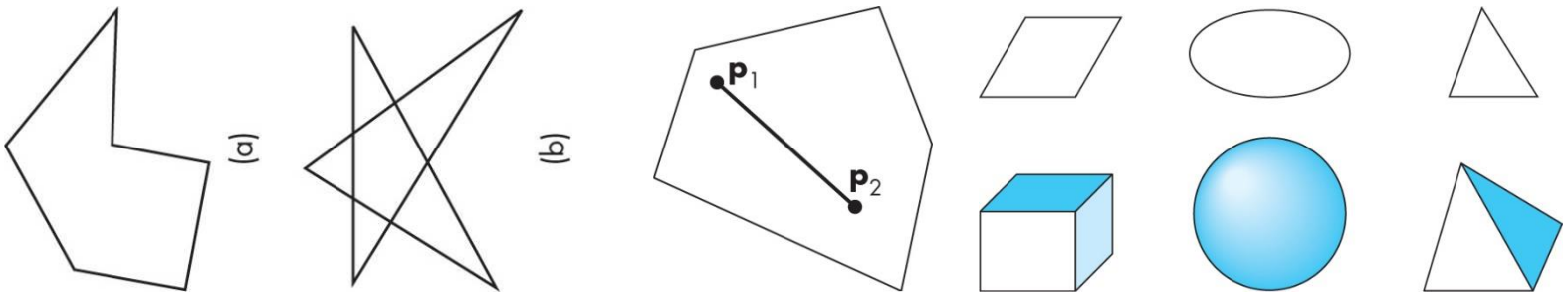
- 多边形的显示方式：绘制边、用单色或模式填充

- 确保正确显示的多边形
  - 简单的
  - 凸的
  - 平面的



定义多边形的限制条件

- 简单性：边除顶点外不相交
- 凸性：对于多边形中任意两点，连接这两点的线段完全在多边形内
- 平面性：所有顶点在同一平面上



- 三角形满足上述所有限制条件

多边形测试

- 用户自己确保上述条件满足
  - 如果不满足上述要求，OpenGL也会有输出，只是结果看起来与期望的不同
- 多边形的简单性和凸性概念上简单，但检测的代价很高
- 大部分图形系统要求应用程序来完成检验
- 新版的OpenGL只绘制三角形

球面的多边形近似

- 单位球面的参数表达:

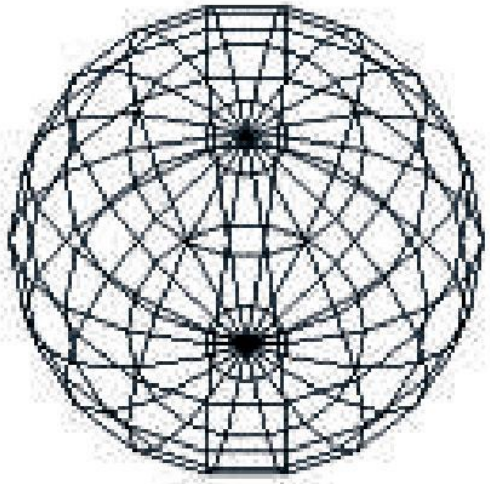
  $x(\theta,\varphi)=\sin\theta\cos\varphi$

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

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

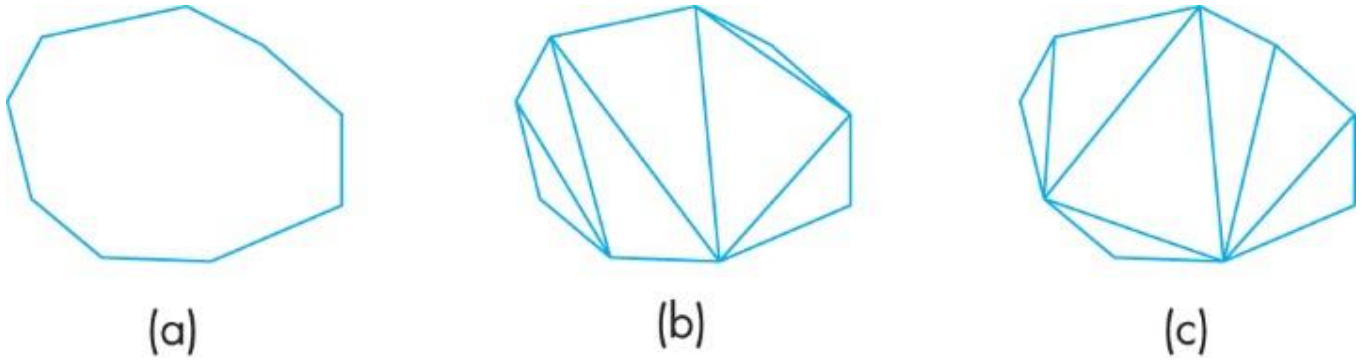
其中$0\le\theta\le 2\pi$, $0\le\varphi\le\pi$

