**OpenGL数据类型**



**GL**

**void glViewport (GLint x, GLint y, GLsizei width, GLsizei height);**

将视口（窗口内部用于绘制裁剪区域的客户区域）的宽度设为w，高度设为h，左下角坐标为(x,y)，单位均为像素。默认情况下，视口为整个初始窗口。

**void glEnable(GLenum cap);**

开启状态变量

**void glDisable(Glenum capability) ;**

关闭状态变量

**Glboolean glIsEnabled(Glenum capability);**

判断状态变量的开关状态

可设置的状态变量如下：

|  |  |  |
| --- | --- | --- |
| 类型 | 值 | 说明 |
| GL\_ALPHA\_TEST | 4864 | 根据函数glAlphaFunc的条件要求来决定图形透明的层度是否显示。具体参见glAlphaFunc |
| GL\_AUTO\_NORMAL | 3456 | 执行后，图形能把光反射到各个方向 |
| GL\_BLEND | 3042 | 启用[颜色混合](http://baike.baidu.com/view/665781.htm)。例如实现半透明效果 |
| GL\_CLIP\_PLANE0 ~ GL\_CLIP\_PLANE5 | 12288 ~ 12283 | 根据函数glClipPlane的条件要求 　　启用图形切割管道。这里指六种缓存管道 |
| GL\_COLOR\_LOGIC\_OP | 3058 | 启用每一像素的色彩为位[逻辑运算](http://baike.baidu.com/subview/397415/397415.htm) |
| GL\_COLOR\_MATERIAL | 2930 | 执行后，图形（材料）将根据光线的照耀进行反射。 　　反射要求由函数[glColorMaterial](http://baike.baidu.com/subview/1280545/1280545.htm)进行设定。 |
| GL\_CULL\_FACE | 2884 | 根据函数[glCullFace](http://baike.baidu.com/subview/1286999/1286999.htm)要求启用隐藏图形材料的面。 |
| GL\_DEPTH\_TEST | 2929 | 启用深度测试。 　　根据坐标的远近自动隐藏被遮住的图形（材料） |
| GL\_DITHER | 3024 | 启用抖动 |
| GL\_FOG | 2912 | 雾化效果 　　例如距离越远越模糊 |
| GL\_INDEX\_LOGIC\_OP | 3057 | 逻辑操作 |
| GL\_LIGHT0 ~ GL\_LIGHT7 | 16384 ~ 16391 | 启用0号灯到7号灯(光源) 　　光源要求由函数glLight函数来完成 |
| GL\_LIGHTING | 2896 | 启用灯源 |
| GL\_LINE\_SMOOTH | 2848 | 执行后，过虑线段的锯齿 |
| GL\_LINE\_STIPPLE | 2852 | 执行后，画虚线 |
| GL\_LOGIC\_OP | 3057 | 逻辑操作 |
| GL\_MAP1\_COLOR\_4 | 3472 | 根据函数Map1对[贝赛尔曲线](http://baike.baidu.com/view/4019466.htm)的设置， 　　启用glEvalCoord1,glEvalMesh1,glEvalPoint1 　　生成[RGBA](http://baike.baidu.com/subview/3731103/3731103.htm)曲线 |
| GL\_MAP1\_INDEX | 3473 | 根据函数Map1对贝赛尔曲线的设置， 　　启用glEvalCoord1,glEvalMesh1,glEvalPoint1 　　生成颜色索引曲线 |
| GL\_MAP1\_NORMAL | 3474 | 根据函数Map1对贝赛尔曲线的设置， 　　启用glEvalCoord1,glEvalMesh1,glEvalPoint1 　　生成[法线](http://baike.baidu.com/subview/357730/357730.htm) |
| GL\_MAP1\_TEXTURE\_COORD\_1 | 3475 | 根据函数Map1对贝赛尔曲线的设置， 　　启用glEvalCoord1,glEvalMesh1,glEvalPoint1 　　生成文理坐标 |
| GL\_MAP1\_TEXTURE\_COORD\_2 | 3476 | 根据函数Map1对贝赛尔曲线的设置， 　　启用glEvalCoord1,glEvalMesh1,glEvalPoint1 　　生成文理坐标 |
| GL\_MAP1\_TEXTURE\_COORD\_3 | 3477 | 根据函数Map1对贝赛尔曲线的设置， 　　启用glEvalCoord1,glEvalMesh1,glEvalPoint1 　　生成文理坐标 |
| GL\_MAP1\_TEXTURE\_COORD\_4 | 3478 | 根据函数Map1对贝赛尔曲线的设置， 　　启用glEvalCoord1,glEvalMesh1,glEvalPoint1 　　生成文理坐标 |
