

湖南科技大学课程教案

(章节、专题首页)

授课教师: 王志喜

职称: 副教授

单位: 计算机科学与工程学院

课 程 名 称	计算机图形图像技术
章节、专题	OpenGL的基本图元
教学目标及 基 本 要 求	掌握基本的OpenGL程序设计技术以及OpenGL中基本图元的定义和属性设置等。
教 学 重 点	OpenGL编程概述, 基本图元的定义
教 学 难 点	基本图元的属性设置
教学内容与 时 间 分 配	(1) OpenGL编程概述(1课时) (2) 基本图元的定义(1.2课时) (3) 基本图元的属性(0.8课时) 共计3课时。
习 题	第3.5.1节(基础知识题)和3.5.2节(程序设计题)。

第3章 OpenGL的基本图元

3.1 OpenGL编程概述

3.1.1 基本语法

本教程只讨论C版本的OpenGL语法。

1. 函数名前缀

- 基本函数使用gl作为函数名的前缀，如glClearColor()。
- 实用函数使用glu作为函数名的前缀，如gluOrtho2D()。
- 实用工具包函数使用glut作为函数名的前缀，如glutInit()。

2. 常量名前缀

- 基本常量的名字以GL_开头，如GL_LINE_LOOP。
- 实用常量的名字以GLU_开头，如GLU_FILL。
- 实用工具包常量的名字以GLUT_开头，GLUT_RGB。

3. 函数名后缀

一组功能相同或相近的函数的函数名使用不同的后缀以支持不同的数据类型和格式，如glEvalCoord2f()、glEvalCoord2d()和glEvalCoord2dv()等，其中2表示有2个参数，*f*、*d*分别表示参数的类型是float和double，*v*表示参数以向量形式出现。

【注】这样一组函数在一般的教科书或说明书中通常写成glEvalCoord*()或glEvalCoord{12}{df}[v]()的形式，实际上代表下列8个函数，读者需要习惯这种书写格式。

```
void glEvalCoord1d(GLdouble u);  
void glEvalCoord1dv(const GLdouble *u);  
void glEvalCoord1f(GLfloat u);  
void glEvalCoord1fv(const GLfloat *u);  
void glEvalCoord2d(GLdouble u, GLdouble v);  
void glEvalCoord2dv(const GLdouble *u);  
void glEvalCoord2f(GLfloat u, GLfloat v);  
void glEvalCoord2fv(const GLfloat *u);
```

4. 特殊类型名

OpenGL定义了一些特殊的类型名，如GLint和GLfloat。其实就是32位C语言中的float和int等类型。在gl.h中可以看到类似如下的定义。

```
.....  
typedef int GLint;  
.....  
typedef float GLfloat;  
.....
```

一些基本的数据类型都有类似的定义。

3.1.2 工作方式

1. 状态机制

OpenGL的工作方式是一种状态机制，它可以进行各种状态或模式设置，这些状态或模式在重新改变之前一直有效。

例如，当前颜色是一个状态变量，在这个状态改变之前，待绘制的每个像素都使用该颜色。

2. 状态设置

许多状态变量可以通过`glEnable()`或`glDisable()`来设置成有效或无效状态，如光照、深度检测等。

在设置成有效状态之后，绝大部分状态变量都有一个默认值，例如，光照有默认的光照效果，深度检测有默认的检测方法。

3.1.3 程序的基本结构

OpenGL程序的基本结构可分为三个部分。

1. 初始化

主要用于设置一些OpenGL的状态开关，如颜色模式的选择，是否作光照处理、进行深度检测等。这些状态一般都用函数glEnable()和glDisable()来设置。

2. 设置投影方式和观察体

主要有glViewport()、glOrtho()、glFrustum()、gluPerspective()和gluOrtho2D()等函数，这里主要介绍glViewport()函数，其他函数在后续章节中介绍。

(1) 什么是视口。

- 视口是程序窗口中的一个矩形绘图区。如图3-1所示。
- 在屏幕上打开一个窗口时，自动把视口设置为整个程序窗口的大小。
- 可以定义一个比程序窗口小的视口。
- 也可以在一个程序窗口中定义多个视口，达到分屏显示的目的。

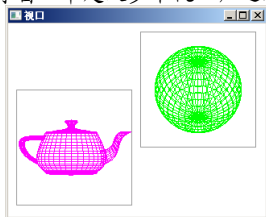


图3-1 视口

(2) glViewport()。

【函数原型】 `void glViewport(int left, int bottom, int width, int height);`

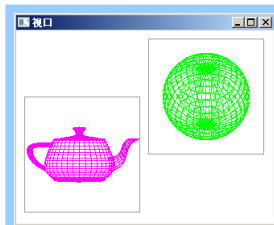
【功能】设置视口在程序窗口中的位置和大小。

【参数】

- (left, bottom)是视口左下角在程序窗口中的坐标。
- (width, height)是视口的宽和高。

【说明】

- 坐标原点位于程序窗口左下角，以像素为单位。
- 默认视口为(0, 0, W , H)，其中 W 、 H 分别表示程序窗口的宽和高。



3. 构造对象的数学描述

OpenGL的主要部分，使用OpenGL的库函数构造几何物体对象的数学描述，包括点线面的位置和拓扑关系、几何变换和光照处理等。

4. C源程序大致框架

```
//Frame.c
```

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

```
void init()
```

```
{ /* 全局初始化。主要用于设置一些全局的状态，如颜色模式、窗口的初始位置和大小等。使用默认值时不需要定义该函数。
```

```
*/
```

```
}
```

```
void init2()
```

```
{ /* 当前窗口初始化。主要用于设置一些与当前窗口关联的状态或开关，如光照处理、光源特性、深度检测和裁剪等。使用默认值时不需要定义该函数。
```

```
*/
```

```
}
```

```
void Reshape(int w, int h)
```

```
{ /* 设置投影方式和观察体。主要使用glViewport()、glOrtho()、glFrustum()、gluPerspective()和gluOrtho2D()等。使用默认值时不需要定义该函数。
```

```
*/
```

```
}
```

```
void Paint()
```

```
{ /* 使用OpenGL库函数构造对象的数学描述，包括点线面的位置、几何变换和光照处理等，是OpenGL的主要部分。
```

```
*/
```

```
}
```

```
int main(int argc, char *argv[])
```

```
{ glutInit(&argc, argv); // 初始化GLUT, 记录main()的参数
```

```
init(); // 全局初始化, 使用默认值时不是必需的
```

```
glutCreateWindow("窗口标题"); // 创建程序窗口, 指定窗口标题
```

```
init2(); // 当前窗口初始化, 使用默认值时不是必需的
```

```
// 注册回调函数
```

```
glutDisplayFunc(Paint); // 指定场景绘制函数, 必需
```

```
glutReshapeFunc(Reshape); // 指定窗口变化回调函数, 缺省则使用默认值
```

```
glutMainLoop(); // 开始循环执行OpenGL命令
```

```
}
```