- 中间四边形带->三角形带
- 两极处三角形扇



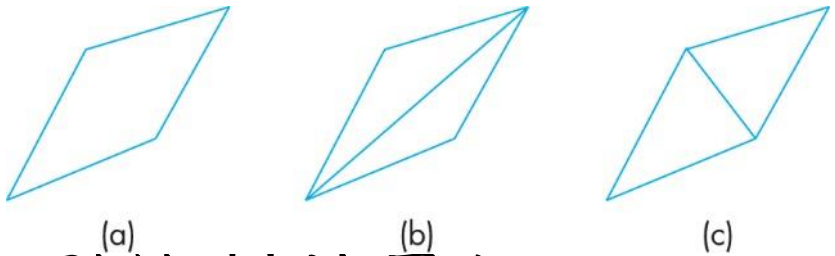
三角剖分

- 对一般多边形，应用程序必须把它剖分成一组三角形
- OpenGL 4.1引入tessellator



三角形的质量

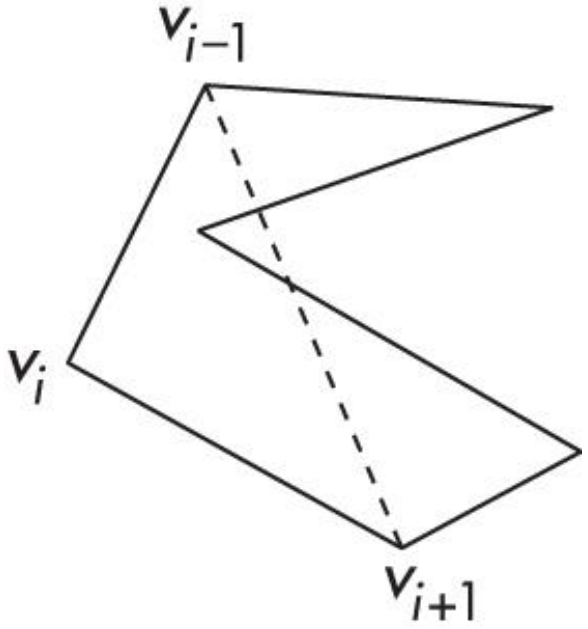
- 狭长的三角形会导致差的绘制效果

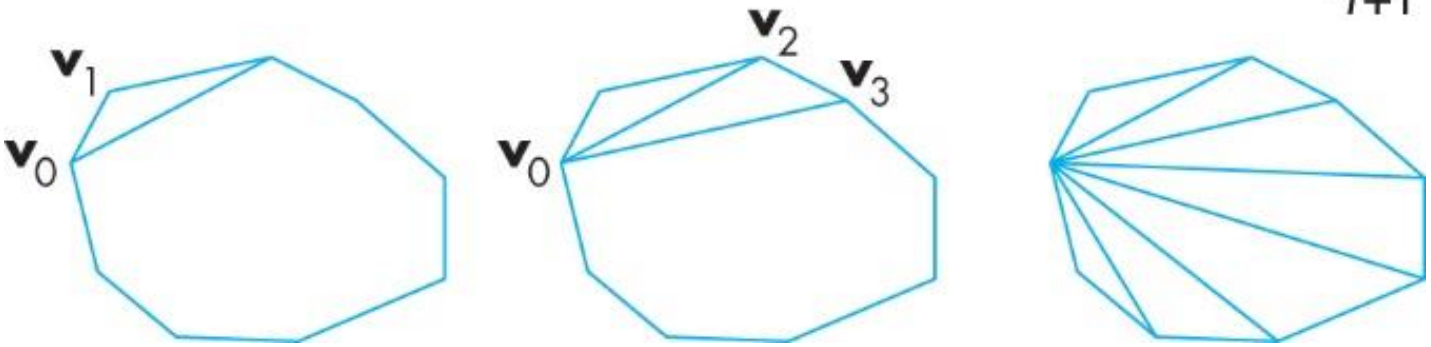


- 等边三角形绘制效果好
- 最大化最小角
- 对无结构点集，Delaunay三角剖分算法能得到一组最优的三角形

凸多边形的三角剖分

- 递归的方式进行剖分





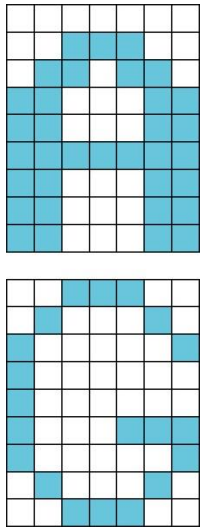
曲线与曲面

