2018-2019年度第二学期 00106501

计算机图形学

童 伟 华　管理科研楼1205室

E-mail: tongwh@ustc.edu.cn

中国科学技术大学　数学科学学院

http://math.ustc.edu.cn/

第三节　二维图形程序

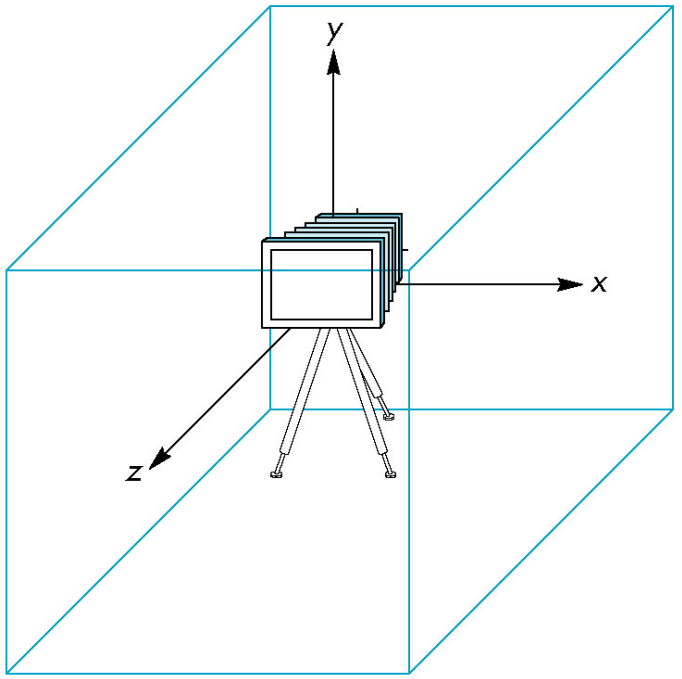
坐标系

- 在glVertex中的单位是由应用程序确定的，称为世界（对象）坐标系
- 视景体也是相对于世界坐标系指定的，视景体确定出现在图像中的对象
- 在OpenGL内部，会把世界坐标转化为照相机（视点）坐标，稍后转化为屏幕坐标
- 实际上，OpenGL也用到一些内部表示，对于shaders来说是很重要的，但是对应用程序是不可见的

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

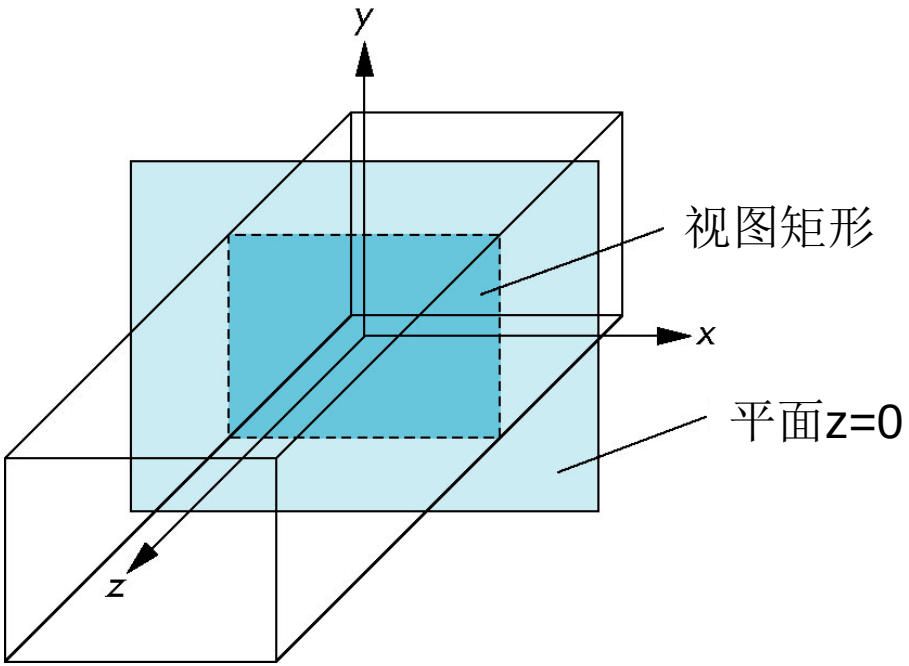
(右，顶，远)

