3D Transformationen in OpenGL, Viewing und Licht

Christoph Bichlmeier

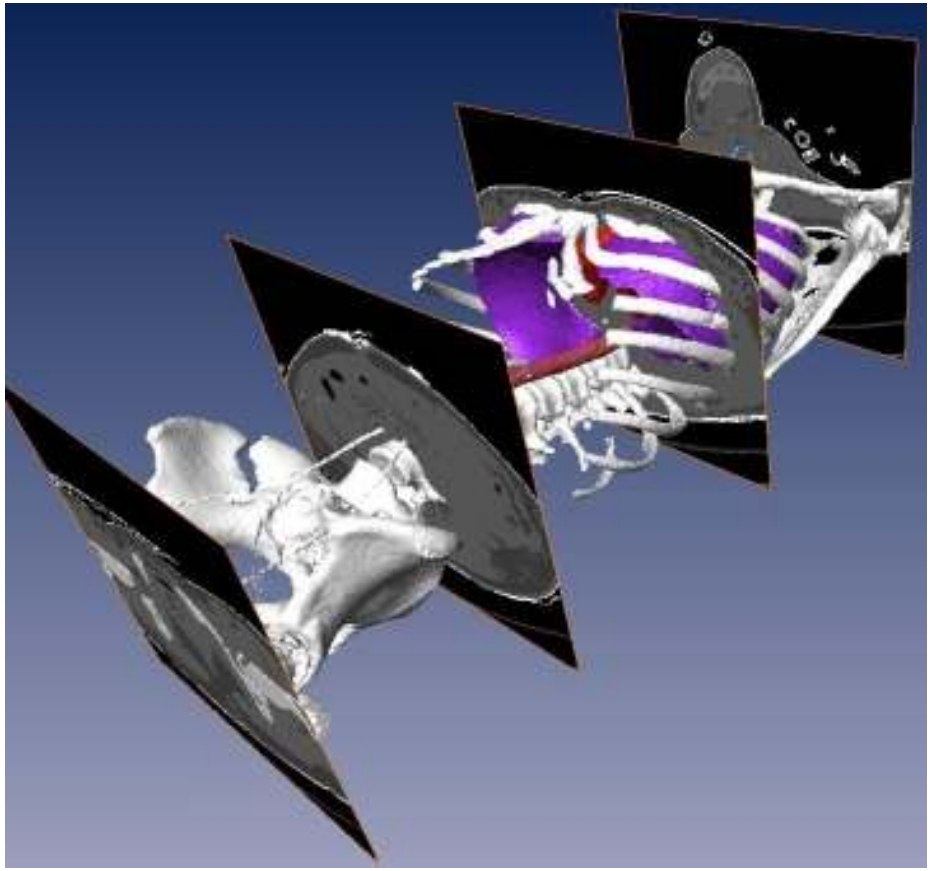
14. November 2006

Chair for Computer Aided Medical Procedures & Augmented Reality

Department of Computer Science | Technische Universität München

Motivation :: Es werde 3D …

Motivation

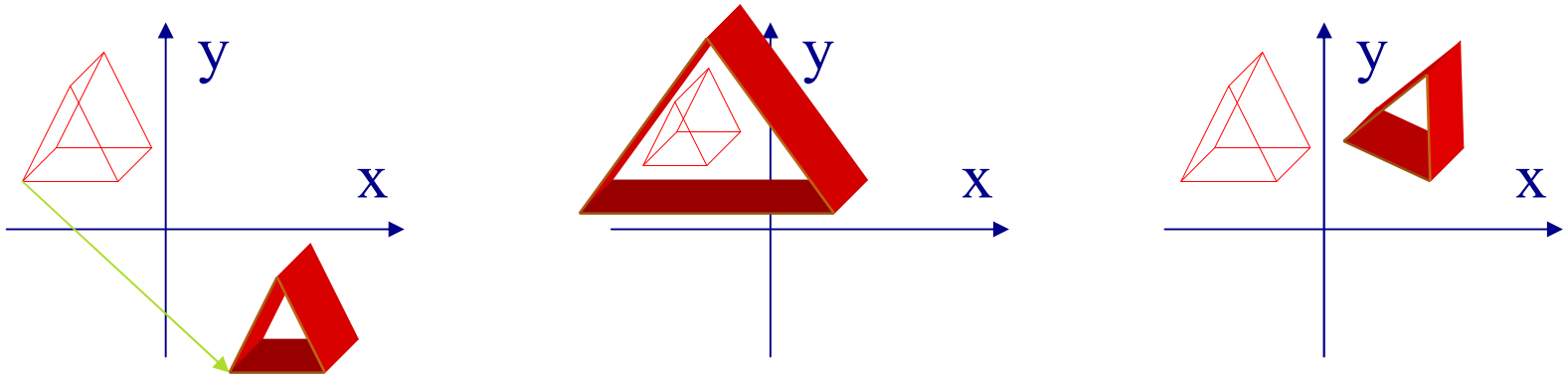


Outline

- Theorie 3D Transformation
- 3D Transformation in OpenGL
- Modelle generieren
- Viewing Pipeline
- Kameramodelle und Perspektive

3D Transformationen

Einfache Koordinaten-Transformationen



Translation Skalierung Rotation

3D Translation

3D Punkt

$$\vec{p} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Translation

$$\vec{p}' = \vec{p} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

3D Translation :: Homogene Koordinaten

Translationsmatrtix

$$\begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix}$$

Translation

$$\vec{p}' = \begin{bmatrix} I & T \\ 0 & 1 \end{bmatrix} \vec{p}$$

3D Rotation :: Homogene Koordinaten

Rotationsmatrix

$$\begin{bmatrix} & & & 0 \\ & R & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} R & \vec{0} \\ \vec{0} & 1 \end{bmatrix}$$

Rotation

$$\vec{p} \rightarrow \begin{bmatrix} R & \vec{0} \\ \vec{0} & 1 \end{bmatrix} \vec{p}$$

Euler-Winkel für die Rotation $R=R_zR_yR_x$

Rotation ψ um die z-Achse:

$$R_x = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotation θ um die y-Achse:

$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Rotation ϕ um die x-Achse:

$$R_z = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

3D Euklidische Transformation

Transformation

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Skalierung

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Transformationen in OpenGL

- Translation

```
glTranslatef(7.0f,0.0f,0.0f);          // 7 Einheiten nach rechts verschieben
                                       // (in positiver x-Richtung)
```

- Rotation

```
