Computergrafik

– für Bachelor-Studierende –

Prof. Dr. Aris Christidis

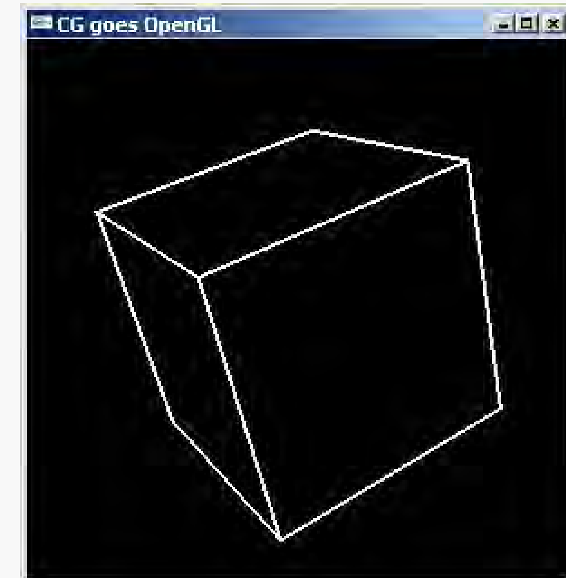
SS 2011

– Auszug –

OpenGL


```
/*CubeGL.c:*/

int main(int argc, char **argv)
{ glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE);
  glutCreateWindow("CG goes OpenGL");
  glutDisplayFunc(draw);
  glutKeyboardFunc(key);
  init();
  glutMainLoop();
  return 0;
}
```



OpenGL


```
/*CubeGL.c (Forts.):*/

void drawBox(void)
{ int jF;
  nah=eyez; fern=nah+2*diag;

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  if (persp) /*Perspektive:*/
   glFrustum (-diag, diag, -diag, diag, nah, fern);
  else       /*Parallel-Projektion:*/
   glOrtho   (-diag, diag, -diag, diag, nah, fern);

  if (backF) glEnable  (GL_CULL_FACE);
  else       glDisable (GL_CULL_FACE);
```

```
float eyez=10.,
      diag=3.464f;
int   backF=0, persp=1;
GLdouble nah=10.,
      fern=10.+2*3.464f;
```

```
/*CubeGL.c (Forts.):*/
glMatrixMode(GL_MODELVIEW); /*Animationshistorie:*/
glPushMatrix();/*Aktuelle Modell-Position merken*/
/*Ins Sichtvolumen verschieben u. positionieren:*/
glTranslatef(0.0, 0.0, -(nah+diag/2));
glRotatef(angle[X], 1.0, 0.0, 0.0);
glRotatef(angle[Y], 0.0, 1.0, 0.0);
glRotatef(angle[Z], 0.0, 0.0, 1.0);
/*Wuerfel zeichnen:*/
for (jF = 0; jF < 6; jF++)
{ glBegin(GL_POLYGON);
  glVertex3fv(Vrtx[faces[jF][0]]);//=(&Vrtx[faces[jF][0]][0]);
  glVertex3fv(Vrtx[faces[jF][1]]);//          ...
  glVertex3fv(Vrtx[faces[jF][2]]);//          ...
  glVertex3fv(Vrtx[faces[jF][3]]);//          ...
  glEnd();
} glPopMatrix(); /*Letzte Berechnungen vergessen*/
}
```

```
float      angle[3];
GLfloat Vrtx[8][3];
GLint   faces[6][4];
```

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

Design-Philosophie:

- Fenster-Einrichtung u. -Verwaltung mit wenigen Aufrufen (für Ein-Fenster-Applikationen genügt Callback +Start) :
  ➥ Schnelle Erlernbarkeit, Augenmerk auf Fenster-Inhalt
- Aufrufe mit möglichst knappen Parameterlisten, Zeiger nur für (Eingabe von) Zeichenketten, keine Zeiger-Rückgabe durch GLUT:
  ➥ Leichte Handhabung, Fehlerrobustheit
- Keine Verwendung v. Daten des nativen Fenstersystems (Fenster-Handler, Schriftarten):
  ➥ Unabhängigkeit vom nativen Fenstersystem

"Handler": Behandlungsroutine

Design-Philosophie: (Forts.)

- Übernahme der Ereignisverarbeitung, Überlassung von OpenGL-Displaylisten der Applikation:

  ➥ Einschränkung gleichzeitiger Verwendung weiterer Ereignis-verarbeitung, keine Einschränkung des OpenGL-Einsatzes

- Zustandshaftes *(stateful)* API:

  Zustand: Datensatz mit d. Beschreibung des Systems für die Bedürfnisse der Anwendung

  ➥ Einfache Applikationen durch Voreinstellungen *(default/ current window/ menu);* wiederholter Datentransfer (interessant z.B. bei unsicheren Verbindungen) wird vermieden

