Charakterisierung von (reinen) Rotationsmatrizen

- Eine Matrix R ist eine Rotationsmatrix $\Leftrightarrow$

  R ist orthogonal $\Leftrightarrow RR^{\mathsf{T}} = I$
- Erinnerung:

  $$R \text{ orthogonal } \Rightarrow \det(R) = \pm 1$$
- Achtung: dabei können noch Spiegelungen enthalten sein!
- R ist eine ordentliche Rotation $\Leftrightarrow$

  $$RR^{\mathsf{T}} = I \ \wedge \ \det(R) = +1$$

Zerlegung einer Rotationsmatrix

- Gegeben: Rotationsmatrix R

1. Aufgabe: den Rotationswinkel θ bestimmen

- Lösung:

  $$1 + 2\cos\theta = \operatorname{spur}(R)$$
- Beweis:
  - Zu R gibt es eine Basiswechselmatrix U, so daß

    $$URU^{-1} = R_x(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$
  - Es gilt:

    $$\operatorname{spur}(R) \underset{\uparrow}{=} \operatorname{spur}\left(URU^{-1}\right) = \operatorname{spur}\left(R_x(\theta)\right) = 1 + 2\cos\theta$$

    Spezielle Eigenschaft der Spur

2. Aufgabe: Rotationsachse **r** bestimmen

- Lösung: **r** ist der Eigenvektor zum Eigenwert 1 der Matrix *R*
- Beweis: alle Vektoren auf der Rotationsachse bleiben fest, d.h.

$$R\mathbf{r} = 1 \cdot \mathbf{r}$$

- Berechnung der Eigenvektoren einer 3x3-Matrix:
  - Zur Erinnerung: zu jeder Matrix *A* gibt es eine adjungierte Matrix $A^\#$
  - Die Elemente $a^\#_{ij}$ der adjungierten Matrix sind definiert als

$$a^\#_{ij} = (-1)^{i+j} \det \begin{pmatrix} a_{11} & \cdots & a_{1i} & \cdots \\ \cdots & & & \\ a_{j1} & \cdots & a_{ji} & \cdots \\ \cdots & & & \end{pmatrix}$$

  - Es gilt: $AA^\# = \det(A) I$

- Falls $\det(A) = 0$, dann ist

$$AA^\# = 0 \cdot I = 0$$

- Also gilt für jeden Spaltenvektor **v** aus $A^\#$, daß

$$A \cdot \mathbf{v} = 0$$

- Gesucht ist der Eigenvektor **r** zum Eigenwert 1 von *R*, also ein **r**, so daß

$$(R - I) \cdot \mathbf{r} = 0$$

- Das sind gerade die Spalten von

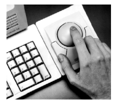
$$(R - I)^\#$$

  - Alle Spalten sind Vielfache einer der Spalten

- Bemerkung: das funktioniert auch für größere Matrizen, ist aber nicht mehr effizient

Der virtuelle Trackball

- Wie gibt man Orientierungen mit der Maus ein?
- Idee:
  - Lege Kugel um das Objekt / die Szene
  - Kugel kann um ihr Zentrum rotieren
  - Maus pickt Punkt auf Oberfläche, den man zieht



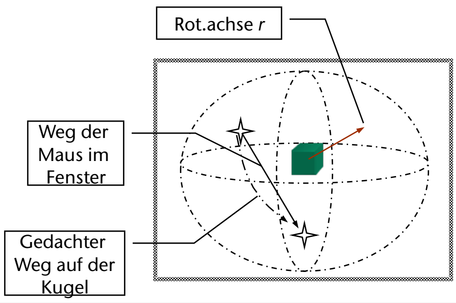
- Geg.: 2D Punkte Startpunkt = (x_1,y_1), Endpunkt = (x_2,y_2)
- Ges.: Rotationsachse r
- Berechnung:
  1. Bestimme 3D Punkte