3.2 一个简单的OpenGL程序

3.2.1 源程序和运行结果

下列程序在程序窗口的中间位置绘制了一个填充的白色三角形。运行结果如图3-2所示。



图3-2 一个简单例子

```
// Simple.c
#include<GL/freeglut.h>

void Reshape(int w, int h) // 窗口变化回调函数(宽度, 高度)
{
    glViewport(w / 4, h / 4, w / 2, h / 2); // 视口(左, 下, 宽, 高)
}

void Paint() // 对象的描述
{
    glClear(GL_COLOR_BUFFER_BIT); // 清除颜色缓冲区
    glBegin(GL_TRIANGLES); // 定义三角形(逆时针指定顶点)
    {
        glVertex2f(-0.95, -0.95); // 左下顶点(二维坐标)
        glVertex2f(0.95, -0.95); // 右下顶点
        glVertex2f(0, 0.95); // 上方顶点
        // 默认的2维坐标值范围是(-1, -1)--(1, 1)
    }
    glEnd(); // 三角形定义结束
    glFlush(); // 强制OpenGL命令序列在有限的时间内完成执行
}
```

```
int main(int argc, char *argv[])
{
    glutInit(&argc, argv); // 初始化GLUT, 记录main()的参数
    // 全局初始化
    glutInitWindowPosition(100, 100); // 设置程序窗口在屏幕上的初始位置
    glutInitWindowSize(160, 160); // 设置程序窗口在屏幕上的初始大小
    glutCreateWindow("一个三角形"); // 创建程序窗口, 指定窗口标题
    // 注册回调函数
    glutDisplayFunc(Paint); // 指定场景绘制循环函数, 必需
    glutReshapeFunc(Reshape); // 指定窗口变化回调函数, 缺省则使用默认值
    glutMainLoop(); // 开始循环执行OpenGL命令
}
```

3.2.2 相关函数说明

1. 初始化和启动事件

(1) 初始化GLUT库。使用glutInit()。

【函数原型】`void glutInit(int *argc, char **argv);`

【参数】

- `argc`: 一个指针，指向从main()函数传递过来的未更改的argc变量。
- `argv`: 一个指针，是从main()函数传递过来的未更改的argv变量。

【说明】该函数获取main()函数的两个参数，对应的main()函数的形式应该是“`int main(int argc, char *argv[]);`”。

(2) 设置初始显示模式。使用glutInitDisplayMode()。

【函数原型】`void glutInitDisplayMode(unsigned int mode);`

【功能】设置程序窗口的初始显示模式。

【参数】最常用的mode值是通过位或运算 (bit_or) 指定颜色模型和缓存数量。

- 颜色模型。GLUT_RGBA表示使用RGBA颜色模型，GLUT_INDEX表示使用索引颜色模型。
- 缓存数量。GLUT_SINGLE表示使用单显示缓冲区，GLUT_DOUBLE表示使用双显示缓冲区。

【说明】

- 显示模式的默认值。显示模式默认为使用RGBA模式的单缓存窗口，即mode的默认值是GLUT_RGBA | GLUT_SINGLE。如果需要使用RGBA模式的双缓存窗口，则mode值应该规定为GLUT_DOUBLE | GLUT_RGBA或GLUT_DOUBLE。
- RGBA颜色模式。使用Red、Green、Blue和Alpha这四个分量表示颜色。其中，Alpha是一个和物体透明程度有关的量，取值范围是[0, 1]，默认值为1，表示物体不透明。
- 双缓存的设置与使用。具体介绍请参阅“OpenGL中的图形变换”一章。

(3) 设置窗口的初始位置。使用glutInitWindowPosition()。

【函数原型】`void glutInitWindowPosition(int x, int y);`

【功能】设置各程序窗口在屏幕上的初始位置（以像素为单位）。

【说明】屏幕坐标系的原点位于屏幕左上角，窗口的默认位置由窗口系统决定。

(4) 设置窗口的初始大小。使用glutInitWindowSize()。

【函数原型】`void glutInitWindowSize(int width, int height);`

【功能】设置各程序窗口在屏幕上的初始大小（以像素为单位）。

【说明】程序窗口的默认大小为(300, 300)。

(5) 进入GLUT事件处理循环。使用glutMainLoop()。

【函数原型】`void glutMainLoop(void);`

【功能】开始循环执行OpenGL命令。

2. 窗口管理

(1) 创建顶层窗口。使用glutCreateWindow()。

【函数原型】`int glutCreateWindow(const char *title);`

【功能】创建程序窗口，指定窗口标题。

(2) 设置窗口位置。使用glutPositionWindow()。

【函数原型】`void glutPositionWindow(int x, int y);`

【功能】设置当前窗口在屏幕上的位置。

(3) 设置窗口大小。使用glutReshapeWindow()。

【函数原型】`void glutReshapeWindow(int width, int height);`

【功能】设置当前窗口在屏幕上的大小。

3. 注册回调函数

(1) 注册窗口变化回调函数。使用glutReshapeFunc()。

【函数原型】`void glutReshapeFunc(void (*func)(int width, int height));`

【功能】为当前窗口指定自定义的窗口变化回调函数。

【参数】func是窗口变化回调函数的名字，函数原型为“`void (*func)(int width, int height);`”，其中参数width和height表示窗口的宽度和高度（以像素为单位）。

