Graphische Datenverarbeitung II

Graphische Programmierung

Prof. Dr. Elke Hergenröther

Allgemeines zu OpenGL

- OpenGL (Open Graphics Library) ist eine Programmbibliothek für 3D-Graphik
- unabhängig von einem Window-System
- dünne Schicht über der Graphikhardware
- OpenGL ist es Renderingsystem: komplexe Modelle müssen aus einfachen graphischen Primitiven aufgebaut werden.
- immediate mode:
  - sehr einfache Befehlte,
  - keine interne Datenstruktur der Szene
- Alle Namen fangen mit gl... an, Konstanten mit GL_

Allgemeines zu OpenGL

OpenGL ist eine State-Machine:

- Funktionen verändern den internen Zustand, bzw. verwenden ihn zur Darstellung.
- Das heißt einmal angeschaltet, bleibt der betreffende Zustand aktiv bis er wieder ausgeschaltet oder umgeschaltet wird.

OpenGL ist sehr „explizit“:

- Was nicht explizit aktiviert wurde, bleibt aus.
- Beispiel: Es nutzt nichts die Transparenz zu setzen, wenn man nicht explizit gesagt hat, dass Transparenzen berechnet werden sollen.

Higher-level Toolkit: GLUT

Initialisierung über:

```
glutInit(&argc, argv);
glutInitDisplayMode( GLUT_DEPTH |GLUT_RGB );
glutCreateWindow(„Name");
```

Zeichenfunktion wird automatisch von der glutMainLoop aufgerufen.

Welche Funktion als Zeichenfunktion genutzt werden soll, wird mit dieser Funktion festgelegt:

```
glutDisplayFunc(display);
```

Aufruf der Hauptschleife:

```
glutMainLoop();
```

1. OpenGL Programm

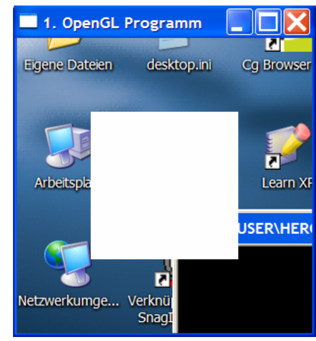
```
#include <GL/glut.h> //GLUT .h-Datei, lädt auch GL .h-Dateien

void display() //Zeichenfunktion
{
   glBegin( GL_POLYGON );
      glVertex3f( -0.5, -0.5, 0);
      glVertex3f(  0.5, -0.5, 0);
      glVertex3f(  0.5,  0.5, 0);
      glVertex3f( -0.5,  0.5, 0);
   glEnd();
   glFlush(); //Buffer leeren
}

int main(int argc, char **argv)
{
   glutInit(&argc, argv); // GLUT initialisieren
   glutInitDisplayMode( GLUT_RGB ); // Fenster Konfiguration
   glutCreateWindow("1. OpenGL Programm"); // Fenster Erzeugung
   glutDisplayFunc(display); // Zeichenfunktion bekannt machen
   glutMainLoop();
   return 0;
}
```

Welche Farbe hat das hier erzeugte Polygon?

Welche Hintergrundfarbe hat das hier erzeugte Fenster?

Ausgabe des 1. OpenGL Programms:

Keine Farbe und der Hintergrund ist nicht gesetzt:

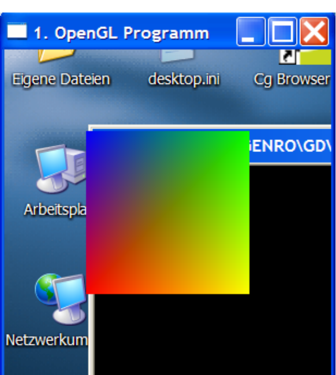


Erweiterung des 1. OpenGL Programm

```
#include <GL/glut.h>

void display()
{
   glBegin( GL_POLYGON );
      // State- Machine: Wenn nur der erste Farbwert angeben wird haben
      // alle folgenden Eckpunkte den gleichen Farbwert
      glColor4f( 1., 0., 0., 1.);
      glVertex3f( -0.5, -0.5, 0);
      glColor4f( 1., 1., 0., 1.); // Gelb
      glVertex3f(  0.5, -0.5, 0);
      glColor4f( 0., 1., 0., 1.);
      glVertex3f(  0.5,  0.5, 0);
      glColor4f( 0., 0., 1., 1.);
      glVertex3f( -0.5,  0.5, 0);
   glEnd();
   glFlush(); //Buffer leeren
}

int main(int argc, char **argv)
{
   ... // wie vorher
}
```


Ausgabe des 1. OpenGL Programm

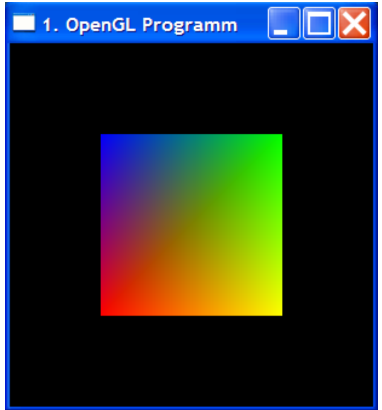
Der Hintergrund ist noch nicht gelöscht worden:



Erweiterung des 1. OpenGL Programm

```
#include <GL/glut.h>

void display()
{
   glClear(GL_COLOR_BUFFER_BIT ); //löschen des Hintergrunds

   glBegin( GL_POLYGON );
      // State- Machine: Wenn nur der erste Farbwert angeben wird haben
      // alle folgenden Eckpunkte den gleichen Farbwert
      glColor4f( 1., 0., 0., 1.);
      glVertex3f( -0.5, -0.5, 0);
      glColor4f( 1., 1., 0., 1.); // Gelb
      glVertex3f(  0.5, -0.5, 0);
      glColor4f( 0., 1., 0., 1.);
      glVertex3f(  0.5,  0.5, 0);
      glColor4f( 0., 0., 1., 1.);
      glVertex3f( -0.5,  0.5, 0);
   glEnd();
   glFlush(); //Buffer leeren
}

int main(int argc, char **argv) // wie vorher
...
```

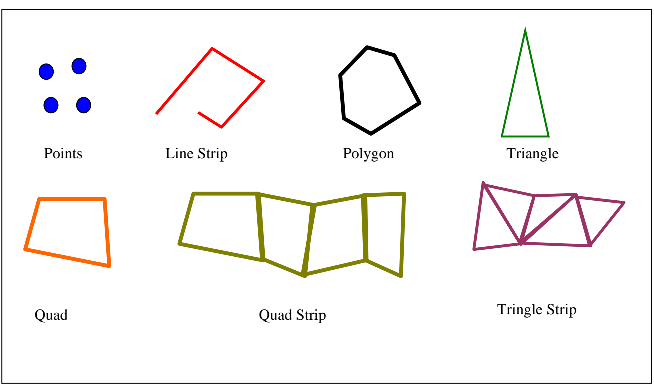

Ausgabe des 1. OpenGL Programm



Erweiterung des 1. OpenGL Programm

```
#include <GL/glut.h>

void display()
{
   glClear(GL_COLOR_BUFFER_BIT );

   glBegin( GL_POLYGON );
      glColor4f( 1., 0., 0., 1.);
      glVertex3f( -0.5, -0.5, 0);
      glColor4f( 1., 1., 0., 1.); // Gelb
      glVertex3f(  0.5, -0.5, 0);
      glColor4f( 0., 1., 0., 1.);
      glVertex3f(  0.5,  0.5, 0);
      glColor4f( 0., 0., 1., 1.);
      glVertex3f( -0.5,  0.5, 0);
   glEnd();
   glFlush(); //Buffer leeren
}

int main(int argc, char **argv) // wie vorher
...
```


OpenGl Primitive

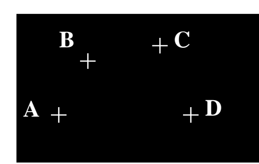


Definition der OpenGL Primitive

```
glBegin ( Typ des Primitivs )
   Liste von Eckpunkten (= Vertices)
glEnd ();
```

Beispiel:

```
glBegin ( GL_POINTS );
   glVertex3f ( xA, yA, zA );
   glVertex3f ( xB, yB, zB );
   glVertex3f ( xC, yC, zC );
   glVertex3f ( xD, yD, zD );
glEnd ();
```