$$\mathbf{p}_i = (x_i, y_i, z_i)$$

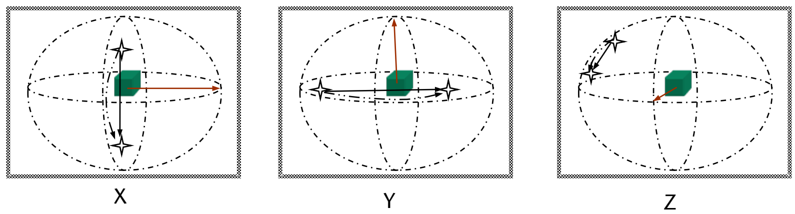
$$z_i = 1 - \sqrt{x_i^2 + y_i^2}$$

  2. Rotationsache

$$\mathbf{r} = \mathbf{p}_1 \times \mathbf{p}_2$$



- Man kann um alle Achsen (bis auf eine) direkt rotieren:



- Verbesserungen:
  - "Spinning trackball" vermeidet häufiges Nachfassen
  - "Locking" für exaktes Rotieren um eine Koord.achse
  - Was macht man, wenn (x,y) die Ellipse verlassen?
    - Nichts(?) → z wird negativ → dann noch x,y am Kreis nach innen spiegeln → p liegt auf der Rückseite der Kugel

Anatomie einer Matrix

- Erst Rotation, dann Translation:

$$P' = (TR)\,P = MP = R_{3x3} \cdot P + T$$

$$M = \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \left(\begin{array}{ccc|c} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right) = \begin{pmatrix} R & T \\ 0 & 1 \end{pmatrix}$$

- Erst Translation, dann Rotation:

$$P' = (RT)\,P = MP \cong R\,(P + T) = RP + RT$$

$$M = \begin{pmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} R_{3x3} & R_{3x3}\,T_{3x1} \\ 0_{1x3} & 1 \end{pmatrix}$$

- Allgemeiner Aufbau:

Rotationen

Skalierung

$$\begin{pmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation

Projektionen (gleich)

Starre Transformationen (*Rigid-Body Transform*)

- Hintereinanderausführung von Translationen und Rotationen
- Erhält Längenverhältnisse und Winkel eines Objektes
  - Objekte werden nicht deformiert / verzerrt
- Allgemeine Form:

$$M = T_t R = \begin{pmatrix} r_{00} & r_{01} & r_{02} & t_x \\ r_{10} & r_{11} & r_{12} & t_y \\ r_{20} & r_{21} & r_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Inverse Rigid-Body Transformation:

$$M^{-1} = (T_t R)^{-1} = R^{-1} T_t^{-1} = R^{\mathsf{T}} T_{-t}$$

$$M = \begin{pmatrix} R & t \\ 0^T & 1 \end{pmatrix} \qquad M^{-1} = \begin{pmatrix} R^{\mathsf{T}} & -R^{\mathsf{T}} t \\ 0 & 1 \end{pmatrix}$$

Transformationen in OpenGL

- Einfache Befehle zur Objekttransformation:

```
glRotate{fd}( TYPE angle, x, y, z );
```

rotiert um **`angle`** Grad(!) um die angegeben Achse;

```
glTranslate{fd}( TYPE x,y,z );
```

transliert um den angegebenen Betrag;

```
glScale{fd}( TYPE x,y,z );
```

skaliert um die angegebenen Faktoren.

- Ein **glRotate** / **glTranslate** (u.ä.) wirkt sich nur auf die nachfolgende Geometrie aus!

Matrizen in OpenGL

- Es gibt eine „globale" Matrix „**`MODELVIEW`**", die anfangs mit der Einheitsmatrix besetzt ist
- Jeder Aufruf von **`glRotate`**, **`glScale`** etc. resultiert in der Multiplikation der entsprechenden Matrix mit der „globalen" Matrix von **rechts**, z.B.

```
glScalef( sx,sy,sz )
```

$$\Longleftrightarrow \quad M_{\text{MODELVIEW}} \cdot \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

```
glTranslatef( tx,ty,tz )
```

$$\Longleftrightarrow \quad M_{\text{MODELVIEW}} \cdot \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Beachte die Reihenfolge in einer Matrixkette:

Reihenfolge der OpenGL-Befehle

$$p' = M_n \cdot \ldots \cdot M_2 \cdot M_1 \cdot p$$

Reihenfolge der Ausführung

- Die Anordnung entspringt aus dem Programmablauf
- Konzeptionell kann man es sich wie folgt vorstellen:

```
glScalef(1.5,1,1);
glTranslatef(.2,0,0);
glRotatef(30,0,0,1);
render geometry
```