(2) 注册场景绘制函数。使用glutDisplayFunc()。

【函数原型】`void glutDisplayFunc(void (*func)(void));`

【功能】为当前窗口指定自定义的场景绘制函数。

【参数】func是自定义场景绘制函数的名字，函数原型为“`void func(void);`”。

4. 其他函数和调用

(1) 清除颜色缓冲区。使用下列调用形式。

【调用形式】 `glClear(GL_COLOR_BUFFER_BIT);`

【功能】清除颜色缓冲区，即将颜色缓冲区设置为预先指定的颜色。

(2) 开始三角形定义。使用下列调用形式。

【调用形式】 `glBegin(GL_TRIANGLES);`

【说明】在该调用之后指定每个三角形的顶点坐标（允许定义多个三角形）。

(3) 指定二维顶点坐标。可以使用 `glVertex2f()`。

【函数原型】 `void glVertex2f(GLfloat x, GLfloat y);`

【说明】默认情况下，2维坐标值的范围是 $(-1, -1) \sim (1, 1)$ 。

(4) 结束图元定义。使用 `glEnd()`。

【函数原型】 `void glEnd(void);`

3.3 基本图元的定义

OpenGL的基本图元有点、线和多边形。

从根本上看，OpenGL绘制的所有复杂三维物体都是由一系列基本图元构成的，曲线、曲面分别是由一系列直线段、多边形近似得到的。

3.3.1 绘图准备

在开始绘制新图形前，屏幕上可能已经有一些图形，必须清除这些内容，以免影响绘制效果。可以使用下列函数。

1. 指定背景颜色

【函数原型】`void glClearColor(float red, float green, float blue, float alpha);`

【功能】指定当前背景颜色。

【参数】(red, green, blue, alpha)为RGBA颜色值。

【说明】默认为黑色，即参数值全为0。

2. 清除缓冲区

【函数原型】`void glClear(GLbitfield mask);`

【功能】清除指定的缓冲区（设置为预先指定的值）。

【参数】标识要清除的缓冲区，使用按位或运算组合。

【说明】共有4个标识，最常用的标志是

- `GL_COLOR_BUFFER_BIT`（颜色缓冲区，设置为背景颜色）
- `GL_DEPTH_BUFFER_BIT`（深度缓冲区，设置为最远深度）

3.3.2 绘图结束

下列两个函数用于结束绘图并返回。

1. glFlush()

【函数原型】 `void glFlush(void);`

【功能】强制OpenGL命令序列在有限的时间内完成执行。

2. glFinish()

【函数原型】 `void glFinish(void);`

【功能】强制完成已发出的全部OpenGL命令的执行，即等到全部命令执行完毕以后才返回。

【说明】应尽量避免使用glFinish()，以免影响性能。

3.3.3 图元定义

OpenGL中的点是三维的，二维坐标 (x, y) 表示 $(x, y, 0)$ ；线用一系列相连的顶点定义；多边形是一个封闭的线段，通过选择属性，既可以得到填充的多边形，也可以是轮廓线，或是一系列点。

OpenGL中的多边形是凸多边形，光滑的曲线、曲面都是由一系列线段、多边形近似得到的，OpenGL不直接提供绘制曲线、曲面的命令。

描述基本图元就是按照某种顺序给出基本图元的每个顶点，并同时当前颜色、当前纹理坐标、当前法向量等值赋给这些顶点。

1. 指定当前颜色

【函数格式】 `void glColor{34}{u}{bdfis}[v]()`

【功能】在绘制图形前，按照指定的参数格式设定对象颜色的RGBA值。

【说明】

- 四个分量的值都会映射到[0, 1]范围，且A分量的默认值为1。
- 当前颜色是一个状态变量，在这个状态改变之前，待绘制的每个像素都使用该颜色。
- 可以在任何时候更改当前颜色，也可以在glBegin()与glEnd()之间调用。

```
void glColor3b(GLbyte red, GLbyte green, GLbyte blue);
void glColor3d(GLdouble red, GLdouble green, GLdouble blue);
void glColor3f(GLfloat red, GLfloat green, GLfloat blue);
void glColor3i(GLint red, GLint green, GLint blue);
void glColor3s(GLshort red, GLshort green, GLshort blue);
void glColor3ub(GLubyte red, GLubyte green, GLubyte blue);
void glColor3ui(GLuint red, GLuint green, GLuint blue);
void glColor3us(GLushort red, GLushort green, GLushort blue);
void glColor4b(GLbyte red, GLbyte green, GLbyte blue, GLbyte alpha);
void glColor4d(GLdouble red, GLdouble green, GLdouble blue, GLdouble alpha);
void glColor4f(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha);
void glColor4i(GLint red, GLint green, GLint blue, GLint alpha);
void glColor4s(GLshort red, GLshort green, GLshort blue, GLshort alpha);
void glColor4ub(GLubyte red, GLubyte green, GLubyte blue, GLubyte alpha);
void glColor4ui(GLuint red, GLuint green, GLuint blue, GLuint alpha);
void glColor4us(GLushort red, GLushort green, GLushort blue, GLushort alpha);
```

```
void glColor3bv(const GLbyte *v);
void glColor3dv(const GLdouble *v);
void glColor3fv(const GLfloat *v);
void glColor3iv(const GLint *v);
void glColor3sv(const GLshort *v);
void glColor3ubv(const GLubyte *v);
void glColor3uiv(const GLuint *v);
void glColor3usv(const GLushort *v);
void glColor4bv(const GLbyte *v);
void glColor4dv(const GLdouble *v);
void glColor4fv(const GLfloat *v);
void glColor4iv(const GLint *v);
void glColor4sv(const GLshort *v);
void glColor4ubv(const GLubyte *v);
void glColor4uiv(const GLuint *v);
void glColor4usv(const GLushort *v);
```

