第3章　命令和例程一览

本章列出了OpenGL、OpenGL实用库以及对X窗口系统的OpenGL扩展的原型。这些原型按功能分组显示如下：

- OpenGL命令

图元	雾
顶点数组	帧缓冲区操作
坐标转换	求值器
着色与光照	选择与反馈
剪切	显示列表
光栅化	模式与执行
像素操作	状态查询
纹理	

- ARB扩展

多纹理	图形子集

- GLU例程

纹理图形	二次对象
坐标转换	NURBS曲线和曲面
多边形镶嵌分块	状态查询

- GLX例程

初始化	控制绘制操作

3.1　注释

有些OpenGL命令仅在它们所接受的数据类型上存在不同。本书中使用如下的约定来简化地表示这些命令：

void **glVertex2{sifd}{v}** (TYPE *x*, TYPE *y*);

上例中，第一对大括号中所包含的字符用来确定数据 TYPE可能的类型。（其中括号前的阿拉伯数字表示命令所包含的自变量的个数。）表3-1列出了每种可能的数据类型、其相应的字符以及OpenGL中用来表示它们的类型定义。

命令中如果包含第二对大括号，则其中的字母 v表示命令中包含向量形式。当你选择使用向量形式时，所有的TYPE参数将合并为一个数组。例如， 下面是一个命令的非向量形式和向量形式，它们都使用一个浮点数据类型：

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

void **glVertex2fv**(GLfloat *v*[2]);

在向量形式的用法是不确定的时候，向量和非向量的形式都被列出。请注意并非所有多变量

的命令都有向量形式，而有些命令就只有一个向量形式，这时字母 v将不包含在大括号中。

表 3-1

字 符	C语言类型	OpenGL类型定义
b	signed char	GLbyte
s	short	GLshort
i	int	GLint
f	float	GLfloat, GLclampf
d	double	GLdouble, GLclampd
ub	unsigned char	GLubyte, GLboolean
us	unsighed short	GLshort
ui	unsigned int	GLuint, GLenum, GLbitfield
	void	GLvoid

3.2 OpenGL命令

3.2.1 图元

指定顶点或矩形:

void **glBegin** (GLenum *mode*);
void **glEnd** (void);
void **glVertex2{sifd}{v}** (TYPE *x*, TYPE *y*);
void **glVertex3{sifd}{v}** (TYPE *x*, TYPE *y*, TYPE *z*);
void **glVertex4{sifd}{v}** (TYPE *x*, TYPE *y*, TYPE *z*, TYPE *w*);
void **glRect{sifd}** (TYPE *x1*, TYPE *y1*, TYPE *x2*, TYPE *y2*);
void **glRect{sifd}v** (const TYPE **v1*, const TYPE **v2*);

指定多边形边界的处理方式：

void **glEdgeFlag** (GLboolean *flag*);
void **glEdgeFlagv** (const GLboolean **flag*);

指定多边形的偏移量：

void **glPolygonOffset** (GLfloat *factor*, GLfloat *units*);

3.2.2 顶点数组

指定顶点数组：

void **glVertexPointer** (GLint *size*, GLenum *type*, GLsizei *stride*,
const GLvoid **pointer*);
void **glEdgeFlagPointer** (GLsizei *stride*, const GLvoid **pointer*);
void **glIndexPointer** (GLenum *type*, GLsizei *stride*,
const GLvoid **pointer*);
void **glColorPointer** (GLint *size*, GLenum *type*, GLsizei *stride*,
const GLvoid **pointer*);
void **glNormalPointer** (GLenum *type*, GLsizei *stride*,
const GLvoid **pointer*);

void **glTexCoordPointer** (GLint *size*, GLenum *type*, GLsizei *stride*,
const GLvoid **pointer*);

控制顶点数组及其组分的绘制：

void **glInterleavedArrays** (GLenum *format*, GLsizei *stride*,
const GLvoid **pointer*);
void **glArrayElement** (GLint *i*);
void **glDisableClientState** (GLenum *array*);
void **glEnableClientState** (GLenum *array*);
void **glDrawElements** (GLenum *mode*, GLsizei *count*, GLenum *type*,
const GLvoid **indices*);
void **glDrawRangeElements** (GLenum *mode*, GLuint *start*, GLuint *end*,
GLsizei *count*, GLenum *type*, const GLvoid **indices*);
void **glDrawArrays** (GLenum *mode*, GLint *first*, GLsizei *count*);

存储与恢复顶点数组的值：

void **glPushClientAttrib** (GLbitfield *mask*);
void **glPopClientAttrib** (void);