- Mehrere, differenziert aufgerufene Callbacks

  ➥ Vermeidung unnötigen Codes je nach Art der Applikation

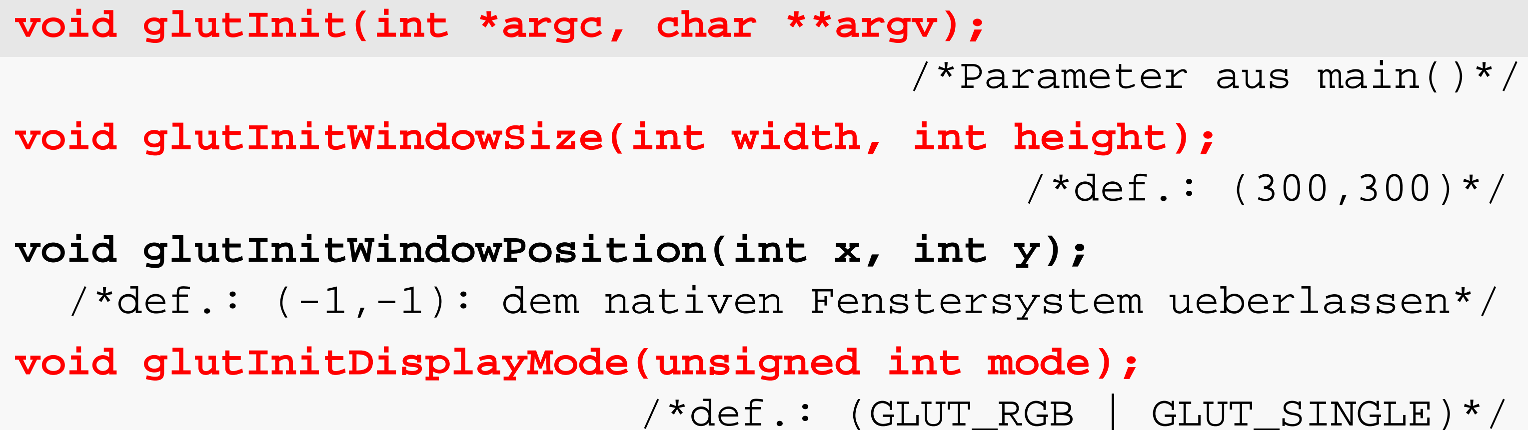
Echtzeit ! *(real time)*

- Logische Organisation in 10 ‚Sub-APIs':

GLUT - Unterschnittstellen

1. Initialisierung:

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




```
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

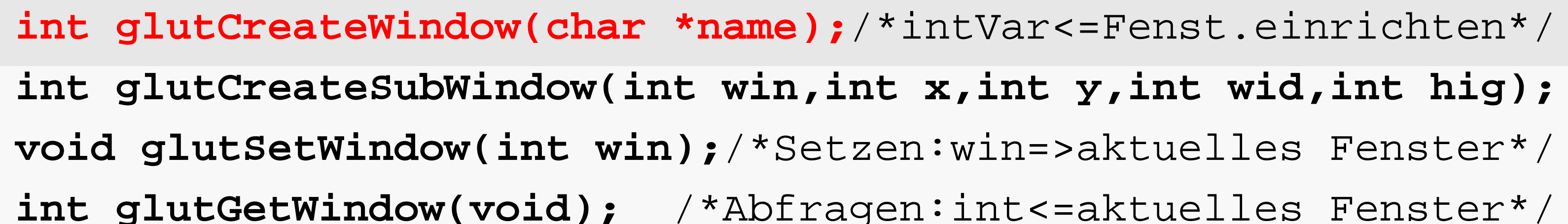
2. Start der Ereignisverarbeitung:

- 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*/
```

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)

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);
```

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)
```