„Die Geometrie wandert rückwärts durch das Programm und sammelt die Transformationen ein"

Direkte Matrizenspezifizierung

- Man kann auch direkt Matrizen angeben:

```
glMultMatrix{fd}( TYPE * m );
```

multipliziert die Matrix auf die aktuelle **MODELVIEW**-Matrix

```
glLoadMatrix{fd}( TYPE * m );
```

ersetzt die aktuelle **MODELVIEW**-Matrix durch die angegebene.

```
glLoadIdentity();
```

Spezialfall: Einheitsmatrix

- Matrixabfrage:

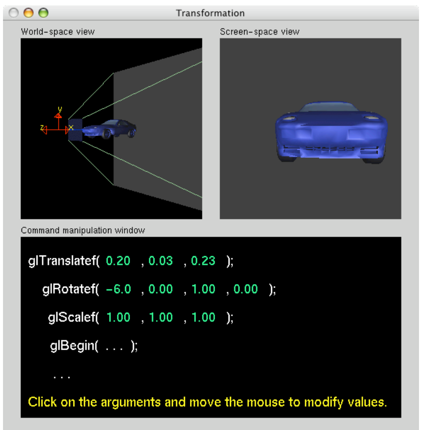
```
glGetFloatv( GL_MODELVIEW_MATRIX, float * m );
```

- Achtung: Matrizen werden **spaltenweise** abgelegt, nicht, wie in C üblich, zeilenweise!
  - Das nent sich *"column-major order"* (der Standard, z.B. in C, ist *row-major order*)

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \Longleftrightarrow$$

```
GLfloat matrix[] =
{
        1, 0, 0, 0,
        0, 1, 0, 0,
        0, 0, 1, 0,
        tx, ty, tz, 1
};
```

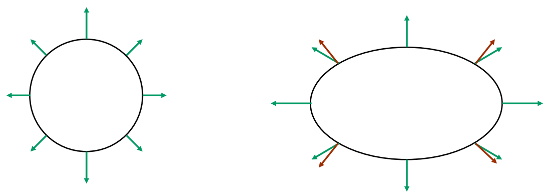

Demo ...



http://www.xmission.com/~nate/tutors.html

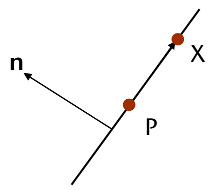
Transformation von Normalen

- Behauptung:
  Wenn das Objekt um M transformiert wird, dann müssen die Normalen der Oberfläche um $N = (M^{-1})^T$ transformiert werden
- Bei starren Transformationen:
  - Translation beeinflusst die Normalen der Oberfläche nicht
  - Im Fall der Rotation ist $M^{-1} = M^T$ und somit $N = M$
- Bei nicht-uniformer Skalierung und Scherung ist $N = (M^{-1})^T \neq M$!
  - Beispiel:



- Beweis:

wir wissen



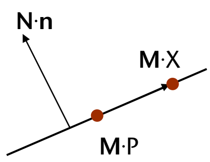
$$(X - P)^\mathsf{T}\mathbf{n} = 0$$

wir hätten gerne

$$(M \cdot X - M \cdot P)^\mathsf{T} \cdot (N \cdot \mathbf{n}) = (X - P)^\mathsf{T} \cdot M^\mathsf{T} \cdot N \cdot \mathbf{n} = 0$$

setze also

$$N = \left(M^\mathsf{T}\right)^{-1}$$

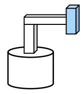


damit ist

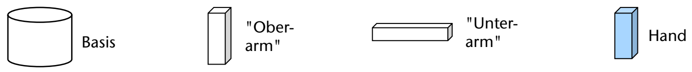
$$(X - P)^\mathsf{T} \cdot M^\mathsf{T}(M^\mathsf{T})^{-1} \cdot \mathbf{n} = (X - P)^\mathsf{T} \cdot I \cdot \mathbf{n} = 0$$