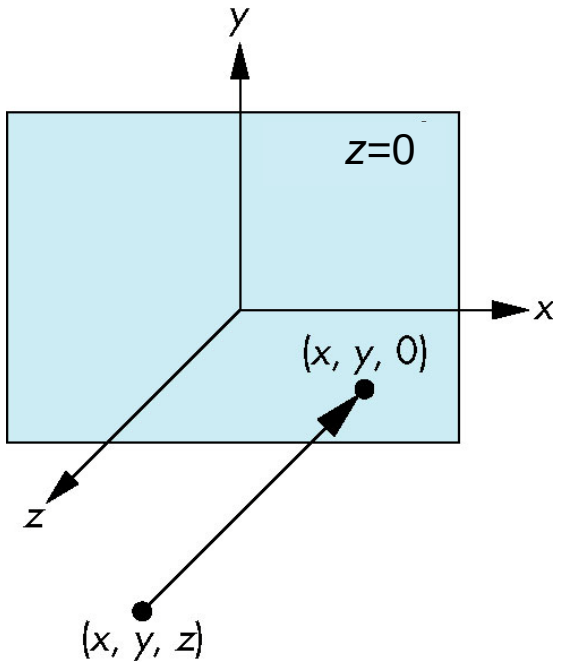


(左，底，近)

正交视图

- 在默认的正交视图中，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
- 因为OpenGL是一个状态机，拥有模型视点（modelview）矩阵栈和投影（projection）矩阵栈，因此须先设置矩阵模式

```
glMatrixMode(GL_PROJECTION);
```

- 变换函数是累加在一起的，因此需要从单位阵开始，然后把它改变为一个投影矩阵以定义视景体

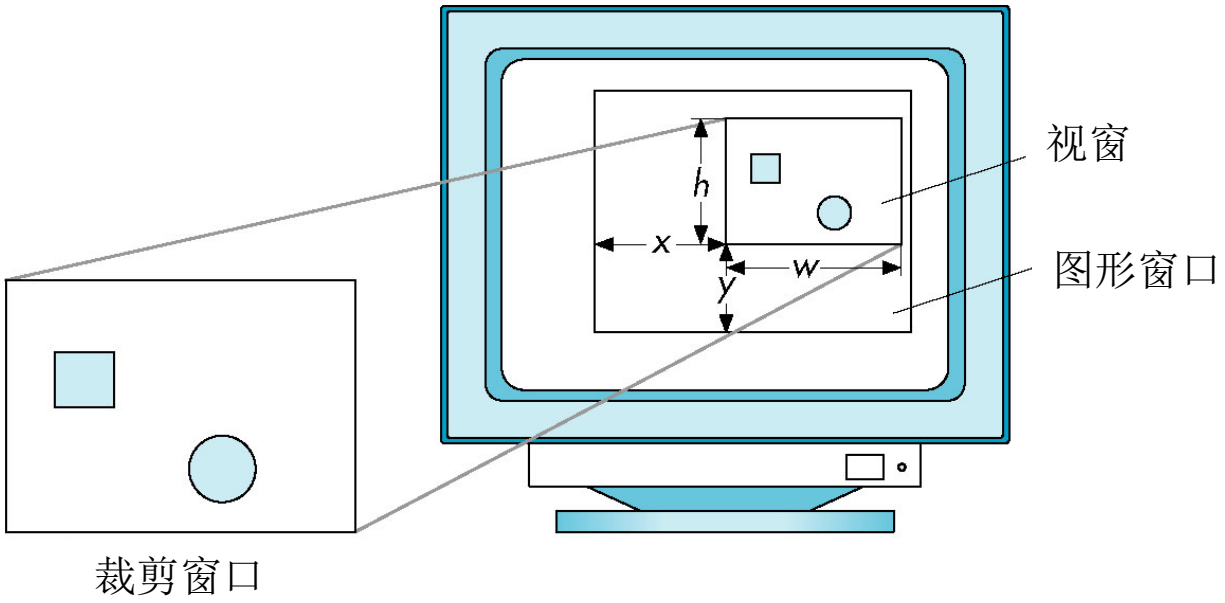
```
glLoadIdentity();
glOrtho(-1.0, 1.0, -1.0, 1.0, -1.0, 1.0);
```

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
- 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：
  gluOrtho2D(左，右，底，顶)
- 对于二维情形，视景体或裁剪体退化为裁剪窗口

视窗

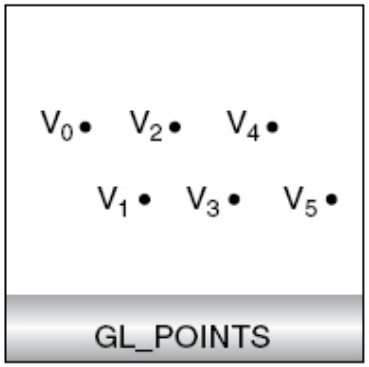
- 并不需要把整个当前窗口用来显示图像：
  `glViewport(x,y,w,h)`
- 参数值以像素为单位（屏幕坐标）

