Technische Universität Dresden
Fakultät Informatik
Prof. Dr.-Ing.habil. Orlamünder

Konzepte der OpenGL

OpenGL ist ein Software-Interface zur Hardware. Zweck ist die Darstellung von zwei- und dreidimensionalen Objekten mittels eines Bildspeichers. Diese Objekte sind entweder Images, die aus Pixeln zusammengesetzt sind, oder geometrische Objekte, die durch Vertices (Raumpunkte) beschrieben sind.

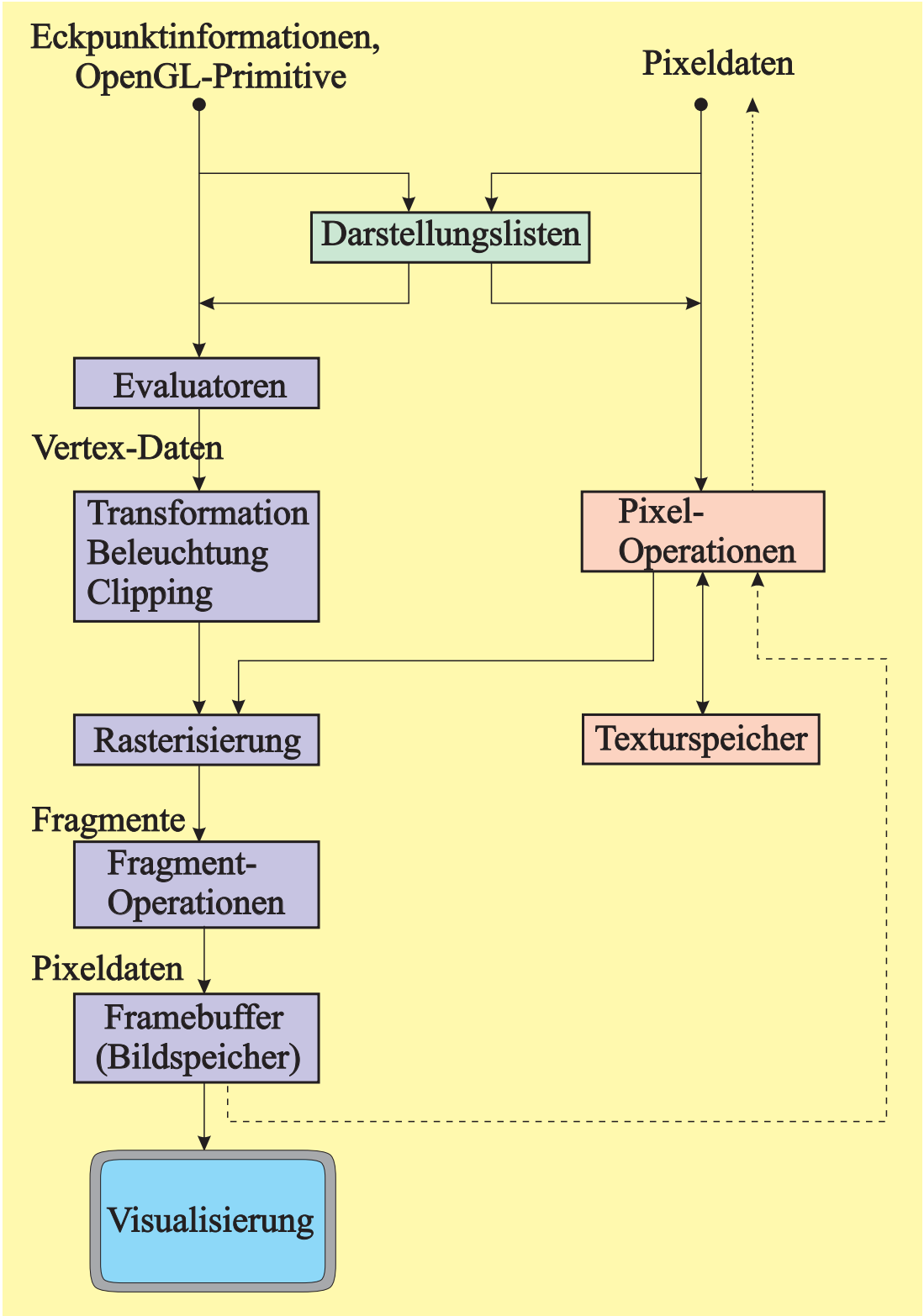
1. Client-Server-Architektur
   Client = Anwendung
   Server = OpenGL
   Netzwerkfähigkeit

