OpenGL 2

Hendri Karisma, S.Kom
Universitas Komputer Indonesia
Informatika
2013

Data Types

Suffix	Data Type	Typical Corresponding C-Language Type	OpenGL Type Definition
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating-point	float	GLfloat, GLclampf
d	64-bit floating-point	double	GLdouble, GLclampd
ub	8-bit unsigned integer	unsigned char	GLubyte, GLboolean
us	16-bit unsigned integer	unsigned short	GLushort
ui	32-bit unsigned integer	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

Display Main

- glutInit(int *argc, char **argv) initializes GLUT and processes any command line arguments (for X, this would be options such as display and-geometry). glutInit() should be called before any other GLUT routine.
- glutInitDisplayMode(unsigned int mode) specifies whether to use an RGBA or color-index color model. You can also specify whether you want a single- or double-buffered window. (If you’re working in color index mode, you’ll want to load certain colors into the color map; use glutSetColor() to do this.)
- glutInitWindowPosition(int x, int y) specifies the screen location for the upper-left corner of your window.
- glutInitWindowSize(int width, int height) specifies the size, in pixels, of your window.

Display #2

- **glutDisplayFunc**(void (*func)(void)) is the first and most important event callback function you will see. Whenever GLUT determines that the contents of the window need to be redisplayed, the callback function registered by glutDisplayFunc() is executed. Therefore, you should put all the routines you need to redraw the scene in the display callback function.

Display #3

- The very last thing you must do is call **glutMainLoop**(). All windows that have been created are now shown, and rendering to those windows is now effective. Event processing begins, and the registered display

glClearColor()

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

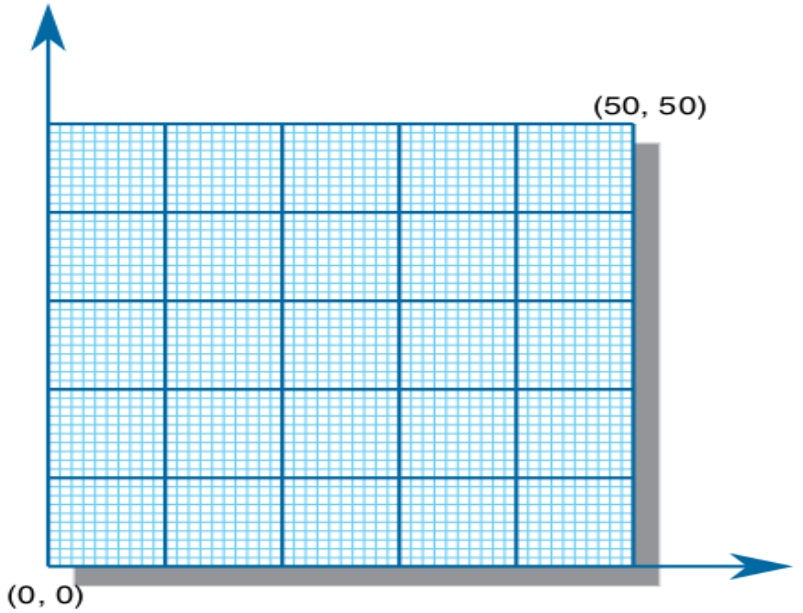
Sets the current clearing color for use in clearing color buffers in RGBA mode. (See Chapter 4 for more information on RGBA mode.) The *red*, *green*, *blue*, and *alpha* values are clamped if necessary to the range [0, 1]. The default clearing color is (0, 0, 0, 0), which is black.

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

Clears the specified buffers to their current clearing values. The *mask* argument is a bitwise logical OR combination of the values listed in Table 2-1.

gluOrtho2D()

- Routine puts the origin, (0, 0), in the lowest, leftmost square, and makes each square represent one unit. Now, when you render the points, lines, and polygons in the rest of this chapter, they will appear on this paper in easily predictable squares.



Specifying Vertices

void **glVertex**[234]{sifd}(*TYPE coords*);
void **glVertex**[234]{sifd}v(const *TYPE* coords*);

Specifies a vertex for use in describing a geometric object. You can supply up to four coordinates (*x*, *y*, *z*, *w*) for a particular vertex or as few as two (*x*, *y*) by selecting the appropriate version of the command. If you use a version that doesn't explicitly specify *z* or *w*, *z* is understood to be 0, and *w* is understood to be 1. Calls to **glVertex*()** are effective only between a **glBegin()** and **glEnd()** pair.

```
glVertex2s(2, 3);
glVertex3d(0.0, 0.0, 3.1415926535898);
glVertex4f(2.3, 1.0, -2.2, 2.0);

GLdouble dvect[3] = {5.0, 9.0, 1992.0};
glVertex3dv(dvect);
```

