OpenGL

„Open Graphics Library“ (OpenGL: ® Silicon Graphics Inc.)
www.opengl.org – Aktuelle Version: 4.4 (2013)

1982-92: IRIS GL (für SGI-Rechner) Seit 1992 frei verfügbar

Definitionsgremium: OpenGL ARB (Architecture Review Board – seit 2006 als Teil der „Khronos Group“)

The Red Book – Version 1.1 (1997):
www.glprogramming.com/red (On-Line, HTML)

Release 1 (1994):
http://fly.cc.fer.hr/~unreal/theredbook/theredbook.zip (HTML)

Code-Beispiele aus dem „Red Book“: www.opengl-redbook.com

The OpenGL Wiki: www.opengl.org/wiki

Deutschsprachige „OpenGL-Community“: www.delphigl.com

„Öffentlich“ ist bei OpenGL nur die Spezifikation!

Open-Source Implementierung der OpenGL-Spezifikation:

The Mesa 3D Graphics Library
www.mesa3d.org – Aktuelle Version: 9.2 (2013)

GLUT

OpenGL ist unabhängig von Fenstersystemen; dies erfordert für Grafik-Sw die Adoption

- entweder eines ‚nativen' Fenstersystems oder
- einer Programmierschnittstelle (Application Programming Interface, API) zur Anbindung an Fenstersysteme.

Eine solche Programmierschnittstelle ist GLUT (*OpenGL Utility Toolkit* - Aussprache „wie *glutttony*").

Aktuelle Version: 3.7 (3.7.6)

Literatur:

- M.J.Kilgard: „OpenGL Programming for the X Window System", Addison-Wesley 1996
- E.Angel: „Interactive Computer Graphics: A Top-Down Approach Using OpenGL", Addison-Wesley 2006

GLUT

HTML-Online-Dokumentation zu GLUT:

- Links zu Quellen und Dokumentation:
  www.opengl.org/resources/libraries/glut

- Benutzungshandbuch:
  www.opengl.org/documentation/specs/glut/spec3/spec3.html
  www.opengl.org/resources/libraries/glut/spec3/spec3.html
  www.opengl.org/resources/libraries/glut/glut-3.spec.pdf – bzw.:
  http://homepages.thm.de/christ/>Computergrafik>aktuell

- Quellen u. Dokumentation für MS-Windows zu GLUT:
  www.xmission.com/~nate/glut.html

GLUT

Funktionalität durch GLUT:

- Einrichtung und Handhabung mehrerer Grafik-Fenster
- Ereignisverarbeitung durch Callbacks *(event processing)*
- Unterstützung vielfältiger Eingabegeräte
- Routinen zur Einhaltung von Zeitvorgaben *(timer)*
- Nutzung “ereignisloser” *("idle”)* Zeitintervalle
- Erzeugung von Pop-Up-Menü-Kaskaden
- Unterprogramme zur Generierung geometrischer Körper
- Mehrere Bitmap-/ Vektor-Schriftarten *(raster/ stroke fonts)*
- Diverse Funktionen zur Fenster- und Overlayverwaltung

GLUT - Unterschnittstellen

1. Initialisierung:

- Initialisierung des Fenstersystems,
- Überprüfug der **`main()`**-Parameter,
- Setzen der Voreinstellungen für Fenster-Position, Größe, Darstellungsmodus (Single/ Double Buffer)
- 4 Routinen:

notwendig

```
void glutInit(int *argc, char **argv);
                                    /*Parameter aus main()*/
void glutInitWindowSize(int width, int height);
                                        /*def.: (300,300)*/
void glutInitWindowPosition(int x, int y);
  /*def.: (-1,-1): dem nativen Fenstersystem ueberlassen*/
void glutInitDisplayMode(unsigned int mode);
                              /*def.: (GLUT_RGB | GLUT_SINGLE)*/
```

