

- Grafik-Bibliothek mit einigen hundert Befehlen `gl...()` (div. Ausführungen) – angeschlossener „Aufsatz“: OpenGL Utility Library (GLU), ca. 50 Befehle `glu...()`
- Plattform-unabhängig (Hardware, Fenster-/ Betriebssystem.), 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 **Lichtintensitäten** bei gegebener **Farbe** oder **Textur** (=aufgesetztem Bild)
 - Erzeugung eines **Pixel-Bildes** aus der geometrisch-farblichen Beschreibung („Rasterisierung“)
- **Rendering**: (Wiedergabe, Darstellung): Erzeugg. digitaler (Pixel-)Bilder aus log.-mathem. Modell-Beschreibungen.

- 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.

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

Puffer löschen – hier: Bild- und Tiefenspeicher

- Löscharben-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);
```

- Implementierungshinweis zu Bitmasken :

Programmeinstellungen mit Bit-Charakter (Ein / Aus) werden oft sehr effizient als Einzel-Bit-Werte eines ganzzahligen Datentyps (`unsigned int`, `int`, `char`,...) codiert – ein Beispiel:

```
#define IRGENDWO    0 /*Werte fuer uTell:      */
#define HIER       1 /*auch: 1<<0 (2er Potenz)*/
#define JETZT      2 /*  "    1<<1      "      */
#define IRGENDWANN 0 /*entbehrlich*/

void WannUndWo (char uTell)
{ if (uTell & HIER) printf("Hier ");
  else             printf("Irgendwo ");
  if (uTell & JETZT) printf("u. jetzt!\n");
  else             printf("u. irgendwann!\n");
}

int main()
{ WannUndWo(HIER | JETZT); /*Ausgabe:"Hier u. jetzt!" */
  WannUndWo(HIER | IRGENDWO); /*Ausgabe:"Hier u. irgendwann!" */
  _getch(); }
```

```
#include <conio.h>
#include <stdio.h>
```

Farben unter OpenGL (additive Farbmischung):

`glColor3f(1.,1.,0.);`

`glColor3f(1.,0.,0.);`

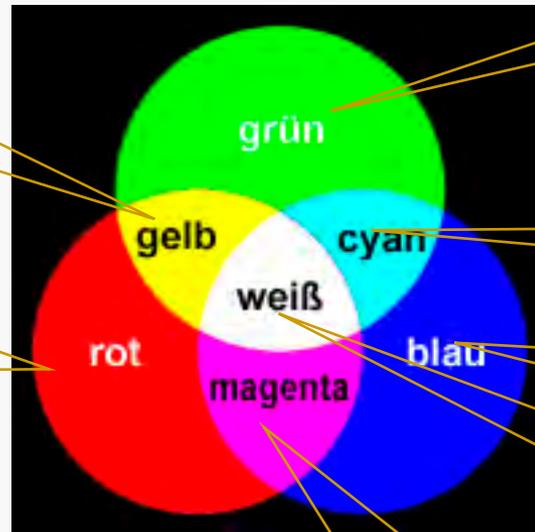
`glColor3f(0.,1.,0.);`

`glColor3f(0.,1.,1.);`

`glColor3f(0.,0.,1.);`

`glColor3f(1.,1.,1.);`

`glColor3f(1.,0.,1.);`



```
glColor3f(1.0, 0.0, 0.0); /*red */
glColor3f(0.0, 1.0, 0.0); /*green */
glColor3f(1.0, 1.0, 0.0); /*yellow */
glColor3f(0.0, 0.0, 1.0); /*blue */
glColor3f(1.0, 0.0, 1.0); /*magenta*/
glColor3f(0.0, 1.0, 1.0); /*cyan */
glColor3f(1.0, 1.0, 1.0); /*white */
```

- 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: {...}: eins daraus

```
void glBegin(GLenum mode);
```

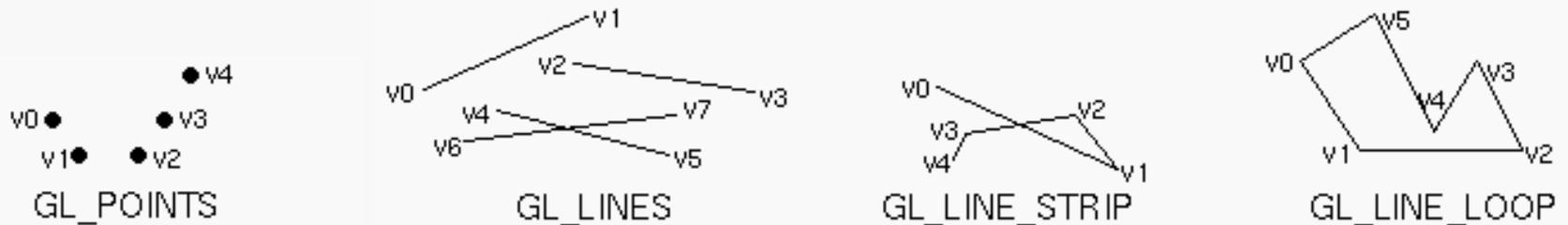
```
void glVertex{234}{sifd}[v](TYPE [*]coords);
```