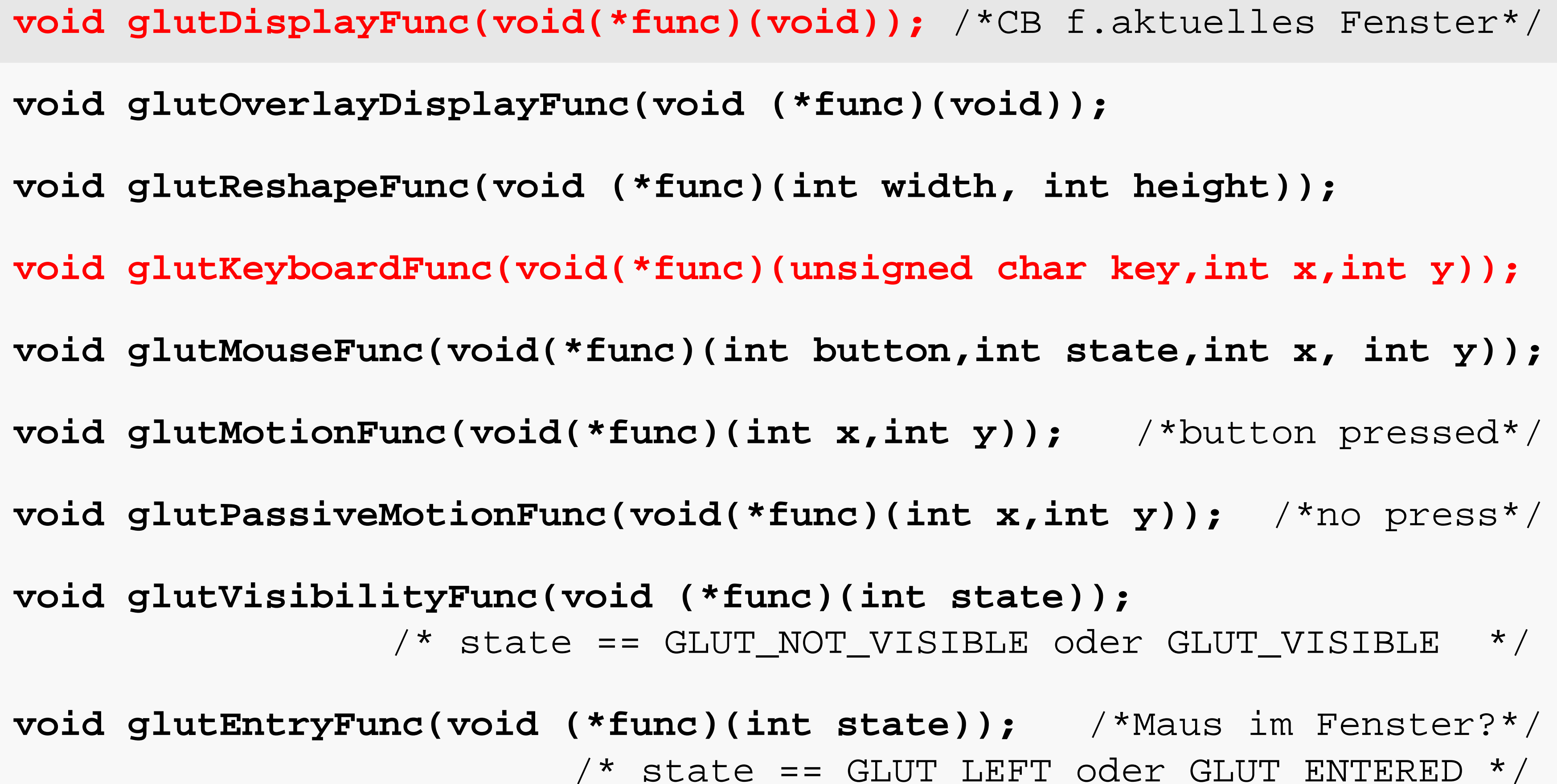
6. Callback-Registrierung:

- 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));
```

GLUT - Unterschnittstellen

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:

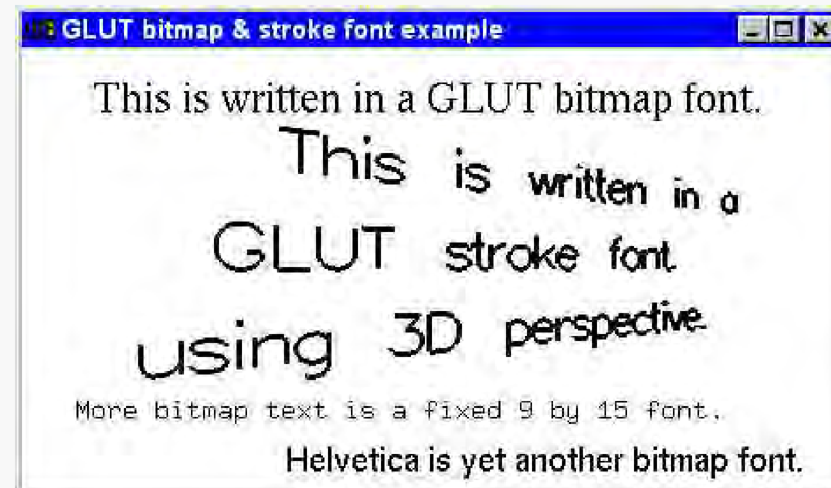
```
int glutGet(GLenum state);
    /* state: 1 v.35 Zustandskennungen, z.B.GLUT_WINDOW_X */
int glutLayerGet(GLenum info);           /* Overlay-Nutzung */
int glutDeviceGet(GLenum info);
                /* info: 1 von 10 GLUT_HAS_... (Maus etc.) */
int glutGetModifiers(void);
                   /* GLUT_ACTIVE_SHIFT,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