获取指定的顶点数组的地址：

void **glGetPointerv** (GLenum *pname*, GLvoid ***params*);

3.2.3 坐标转换

转换当前矩阵：

void **glRotate{fd}** (TYPE *angle*, TYPE *x*, TYPE *y*, TYPE *z*);
void **glTranslate{fd}** (TYPE *x*, TYPE *y*, TYPE *z*);
void **glScale{fd}** (TYPE *x*, TYPE *y*, TYPE *z*);
void **glMultMatrix{fd}** (const TYPE **m*);
void **glFrustum** (GLdouble *left*, GLdouble *right*, GLdouble *bottom*,
GLdouble *top*, GLdouble *near*, GLdouble *far*);
void **glOrtho** (GLdouble *left*, GLdouble *right*, GLdouble *bottom*,
GLdouble *top*, GLdouble *near*, GLdouble *far*);

替换当前矩阵：

void **glLoadMatrix{fd}** (const TYPE **m*);
void **glLoadIdentity** (void);

操纵矩阵堆栈：

void **glMatrixMode** (GLenum *mode*);
void **glPushMatrix** (void);
void **glPopMatrix** (void);

指定视口：

void **glDepthRange** (GLclampd *near*, GLclampd *far*);
void **glViewport** (GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*);

3.2.4 着色与光照

设置当前颜色、颜色索引或法向量：

void **glColor3{bsifd ubusui}{v}** (TYPE *red*, TYPE *green*, TYPE *blue*);
void **glColor4{bsifd ubusui}{v}** (TYPE *red*, TYPE *green*, TYPE *blue*,
TYPE *alpha*);

指定光源、材料或光照模式参数值：

void **glLight{if}{v}** (GLenum *light*, GLenum *pname*, TYPE *param*);
void **glMaterial{if}{v}** (GLenum *face*, GLenum *pname*, TYPE *param*);
void **glLightModel{if}{v}** (GLenum *pname*, TYPE *param*);

选择一种阴影模式：

void **glShadeModel** (GLenum *mode*);

指定正面多边形：

void **glFrontFace** (GLenum *dir*);

使一种颜色跟踪当前颜色：

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

获取光源或材料的参数值：

void **glGetLight{if}v** (GLenum *light*, GLenum *pname*, TYPE **params*);
void **glGetMaterial{if}v** (GLenum *face*, GLenum *pname*, TYPE **params*);

3.2.5 剪切

指定一个剪切平面：

void **glClipPlane** (GLenum *plane*, const GLdouble **equation*);

返回剪切平面系数：

void **glGetClipPlane** (GLenum *plane*, GLdouble **equation*);

3.2.6 光栅化

设置当前光栅位置：

void **glRasterPos2{sifd}{v}**(TYPE *x*, TYPE *y*);
void **glRasterPos3{sifd}{v}**(TYPE *x*, TYPE *y*, TYPE *z*);
void **glRasterPos4{sifd}{v}**(TYPE *x*, TYPE *y*, TYPE *z*, TYPE *w*);

指定一个位图：

void **glBitmap** (GLsizei *width*, GLsizei *height*, GLfloat *xorig*, GLfloat *yorig*,
GLfloat *xmove*, GLfloat *ymove*, const GLubyte **bitmap*);

指定点或线的尺寸：

void **glPointSize** (GLfloat *size*);
void **glLineWidth** (GLfloat *width*);

指定或返回线或多边形的点画模式：

void **glLineStipple** (GLint *factor*, GLushort *pattern*);
void **glPolygonStipple** (const GLubyte **mask*);
void **glGetPolygonStipple** (GLubyte **mask*);

选择如何光栅化多边形：

void **glCullFace** (GLenum *mode*);

3.2.7 像素操作

选择像素读取或复制操作的源：

void **glReadBuffer** (GLenum *mode*);

读、写或复制像素：

void **glReadPixels** (GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*,
GLenum *format*, GLenum *type*, GLvoid **pixels*);
void **glDrawPixels** (GLsizei *width*, GLsizei *height*, GLenum *format*,
GLenum *type*, const GLvoid **pixels*);
void **glCopyPixels** (GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*,
GLenum *type*);

指定或查询像素的编码或处理方式：

void **glPixelStore{if}** (GLenum *pname*, TYPE *param*);
void **glPixelTransfer{if}** (GLenum *pname*, TYPE *param*);
void **glPixelMap{f usui}v** (GLenum *map*, GLint *mapsize*, const TYPE **values*);
void **glGetPixelMap{f usui}v** (GLenum *map*, TYPE **values*);