2. Zustandsautomat
   Steuerung durch einen inneren, über Befehle meist einstellbaren Zustand

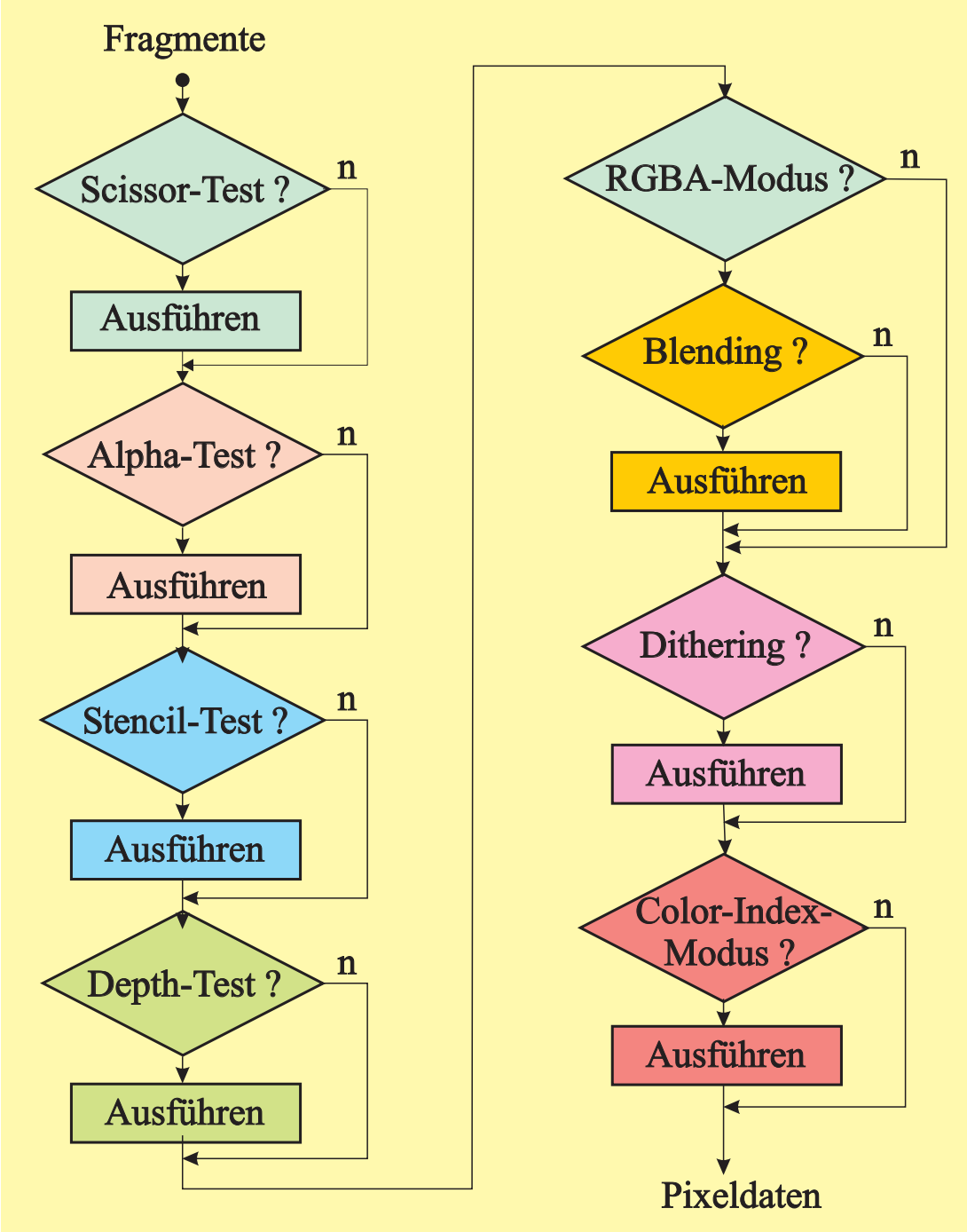
3. Hardwareorientierung
   Interface ist als Programmierschnittstelle hardwareunabhängig, die OpenGL-Implementierung nutzt Hardwarefähigkeiten jedoch bestmöglich.

4. Einbettendes Windows-System
   OpenGL ist von Betriebssystem und Eingabemöglichkeiten unabhängig definiert. Sie nutzt die Möglichkeiten des Fenstersytems.

Renderingpipeline



Fragmentoperationen



Grafische Primitive (1)

Grafische Primitive (Punkte, Linien, Polygone) werden in OpenGL durch Vertices (Eckpunkte) beschrieben. Die Interpretation derselben erfolgt durch spezielle Funktionen und Parameter.

Setzen eines Vertex (Auswahl):

void **glVertex2f**(GLfloat ***x***, Glfloat ***y***)
void **glVertex3f**(GLfloat ***x***, GLfloat ***y***, GLfloat ***z***)
void **glVertex4f**(GLfloat ***x***, GLfloat ***y***, GLfloat ***z***, GLfloat ***w***)

Zeichnen eines grafischen Primitivs (Auswahl):

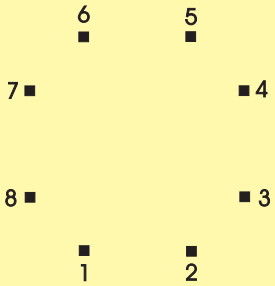
void **glBegin**(Glenum ***mode***);
:
void **glEnd**(void);

Die Werte von ***mode*** bestimmen, wie die Vertex-Liste interpretiert wird. Zwischen **glBegin** und **glEnd** dürfen nur bestimmte Anweisungen stehen.

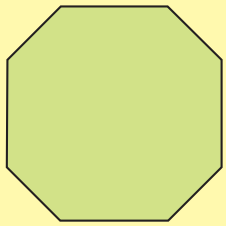
Grafische Primitive (2)

Werte und Wirkung von ***mode***	
GL_POINTS	Es werden Punkte an den durch die Vertices spezifizierten Positionen gezeichnet.
GL_LINES	Jeweils 2 aufeinanderfolgende Vertices beschreiben eine Linie.
GL_LINE_STRIP	Die Punkte werden entsprechend ihrer Reihenfolge verbunden.
GL_LINE_LOOP	Die Linienfolge wird am Ende geschlossen.
GL_POLYGON	Die Vertices beschreiben eine gefüllte Fläche.
GL_QUADS	Jeweils 4 aufeinanderfolgende Punkte bilden ein Viereck.
GL_QUAD_STRIP	Die Vertices bilden eine Folge von Vierecken. (A,B,C,D)(C,D,E,F)(E,F,G,H)....
GL_TRIANGLES	Jeweils 3 aufeinanderfolgende Punkte bilden ein Dreieck.
GL_TRIANGLE_STRIP	Die Vertices bilden eine Folge von Dreiecken (1,2,3)(2,3,4)(3,4,5)...
GL_TRIANGLE_FAN	Die Punkte spannen einen Fächer auf und der der erste Punkt bildet das Zentrum. (1,2,3)(1,3,4)(1,4,5)...