GLUT - Unterschnittstellen

2. Start der Ereignisverarbeitung:

- Letzte Anweisung (meist in **`main()`**) vor Überlassung der Programmsteuerung an GLUT und den dort registrierten Callbacks der Anwendung – keine Rückkehr!
- 1 Routine:

```
void glutMainLoop(void);
```

3. Fenster-Verwaltung:

- Fenster-Generierung und -Steuerung
- 18 Routinen:

```
int glutCreateWindow(char *name);/*intVar<=Fenst.einrichten*/
int glutCreateSubWindow(int win,int x,int y,int wid,int hig);
void glutSetWindow(int win);/*Setzen:win=>aktuelles Fenster*/
int glutGetWindow(void);  /*Abfragen:int<=aktuelles Fenster*/
```

GLUT - Unterschnittstellen

3. Fenster-Verwaltung (Forts.):

```
                          /*meist: "execution upon return to GLUT"*/
void glutDestroyWindow(int win);
void glutPostRedisplay(void);   /*Kennzeichnung: aktualisieren!*/
void glutSwapBuffers(void);
void glutPositionWindow(int x, int y);
void glutReshapeWindow(int width, int height);
void glutFullScreen(void);          /*may vary by window system*/
void glutPopWindow(void);         /*Aendert Fenster-Reihenfolge*/
void glutPushWindow(void);        /*     "     "          "     */
void glutShowWindow(void);             /*macht Fenster sichtbar*/
void glutHideWindow(void);           /*macht Fenster unsichtbar*/
void glutIconifyWindow(void);
void glutSetWindowTitle(char *name);
void glutSetIconTitle(char *name);
void glutSetCursor(int cursor);/*23 cursor images GLUT_CURSOR..*/
```

(Test7.exe)

GLUT - Unterschnittstellen

4. Overlay-Verwaltung:

- Einrichtung u. Nutzung von Overlay-Hw (falls vorhanden)
- 6 Routinen:

```
void glutEstablishOverlay(void);
void glutUseLayer(GLenum layer);
void glutRemoveOverlay(void);
void glutPostOverlayRedisplay(void);
void glutShowOverlay(void);
void glutHideOverlay(void);
```

GLUT - Unterschnittstellen

5. Menü-Verwaltung:

- Menü-Generierung und -Steuerung
- 11 Routinen:

```
int  glutCreateMenu(void (*func)(int value));/*Callback-Reg.*/
void glutSetMenu(int menu);
int  glutGetMenu(void);
void glutDestroyMenu(int menu);
void glutAddMenuEntry(char *name, int value);
   /*Eintrag und an glutCreateMenu-Callback uebergebener Wert*/
void glutAddSubMenu(char *name, int menu);
void glutChangeToMenuEntry(int entry, char *name, int value);
void glutChangeToSubMenu(int entry, char *name, int menu);
void glutRemoveMenuItem(int entry);
void glutAttachMenu(int button);          /*Maustaste~Menue*/
void glutDetachMenu(int button);
```

(Glutmech.exe)

6. Callback-Registrierung:

- Anmeldung von Applikationsfunktionen (Callbacks).
- Callback-Aufruf durch die Verarbeitungsschleife der GLUT- Ereignisse *(GLUT event processing loop)*.
- Drei Callback-Typen:
  - Fenster-Callbacks – zur Änderung von Größe, Form, Sichtbarkeit von Fenstern bzw. zur Anzeige des fertiggestellten Fensterinhalts
  - Menü-Callbacks – zur Menü-Darstellung
  - Globale Callbacks – für die Einbeziehung von Zeitabläufen und Menü-Nutzung.
- insg. 20 Routinen:

GLUT - Unterschnittstellen

6. Callback-Registrierung (Forts.):

- **Fenster-Callbacks:**

```
void glutDisplayFunc(void(*func)(void)); /*CB f.aktuelles Fenster*/

void glutOverlayDisplayFunc(void (*func)(void));

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

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

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

void glutMotionFunc(void(*func)(int x,int y));   /*button pressed*/

void glutPassiveMotionFunc(void(*func)(int x,int y));  /*no press*/

void glutVisibilityFunc(void (*func)(int state));
                /* state == GLUT_NOT_VISIBLE oder GLUT_VISIBLE  */

void glutEntryFunc(void (*func)(int state));   /*Maus im Fenster?*/
                        /* state == GLUT_LEFT oder GLUT_ENTERED */
```

6. Callback-Registrierung (Forts.):

- Fenster-Callbacks (Forts.):

```
void glutSpecialFunc(void (*func)(int key, int x, int y));
     /* key == Funktions-, Cursor- (Pfeil-) und Spezialtasten */

void glutSpaceballMotionFunc(void (*func)(int x, int y, int z));

void glutSpaceballRotateFunc(void (*func)(int x, int y, int z));

void glutSpaceballButtonFunc(void (*func)(int button, int state));

void glutButtonBoxFunc(void (*func)(int button, int state));

void glutDialsFunc(void (*func)(int dial, int value));

void glutTabletMotionFunc(void (*func)(int x, int y));

void glutTabletButtonFunc(void (*func)(int button, int state,
   int x, int y));
```

6. Callback-Registrierung (Forts.):

- **Menü-Callbacks:**

```
void glutMenuStatusFunc(void(*func)(int status,int x,int y));
   [auch: void glutMenuStateFunc(void (*func)(int status));]
   /* status == GLUT_MENU_IN_USE oder GLUT_MENU_NOT_IN_USE */
```

- **Globale Callbacks:**

```
void glutIdleFunc(void (*func)(void));  /*wenn kein Ereignis*/

void glutTimerFunc(unsigned int msecs,(*func)(int val),val);
            /*msecs Millisekunden spaeter: Aufruf func(val)*/
```

GLUT - Unterschnittstellen

7. Verwaltung von Farbtabellen mit Farbindex

(*Color Index Colormap Management)* – 3 Routinen:

```
void glutSetColor(int cell, GLfloat red, GLfloat green,
                            GLfloat blue);      /*LUT-Eintrag*/
GLfloat glutGetColor(int cell, int component);
void glutCopyColormap(int win);
```

8. Zustands-Abfrage *(State Retrieval)* – 5 Routinen:

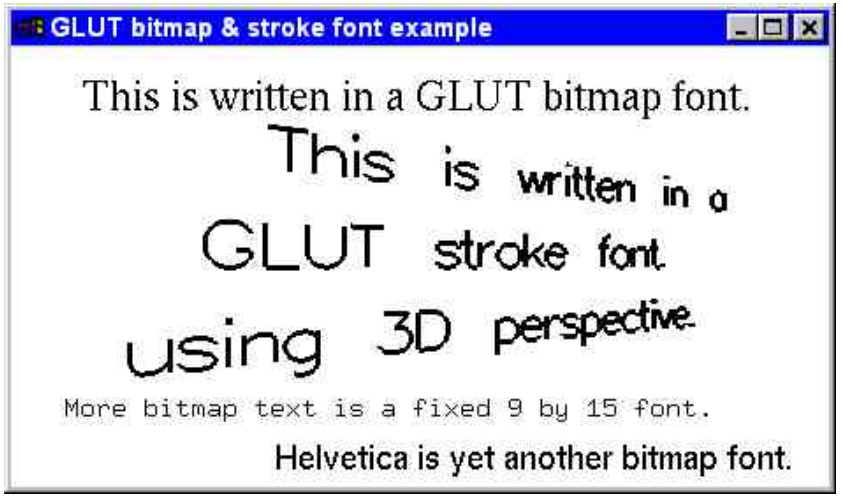
```
int glutGet(GLenum state);
    /* state: 1 v.35 Zustandskennungen, z.B.GLUT_WINDOW_X */
int glutLayerGet(GLenum info);          /* Overlay-Nutzung */
int glutDeviceGet(GLenum info);
                /* info: 1 von 10 GLUT_HAS_... (Maus etc.) */
int glutGetModifiers(void);
                  /* GLUT_ACTIVE_SHIFT, ..._CTRL, ..._ALT */
int glutExtensionSupported(char *extension);/*OpenGL-Extens.*/
```