Achtung:
Dies ist nicht die Bildschirm-ausgabe. Wegen der Kleinheit der Punkte
(1 Pixel) wird auf dem Schirm nur sehr wenig zu sehen sein

OpenGL Primitiv: Punkt

```
glPointSize (  15.0 );
glBegin ( GL_POINTS );
   glColor4f  ( 0.0f, 0.0f, 1.0f, 1.0f );
   glVertex3f ( -0.9f, 0.0f, 0.0f );
   glVertex3f ( -0.2f, 0.8f, 0.0f );
   glVertex3f ( +0.2f, 0.8f, 0.0f );
   glVertex3f ( +0.9f, 0.0f, 0.0f );
glEnd ( );
```

OpenGL ist eine State-Machine:

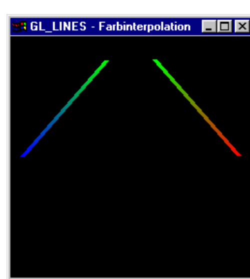
- Funktionen verändern den internen Zustand.
- Das heißt, einmal angeschaltet, bleibt der betreffende Zustand aktiv, bis er wieder ausgeschaltet oder umgeschaltet wird.
- **Beispiel:** `glColor4f`

OpenGL Primitiv: Linie

```
glLineWidth (  5.0 );
glBegin ( GL_LINES );
  glColor4f  ( 0.0f, 0.0f, 1.0f, 1.0f );
  glVertex3f ( -0.9f, 0.0f, 0.0f );
  glVertex3f ( -0.2f, 0.8f, 0.0f );
  glVertex3f ( +0.2f, 0.8f, 0.0f );
  glVertex3f ( +0.9f, 0.0f, 0.0f );
glEnd ( );
```


OpenGL Primitiv: Linie

```
glLineWidth (  5.0 );
glBegin ( GL_LINES );
   glColor4f  ( 0.0f, 0.0f, 1.0f, 1.0f );
   glVertex3f ( -0.9f, 0.0f, 0.0f );
   glColor4f  ( 0.0f, 1.0f, 0.0f, 1.0f );
   glVertex3f ( -0.2f, 0.8f, 0.0f );
   glVertex3f ( +0.2f, 0.8f, 0.0f );
   glColor4f  ( 1.0f, 0.0f, 0.0f, 1.0f );
   glVertex3f ( +0.9f, 0.0f, 0.0f );
glEnd ( );
```



OpenGL Primitiv: Linienzug

```
glLineWidth (  5.0 );
glBegin ( GL_LINE_STRIP );
   glColor4f  ( 0.0f, 0.0f, 1.0f, 1.0f );
   glVertex3f ( -0.9f, 0.0f, 0.0f );
   glColor4f  ( 0.0f, 1.0f, 0.0f, 1.0f );
   glVertex3f ( -0.2f, 0.8f, 0.0f );
   glVertex3f ( +0.2f, 0.8f, 0.0f );
   glColor4f  ( 1.0f, 0.0f, 0.0f, 1.0f );
   glVertex3f ( +0.9f, 0.0f, 0.0f );
glEnd ( );
```


OpenGL Primitiv: Geschlossener Linienzug

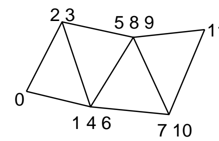
```
glLineWidth (  5.0 );
glBegin ( GL_LINE_LOOP );
   glColor4f  ( 0.0f, 0.0f, 1.0f, 1.0f );
   glVertex3f ( -0.9f, 0.0f, 0.0f );
   glColor4f  ( 0.0f, 1.0f, 0.0f, 1.0f );
   glVertex3f ( -0.2f, 0.8f, 0.0f );
   glVertex3f ( +0.2f, 0.8f, 0.0f );
   glColor4f  ( 1.0f, 0.0f, 0.0f, 1.0f );
   glVertex3f ( +0.9f, 0.0f, 0.0f );
glEnd ( );
```

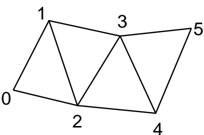

OpenGL Primitiv: Dreieck

```
glBegin(GL_TRIANGLES);
  glColor4f ( 0.0f, 0.0f, 1.0f, 1.0f );
  glVertex3f(-0.9f, 0.0f, 0.0f );
  glColor4f ( 0.0f, 1.0f, 0.0f, 1.0f );
  glVertex3f(-0.2f, 0.8f, 0.0f );
  glVertex3f(+0.2f, 0.8f, 0.0f );
  glColor4f ( 1.0f, 0.0f, 0.0f, 1.0f );
  glVertex3f(+0.9f, 0.0f, 0.0f );
  glVertex3f(+0.2f,-0.8f, 0.0f );
  glVertex3f(-0.2f,-0.8f, 0.0f );
glEnd();
```

OpenGL Primitiv: Dreiecksstreifen

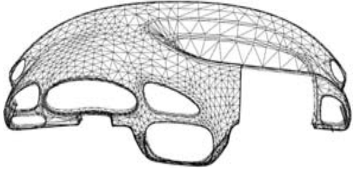
- Ziel ist es, möglichst wenig Elemente anzulegen.
- Eckpunkte können durch zusammenhängende Strips recycelt werden.



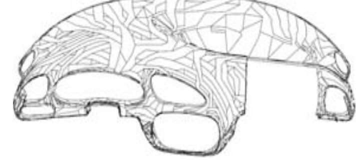


Anzahl der Punkte ohne Streifen- und mit Streifenbildung

OpenGL Primitiv: Dreiecksstreifen



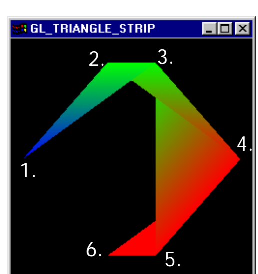
4320 Dreiecke
12960 Eckpunkte



905 Strips
6127 Eckpunkte

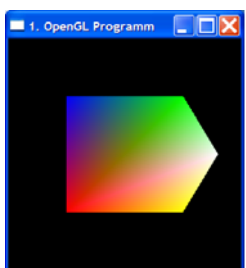
OpenGL Primitiv: Dreiecksstreifen (Auswertungsreihenfolge beachten!)

```
glBegin(GL_TRIANGLE_STRIP);
     glColor4f ( 0.0, 0.0, 0.0, 1.0f );
1.   glVertex3f(-0.9, 0.0, 0.0 );
     glColor4f ( 0.0, 1.0, 0.0, 1.0f );
2.   glVertex3f(-0.2, 0.8, 0.0 );
3.   glVertex3f( 0.2, 0.8, 0.0 );
     glColor4f ( 1.0, 0.0, 0.0, 1.0f );
4.   glVertex3f( 0.9, 0.0, 0.0 );
5.   glVertex3f( 0.2,-0.8, 0.0 );
6.   glVertex3f(-0.2,-0.8, 0.0 );
glEnd();
```



OpenGL Primitiv: Polygon

```
glBegin( GL_POLYGON );
  glColor4f( 1., 0., 0., 1.);
  glVertex2f( -0.5, -0.5);
  glColor4f( 1., 1., 0., 1.);
  glVertex2f(  0.5, -0.5);
  glColor4f( 1., 1., 1., 1.);
  glVertex2f(  0.8, 0.);
  glColor4f( 0., 1., 0., 1.);
  glVertex2f(  0.5,  0.5);
  glColor4f( 0., 0., 1., 1.);
  glVertex2f( -0.5,  0.5);
glEnd();
```



Transformationen in OpenGL

Translation: *Geometrie wird um den Vektor (x, y, z) verschoben!*

`glTranslatef( x, y, z);`

Rotationen: *Geometrie wird um den angegebenen Winkel (gegen den Uhrzeigersinn) um die Achse (x, y, z) rotiert.*

`glRotatef( Winkel, x, y, z);`

`x, y, z: Rotationsachse`

Skalierung:

`glScalef( x, y, z);`

Beispiele für Transformationen im 3D-Raum

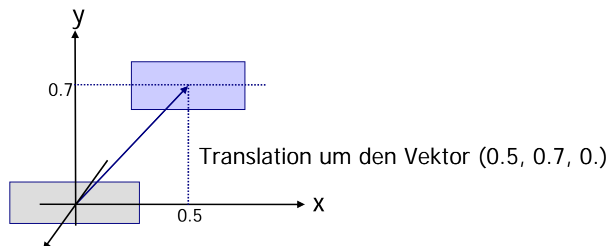
Y X Z

Rechthändiges Koordinatensystem

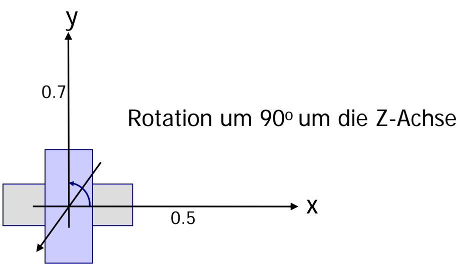
Translation
glTranslatef(x, y, z);

Rotation um die Z-Achse
glRotatef(Winkel, x, y, z);
Mit x, y, z: Rotationsachse

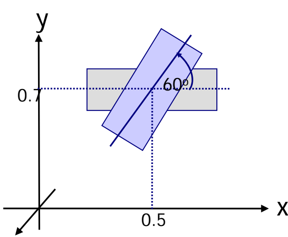
Beispiel für eine Translation



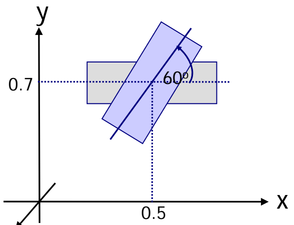
Beispiel für eine Rotation



Welche Transformationen sind notwendig um das Rechteck, wie vorgegeben, zu drehen?



Quadrat soll um 60° gedreht werden:



3. Rücktransformation in die Originalposition
2. Rotation um die Z-Achse
1. Transformation in den Ursprung

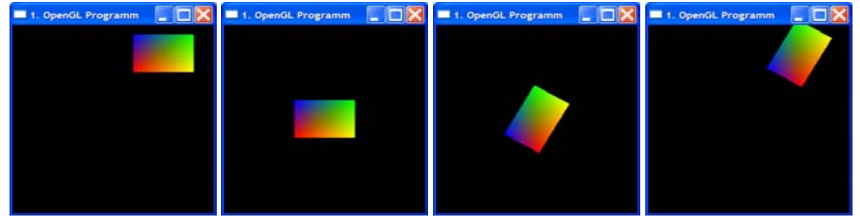
Reihenfolge in der die Transformationen angewendet werden ist wichtig!

Quadrat soll um 60° gedreht werden:

Transformationen werden in **umgekehrter** Reihenfolge angegeben:

Reihenfolge in der, die Transformationen auf das Quadrat angewendet werden:

```
glTranslatef( 0.5, 0.7, 0.);
glRotatef( 60., 0., 0., 1.);
glTranslatef( -0.5, -0.7, 0.);
Quadrat();
```



Quadrat soll um 60° gedreht werden:

```
glTranslatef( 0.5, 0.7, 0.);
glRotatef( 60., 0., 0., 1.);
glTranslatef( -0.5, -0.7, 0.);

glBegin( GL_POLYGON );
      glColor4f( 1., 0., 0., 1.);
      glVertex3f( 0.2, 0.5, 0);
      glVertex3f(  0.8, 0.5, 0);
      glVertex3f(  0.8,  0.9, 0);
      glVertex3f( 0.2,  0.9, 0);
glEnd();
```

Entspricht der Funktion: Quadrat()

Warum werden die Transformationen in umgekehrter Reihenfolge angegeben?

- Alle Transformationen werden durch Matrixmultiplikationen realisiert.
- Matrixmultiplikationen sind nicht kommutativ (d.h. sie sind nicht in ihrer Reihenfolge vertauschbar)
- Beispiel: a.) liefert nicht dasselbe Ergebnis, wie b.)

a.)

$$\begin{bmatrix}3\\2\\3\\1\end{bmatrix} = \begin{bmatrix}1&0&0&2\\0&1&0&3\\0&0&1&0\\0&0&0&1\end{bmatrix} \cdot \begin{bmatrix}0&1&0&0\\-1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \cdot \begin{bmatrix}1\\2\\3\\1\end{bmatrix}$$

b.)

$$\begin{bmatrix}5\\-3\\3\\1\end{bmatrix} = \begin{bmatrix}0&1&0&0\\-1&0&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} \cdot \begin{bmatrix}1&0&0&2\\0&1&0&3\\0&0&1&0\\0&0&0&1\end{bmatrix} \cdot \begin{bmatrix}1\\2\\3\\1\end{bmatrix}$$

Warum werden die Transformationen in umgekehrter Reihenfolge ausgewertet?

- Alle Transformationen werden durch Matrixmultiplikationen realisiert.
- Matrixmultiplikationen sind nicht kommutativ (d.h. sie sind nicht in ihrer Reihenfolge vertauschbar)
- Äquivalente Berechnungen von P':

$$\vec{P}' = M_2 \cdot (M_1 \cdot \vec{P}) \Leftrightarrow \vec{P}' = (M_2 \cdot M_1) \cdot \vec{P}$$

$$\vec{P}' = M_2 \cdot \vec{P}^{M_1} \Leftrightarrow \vec{P}' = M \cdot \vec{P}$$

Intuitive Vorgehensweise wenn man „von Hand“ transformiert:

P wird zunächst mit M_1 und dann wird der Ergebnisvektor mit M_2 multipliziert

Vorgehensweise von OpenGL:

Erstellen einer akkumulierten Matrix (M_2 wird mit M_1 multipliziert).

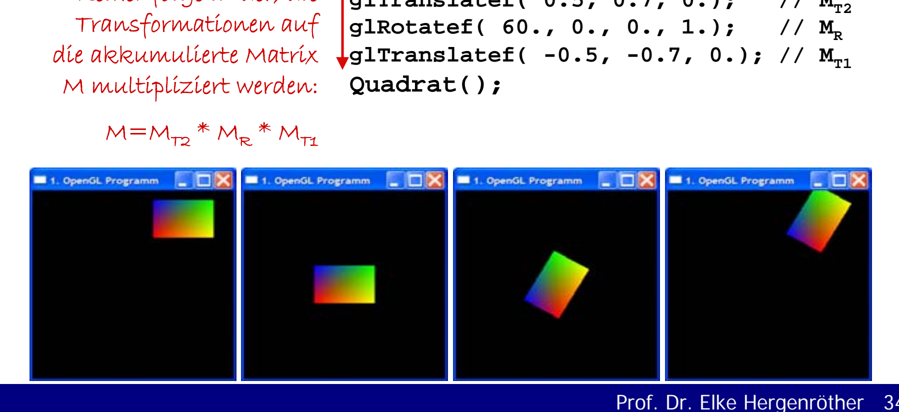
Quadrat soll um 60° gedreht werden:

Transformationen werden in **umgekehrter** Reihenfolge angegeben:

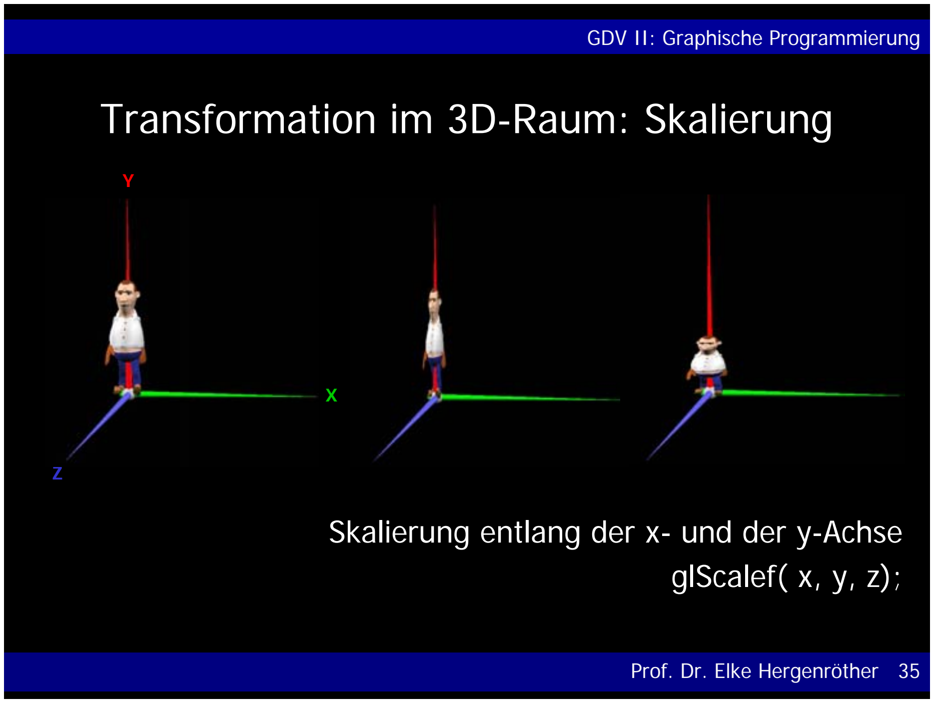
Reihenfolge in der, die Transformationen auf die akkumulierte Matrix M multipliziert werden:

```
glTranslatef( 0.5, 0.7, 0.);    // M_T2
glRotatef( 60., 0., 0., 1.);    // M_R
glTranslatef( -0.5, -0.7, 0.);  // M_T1
Quadrat();
```

$M = M_{T2} * M_R * M_{T1}$



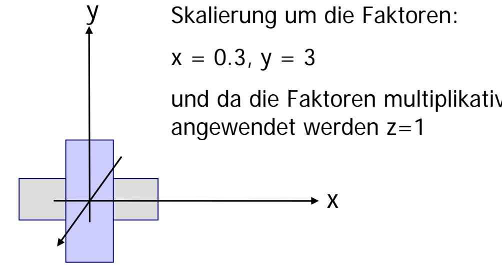
Transformation im 3D-Raum: Skalierung



Skalierung entlang der x- und der y-Achse

glScalef(x, y, z);

Beispiel für eine Skalierung



Skalierung um die Faktoren:

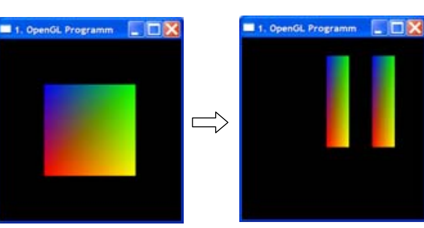
x = 0.3, y = 3

und da die Faktoren multiplikativ angewendet werden z=1

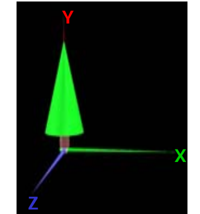
Beispiel für den Gebrauch von Transformationen

Aufgabe: Der Ausgangswürfel (links) soll so transformiert werden, dass das rechte Bild entsteht.

```
void Quadrat()
{
 glBegin( GL_POLYGON );
   glColor4f( 1., 0., 0., 1.);
   glVertex3f( -0.5, -0.5, 0);
   glColor4f( 1., 1., 0., 1.);
   glVertex3f(  0.5, -0.5, 0);
   glColor4f( 0., 1., 0., 1.);
   glVertex3f(  0.5,  0.5, 0);
   glColor4f( 0., 0., 1., 1.);
   glVertex3f( -0.5,  0.5, 0);
 glEnd();
}
```



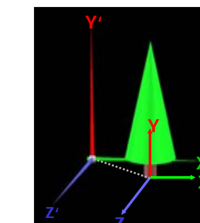
Beispiel für den Gebrauch von Transformationen

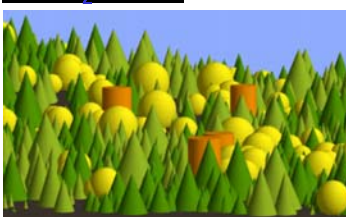


Modellierung des Objekts im lokalen Koordinatensystem

(Modellierungs-koordinatensystem oder körpereigenes Koordinatensystem)

Transformation des Objekts ins Weltkoordinaten-system





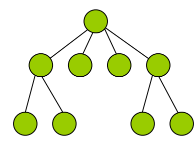
Beispiel für den Gebrauch von Transformationen

Aufbau eines komplexeren Modells

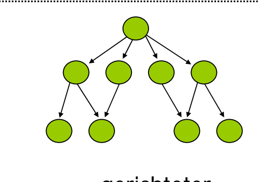
Um den Code übersichtlich zu halten benötigt man eine Strukturierungshilfe:
Den Szenengraph

Der Szenengraph

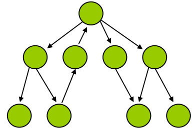
Unterschied Baum und Graph



Baum

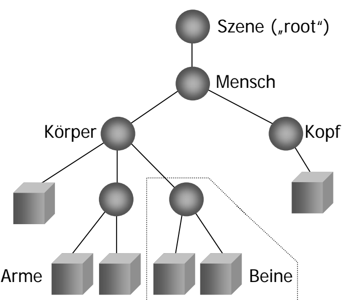


gerichteter zyklenfreier Graph



gerichteter nicht-zyklenfreier Graph

Szenengraph



Szenengraph zur Konstruktion einer Szene:

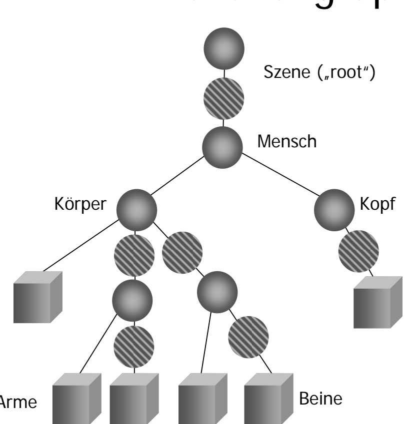
- Gruppierung von Geometrien zu Gruppen
- Gruppierung von Gruppen zu Gruppen
- Gruppierung von Gruppen zu einer Szene

Szenengraph

Szenengraph besteht aus mindestens 3 Knotentypen:

Gruppen
Geometrien (inkl. Materialeigenschaften)
Transformationen

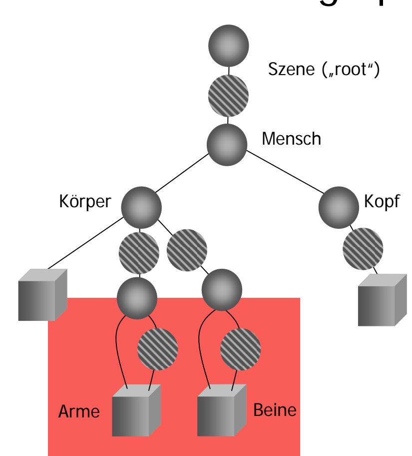
Szenengraphstruktur:



- gerichtet
- **azyklisch**
- heterogen
- Geometrie / darstellbare Primitive in den Blättern

Welchen Vorteil hat ein gerichteter azyklischer Szenengraphen gegenüber Szenengraphen, die als Bäume realisiert wurden?

Szenengraphstruktur:

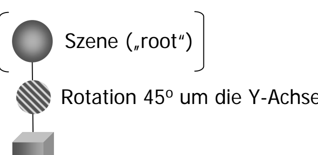


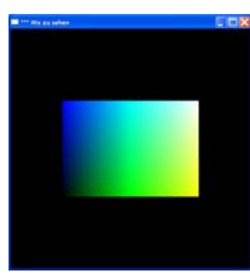
- gerichtet
- **azyklisch**
- heterogen
- Geometrie / darstellbare Primitive in den Blättern

Vorteil:
Geometrien werden nur einmal erzeugt und durch Transformationen mehrmals verwendet.