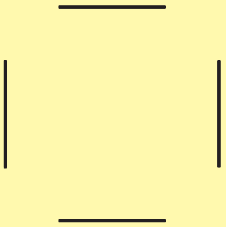
Grafische Primitive (3)



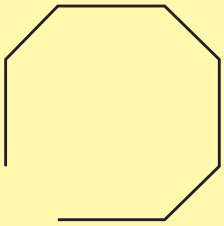
GL_POINTS



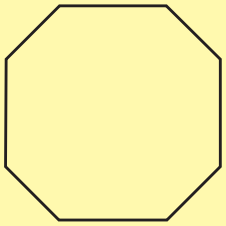
GL_POLYGON



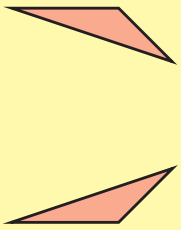
GL_LINES



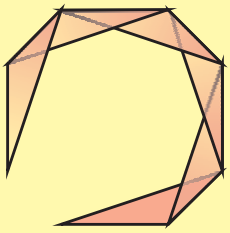
GL_LINE_STRIP



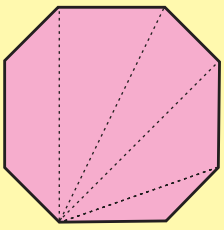
GL_LINE_LOOP



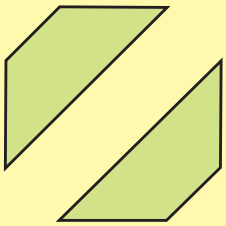
GL_TRIANGLES



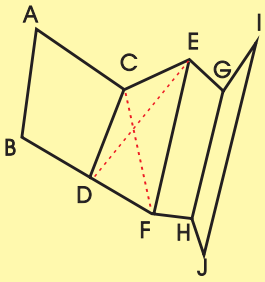
GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP

Grafische Primitive (4)

Zwischen **glBegin** und **glEnd** zugelassene OpenGL-Befehle

glVertex	Definition der Koordinaten eines Eckpunktes
glColor	Definition der Farbe eines Eckpunktes
glNormal	Definition des aktuellen Normalenvektors
glMaterial	Definition der Oberflächenbeschaffenheit
glTexCoord, glEvalCoord, glEvalPoint	Definition der Texturkoordinaten bzw. der über Evaluatoren berechneten Koordinaten
glEdgeFlag	Festlegung, ob eine Kante zur Kontur eines Objekts gehört
glCallList, glCallLists	Aufruf einer oder mehrerer Darstellungs-listen

Anmerkung: Die Befehle können in sehr unterschiedlicher Form bezüglich des Typs und der Anzahl der Parameter auftreten. Außerdem ist die Notation der Parameter einzeln oder als Vektor möglich.

Grafische Primitive (5)

Attribute für Primitive (Auswahl)

Setzen der Punktgröße

```
void glPointSize(GLfloat size)          // Standard: size=1
```

Setzen von Linienbreite und -typ

```
void glLineWidth(GLfloat width)         // Standard: width=1

void glLineStipple(GLint factor,        // Standard: factor=1, Wiederholung der
                                        // einzelnen Bits factor-mal
                   Glushort pattern)    // Standard: pattern=1111 1111 1111 1111
                                        // pattern enthält 16 Bit (1 aktiviert die
                                        // Darstellung, das niedrigste Bit wird
                                        // zuerst gezeichnet)
```

Der Linientyp muß mit **glEnable**(GL_LINE_STIPPLE) aktiviert, bzw. Mit **glDisable**(GL_LINE_STIPPLE) deaktiviert werden.

Grafische Primitive (6)

Attribute für Primitive (Auswahl)

Steuerung der Darstellung von Polygonen

```
void glPolygonMode(GLenum face,   // Standard: face=GL_FRONT
                                  // Werte: GL_FRONT, GL_BACK oder
                                  //        GL_FRONT_AND_BACK
                   GLenum mode)   // Standard: mode=GL_FILL
                                  // Werte: GL_POINT, GL_LINE,
                                  //        GL_FILL
```

Automatisches Entfernen von Vorder- und Rückseiten

```
void glCullFace(GLenum mode)      // Standard: mode=GL_BACK
                                  // Werte: GL_FRONT, GL_BACK,
                                  //        GL_FRONT_AND_BACK
```

Das automatische Entfernen muß mit **glEnable**(GL_CULL_FACE) aktiviert, bzw. mit **glDisable**(GL_CULL_FACE) deaktiviert werden.

Erzeugen von Mustern

```
void glPolygonStipple(const Glubyte *mask)
                                  // mask enthält in 32*32 Bit das
                                  // darzustellende Muster. Dieses
                                  // ist noch vom Wert der Funktion
                                  // glPixelStore(...) abhängig
```

Das Muster muß mit **glEnable**(GL_POLYGON_STIPPLE) aktiviert und mit **glDisable**(GL_POLYGON_STIPPLE) deaktiviert werden.