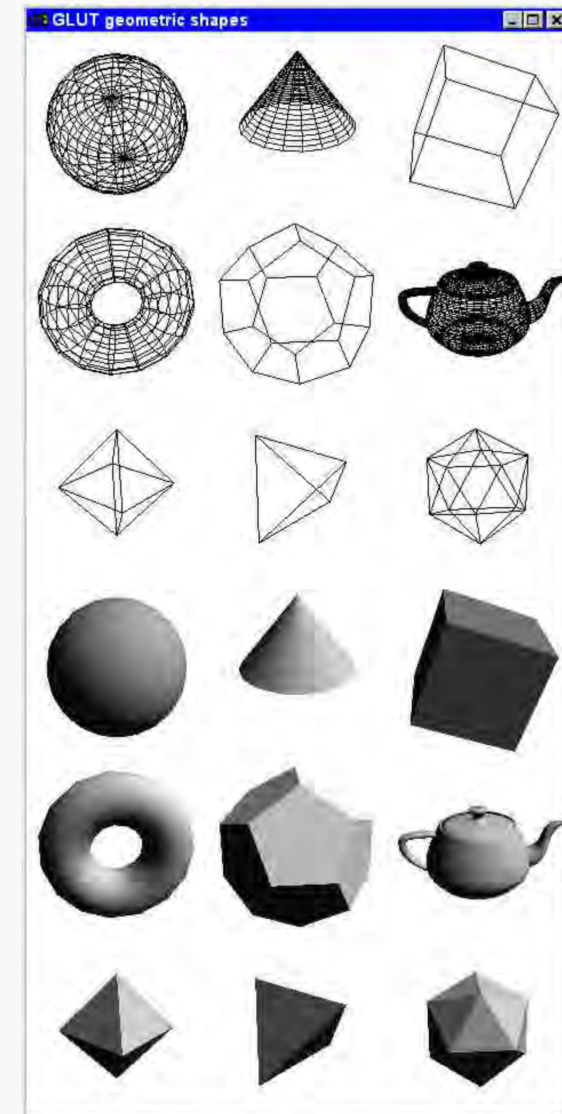
```
int glutStrokeWidth(GLUTstrokeFont font, int character);
```

(s.u.)

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 (z.B. zum Setzen von Voreinstellungen).
- 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).

OpenGL

- Grafik-Bibliothek mit einigen hundert Befehlen `gl...()` (div. Ausführungen) – angeschlossener „Aufsatz“: OpenGL Utility Library (GLU), ca. 50 Befehle `glu...()`
- Plattform-unabhängig (Hardware, Fenster-/ Betriebssyst.), netzwerkfähig (Client: Programm / Server: Darstellung)
  - Konstruktion bel. **Modelle** aus Grafik-Primitiven: Punkten, Linien, Polygonen, Bildern, Bitmaps (=Bitmustern)
  - Zusammenfassung mehrerer Modelle (Objekte) zu **Szenen**; Sicht-Berechnung bei gegebenem Augenpunkt
  - Farbgebung durch Berechnung von **Licht**intensitäten bei gegebener **Farbe** oder **Textur** (=aufgesetztem Bild)
  - Erzeugung eines **Pixel-Bild**es aus der geometrisch-farblichen Beschreibung („Rasterisierung“)
- **Rendering**: (Wiedergabe, Darstellung): Erzeugg. digitaler (Pixel-)Bilder aus log.-mathem. Modell-Beschreibungen.

OpenGL

- 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.)
```

  Vektor-Version des Befehls (weniger Datentransfer):

```
GLfloat farbe[]={1.,0.,0.};                  //RGB:rot
glColor3fv(farbe);                          //Einstellg
glGetFloatv(GL_CURRENT_COLOR, farbe); //Abfrage
```

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

```
void glEnable (GLenum cap); //z.B. GL_CULL_FACE
void glDisable(GLenum cap);
```

