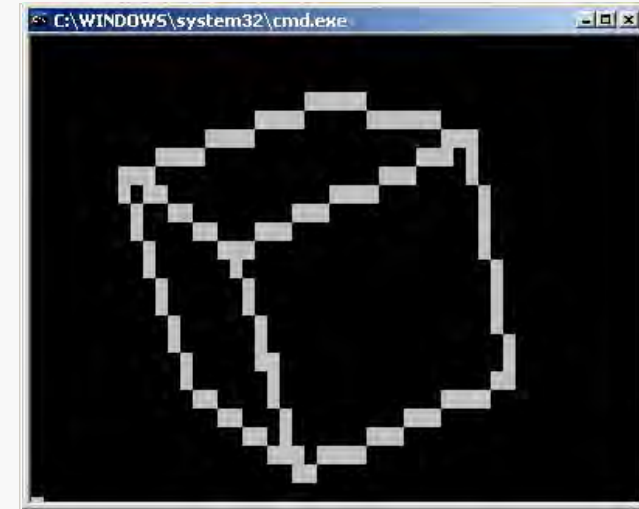


3D-Sicht, Projektionen

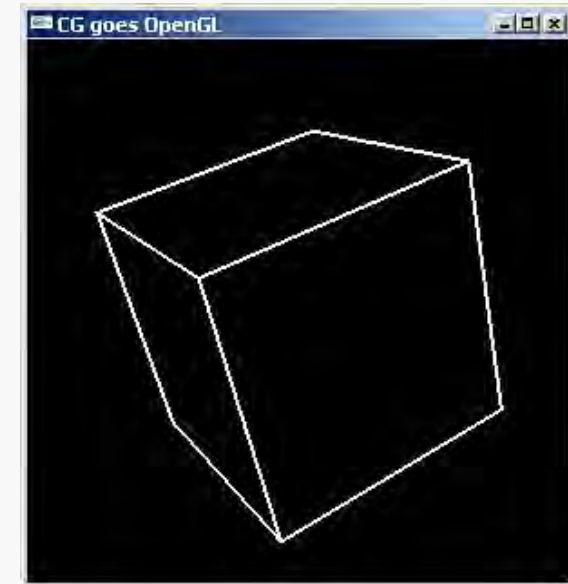
Typische Struktur eines Grafik-Programms, das erst in einem „verborgenen“ Speicher zeichnet, den es nach Fertigstellung der Grafik sichtbar macht (Double Buffering):



```
void draw (void)  
{ /*Hintergrund-Puffer loeschen:*/  
    conClrBuff();  
    /*Erstellung der Grafik (im Hintergrund):*/  
    conDrawCGFobj(&obj,eyez);  
    /*Hintergrund-Puffer sichtbar machen:*/  
    conSwapBuff();  
    return;  
}
```

Typische Struktur eines Grafik-Programms, das nach aktuellem Industrie-Standard mit Double Buffering zeichnet:

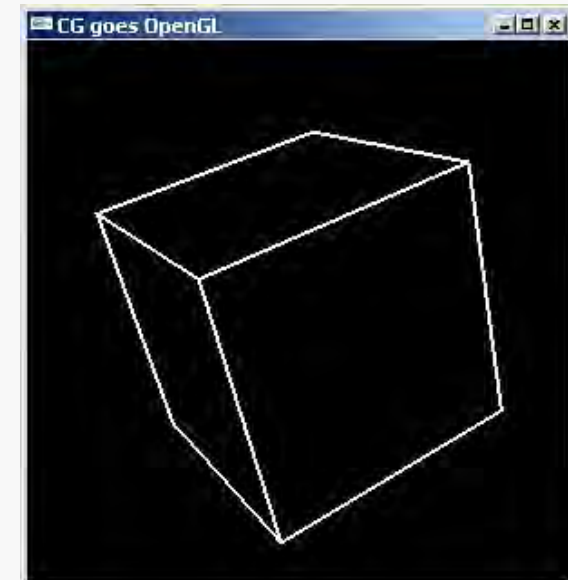
```
void draw (void)
{ /*Hintergrund-Puffer loeschen:*/
  glClear(GL_COLOR_BUFFER_BIT);
  /*Erstellung der Grafik (im Hintergrund):*/
  drawBox( );
  /*Hintergrund-Puffer sichtbar machen:*/
  glutSwapBuffers( );
  return;
}
```



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;
}
```



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

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

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
```

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

```
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);
```

OpenGL

```
/*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];
```

„Open Graphics Library“ (OpenGL: ® Silicon Graphics Inc.)
www.opengl.org – Aktuelle Version: 4.1 (Aug. 2010)