| GL\_MAP1\_VERTEX\_3 | 3479 | 根据函数Map1对贝赛尔曲线的设置， 　　启用glEvalCoord1,glEvalMesh1,glEvalPoint1 　　在三维空间里生成曲线 |
| GL\_MAP1\_VERTEX\_4 | 3480 | 根据函数Map1对贝赛尔曲线的设置， 　　启用glEvalCoord1,glEvalMesh1,glEvalPoint1 　　在[四维空间](http://baike.baidu.com/subview/1152/1152.htm)里生成[法线](http://baike.baidu.com/subview/357730/357730.htm) |
| GL\_MAP2\_COLOR\_4 | 3504 | 根据函数Map2对贝赛尔曲线的设置， 　　启用glEvalCoord2,glEvalMesh2,glEvalPoint2 　　生成[RGBA](http://baike.baidu.com/subview/3731103/3731103.htm)曲线 |
| GL\_MAP2\_INDEX | 3505 | 根据函数Map2对贝赛尔曲线的设置， 　　启用glEvalCoord2,glEvalMesh2,glEvalPoint2 　　生成颜色索引 |
| GL\_MAP2\_NORMAL | 3506 | 根据函数Map2对贝赛尔曲线的设置， 　　启用glEvalCoord2,glEvalMesh2,glEvalPoint2 　　生成[法线](http://baike.baidu.com/subview/357730/357730.htm) |
| GL\_MAP2\_TEXTURE\_COORD\_1 | 3507 | 根据函数Map2对贝赛尔曲线的设置， 　　启用glEvalCoord2,glEvalMesh2,glEvalPoint2 　　生成纹理坐标 |
| GL\_MAP2\_TEXTURE\_COORD\_2 | 3508 | 根据函数Map2对贝赛尔曲线的设置， 　　启用glEvalCoord2,glEvalMesh2,glEvalPoint2 　　生成纹理坐标 |
| GL\_MAP2\_TEXTURE\_COORD\_3 | 3509 | 根据函数Map2对贝赛尔曲线的设置， 　　启用glEvalCoord2,glEvalMesh2,glEvalPoint2 　　生成纹理坐标 |
| GL\_MAP2\_TEXTURE\_COORD\_4 | 3510 | 根据函数Map2对贝赛尔曲线的设置， 　　启用glEvalCoord2,glEvalMesh2,glEvalPoint2 　　生成纹理坐标 |
| GL\_MAP2\_VERTEX\_3 | 3511 | 根据函数Map2对贝赛尔曲线的设置， 　　启用glEvalCoord2,glEvalMesh2,glEvalPoint2 　　在三维空间里生成曲线 |
| GL\_MAP2\_VERTEX\_4 | 3512 | 根据函数Map2对贝赛尔曲线的设置， 　　启用glEvalCoord2,glEvalMesh2,glEvalPoint2 　　在[三维空间](http://baike.baidu.com/view/175999.htm)里生成曲线 |
| GL\_NORMALIZE | 2977 | 根据函数glNormal的设置条件，启用[法向量](http://baike.baidu.com/subview/1486647/1486647.htm) |
| GL\_POINT\_SMOOTH | 2832 | 执行后，过虑线点的锯齿 |
| GL\_POLYGON\_OFFSET\_FILL | 32823 | 根据函数glPolygonOffset的设置，启用面的深度偏移 |
| GL\_POLYGON\_OFFSET\_LINE | 10754 | 根据函数glPolygonOffset的设置，启用线的深度偏移 |
| GL\_POLYGON\_OFFSET\_POINT | 10753 | 根据函数glPolygonOffset的设置，启用点的深度偏移 |
| GL\_POLYGON\_SMOOTH | 2881 | 过虑图形（多边形）的锯齿 |
| GL\_POLYGON\_STIPPLE | 2882 | 执行后，多边形为矢量画图 |
| GL\_SCISSOR\_TEST | 3089 | 根据函数glScissor设置，启用图形剪切 |
| GL\_STENCIL\_TEST | 2960 | 开启使用[模板测试](http://baike.baidu.com/subview/2895466/2895466.htm)并且更新模版缓存。参见[**glStencilFunc**](http://baike.baidu.com/subview/3677254/3677254.htm)和**glStencilOp**. |
| GL\_TEXTURE\_1D | 3552 | 启用一维文理 |
| GL\_TEXTURE\_2D | 3553 | 启用二维文理 |
| GL\_TEXTURE\_GEN\_Q | 3171 | 根据函数glTexGen，启用纹理处理 |
| GL\_TEXTURE\_GEN\_R | 3170 | 根据函数glTexGen，启用纹理处理 |
| GL\_TEXTURE\_GEN\_S | 3168 | 根据函数glTexGen，启用纹理处理 |
| GL\_TEXTURE\_GEN\_T | 3169 | 根据函数glTexGen，启用纹理处理 |