2. 定义顶点

【函数格式】 `void glVertex{234}{sifd}[v]()`

【功能】按照指定的参数格式指定一个顶点坐标 (x, y, z, w) 。

【说明】

- 这些函数只有在 `glBegin()` 与 `glEnd()` 之间调用才有效。
- OpenGL使用齐次坐标 (x, y, z, w) 表示一个顶点坐标。其中， z 和 w 的默认值分别是0和1。
- 在齐次坐标 (x, y, z, w) 中，如果 $w \neq 0$ ，则 $(x/w, y/w, z/w)$ 表示一个实际的顶点坐标，否则， (x, y, z, w) 表示一个无穷远的位置，该无穷远位置的方向为 $(0, 0, 0) \sim (x, y, z)$ 。

```
void glVertex2d(GLdouble x, GLdouble y);
void glVertex2f(GLfloat x, GLfloat y);
void glVertex2i(GLint x, GLint y);
void glVertex2s(GLshort x, GLshort y);
void glVertex3d(GLdouble x, GLdouble y, GLdouble z);
void glVertex3f(GLfloat x, GLfloat y, GLfloat z);
void glVertex3i(GLint x, GLint y, GLint z);
void glVertex3s(GLshort x, GLshort y, GLshort z);
void glVertex4d(GLdouble x, GLdouble y, GLdouble z, GLdouble w);
void glVertex4f(GLfloat x, GLfloat y, GLfloat z, GLfloat w);
void glVertex4i(GLint x, GLint y, GLint z, GLint w);
void glVertex4s(GLshort x, GLshort y, GLshort z, GLshort w);

void glVertex2dv(const GLdouble *v);
void glVertex2fv(const GLfloat *v);
void glVertex2iv(const GLint *v);
void glVertex2sv(const GLshort *v);
void glVertex3dv(const GLdouble *v);
void glVertex3fv(const GLfloat *v);
void glVertex3iv(const GLint *v);
void glVertex3sv(const GLshort *v);
void glVertex4dv(const GLdouble *v);
void glVertex4fv(const GLfloat *v);
void glVertex4iv(const GLint *v);
void glVertex4sv(const GLshort *v);
```

3. 基本图元定义的开始

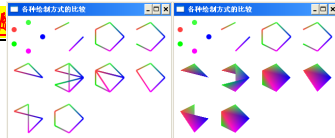
【函数原型】 `void glBegin(GLenum mode);`

【功能】表示一个基本图元定义的开始。

【参数】mode可选下列符号常量。

- GL_POINTS: 每一个顶点作为一个独立的点。
- GL_LINES: 每一对顶点作为一条独立的线段。
- GL_LINE_STRIP: 顶点依次相连成一组线段。
- GL_LINE_LOOP: 顶点依次相连成一组线段，并连接末顶点与首顶点。
- GL_TRIANGLES: 每三个顶点作为一个独立的三角形。
- GL_TRIANGLE_STRIP: 三角形带，顶点 n 、 $n+1$ 和 $n+2$ 定义第 n 个三角形。
- GL_TRIANGLE_FAN: 三角形扇形，顶点1、 $n+1$ 和 $n+2$ 定义第 n 个三角形。
- GL_QUADS: 每四个顶点作为一个独立四边形。
- GL_QUAD_STRIP: 四边形带，顶点 $2n-1$ 、 $2n$ 、 $2n+2$ 和 $2n+1$ 定义第 n 个四边形。可能形成有交叉的四边形，不好把握。
- GL_POLYGON: 所有顶点作为一个简单多边形。最好不要定义凹多边形，多边形在处理时会分解成三角形扇形，凹多边形可能得不到预期效果。

图3-3依次演示了上述10种基本图元的绘制效果，每个图元都使用5个顶点定义，顶点坐标从左上顶点开始按顺时针顺序指定。



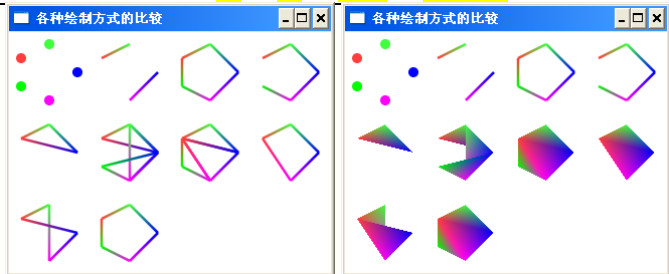


图3-3 各种绘制方式的演示

4. 基本图元定义的结束

【函数原型】 `void glEnd(void);`

【功能】表示一个基本图元定义的结束。

3.3.4 几种预定义的几何形体

这里只介绍矩形、立方体、球面、圆锥和犹他茶壶等5种在OpenGL和GLUT中已经预定义的几何形体。图3-4给出了后4种几何形体的线框图。

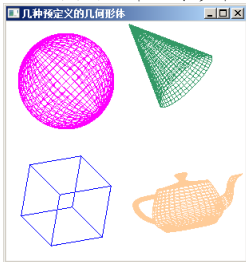


图3-4 几种预定义的几何形体

1. 矩形

【函数原型】

- `void glRectd(GLdouble x1, GLdouble y1, GLdouble x2, GLdouble y2);`
- `void glRectf(GLfloat x1, GLfloat y1, GLfloat x2, GLfloat y2);`
- `void glRecti(GLint x1, GLint y1, GLint x2, GLint y2);`
- `void glRects(GLshort x1, GLshort y1, GLshort x2, GLshort y2);`
- `void glRectdv(const GLdouble *v1, const GLdouble *v2);`
- `void glRectfv(const GLfloat *v1, const GLfloat *v2);`
- `void glRectiv(const GLint *v1, const GLint *v2);`
- `void glRectsv(const GLshort *v1, const GLshort *v2);`