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.labri.fr/perso/knoedel/cmsimple/data/pdf/redbook.pdf

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: 7.8 (2010)

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

HTML-Online-Dokumentation zu GLUT:

- Links zu Quellen und Dokumentation:

www.opengl.org/resources/libraries/glut

darunter erreichbar:

- 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.th-mittelhessen.de/christ/>Computergrafik>aktuell>

- Einführung in GLUT:

<http://mindfuck.de-brauwer.be/articles/glut/>

- Quellen u. Dokumentation für MS-Windows zu GLUT:

www.xmission.com/~nate/glut.html

Funktionalität durch GLUT:

- Einrichtung und Handhabung mehrerer Grafik-Fenster
- Ereignisverarbeitung durch Callbacks (*event processing*)
- Unterstützung vielfältiger Eingabegeräte
- Routinen zur Einhaltung von Zeitvorgaben (*timer*)
- Ausnutzung 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

- Definition: Als **Ereignis** (*engl. event*) bezeichnen wir jedes Vorkommnis (*occurrence*), das eine nicht-sequentielle Bearbeitung eines Programms bewirkt.

Ein solches Vorkommnis kann eine Zustandsänderung oder die Erfüllung einer Bedingung ($x == y$) sein, was die Abarbeitung einer **if**-Abfrage im Programm bewirkt.

- **Synchrone** Ereignisse treten zu vorhersagbaren Zeitpunkten, **asynchrone** zu nicht-vorhersagbaren ein.

d.h.:
Programm-
stelle!

Synchrone Ereignisse sind z.B. **if**-Abfragen im prozedur-orientierten Programm: sie treten vorhersagbar ein (z.B.: ... Zeilen ab Programmbeginn).

Asynchrone Ereignisse sind z.B. Tasten- o. Maus-Aktionen (Zeit/Situation nicht-vorhersagbar: „Abbrechen“ vs. „OK“).

- Ereignisgesteuerte Arbeitsweise bedient sich asynchroner Ereignisse.

- Zentraler Begriff bei Sw-Plattformen (ebenso: Sw-Umgebungen)
Callback [-Funktion]: **Funktion** des **Anwendungs**programms, die in vorgegebenen Situationen **von der Plattform aufgerufen** wird.

Hintergrund:

- Multitasking \Leftrightarrow Teilen von Ressourcen (I/O, Rechenzeit etc.)
 - Koordination nur durch Plattform möglich (dort Information über laufende Prozesse und Ressourcen-Bedarf)
 - \Rightarrow Maßnahmen nach Wiederzuweisung von Ressourcen sind nur durch die Plattform einzuleiten – aber:
 - Code nur als Bestandteil der Applikation sinnvoll (nur dort Wissen über Umgang mit Ressourcen vorhanden)
 - \Rightarrow Einrichtung einer Plattform-Routine zur Aufnahme eines Zeigers auf eine (Callback-) Funktion, mit der die Applikation der Plattform mitteilt, welche Funktion aufzurufen ist.
- („Callback“ = Rückruf – eigentlich: Aufruf der Callback-Funktion)

GLUT

- bietet ein (fast) komplettes API für d. native Fenstersystem
 - ↳ Sw-Entwicklung ohne Einarbeitung in Fenstersystem
- überläßt Fenster-Einrichtung dem nativen Fenstersystem
 - ↳ Erhaltung des plattformeigenen Look & Feel
- wird (wurde) regelmäßig aktualisiert
 - ↳ Befolgung von Trends, Beseitigung von Bugs
- ist kostenlos erhältlich inkl. Source-Code
 - ↳ Verwendbarkeit in Ausbildung
 - ↳ Anpassung an individuelle Sw-Entwicklung möglich

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

Design-Philosophie: (Forts.)

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

↳ Einschränkung gleichzeitiger Verwendung weiterer Ereignisverarbeitung, 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


- Logische Organisation in 10 „Sub-APIs“:



Echtzeit !
(*real time*)

1. Initialisierung:

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

A red exclamation mark icon inside a small square frame, pointing towards the first line of code.