控制像素的光栅化：

void **glPixelZoom** (GLfloat *xfactor*, GLfloat *yfactor*);

3.2.8 纹理

控制如何将一个纹理应用于片断：

void **glTexParameter{if}{v}** (GLenum *target*, GLenum *pname*, TYPE *param*);
void **glTexEnv{if}{v}** (GLenum *target*, GLenum *pname*, TYPE *param*);

设置当前纹理坐标：

void **glTexCoord1{sifd}{v}** (TYPE *s*);
void **glTexCoord2{sifd}{v}** (TYPE *s*, TYPE *t*);
void **glTexCoord3{sifd}{v}** (TYPE *s*, TYPE *t*, TYPE *r*);
void **glTexCoord4{sifd}{v}** (TYPE *s*, TYPE *t*, TYPE *r*, TYPE *q*);

控制纹理坐标的生成：

void **glTexGen{ifd}{v}** (GLenum *coord*, GLenum *pname*, TYPE *param*);

指定一个一维或二维的纹理图像或纹理子图：

void **glTexImage1D** (GLenum *target*, GLint *level*, GLint *internalformat*,
GLsizei *width*, GLint *border*, GLenum *format*,
GLenum *type*, const GLvoid **pixels*);
void **glTexImage2D** (GLenum *target*, GLint *level*, GLint *internalformat*,
GLsizei *width*, GLsizei *height*, GLint *border*,
GLenum *format*, GLenum *type*, const GLvoid **pixels*);
void **glTexImage3D** (GLenum *target*, GLint *level*, GLenum *internalformat*,
GLsizei *width*, GLsizei *height*, GLsizei *depth*, GLint *border*,

GLenum *format*, GLenum *type*, const GLvoid **pixels*);
void **glTexSubImage1D** (GLenum *target*, GLint *level*, GLint *xoffset*,
GLsizei *width*, GLenum *format*, GLenum *type*,
const GLvoid **pixels*);
void **glTexSubImage2D** (GLenum *target*, GLint *level*, GLint *xoffset*,
GLint *yoffset*, GLsizei *width*, GLsizei *height*,
GLenum *format*, GLenum *type*, const GLvoid **pixels*);
void **glTexSubImage3D** (GLenum *target*, GLint *level*, GLint *xoffset*,
GLint *yoffset*, GLint *zoffset*, GLsizei *width*, GLsizei *height*,
GLsizei *depth*, GLenum *format*, GLenum *type*,
const GLvoid **pixels*);

测试一个名称是否对应于一个纹理并获取与纹理有关的参数值：

void **glIsTexture** (GLuint *texture*);
void **glGetTexEnv{if}v** (GLenum *target*, GLenum *pname*, TYPE **params*);
void **glGetTexGen{ifd}v** (GLenum *coord*, GLenum *pname*, TYPE **params*);
void **glGetTexImage** (GLenum *target*, GLint *level*, GLenum *format*,
GLenum *type*, GLvoid **pixels*);
void **glGetTexLevelParameter{if}v** (GLenum *target*, GLint *level*,
GLenum *pname*, TYPE **params*);
void **glGetTexParameter{if}v** (GLenum *target*, GLenum *pname*, TYPE **params*);

复制一个或部分纹理：

void **glCopyTexImage1D** (GLenum *target*, GLint *level*,
GLenum *internalformat*, GLint *x*, GLint *y*, GLsizei *v*,
GLint *border*);
void **glCopyTexImage2D** (GLenum *target*, GLint *level*,
GLenum *internalformat*, GLint *x*, GLint *y*,
GLsizei *width*, GLsizei *height*, GLint *border*);
void **glCopyTexSubImage1D** (GLenum *target*, GLint *level*, GLint *xoffset*,
GLint *x*, GLint *y*, GLsizei *width*);
void **glCopyTexSubImage2D** (GLenum *target*, GLint *level*, GLint *xoffset*,
GLint *yoffset*, GLint *x*, GLint *y*, GLsizei *width*,
GLsizei *height*);
void **glCopyTexSubImage3D** (GLenum *target*, GLint *level*, GLint *xoffset*,
GLint *yoffset*, GLint *zoffset*, GLint *x*, GLint *y*,
GLsizei *width*, GLsizei *height*);