glRotatef(35.0f,0.0f,0.0f,1.0f);       // 35 Grad um die z-Achse rotieren
                                       // (Gegen den Uhrzeigersinn)
```

- Skalierung

```
glScalef(0.5f,0.5f,0.5f);              // in x-, y- und z-Richtung
                                       // um Faktor 0.5 skalieren
```

- Transformation in homogenen Koordinaten

```
glMultMatrixf(pointerToMatrix);        // Multipliziert die aktuelle 4x4
                                       // Matrix mit der 4x4 Matrix, auf die
                                       // von pointerToMatrix gezeigt wird
```

- Vorsicht: Bei Matrixmultiplikationen für Transformationen
  - Muss man sich im GL_MODELVIEW mode befinden!
  - Darf man oft ein glLoadIdentity() nicht vergessen!
  - Darf man nicht vergessen, dass
    - OpenGL die neue Matrix von rechts multipliziert
    - OpenGL seine Matrizen im Speicher spaltenweise ablegt: $\begin{pmatrix} a_0 & a_4 & a_8 & a_{12} \\ a_1 & a_5 & a_9 & a_{13} \\ a_2 & a_6 & a_{10} & a_{14} \\ a_3 & a_7 & a_{11} & a_{15} \end{pmatrix}$

Matrix Mode :: MODELVIEW_MATRIX

- `glLoadIdentity();`
  setzt die Modelview-Matrix auf die Identität
- `glLoadMatrix*(pointerToMatrix);`
  setzt den Wert der Modelview-Matrix auf die `pointerToMatrix`
- `glMultMatrix*(pointerToMatrix);`
  multipliziert den Wert der Modelview-Matrix mit `pointerToMatrix`
- `GLdouble model[16];`
  `glGetDoublev(GL_MODELVIEW_MATRIX,model);`
  liest den Inhalt der Modelview-Matrix in die `model` Variable

* ist ein Platzhalter fuer den Datentyp. Exemplarische Werte sind (i) 32-bit integer, (f) 32-bit floating-point or (d) 64-bit floating-point double

Transformationen mit Matrixmultiplikationen

```
TransformationMatrix myMatrix;
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
GLdouble multMat[16]
multMat = myMatrix.getTransposed().getArray();
glMultMatrixd(multMat);
```

Hinweis:

- Matrizen in openGL sind die Transponierten der obern hergeleiteten Theorie (Spaltenmatrizen).