**void glGetDoublev(Glenum pname,Gldouble \*params);**

**glGetFloatv,glGetIntegerf…**

查询状态变量的值

pname:

GL\_POINT\_SIZE\_RANGE 点的最大最小值

GL\_POINT\_SIZE\_GRANULARITY 点的步长

GL\_LINE\_WIDTH \_RANGE 线宽最大最小值

GL\_ LINE\_WIDTH \_GRANULARITY 线宽步长

**void glClear (GLbitfield mask);**

从窗口中清除最后一次所绘制的图形

mask:当前可写的颜色缓冲

GL\_COLOR\_BUFFER\_BIT 颜色缓冲

GL\_DEPTH\_BUFFER\_BIT 深度缓冲

GL\_ACCUM\_BUFFER\_BIT 累积缓冲

GL\_STENCIL\_BUFFER\_BIT 模板缓冲

**void glClearColor (GLclampf red, GLclampf green, GLclampf blue,GLclampf alpha);**

清除颜色缓冲区(背景色)

**void glColor3f (GLfloat red, GLfloat green, GLfloat blue);**

**glColor3b, glColor3d, glColor3f, glColor3i, glColor3s, glColor3ub, glColor3ui, glColor3us, glColor4b, glColor4d, glColor4f, glColor4i, glColor4s, glColor4ub, glColor4ui, glColor4us, glColor3bv, glColor3dv, glColor3fv, glColor3iv, glColor3sv, glColor3ubv, glColor3uiv, glColor3usv, glColor4bv, glColor4dv, glColor4fv, glColor4iv, glColor4sv, glColor4ubv, glColor4uiv, glColor4usv**

设置当前颜色（前景色）

**void glShadeModel ( GLenum mode);**

设定着色模式。

GL\_SMOOTH（默认值） 恒定着色，使用图元中某个顶点的颜色来渲染整个图元。

GL\_FLAT 光滑着色时，独立的处理图元中各个顶点的颜色。对于线段图元，线段上各点的颜色将根据两个顶点的颜色通过插值得到。对于多边形图元，多边形内部区域的颜色将根据所有顶点的颜色插值得到。

**void glPointSize(GLfloat size);**

设定点直径大小(像素)

在glBegin()和glEnd()之外调用，存在有效范围。

glBegin的参数一定要是GL\_POINTS或GL\_POINT,GL\_LINE是不行的。

**void glLineWidth(GLfloat width);**

设定线宽。

放置到begin~end间。

**void glLineStipple(GLint factor,GLushort pattern);**

设定画线的点划模式（画虚线）。需要使用glEnable/glDisable设定GL\_LINE\_STIPPLE。

Pattern是一个16位模式。如果模式中的某一位为1，则直线上相应的像素被绘制，如果为0则相应像素不绘制。Pattern中的1和0的连续组将被绘制factor次，factor的取值范围为1-256.pattern中的位使用是从最低位开始的。

**void glFrontFace(GLenum mode);**

设定多边形绕法（正反面）。

mode:

GL\_CCW(默认) 为逆时针方向的表面为正面。

GL\_CW 顺时针方向的表面为正面。

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

设置多边形的填充模式。

face指定哪面受模式改变的影响：GL\_FRONT，GL\_BACK 或GL\_FRONT\_AND\_BACK。

mode用于指定多边形的绘图模式:GL\_FILL ，GL\_LINE ，GL\_POINT

**void glMatrixMode (GLenum mode);**

设置矩阵模式

mode:

GL\_MODELVIEW,对模型视图矩阵（矩阵变换）堆栈应用随后的矩阵操作.