- 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

文本

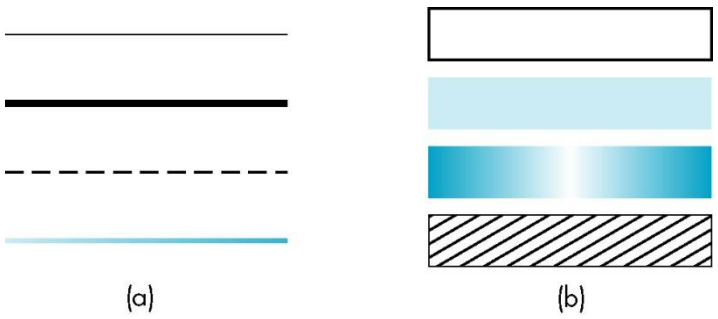
- 笔划文本：矢量字体，字符轮廓是线段或曲线
  - 几何图元，变换和观察操作
- 光栅文本：点阵字体，字符定义为0和1的矩形阵列
  - 光栅图元
- GLUT库提供预定义好的笔划字符集和光栅字符集

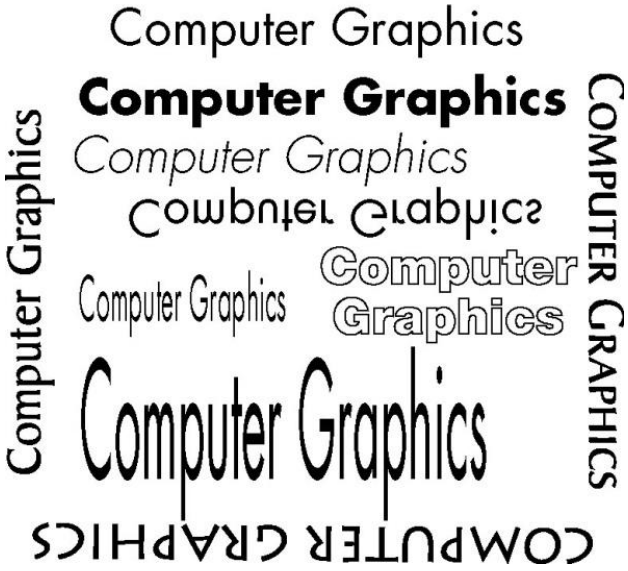
Computer Graphics



属性

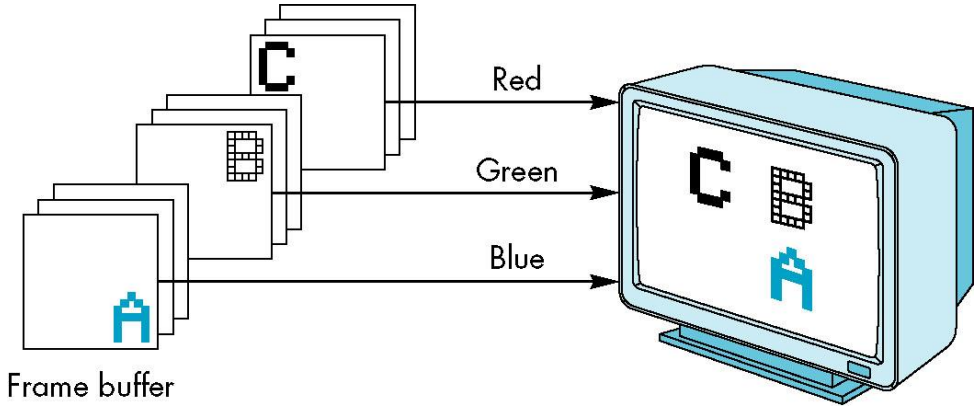
- 属性(attribute)决定图元将如何绘制，OpenGL状态的一部分
  - 点：颜色、大小
  - 线段：颜色、宽度、模式（实线、虚线）
  - 多边形：绘制模式、填充模式
  - 笔划文本：字体、字号、方向