Grundobjekte

- Außer mit `glBegin(…)` und `glEnd()` können mit GLUT einfach einige Grundobjekte gezeichnet werden
- Kugel `glutSolidSphere(GLdouble radius, GLint slices, GLint stacks)`
- Würfel `glutSolidCube(GLdouble size)`
- Kegel `glutSolidCone(GLdouble base, GLdouble height, GLint slices, GLint stacks)`
- Teekessel `glutSolidTeapot(GLdouble size)`
- Alle GLUT Objekte können auch als Drahtmodell gezeichnet werden z.B. `glutWireTeapot(GLdouble size)`

Modell generieren :: einfaches Beispiel

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glutSolidCube(2.0f);
glTranslatef(0.0f, 3.0f, 0.0f);
glRotatef(45.0f, 0.0f, 0.0f, 1.0f);
glutSolidCube(1.0f);
```

OpenGL ist ein Zustandsautomat

- D.h. wenn man Sachen ändert, bleiben sie solange bestehen, bis man sie wieder ändert
- Man kann einen Zustand in den Keller [Stack] „retten“, um ihn später wiederherzustellen
  - `glPushAttrib(GL_EIGENSCHAFT)` rettet den Wert von `GL_EIGENSCHAFT` in den Keller [Stack]
  - `glPopAttrib(GL_EIGENSCHAFT)` stellt den Wert wieder her
  - `glPushAttrib()` rettet alle Attribute / Eigenschaften
- Das gleiche gilt auch für die Matrizen
  - `glPushMatrix()` rettet die aktuelle Matrix
  - `glPopMatrix()` stellt die gerettete Matrix wieder her

Beispiel: Ein Adventskranz mit Kegelkerzen

```
void adventskranz(){
glPushMatrix();
 glTranslate(-5.0,0,5.0); // Verschiebung nach vorn rechts
 glutSolidCone(5.0, 8.0, 10, 10);//Breite 5, Höhe 8
 glTranslate(10.0,0.0,0.0); // Verschiebung nach rechts
 glutSolidCone(5.0, 8.0, 10, 10);//Breite 5, Höhe 8
 glTranslate(0.0,0.0,-10.0); // Verschiebung von Kamera weg
 glutSolidCone(5.0, 8.0, 10, 10);//Breite 5, Höhe 8
 glTranslate(-10.0,0.0,0.0); // Verschiebung nach links
 glutSolidCone(5.0, 8.0, 10, 10);//Breite 5, Höhe 8
glPopMatrix();
}
```