【功能】在 xy 平面上定义一个矩形。

【参数】

- $(x1, y1)$ 或 $v1$ 是左下角顶点的坐标。
- $(x2, y2)$ 或 $v2$ 是右上角顶点的坐标。

2. 立方体

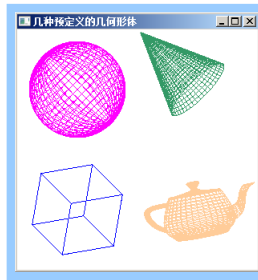
【函数原型】

- `void glutSolidCube(GLdouble size);`
- `void glutWireCube(GLdouble size);`

【功能】绘制实心立方体或网格线立方体。

【参数】size是立方体的边长。

【说明】中心在模型坐标原点。



3. 犹他茶壶

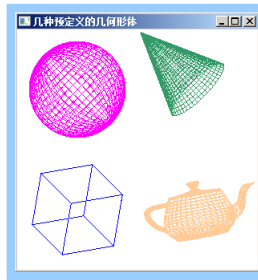
【函数原型】

- `void glutSolidTeapot(GLdouble size);`
- `void glutWireTeapot(GLdouble size);`

【功能】绘制实心茶壶或网格线茶壶。

【参数】size是茶壶外接球面的半径。

【说明】中心在模型坐标原点。



4. 球面

【函数原型】

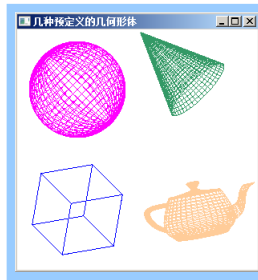
- `void glutSolidSphere(GLdouble radius, int slices, int stacks);`
- `void glutWireSphere(GLdouble radius, int slices, int stacks);`

【功能】绘制实心球面或网格线球面。

【参数】

- `radius`是球半径。
- `slices`是围绕 z 轴（侧面）的分割数（经线数）。
- `stacks`是沿 z 轴（高度方向）的分割数（纬线数）。

【说明】中心在模型坐标原点。



5. 圆锥

【函数原型】

- `void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);`
- `void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);`

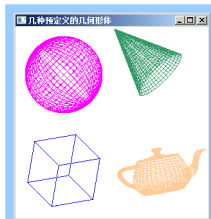
【功能】绘制一个实心圆锥或网格线圆锥。

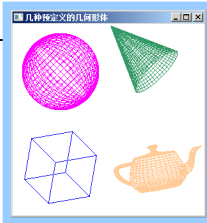
【参数】

- `base`是底部半径。
- `height`是圆锥的高度。
- `slices`是底部圆弧的分割数（侧面分割数，经线数）。
- `stacks`是沿高度方向（ z 方向）的分割数（纬线数）。

【说明】

- 底部位于模型坐标 xy 平面内。
- 底部中心在模型坐标原点。





3.3.5 预定义几何形体示例

1. 示例说明

这里给出的示例程序在白色屏幕窗口的左下区域显示一个蓝色的线框立方体，右下区域显示一个茶色的线框茶壶，左上区域显示一个紫色的线框球，右上区域显示一个海绿色的线框圆锥。运行结果如图3-4所示。

2. 相关函数调用

- `glutGet(GLUT_WINDOW_X)`。获得程序窗口左上角X坐标（屏幕坐标）。
- `glutGet(GLUT_WINDOW_Y)`。获得程序窗口左上角Y坐标（屏幕坐标）。
- `glutGet(GLUT_WINDOW_WIDTH)`。获得程序窗口宽度。
- `glutGet(GLUT_WINDOW_HEIGHT)`。获得程序窗口高度。
- `glLoadIdentity()`。本例中的作用是消除其他视口中的物体变换对当前视口的影响，详细介绍见后续章节。
- `glRotatef(th, x, y, z)`。将物体绕旋转轴 $(0,0,0) \sim (x,y,z)$ 旋转 `th` 度以调整物体的方向，详细介绍见后续章节。

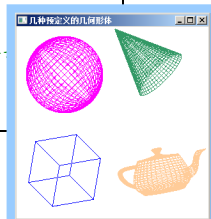
3. 源程序

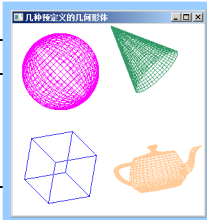
```
// PreObject.c
#include <gl/freeglut.h>

void Viewport(int x, int y, int w, int h) // 含glLoadIdentity()的视口定义
{
    glViewport(x, y, w, h); // 定义视口
    glLoadIdentity(); // 消除其他视口的影响，函数介绍见后续章节
    // 注意，调用顺序不能换
}

void Paint()
{
    int w = glutGet(GLUT_WINDOW_WIDTH) / 2; // 计算视区宽度
    int h = glutGet(GLUT_WINDOW_HEIGHT) / 2; // 计算视区高度
    glClearColor(1, 1, 1, 1); // 白色背景
    glClear(GL_COLOR_BUFFER_BIT); // 清除颜色缓存

    Viewport(0, 0, w, h); // 左下方视口
    glColor3f(0, 0, 1); // 设置当前颜色，蓝色
    glRotatef(-45, 1, 1, 0); // 调整立方体方向，函数介绍见后续章节
    glutWireCube(1); // 线框立方体，边长为1
    // 注意，除glColor3f()外，其余调用顺序不能换
}
```





```
Viewport(w, 0, w, h); // 右下方视口  
glColor3f(1, 0.8, 0.6); // 设置当前颜色, 茶色  
glRotatef(15, 0, 1, 1); // 调整茶壶方向, 函数介绍见后续章节  
glutWireTeapot(0.6); // 线框茶壶, 外接球半径为1
```

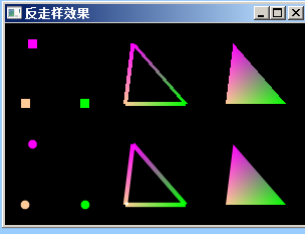
```
Viewport(0, h, w, h); // 左上方视口  
glColor3f(1, 0, 1); // 设置当前颜色, 紫色  
glRotatef(-75, 1, 1, 0); // 调整球体方向, 函数介绍见后续章节  
glutWireSphere(0.8, 48, 24); // 线框球(半径, 经线数, 纬线数)
```

```
Viewport(w, h, w, h); // 右上方视口  
glColor3f(0.2, 0.6, 0.4); // 设置当前颜色, 海绿色  
glRotatef(-75, 1, 1, 0); // 调整圆锥方向, 函数介绍见后续章节  
glutWireCone(0.6, 1.4, 48, 24); /* 线框圆锥(底部半径, 高度, 经线数, 纬线数) */
```

```
glFlush(); // 强制OpenGL命令序列在有限的时间内完成执行
```

```
}
```

```
int main(int argc, char *argv[])
{
    glutInit(&argc, argv); // 初始化GLUT, 记录main()的参数
    glutCreateWindow("四个预定义几何形体"); // 指定窗口标题
    glutDisplayFunc(Paint); // 指定场景绘制函数
    glutMainLoop(); // 开始执行
}
```



