Grundlagen der 3D-Grafik mit OpenGl

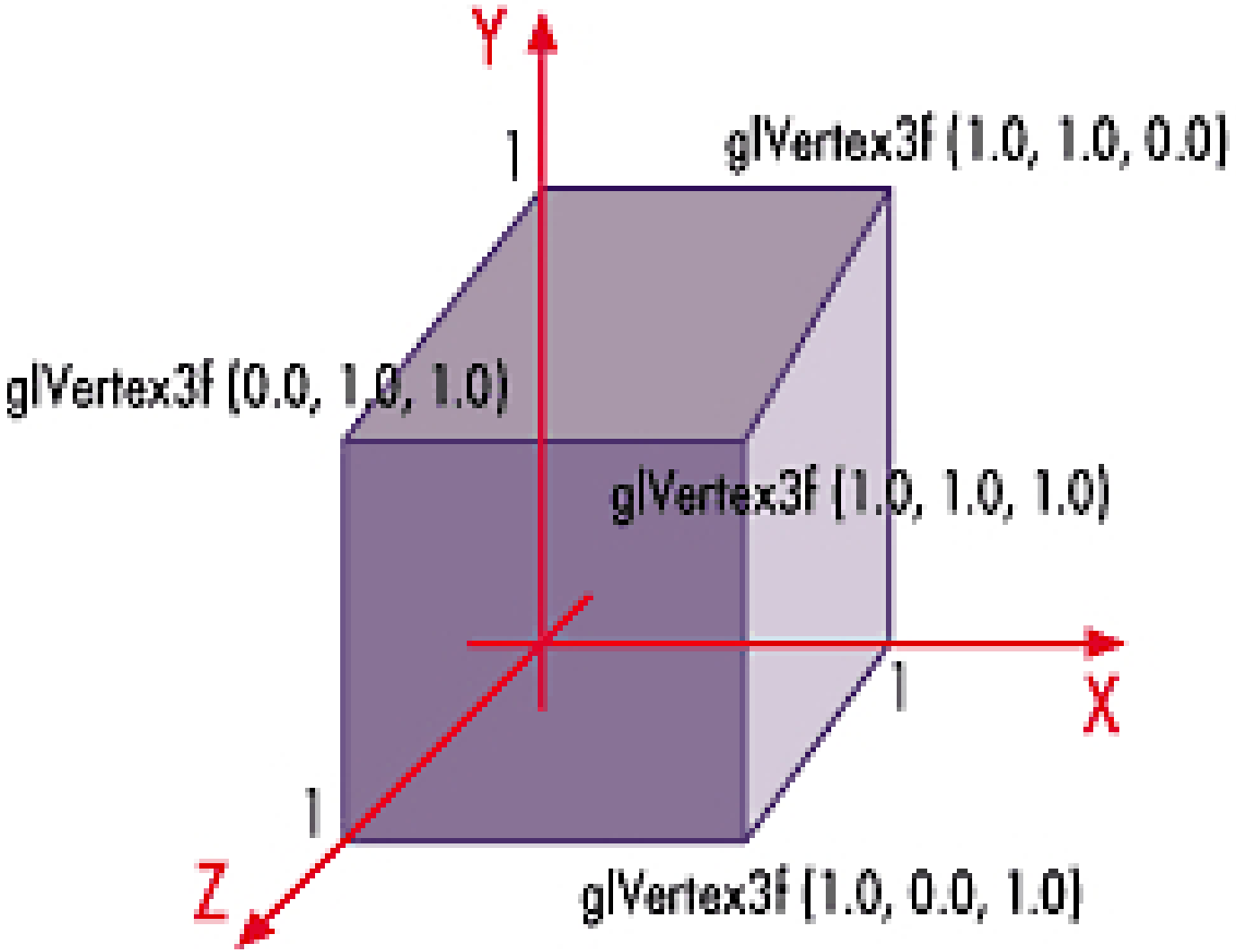
Dipl.-Inform.(FH) Martin Ongsiek

12.01.2006 - www.das-labor.org

Gliederung

Gliederung

Das OpenGl Koordinatensystem



Homogene Koordinaten

$$P = \begin{pmatrix} p_x \\ p_y \\ p_z \\ p_w \end{pmatrix} \tag{1}$$

- p_w ist üblicherweise 1
- Uniforme Behandlung von geometrischen Transformationen durch eine 4×4 Matrix
- komplexe Transformationen können durch die Kombination von elementaren Transformationen gebildet werden

Homogene Koordinaten

$$P = \begin{pmatrix} p_x \\ p_y \\ p_z \\ p_w \end{pmatrix} \tag{1}$$

- p_w ist üblicherweise 1
- Uniforme Behandlung von geometrischen Transformationen durch eine 4×4 Matrix
- komplexe Transformationen können durch die Kombination von elementaren Transformationen gebildet werden

Homogene Koordinaten

$$P = \begin{pmatrix} p_x \\ p_y \\ p_z \\ p_w \end{pmatrix} \tag{1}$$

- p_w ist üblicherweise 1
- Uniforme Behandlung von geometrischen Transformationen durch eine 4×4 Matrix
- komplexe Transformationen können durch die Kombination von elementaren Transformationen gebildet werden