```
adventskranz();
glTranslate(-20.0,0.0,0.0)
glutSolidTeapot(10.0);
```

Tiefenpuffer

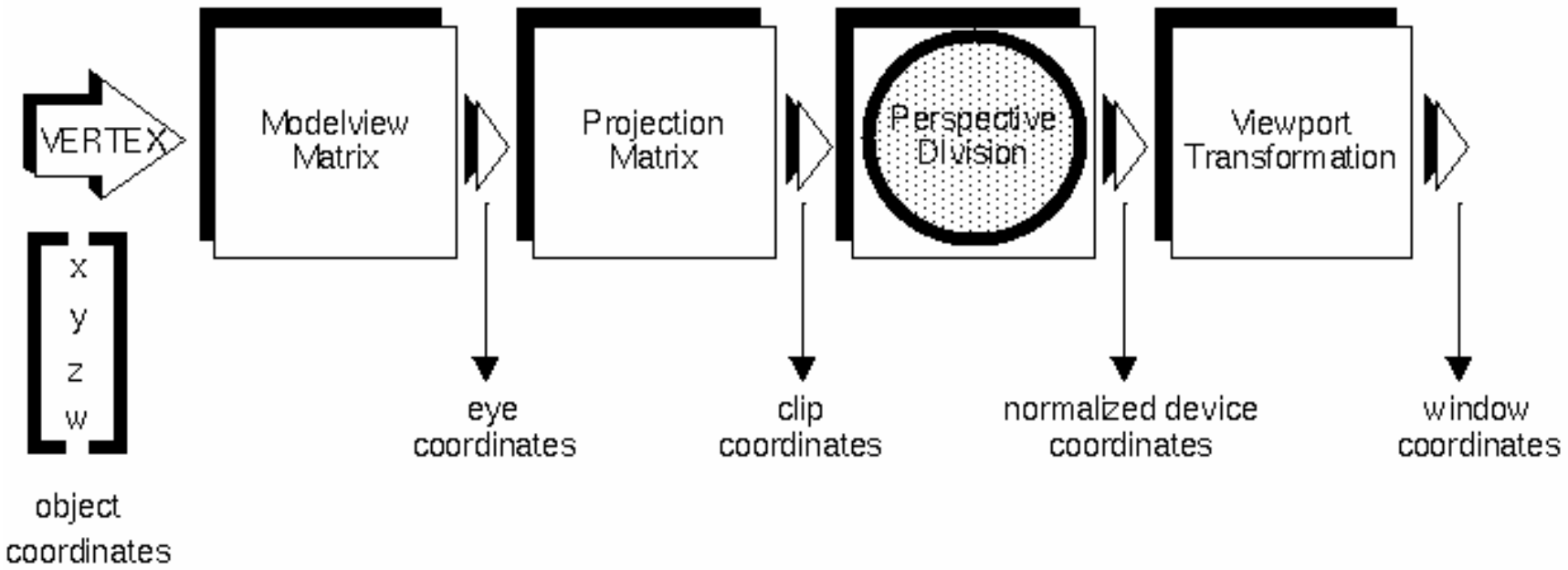
```
glColor3d(0.0,0.0,1.0); // Blau
glutSolidTeapot(10.0);
glTranslate(0.0,0.0,20.0); // 20 nach hinten
glColor3d(1.0,0.0,0.0); // Rot
glutSolidTeapot(10.0);
```

- Ohne Tiefenpuffer werden die Objekte einfach übereinander gelegt.

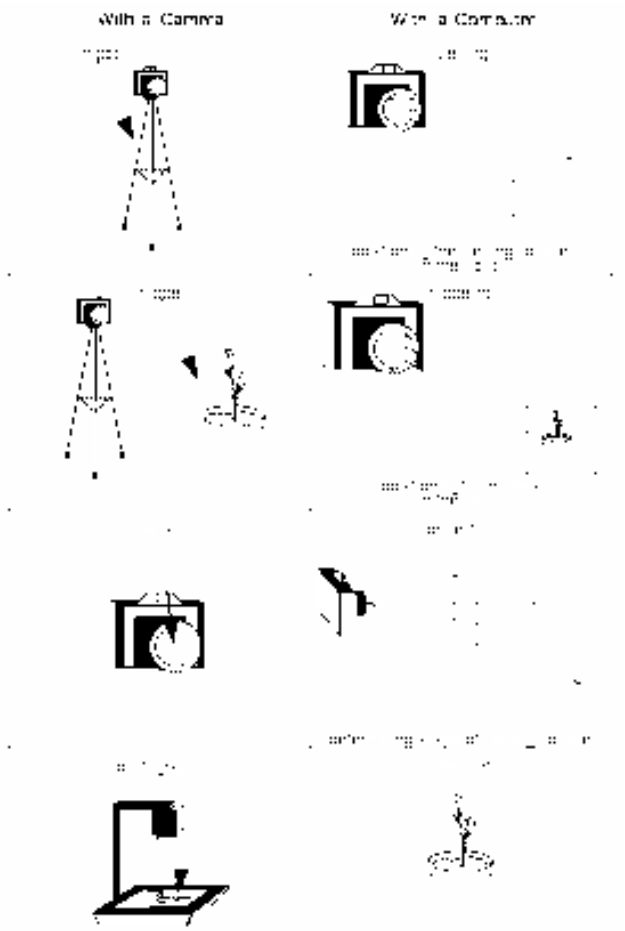
  `glEnable(GL_DEPTH_TEST)`