建立一个指定的纹理并优化纹理存储驻留：

void **glBindTexture** (GLenum *target*, GLuint *texture*);
void **glDeleteTextures** (GLsizei *n*, const GLuint **textures*);
GLboolean **glAreTexturesResident** (GLsizei *n*, const GLuint **textures*,
GLboolean **residences*);
void **glGenTextures** (GLsizei *n*, GLuint **textures*);
void **glPrioritizeTextures** (GLsizei *n*, const GLuint **textures*,
const GLclampf **priorities*);

3.2.9 雾

设置雾参数：

void **glFog{if}{v}** (GLenum *pname*, TYPE *param*);

3.2.10 帧缓冲区操作

控制每个片断的测试：

void **glScissor** (GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*);
void **glAlphaFunc** (GLenum *func*, GLclampf *ref*);
void **glStencilFunc** (GLenum *func*, GLint *ref*, GLuint *mask*);
void **glStencilOp** (GLenum *fail*, GLenum *pass*, GLenum *zpass*);
void **glDepthFunc** (GLenum *func*);

结合片断与帧缓冲区中的值：

void **glBlendFunc** (GLenum *sfactor*, GLenum *dfactor*);
void **glLogicOp** (GLenum *opcode*);

清除部分或全部缓冲区：

void **glClear** (GLbitfield *mask*);

指定清除操作所需的颜色、深度和模板值：

void **glClearAccum** (GLfloat *red*, GLfloat *green*, GLfloat *blue*, GLfloat *alpha*);
void **glClearColor** (GLclampf *red*, GLclampf *green*, GLclampf *blue*,
GLclampf *alpha*);
void **glClearDepth** (GLclampd *depth*);
void **glClearIndex** (GLfloat *c*);
void **glClearStencil** (GLint *s*);

控制向缓冲区中写入：

void **glDrawBuffer** (GLenum *mode*);
void **glIndexMask** (GLuint *mask*);
void **glColorMask** (GLboolean *red*, GLboolean *green*, GLboolean *blue*,
GLboolean *alpha*);
void **glDepthMask** (GLboolean *flag*);
void **glStencilMask** (GLuint *mask*);

操作累积缓冲区：

void **glAccum** (GLenum *op*, GLfloat *value*);

3.2.11 求值器

定义一个一维或二维求值器：

void **glMap1{fd}** (GLenum *target*, TYPE *u1*, TYPE *u2*, GLint *stride*,
GLint *order*, const TYPE **points*);
void **glMap2{fd}** (GLenum *target*, TYPE *u1*, TYPE *u2*, GLint *ustride*,
GLint *uorder*, TYPE *v1*, TYPE *v2*, GLint *vstride*,
GLint *vorder*, const TYPE **points*);

生成并求取一系列映射的域值：

void **glMapGrid1{fd}** (GLint *n*, TYPE *u1*, TYPE *u2*);
void **glMapGrid2{fd}** (GLint *un*, TYPE *u1*, TYPE *u2*, GLint *vn*, TYPE *v1*,
TYPE *v2*);
void **glEvalMesh1** (GLenum *mode*, GLint *i1*, GLint *i2*);
void **glEvalMesh2** (GLenum *mode*, GLint *i1*, GLint *i2*, GLint *j1*, GLint *j2*);
void **glEvalPoint1** (GLint *i*);
void **glEvalPoint2** (GLint *i*, GLint *j*);

在一个指定的域坐标中求取一维或二维映射值：

void **glEvalCoord1{fd}{v}** (TYPE *u*);
void **glEvalCoord2{fd}{v}** (TYPE *u*, TYPE *v*);

获取求值器的参数值：

void **glGetMap{idf}v** (GLenum *target*, GLenum *query*, TYPE **v*);

3.2.12 选择与反馈

控制模式及相应的缓冲区：

GLint **glRenderMode** (GLenum *mode*);
void **glSelectBuffer** (GLsizei *size*, GLuint **buffer*);
void **glFeedbackBuffer** (GLsizei *size*, GLenum *type*, GLfloat **buffer*);

为反馈模式提供一个标记：

void **glPassThrough** (GLfloat *token*);

控制选取模式中的名称堆栈：

void **glInitNames** (void);
void **glLoadName** (GLuint *name*);
void **glPushName** (GLuint *name*);
void **glPopName** (void);

3.2.13 显示列表

建立或删除显示列表：

void **glNewList** (GLuint *list*, GLenum *mode*);
void **glEndList** (void);
void **glDeleteLists** (GLuint *list*, GLsizei *range*);

执行一个或一组显示列表：

void **glCallList** (GLuint *list*);
void **glCallLists** (GLsizei *n*, GLenum *type*, const GLvoid **lists*);