`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)*/

notwendig

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<=Fenster.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..*/
```

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 uebergegebener 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:

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 msec,(*func)(int val),val);  
/*msec Millisekunden spaeter: Aufruf func(val)*/
```

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:

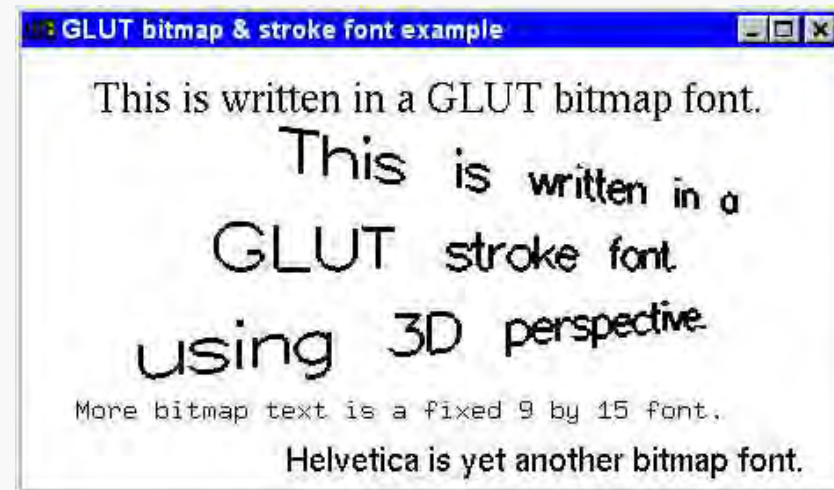
```
int glutGet(GLenum state);  
    /* state: 1 v.35 Zustandskennungen, z.B.GLUT_WINDOW_X */  
int glutLayerGet(GLenum info);          /* Overlay-Nutzung */  
int glutDeviceGet(GLenum info);  
    /* info: 1 von 10 GLUT_HAS_... (Maus etc.) */  
int glutGetModifiers(void);  
    /* GLUT_ACTIVE_SHIFT,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

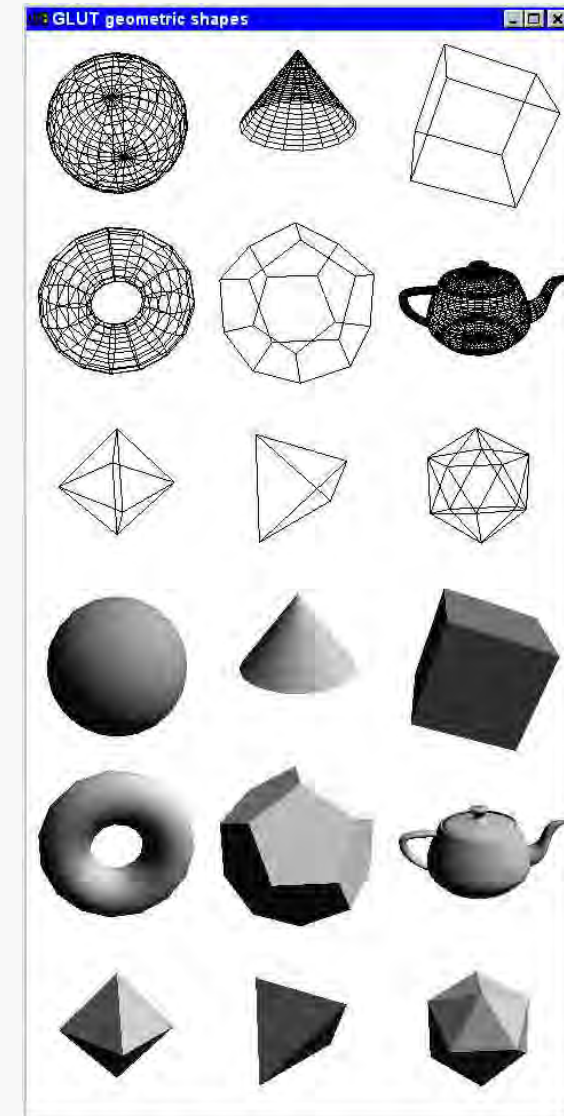
```
void glutStrokeCharacter(void *font, int character);
```

font aus: GLUT_STROKE_ROMAN
GLUT_STROKE_MONO_ROMAN

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

10. Wiedergabe Geometrischer Figuren (*Geometric Shape Rendering*)—18 Routinen:

<code>glutSolidSphere,</code>	<code>glutWireSphere</code>
<code>glutSolidCube,</code>	<code>glutWireCube</code>
<code>glutSolidCone,</code>	<code>glutWireCone</code>
<code>glutSolidTorus,</code>	<code>glutWireTorus</code>
<code>glutSolidDodecahedron,</code>	<code>glutWireDodecahedron</code>
<code>glutSolidOctahedron,</code>	<code>glutWireOctahedron</code>
<code>glutSolidTetrahedron,</code>	<code>glutWireTetrahedron</code>
<code>glutSolidIcosahedron,</code>	<code>glutWireIcosahedron</code>
<code>glutSolidTeapot,</code>	<code>glutWireTeapot</code>



Anmerkungen zu GLUT:

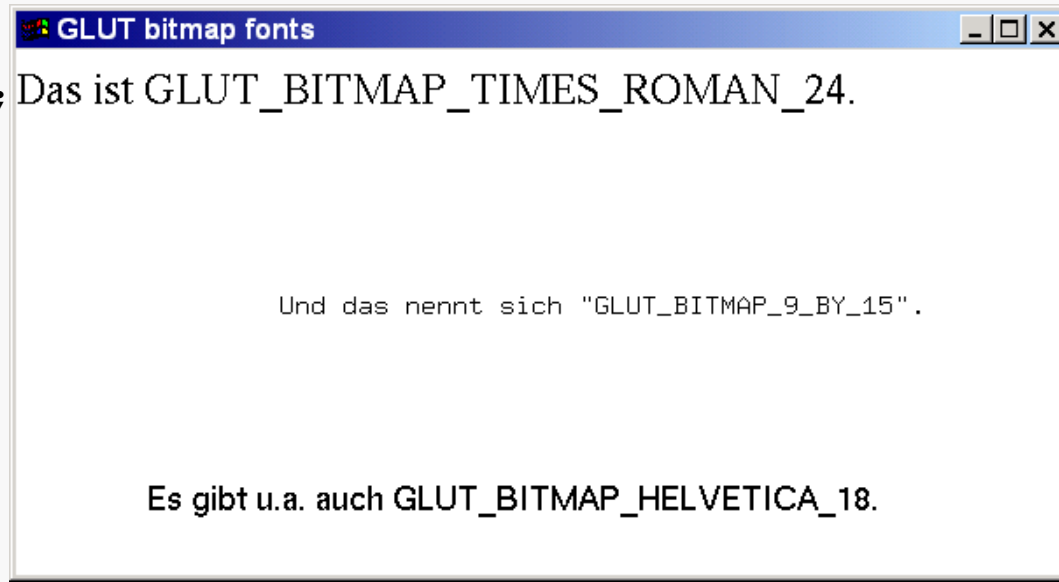
- GLUT-Koordinaten (Bildschirm, Fenster) in Pixel; **Ursprung oben links** (wie d. meisten Fenstersysteme \Leftrightarrow OpenGL: math. Koord.).