Bezierkurven

Spezifikation und Darstellung

Spezifikation

```
void glMap1f (GLenum      target,   // Typ der Kontrollpunkte (Tabelle)
              GLfloat     u1,       // Anfangswert der Bezier-Kurve
              GLfloat     u2,       // Endwert der Bezier-Kurve
              GLint       stride,   // Anzahl der zu berechnenden Werte
                                    // zwischen den Kontrollpunkten
              GLint       order,    // Ordnung der Bezierkurve
              const GLfloat *points, // Vektor der Kontrollpunkte
               );
```

Darstellung

```
void glEvalCoord1f (Glfloat u);      // Berechnung der Bezierkurve an der
                                     // Position u   (u1<=u<=u2)
```

Werte von ***target*** (Auswahl):

GL_MAP1_VERTEX_3	3 Koordinaten x,y,z
GL_MAP1_VERTEX_4	4 Koordinaten x,y,z,w
GL_MAP1_COLOR_4	4 Farbwerte R,G,B,A

Beispiel:

```
glEnable(GL_MAP1_VERTEX_3);
glMap1f(GL_MAP1_VERTEX_3,0,1,3,10,@P);
glBegin(GL_LINE_STRIP);
   glColor3f(1,1,0);
   for i:=0 to 50 do glEvalCoord1f(i/50);
glEnd;
```

NURBS- und B-Spline-Kurven

Spezifikation und Darstellung

Erzeugung eines neuen NURBS-Objekts und Rückgabe des Zeigers darauf

```
GLUNurbs*  gluNewNurbsRenderer(void);
```

Definition der NURBS-Eigenschaften

```
void gluNurbsProperty(GLUNurbs*  nurb,      // erzeugtes NURBS-Objekt
                      GLenum property,      // Eigenschaft (Tabelle)
                      GLfloat value         // Eigenschaftswert (Tabelle)
                     );
```

Tabelle NURBS-Eigenschaften und Eigenschaftswerte (Auswahl)

Eigenschaften	mögliche Werte für value
GLU_SAMPLING_TOLERANCE	value>0
GLU_DISPLAY_MODE	GLU_FILL | GLU_OUTLINE_POLYGON | GLU_OUTLINE_PATCH
GLU_CULLING	GL_TRUE | GL_FALSE
GLU_AUTO_LOAD_MATRIX	GL_TRUE | GL_FALSE
:	
:	

NURBS- und B-Spline-Kurven

Spezifikation und Darstellung

Spezifikation der NURBS-/B-Spline-Kurve

```
void gluNurbsCurve(GLUNurbs* nurb,      // NURBS-Objekt
                   GLint knotCount,     // Knotenanzahl
                   GLfloat *knots,      // Adresse des Knotenvektors
                   GLint stride,        // Anzahl Zwischenwerte
                   GLfloat *control,    // Adresse des Kontrollpunktfelds
                   GLint order,         // Ordnung der Kurve
                   GLenum type          // Anzahl der Koordinaten
                                        // der Vertices (Tabelle)
                  );
```

Tabelle: Mögliche Werte von type (Auswahl)

```
GL_MAP1_VERTEX_3,
GL_MAP1_VERTEX_4,
      :
GL_MAP1_COLOR_4,
```

Darstellung der NURBS-/B-Spline-Kurve

```
void gluBeginCurve(GLUNurbs*  nurb);
      gluNurbsCurve(.....);
void gluEndCurve(GLUNurbs*  nurb);
```

NURBS- und B-Spline-Kurven

Beispiel für eine B-Spline-Kurve

Beispiel (Prinzip):

```
type Point3D=record x,y,z: glFloat end;

var P:          array [0..15] of Point3D;
    Knoten:     array [0..19] of glFloat;
    Kurve:      PGLUNurbsobj;

// Erzeugen eines Zeigers auf eine B-Spline-Kurve
Kurve:=gluNewNurbsRenderer;

// Definition der Vertices mit den Koordinaten P[i].x, P[i].y und P[i].z
          :
// Definition des Knotenvektors für eine nichtuniforme B-Spline-Kurve
      for i:=0   to 3   do Knoten[i]:=0;
      for i:=4   to 15 do Knoten[i]:=i-3;
      for i:=16 to 19 do Knoten[i]:=13;

gluNurbsProperty(Kurve,GLU_SAMPLING_TOLERANCE,5);
gluBeginCurve(Kurve);
   gluNurbsCurve(Kurve,                    // NURBS-Kurve
                 20,                       // Knotenanzahl
                 @Knoten,                  // Knotenvektor
                 3,                        // Anzahl Zwischenwerte
                 @P,                       // Kontrollpunktfeld
                 4,                        // Ordnung der NURBS-Kurve
                 GL_MAP1_VERTEX_3);  // Vertex mit 3 Koordinaten
gluEndCurve(Kurve);
```

Die Transformationspipeline von OpenGL

Modell- oder Objektkoordinaten

Modell- und Ansichtstransformation
- Plazierung und Orientierung der Objekte im WC-System
- Einheitliche Darstellung für die Szene
- Plazieren von Kamera und Lichtquellen

Weltkoordinaten
(Blickpunkt- oder Augpunktkoordinaten)

Projektionstransformation
- Festlegung des Viewing- Volumens
- Festlegung der Projektionsart
- Standardclipping am Sichtvolumen

Clip-Koordinaten

Projektive Normierung
- Abbildung des Viewing-Volumens wird auf eine 2D-Ebene in Form eines normierten dreidimensionalen Raumes projiziert

Normierte Projektionskoordinaten
(normierte Gerätekoordinaten)

Viewport- Transformation
- erfolgt auf dem Bildschirm
- die Abb. erfolgt in einem Fenster (Window), dessen Größe manipuliert und in Viewports unterteilt werden kann

Gerätekoordinaten
(Bildschirm- oder Viewport-Koordinaten)