```
void glEnd(void);
```

[...]:kann fehlen

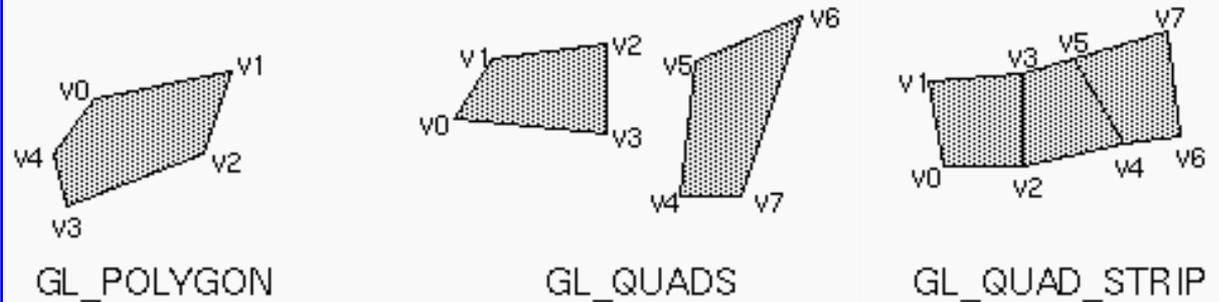
OpenGL TYPE Definition (s.o.)

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

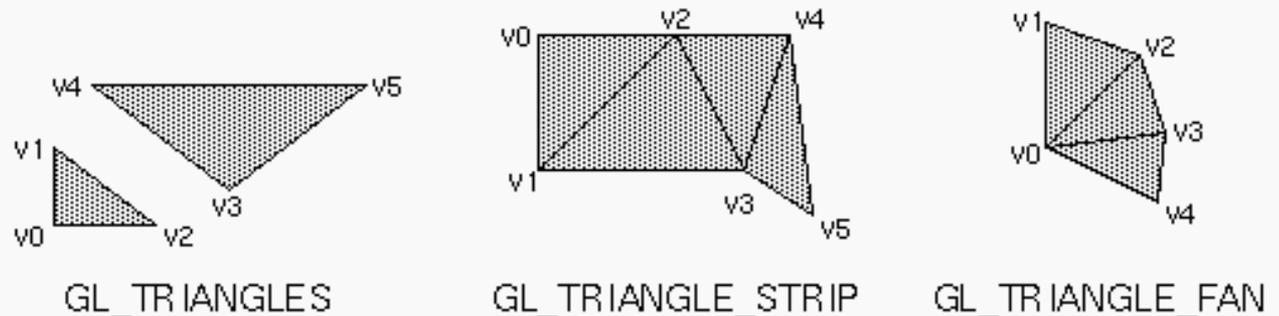


Beispiel:

```
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!