3.4 基本图元的属性

3.4.1 点宽、线宽和多边形的绘制方式

1. 点宽

点宽表示点的宽度或点的大小。

(1) 点宽的指定。使用 `glPointSize()` 指定。

【函数原型】 `void glPointSize(GLfloat size);`

【功能】以像素为单位设置点宽。

【参数】 `size` 表示点宽，必须大于0。默认值为1.0。

(2) 部分说明。

- 点宽可以不是整数。
- 如果没有设置反走样处理，则宽度截断为整数。例如，宽度为8.8的点显示为8×8像素的正方形。
- 如果设置了反走样处理，则宽度不作取整运算，直接用于反走样处理。关于反走样的相关介绍和示例请参阅“OpenGL的真实感图形”一章。

2. 线宽

线宽表示线的宽度或线的粗细程度。

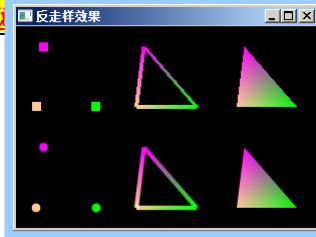
(1) 线宽的指定。使用`glLineWidth()`指定。

【函数原型】`void glLineWidth(GLfloat width);`

【功能】以像素为单位设置线宽度。

(2) 部分说明。

- 线宽可以不是整数。
 - 如果没有设置反走样处理，则宽度截断为整数。
 - 如果设置了反走样处理，则宽度不作取整运算，直接用于反走样处理。
- 关于反走样的相关介绍和示例请参阅“OpenGL的真实感图形”一章。



3. 多边形的绘制方式

(1) 绘制方式的指定。使用glPolygonMode()指定。

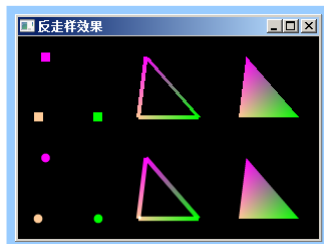
【函数原型】void glPolygonMode(GLenum face, GLenum mode);

【功能】选择多边形的绘制方式。

【参数】

- face选取多边形的正面或反面，可选GL_FRONT(选取正面)、GL_BACK(选取反面)和GL_FRONT_AND_BACK(选取正面和反面)。
- mode指定多边形的绘制方式，可选GL_POINT(只绘制顶点)、GL_LINE(只绘制边框)和GL_FILL(绘制填充多边形)。

(2) 默认情形。默认情形与调用glPolygonMode(GL_FRONT_AND_BACK, GL_FILL)等价，即正面和反面都是填充的多边形。



4. 示例

下列程序用于演示同一在各种绘制方式下的绘制效果。运行结果如图3-6所示。

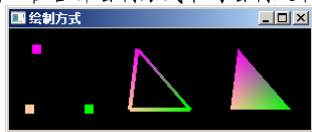


图3-6 绘制方式演示

```
// PolygonMode.c
```

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

```
void Hint() // 当前窗口初始化(点宽和线宽)
```

```
{   glPointSize(8.8); // 点宽=8.8  
    glLineWidth(4.4); // 线宽=4.4  
}
```

```
void Triangle() // 定义彩色三角形
```

```
{   glBegin(GL_TRIANGLES); // 定义三角形(逆时针指定顶点)  
    {   glColor3f(1, 0.8, 0.6); // 茶色  
        glVertex2f(-0.6, -0.6); // 左下顶点  
        glColor3f(0, 1, 0); // 绿色  
        glVertex2f(0.6, -0.6); // 右下顶点  
        glColor3f(1, 0, 1); // 紫色  
        glVertex2f(-0.45, 0.6); // 上方顶点  
    }  
    glEnd();  
}
```



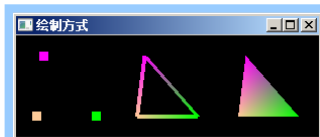
```
void Paint() // 场景绘制函数
{
    int w = glGet(GLUT_WINDOW_WIDTH)/3; // 视口宽度
    int h = glGet(GLUT_WINDOW_HEIGHT); // 视口高度
    glClear(GL_COLOR_BUFFER_BIT); // 清除颜色缓冲区

    glViewport(0, 0, w, h); // 第0列
    glPolygonMode(GL_FRONT, GL_POINT); // 正面，只绘制顶点
    Triangle();

    glViewport(w, 0, w, h); // 第1列
    glPolygonMode(GL_FRONT, GL_LINE); // 正面，只绘制边框
    Triangle();

    glViewport(2 * w, 0, w, h); // 第2列
    glPolygonMode(GL_FRONT, GL_FILL); // 正面，绘制填充多边形
    Triangle();

    glFlush(); // 强制OpenGL命令序列在有限时间内完成执行
}
```



```
int main(int argc, char *argv[])
{
    glutInit(&argc, argv); // 初始化GLUT, 记录main()的参数
    glutInitWindowSize(300, 100); // 程序窗口的初始大小(3:1)
    glutCreateWindow("绘制方式"); // 创建程序窗口, 指定窗口标题
    Hint(); // 当前窗口初始化(点宽和线宽)
    glutDisplayFunc(Paint); // 指定场景绘制函数
    glutMainLoop(); // 开始循环执行OpenGL命令
}
```

3.4.2 点画线与点画多边形※

1. 点画线的线型

(1) 线型的指定。使用glLineStipple()指定。

【函数原型】void glLineStipple(GLint factor, GLushort pattern);

【功能】指定点画模式(线型)。

【参数】

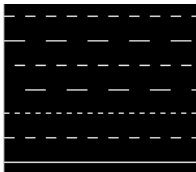
- factor: 线型模式中每位的倍数，值在[1, 255]之间，默认值为1。
- pattern: 位模式，用16位整数指定。位为1时，指定要绘；位为0时，指定不绘。默认时，全部为1（实模式）。位模式从低位开始（如图3-7所示）。