9. Wiedergabe von Bitmap- und Vektor-Schriftarten

(Font Rendering) – 9 Routinen:

```
void glutBitmapCharacter(void *font, int character);
```

font aus: **GLUT_BITMAP_8_BY_13**
GLUT_BITMAP_9_BY_15
GLUT_BITMAP_TIMES_ROMAN_10
GLUT_BITMAP_TIMES_ROMAN_24
GLUT_BITMAP_HELVETICA_10
GLUT_BITMAP_HELVETICA_12
GLUT_BITMAP_HELVETICA_18




```
int glutBitmapWidth(GLUTbitmapFont font, int character)
                    /*gibt Breite der Bitmap-Schrift in Pixel*/
void glutStrokeCharacter(void *font, int character);
```

font aus: **GLUT_STROKE_ROMAN**
GLUT_STROKE_MONO_ROMAN

```
int glutStrokeWidth(GLUTstrokeFont font, int character);
```

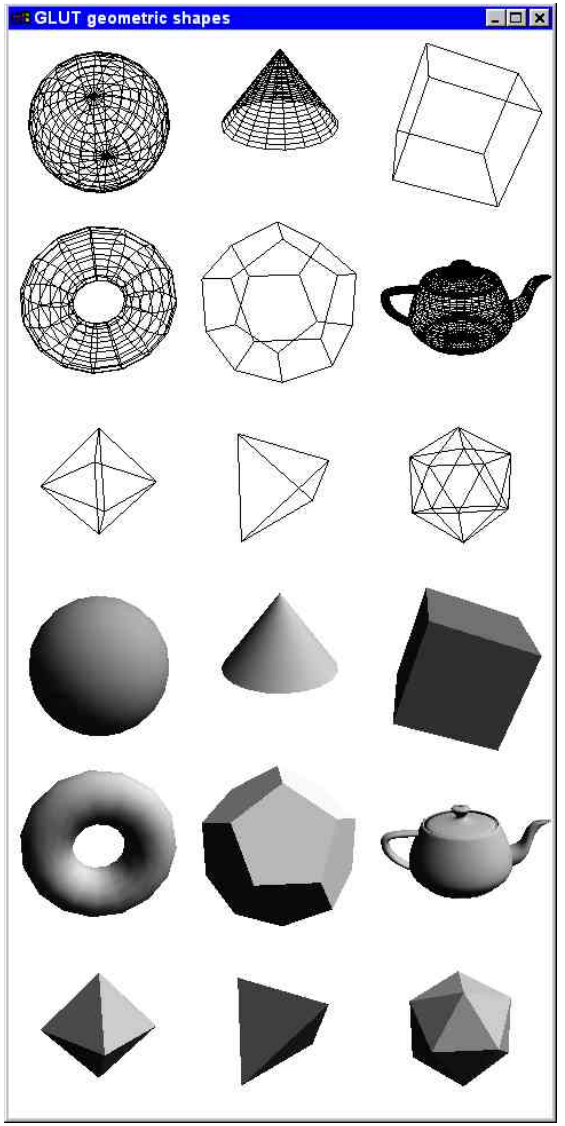
(s.u.)