Modelltransformationen der OpenGL

Setzen des aktuellen Matrizenmodus:

void **glMatrixMode** (Glenum ***mode***)

Wert des Parameters ***mode***	Bedeutung	Vorbelegung
GL_MODELVIEW GL_PROJEKTION GL_TEXTURE	Modelltransformation Projektionstransformation Texturtransformation	identity Matrix = Einheitsmatrix

Bearbeiten des Matrizenstapels mit:

void **glPushMatrix** ( ) ; Dupliziert d. aktuelle Matrix u. speichert sie als neues oberstes Element.

void **glPopMatrix** ( ) ; Löscht oberstes Element. Darunter liegende ist jetzt aktuelle Matrix.

void **glLoadIdentity** ( ) ; Ersetzt aktuelle Matrix durch die Einheitsmatrix.

void **glLoadMatrix** {f d} (const TYPE * m) ;
Ersetzt die aktuelle Matrix durch die als Parameter übergebene Matrix.

Standardmatrizen:

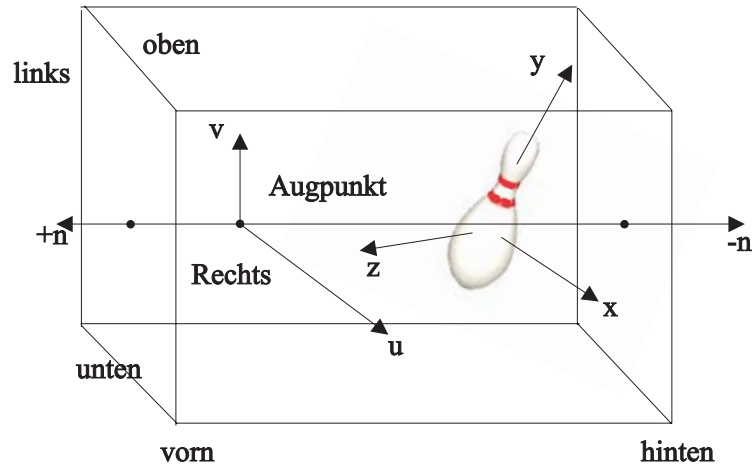
void **glRotate** {f d} (TYPE angle, TYPE x, TYPE y, TYPE z) ;

void **glTranslate** {f d} (TYPE x, TYPE y, TYPE z) ;

void **glScale** {f d} (TYPE x, TYPE y, TYPE z) ;

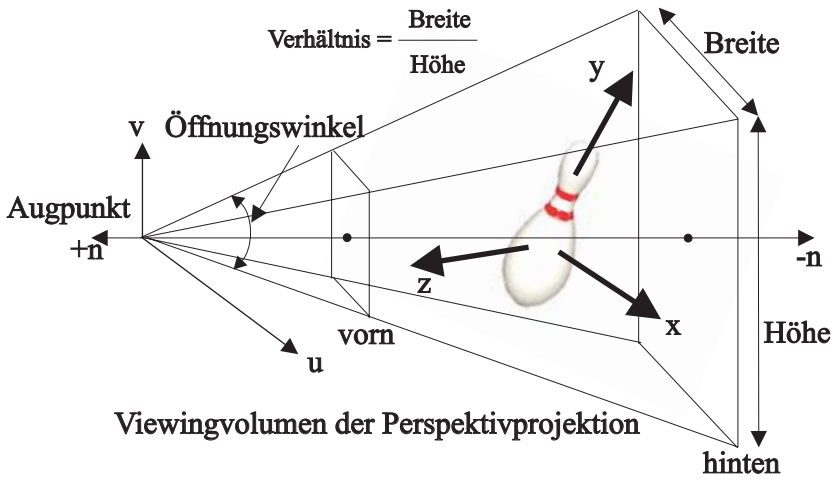
void **glMultMatrix** {f d} (const TYPE * m) ;

Viewingvolumen der Parallel- und Perspektivprojektion



Viewingvolumen der Parallelprojektion

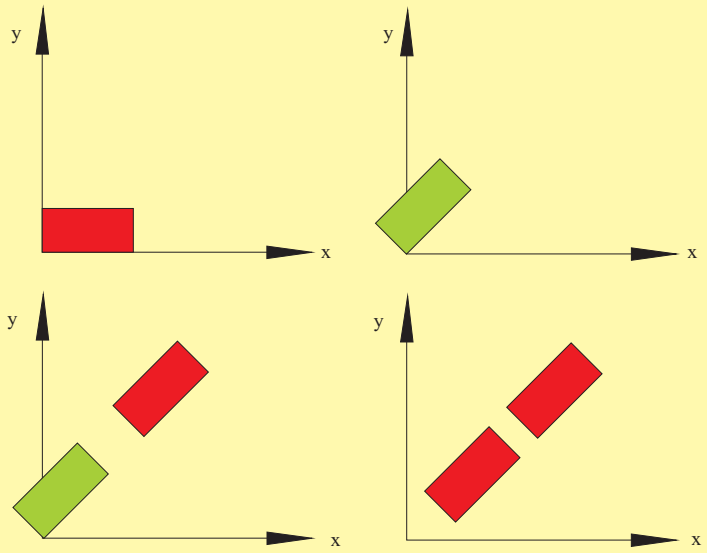
```
void gluLookAt(GLdouble eyex,...eyey,...eyez,        // Augpunkt des Betrachters
               ...centerx,...centery,...centerz,     // Blickpunkt
               ...upx,...upy,...upz)                 //view up vector
```



Viewingvolumen der Perspektivprojektion

```
Void gluPerspective(GLdouble angle,     // Öffnungswinkel der Kamera
                    GLdouble aspect,    // Verhältnis H:B des Viewing-Volumens
                    GLdouble znear,     // Entfernung Augpkt.- Vorderfront
                    GLdouble zfar)      // Entfernung Augpkt.- Rückfront
```