Relative Transformationen

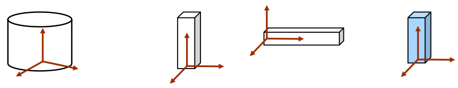
- Eine Konkatenierung von Transformationen kann man auch als eine Folge von (voneinander abhängigen) Koordinatensystemen ansehen
- Beispiel: Roboter



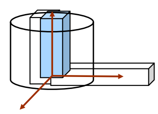
  - Besteht aus diesen Einzelteilen



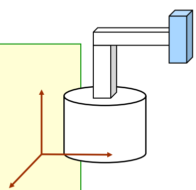
  - Jedes Teil wurde in seinem eigenen Koordinatensystem spezifiziert (als Array von Punkten) → heißt Objektkoordinatensystem



  - Rendert man alle Teile ohne jede Transformation, entsteht folgendes:



- Würde man jedes Teil, ausgehend vom Ursprung des Weltkoordinatensystems, an seinen Platz transformieren, sähe das ungefähr so aus:



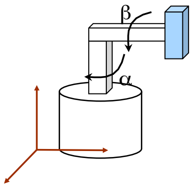
```
glLoadIdentity();
// set up camera
glTranslatef( tr_base_x, tr_base_y , … );
render base …

glLoadIdentity();
// set up camera
glTranslatef( tr_base_x, tr_base_y + 10, … );
render upper arm …

glLoadIdentity();
// set up camera
glTranslatef( tr_base_x, tr_base_y + 10 + 5, … );
render lower arm …
. . .
```

Ann.: Höhe der Basis ist 10

Ann.: Höhe des Oberarms ist 5

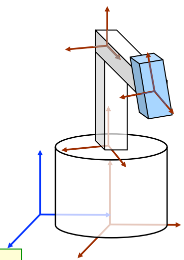
- Natürlich macht man es ungefähr so:

```
glLoadIdentity();
// set up camera
glTranslatef( tr_base_x, tr_base_y , … );
render base …
glTranslatef( 0, HEIGHT_BASE, 0 );
glRotatef( alpha, 0, 1, 0 );
render upper arm …
glTranslatef( 0, HEIGHT_UPPER_ARM, 0 );
glRotatef( beta, 1, 0, 0 );
render lower arm …
glTranslatef( X_SIZE_LOWER_ARM, 0, 0 );
render hand …
```



Solche Parameter würde man natürlich in einer Klasse 'Roboter' als Instanzvariablen speichern

- Alternative Betrachtungsweise ist, daß bei jeder Transformation ein neues lokales Koordinaten-system entsteht, das bezüglich seines Vater-Koordinatensystems um genau diese Transf. transformiert ist



In dieser Reihenfolge entstehen die lokalen Koordinaten-systeme aus dem Weltkoordinaten-system

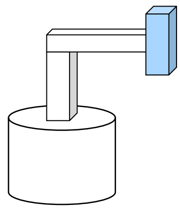
```
glLoadIdentity();
// set up camera
glTranslatef( tr_base_x, tr_base_y , … );
render base …
glTranslatef( 0, HEIGHT_BASE, 0 );
glRotatef( alpha, 0, 1, 0 );
render upper arm …
glTranslatef( 0, HEIGHT_UPPER_ARM, 0 );
glRotatef( beta, 1, 0, 0 );
render lower arm …
glTranslatef( X_SIZE_LOWER_ARM, 0, 0 );
render hand …
```

In dieser Reihenfolge werden die Transfor-mationen auf die Geometrie (d.h., die Punkte) angewendet

Objekthierarchien

- Dadurch ergibt sich eine Abhängigkeit der Objekte
  - Sie betrifft vor allem deren Transformationen
  - Betrifft später auch andere Attribute (z.B. Farbe)
- Der so definierte Baum heißt Szenengraph

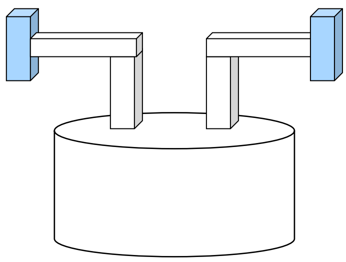
Basis → Unterarm → Oberarm → Hand



- Bemerkung: wir werden in diesem Semester Szenengraphen noch nicht explizit darstellen

