OpenGL – Open Graphics Library

Datentypen:

8 Bit Ganzzahl:	`signed char`	`GLbyte`
16 Bit Ganzzahl:	`short`	`GLshort`
32 Bit Ganzzahl:	`int oder long`	`GLint, GLsizei`
32 Bit Fließkomma:	`float`	`GLfloat, GLclampf`
64 Bit Fließkomma:	`double`	`GLdouble, GLclampd`
8 Bit Ganzzahl vorzeichenlos:	`unsigned char`	`GLubyte, GLboolean`
16 Bit Ganzzahl vorzeichenlos:	`unsigned short`	`GLushort`
32 Bit Ganzzahl vorzeichenlos:	`unsigned int oder long`	`GLuint, GLenum, GLbitfield`

Programmrumpf (für einfache Anwendung):

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <GL/glut.h>

void Display(void) {
    glClear(GL_COLOR_BUFFER_BIT);                   // löschen des Fensters
    glColor3f(1.0,0.0,0.0);                         // Farbe setzen

    // ab hier kommen die Einträge

    glFlush();                                      // Leeren der Ausgabepipeline
}

int main(void) {
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);      // Display-Modus
    glutCreateWindow("First");                      // Fenster mit Name "First" öffnen
    glutDisplayFunc(Display);                       // Display-Funktion setzen
    glutMainLoop();                                 // ... und los geht's
}
```

Primitive:

```
glBegin(GL_POINTS);                         // Primitive-Typ vereinbaren, hier: Punkte
    glVertex3f(0.0,0.9,0.0);                // Punkt setzen
    glVertex2f(0.0,0.9);                    // Punkt an die gleiche Stelle, nur ohne z-Komponente
glEnd();
```

Auswahl an Primitiven:

```
GL_POINTS             // Punkt
GL_LINES              // Linie
GL_LINE_STRIP         // Linien aneinander hängend
GL_LINE_LOOP          // Linien in einer Schleife

GL_TRIANGLES          // Dreieck
GL_TRIANGLE_STRIP     // Dreiecke aneinander hängend
GL_TRIANGLE_FAN       // Dreiecke als Fächer nebeneinander
GL_QUADS              // Quadrate, Trapez, ...
GL_QUAD_STRIP         // Quadrate nebeneinander
GL_POLYGON            // aus beliebigen Punkten eine Fläche
```

Normalen für Primitive setzen:

```
glBegin(GL_QUAD);
    glNormal3f(0.0, 0.0, 1.0);       // Hier:
    glVertex2f(0.0, 0.0);            // Verläuft die Reihenfolge der Vertex gegen den Uhrzeigersinn,
    glVertex2f(1.0, 0.0);            // so steht die Normale senkrecht auf der Fläche.
    glVertex2f(1.0, 1.0);
    glVertex2f(0.0, 1.0);            // Merke: Rechter-Daumen-Regel
glEnd();
```

komplexere Primitive mit GLUT:

```
Wire = Drahtgeflecht          Solid = fest

Kugel:
    glutWireSphere(radius, slices, stacks);         // slices = Anz. der Quersch. in Richt. der z-Achse
    glutSolidSphere(radius, slices, stacks);        // stacks = Anz. der Quersch. senkrecht zur z-Achse
Würfel:
    glutWireCube(size);
    glutSolidCube(size);
Konus, Kegel (in z-Richtung zeigend):
    glutWireCone(base, height, slices, stacks);     // base = Basisdurchmesser
    glutSolidCone(base, height, slices, stacks);    // height = Höhe
Ring (Durchblick in z-Richtung):
    glutWireTorus(innerRadius, outerRadius, slices, stacks);
    glutSolidTorus(innerRadius, outerRadius, slices, stacks);
14-Eck:
    glutWireDodecahedron(void);
    glutSolidDodecahedron(void);
8-Eck:
    glutWireOctahedron(void);
    glutSolidOctahedron(void);
4-Eck:
    glutWireTetrahedron(void);
    glutSolidTetrahedron(void);
25-Eck:
    glutWireIcosahedron(void);
    glutSolidIcosahedron(void);
Teekanne:
    glutWireTeapot(size);                   // size = Größe, alles ändert sich proportional
    glutSolidTeapot(size);
```

Translation, Rotation, Skalierung:

```
glScalef(dx, dy, dz);               // skalieren
glTranslatef(dx, dy, dz);           // translieren
glRotatef(alpha, dx, dy, dz);       // drehen (vom Koordinatensystem ausgehend im Uhrzeigersinn)
```

Matrix-Stack (Umsetzung des Szenengraphs auf Low-Level):

```
glPushMatrix();       // dupliziert oberste Matrix
    gl...(...);       // Translation, Skalierung ... immer auf oberste Matrix
glPopMatrix();        // löscht oberste Matrix
```

Modelview- und Projektionsmatrix (immer zusammen):

```
glMatrixMode(GL_PROJECTION);        // umschalten auf Projetionsmatrix
glLoadIdentity();                   // Nullpunkt
gl...(...);                         // Parallelprojektion oder Zentralprojektion

glMatrixMode(GL_MODELVIEW);         // umschalten auf Modelviewmatrix
glLoadIdentity();                   // Nullpunkt
gl...(...);                         // Viewing und Modelling
```

Parallelprojektion:

```
glOrtho(left, right, bottom, top, near, far);
```

Zentralprojektion:

```
glFrustum(left, right, bottom, top, near, far);
```

Viewport:

```
glMatrixMode(GL_VIEWPORT);                          // umschalten auf Viewportmatrix
glViewport(begin_x, begin_y, width, height);
glutPostRedisplay();                                // neuzeichnen
```

Alternative zu glFrustum() oder glOrtho() - gluPerspective():

```
glMatrixMode(GL_VIEWPORT);
glViewport(0, 0, width, height);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(fovy, aspect, zNear, zFar);          // fovy = Ausblickwinkel
                                                    // aspect = Verhältnis von width zu height
```

Lighting:

```
GLfloat light_pos[]      = {1.0, 1.0, 1.0, 0.0};
GLfloat light_ambient[]  = {0.5, 0.5, 0.5, 1.0};
GLfloat light_diffuse[]  = {1.0, 1.0, 1.0, 1.0};
GLfloat light_specular[] = {1.0, 1.0, 1.0, 1.0};
GLfloat mat_ambient []   = {0.3, 0.3, 0.3, 1.0};
GLfloat mat_diffuse []   = {1.0, 0.0, 0.0, 1.0};
GLfloat mat_specular [] = {0.0, 0.0, 0.0, 1.0};
GLfloat mat_emission [] = {0.0, 0.0, 0.0, 1.0};
Glfloat mat-shininess   = {10.0};                   // 0.0 ... 128.0

glMatrixMode(GL_MODELVIEW);                         // Lichtquellen im Modelview-Modus setzen
glLoadIdentity();

glLightfv(GL_LIGHT0, GL_POSITION, light_pos);       // Position immer vor erste Trans. / Rot. setzen,
                                                    // sonst dreht sich die Lichtquelle mit
glLightfv(GL_LIGHT0, GL_AMBIENT,  light_ambient);   // Lichtquelle gibt ambientes Licht ab
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);  // Lichtquelle gibt Glanzlicht ab
glLightfv(GL_LIGHT0, GL_DIFFUSE,  light_diffuse);   // Lichtquelle gibt diffuses Licht ab
glEnable(GL_LIGHT0);                                // Lichtquelle Nr 0 einschalten

glMaterialfv(GL_FRONT, GL_AMBIENT,  mat_ambient);   // Material nimmt ambientes Licht auf
glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);  // Material nimmt Glanzlicht auf
glMaterialfv(GL_FRONT, GL_DIFFUSE,  mat_diffuse);   // Material nimmt diffuses Licht auf
glMaterialfv(GL_FRONT, GL_EMISSION, mat_emission);  // Material gibt Licht ab
glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);// specular-light exponent
```

Z-Buffering mit Double-Buffering:

```
display(void){
    ...
    glClear_(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);  // COLOR_BUFFER nur, um Platz zu sparen
    ...
    glFlush();               // am Ende der Display-Funktion
    glutSwapBuffers();
}

main(void) {
    ...
    glEnable(GL_DEPTH_TEST);
    ...
    glutInitDisplayMode(GLUT_DOUBLE |...);
    ....
}
```