Homogene Koordinaten

$$P = \begin{pmatrix} p_x \\ p_y \\ p_z \\ p_w \end{pmatrix} \tag{1}$$

- p_w ist üblicherweise 1
- Uniforme Behandlung von geometrischen Transformationen durch eine 4×4 Matrix
- komplexe Transformationen können durch die Kombination von elementaren Transformationen gebildet werden

Translatieren

glTranslate(t_x, t_y, t_z);

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2)$$

Translatieren

glTranslate(t_x, t_y, t_z);

$$\mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2}$$

Skalieren

glScale(s_x, s_y, s_z);

$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

Skalieren

glScale(s_x, s_y, s_z);

$$\mathbf{S}(s_x, s_y, s_z) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3}$$

Rotieren

glRotate(<Winkel in °>, <Anteil x>, <Anteil y>, <Anteil z>);

$$\mathbf{RX}(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

$$\mathbf{RY}(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$\mathbf{RZ}(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

Rotieren

glRotate(<Winkel in °>, <Anteil x>, <Anteil y>, <Anteil z>);

$$\mathbf{RX}(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

$$\mathbf{RY}(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$\mathbf{RZ}(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

Rotieren

glRotate(<Winkel in °>, <Anteil x>, <Anteil y>, <Anteil z>);

$$\mathbf{RX}(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

$$\mathbf{RY}(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$\mathbf{RZ}(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

Rotieren

glRotate(<Winkel in °>, <Anteil x>, <Anteil y>, <Anteil z>);

$$\mathbf{RX}(\alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4}$$

$$\mathbf{RY}(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{5}$$

$$\mathbf{RZ}(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

Gliederung

OpenGL Befehlssatz

- In Specifikation 1.1 ca. 150 recht simplen Funktionen
- Arbeitet intern als State-Maschine
- Alle Funktionen fangen mit gl an. Makros mit GL_
- Funktionen mit meheren Datentypen.
- Suffix mit Anzahl und Art der Parameter. Z.B. glVertex3f(), glVertex4d()

OpenGL Befehlssatz

- In Specifikation 1.1 ca. 150 recht simplen Funktionen
- Arbeitet intern als State-Maschine
- Alle Funktionen fangen mit gl an. Makros mit GL_
- Funktionen mit meheren Datentypen.
- Suffix mit Anzahl und Art der Parameter. Z.B. glVertex3f(), glVertex4d()

OpenGL Befehlssatz

- In Specifikation 1.1 ca. 150 recht simplen Funktionen
- Arbeitet intern als State-Maschine
- Alle Funktionen fangen mit gl an. Makros mit GL_
- Funktionen mit meheren Datentypen.
- Suffix mit Anzahl und Art der Parameter. Z.B. glVertex3f(), glVertex4d()

OpenGL Befehlssatz

- In Specifikation 1.1 ca. 150 recht simplen Funktionen
- Arbeitet intern als State-Maschine
- Alle Funktionen fangen mit gl an. Makros mit GL_
- Funktionen mit meheren Datentypen.
- Suffix mit Anzahl und Art der Parameter. Z.B. glVertex3f(), glVertex4d()

OpenGL Befehlssatz

- In Specifikation 1.1 ca. 150 recht simplen Funktionen
- Arbeitet intern als State-Maschine
- Alle Funktionen fangen mit gl an. Makros mit GL_
- Funktionen mit meheren Datentypen.
- Suffix mit Anzahl und Art der Parameter. Z.B. glVertex3f(), glVertex4d()

Prinzip

Man schreibt Punktkoordinaten
glVertex*(x, y, z);
glBegin(GL_POINTS);
... hier hinein
glEnd(GL_POINTS);
um Punkte zu malen.