```
GLboolean glIsEnabled(GLenum capability);
```

  gibt zurück `GL_TRUE` oder `GL_FALSE` .

OpenGL

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

OpenGL: Rendering-Pipeline

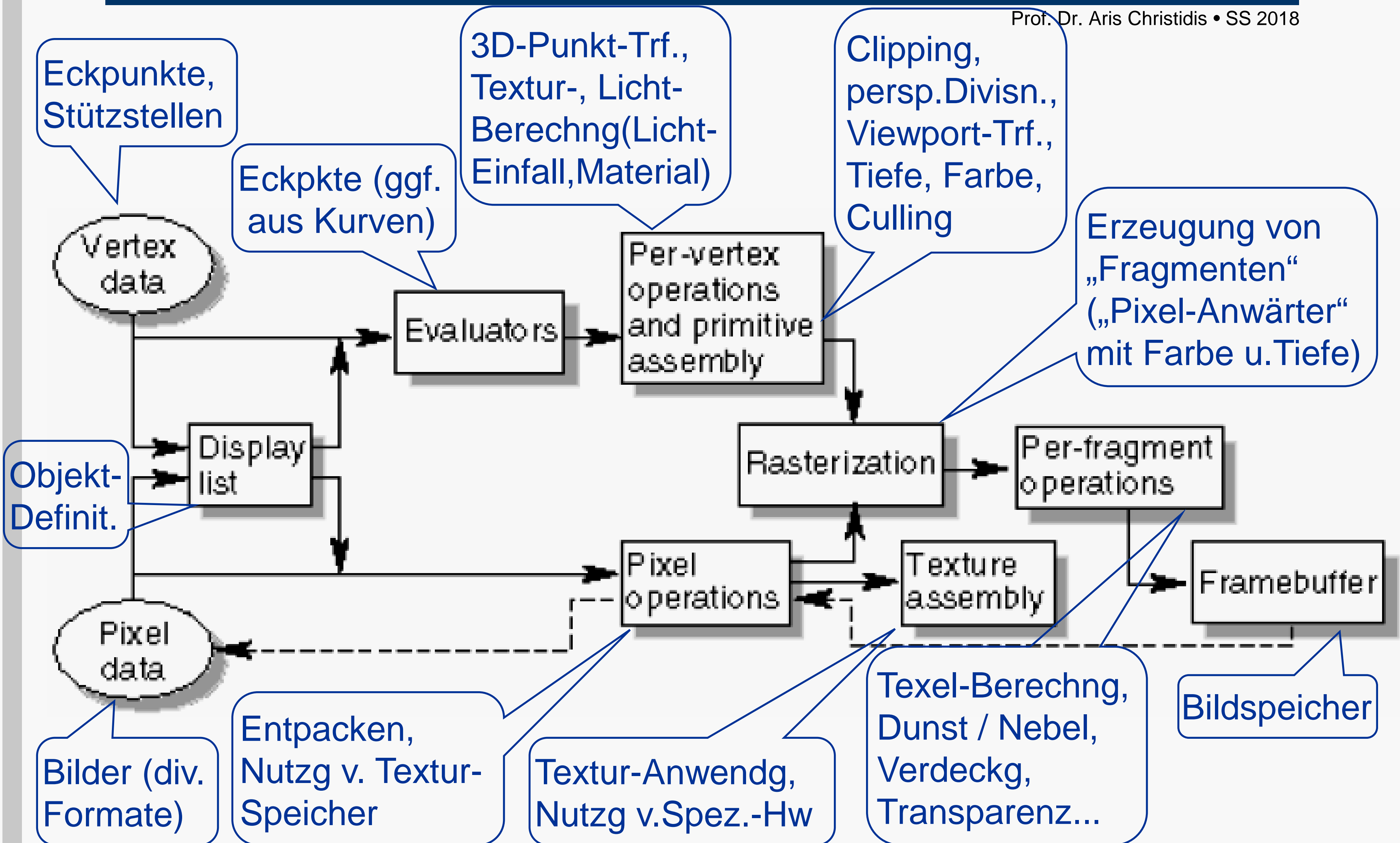


Bild: OpenGL Programming Guide (Addison-Wesley Publishing Company) Second Edition, 1997

Puffer löschen – hier: Bild- und Tiefenspeicher

- Löschfarben-Festlegung aus Rot, Grün, Blau,Transparenz („Alpha“); Werte-Intervall: [0,1]; Voreinst.: schwarz, opak

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

- Festlegung eines Löschwertes für Tiefe (Berechnung der Verdeckung); Werte-Intervall: [0,1]; Voreinst.: 1.0 (fern)

```
void glClearDepth(GLclampd depth);
```

- Löschung ist eine teure (langsame) Operation; Bitmasken zur Nutzung von Spezial-Hw (gleichzeitige Löschung):

```
void glClear(GLbitfield mask);
```

Bsp.: Löschung v. Bild- u.Tiefenspeicher, ggf. gleichzeitig:

```
glClearColor(0.0, 0.0, 0.0, 0.0);
glClearDepth(1.0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

OpenGL: A Drawing Survival Kit

Prof. Dr. Aris Christidis • SS 2018

- Erzwingung der Ausführung eingegebener Anweisungen:

  Hintergrund: Manche Spezial-Hw (z.B. Netzwerk) sammelt oft mehrere Anweisungen, bevor sie sie verarbeitet.

  Erzwingung der Ausführung („Weiter mit nächstem Bild!“):

  ```
  void glFlush(void);/*erzwingt AusfuehrgsStart*/
  ```

  Erzwingung der Fertigstellung („Warte auf Pixelbild!“):

  ```
  void glFinish(void);/*wartetAusfuehrgsEnde ab*/
  ```

- OpenGL-intern werden schließlich alle Grafik-Objekte (Punkte, Linien und Polygone) als geordnete Menge von Eckpunkten beschrieben, immer in (4D-)homogenen Koordinaten, ggf. mit z=0;w=1. Polygone müssen konvex, geschlossen und eben sein (sonst Ergebnis unbestimmt).

  Code-Struktur zur Konstruktion geometrischer Figuren:

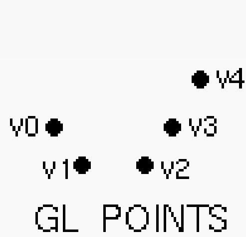
  ```
  void glBegin(GLenum mode);
    void glVertex{234}{sifd}[v](TYPE [*]coords);
  void glEnd(void);
  ```

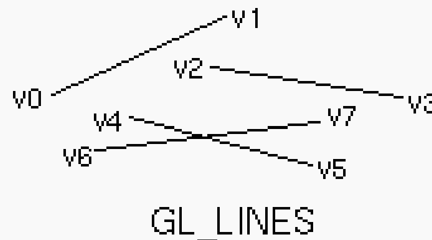
  {...}: eins daraus
  [...]:kann fehlen

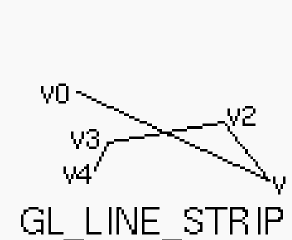
  OpenGL **TYPE** Definition (s.o.)