管理显示列表的索引：

GLuint **glGenLists** (GLsizei *range*);
GLboolean **glIsList** (GLuint *list*);
void **glListBase** (GLuint *base*);

3.2.14 模式与执行

启动、关闭及查询模式：

void **glEnable** (GLenum *cap*);
void **glDisable** (GLenum *cap*);
GLboolean **glIsEnabled** (GLenum *cap*);

等待，直到所有OpenGL命令执行完成：

void **glFinish** (void);

强迫执行所有已发布的OpenGL命令：

void **glFlush** (void);

指定OpenGL操作的提示：

void **glHint** (GLenum *target*, GLenum *mode*);

3.2.15 状态查询

获取有关错误及OpenGL当前连接的信息：

GLenum **glGetError** (void);
const GLubyte * **glGetString** (GLenum *name*);

查询状态变量：

void **glGetBooleanv** (GLenum *pname*, GLboolean **params*);
void **glGetDoublev** (GLenum *pname*, GLdouble **params*);
void **glGetFloatv** (GLenum *pname*, GLfloat **params*);
void **glGetIntegerv** (GLenum *pname*, GLint **params*);

存储与恢复状态变量集：

void **glPushAttrib** (GLbitfield *mask*);
void **glPopAttrib** (void);

3.3 ARB扩展

3.3.1 多重纹理

设置当前纹理坐标：

void **glMultiTexCoord1{sifd}{v}ARB** (GLenum *target*, TYPE *s*);
void **glMultiTexCoord2{sifd}{v}ARB** (GLenum *target*, TYPE *s*, TYPE *t*);
void **glMultiTexCoord3{sifd}{v}ARB** (GLenum *target*, TYPE *s*, TYPE *t*, TYPE *r*);
void **glMultiTexCoord4{sifd}{v}ARB** (GLenum *target*, TYPE *s*, TYPE *t*,
TYPE *r*, TYPE *q*);

选择当前有效的纹理单元：

void **glActiveTextureARB** (GLenum *texture*);

选择当前有效的顶点数组：

void **glClientActiveTextureARB** (GLenum *texture*);

3.3.2 绘图子集

1. 颜色表

指定一个颜色查询表或子表：

void **glColorTable** (GLenum *target*, GLenum *internalformat*, GLsizei *width*,
GLenum *format*, GLenum *type*, const GLvoid **table*);
void **glColorSubTable** (GLenum *target*, GLsizei *start*, GLsizei *count*,
GLenum *format*, GLenum *type*, const GLvoid **data*);

复制一个颜色查询表或子表：

void **glCopyColorTable** (GLenum *target*, GLenum *internalformat*, GLint *x*, GLint *y*, GLsizei *width*);
void **glCopyColorSubTable** (GLenum *target*, GLsizei *start*, GLint *x*, GLint *y*, GLsizei *width*);

获取颜色表的相应值：

void **glGetColorTable** (GLenum *target*, GLenum *format*, GLenum *type*, GLvoid **table*);
void **glGetColorTableParameter{if}v** (GLenum *target*, GLenum *pname*, TYPE **params*);

2. 卷积

控制如何将一个卷积滤波器应用于输入的图像：

void **glConvolutionParameter{if}{v}** (GLenum *target*, GLenum *pname*, TYPE *params*);

指定一维或二维卷积滤波器：

void **glConvolutionFilter1D** (GLenum *target*, GLenum *internalformat*, GLsizei *width*, GLenum *format*, GLenum *type*, const GLvoid **image*);
void **glConvolutionFilter2D** (GLenum *target*, GLenum *internalformat*, GLsizei *width*, GLsizei *height*, GLenum *format*, GLenum *type*, const GLvoid **image*);

指定二维可分离卷积滤波器：

void **glSeparableFilter2D** (GLenum *target*, GLenum *internalformat*, GLsizei *width*, GLsizei *height*, GLenum *format*, GLenum *type*, const GLvoid **row*, const GLvoid **column*);

获取卷积的相关参数值：

void **glGetConvolutionFilter** (GLenum *target*, GLenum *format*, GLenum *type*, GLvoid **image*);
void **glGetConvolutionParameter{if}v** (GLenum *target*, GLenum *pname*, TYPE **params*);
void **glGetSeparableFilter** (GLenum *target*, GLenum *format*, GLenum *type*, GLvoid **row*, GLvoid **column*, GLvoid **span*);

