Aufgabe 14

Im wesentlichen ist der Quader lediglich in den Ursprung zu translieren (T_1), zu skalieren (S) und anschließend an die neue Position zu verschieben (T_2). Der Vergrößerungsfaktor entsteht intuitiv durch Anwendung des Strahlensatzes.

$$M = T_2(x_3, y_3, z_3) \bullet S\left(\frac{x_4 - x_3}{x_2 - x_1}, \frac{y_4 - y_3}{y_2 - y_1}, \frac{z_4 - z_3}{z_2 - z_1}\right) \bullet T_1(-x_1, -y_1, -z_1)$$

$$= \begin{pmatrix} 1 & 0 & 0 & x_3 \\ 0 & 1 & 0 & y_3 \\ 0 & 0 & 1 & z_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \bullet \begin{pmatrix} \frac{x_4 - x_3}{x_2 - x_1} & 0 & 0 & 0 \\ 0 & \frac{y_4 - y_3}{y_2 - y_1} & 0 & 0 \\ 0 & 0 & \frac{z_4 - z_3}{z_2 - z_1} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \bullet \begin{pmatrix} 1 & 0 & 0 & -x_1 \\ 0 & 1 & 0 & -y_1 \\ 0 & 0 & 1 & -z_1 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{x_4 - x_3}{x_2 - x_1} & 0 & 0 & x_3 - x_1 \cdot \frac{x_4 - x_3}{x_2 - x_1} \\ 0 & \frac{y_4 - y_3}{y_2 - y_1} & 0 & y_3 - y_1 \cdot \frac{y_4 - y_3}{y_2 - y_1} \\ 0 & 0 & \frac{z_4 - z_3}{z_2 - z_1} & z_3 - z_1 \cdot \frac{z_4 - z_3}{z_2 - z_1} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Zu beachten ist, dass wenn der Nenner eines Bruches Null ist, eine unzulässige Division entstehen würde. Um das zu umgehen, wird dann der komplette Bruch als Null angesehen, weil der Quader in diese Richtung keine Ausdehnung hat. Allerdings darf der Zielquader in dieser Richtung ebenfalls nur die Ausdehnung Null besitzen, weil sonst die eine Skalierung nicht darstellbar wäre.

Aufgabe 15

Unter Benutzung der Formel, die ich in Aufgabe 16 entwickelt habe und der Beziehung

$$N = \frac{U}{|U|} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \frac{1}{\sqrt{3}}$$

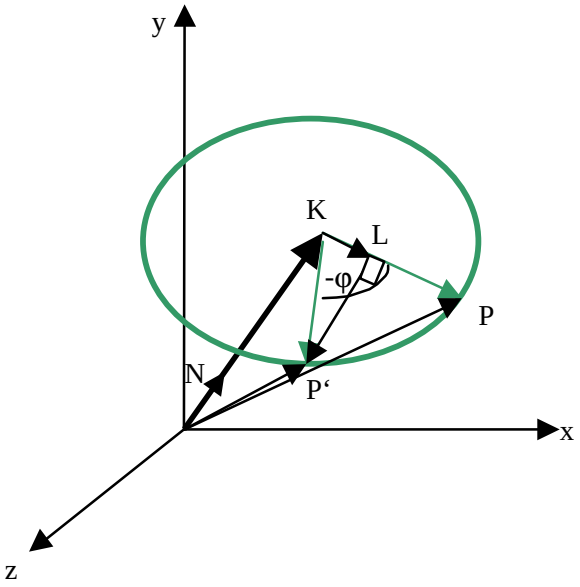
erhält man die Transformationsmatrix:

$$R(N,\varphi) = \begin{pmatrix} \cos\varphi + n_x^2 \cdot (1-\cos\varphi) & n_x n_y \cdot (1-\cos\varphi) - n_z \cdot \sin\varphi & n_x n_z \cdot (1-\cos\varphi) + n_y \cdot \sin\varphi & 0 \\ n_x n_y \cdot (1-\cos\varphi) + n_z \cdot \sin\varphi & \cos\varphi + n_y^2 \cdot (1-\cos\varphi) & n_y n_z \cdot (1-\cos\varphi) - n_x \cdot \sin\varphi & 0 \\ n_x n_z \cdot (1-\cos\varphi) - n_y \cdot \sin\varphi & n_y n_z \cdot (1-\cos\varphi) + n_x \cdot \sin\varphi & \cos\varphi + n_z^2 \cdot (1-\cos\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} \frac{1+2\cos\varphi}{3} & \frac{1-\cos\varphi}{3} - \frac{\sin\varphi}{\sqrt{3}} & \frac{1-\cos\varphi}{3} + \frac{\sin\varphi}{\sqrt{3}} & 0 \\ \frac{1-\cos\varphi}{3} + \frac{\sin\varphi}{\sqrt{3}} & \frac{1+2\cos\varphi}{3} & \frac{1-\cos\varphi}{3} - \frac{\sin\varphi}{\sqrt{3}} & 0 \\ \frac{1-\cos\varphi}{3} - \frac{\sin\varphi}{\sqrt{3}} & \frac{1-\cos\varphi}{3} + \frac{\sin\varphi}{\sqrt{3}} & \frac{1+2\cos\varphi}{3} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Aufgabe 16

Um die Berechnungen zu vereinfachen, gehe ich davon aus, dass die Rotationsachse U normiert ist, d.h. $N = |U| = 1$.



Die Rotation von P auf $P‘$ hat dann die Gestalt:

$$P' = K + L + (L - P')$$

Der Vektor K ist richtungsgleich mit der Normalen, er hat jedoch eine derart gestaltete Länge, dass sie dem Abstand der gedachten Ebene, in der die Rotation vollzogen wird, vom Koordinatenursprung entspricht. L-$P‘$ steht senkrecht auf P-K. L ist eine Verkürzung von P-K.

$$\begin{aligned} K &= N \cdot (N \bullet P) \\ L &= (P - K) \cdot \cos(-\varphi) \\ &= (P - K) \cdot \cos\varphi \\ L - P' &= (P \times N) \cdot \sin(-\varphi) \\ &= -(P \times N) \cdot \sin\varphi \\ &= (N \times P) \cdot \sin\varphi \end{aligned}$$

folgt

$$\begin{aligned} P' &= N \cdot (N \bullet P) + (P - N \cdot (N \bullet P)) \cdot \cos\varphi + (N \times P) \cdot \sin\varphi \\ &= N \cdot (N \bullet P) \cdot (1 - \cos\varphi) + P \cdot \cos\varphi + (N \times P) \cdot \sin\varphi \end{aligned}$$

Zerlegt in die einzelnen Vektorkomponenten:

$$\begin{pmatrix} p_x' \\ p_y' \\ p_z' \end{pmatrix} = \begin{pmatrix} n_x \cdot (n_x p_x + n_y p_y + n_z p_z) \cdot (1-\cos\varphi) + p_x \cdot \cos\varphi + (n_y p_z - n_z p_y) \cdot \sin\varphi \\ n_y \cdot (n_x p_x + n_y p_y + n_z p_z) \cdot (1-\cos\varphi) + p_y \cdot \cos\varphi + (n_z p_x - n_x p_z) \cdot \sin\varphi \\ n_z \cdot (n_x p_x + n_y p_y + n_z p_z) \cdot (1-\cos\varphi) + p_z \cdot \cos\varphi + (n_x p_y - n_y p_x) \cdot \sin\varphi \end{pmatrix}$$