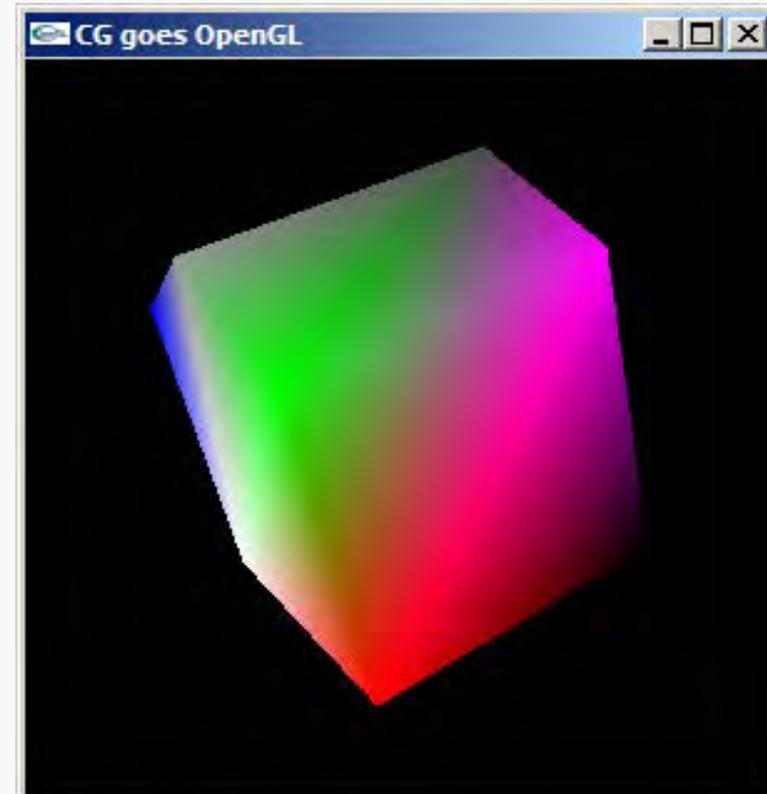


Valid Commands between `glBegin()` and `glEnd()` (OpenGL 1.1):

Command	Purpose of Command
<code>glVertex*()</code>	set vertex coordinates
<code>glColor*()</code>	set current RGBA color
<code>glIndex*()</code>	set current color index
<code>glNormal*()</code>	set normal vector coordinates
<code>glTexCoord*()</code>	set texture coordinates
<code>glEdgeFlag*()</code>	control drawing of edges
<code>glMaterial*()</code>	set material properties
<code>glArrayElement()</code>	extract vertex array data
<code>glEvalCoord*()</code> , <code>glEvalPoint*()</code>	generate coordinates
<code>glCallList()</code> , <code>glCallLists()</code>	execute display list(s)

Übung:

In einem GLUT-Fenster sollen mit OpenGL geladene 3D-Modelle dargestellt werden mit den Funktionalitäten, die im Bedienungsmenü vorgesehen sind.



ObjElabGL.exe

- Einstellbare Punkt- und Strichstärken (in Pixel):

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

Nachkomma-Stellen bei Anti-Aliasing („anti-éiliæsing“):

```
glEnable (GL_POINT_SMOOTH); bzw.  
glEnable (GL_LINE_SMOOTH);
```

Dazu notwendig: Farbmischung und Mischungsfunktion:

```
glEnable (GL_BLEND); source (incoming) destination (current)  
glBlendFunc(GL_SRC_ALPHA,GL_ONE_MINUS_SRC_ALPHA);
```

- Musterung/Strichelung von Linien nach Bitmustern (mit Berücksichtigung der Übergänge innerhalb eines **glBegin**/**glEnd**-Paars – Bits nach steigender Wertigkeit ausgewertet)

```
glEnable(GL_LINE_STIPPLE);  
void glLineStipple(GLint factor,GLushort pattern);  
/* z.B.: glLineStipple(2, 0x3F07); */
```

Ohne Anti-Aliasing (*)



Mit Anti-Aliasing



(*) „éiliæsing“ (< alias < áλλωç = anders): Veränderung, Entstellung

Beispiel: Strichmuster-Verarbeitung d. **pattern**-Parameters in

```
glLineStipple(2, 0x3F07); //(factor, pattern);
```

Muster-Darstellung binär: $3F07_{16} = 0011\ 1111\ 0000\ 0111_2$

Abarbeitung der 16 Bit einzeln -gestreckt durch **factor**-
von rechts ← nach links:

(2x3=) 6 Pixel setzen / (2x5=) 10 Pixel auslassen /

(2x6=) 12 Pixel setzen / (2x2=) 4 Pixel auslassen

Weitere Beispiele:

	PATTERN	FACTOR	
0000 0000 1111 1111	0x00FF	1	_____
	0x00FF	2	_____
0000 1100 0000 1111	0x0C0F	1	____ _
	0x0C0F	3	_____
1010 1010 1010 1010	0xAAAA	1	- - - - -
	0xAAAA	2	____ _
	0xAAAA	3	____ _
	0xAAAA	4	____ _

Linien gezogen v.l.n.r.

- Zeichenmodi für Polygone:

```
void glPolygonMode(GLenum face, GLenum mode);
```

Zulässige Werte für **face** :

GL_FRONT_AND_BACK (default), **GL_FRONT**, **GL_BACK**

Zulässige Werte für **mode**:

GL_POINT, **GL_LINE**, oder **GL_FILL** (default).