GLUT - Unterschnittstellen

10. Wiedergabe Geometrischer Figuren

(Geometric Shape Rendering)–18 Routinen:

`glutSolidSphere,`	`glutWireSphere`
`glutSolidCube,`	`glutWireCube`
`glutSolidCone,`	`glutWireCone`
`glutSolidTorus,`	`glutWireTorus`
`glutSolidDodecahedron,`	`glutWireDodecahedron`
`glutSolidOctahedron,`	`glutWireOctahedron`
`glutSolidTetrahedron,`	`glutWireTetrahedron`
`glutSolidIcosahedron,`	`glutWireIcosahedron`
`glutSolidTeapot,`	`glutWireTeapot`



GLUT

Anmerkungen zu GLUT:

- GLUT-Koordinaten (Bildschirm, Fenster) in Pixel; Ursprung oben links (wie d. meisten Fenstersysteme ⇔ OpenGL: math. Koord.).
- Fenster-, Menü- u. Menüpunkt-Kennungen beginnen mit 1.
- GLUT-Header enthält OpenGL- u. GLU-Header:
  `#include <GL/glut.h>`
- **`glutInit`** soll zur Hauptinitialisierung genau einmal, möglichst am Programm-Anfang aufgerufen werden. Nur Aufrufe mit **`glutInit`**–Präfix dürfen davor stehen (Setzen v. Voreinstellgn)
- OpenGL-Einstellgn (**`gl*`**-Aufrufe) nach **`glutInit*()`**!
- GLUT übernimmt u.a.:
  - die Festlegung des Zeitpunktes für Fenster-Aktionen (Darstellung, Aktualisierung etc. immer erst nach Rückgabe der Kontrolle an die Ereignisverarbeitung von GLUT)
  - die Behandlung mehrerer verwandter Aufrufe (z.B. nach mehrmaligem **`glutPostRedisplay`** oder sich gegenseitig aufhebenden Fenster-Aktivierungen).

GLUT / OpenGL

Neben GLUT-Unterschnittstellen: OpenGL-Aufrufe, z.B.:

- Aktuelle Rasterposition an (x, y) setzen (def.: **-1.≤ x,y ≤ +1.**):
  `void glRasterPos2{sifd}(TYPE x, TYPE y);`
  Untere linke Fenster-Ecke bei (**-1.;-1.**)
- Erzwingung der Ausführung bisheriger Anweisungen
  („Weiter mit nächstem Bild!“ – vgl. `fflush( )`):
  `void glFlush(void);/*erzwingt AusfuehrgsStart*/`
  Erzwingung der Fertigstellung („Warte auf Pixelbild!“):
  `void glFinish(void);/*wartetAusfuehrgsEnde ab*/`
  (Hintergrund: Manche Spezial-Hw -z.B. Netzwerk- sammelt mehrere Anweisungen, bevor sie sie verarbeitet.)
- Fenster (Bildspeicher) löschen – eine teure (langsame) Operation:
  `void glClear(GL_COLOR_BUFFER_BIT);`
- Lösch-/Hintergrundfarbe wählen (Def.: **(0.,0.,0.,1.)**):

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

Farbwerte werden intern auf d. Bereich [0,1] begrenzt („clamped“).

- OpenGL in ISO C implementiert: unterschiedliche Namen je nach Parameterliste. Typische Schreibweise:

  `glFunktionsNameTyp (GL_KONST_NAME, param);`

  z.B. Setzen der Zeichenfarbe:

```
glColor3f(1.,1.,1.);          //R,G,B:weiss(def.)
glColor3i(255,255,255);
```