复制一个或部分卷积滤波器：

void **glCopyConvolutionFilter1D** (GLenum *target*, GLenum *internalformat*, GLint *x*, GLint *y*, GLsizei *width*);
void **glCopyConvolutionFilter2D** (GLenum *target*, GLenum *internalformat*, GLint *x*, GLint *y*, GLsizei *width*, GLsizei *height*);

3. 直方图

指定直方图格式：

void **glHistogram** (GLenum *target*, GLsizei *width*, GLenum *internalformat*, GLboolean *sink*);

获取直方图的值与参数：

void **glGetHistogram** (GLenum *target*, GLboolean *reset*, GLenum *format*,
GLenum *type*, GLvoid **values*);
void **glGetHistogramParameter{if}v** (GLenum *target*, GLenum *pname*,
GLfloat **params*);

重新设置内部直方图表：

void **glResetHistogram** (GLenum *target*);

4. 最值

指定最值格式：

void **glMinmax** (GLenum *target*, GLenum *internalformat*, GLboolean *sink*);

获取最值及参数：

void **glGetMinmax** (GLenum *target*, GLboolean *reset*, GLenum *format*,
GLenum *type*, GLvoid **values*);
void **glGetMinmaxParameter{if}v** (GLenum *target*, GLenum *pname*,
GLfloat **params*);

重新设置内部最值表：

void **glResetMinmax** (GLenum *target*);

3.4 GLU例程

3.4.1 纹理图像

缩放一个图像：

int **gluScaleImage** (GLenum *format*, GLint *widthin*, GLint *heightin*,
GLenum *typein*, const void **datain*, GLint *widthout*,
GLint *heightout*, GLenum *typeout*, void **dataout*);

生成一个图像的mipmap**：**

int **gluBuild1DMipmaps** (GLenum *target*, GLint *internalformat*, GLint *width*,
GLenum *format*, GLenum *type*, const void **data*);
int **gluBuild2DMipmaps** (GLenum *target*, GLint *internalformat*, GLint *width*,
GLint *height*, GLenum *format*, GLenum *type*,
const void **data*);
int **gluBuild3DMipmaps** (GLenum *target*, GLint *internalformat*, GLsizei *width*,
GLsizei *height*, GLsizei *depth*, GLenum *format*,
GLenum *type*, const void **data*);

生成一个图像的mipmap**图层的范围：**

int **gluBuild1DMipmapLevels** (GLenum *target*, GLint *internalformat*,
GLsizei *width*, GLenum *format*, GLenum *type*,
GLint *level*, GLint *base*, GLint *max*, const void **data*);
int **gluBuild2DMipmapLevels** (GLenum *target*, GLint *internalformat*,
GLsizei *width*, GLsizei *height*, GLenum *format*,
GLenum *type*, GLint *level*, GLint *base*, GLint *max*,
const void **data*);

int **gluBuild3DMipmapLevels** (GLenum *target*, GLint *internalformat*,
GLsizei *width*, GLsizei *height*, GLsizei *depth*,
GLenum *format*, GLenum *type*, GLint *level*,
GLint *base*, GLint *max*, const void **data*);

3.4.2 坐标转换

建立投影或观察矩阵：

void **gluOrtho2D** (GLdouble *left*, GLdouble *right*, GLdouble *bottom*,
GLdouble *top*);
void **gluPerspective** (GLdouble *fovy*, GLdouble *aspect*, GLdouble *zNear*,
GLdouble *zFar*);
void **gluPickMatrix** (GLdouble *x*, GLdouble *y*, GLdouble *width*,
GLdouble *height*, GLint *viewport*[4]);
void **gluLookAt** (GLdouble *eyex*, GLdouble *eyey*, GLdouble *eyez*,
GLdouble *centerx*, GLdouble *centery*, GLdouble *centerz*,
GLdouble *upx*, GLdouble *upy*, GLdouble *upz*);

将对象坐标转换成屏幕坐标：

int **gluProject** (GLdouble *objx*, GLdouble *objy*, GLdouble *objz*,
const GLdouble *modelMatrix*[16],
const GLdouble *projMatrix*[16],
const GLint *viewport*[4], GLdouble **winx*,
GLdouble **winy*, GLdouble **winz*);
int **gluUnProject** (GLdouble *winx*, GLdouble *winy*, GLdouble *winz*,
const GLdouble *modelMatrix*[16],
const GLdouble *projMatrix*[16],
const GLint *viewport*[4], GLdouble **objx*,
GLdouble **objy*, GLdouble **objz*);
int **gluUnProject4** (GLdouble *winx*, GLdouble *winy*, GLdouble *winz*,
GLdouble *clipw*, const GLdouble *modelMatrix[16]*,
const GLdouble *projMatrix[16]*,
const GLint *viewport[4]*, GLdouble *near*,
GLdouble *far*, GLdouble* *objx*, GLdouble* *objy*,
GLdouble* *objz*, GLdouble **objw*);