- Eliminierung von Objektflächen („Blick in die Schachtel“):

```
void glCullFace(GLenum mode);
```

Zulässige Werte für **mode**:

GL_FRONT, **GL_BACK** (default), **GL_FRONT_AND_BACK**

wirksam zwischen: **glEnable** (**GL_CULL_FACE**);

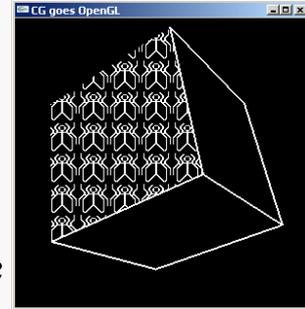
und: **glDisable**(**GL_CULL_FACE**);

- Drehsinn sichtbarer Flächen (default: „counterclockwise“):

```
void glFrontFace(GLenum mode);
```

Zulässige Werte für **mode**: **GL_CCW**, **GL_CW**

- Alternativ zur Einfarbigkeit (=default): Polygon-Musterung
zwischen: `glEnable (GL_POLYGON_STIPPLE);`
und: `glDisable(GL_POLYGON_STIPPLE);`
bei: `glPolygonMode(face, GL_FILL); //default`
`void glPolygonStipple(const GLubyte *mask);`
setzt das aktuelle 32x32-Bit-Füllmuster, das **am Fenster links unten aligniert** bleibt: 0 steht für „Pixel auslassen“, 1 für „Pixel setzen“ (v.links → n.rechts, v.unten ↑ n.oben).

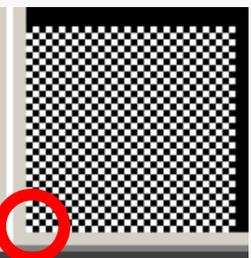


Beispiel: Definition einer Halbton-Maske [$A_{16}=1010_2$; $5_{16}=0101_2$]

`GLubyte halftone[] =`

```
{ 0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,  
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,  
  /* (...ueber mehrere Zeilen...) */  
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55,  
  0xAA, 0xAA, 0xAA, 0xAA, 0x55, 0x55, 0x55, 0x55 };
```

CubeElabFly.exe, CubeElabHalftone.exe



- (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):

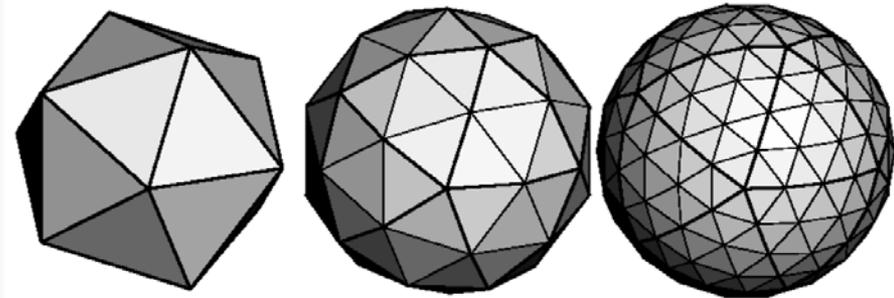
OpenGL **TYPE** Definition (s.o.)

```
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 $[-1.0, 1.0]$)

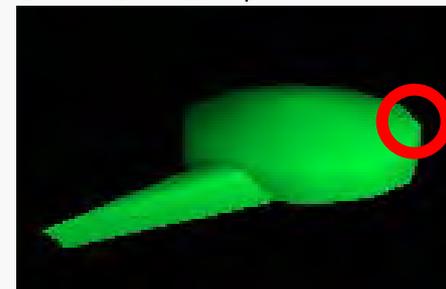
Beispiele:

```
/*Zeichnet Dreiecke auf der Projektionsebene:*/  
void SingleTri(float *v1,float *v2,float *v3)  
{ glBegin(GL_TRIANGLES);/*Normale aus Voreinst.*/  
  glNormal3f(0.,0.,1.);  
  glVertex3fv(v1);  
  glVertex3fv(v2);  
  glVertex3fv(v3);  
  glEnd();  
}
```



20 Dreiecke 80 Dreiecke 320 Dreiecke

```
/*Zeichnet approx. Kugel um Koord.-Ursprung:*/  
void SphereTri(float *v1,float *v2,float *v3)  
{ glBegin(GL_TRIANGLES);    /*Ortsvektor=Normale*/  
  glNormal3fv(v1); glVertex3fv(v1);  
  glNormal3fv(v2); glVertex3fv(v2);  
  glNormal3fv(v3); glVertex3fv(v3);  
  glEnd();  
}
```



Wirkung der Normalen bei unveränderter Kontur:



Lara Croft (retuschiert)



Lara Croft (Original)

- Beleuchtungseffekte verwenden normierte Normalen:

$$\underline{n} = [n_x, n_y, n_z, 0]^T \cdot 1/(n_x^2+n_y^2+n_z^2)^{1/2} \quad (\Rightarrow |\underline{n}| = 1)$$

- 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 \Rightarrow eher programmiertechnisch interessant)