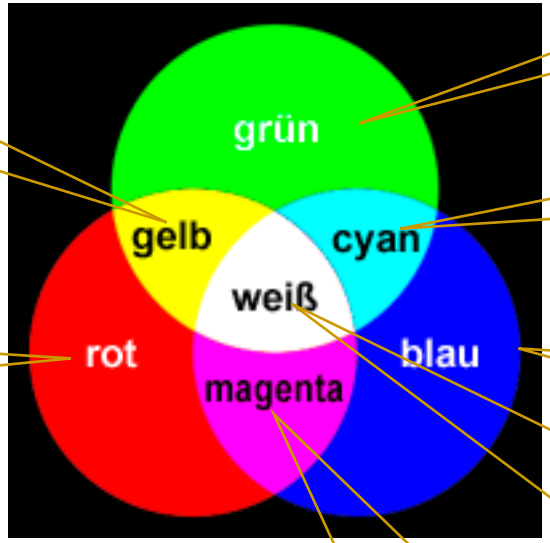
  Vektor-Version des Befehls (weniger Datentransfer):

```
GLfloat ffarbe[]={1.,0.,0.};          //RGB:rot
GLint   ifarbe[]={255,0,0};
glColor3fv(ffarbe);                   //Einstellg
glColor3iv(ifarbe);
glGetFloatv(GL_CURRENT_COLOR, ffarbe);//Abfrage
glGetIntegerv(GL_CURRENT_COLOR,ifarbe);
```

- ➡ OpenGL als Zustandsautomat (engl. *state machine*) implementiert: letzte (bzw.: Vor-) Einstellung (Default) gültig.
  ⇨ Weniger Datentransfer, günstig für Echtzeit- und C/S-Apps

OpenGL: A Drawing Survival Kit

Farben unter OpenGL (additive Farbmischung):




```
glColor3ub(0,255,0);
```

```
glColor3ub(255,255,0);
```

```
glColor3ub(0,255,255);
```

```
glColor3ub(255,0,0);
```

```
glColor3ub(0,0,255);
```

```
glColor3ub(255,255,255);
```

```
glColor3ub(255,0,255);
```

```
glColor3ub(255,0  ,0  );/*red    */
glColor3ub(0  ,255,0  );/*green  */
glColor3ub(255,255,0  );/*yellow */
glColor3ub(0  ,0  ,255);/*blue   */
glColor3ub(255,0  ,255);/*magenta*/
glColor3ub(0  ,255,255);/*cyan   */
glColor3ub(255,255,255);/*white  */
```

Bild:http://de.wikipedia.org/wiki/Bild:AdditiveFarbmischung.png

- Variable Größe in der Darstellung der Pixel mit:

```
void glPixelZoom(GLfloat zoomx, GLfloat zoomy);
```

- Einstellbare Punkt- und Strichstärken (in Pixel):

```
void glPointSize(GLfloat size); /*(def.: 1.0)*/
void glLineWidth(GLfloat width);/*(def.: 1.0)*/
```

  Nachkomma-Stellen wegen Anti-Aliasing („anti-éiliæsing“)

- OpenGL-intern schließlich alle Grafik-Objekte (Punkte, Linien u. Polygone) als geordnete Menge von Eckpunkten beschrieben;

  immer in (4D-)homogenen Koordinaten, ggf. mit z=0, w=1.

  Polygone müssen eben, geschlossen und konvex sein (sonst Ergebnis unbestimmt).