Transformation eines Würfels

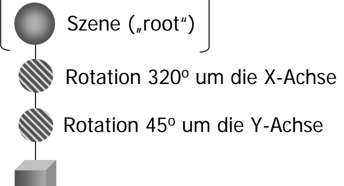
```
glRotatef(45., 0., 1., 0.);
Wuerfel(0.8);
```

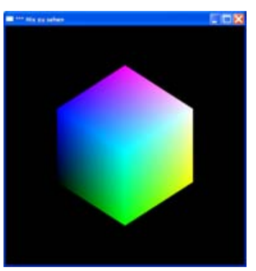




Transformation eines Würfels

```
glRotatef(320., 1., 0., 0.);
glRotatef(45., 0., 1., 0.);
Wuerfel(0.8);
```





Transformationen in OpenGL

Matrizenstack: Ermöglicht das temporäre Speichern von Matrizen

Akkumulierte Matrix wird auf den Stack gesichert:
`glPushMatrix();`

Akkumulierte Matrix wird wieder aktiviert mit: **`glPopMatrix();`**

Mit dem Aufruf von **`glPopMatrix()`** werden alle Änderungen der aktuellen Matrix, die zwischen dem Aufruf **`glPushMatrix()`** und **`glPopMatrix()`** gemacht worden sind gelöscht.

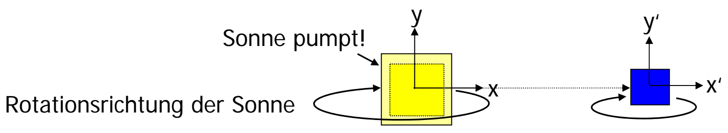
Transformationen in OpenGL

```
glTranslatef(0.3, 0.3, 0); //T1
glPushMatrix();
glTranslatef( -0.25, 0., 0.); //T2
glScalef(0.25, 1., 1.); //S1
Quadrat();
glPopMatrix();
glTranslatef( 0.25, 0., 0.); //T3
glScalef(0.25, 1., 1.); //S2
Quadrat();
```

Welche Transformationen werden jeweils auf das Quadrat angewendet?

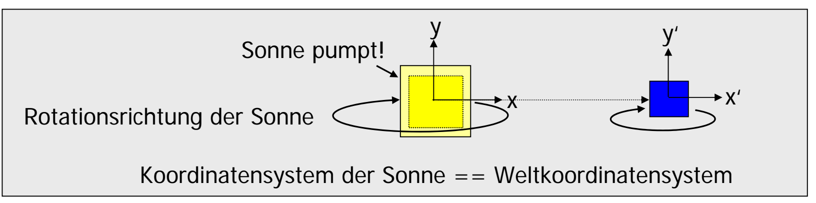
Aufgabe: Miniatursonnensystem

- Sonne rotiert um ihre Y-Achse.
- Während sie rotiert, pumpt sie sich auf und fällt wieder zusammen.
- Die Erde rotiert in einiger Entfernung mit gleicher Winkel-geschwindigkeit (d.h. überstrichener Winkel pro Zeiteinheit ist gleich) um die Y-Achse der Sonne.
- Zusätzlich rotiert die Erde um ihre eigene Y-Achse.
- Erde und Sonne sind eckig ;-)

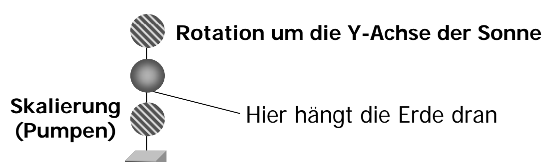


Koordinatensystem der Sonne == Weltkoordinatensystem

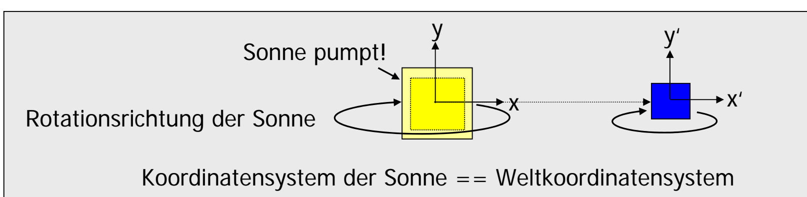
Szenengraph des Miniatursonnensystems



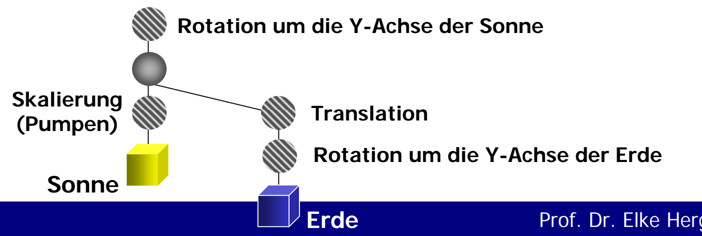
- Sonne und Erde rotieren mit gleicher Winkelgeschwindigkeit um die Y-Achse der Sonne
- Aber nur die Sonne pumpt sich auf und fällt zusammen



Szenengraph des Miniatursonnensystem



- Erde rotiert zusätzlich um ihre eigne Y-Achse



Prinzipielle Codierung des Szenengraphs

```
//Gruppenknoten zeigen an, wann die akkumulierte Matrix auf
//dem Stack gesichert werden muss.
glLoadIdentity();
glRotatef(fRotSonne, 0., 1., 0.);
glPushMatrix(); //Matrix wird auf den Stack gesichert
glScalef(fX, fY, fZ); // hier wird gepumpt
Wuerfel ( 0.5);
glPopMatrix(); //Matrix wird wieder aktiviert
glTranslatef( 0.5, 0., 0.);
glRotatef( fRotErde, 0., 1., 0.);
Wuerfel ( 0.2);
```

Rotation um die Y-Achse der Sonne

Skalierung (Pumpen)

Translation

Rotation um die Y-Achse der Erde

Sonne

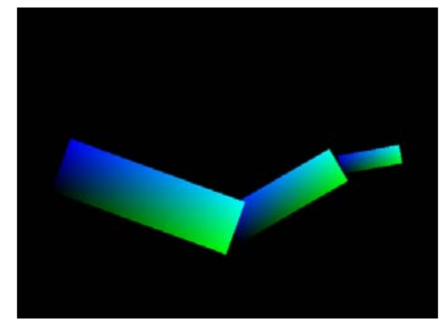
Erde

```
void Animate ()
{
   static float fRotSonne = 0., fRotErde=0;
   static float fX=1, fY=1, fZ=1;
   glLoadIdentity();
   glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   // Puffer loeschen
   glRotatef(fRotSonne+= 5., 0., 1., 0.);
   glPushMatrix(); //Matrix wird auf den Stack gesichert
   if( fX == 1){ fX=1.12; fY=1.12; fZ=1.12;}
   else {fX=1; fY=1; fZ=1;}
   glScalef(fX, fY, fZ);
   Wuerfel ( 0.5);
   glPopMatrix(); //Matrix wird wieder aktiviert
   glTranslatef( 0.5, 0., 0.);
   glRotatef( fRotErde+= 25., 0., 1., 0.);
   Wuerfel ( 0.2);
   glFlush();
   Sleep(200);
}
```

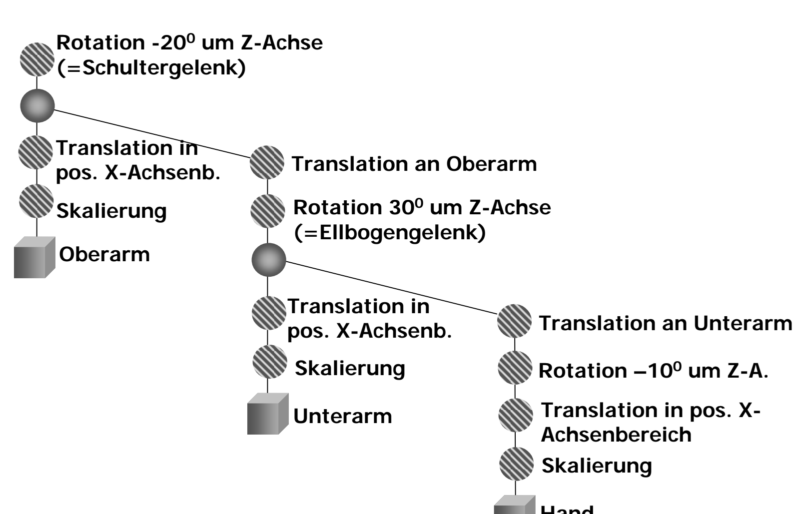
Miniatursonnensystem

Roboterarm

Ausgehend von unserem Würfel, soll dieser Roboterarm erstellt werden:

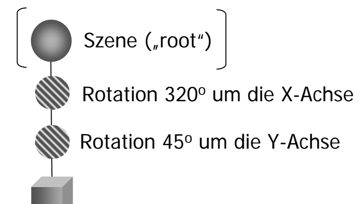


Szenengraph des Roboterarms

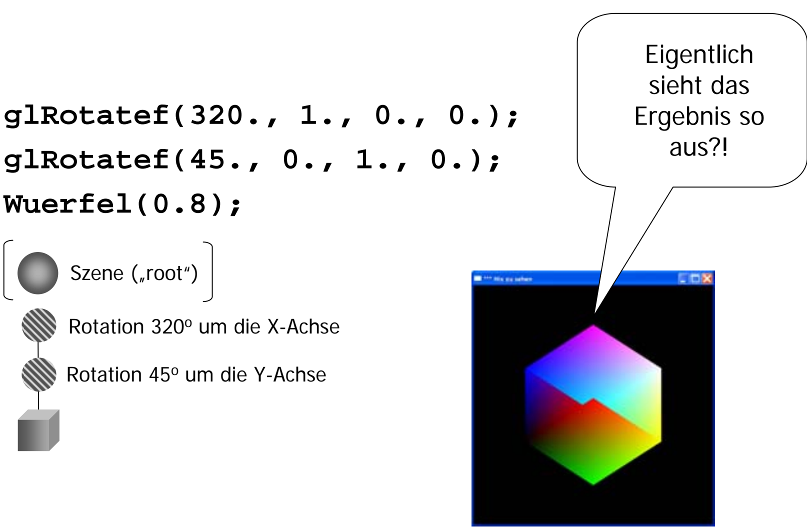


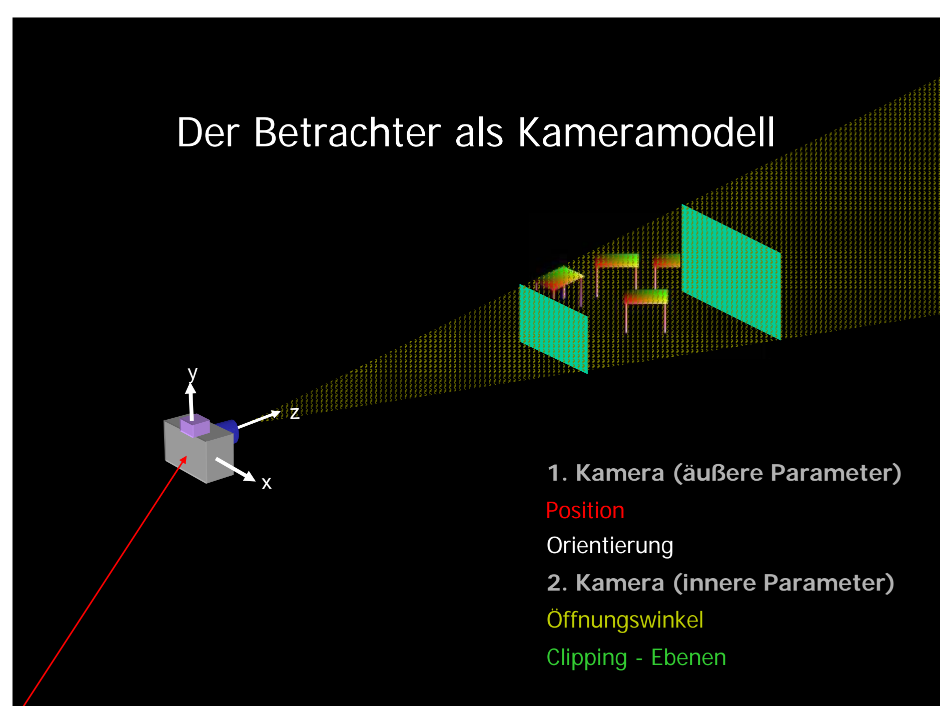
Erinnern Sie sich noch an dieses Beispiel?

```
glRotatef(320., 1., 0., 0.);
glRotatef(45., 0., 1., 0.);
Wuerfel(0.8);
```



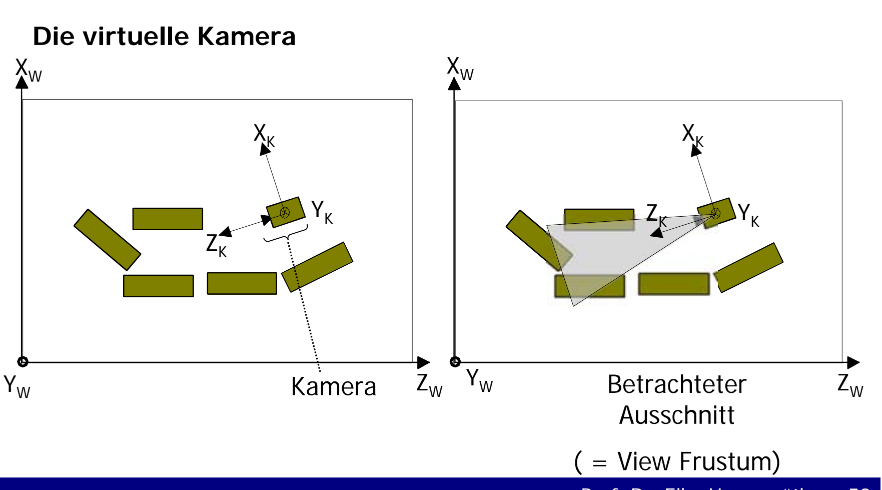
```
glRotatef(320., 1., 0., 0.);
glRotatef(45., 0., 1., 0.);
Wuerfel(0.8);
```



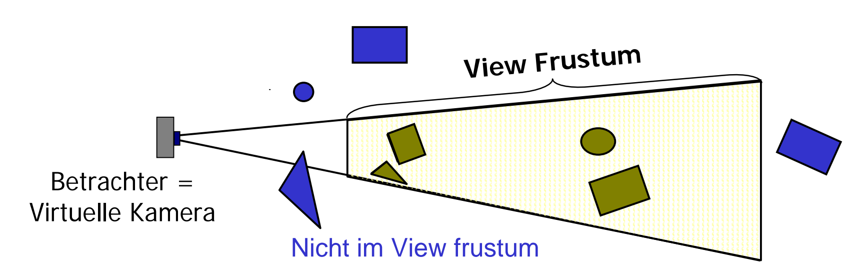


Der Betrachter als Kameramodell

Die virtuelle Kamera

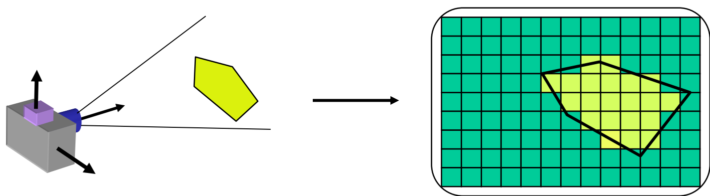


Der Betrachter als Kameramodell



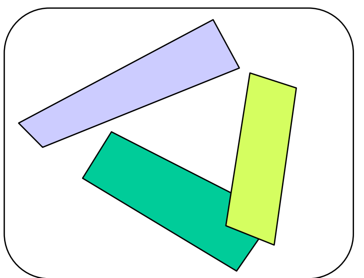
Rasterisierung des projizierten Bildes

- 3D Geometrie wird auf die Bildebene (2D) projiziert
- Das projizierte Bild wird zur Darstellung auf dem Bildschirm in Pixel zerlegt.



Verdeckungsproblem

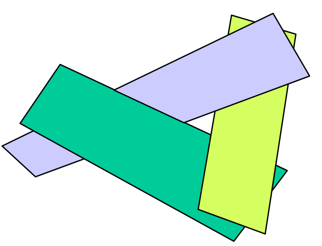
- **Problem:** In der Bildebene dürfen Bereiche, die von anderen Objekten verdeckt werden, nicht dargestellt werden.
- **Mögliche Lösung:** Hinten liegende Objekte zuerst darstellen.