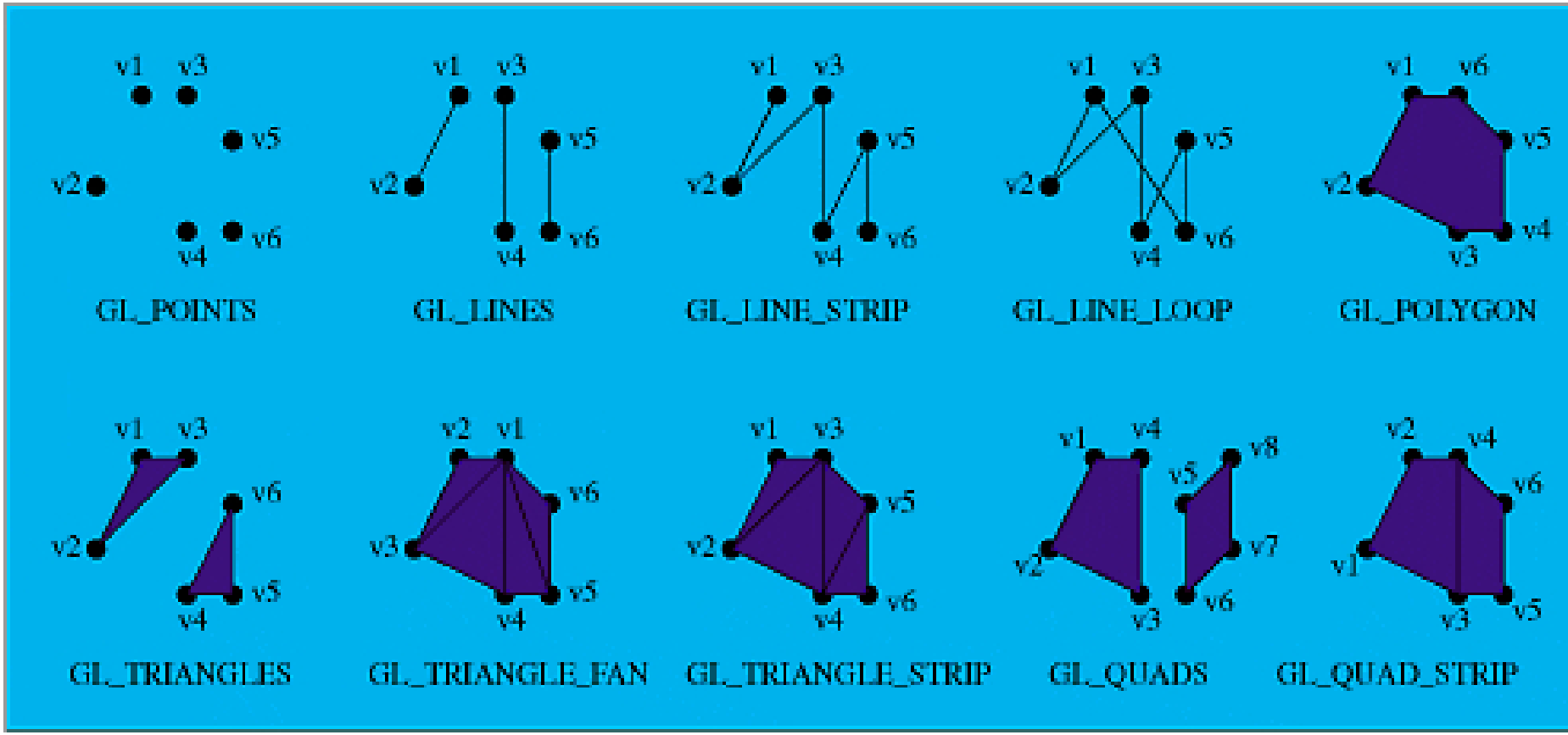
Prinzip

Man schreibt Punktkoordinaten
glVertex*(x, y, z);
glBegin(GL_POINTS);
... hier hinein
glEnd(GL_POINTS);
um Punkte zu malen.

Prinzip

Man schreibt Punktkoordinaten
glVertex*(x, y, z);
glBegin(GL_POINTS);
... hier hinein
glEnd(GL_POINTS);
um Punkte zu malen.

Mehr als nur Punkte malen



Oben und Unten?

Punkte müssen gegen der Uhrzeigersinn, also in mathematische positiver Richtung definiert werden, damit OpenGl weiß, welche Seite oben ist und daher sichtbar sind.

Oben und Unten?

Punkte müssen gegen der Uhrzeigersinn, also in mathematische positiver Richtung definiert werden, damit OpenGl weiß, welche Seite oben ist und daher sichtbar sind.

Problem bei Polygonen mit mehr als 3 Ecken

- Alle Punkte müssen auf einer Ebene liegen.
- Nur stumpfe Winkeln bei Polygonen mit mehr als 4 Ecken.

Problem bei Polygonen mit mehr als 3 Ecken

- Alle Punkte müssen auf einer Ebene liegen.
- Nur stumpfe Winkeln bei Polygonen mit mehr als 4 Ecken.

Einfärben

Mit glColor4f(c_R, c_G, c_B, c_A);
setz man die aktuelle Farbe.

- Farbwerte sind im Bereich $0 \leq c \leq 1$ definiert
- Farbwert gilt bis zum erneuten setzen

Einfärben

Mit glColor4f(c_R, c_G, c_B, c_A);
setz man die aktuelle Farbe.

- Farbwerte sind im Bereich $0 \leq c \leq 1$ definiert
- Farbwert gilt bis zum erneuten setzen

Einfärben

Mit glColor4f(c_R, c_G, c_B, c_A);
setz man die aktuelle Farbe.

- Farbwerte sind im Bereich $0 \leq c \leq 1$ definiert
- Farbwert gilt bis zum erneuten setzen

Gliederung

Pivotpunkt

- Der Pivotpunkt dient als Ausgangspunkt für Transformationen.

Matrix Stack

- glPushMatrix()
- glPopMatrix();

Benötigt man, wenn an einem Objekt mehrere Teilobjekte hängen.

Matrix Stack

- glPushMatrix()
- glPopMatrix();

Benötigt man, wenn an einem Objekt mehrere Teilobjekte hängen.

Matrix Stack

- glPushMatrix()
- glPopMatrix();

Benötigt man, wenn an einem Objekt mehrere Teilobjekte hängen.

Ende

Vielen Dank für euer Aufmerksamkeit.
Stellt euer Fragen !