3.4.3 多边形镶嵌分块

管理镶嵌分块对象：

GLUtesselator* **gluNewTess** (void);
void **gluTessCallback** (GLUtesselator **tobj*, GLenum *which*, void (**fn*)());
void **gluDeleteTess** (GLUtesselator **tobj*);
void **gluGetTessProperty** (GLUtesselator* *tess*, GLenum *which*,
GLdouble* *data*);

描述输入的多边形：

void **gluTessBeginPolygon** (GLUtesselator **tobj*);
void **gluTessEndPolygon** (GLUtesselator **tobj*);
void **gluTessBeginContour** (GLUtesselator **tess*);

void **gluTessEndContour** (GLUtesselator *tess);
void **gluTessVertex** (GLUtesselator *tobj, GLdouble v[3], void *data);
void **gluTessNormal** (GLUtesselator *tess, GLdouble valuex, GLdouble valuey,
GLdouble valuez);
void **gluTessProperty** (GLUtesselator *tess, GLenum which, GLdouble data);

3.4.4 二次对象

管理二次对象：

GLUquadric* **gluNewQuadric** (void);
void **gluDeleteQuadric** (GLUquadric *state);
void **gluQuadricCallback** (GLUquadric *qobj, GLenum which, void (*fn)());

控制绘图操作：

void **gluQuadricNormals** (GLUquadric *quadObject, GLenum normals);
void **gluQuadricTexture** (GLUquadric *quadObject,
GLboolean textureCoords);
void **gluQuadricOrientation** (GLUquadric *quadObject,
GLenum orientation);
void **gluQuadricDrawStyle** (GLUquadric *quadObject, GLenum drawStyle);

指定一个二次图元：

void **gluCylinder** (GLUquadric *qobj, GLdouble baseRadius,
GLdouble topRadius, GLdouble height, GLint slices,
GLint stacks);
void **gluDisk** (GLUquadric *qobj, GLdouble innerRadius,
GLdouble outerRadius, GLint slices, GLint loops);
void **gluPartialDisk** (GLUquadric *qobj, GLdouble innerRadius,
GLdouble outerRadius, GLint slices, GLint loops,
GLdouble startAngle, GLdouble sweepAngle);
void **gluSphere** (GLUquadric *qobj, GLdouble radius, GLint slices,
GLint stacks);

3.4.5 NURBS曲线和曲面

管理一个NURBS对象：

GLUnurbs* **gluNewNurbsRenderer** (void);
void **gluDeleteNurbsRenderer** (GLUnurbs *nobj);
void **gluNurbsCallback** (GLUnurbs *nobj, GLenum which, void (*fn)());
void **gluNurbsCallbackData** (GLUnurbs *nurb, GLvoid *userData);

创建一条NURBS曲线：

void **gluBeginCurve** (GLUnurbs *nobj);
void **gluEndCurve** (GLUnurbs *nobj);
void **gluNurbsCurve** (GLUnurbs *nobj, GLint nknots, GLfloat *knot,
GLint stride, GLfloat *ctlarray,
GLint order, GLenum type);

创建一个NURBS曲面：

```
void gluBeginSurface (GLUnurbs *nobj);
void gluEndSurface (GLUnurbs *nobj);
void gluNurbsSurface (GLUnurbs *nobj, GLint uknot_count,
                      GLfloat *uknot, GLint vknot_count, GLfloat *vknot,
                      GLint u_stride, GLint v_stride, GLfloat *ctlarray,
                      GLint sorder, GLint torder, GLenum type);
```

定义一个修剪区域：

```
void gluBeginTrim (GLUnurbs *nobj);
void gluEndTrim (GLUnurbs *nobj);
void gluPwlCurve (GLUnurbs *nobj, GLint count, GLfloat *array,
                  GLint stride, GLenum type);
```

控制NURBS绘图：

```
void gluLoadSamplingMatrices (GLUnurbs *nobj,
                              const GLfloat modelMatrix[16],
                              const GLfloat projMatrix[16],
                              const GLint viewport[4]);
void gluNurbsProperty (GLUnurbs *nobj, GLenum property, GLfloat value);
void gluGetNurbsProperty (GLUnurbs *nobj, GLenum property,
                          GLfloat *value);
```