Verdeckungsproblem

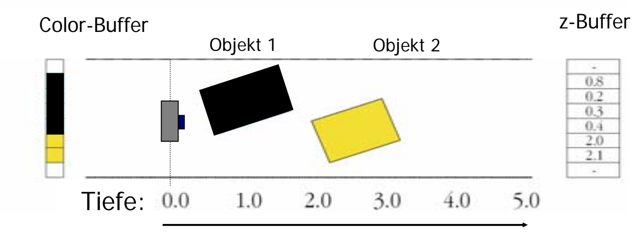
Von hinten nach vorne zeichnen führt zu keiner eindeutigen Lösung!

Lösung: Tiefen- oder Z-Buffer



Z-Buffer

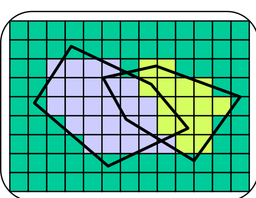
Pixel, die von der Kamera weiter entfernt liegen, als der bereits gespeicherte Z-Bufferwert, werden nicht dargestellt: **Verdeckung**

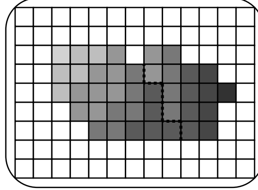


Z-Buffer

Lösung des Verdeckungsproblems mit Z-Buffer:

- Zu jedem Pixel wird die aktuelle Entfernung zur Kamera gespeichert
- Nur wenn die Tiefe des aktuell zu schreibenden Pixels kleiner als die gespeicherte Tiefe, wird er in die Buffer (Color- und Z-Buffer) eingetragen.





Tiefenwert pro Pixel: Je näher am Betrachter, desto dunkler

OpenGL: Z-Buffer

Initialisierung:

```
glutInitDisplayMode (GLUT_RGB | GLUT_DEPTH);
```

Einschalten:

```
glEnable(GL_DEPTH_TEST);
```

Löschen:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

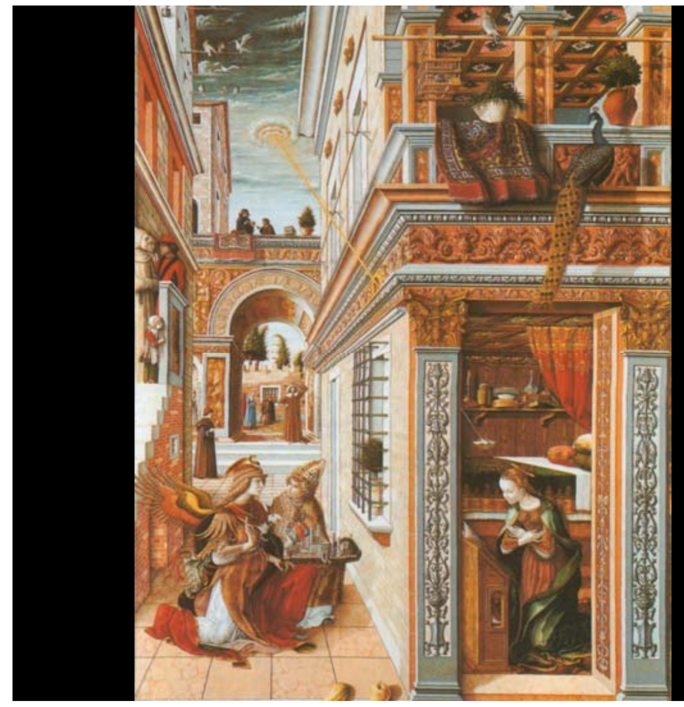
Und die Near- und Far-Plane müssen gesetzt werden:

```
z.Bsp.: glOrtho( GLdouble left, GLdouble right,
                 GLdouble bottom, GLdouble top,
                 GLdouble near, GLdouble far);
```


Allgemeines zu OpenGL

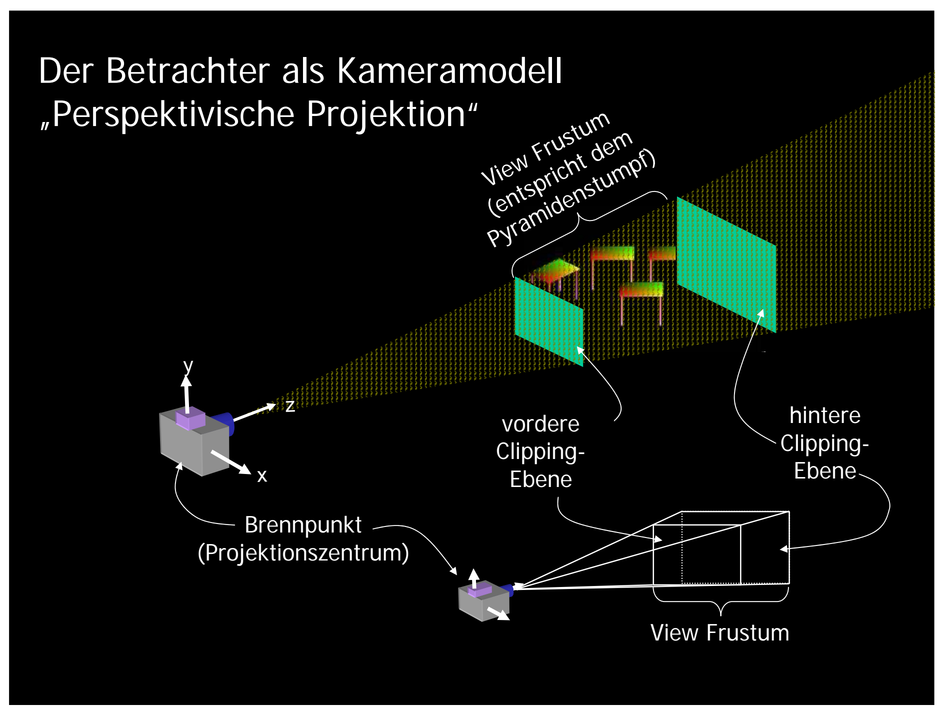
OpenGL ist sehr „explizit“:

- Was nicht explizit aktiviert wurde, bleibt aus.
- Beispiel: Der Z-Buffer wird nur genutzt, wenn er eingeschaltet wurde.

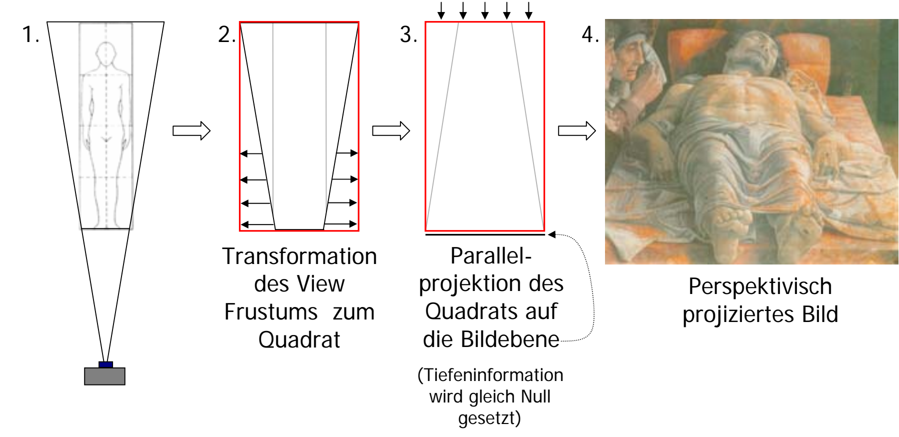


Perspektivische Projektion

- „Natürliche Darstellung“
- Am meisten verwendete Projektionsart
- Sehstrahlen treffen sich in einem Fluchtpunkt



„Berechnung" der perspektivische Projektion



Projektionsverfahren

Setzen des Volumenraums und Berechnung der Projektionsmatrix geschieht durch:

Perspektivische Projektion:

```
gluPerspective(  GLdouble fovy,
                 /*vertikaler Öffnungswinkel*/
                 GLdouble aspect,
                 /*Breiten zu Höhenverhältnis*/
                 Gldouble near,    /* minimaler und */
                 Gldouble far );   /* maximaler Abstand*/
                                   /* von der Kamera zum
                                      Frustum */
```


Projektionsverfahren

Setzen des Volumenraums und Berechnung der Projektionsmatrix geschieht durch:

Perspektivische Projektion:

- Prinzipiell gleichwertig zu **gluPerspective:**

```
glFrustum( GLdouble left,    GLdouble right,
           Gldouble bottom,  Gldouble top,
           Gldouble near,    Gldouble far );
```

Diese 4 Werte beziehen sich auf die near-Entfernung

- Während mit **gluPerspective** nur ein symmetrisches Viewfrustum erstellt werden kann, kann mit glFrustum ein assymetrisches Frustum erstellt werden, was sich dann auch für Stereoprojektionen eignet.