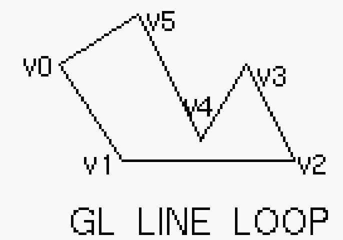
OpenGL: A Drawing Survival Kit

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





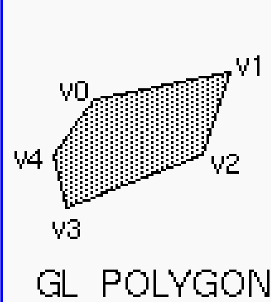


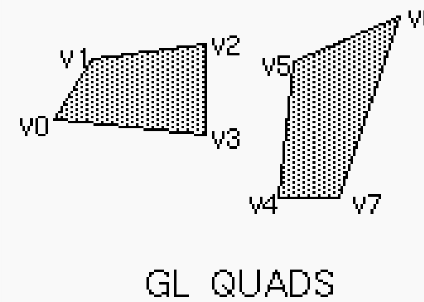


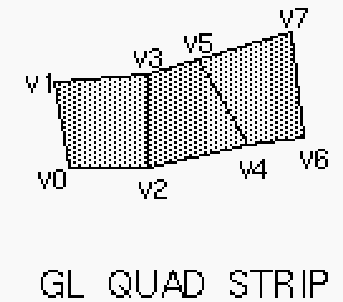
Beispiel:

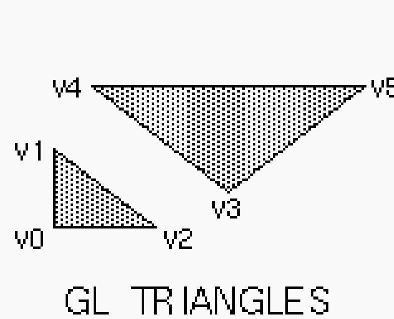
```
glBegin(GL_LINES);
  glVertex2f(0.,0.);
  glVertex2f(0.,3.);
glEnd();//(z=0; w=1)
```

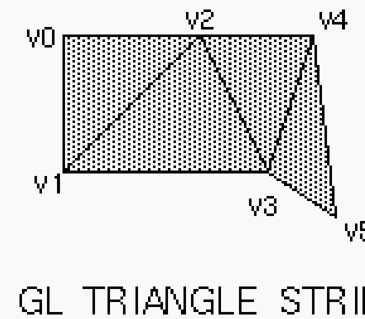
Nur wenige Befehle zwischen **`glBegin()`** u. **`glEnd()`** wirksam, **`glVertex2f*()`** nur dort!











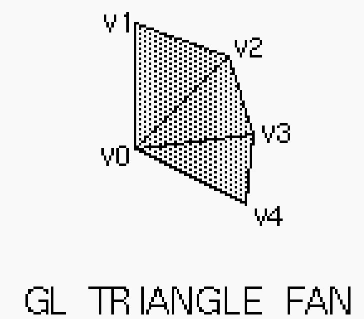


Bild: OpenGL Programming Guide (Addison-Wesley Publishing Company) First Printing, 1993

OpenGL: Punkte, Linien, Polygone

- (Flächen-)Normalen werden in OpenGL/CG aufgefaßt als nach außen (zur Modell-Umgebung) gerichtete, im voraus berechnete Vektoren, die senkrecht zum „Gesamtverlauf" der Modell-Oberfläche an deren Eckpunkten stehen; sie sind somit nicht immer mathematisch exakt definiert.
- Normalen werden ausschließlich Eckpunkten zugeordnet; dabei kann ein Eckpunkt für jede Fläche, zu der er gehört, eine andere Normale bekommen. Setzen der aktuellen Normalen (def.: [**0,0,1**]^T) vor jedem **glVertex*()**-Aufruf möglich:

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

```
void glNormal3{bsidf}(TYPE nx,TYPE ny,TYPE nz);
void glNormal3{bsidf}v (const TYPE *v);//vector
```

(Die ganzzahligen Versionen **b**, **s**, **i** skalieren intern die Wertebereiche linear auf das Intervall [-1.0,1.0])

OpenGL: Punkte, Linien, Polygone

- Beleuchtungseffekte verwenden normierte Normalen:

  $\underline{n} = [\ n_x, n_y, n_z, 0\ ]^T \bullet 1/(n_x^2+n_y^2+n_z^2)^{½}$ (⇨ $|\underline{n}| = 1$)
- Normalen bleiben nach Rotation und Translation normiert; Automatismus für Scherung u.ungleichmäßige Skalierung

  ```
  glEnable(GL_NORMALIZE);
  ```

  bei gleichmäßiger (Gesamt-)Skalierung ändert sich nur der Betrag (Länge) – etwas schnellere Nachskalierung:

  ```
  glEnable(GL_RESCALE_NORMAL);
  ```
- OpenGL-Vers. 1.1 ff.: Daten-Felder (Arrays): Einsparung v. Rechenzeit durch Zusammenfassung v. Koordinaten-, Farb-, Textur- u.a. Daten: Weniger Aufrufe (z.B. **`glVertex*()`**) Nutzung von Synergien (z.B. mehrere Flächen teilen sich einen Eckpunkt oder eine Normale).