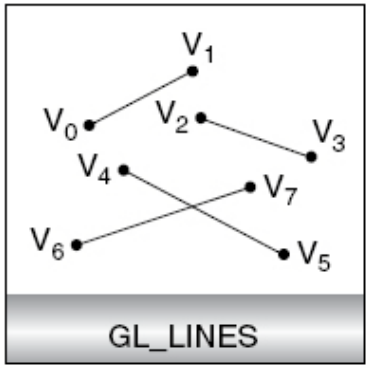


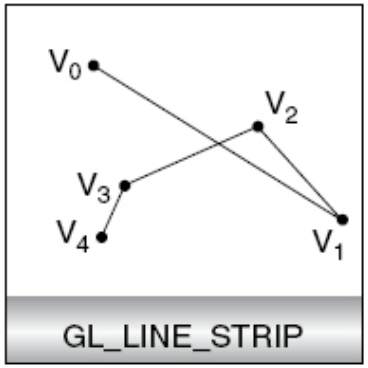
绘制代码

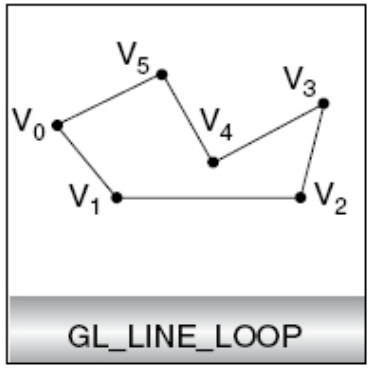
```
void MainWindow::paintGL()
{
    // render scenes
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex2d(-0.5, -0.5);
        glVertex2d(-0.5, 0.5);
        glVertex2d(0.5, 0.5);
        glVertex2d(0.5, -0.5);
    glEnd();
    glFlush();
}
```