RGB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

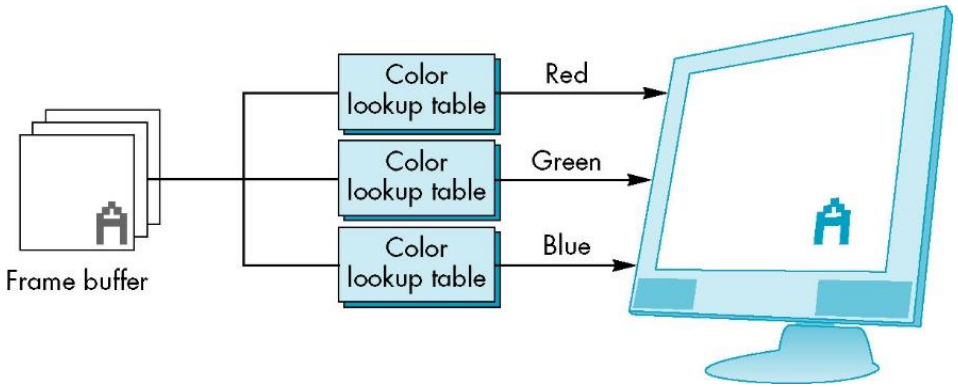
- 第四个分量（A或者α）是不透明度或透明度
  - 完全透明：A=0.0
  - 完全不透明：A=1.0
- 和RGB值一样存储在帧缓冲区中
- 用于图像融合
- 设置清屏颜色（背景色），默认黑色:

```
glClearColor(0.0, 0.0, 0.0, 1.0);
```

索引颜色

- 由一组RGB值构成一张表，“颜色”是表中项的索引
- 需要更少的帧缓存
  - 索引通常只有8位
  - 现在重要性下降
    - 内存价格下降
    - 需要更多的颜色

Input	Red	Green	Blue
0	0	0	0
1	$2^m - 1$	0	0
.	0	$2^m - 1$	0
.	.	.	.
.	.	.	.
$2^k - 1$	.	.	.
	m bits	*m* bits	*m* bits



颜色与状态

- 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

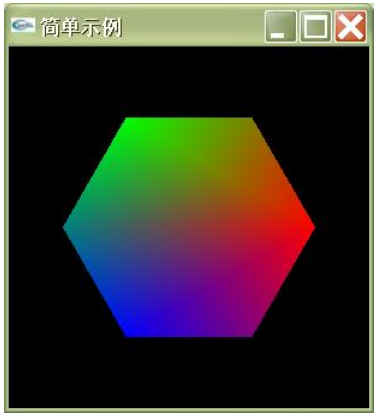
```
glColor(…);
glVertex(…);
glColor(…);
glVertex(…);
```

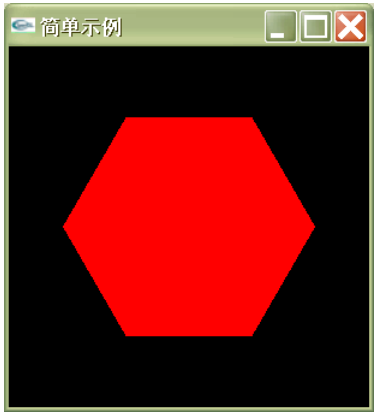
着色模型

- 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)