Hierarchische Transformationen




```
//Eingangsmatrix ohne Transformation retten
glPushMatrix
        //Rotation definieren
        glRotatef 45.0 0.0 0.0 1.0
        //Matrix mit Rotation retten
        glPushMatrix
            //Verschiebung definieren
            glTranslatef 3.0 0.0 0.0
            draw
        //Matrix nur mit Rotation restaurieren
        glPopMatrix
        //Matrix mit Rotation wieder auf den Stack
        glPushMatrix
             //zweite Verschiebung definieren
             glTranslatef 1.0 0.0 0.0
             draw
        glPopMatrix
   //Anfangsmatrix holen
   glPopMatrix

   ...
```

Technische Universität Dresden
Fakultät Informatik
Prof. Dr.-Ing.habil. Orlamünder

Lokales Beleuchtungsmodell der *OpenGl* für n Lichtquellen

$$C_{OpenGL}(\lambda)=C_e(\lambda)+C_a(\lambda)+\sum_{j=1}^{n} f(r_j)*s(a_j)*\{C_{La,j}(\lambda)+C_{Ld,j}(\lambda)+C_{Ls,j}(\lambda)\}$$

$C_e(\lambda)$ = Emission (Eigenleuchten) des Materials

$C_a(\lambda) = k_a(\lambda)*C_{amb}(\lambda)$ globales ambientes Licht

$$f(r_j) = \frac{1}{k_{quad} r_j^2 + k_{lin} r_j + k_{konst}}$$ Abschwächungsfaktor

$$s(a_j) = \begin{cases} 1 & \text{falls kein Spotlicht} \\ 0 & \text{falls Punkt außerhalb des Öffnungswinkels } \alpha \\ \max(\vec{L_j} * \vec{L_{s,j}}, 0) & \text{sonst} \quad (L_{s,j} = \text{Hauptstrahlrichtung}) \end{cases}$$

$$C_{La,j}(\lambda) = k_a(\lambda)*C_{Lamb}(\lambda)$$

$$C_{Ld,j}(\lambda) = k_d(\lambda)*C_{Ldiff}(\lambda)*\max(\vec{N}*\vec{L_j},0)$$

$$C_{Ls,j}(\lambda) = \begin{cases} K_s(\lambda)*C_{Lspek}(\lambda)*\max(\vec{N}*\vec{H_j},0) & \text{für } (\vec{N}*\vec{L_j})>0 \\ 0 & \text{sonst} \end{cases}$$

Lichtquellen der OpenGL

Definition der Lichtquellen

void **glLight**{i f} {v}(GLenum ***light***, Glenum ***pname***, TYPE ***param***)

Wert des Parameters ***light***	Bedeutung	Vorbelegung
GL_LIGHTi	Lichtquelle i (i=0...7)	GL_LIGHT0

Wert des Parameters ***pname***	Bedeutung	Vorbelegung
GL_AMBIENT	ambiente Lichtquellenfarbe C_{amb}	(0.,0.,0.,1.)
GL_DIFFUSE	diffuse Lichtquellenfarbe C_{diff}	für GL_LIGHT0: (1.,1.,1.,1.) (0.,0.,0.,1.) sonst
GL_SPECULAR	spiegelnde Lichtquellenfarbe C_{spec}	für GL_LIGHT0: (1.,1.,1.,1.) (0.,0.,0.,1.) sonst
GL_POSITION	Position bzw. Strahlricht.	(0.,0.,1.,0.)
GL_SPOT_DIRECTION	Spot- Strahlrichtung	(0.,0.,-1.)
GL_SPOT_CUTOFF	Öffnungswinkel	180.
GL_SPOT_EXPONENT	Konzentrationsexponent k	0.

GL_CONSTANT_ATTENUATION	konstante Abschwächung	k_c
GL_LINEAR_ATTENUATION	lineare “	k_l
GL_QUADRATIC_ATTENUATION	quadratische “	k_q

Materialparameter der OpenGL

Definition der Materialparameter

void **glMaterial**{i f} {v}(GLenum ***face***, Glenum ***pname***, TYPE ***param***)

Wert des Parameters ***face***	Bedeutung	Vorbelegung
GL_FRONT GL_BACK GL_FRONT_AND_BACK	Vorderseite Rückseite Vorder- und Rückseite	GL_FRONT

Wert des Parameters ***pname***	Bedeutung	Vorbelegung
GL_AMBIENT	ambient reflektierte Farbe k_a	(0.2,0.2,0.2,1.0)
GL_DIFFUSE	diffus reflektierte Farbe k_d	(0.8,0.8,0.8,1.0)
GL_SPECULAR	spiegelnd reflektierte Farbe k_s	(0.0,0.0,0.0,1.0)
GL_AMBIENT_AND_DIFFUSE	gemeinsame Definition k_a=k_d	
GL_EMISSION	Emissionsfarbe C_e der Fläche	(0.0,0.0,0.0,1.0)
GL_SHININESS	Spiegelexponent m(0...128)	0.0

Parameter ***param***

Vektorieller Aufruf: Adresse auf den Vektor mit den definierten Farben

Nichtvektorieller Aufruf: Wert des Spiegelexponenten m

Beleuchtungs- und Shading-Modi der OpenGL

Aktivieren/Deaktivieren der Beleuchtungsrechnung

void glEnable (GL_LIGHTING)
void glDisable (GL_LIGHTING) ←——— Vorbelegung

Aktivieren/Deaktivieren der Lichtquelle i

void glEnable (GL_LIGHTi)
void glDisable (GL_LIGHTi)