图3-7 位模式0XF0F0

(2) 线型的启用和禁用。使用状态变量GL_LINE_STIPPLE。

- glEnable(GL_LINE_STIPPLE)。激活线型。
- glDisable(GL_LINE_STIPPLE)。关闭线型。



2. 点画线示例

下列程序用于演示各种参数下的线型。运行结果如图3-8所示。

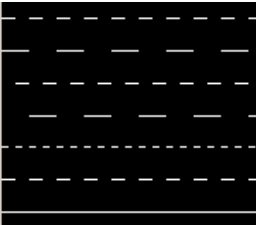
Factor	Pattern	Line
1	0X00FF	
2	0X00FF	
1	0XFF00	
2	0XFF00	
1	0X0F0F	
2	0X0F0F	
1	0XFFFF	

图3-8 各种参数下的线型

```
// LineStipple.c
#include<gl/freeglut.h>

void Line(float x0, float y0, float x1, float y1) // 定义线段
{
    glBegin(GL_LINES); // 定义线段
    {
        glVertex2f(x0, y0);
        glVertex2f(x1, y1);
    }
    glEnd(); // 线段定义结束
}

void Paint() // 场景绘制函数
{
    short stipples[7][2] = // 定义7个点画模式
    {
        {1, 0X00FF}, {2, 0X00FF},
        {1, 0XFF00}, {2, 0XFF00},
        {1, 0X0F0F}, {2, 0X0F0F},
        {1, 0XFFFF} // 倍数, 位模式
    };
    float y = (float)6 / 7; // 最上方线段的y坐标为6/7
    float dy = (float)2 / 7; // 两条线段间的间距为2/7
}
```



```
glClear(GL_COLOR_BUFFER_BIT); // 清除颜色缓冲区  
glEnable(GL_LINE_STIPPLE); // 启用线段点画模式
```

```
for(int i = 0; i < 7; ++i) // 从上到下定义7条线段  
{   glLineStipple(stipples[i][0], stipples[i][1]); // 线段点画模式  
    Line(-1, y, 1, y), y -= dy; // 定义线段, 并调整y坐标  
}
```

```
glDisable(GL_LINE_STIPPLE); // 关闭线段点画模式  
glFlush(); // 强制OpenGL命令序列在有限时间内完成执行  
}
```

```
int main(int argc, char *argv[])  
{   glutInit(&argc, argv); // 初始化GLUT, 记录main()的参数  
    glutInitWindowSize(150, 150); // 程序窗口的初始大小  
    glutCreateWindow("点画线"); // 创建程序窗口, 指定窗口标题  
    glutDisplayFunc(Paint); // 指定场景绘制函数  
    glutMainLoop(); // 开始循环执行OpenGL命令  
}
```

3. 点画多边形的点画模式

(1) 点画模式的指定。可以利用命令glPolygonStipple()指定某种点画模式(图案)来填充多边形内部。

【函数原型】 `void glPolygonStipple(const Glubyte *mask);`

【功能】 指定多边形点画模式。

【参数】 mask用于指定32×32(位)点画模式(位图)的指针，当值为1时绘制；当值为0时不绘制。

(2) 点画模式的启用和禁用。使用状态变量GL_POLYGON_STIPPLE。

- glEnable(GL_POLYGON_STIPPLE)。启用多边形点画模式。
- glDisable(GL_POLYGON_STIPPLE)。关闭多边形点画模式，填充时使用实模式填充。

4. 点画多边形示例

(1) 源程序及运行结果。下列程序绘制了2个三角形区域，第1个三角形使用实模式，第2个三角形使用指定的点画模式。运行结果如图3-9所示。

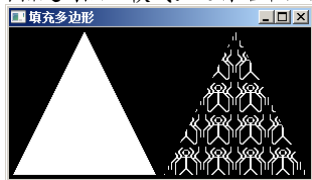


图3-9 一个填充多边形的例子

```
// PolyStipple.c
#include<gl/freeglut.h>
void Hint(); // 当前窗口初始化, 指定点画模式(填充图案), 定义附后
void Triangle() // 定义一个三角形
{
    glBegin(GL_TRIANGLES); // 定义三角形(逆时针指定顶点)
    {
        glVertex2f(-0.95, -0.95); // 左下顶点
        glVertex2f(0.95, -0.95); // 右下顶点
        glVertex2f(0, 0.95); // 上方顶点
    }
    glEnd();
}
```

```
void Paint() // 场景绘制函数
{
    int w = glutGet(GLUT_WINDOW_WIDTH); // 程序窗口宽度
    int h = glutGet(GLUT_WINDOW_HEIGHT); // 程序窗口高度
    glClear(GL_COLOR_BUFFER_BIT); // 清除颜色缓冲区
    glViewport(0, 0, w / 2, h); // 第一个视口，显示第一个三角形
    glDisable(GL_POLYGON_STIPPLE); // 关闭点画模式
    Triangle(); // 第一个三角形
    glViewport(w / 2, 0, w / 2, h); // 第二个视口，显示第二个三角形
    glEnable(GL_POLYGON_STIPPLE); // 启用点画模式
    Triangle(); // 第二个三角形
    glFlush(); // 强制OpenGL命令序列在有限的时间内完成执行
}
```

```
int main(int argc, char *argv[])
{
    glutInit(&argc, argv); // 初始化GLUT, 记录main()的参数
    glutInitWindowSize(300, 150); // 程序窗口的初始大小(2:1)
    glutCreateWindow("填充多边形"); // 创建程序窗口, 指定窗口标题
    Hint(); // 当前窗口初始化, 指定点画模式(填充图案)
    glutDisplayFunc(Paint); // 指定场景绘制函数
    glutMainLoop(); // 开始循环执行OpenGL命令
}
```

```
void Hint() // 当前窗口初始化, 指定点画模式(填充图案)
{   GLubyte pattern[] = // 点画模式的mask值
    {   0X00, 0X00, 0X00, 0X00, /**/ 0X00, 0X00, 0X00, 0X00, // 0, 1
        0X03, 0X80, 0X01, 0XC0, /**/ 0X06, 0XC0, 0X03, 0X60, // 2, 3
        0X04, 0X60, 0X06, 0X20, /**/ 0X04, 0X30, 0X0C, 0X20, // 4, 5
        0X04, 0X18, 0X18, 0X20, /**/ 0X04, 0X0C, 0X30, 0X20, // 6, 7
        0X04, 0X06, 0X60, 0X20, /**/ 0X44, 0X03, 0XC0, 0X22, // 8, 9
        0X44, 0X01, 0X80, 0X22, /**/ 0X44, 0X01, 0X80, 0X22, // 10, 11
        0X44, 0X01, 0X80, 0X22, /**/ 0X44, 0X01, 0X80, 0X22, // 12, 13
        0X44, 0X01, 0X80, 0X22, /**/ 0X44, 0X01, 0X80, 0X22, // 14, 15
        0X66, 0X01, 0X80, 0X66, /**/ 0X33, 0X01, 0X80, 0XCC, // 16, 17
        0X19, 0X81, 0X81, 0X98, /**/ 0X0C, 0XC1, 0X83, 0X30, // 18, 19
        0X07, 0XE1, 0X87, 0XE0, /**/ 0X03, 0X3F, 0XFC, 0XC0, // 20, 21
        0X03, 0X31, 0X8C, 0XC0, /**/ 0X03, 0X33, 0XCC, 0XC0, // 22, 23
        0X06, 0X64, 0X26, 0X60, /**/ 0X0C, 0XCC, 0X33, 0X30, // 24, 25
        0X18, 0XCC, 0X33, 0X18, /**/ 0X10, 0XC4, 0X23, 0X08, // 26, 27
        0X10, 0X63, 0XC6, 0X08, /**/ 0X10, 0X30, 0X0C, 0X08, // 28, 29
        0X10, 0X18, 0X18, 0X08, /**/ 0X10, 0X00, 0X00, 0X08 // 30, 31
    };
    glPolygonStipple(pattern); // 指定多边形的点画模式(填充)
}
```