GL\_PROJECTION,对投影矩阵（裁剪）应用随后的矩阵操作.

GL\_TEXTURE,对纹理矩阵堆栈应用随后的矩阵操作.

**void glLoadIdentity (void);**

读取默认矩阵。

**glLoadMatrix<fd>(type \*m)**

将数组m加载为当前矩阵

**glMultMatrix<fd>(type \*m)**

用矩阵m右乘当前矩阵

**void glPushMatrix (void);**

矩阵入栈。注：glPushMatrix与glPopMatrix必须成对使用。

**void glPopMatrix (void);**

矩阵出栈。

**void glTranslated (GLdouble x, GLdouble y, GLdouble z);**

坐标系平移。需要在矩阵模式为GL\_MODELVIEW模型视图矩阵时使用。

**void glScaled (GLdouble x, GLdouble y, GLdouble z);**

坐标系缩放。需要在矩阵模式为GL\_MODELVIEW模型视图矩阵时使用。

**void glRotatef (GLfloat angle, GLfloat x, GLfloat y, GLfloat z);**

坐标系旋转。需要在矩阵模式为GL\_MODELVIEW模型视图矩阵时使用。

**void gluLookAt(GLdouble eyex, GLdouble eyey, GLdouble eyez, GLdouble atx, GLdouble aty, GLdouble atz, GLdouble upx, GLdouble upy, GLdouble upz)**

设定了一个可为摄像机定位和定向的矩阵，参数分别为摄像机的位置、被观察点的位置以及所期望的up方向

注：需指定矩阵模式为GL\_MODELVIEW。

**void glFrustum(GLdouble left,GLdouble Right,GLdouble bottom,GLdouble top,GLdouble near,GLdouble far);**

它是将当前矩阵与一个透视矩阵相乘，把当前矩阵转变成透视矩阵，在使用它之前，通常会先调用glMatrixMode(GL\_PROJECTION).

透视投影下，视域体必须位于摄像机的前方。所有的参数都是相对于摄像机坐标系中定义的，所以near和far必须为正，且 far>near

**void gluPerspective(GLdouble fovy,GLdouble aspect, GLdouble zNear, GLdouble zFar);**

根据y方向的视角(fovy，单位为度数)，近裁剪平面和远裁剪平面的纵横比以及摄像机到两个裁剪平面的距离指定一个投影矩阵。

注：启用深度测试时，zNear不能为0。

**void glClipPlane(GLenum plane,Const GLdouble \*equation);**

equation指向一个包含四个双精度浮点数的数组，这四个系数分别是裁剪平面Ax+By+Cz+D=0的A、B、C、D值。由这四个系数就能确定一个裁剪平面。

参数plane是符号常数GL\_CLIP\_PLANEi(i=0,1,...)，用于指定裁剪面号。

注：在调用附加裁剪函数之前，必须先启动glEnable(GL\_CLIP\_PLANEi)，使得当前所定义的裁剪平面有效；当不再调用某个附加裁剪平面时，可用glDisable(GL\_CLIP\_PLANEi)关闭相应的附加裁剪功能

**void glBegin(GLenum mode);**

**void glEnd(void);**

glBegin和glEnd函数限定了一组或多组图元的定点定义。

mode：创建图元的类型。可以是以下数值

GL\_POINTS：把每一个顶点作为一个点进行处理，顶点n即定义了点n，共绘制N个点

　　GL\_LINES：把每一个顶点作为一个独立的线段，顶点2n－1和2n之间共定义了n条线段，总共绘制N/2条线段

　　GL\_LINE\_STRIP：绘制从第一个顶点到最后一个顶点依次相连的一组线段，第n和n+1个顶点定义了线段n，总共绘制n－1条线段

　　GL\_LINE\_LOOP：绘制从第一个顶点到最后一个顶点依次相连的一组线段，然后最后一个顶点和第一个顶点相连，第n和n+1个顶点定义了线段n，总共绘制n条线段