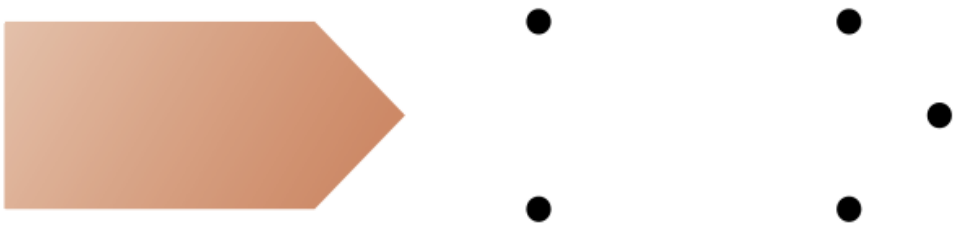
void **glBegin**(GLenum *mode*);

Marks the beginning of a vertex-data list that describes a geometric primitive. The type of primitive is indicated by *mode*, which can be any of the values shown in Table 2-2.

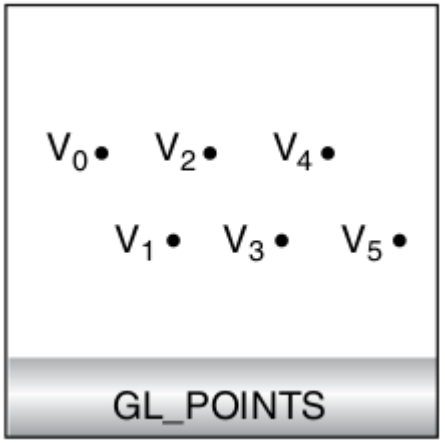
void **glEnd**(void);

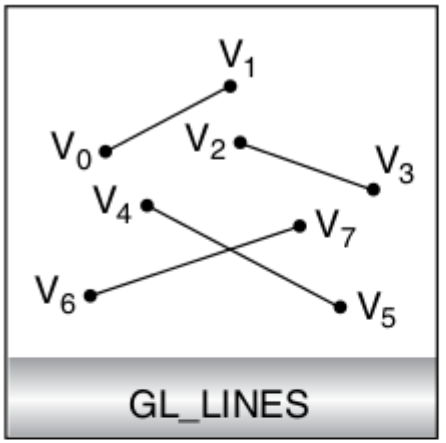
Marks the end of a vertex-data list.

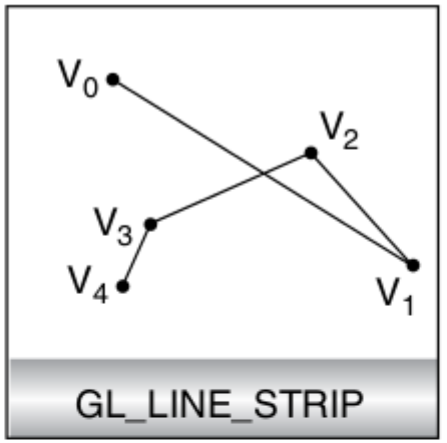
```
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
    glVertex2f(0.0, 3.0);
    glVertex2f(4.0, 3.0);
    glVertex2f(6.0, 1.5);
    glVertex2f(4.0, 0.0);
glEnd();
```

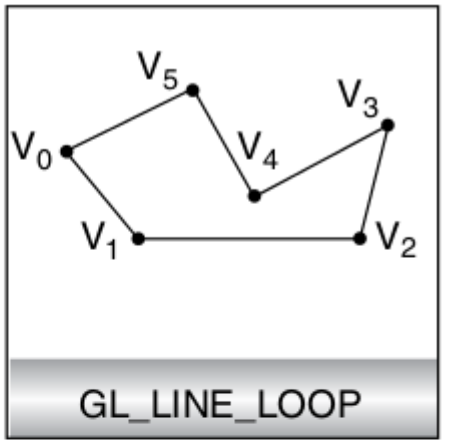


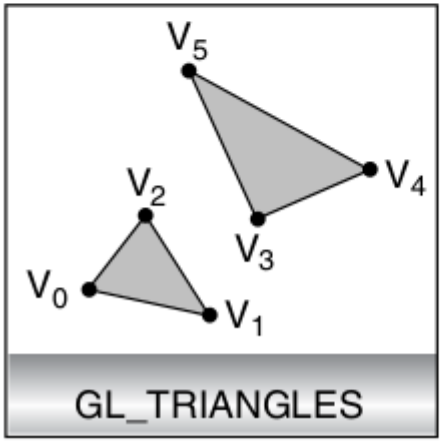
Value	Meaning
GL_POINTS	Individual points
GL_LINES	Pairs of vertices interpreted as individual line segments
GL_LINE_STRIP	Series of connected line segments
GL_LINE_LOOP	Same as above, with a segment added between last and first vertices
GL_TRIANGLES	Triples of vertices interpreted as triangles
GL_TRIANGLE_STRIP	Linked strip of triangles
GL_TRIANGLE_FAN	Linked fan of triangles
GL_QUADS	Quadruples of vertices interpreted as four-sided polygons
GL_QUAD_STRIP	Linked strip of quadrilaterals
GL_POLYGON	Boundary of a simple, convex polygon