- Ein etwas komplizierteres Beispiel:

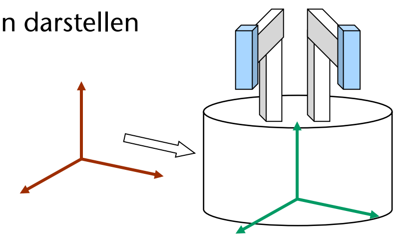
Linker Arm Rechter Arm



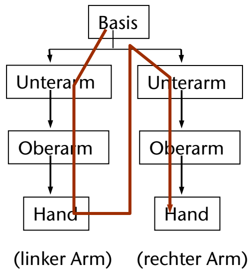
Basis

Basis
- Unterarm → Oberarm → Hand (linker Arm)
- Unterarm → Oberarm → Hand (rechter Arm)

- Aufgabe: folgende Konfiguration darstellen



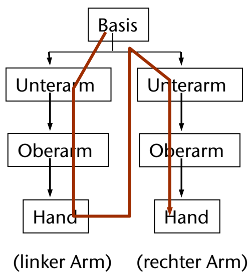
- Natürliche Vorgehensweise ist Depth-First-Traversal durch den Szenengraph:

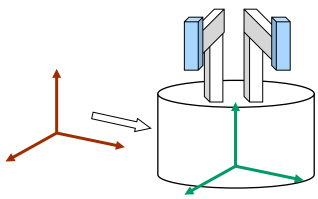



```
Do transformation(s)
Draw base
Do transformation(s)
Draw left arm
Do transformation(s)
Draw right arm
```

Welche sind das ??

Erster (falscher) Versuch





```
Translate(5,0,0)
Draw base
Rotate(75, 0, 1, 0)
Draw left arm
Rotate(-75, 0, 1, 0)
Draw right arm
```

Was ist hier falsch?!

Antwort: der rechte Arm soll relativ zur Basis um -75 Grad gedreht sein, in diesem Programm aber wird er relativ zum linken Arm gedreht!

Lösung

```
Initiale MODELVIEW Matrix M
Translate(5,0,0) → M = M·T
Draw base
Rotate(75, 0, 1, 0)
Draw left arm
Rotate(-75, 0, 1, 0)
Draw right arm
```

Speichere die MODELVIEW-Matrix an dieser Stelle in einem Zwischenspeicher

Restauriere diese gemerkte MODELVIEW-Matrix an dieser Stelle aus dem Zwischenspeicher

⇨ Lösung: ein Matrix-Stack

```
Initiale MODELVIEW Matrix M
Translate(5,0,0) → M = M·T
Draw base
Rotate(75, 0, 1, 0)
Draw left arm
Rotate(-75, 0, 1, 0)
Draw right arm
```

An dieser Stelle die aktuelle MODELVIEW-Matrix auf den Stack pushen

An dieser Stelle die oberste Matrix vom Stack pop-en und in die MODELVIEW-Matrix schreiben

Der Matrix-Stack in OpenGL

- In OpenGL gibt es einen MODELVIEW-Matrix-Stack
- Die oberste Matrix auf diesem Stack ist die aktuelle MODELVIEW-Matrix, die für die Geometrie-Transformation verwendet wird
- Alle Transformations-Kommandos (glLoadMatrix, glMultMatrix, glTranslate, ...) operieren auf dieser obersten Matrix!
- Operationen:

```
glPushMatrix();
```

dupliziert die oberste Matrix auf dem Stack und legt diese oben auf dem Stack ab;

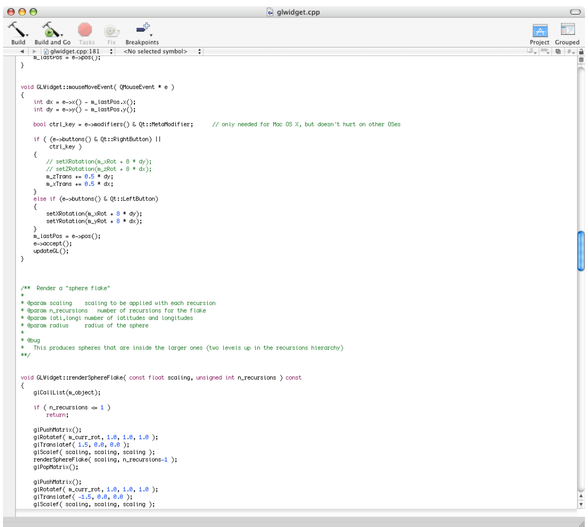
```
glPopMatrix();
```

wirft die oberste Matrix vom Stack weg.