　　GL\_TRIANGLES：把每个顶点作为一个独立的三角形，顶点3n－2、3n－1和3n定义了第n个三角形，总共绘制N/3个三角形

　　GL\_TRIANGLE\_STRIP：绘制一组相连的三角形，对于奇数n，顶点n、n+1和n+2定义了第n个三角形；对于偶数n，顶点n+1、n和n+2定义了第n个三角形，总共绘制N-2个三角形

　　GL\_TRIANGLE\_FAN：绘制一组相连的三角形，三角形是由第一个顶点及其后给定的顶点确定，顶点1、n+1和n+2定义了第n个三角形，总共绘制N-2个三角形

　　GL\_QUADS：绘制由四个顶点组成的一组单独的四边形。顶点4n－3、4n－2、4n－1和4n定义了第n个四边形。总共绘制N/4个四边形

　　GL\_QUAD\_STRIP：绘制一组相连的四边形。每个四边形是由一对顶点及其后给定的一对顶点共同确定的。顶点2n－1、2n、2n+2和2n+1定义了第n个四边形，总共绘制N/2-1个四边形

GL\_POLYGON：绘制一个凸多边形。顶点1到n定义了这个多边形。

**void glVertex2f (GLfloat x, GLfloat y);**

**glVertex2d, glVertex2f, glVertex2i, glVertex2s, glVertex3d, glVertex3f, glVertex3i, glVertex3s, glVertex4d, glVertex4f, glVertex4i, glVertex4s, glVertex2dv, glVertex2fv, glVertex2iv, glVertex2sv, glVertex3dv, glVertex3fv, glVertex3iv, glVertex3sv, glVertex4dv, glVertex4fv, glVertex4iv, glVertex4sv**

定义一个点

**void glFlush()**

强制执行之前缓存的所有OpenGL命令。注：在显示模式为GLUT\_SINGLE时使用。

**void glutSwapBuffers();**

交换前后缓冲区。注：在显示模式为GLUT\_DOUBLE时使用。

**GLU**

**void gluOrtho2D(GLdouble left,GLdouble right,GLdouble bottom,GLdouble top );**

指定一个二维矩阵裁剪区域，其左下角坐标为（left,bottom）,右上角坐标为(right,top)。

建立了一个可视的二维平面区域。这个和用glOrtho函数效果是一样的，当glOrtho的near=0，far=1时。

需要在矩阵模式为GL\_PROJECTION投影矩阵时使用。

默认为-1,1,-1,1。

注：需制定矩阵模式为GL\_PROJECTION。

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

这个函数描述了一个平行修剪空间。

摄像机可以位于裁剪体内，所以near和far可以取两个正值或者一正一负 。如near的负值表示摄像机在视景体内。

默认为-1,1,-1,1,-1,1。

**GLUquadricObj \* gluNewQuadric();**

创建一个新的二次曲面对象，并返回其指针。

二次曲面是复杂的工作，涉及到各种数学理论的使用和计算，glu库简化了这一步骤，可以把二次曲面对象理解为一个结构体，保存着各种glu看得懂的信息。

**void gluDeleteQuadric(GLUquadricObj \*obj);**

删除二次曲面对象obj

**void gluQuadricDrawStyle(GLUquadricObj \*obj,GLenum style);**

设置二次曲面绘制风格。一般都是选用GLU\_FILL风格，采用多边形来模拟

style参数： GLU\_POINT、GLU\_LINE、GLU\_FILL、GLU\_SILHOUETTE

**void gluQuadricNormals(GLUquadricObj \*obj, GLenum mode);**

设置法线风格。一般都是使用GLU\_SMOOTH风格，对每个顶点都计算法线向量，是默认方式

mode参数：GL\_NONE、GLU\_FLAT、GLU\_SMOOTH

**gluQuadricOrientation**

设置二次曲面的绘制方向。一般使用GLU\_OUTSIDE, 按照所有的法线都指向外面的方式绘制。是默认方式

**void gluQuadricTexture(GLUquadricObj \*obj,GLboolean mode);**

设置纹理。设置是否自动计算纹理。默认是GLU\_FALSE。当需要使用纹理时应修改为GLU\_TRUE.