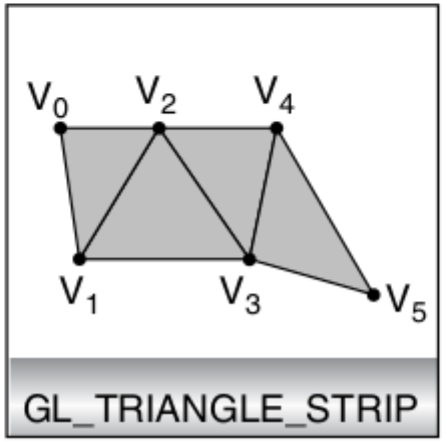

V0
V2
V4
V1
V3
V5
GL_POINTS

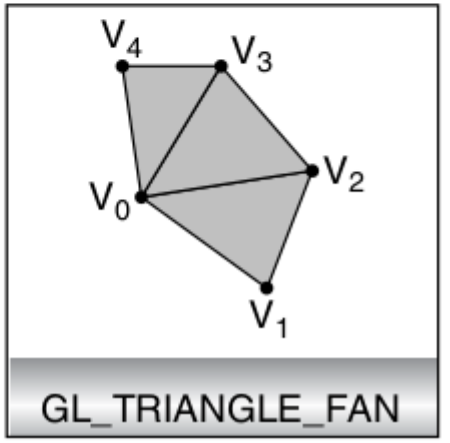

V1
V0
V2
V3
V4
V7
V6
V5
GL_LINES

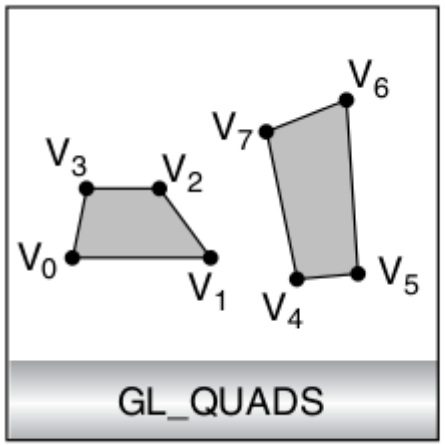

V0
V2
V3
V4
V1
GL_LINE_STRIP

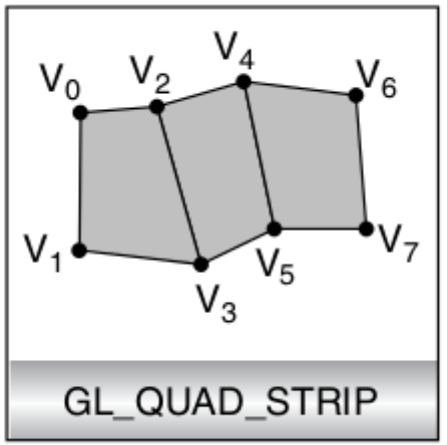

V5
V3
V0
V4
V1
V2
GL_LINE_LOOP

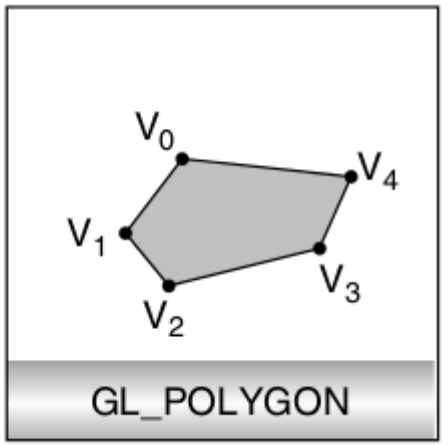

V5
V4
V2
V3
V0
V1
GL_TRIANGLES


V0
V2
V4
V1
V3
V5
GL_TRIANGLE_STRIP


V4
V3
V2
V0
V1
GL_TRIANGLE_FAN


V6
V7
V3
V2
V0
V1
V4
V5
GL_QUADS


V0
V2
V4
V6
V1
V5
V7
V3
GL_QUAD_STRIP


V0
V4
V1
V3
V2
GL_POLYGON

glEnable

void **glEnable**(GLenum *capability*);

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

lines), and GL_LIGHTING (you get the idea).