  (Vertex Arrays ⇨ eher programmiertechnisch interessant)

Wirkung der Normalen bei unveränderter Kontur:

Lara Croft (retuschiert)

Lara Croft (Original)

OpenGL–Transformationen: Modeling

- Punkt-Trfn = Matrizen-Multiplikationen von links (s.o.):
  $\underline{v}_{neu} = \underline{T}_n \bullet (\dots) \bullet \underline{T}_2 \bullet \underline{T}_1 \bullet \underline{v}_{alt} = \underline{T}_{gesamt} \bullet \underline{v}_{alt}$
- OpenGL: Laden **`mat[16]`**: **`glLoadMatrix{fd}(mat)`**
  Matrizen-Multiplikation: **`glMultMatrix{fd}(mat)`**

  ⇨ eigene Matrizen (z.B.: OpenGL bietet keine Scherung)

  trans-poniert!

  Aber: OpenGL multipliziert von rechts (engl.: *postmultiply*)

  ⇨ Aufbau von $\underline{T}^T_{gesamt}$ in umgekehrter Reihenfolge:

$\underline{C}_0 = \underline{I}$

$\underline{C}_1 = \underline{C}_0 \bullet \underline{T}^T_n$

(...)

$\underline{C}_i = \underline{C}_{i-1} \bullet \underline{T}^T_{n-i+1}$

(...)

$\underline{C}_n = \underline{C}_{n-1} \bullet \underline{T}^T_1 = \underline{T}^T_n \bullet (\dots) \bullet \underline{T}^T_2 \bullet \underline{T}^T_1 = \underline{T}^T_{gesamt}$

```
/*Betr.: Matrix Modell-Trf.:*/
glMatrixMode(GL_MODELVIEW);

glLoadIdentity();

glMultMatrixf(TN); /*...*/
glMultMatrixf(T1);
```

$\underline{C}$: aktuelle (engl. *current*) Positionierungsmatrix; $\underline{I}$: Einheitsmatrix

OpenGL–Transformationen: Modeling

Bequemer (oft: schneller) als **glLoadMatrix***(),**glMultMatrix***():

- Translation eines Objektes (bzw. lokalen Koord.Systems):

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

  (Keine Änderung für **(0.,0.,0.)**)

- Rotation eines Objektes (bzw. lokalen Koord.Systems) um d. Winkel **angle** (in Grad) um eine Achse vom Koord.-Ursprung zum Punkt (**x, y, z**) gegen den Uhrzeigersinn:

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

  (Abstand von Achse = Radius „Umlaufbahn“; Keine Änderg für **(0.,*,*,*)**; **(0.,0.,0.,0.)** zulässig/wirkungslos)

- Skalierungsfaktoren entlang d. Achsen d. Koord.Systems:

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

  (Keine Änderung für **(1.,1.,1.)**; Skalierung mit 0. meist problematisch ⇨ stattdessen: Projektion!)

- Alle OpenGL-Trfn sind Multiplikationen von <u>rechts</u>!

Code-Beispiele:

Vor Viewing & Modeling sicherstellen (ggf. wiederherstellen):

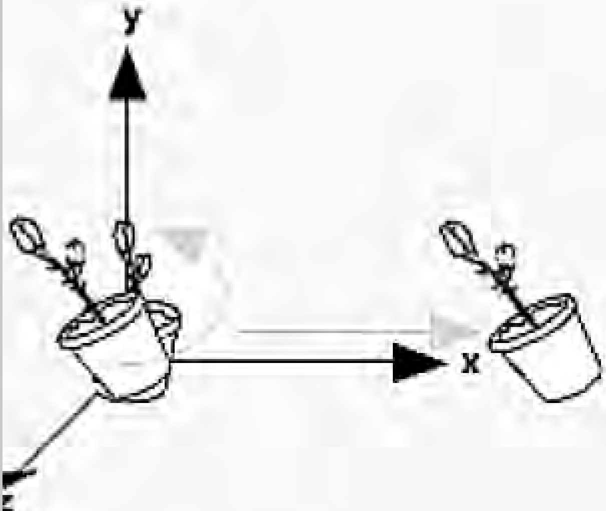
Betr.: Modellierung

Neuer Start

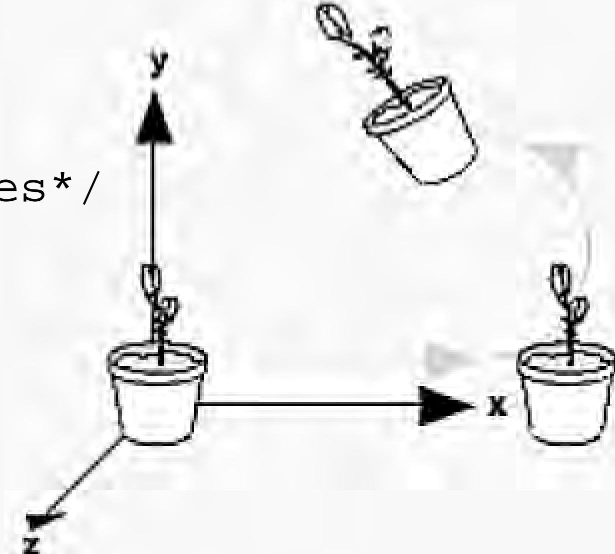
```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
```