- Die Graphikkarten besitzen zusätzlich zu dem Puffer für Farbwerte für jeden Pixel auch Puffer für Tiefenwert. Wird dieser aktiviert, wird ein Pixel nur neu gezeichnet, wenn der Pixel näher zur Kamera ist!

Rendering Pipline



Viewing



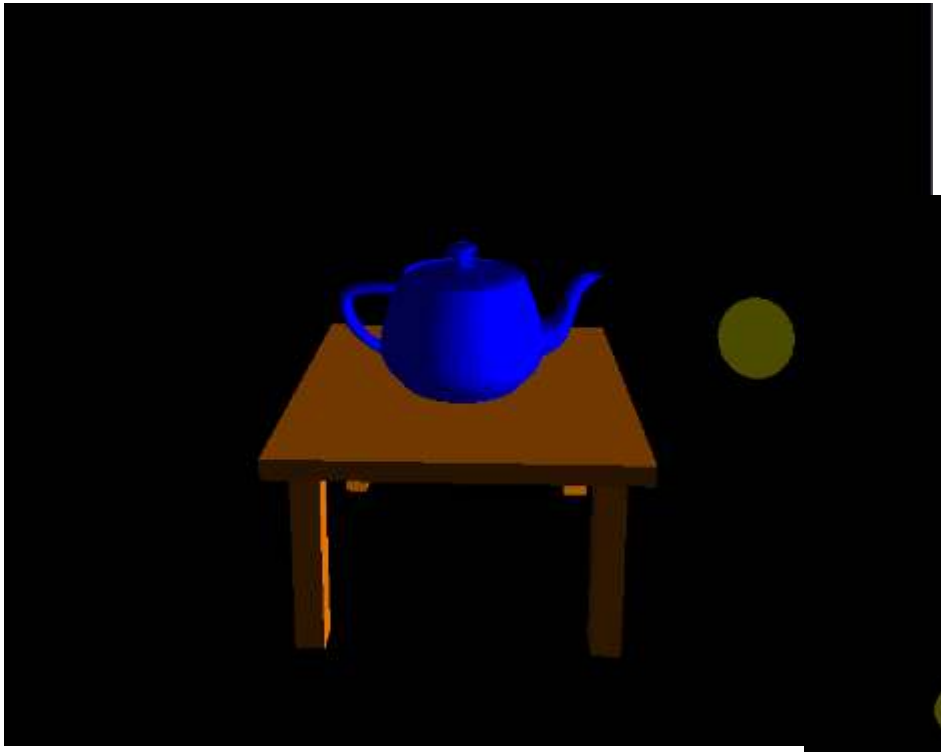
- Kameraposition mit `gluLookAt(eyex, eyey, eyez, centerx, centery, centerz, upx, upy, upz)`
- Modell generieren `glBegin()… glEnd()`
- Projektionsparameter setzen

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45.0,1.0,1.0, 1000.0);
```