mode参数：默认GL\_FALSE、GLU\_TRUE

**void gluSphere(GLUquadricObj \*obj, GLdouble radius, GLint slices,GLint stacks);**

绘制球体，中心位于原点，半径为radius。用slices行经线和stacks行纬线组成的多边形来近似的。

**void gluCylinder(GLUquadricObj \*obj, Gldouble base, Gldouble top, GLdouble height, GLdouble slices, GLdouble Stacks);**

绘制圆柱体。其轴线与z轴平行，底面z=0，半径为base；顶面z=height，半径为top；圆柱沿z轴方向被细分为stacks层，绕着z轴方向被细分为slices段

**void gluDisk(GLUquadricObj \*obj, GLdouble inner, GLdouble outer,GLint slices, GLint rings);**

绘制由inner和outer指定内外径的圆盘。构成圆盘的多边形是由rings个同心圆和围绕圆心的slices次分割所确定。

**void gluPartialDisk(GLUquadricObj \*obj, GLdouble inner, GLdouble outer, GLint slices, GLint rings, GLdouble start, GLdouble angle);**

指定了一个移除了顶角为angle度的圆盘。

**GLUT**

**void glutInit(int \* arg,char \*\* argv);**

初始化GLUT，应在其他GLUT函数和OpenGL函数之前进行调用。

**void glutInitWindowSize(int width,int height);**

设定窗口初始宽/高，单位：像素

**void glutInitWindowPosition(int x,int y);**

设定窗口左上角相对于屏幕左上角的位置，单位：像素

**int glutCreateWindow(char \* name);**

创建一个能够使用OpenGL的窗口。

**void glutInitDisplayMode(unsigned int mode);**

初始化GLUT函数库OpenGL窗口的显示模式。这是基于GLUT的程序在设置OpenGL窗口时必须调用的第一个函数，用以设置窗口特征。



**void glutDisplayFunc(void (\*func) (void));**

为当前窗口设置显示回调窗口。当窗口初次创建、改变大小、被遮挡后重现时发生(需重绘)

**void glutReshapeFunc(void (\* func)(int width,int height));**

为当前窗口指定重设形状回调函数。两个参数为当前窗口的宽度和高度。

注：新建窗口时也会调用一次此函数。

**void glutIdleFunc(void (\*func)(void));**

当事件队列中没有事件需要处理时(即事件队列为空)，该事件被执行。在main中注册空闲回调函数。

**void glutTimerFunc(unsigned int msecs,(\*fun)(int value),int value);**

注册一个回调函数，当指定的毫秒数到期后，这个函数就会被调用。Value参数向这个回调函数传递一个用户指定的值。

注：只调用一次，若想循环调用可在回调函数内在此注册。

**void glutKeyboardFunc(void (\*func)(unsigned char key,int x,int y));**

指定当任何ASCII键被按下时，函数func()被调用。所按下的键及光标位置(屏幕坐标)传入func();

**void glutSpecialFunc(void(\*func)(int key,int x,int y));**

指定当任何特殊键被按下时，函数func()被调用。所按下的键及光标位置(屏幕坐标)传入func();



**int glutGetModifiers()**

该函数在鼠标或键盘回调函数中进行调用。如果键在鼠标或键盘事件产生的同时被按下，则该函数返回GLUT\_ACTIVE\_SHIFT，GLUT\_ACTIVE\_CTRL 或者GLUT\_ACTIVE\_ALT的逻辑与结果。