Vorbelegung: **glEnable**(GL_LIGHT0)

Definition des Beleuchtungsmodells

void **glLightModel**{if}{v}(GLenum ***pname***, TYPE ***param***)

Wert des Parameters ***pname***	Bedeutung	Vorbelegung
GL_LIGHT_MODEL_AMBIENT	globales ambientes Licht	(0.2,0.2,0.2,1.0)
GL_LIGHT_MODEL_TWO_SIDE	Beleuchtung beider Seiten	GL_FALSE (nur Vorderseite beleuchten)
GL_LIGHT_MODEL_LOCAL_VIEWER	lokaler Betrachtungspunkt	GL_FALSE (globaler Betrachtungspunkt)

Definition des Shading-Modells

void **glShadeModel**(GLenum ***mode***)

Wert des Parameters ***mode***	Bedeutung	Vorbelegung
GL_FLAT	Flat-Shading	
GL_SMOOTH	Gouraud-Shading	GL_SMOOTH

Nebeleffekte der OpenGL

Definition des Nebels

void **glFog**{i f} {v}(Glenum ***pname***, TYPE ***param***)

Wert des Parameters ***pname***	Bedeutung	Vorbelegung
GL_FOG_MODE	Gewichtsfunktion GL_LINEAR, GL_EXP, GL_EXP2	GL_EXP
GL_FOG_DENSITY	Nebeldichte f_d	1.0
GL_FOG_START	Tiefenanfangswert f_{min} des Nebelbereiches	0.0
GL_FOG_END	Tiefenendwert f_{max} d. Nebelbereiches	1.0
GL_FOG_COLOR	Nebelfarbe C_{nebel}	(0.,0.,0.,0.)

Aktivieren/Deaktivieren der Nebelberechnung

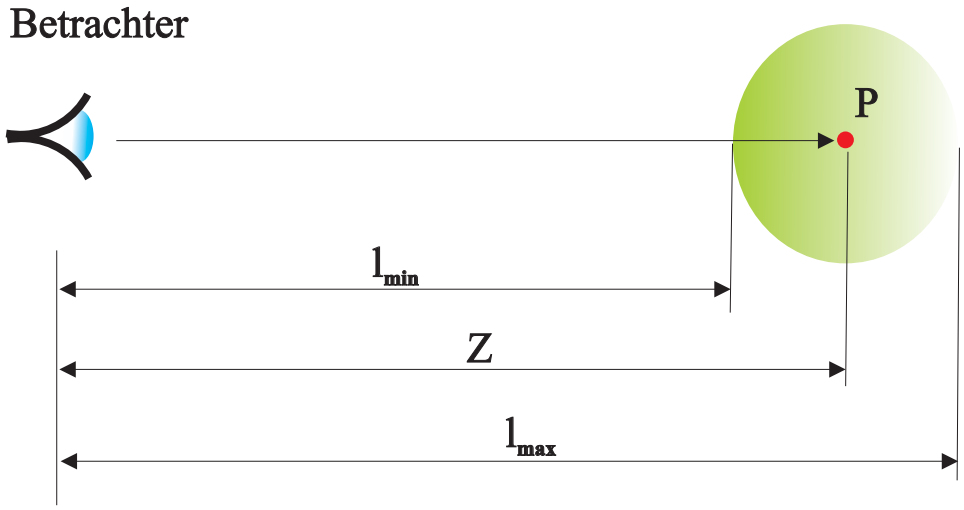
void **glEnable** (GL_FOG)
void **glDisable** (GL_FOG) ← Vorbelegung

Qualität der Nebelberechnung

void **glHint** (GL_FOG_HINT, Glenum ***mode***)

Wert des Parameters ***mode***	Bedeutung	Vorbelegung
GL_FASTEST	niedrigste Qualität	GL_FASTEST
GL_NICEST	höchste Qualität	
GL_DONT_CARE	systemabhängige Qualität	

Berücksichtigung des Nebeleffektes innerhalb der OpenGL



$$C = f * C_{Fragment} + (1-f) * C_{Nebel}$$

Lineare Gewichtung: $$f = \frac{l_{max} - z}{l_{max} - l_{min}}$$

Exponentielle Gewichtung: $$f = \exp(-f_d * z)$$

Quadratisch exponentielle Gewichtung: $$f = \exp(-(f_d * z)^2)$$

$C_{Fragment}$	Fragmentfarbe ohne Nebel
C_{Nebel}	Nebelfarbe
z	Entfernung des Objekts vom Betrachter
f	Gewichtsfunktionen des Nebels

Texturen (1)

Spezifikation einer 2D-Textur

Spezifikation einer zweidimensionalen Textur durch Angabe der Texturebene, der Breite und Höhe, dem Format und der Textur selbst.

```
void glTexImage2D (GLenum target,          // GL_TEXTURE_2D
                   GLint level,            // Textur-Ebene (MipMapping)
                   GLint components,       // Farbkomponenten der Textur
                   GLsizei width,          // Texturbreite, 2^n (oder 2^n+2)
                   GLsizei height,         // Texturhöhe, 2^m (oder 2^m+2)
                   GLint border,           // Rand, 0 (oder 1)
                   GLenum format,          // Format der Texturdaten
                   GLenum type,            // Typ der Texturdaten
                   const GLvoid *pixels    // Zeiger auf die Textur
                  );
```

Werte von *type* (Auswahl): GL_RGB, GL_RGBA

Werte von *format* (Auswahl): GL_UNSIGNED_BYTE

Beispiel:

```
glTexImage2D(GL_TEXTURE_2D, 0, 3, 256, 256, 0, GL_RGB,
             GL_UNSIGNED_BYTE, PBMP^);
```