OpenGL的基本几何形状

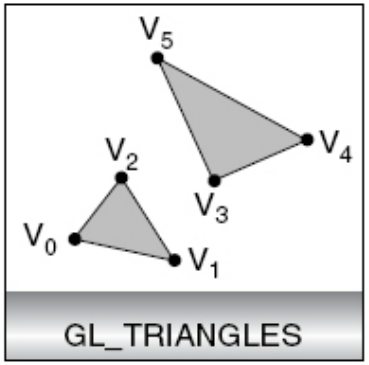


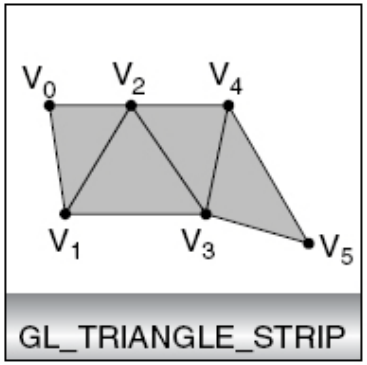


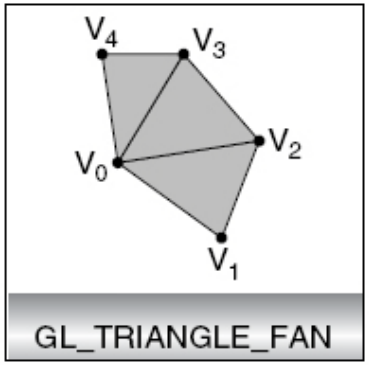


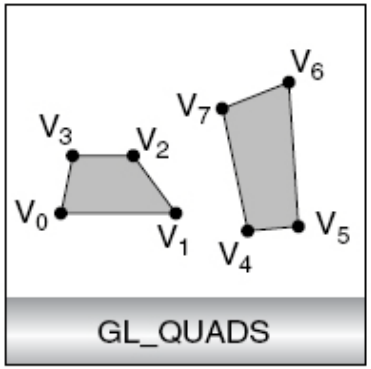


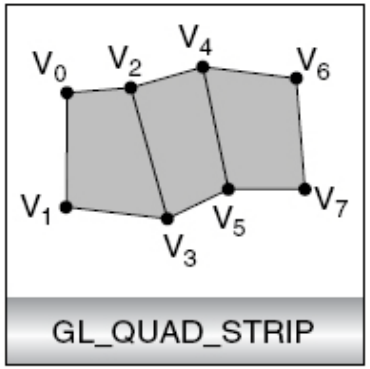
OpenGL的基本几何形状

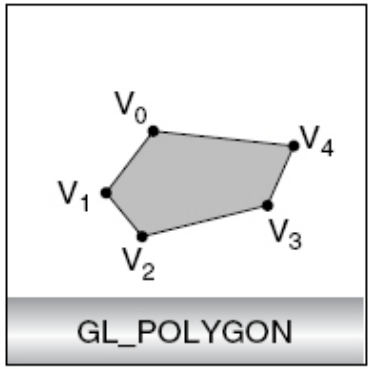






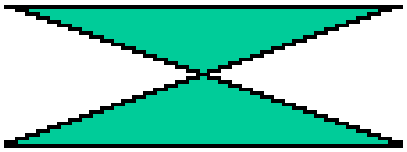






定义多边形的限制条件

- OpenGL只能显示满足下述条件的多边形
  - 简单多边形：边除顶点外不相交
  - 凸多边形：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
  - 平面多边形：所有顶点在同一平面上



非简单多边形

非凸多边形

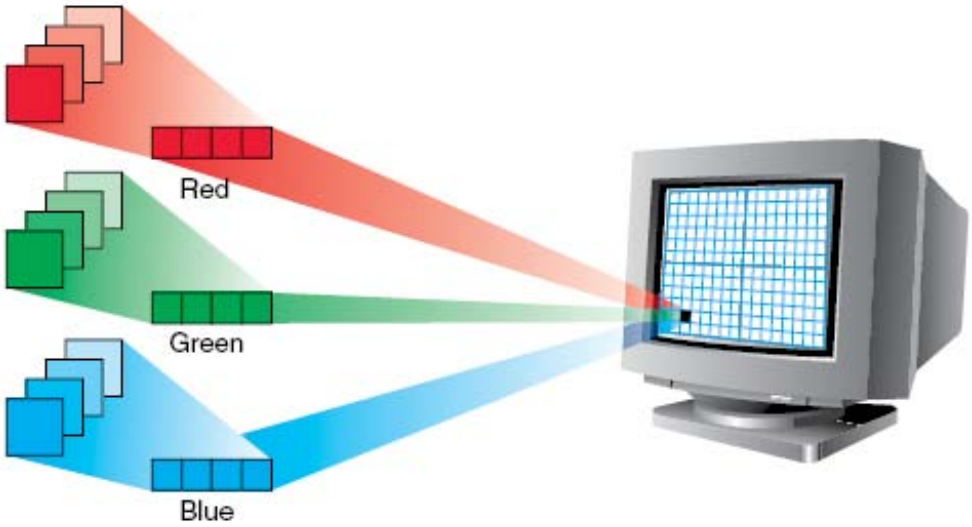
- 用户自己确保上述条件满足
  - 如果不满足上述要求，OpenGL也会有输出，只是结果看起来与期望的不同
  - 三角形自动满足上述所有限制条件，因此尽量应使用三角形网格

属性

- 属性是OpenGL中状态的一部分，确定对象的外观
  - 颜色（点、线、多边形）
  - 点的大小
  - 线段的宽度与实虚模式
  - 多边形的模式
    - 前后面
    - 填充模式：颜色或模式
    - 显示为实心多边形或者只显示边界

RGB颜色

- 颜色的每个分量在帧缓冲区中是分开存贮的
- 在缓冲区中通常每个分量占用8位字节
- 注意在函数glColor3f中颜色值的变化范围是从0.0（无）到1.0（全部），而在glColor3ub中颜色值的变化范围是从0到255