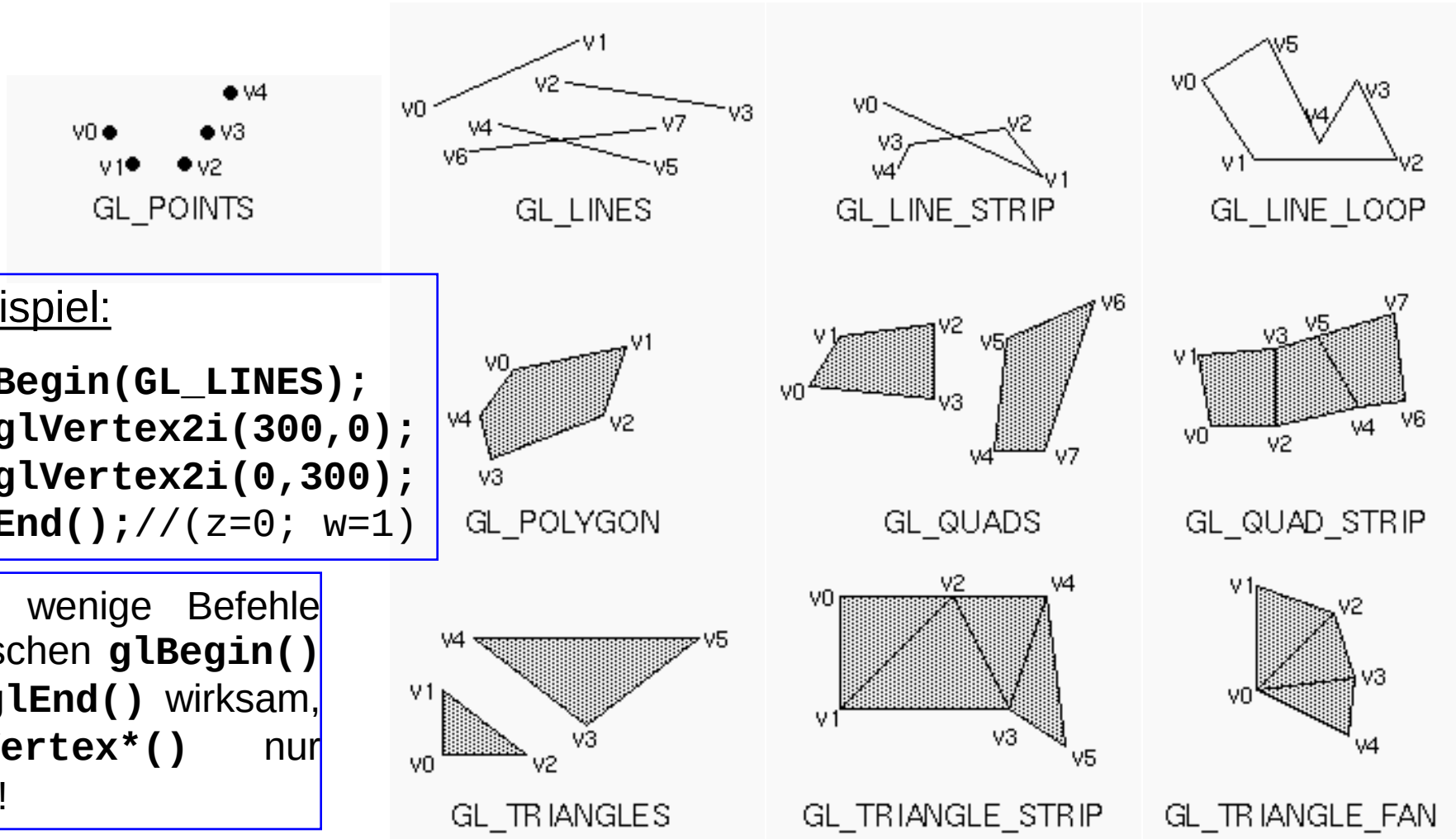
  Code-Struktur zur Konstruktion geometrischer Figuren:

```
void glBegin(GLenum mode);
  void glVertex{234}{sifd}[v](TYPE [*]coords);
void glEnd(void);
```

{...}: eins daraus
[...]:kann fehlen

OpenGL **TYPE** Definition (s.o.)

Werte für **mode** in **glBegin(GLenum mode);**



Beispiel:

```
glBegin(GL_LINES);
  glVertex2i(300,0);
  glVertex2i(0,300);
glEnd();//(z=0; w=1)
```

Nur wenige Befehle zwischen **glBegin()** u. **glEnd()** wirksam, **glVertex*()** nur dort!