**void glutMouseFunc(void (\*func)(int button,int state,int x,int y));**

鼠标按下回调函数注册。

button参数：GLUT\_LEFT\_BUTTON 、GLUT\_MIDDLE\_BUTTON、 GLUT\_RIGHT\_BUTTON

state参数：GLUT\_UP、 GLUT\_DOWN

**void glutMotionFunc(void (\*func)(int x,int y));**

鼠标在键被按下的同时发生移动。当鼠标移动一个系统相关的小量时，会引发鼠标移动事件，如绘制曲线等。

**void glutPassiveMotionFunc(void (\*func)(int x,int y));**

鼠标在发生移动时没有被按下。当鼠标移动一个系统相关的小量时，会引发鼠标移动事件，如绘制曲线等。

**void glutEntryFunc(void(\*func)(int state))**

鼠标进入/离开窗口时。state的值为GLUT\_ENTERED或GLUT\_LEFT

**glutIdleFunc(void (\*func)(void));**

不断调用某函数。

注：注销某一回调函数：使用NULL作参数。注：此时需要引入头文件 <windows.h>

**int glutCreateMenu(void (\*func)(int value));**

创建(注册)顶级菜单

创建一个使用回调函数f()的顶级菜单，f()将由菜单项传入一个整数值value。该函数为所创建的菜单返回一个唯一的整形标识符。

**void glutSetMenu(int id);**

改变当前操作的菜单。所创建的菜单称为当前菜单。

**int glutGetMenu()**

获取当前菜单。返回当前菜单的标识符。

**void glutAttachMenu(int button);**

将顶级菜单依附到鼠标键。将当前菜单依附到一个鼠标按钮上。button参数：GLUT\_LEFT\_BUTTON 、GLUT\_MIDDLE\_BUTTON、 GLUT\_RIGHT\_BUTTON。

**void glutAddMenuEntry(char \* name,int value);**

创建菜单项。为当前菜单增加一个名称为name的菜单项；value将被返回给菜单回调函数。

**void glutAddSubMenu(char \* name, int menu);**

创建子菜单。增加一个子菜单项name作为当前菜单的下一项。menu为当该子菜单被创建时所返回的子菜单id。

**void glutMainLoop();**

启动主GLUT事件处理循环。事件循环就是所有的键盘、鼠标、计时器、重绘和其他窗口消息被处理的地方。这个函数在程序终止之前并不会返回。

**void glutPostRedisplay(void);**

通知GLUT刷新（重绘）当前窗口

函数glutPostRedisplay(void)所执行的功能类似于直接调用显示回调函数（如直接调用display()）,但是该函数允许实现在对何时真正需要调用显示回调函数而做出决策时，变得更加“智能”：它可以在遍历消息队列的整个过程中，整合所有的重绘请求，只对窗口重绘一次。

**void glutWireSphere(GLdouble radius, GLint slices, GLint stacks);**

绘制线框球体。

**void glutSolidSphere(GLdouble radius, GLint slices, GLint stacks);**

绘制实心球体。

radius:球体的半径

slices:球体围绕z轴分割的数目

stacks:球体沿着z轴分割的数目

绘制中心在模型坐标原点,半径为radius的球体,球体围绕z轴分割slices次,球体沿着z轴分割stacks次

**void glutWireCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);**

 线框圆锥体

**void glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks);**

实心圆锥体

radius:圆锥体的半径

height:圆锥体的高

slices:圆锥体围绕z轴分割的数目

stacks:圆锥体沿着z轴分割的数目

圆锥体的轴为z轴方向,它的基底位于z=0的平面内,顶点z=height,圆锥体围绕z轴分割slices个数目,沿着z轴分割stacks个数目.

**void glutWireCube(GLdouble size);**

绘制线框立方体

**void glutSolidCube(GLdouble size);**

绘制实心立方体

size:立方体的边长

**void glutWireTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);**

绘制线框圆环

**void glutSolidTorus(GLdouble innerRadius, GLdouble outerRadius, GLint sides, GLint rings);**

绘制实心圆环

innerRadius:圆环的内半径

outerRadius:圆环的外半径

nsides:圆环腔的分割数

rings:圆环的环线数

**void glutWireDodecahedron(void);**

绘制线框十二面体

**void glutSolidDodecahedron(void);**

绘制实心十二面体

**void glutWireTeapot(GLdouble size);**

绘制线框茶壶

**void glutSolidTeapot(GLdouble size);**

 绘制实心茶壶

size:茶壶的相对大小

**void glutWireOctahedron(void);**

绘制线框八面体

**void glutSolidOctahedron(void);**

绘制实心八面体

**void glutWireTetrahedron(void);**

绘制线框四面体

**void glutSolidTetrahedron(void);**

绘制实心四面体

**void glutWireIcosahedron(void);**

 绘制线框二十面体

**void glutSolidIcosahedron(void);**

绘制实心二十面体