Projektionsverfahren

Setzen des Volumenraums und Berechnung der Projektionsmatrix geschieht durch:

Parallel Projektion (orthographische Projektion):

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


Projektionen

- Projektionsberechnungen werden auch mit Matrixberechnungen durchgeführt; allerdings liegen diese Matrizen auf einem anderen Matrix-Stack: GL_PROJECTION
- Umschalten kann man mittels:

```
glMatrixMode( { GL_MODELVIEW,
                GL_PROJECTION,
                GL_TEXTURE});
```

Eins von drei muss eingeschaltet sein.

- Unterschieden wird zwischen folgenden Projektionsarten:
  - parallel
  - perspektivisch

Der sich bewegende Betrachter

- Es gibt keinen expliziten Unterschied zwischen Kamera und Betrachter.
- Es bleibt dem Programmierer überlassen, ob der Betrachter sich um das Objekt bewegt oder ob sich das Objekt um den Betrachter bewegt. (Auswahl : Anwendungsbedingt)
- D.h. die Betrachtertransformation wird auch zu den Transformationen hinzugezählt.
- Zuständige Matrix heißt: GL_MODELVIEW (im 3D Raum)

Simulation des bewegten Betrachters in OpenGL

- OpenGL: Rechtshändiges Koordinatensystem
- Standardmäßig guckt die Kamera von Nullpunkt entlang der negativen Z-Achse.
- Um die Betrachterposition zu ändern, gibt es den folgdenden GLUT-Befehl:

```
gluLookAt( Gldouble eyex, Gldouble eyey, Gldouble eyez,
           /* Betrachterpos. */
           Gldouble centerx, Gldouble centery,
           Gldouble centerz,        /*Blickziel*/
           Gldouble upx, Gldouble upy, Gldouble upz );
           /*Oben-Betrachter*/
```

- gluLookAt setzt die äußeren Kameraparameter
- Transformationen des Betrachters werden in der Matrix GL_MODELVIEW gespeichert

Code mit zugehörigem Szenengraph:

```
void RenderScene(void)
{
   float extent = 0.8;
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho( -1, +1,   /* links, rechts: Aus Sicht der Kamera */
            -1., 1.,  /* unten, oben */
            0.0, 10); /*near, far */
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt( 0, 0, -2*extent,
              0.0, 0.0, 0.0,
              0.0, 1.0, 0.0);
   glRotatef(45, 0., 1., 0.);
   Wuerfel(extent);
   glFlush ();
}
```

Was passiert hier, wenn man die blau markierten Bereiche weglässt?

Orthographische Projektion

Betrachter transformieren (glLookAt)

Rotation 45^0 um die Y-Achse

Würfel

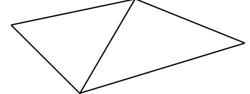
SwapBuffer

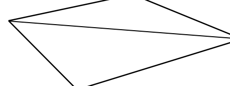
```
// 2 Buffer werden bereitgestellt
glutInitDisplayMode ( GLUT_DOUBLE| ...);
// Standardmäßig: Single-Modus eingeschaltet
// glutInitDisplayMode ( GLUT_SINGLE | ...);

// Jetzt wird der im Hintergrund gefüllte
// Buffer ausgegeben und der andere
// Buffer steht bereit, um für den Betrachter
// unsichtbar gefüllt zu werden.
glutSwapBuffers();
```


Geometrische Beschreibung der Objekte

Polygonale Darstellung:

- Polygone (mehr als 3 Kanten) müssen eben sein.
- Mehr 4 Punkte können bereits eine unebene Fläche bilden.

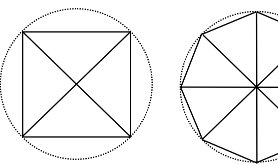




Geometrische Beschreibung der Objekte

Alle Formen der darzustellenden Objekte werden durch Dreieckgitter angenähert!

Problembeschreibung am Beispiel des Kreises:



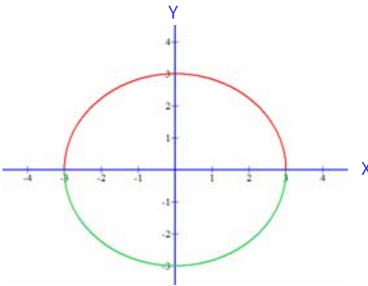
...

Form des Kreises wird angenähert durch 4, 8, ... usw. Dreieichen

Geometrische Beschreibung der Objekte

Idee: Geometrie des Objektes als Gleichung beschreiben

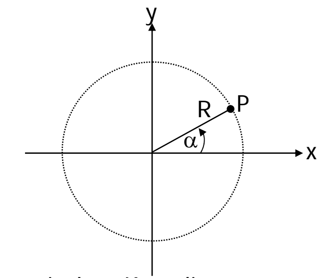
Formel: $y = \pm\sqrt{R^2 - x^2}$, R = Radius



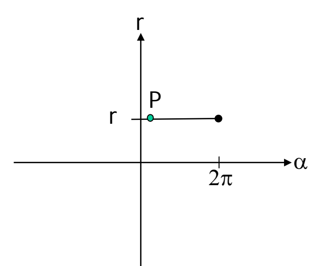
Y-Koordinate wird in Abhängigkeit der X-Koordinate berechnet

Geometrische Beschreibung der Objekte

Polarkoordinaten:



Kartesisches Koordinatensystem

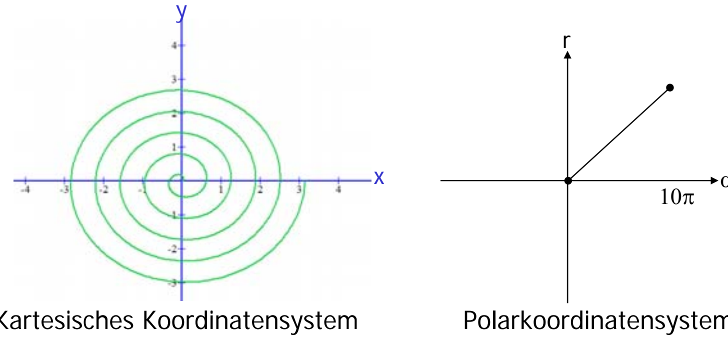


Polarkoordinatensystem

Beschreibung des Objektes erfolgt nicht in x- und y-Koordinaten sondern in r- (Radius) und α-Koordinaten (Winkel)

Geometrische Beschreibung der Objekte

Polarkoordinaten werden zur Beschreibung von Objekten verwendet, die durch die Drehung entlang einer Achse erzeugt werden können: Bsp. spiralförmige Objekte

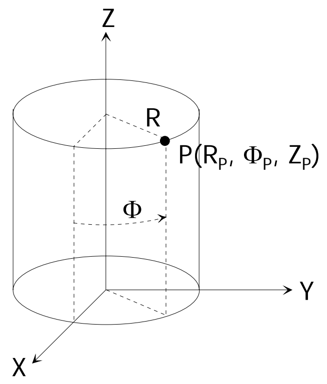


Kartesisches Koordinatensystem

Polarkoordinatensystem

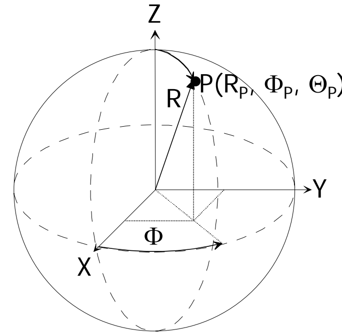
Geometrische Beschreibung der Objekte

Zylinderkoordinatensystem



Geometrische Beschreibung der Objekte

Kugelkoordinatensystem



Geometrische Beschreibung der Objekte

Kugelkoordinatensystem