- Bildauschnitt setzen

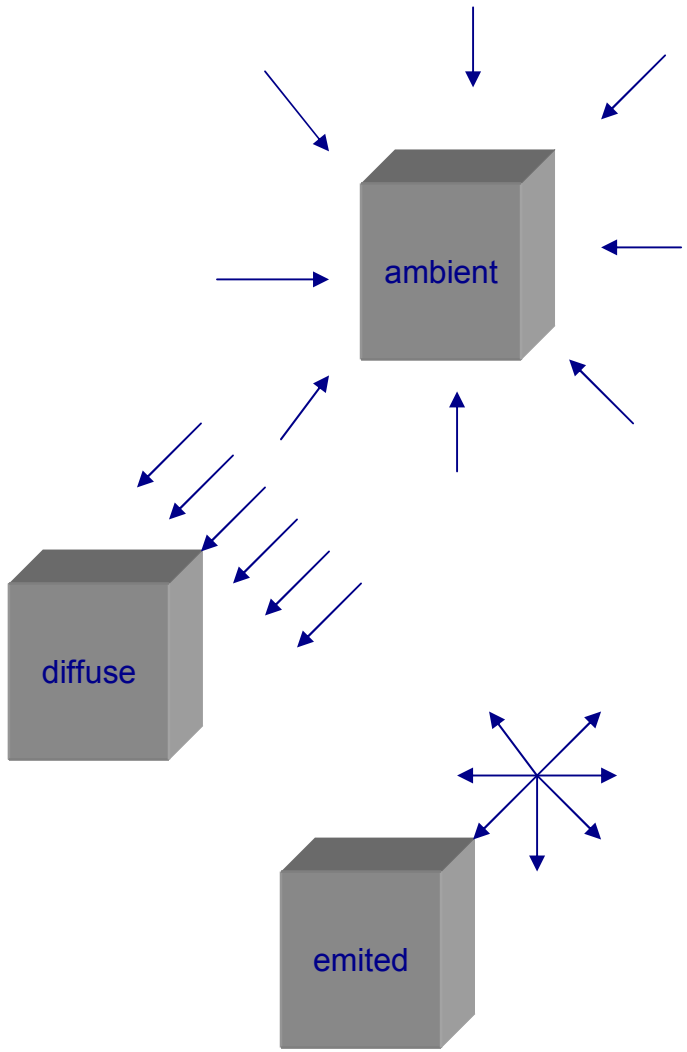
```
glViewport(0,0,width,height);
```

Es werde Licht …



Verschieden Arten von Licht

- „Ambient Light”
  - Setzt die Grundhelligkeit einer Szene fest
  - Abhängig vom Material
- „ Diffuse Light”
  - Paralleles, gerichtetes Licht
  - Quelle im unendlichen
- „Emitted Light“
  - Punktquelle
- „Specular Light“
  - Gerichtetes Licht mir speziellen Reflexionseigenschaften



Licht in OpenGL

```
GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
GLfloat light_position[] = { 1.0, 1.0, 1.0, 0.0 };

glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
glLightfv(GL_LIGHT0, GL_POSITION, light_position);

glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);
```

Materialeigenschaften

- Orientierung (z.B. bei diffusem Licht)
  - `glNormal3f(-1.0f, 0.0f, 0.0f);`
    Der Befehl `glNormal3f(x,y,z)` wird immer vor `glVertex` verwendet
    (ist in glut Funktionen dabei – `glutSolidTeapot` ...)

- Weitere Materialeigenschaften
  - `glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);`
    `glEnable(GL_COLOR_MATERIAL);`

Wozu Displaylisten?

- Statische Objekte, die sich zwischen glBegin() und glEnd() beim erneuten Aufruf nicht ändern, kann man auf der Graphikkarte zwischenspeichern
- Weniger Belastung für den Bus → Schnellere Ausführung
- Besonders lohnenswert bei großen Objekten

Vorgehensweise

- Leere Liste erzeugen

```
GLuint table;
table = glGenLists(2); // erzeugt die Liste n und n+1
```

- Liste befüllen

```
glNewList(table, GL_COMPILE);
 // glBegin und glEnd() u.s.w.
glEndList;
```

- Liste anzeigen

```
glCallList( table );
```

- Überprüfen, ob es die Liste n gibt

```
GLboolean glIsList( table );
```

- Zweite Liste

```
 GLuint tableLeg = table+1;
```

Quellen

- Olivier Faugeras; *Three-Dimensional Computer Vision (Artificial Intelligence)*; 1993
- Richard Hartley, Andrew Zisserman; *Multiple View Geometry in Computer Vision*; 2003
- Emanuele Trucco, Alessandro Verri; *Introductory Techniques for 3-D Computer Vision*; 1998
- Gary Bishop, Greg Welch, and B. Danette Allen; *Course 11—Tracking: Beyond 15 Minutes of Thought; 2001;* http://www.cs.unc.edu/~tracker/ref/s2001/tracker/
- Dave Shreiner, Mason Woo, Jackie Neider, OpenGL Programming Guide - 'The Red Book'
- NeHe Productions Tutorials: http://nehe.gamedev.net/