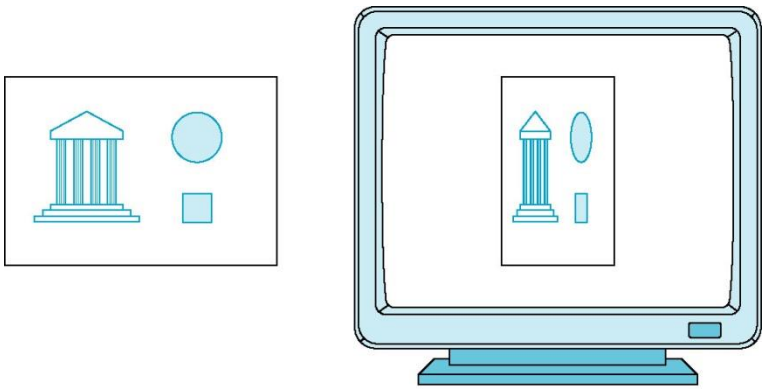
  - 平滑模式: GL_SMOOTH
  - 平面模式: GL_FLAT





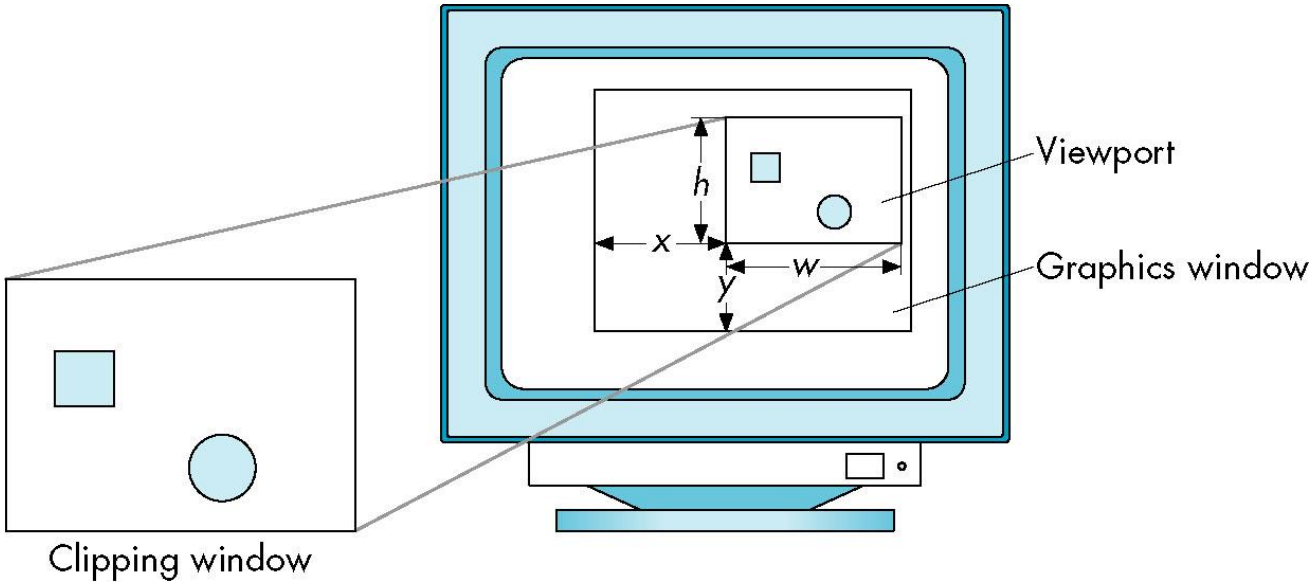
宽高比

- 宽高比(aspect ratio)：矩形的宽度与高度之比
- 如果glOrtho和glutInitWindowSize设置矩形的宽高比不同，引起对象变形
  - 保证宽高比