Texturen (2)

Setzen der Umgebungsparameter für Texturen

Setzen der Umgebungsparameter für Texturen. Der Aufruf beeinflußt die Art und Weise der Berechnung der Fragmentfarbe auf der Basis der bereits berechneten Farbe und der Texturfarbe.

```
void glTexEnvf(GLenum target,   // GL_TEXTURE_ENV
               GLenum pname,    // GL_TEXTURE_ENV_MODE
               Glfloat param    // GL_MODULATE, GL_DECAL,
                                // GL_BLEND
              );
```

Anzahl Komponenten	Wirkung des Parameters *param*		
	GL_MODULATE	GL_DECAL	GL_BLEND
3 (RGB)	$C=C_TC_F$ $A=A_F$	$C=C_T$ $A=A_F$	undefiniert
4 (RGBA)	$C=C_TC_F$ $A=A_TA_F$	$C=(1-A_T)C_F+A_TC_T$ $A=A_F$	undefiniert

Beispiel:

```
glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE,
           GL_MODULATE);
```

Texturen (3)

Setzen der Parameter für Texturebenen und Filterung

Festlegung der Parameter für das Aufbringen von Texturen auf Flächen, u.a. der Abbildungsvorschrift für Texturen, die im Verhältnis zur Fläche zu groß bzw. zu klein sind (Mipmapping), und einer Festlegung, ob und wie Fortsetzungen von Texturen möglich sind.

```
void glTexParameterf(GLenum target,   // GL_TEXTURE_2D
                     GLenum pname,    // GL_TEXTURE_MIN_FILTER,
                                      // GL_TEXTURE_MAG_FILTER
                                      // GL_TEXTURE_WRAP_S
                                      // GL_TEXTURE_WRAP_T
                     GLfloat param    // siehe Tabelle
                    );
```

GL_TEXTURE_MIN_FILTER:	GL_NEAREST, GL_LINEAR, GL_NEAREST_MIPMAP_NEAREST, GL_LINEAR_MIPMAP_NEAREST, GL_NEAREST_MIPMAP_LINEAR, GL_LINEAR_MIPMAP_LINEAR
GL_TEXTURE_MAG_FILTER:	GL_NEAREST, GL_LINEAR,
GL_TEXTURE_WRAP_S:	GL_CLAMP, GL_REPEAT
, GL_TEXTURE_WRAP_T:	GL_CLAMP, GL_REPEAT

Beispiel:

```
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST);
```

Texturen (4)

Definition der Texturkoordinaten und Aktivieren der Texturierung

Definition der aktuellen Texturkoordinate, die mit den Eckpunkten von Polygonen assoziiert wird.

```
void glTexCoord2f(GLfloat s, GLfloat t);
void glTexCoord2fv(const GLfloat *v);
```

Beispiel:

```
glBegin (GL_POLYGON);
  glTexCoord2f(0,0); glVertex3f(-bx,-0.8*by,bz);
  glTexCoord2f(1,0); glVertex3f( bx,-0.8*by,dz*bz);
  glTexCoord2f(1,1); glVertex3f( bx, by,dz*bz);
  glTexCoord2f(0,1); glVertex3f(-bx, by,bz);
glEnd;
```

Aktivieren/Deaktivieren der Texturierung

```
void glEnable(GLenum cap);  //  GL_TEXTURE_2D
void glDisable(GLenum cap); //  GL_TEXTURE_2D
```

Beispiel:

```
glEnable(GL_TEXTURE_2D);
```

Texturen (5)

Generierung von Texturebenen (Mipmaps) für eine Textur

Erzeugen aller 2D-Mipmaps (Textur-Ebenen) einer gegebenen Texutur

```
GluBuild2DMipmaps(GLenum target,      // GL_TEXTURE_2D
                  GLint components,   // Farbkomponenten
                  GLint width,        // Texturbreite
                  GLint height,       // Texturhöhe
                  GLenum format,      // Format der Texturdaten
                  GLenum type,        // Typ der Texturdaten
                  void *data          // Zeiger auf die Textur
                 );
```

Werte von ***type*** (Auswahl): GL_RGB, GL_RGBA

Werte von ***format*** (Auswahl): GL_UNSIGNED_BYTE

Beispiel:

```
gluBuild2DMipmaps(GL_TEXTURE_2D,3,256,256,GL_RGB,
                  GL_UNSIGNED_BYTE,PBMP^);
```

Texturen (6)

Darstellung einer Textur

Beispiel für die prinzipielle Vorgehensweise zur Darstellung einer Textur.

```
GlPixelStorei (GL_UNPACK_ALIGNMENT ,1);

gluBuild2DMipmaps(GL_TEXTURE_2D,3,256,256,GL_RGB,GL_UNSIGNED_BYTE,PBMP^);

glTexImage2D(GL_TEXTURE_2D,0,3,256,256,0,GL_RGB,GL_UNSIGNED_BYTE, PBMP^);

glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_REPEAT);

glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,GL_NEAREST);

glTexEnvf (GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

glBegin (GL_POLYGON);
   glTexCoord2f(0,0);   glVertex3f(-bx,-0.8*by,bz);
   glTexCoord2f(1,0);   glVertex3f( bx,-0.8*by,dz*bz);
   glTexCoord2f(1,1);   glVertex3f( bx, by,dz*bz);
   glTexCoord2f(0,1);   glVertex3f(-bx, by,bz);
glEnd;

glEnable(GL_TEXTURE_2D);
```