- Fenster-, Menü- u. Menüpunkt-**Kennungen beginnen mit 1**.
- GLUT-**Header** enthält OpenGL- u. GLU-Header:
`#include <GL/glut.h>`
- **glutInit** soll zur 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).

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

- Cursor setzen an Position (x, y): $-1. \leq x, y \leq +1.$:
`void glRasterPos2f(GLfloat x, GLfloat y);`
Untere linke Fenster-Ecke bei $(-1.; -1.)$
- Vor Fortsetzung vorausgegangene Aufrufe ausführen:
`void glFlush(void); /*(vgl. fflush)*/`
- Fenster löschen:
`void glClear(GLbitfield mask);`
Für `mask` hier immer: `GL_COLOR_BUFFER_BIT`
- Zeichenfarbe wählen (GLUT-Def.: $(1., 1., 1.)$):
`void glColor3f(GLfloat red, GLfloat green, GLfloat blue);`
- Lösch-/Hintergrundfarbe wählen (GLUT-Def.: $(0., 0., 0., 1.)$):
`void glClearColor(GLclampf red, GLclampf green, GLclampf blue, GLclampf alpha);`
Farbkomponenten werden intern auf den Bereich $[0,1]$ begrenzt („clamped“).

Beispiel: Text-Ausgabe mit GLUT:

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 300);
    glutCreateWindow("GLUT bitmap fonts");
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f(0., 0., 0.);
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```



Beispiel: Text-Ausgabe mit GLUT (Forts.)

```
#include <GL/glut.h>
void bmpOut(GLfloat x, GLfloat y, void *font, char *string)
{ int len, i;
  glRasterPos2f(x, y);
  len = (int) strlen(string);
  for (i = 0; i < len; i++)
    glutBitmapCharacter(font, string[i]);
}
void display(void)
{ glClear(GL_COLOR_BUFFER_BIT);
  bmpOut(-1.f, 0.8f, GLUT_BITMAP_TIMES_ROMAN_24,
        "Das ist GLUT_BITMAP_TIMES_ROMAN_24.");
  bmpOut(-0.5f, 0.0f, GLUT_BITMAP_9_BY_15,
        "Und das nennt sich \"GLUT_BITMAP_9_BY_15\".");
  bmpOut(-0.75f, -0.75f, GLUT_BITMAP_HELVETICA_18,
        "Es gibt u.a. auch GLUT_BITMAP_HELVETICA_18.");
  glFlush();
}
```