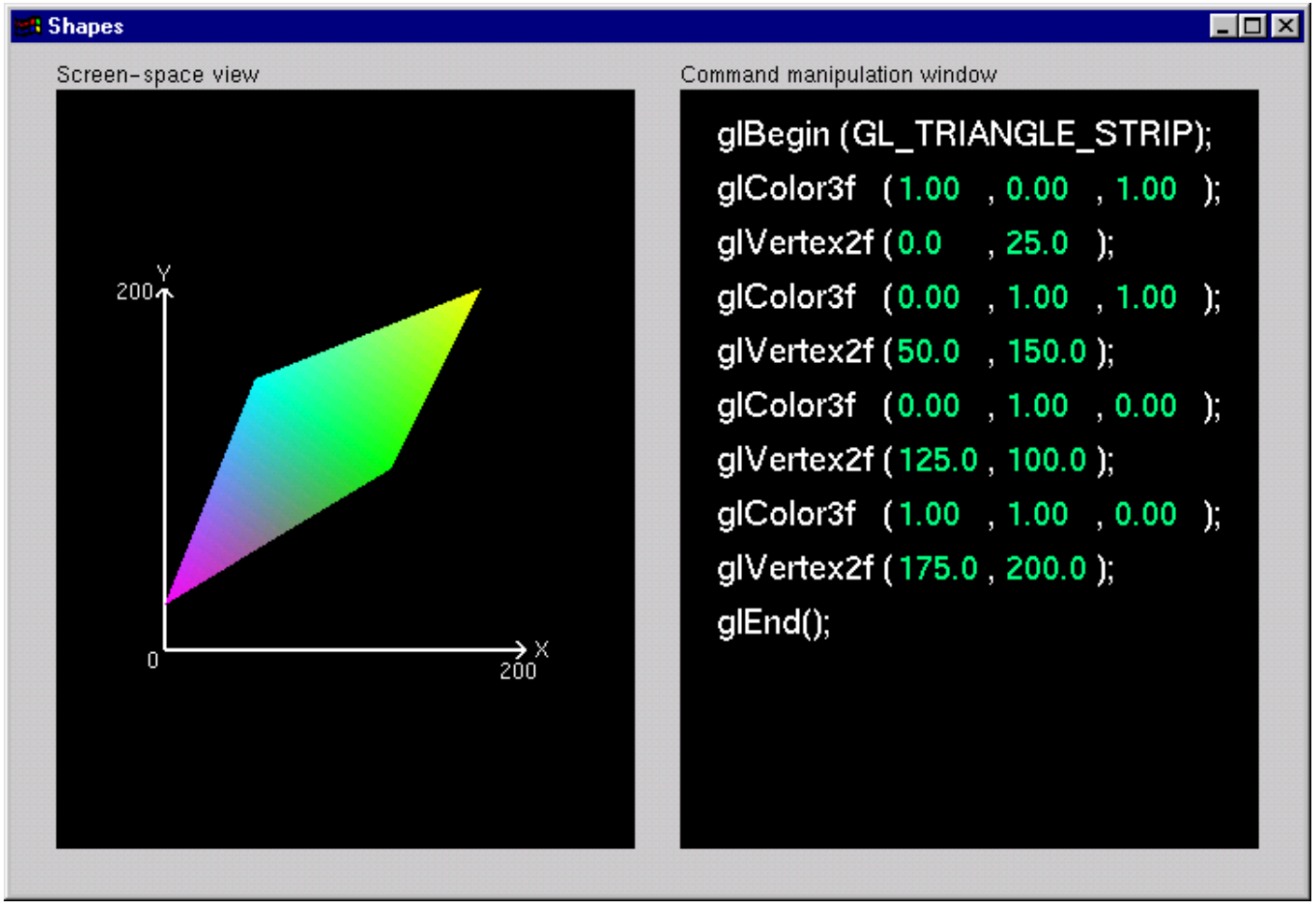


视口

- 可在当前显示窗口中定义视口用来显示图像:
  `void glViewport(GLint x,GLint y,Glint w,Glint h)`
  - 左下角为（x,y)，以像素为单位（窗口坐标）
- 通过调节视口的高度和宽度来匹配裁剪矩形的宽高比
- 视口也是状态的一部分



Shapes


Shapes
Screen-space view
Command manipulation window
glBegin (GL_TRIANGLE_STRIP);
glColor3f (1.00 , 0.00 , 1.00);
glVertex2f (0.0 , 25.0);
glColor3f (0.00 , 1.00 , 1.00);
glVertex2f (50.0 , 150.0);
glColor3f (0.00 , 1.00 , 0.00);
glVertex2f (125.0 , 100.0);
glColor3f (1.00 , 1.00 , 0.00);
glVertex2f (175.0 , 200.0);
glEnd();
Y
200
0
X
200

OpenGL程序的一般结构