Beispiel

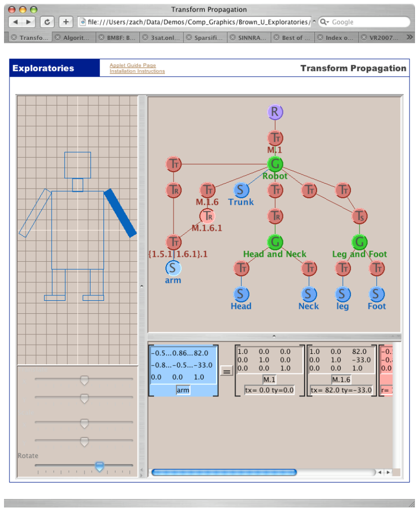
Code	Aktuelle MODELVIEW-Matrix:	Zustand des Matrix-Stacks:		
`glMatrixMode(GL_MODELVIEW);`				
`glLoadIdentity();`	I	I		
`glMultMatrix( M1 );`	M1	M1		
`glTranslate( T );`	M1·T	M1·T		
`glPushMatrix();`	M1·T	M1·T	M1·T	
`glRotate( R );`	M1·T·R	M1·T	M1·T·R	
`glPushMatrix();`	M1·T·R	M1·T	M1·T·R	M1·T·R
`glMultMatrix( M2 );`	M1·T·R·M2	M1·T	M1·T·R	M1·T·R·M2
`glPopMatrix();`	M1·T·R	M1·T	M1·T·R	
`glScale( S );`	M1·T·R·S	M1·T	M1·T·R·S	
`glPopMatrix();`	M1·T·R	M1·T		

Beispiel im Code




```
    m_lastPos = e->pos();
}


void GLWidget::mouseMoveEvent( QMouseEvent * e )
{
    int dx = e->x() - m_lastPos.x();
    int dy = e->y() - m_lastPos.y();

    bool ctrl_key = e->modifiers() & Qt::MetaModifier;      // only needed for Mac OS X, but doesn't hurt on other OSes

    if ( (e->buttons() & Qt::RightButton) ||
         ctrl_key )
    {
        // setXRotation(m_xRot + 8 * dy);
        // setZRotation(m_zRot + 8 * dx);
        m_zTrans += 0.5 * dy;
        m_xTrans += 0.5 * dx;
    }
    else if (e->buttons() & Qt::LeftButton)
    {
        setXRotation(m_xRot + 8 * dy);
        setYRotation(m_yRot + 8 * dx);
    }
    m_lastPos = e->pos();
    e->accept();
    updateGL();
}



/**  Render a "sphere flake"
*
* @param scaling    scaling to be applied with each recursion
* @param n_recursions   number of recursions for the flake
* @param lati,longi number of latitudes and longitudes
* @param radius     radius of the sphere
*
* @bug
*   This produces spheres that are inside the larger ones (two levels up in the recursions hierarchy)
**/


void GLWidget::renderSphereFlake( const float scaling, unsigned int n_recursions ) const
{
    glCallList(m_object);

    if ( n_recursions <= 1 )
        return;

    glPushMatrix();
    glRotatef( m_curr_rot, 1.0, 1.0, 1.0 );
    glTranslatef( 1.5, 0.0, 0.0 );
    glScalef( scaling, scaling, scaling );
    renderSphereFlake( scaling, n_recursions-1 );
    glPopMatrix();

    glPushMatrix();
    glRotatef( m_curr_rot, 1.0, 1.0, 1.0 );
    glTranslatef( -1.5, 0.0, 0.0 );
    glScalef( scaling, scaling, scaling );
```

Demo zum Szenengraph



http://www.cs.brown.edu/exploratories → Transformation Propagation

Klassifikation aller Transformationen