Multiplikationen von rechts mit der Transponierten

`glTranslatef (x, 0.,0.);` `glRotatef(angle[Z],0.,0.,1.);`	`glRotatef(angle[Z],0.,0.,1.);` `glTranslatef (x, 0.,0.);`




```
glBegin(GL_POLYGON);
/*(...) */
/*transformed vertices*/
glVertex3fv(v1);
/*(...) */
glEnd();
```



Bilder: OpenGL Programming Guide (Addison-Wesley Publishing Company) First Printing, 1993

OpenGL–Transformationen: Projection

- Projektions-Transformation legt Sichtvolumen-Form fest:
  - Art der Projektion (perspektivisch / orthographisch)
  - Objekte (bzw. O.-Teile), die ins Ergebnis-Bild kommen
- Vor Projektions-Transformation sicherstellen:
  ```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  ```
- Festlegung d. Pyramidenstumpfs für die persp. Projektion und Multiplikation mit der aktuellen Positionierungsmatrix:
  ```
  void glFrustum(GLdouble left  , GLdouble right,
                 GLdouble bottom, GLdouble top  ,
                 GLdouble near  , GLdouble far );
  ```
  Eckpunkte der Deckfläche: **(left, bottom, -near)** und **(right, top, -near)**; **far**: Abstand d.Augenpunkts zur Grundfläche des Pyramidenstumpfs. Alle Größen >0.

Tip: Irreal großes Sichtvolumen (z.B. $10^{-3}...10^{6}$) kann vorläufig helfen, „verlorene“ Objekte („black screen“) wieder zu finden.

OpenGL–Transformationen: Viewport

- OpenGL-Parallelprojektion (Matrix-Def., -Multiplikation):

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

Sicht-Quader mit Eckpunkten: **(left, bottom, -near)**, **(right, top, -near)**, **(left, bottom, -far)**, **(right, top, -far)**; alle Größen >0, **near** ≠ **far**.

- Viewport-Festlegung:

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

mit: **x,y**: linke, untere Fenster-Ecke (def.: 0,0)
width, **height**: Breite, Höhe des generierten Bildes (def.: Breite, Höhe des Fensters)

Verzerrungen, falls **width** : **height** ≠ Breite : Höhe!

OpenGL: Matrix-Stack

- Modular-hierarchische Modellierung / Animation erfordern mehrfache Zwischenablage v.Matrizen (In eigenen Koord. Systemen modellierte/animierte Roboter-Hand /-Unterarm /-Oberarm – jeweils an linker / rechter R.-Schulter).
- Matrizen-Stapelspeicher für ≥32 (**`GL_MODELVIEW`**) bzw. ≥2 (**`GL_PROJECTION`**, **`GL_TEXTURE`**, **`GL_COLOR`**) 4x4-Matrizen.
- Ablage einer Kopie der aktuellen **`glMatrixMode()`**-Matrix als 2.im Stapel (Verschiebung gespeicherter Mat. um eine Position; kein Einfluß auf darauffolgende Berechnungen):

```
void glPushMatrix(void);          /*"Merke Dir!"*/
```

- Aufgeben der aktuellen Matrix (der 1. im Stapel), weitere Verwendung der bisher 2. Stapel-Matrix (Rück-Verschiebung der anderen Matrizen im Stapel):

```
void glPopMatrix(void);      /*"Erinnere Dich!"*/
```

(Fehlermeldungen, falls keine Speicherung mehr möglich bzw. noch keine Ablage erhältlich)

- Grundsätzlich: gleiche Farbdarstellg. für alle Geräte-Pixel; Farbdaten auf mehreren (Hw-)„Bitebenen“ (*bitplanes*) = Teilspeicher in Bildspeicher-Dimension mit je 1 Bit / Pixel; Bildspeicher (*framebuffer*) mit 8 Bitebenen können 2^8, mit 24 (meist: 8R, 8G, 8B) 2^{24}=16.777.216 Farben darstellen.
- **(Color) Index Mode**: Einmalig (je Fenster) einstellbare Farb-Palette (-Tabelle: engl. *look up table* bzw. *color map*)

  Bildspeicher (*color buffer*), meist für 256 Farben/Grautöne („*8-bit buffer*“), jeweils wählbar über Farb-/Grauton-Index
- **RGBA Mode**: Individuelle Pixel-Farbgebung durch R/G/B-Farbwerte, Alpha (1.=opak, def.) für Transparenz u./o. Farbmischung (*blending*) / lineare Berechnung, ggf. Gamma-Korrektur/ RGBA aufwendiger, aber verbreiteter
- Farbmodus nur beim Start (Initialisierung) einmal wählbar; Farben werden Eckpunkten (*vertices*) zugewiesen