Ludwig-Maximilians-Universität München
LFE Medieninformatik
Prof. Dr. Andreas Butz
Dominikus Baur, Sebastian Boring

Sommersemester 2009
Übungsblatt 3
08. Mai 2009

Computergrafik 1

Abgabetermin:
Die Lösung zu diesem Übungsblatt ist bis zum Freitag den **15. Mai 2009, 12:00 Uhr s.t.** über UniWorx abzugeben.

Inhalt:
Dieses Blatt dient zum Erlernen von grundlegenden Transformationen in *OpenGL*. Hierzu werden Sie in Ihrem bereits erstellten Fenster weitere Interaktionsmöglichkeiten für die Kamera realisieren. Diese sind wiederum an die gängigen 3D Modellierungswerkzeuge angelehnt. Zusätzlich erlernen Sie die mathematischen Prinzipien der Transformationen in einem Szenegraphen. Dabei lernen Sie zudem das Konzept der hierarchischen Transformationen. Pro Aufgabe können maximal zehn Punkte erreicht werden. Zum Bestehen des Übungsblatts müssen in **JEDER** Aufgabe mindestens fünf Punkte erreicht werden.

Geben Sie zu jeder Aufgabe alle benötigten Header-, Source-, und Projektdateien (von *Qt Creator* erzeugt) mit ab, d.h. `*.h`, `*.cpp` und `*.pro` Dateien. Abgaben die nicht kompilieren werden mit 0 Punkten bewertet. Fassen Sie alle Aufgaben zu einer zip-Datei zusammen und laden Sie sie bei UniWorx hoch. Für die Mathematik-Aufgaben (Aufgaben 8 und 9) fügen Sie jeweils ein eingescanntes Bild oder ein PDF (falls mit Word oder LaTeX erstellt) in die Ordner „auf3-8“ bzw. „auf3-9“ ein.

Aufgabe 8 Allgemeine Transformationen, Homogene Transformationen (10 Punkte)

Bewegungen von Objekten im mehrdimensionalen Raum können auf verschiedene Weisen dargestellt werden. Die drei Basis-Transformationen sind *Translation*, *Rotation* und *Skalierung*. Eine weitere Transformation ist die *Scherung*, die jedoch hier nicht näher betrachtet werden soll. Im drei-dimensionalen Raum werden die Transformationen folgendermaßen beschrieben:

- Translation: $\vec{v}' = \vec{v} + \vec{t}$ wobei $\vec{t} = (t_x, t_y, t_z)^T$.
- Rotation um x-Achse: $R(\omega, 0, 0) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\omega & -\sin\omega \\ 0 & \sin\omega & \cos\omega \end{pmatrix}$
- Rotation um y-Achse: $R(0, \varphi, 0) = \begin{pmatrix} \cos\varphi & 0 & \sin\varphi \\ 0 & 1 & 0 \\ -\sin\varphi & 0 & \cos\varphi \end{pmatrix}$
- Rotation um z-Achse: $R(0, 0, \kappa) = \begin{pmatrix} \cos\kappa & -\sin\kappa & 0 \\ \sin\kappa & \cos\kappa & 0 \\ 0 & 0 & 1 \end{pmatrix}$
- Skalierung: $\vec{v}' = (s_x \cdot v_x, s_y \cdot v_y, s_z \cdot v_z)^T$ wobei s_x, s_y und s_z Skalare (ungleich 0) sind.

Die *Translationsmatrix* $T = (t_x, t_y, t_z)^T$ stellt eine Verschiebung vom Ursprung des Koordinatensystems dar. Die allgemeine Transformationsgleichung

$$p' = R \cdot p + t$$

ist nicht linear und kann daher nicht invertiert werden. Für Szenegraphen ist die Invertierung jedoch zwingend notwendig. Durch Hinzufügen einer weiteren Dimension, d.h.

$$\begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \mapsto \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda \cdot p_x \\ \lambda \cdot p_y \\ \lambda \cdot p_z \\ \lambda \end{pmatrix}$$

kann diese Gleichung wieder in eine lineare Abbildung umgewandelt werden:

$$p^* = \begin{pmatrix} p_{x'} \\ p_{y'} \\ p_{z'} \\ 1 \end{pmatrix} = \begin{pmatrix} & & & t_x \\ & R(\omega, \varphi, \kappa) & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

Diese 4×4 Matrixdarstellung rigider Transformationen im 3D-Raum ist in der Computergraphik sehr gebräuchlich und wird *Homogene Transformationsmatrix* genannt.

a) Zeigen Sie, dass die o.g. lineare Gleichung genau der Berechnung $p' = R \cdot p + t$ entspricht. (**3 Punkte**)

b) Erstellen Sie für alle fünf oben genannten Matrizen (Translation, 3 Rotationsmatrizen und die Skalierung) deren *Homogene Transformationsmatrix*. Welcher Vorteil ergibt sich aus der einheitlichen 4×4 Matrix? (**4 Punkte**)

c) Transformieren Sie nun den Punkt $\vec{t} = (1,1,1)^T$ mit den zu Beginn angegebenen Transformationen wie folgt. Drehen Sie ihn zunächst um 45° um die x-Achse, verschieben Sie ihn anschließend um den Vektor $\vec{t} = (5,2,-3)^T$, und skalieren Sie ihn dann um den Faktor 5 (auf x- und y-Achse). Wie lassen sich diese Transformationen durch **eine einzige** *Homogene Transformationsmatrix* darstellen? (**3 Punkte**)