3.4.6 状态查询

由一个OpenGL出错代码生成一个出错字符串或描述GLU的版本或扩展：

```
const GLubyte* gluErrorString (GLenum errorCode);
const GLubyte* gluGetString (GLenum name);
GLboolean gluCheckExtension (const GLubyte *extName,
                             const GLubyte *extString);
```

3.5 GLX例程

3.5.1 初始化

确定GLX扩展是否已经在X服务器上被定义：

```
Bool glXQueryExtension (Display *dpy, int *errorBase, int *eventBase);
Bool glXQueryVersion (Display *dpy, int *major, int *minor);
```

获取所需的视觉环境信息：

```
XVisualInfo* glXChooseVisual (Display *dpy, int screen, int *attribList);
int glXGetConfig (Display *dpy, XVisualInfo *vis, int attrib, int *value);
```

查询服务器端或客户端所支持的特性：

3.5.2 控制绘图操作

管理或查询一个OpenGL绘图环境：

GLXContext **glXCreateContext** (Display **dpy*, XVisualInfo **vis*,
GLXContext *shareList*, Bool *direct*);
void **glXDestroyContext** (Display **dpy*, GLXContext *ctx*);
void **glXCopyContext** (Display **dpy*, GLXContext *src*,
GLXContext *dst*, GLuint *mask*);
Bool **glXIsDirect** (Display **dpy*, GLXContext *ctx*);
Bool **glXMakeCurrent** (Display **dpy*, GLXDrawable *draw*, GLXContext *ctx*);
GLXContext **glXGetCurrentContext** (void);
GLXDrawable **glXGetCurrentDrawable** (void);

执行屏幕外的绘制：

GLXPixmap **glXCreateGLXPixmap** (Display **dpy*, XVisualInfo **vis*,
Pixmap *pixmap*);
void **glXDestroyGLXPixmap** (Display **dpy*, GLXPixmap *pix*);

同步执行：

void **glXWaitGL** (void);
void **glXWaitX** (void);

交换前后缓冲区：

void **glXSwapBuffers** (Display **dpy*, Window *window*);

使用一种X字体：

void **glXUseXFont** (Font *font*, int *first*, int *count*, int *listBase*);

注：在GLX1.3机制中，下面的命令为前面所列的命令提供了一组高级功能。

查询或请求关于帧缓冲区或视觉环境的信息：

GLXFBConfig* **glXGetFBConfigs** (Display **dpy*, int *screen*, int **nelements*);
GLXFBConfig* **glXChooseFBConfig** (Display **dpy*, int *screen*,
const int **attribList*, int **nelements*);
int **glXGetFBConfigAttrib** (Display **dpy*, GLXFBConfig *config*, int *attribute*,
int **value*);
XVisualInfo* **glXGetVisualFromFBConfig** (Display **dpy*, GLXFBConfig
config);

管理GLX的可绘区域：

GLXWindow **glXCreateWindow** (Display **dpy*, GLXFBConfig *config*,
Window *window*, const int **attribList*);
void **glXDestroyWindow** (Display **dpy*, GLXWindow *window*);
GLXPixmap **glXCreatePixmap** (Display **dpy*, GLXFBConfig *config*,
Pixmap *pixmap*, const int **attribList*);
void **glXDestroyPixmap** (Display **dpy*, GLXPixmap *pixmap*);
GLXPbuffer **glXCreatePbuffer** (Display **dpy*, GLXFBConfig *config*,
const int **attribList*);
void **glXDestroyPbuffer** (Display **dpy*, GLXPbuffer *pbuffer*);
void **glXQueryDrawable** (Display **dpy*, GLXDrawable *drawable*,
int *attribute*, unsigned int **value*);

管理OpenGL绘图环境：

GLXContext **glXCreateNewContext** (Display **dpy*, GLXFBConfig *config*,
int *renderType*, GLXContext *shareList*, Bool *direct*);
Bool **glXMakeContextCurrent** (Display **dpy*, GLXDrawable *drawable*,
GLXDrawable *read*, GLXContext *ctx*);
GLXDrawable **glXGetCurrentReadDrawable** (void);
Display* **glXGetCurrentDisplay** (void);
int **glXQueryContext** (Display **dpy*, GLXContext *ctx*, int *attribute*, int **value*);

选择和返回GLX**事件：**

void **glXSelectEvent** (Display **dpy*, GLXDrawable *drawable*,
unsigned long *eventMask*);
void **glXGetSelectedEvent** (Display **dpy*, GLXDrawable *drawable*,
unsigned long **eventMask*);