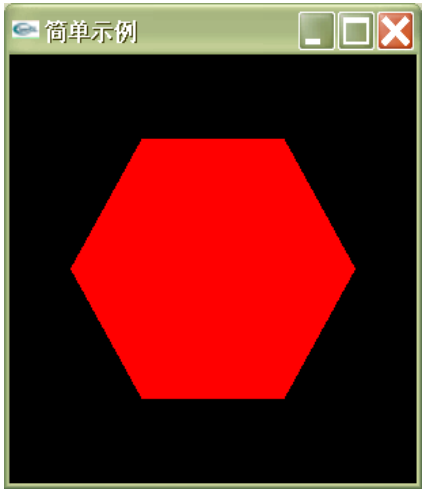
颜色与状态

- 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
```

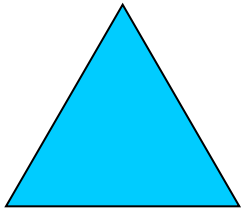
颜色的光滑化过渡

- 默认状态是光滑过渡
  - OpenGL根据多边形顶点的颜色插值出来内部的颜色
- 另外一种状态是平坦过渡
  - 第一个顶点的颜色确定填充颜色
- glShadeModel(GL_SMOOTH)或glShadeModel(GL_FLAT)

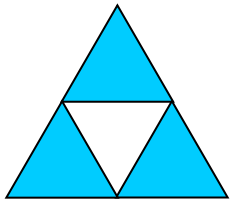


Sierpinski镂垫（2D）

- 从一个三角形开始

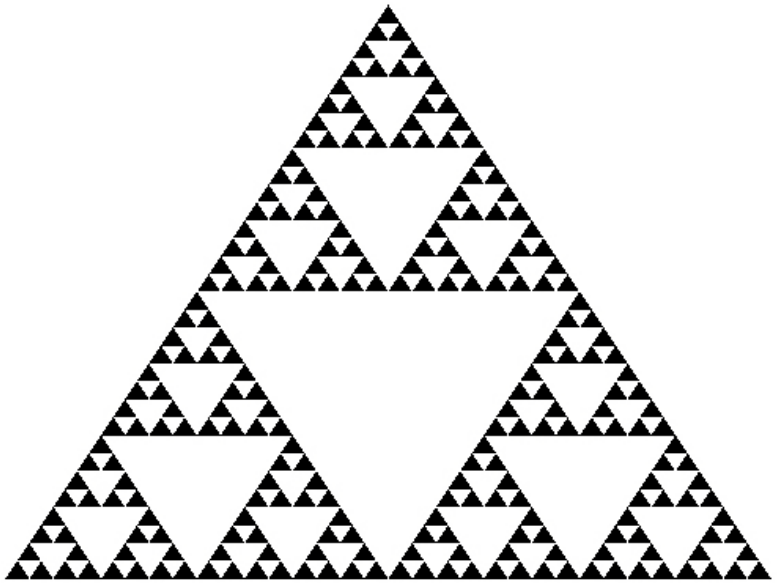


- 连接三边的中点并去掉中间的三角形



- 重复上述过程

五次细分后的结果



分形

- 考虑黑色填充区域的面积与周长（即包含填充区域的所有线段总长）
- 当持续细分时
  - 面积趋向于零
  - 但周长趋向于无穷
- 因此无穷细分后的结果不是通常的几何形状
  - 它的维数既不是一维的，也不是二维的
- 我们称之为分形（分数维1.585）

19

程序开头

```
#include <GL/glut.h>
// a point data type
typedef GLfloat point2[2];


//initial triangle
point2 v[] = {{-1.0, -0.58}, {1.0, -0.58},
       {0.0, 1.15}};
int n; // number of recursive steps
```


20

绘制三角形

```
void triangle( point2 a, point2 b, point2 c)
// display one triangle
{
      //glBegin(GL_TRIANGLES);
            glVertex2fv(a);
            glVertex2fv(b);
            glVertex2fv(c);
      //glEnd();
}
```


21

三角形细分

```
void divide_triangle(point2 a, point2 b, point2 c, int m)
{
// triangle subdivision using vertex numbers
       point2 v0, v1, v2;
       int j;
       if(m>0) {
              for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
              for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
              for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
              divide_triangle(a, v0, v1, m-1);
              divide_triangle(c, v1, v2, m-1);
              divide_triangle(b, v2, v0, m-1);
       }
       else(triangle(a,b,c));
       // draw triangle at end of recursion
}
```

显示与初始化函数

```
void display(void) {
       glClear(GL_COLOR_BUFFER_BIT);
       glBegin(GL_TRIANGLES);
           divide_triangle(v[0], v[1], v[2], n);
       glEnd();
       glFlush();
}

void myinit() {
       glMatrixMode(GL_PROJECTION);
       glLoadIdentity();
       gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
       glMatrixMode(GL_MODELVIEW);
       glClearColor (1.0, 1.0, 1.0,1.0); // 背景白色
       glColor3f(0.0,0.0,0.0);
}
```

main()函数

```
int main(int argc, char **argv)
{
       n=atoi(argv[1]); /* or set number of
subdivision steps here */
      glutInit(&argc, argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
      glutInitWindowSize(500, 500);
      glutCreateWindow("2D Gasket");
      glutDisplayFunc(display);
      myinit();
      glutMainLoop();
      return 1;
}
```

注解：效率

- 通过把glBegin和glEnd放在显示回调函数中，而不是triangle函数中，把glBegin的参数写为GL_TRIANGLES，而不是GL_POLYGON，绘制整个图形只需要调用一次glBegin和glEnd。

Thanks for your attention!