Aufgabe 9 Der Szenegraph (10 Punkte)

Der Szenegraph ist eine in der Computergraphik häufig verwendete hierarchische Darstellungsform von 3D-Szenen. Es handelt sich dabei um einen DAG (Directed Acyclic Graph), oft auch um einen Baum. In diesem Graph kann man drei Typen von Knoten unterscheiden. Gruppenknoten dienen der Zusammenfassung mehrerer Kinder zur Hierarchisierung von Objekten, Transformationsknoten beschreiben geometrische Transformationen, also z.B. Rotationen oder Translationen, und Strukturknoten beschreiben geometrische Objekte, z.B. ein Tisch oder Stühle.

Beim Rendern durchläuft man den Graphen nun mit einer Tiefensuche, wobei die Kinder von Gruppenknoten in fester Ordnung abgearbeitet werden. Geht man in einem Szenengraphen "von unten nach oben", so entspricht dies der Ausführung von *invertierten* Transformationen. Hierdurch ist es z.B. möglich, die Position eines Stuhls relativ zum Tisch durch folgende Berechnung zu bestimmen:

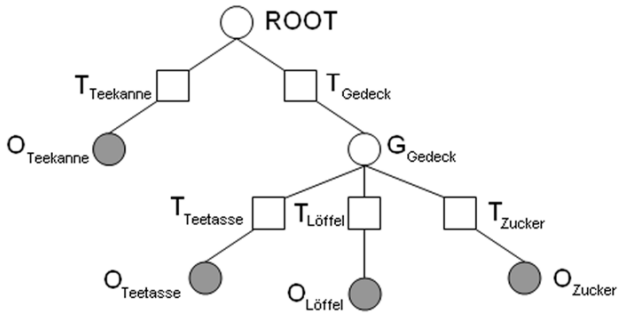

Abbildung 1: Beispielszenegraph: Ein Teegedeck

$$\mathbf{T}_{gesucht} = \mathbf{T}_{Tisch}^{-1}$$

a) Berechnen Sie analytisch die Linksinverse $\mathbf{H}^{-1,L}$

$$\mathbf{H}^{-1,L} = \begin{pmatrix} \mathbf{R}^* & t^* \\ 0 & 1 \end{pmatrix} \text{ mit } \mathbf{H}^{-1,L} \cdot \begin{pmatrix} \mathbf{R}_x & t_x \\ 0 & 1 \end{pmatrix} = \mathbf{I}$$

in Abhängigkeit von den Parametern $\mathbf{R}_x$ und t_x einer homogenen Transformationsmatrix. **Tipp:** Da $\mathbf{R}_x$ eine orthonormale Rotationsmatrix ist, gilt $\mathbf{R}_x^{-1} = \mathbf{R}_x^T$. (**6 Punkte**)

b) Die Kamera sei nun ein weiteres Objekt im Szenegraph. Stellen Sie allgemein die Transformation vom Zucker zur Kamera und umgekehrt auf. Welche der beiden Transformationen benötigen Sie, wenn Sie mit *OpenGL* üblicherweise zeichnen? (**4 Punkte**)

Aufgabe 10 Wechseln der Ansicht (10 Punkte)

Neben den bereits erstellten Ansichten *Top*, *Left*, *Front* und *Perspective* sollen nun noch drei weitere Ansichten (*Bottom*, *Right* und *Back*) hinzugefügt werden. Da Sie jedoch nach wie vor nur vier Ansichten haben, soll der Benutzer die Ansichten identifizieren (d.h. welche Ansicht ist gerade in dem jeweiligen Fenster aktiv) und wechseln können.

a) Zeigen Sie den Namen der Ansicht (wahlweise deutsch oder englisch) in der linken oberen Ecke der jeweiligen Ansicht an. Verwenden Sie hierzu eine Farbe, die aufgrund der Hintergrundfarbe des Fensters (gesetzt durch `qglClearColor`) deutlich lesbar ist. Verwenden Sie zum Zeichnen des `QString` (der den Namen der Ansicht enthält) die Funktion `renderText`. Diese Funktion erfordert x- und y-Koordinaten (linker, **unterer** Punkt des zu zeichnenden Textes), den `QString` selbst, sowie eine Schrift `QFont`. Für die Schrift verwenden Sie bitte *Arial* mit *normalem* Stil (d.h. weder fett noch kursiv) und der Größe 10. (**3 Punkte**)

b) Erlauben Sie dem Benutzer das Wechseln der Ansicht mit Hilfe eines Kontextmenüs. Der Benutzer soll die Möglichkeit haben, ein Kontextmenü durch einen Rechtsklick auf den jeweiligen Ansichtsnamen geöffnet werden. Die Name der aktuellen Ansicht ist dann selbstverständlich ausgegraut, da diese nicht erneut gewählt werden soll. Gehen Sie dabei folgendermaßen vor:

- Überschreiben Sie in Ihrem `GLWidget` zunächst die für das Kontextmenü verantwortliche Funktion `contextMenuEvent(`**`QContextMenuEvent`** `*event)`. Diese Funktion wird immer dann ausgeführt, wenn ein Kontextmenü-Event (definiert durch das Betriebssystem, d.h. üblicherweise ein Rechtsklick erfolgt). Dies bedeutet, dass Sie

nicht in der bereits überschriebenen Methode `mousePressEvent` auf das Ereignis reagieren sollen! Setzen Sie nun noch die `ContextMenuPolicy` in Ihrem `GLWidget` auf `Qt::DefaultContextMenu` (bereits im Konstruktor).

- Berechnen Sie sich die *Breite* und *Höhe* des Namens (d.h. des gezeichneten `QString`) in der jeweiligen Ansicht, da Sie nur in dieser Region überhaupt auf das Event reagieren sollen (**Hinweis:** Später werden Sie Kontextmenüs direkt in der Szene für andere Zwecke verwenden!). Verwenden Sie hierzu die Klasse `QFontMetrics` (siehe Dokumentation). **Tipp:** Die Position, die im `event` enthalten ist, beschreibt die relative Position in Ihrem `GLWidget`.
- Erzeugen Sie im Erfolgsfall nun das Menü mit der Klasse `QMenu` und zeigen Sie es an. Dazu müssen Sie zunächst eine `QAction` für alle Ansichten erzeugen (auch für die jeweils aktive). Diesen Aktionen geben Sie einen Titel (d.h. den Namen der Ansicht, die im Kontextmenü stehen soll), und das `GLWidget`. Danach verbinden Sie jede Aktion mit einem eigens definierten `slot` mit Hilfe der Funktion `connect` (**Tipp:** das Signal lautet hier `triggered`). Fügen Sie nun jede Aktion dem gerade erstellten Kontextmenü hinzu und achten Sie darauf, dass die Aktion, die zur gerade aktiven Ansicht gehört, inaktiv ist.
- Zeigen Sie das Menü nun mit der Funktion `exec` der Klasse `QMenu` an. Diese erwartet allerdings absolute Koordinaten, das `event` bietet jedoch nur relative Koordinaten an. **Tipp:** Hierfür eignet sich die Funktion `mapToGlobal`.

(**4 Punkte**)

c) Reagieren Sie nun in den eigens definierten `Slots` auf einen Wechsel der Ansicht (also auf eine Ausführung der vorhin erzeugten Aktionen), d.h. repositionieren und drehen Sie die Kamera entsprechend der gewählten Ansicht. Zur Vereinfachung können hier Standard-Werte angegeben werden. (**3 Punkte**)

Aufgabe 11 Rotation der Kamera (10 Punkte)

Bisher können Sie die Kamera in der perspektivischen Ansicht lediglich in alle Richtungen verschieben. Allerdings sollen auch Rotationen möglich sein. Während die *Translation* mit der linken Maustaste durchgeführt wird, sollen *Rotationen* nun mit der rechten Maustaste realisiert werden.

a) Erweitern Sie die Funktionen `mousePressEvent` sowie `mouseMoveEvent` um auf ein Ereignis reagieren zu können, das bei gedrückter rechter Maustaste ausgelöst wird. (**1 Punkt**)

b) Da Sie drei mögliche *Rotationen* haben, sich jedoch nur zweidimensional mit der Maus bewegen können, soll bei Verschieben der Maus in x-Richtung (y-Richtung) um die y-Achse (x-Achse) der Kamera gedreht werden. Bedenken Sie hierbei, dass die Kamera um Ihre **lokalen** Achsen gedreht werden soll. (**2 Punkte**)

c) Die dritte Achse (also die **lokale** z-Achse) der Kamera soll nun durch Drücken der *Alt*-Taste realisiert werden. Erweitern Sie im `GLWidget` die Funktionen `keyPressEvent` und `keyReleaseEvent` um eine Überprüfung auf diese Taste und setzen Sie eine Variable in `GLWidget`, um in den Maus-Events entsprechend darauf zu reagieren. (**4 Punkte**)

d) Bedenken Sie nun beim Verschieben der Kamera, dass diese sich jeweils um Ihre **lokalen** x- und y-Achsen bewegen soll. Passen Sie dazu die Funktionen der Maus-Events entsprechend an. **Tipp:** Die Szenegraphen-Aufgabe könnte Ihnen hier extrem von Vorteil sein, d.h. überlegen Sie sich, welche tatsächliche Bewegung die Maus in der Szene macht (basierend auf den lokalen Koordinaten im Kamera-Koordinatensystem)! (**3 Punkte**)