(2) 填充图案说明。这里以图3-9中右侧三角形中的填充图案为例说明如何给出图案数据。如图3-10所示，图案数据使用“从左至右，自底向上”的顺序依次给出，每次使用8个像素给出一个字节（通常使用16进制数给出），且每个字节的最高位与相应像素组的最先像素对应。

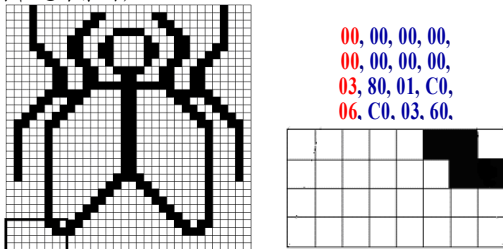


图3-10 构成多边形点画模式

3.5 练习题

3.5.1 基础知识题

1. 请写出OpenGL中指定点宽和线宽的函数，要求写出完整的函数原型。
2. 请使用OpenGL和GLUT编写一个简单的图形程序，用于显示一个填充的白色正方形。其中正方形的左下角顶点是 $(-0.8, -0.8)$ ，右下角顶点是 $(0.8, -0.8)$ ，程序窗口的大小为 $(200, 200)$ ，标题为“白色正方形”。
3. 请使用OpenGL和GLUT编写一个简单的图形程序，用于显示一个填充的红色正三角形。其中正三角形的左下角顶点是 $(-0.5, 0)$ ，右下角顶点是 $(0.5, 0)$ ，程序窗口大小为 $(200, 200)$ ，标题为“红色正三角形”。
4. 请使用OpenGL和GLUT编写一个简单的图形程序，用于显示一个填充的蓝色四边形。其中四边形的4个顶点分别是 $(-0.8, -0.8)$ 、 $(0.5, -0.8)$ 、 $(0.8, 0.8)$ 和 $(-0.5, 0.8)$ ，程序窗口的大小为 $(200, 200)$ ，背景为白色，标题为“蓝色四边形”。
5. 请使用OpenGL和GLUT编写一个简单的图形程序，用于演示点宽。要求使用线段 $(-0.6, -0.6) \sim (0.6, 0.6)$ 上均匀分布的5个点（含端点），点宽为10.5像素，程序窗口的宽度为 $(200, 200)$ ，标题为“点宽”。

6. 请使用OpenGL和GLUT编写一个简单的图形程序，用于演示线宽。其中线段的端点为 $(-0.6, -0.3)$ 和 $(0.6, 0.3)$ ，线宽为4.5像素，程序窗口的大小为 $(200, 200)$ ，标题为“线宽”。

3.5.2 程序设计题

1. 请使用OpenGL、GLU和GLUT编写一个简单的多视口演示程序。要求：(1)在屏幕窗口左侧的1/2部分显示一个红色的填充矩形，该矩形的一对对角顶点是 $(-0.8, -0.8)$ 和 $(0.8, 0.8)$ ；(2)在屏幕窗口右侧的1/2部分显示一个蓝色的填充犹他茶壶，茶壶半径为0.6；(3)程序窗口的大小为 $(400, 200)$ ，背景为黑色，标题为“多视口演示”。

2. 请使用OpenGL、GLU和GLUT编写一个多视口演示程序。要求：(1)在屏幕窗口左下角的1/4部分显示一个红色的填充矩形，该矩形的一对对角顶点是 $(-0.8, -0.8)$ 和 $(0.8, 0.8)$ ；(2)在屏幕窗口右下角的1/4部分显示一个绿色的填充犹他茶壶，茶壶半径为0.6；(3)在屏幕窗口上部居中的1/4部分显示一个蓝色的填充三角形，该三角形的顶点分别是 $(-0.8, -0.8)$ 、 $(0.8, -0.8)$ 和 $(0, 0.8)$ ；(4)程序窗口的大小为 $(200, 200)$ ，背景为黑色，标题为“多视口演示”。

3.5.3 阶段实习题

1. 构造完整的DDA画线算法程序，并对各种情况进行测试。
2. 编制完整的中点画线算法程序，并对各种情况进行测试。
3. 分别绘制2个正方形区域，左边正方形使用实模式绘制，右边正方形选用一个自己喜欢的图案填充。