Bild: OpenGL Programming Guide (Addison-Wesley Publishing Company) First Printing, 1997

Bei der Behandlung v. Bildern mit OpenGL Unterscheidung:

- **Format** : Bedeutung der gespeicherten Pixeldaten
  (S/W-Intensität, Look-Up-Table, RGB, RGBA etc.), z.B.:
  `GL_LUMINANCE`, **`GL_COLOR_INDEX`**, **`GL_RGB`**, **`GL_RGBA`**

- **Typ** : Datentyp der gespeicherten Pixel-Elemente
  (**`float`**, **`signed`** oder **`unsigned char`** oder **`int`** etc.), z.B.:
  `GL_FLOAT`, **`GL_BYTE`**, **`GL_UNSIGNED_BYTE`**, **`GL_UNSIGNED_INT`**
  Für den C-Datentyp (z.B. **`char`** / **`unsigned int`**) nicht bindend!

OpenGL verwendet oft Zeiger auf Variable ohne Wert (**`GLvoid *`**).

[**`void`** ist der C-Datentyp ohne Wert, von dem es keine Variablen geben darf. Zeiger auf ein Datenobjekt dieses Typs (**`void *`**) kann beliebig umgewandelt werden, ohne daß Information (Alignment o.ä.) verlorengeht.]

Zwei wichtige Aufrufe für die Arbeit mit Bildern in OpenGL:

vgl. Übung `colorPlay`

```
void glDrawPixels
(GLsizei width, GLsizei height, GLenum format,
 GLenum type, const GLvoid *pixels);
```

Zeichnet rechts oberhalb der aktuellen Rasterposition (vgl. **`glRasterPos2f()`**) ein Rechteck mit Kantenlängen **`width`** und **`height`** (in Pixel), mit Pixeldaten ab Adresse **`pixels`**.

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

Liest aus dem Bildspeicher (Framebuffer) ein Rechteck mit Pixeldaten, mit Kantenlängen **`width`** und **`height`**, dessen untere linke Ecke bei (**`x`**, **`y`**) liegt, und speichert sie ab Adresse **`pixels`**.

Weitere benötigte OpenGL-Anweisungen zur Bildausgabe:

- Festlegung des rechteckigen Ausschnitts („Viewport“) der Geräte-Ausgabefläche (Bildschirm, Fenster, Plotter) mit unterer linker Ecke bei (**x**, **y**) und Kantenlängen **width** und **height** (in Pixel) als Zeichenfläche:

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

- Einstellung, ob die nächsten geometr. Transformationen (Translationen, Rotationen etc.) auf die dargestellten Objekte oder auf den Augenpunkt (= Projektionszentrum) anzuwenden sind:

```
glMatrixMode (GL_MODELVIEW);  //Objekt-Trf.
glMatrixMode (GL_PROJECTION); //Augenpunkt-Trf.
```

- Parallelprojektion der bislang positionierten grafischen Objekte im quaderförmigen Raumabschnitt mit den (x-, y-) Eckpunkt-Koordinaten (**`left`**, **`bottom`**) und (**`right`**, **`top`**), jeweils bei z=**`-near`** und z=**`-far`**:

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

- Verschiebung (eines Objektes oder des Augenpunktes) parallel zu den Hauptachsen um **`x`**, **`y`**, **`z`**:

```
void glTranslate{fd}(TYPE x, TYPE y, TYPE z);
```

- Neutrales Element / Aufhebung aller bisherigen (Objekt- / Augenpunkt-) Transformationen:

```
glLoadIdentity();
```

Funktion zur Systemvorbereitung für OpenGL-Arbeit in 2D:

```
void DrawIn2D (int width, int height)
{ glViewport(0,0,width,height); //Zeichenflaeche
  glMatrixMode(GL_PROJECTION);  //Augenpkt-Trf.
  glLoadIdentity();             //Tabula rasa
  glOrtho(0,width,0,height,-1,1);//Parallel-Proj.
  glMatrixMode(GL_MODELVIEW);   //Grafiken-Trf.
  glLoadIdentity();             //Tabula rasa
  glTranslatef(0.375,0.375,.0); //optm.Rasterisrg
return;
}
```