Und geordnet:

$$= \begin{pmatrix} p_x \cdot (\cos\varphi + n_x^2 \cdot (1-\cos\varphi)) + & p_y \cdot (n_x n_y \cdot (1-\cos\varphi) - n_z \cdot \sin\varphi) + & p_z \cdot (n_x n_z \cdot (1-\cos\varphi) n_y \cdot \sin\varphi) \\ p_x \cdot (n_x n_y \cdot (1-\cos\varphi) + n_z \cdot \sin\varphi) + & p_y \cdot (\cos\varphi + n_y^2 \cdot (1-\cos\varphi)) + & p_z \cdot (n_y n_z \cdot (1-\cos\varphi) - n_x \cdot \sin\varphi) \\ p_x \cdot (n_x n_z \cdot (1-\cos\varphi) - n_y \cdot \sin\varphi) + & p_y \cdot (n_y n_z \cdot (1-\cos\varphi) + n_x \cdot \sin\varphi) + & p_z \cdot (\cos\varphi + n_z^2 \cdot (1-\cos\varphi)) \end{pmatrix}$$

$$= p_x \cdot \begin{pmatrix} \cos\varphi + n_x^2 \cdot (1-\cos\varphi) \\ n_x n_y \cdot (1-\cos\varphi) + n_z \cdot \sin\varphi \\ n_x n_z \cdot (1-\cos\varphi) - n_y \cdot \sin\varphi \end{pmatrix} + p_y \cdot \begin{pmatrix} n_x n_y \cdot (1-\cos\varphi) - n_z \cdot \sin\varphi \\ \cos\varphi + n_y^2 \cdot (1-\cos\varphi) \\ n_y n_z \cdot (1-\cos\varphi) + n_x \cdot \sin\varphi \end{pmatrix} + p_z \cdot \begin{pmatrix} n_x n_z \cdot (1-\cos\varphi) + n_y \cdot \sin\varphi \\ n_y n_z \cdot (1-\cos\varphi) - n_x \cdot \sin\varphi \\ \cos\varphi + n_z^2 \cdot (1-\cos\varphi) \end{pmatrix}$$

$$= \begin{pmatrix} \cos\varphi + n_x^2 \cdot (1-\cos\varphi) & n_x n_y \cdot (1-\cos\varphi) - n_z \cdot \sin\varphi & n_x n_z \cdot (1-\cos\varphi) + n_y \cdot \sin\varphi & 0 \\ n_x n_y \cdot (1-\cos\varphi) + n_z \cdot \sin\varphi & \cos\varphi + n_y^2 \cdot (1-\cos\varphi) & n_y n_z \cdot (1-\cos\varphi) - n_x \cdot \sin\varphi & 0 \\ n_x n_z \cdot (1-\cos\varphi) - n_y \cdot \sin\varphi & n_y n_z \cdot (1-\cos\varphi) + n_x \cdot \sin\varphi & \cos\varphi + n_z^2 \cdot (1-\cos\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

Die gesamte Transformationsmatrix lautet daher:

$$R(N,\varphi) = \begin{pmatrix} \cos\varphi + n_x^2 \cdot (1-\cos\varphi) & n_x n_y \cdot (1-\cos\varphi) - n_z \cdot \sin\varphi & n_x n_z \cdot (1-\cos\varphi) + n_y \cdot \sin\varphi & 0 \\ n_x n_y \cdot (1-\cos\varphi) + n_z \cdot \sin\varphi & \cos\varphi + n_y^2 \cdot (1-\cos\varphi) & n_y n_z \cdot (1-\cos\varphi) - n_x \cdot \sin\varphi & 0 \\ n_x n_z \cdot (1-\cos\varphi) - n_y \cdot \sin\varphi & n_y n_z \cdot (1-\cos\varphi) + n_x \cdot \sin\varphi & \cos\varphi + n_z^2 \cdot (1-\cos\varphi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Wendet man das Distributivgesetzt auf die Hauptdiagonale an, so erhält man das gleiche Ergebnis, wie es sich auch in *Computer Graphics: Principles and Practice* von James D. Foley [et al.] auf Seite 227 findet:

$$R(N,\varphi) = \begin{pmatrix} n_x^2 + \cos\varphi(1-n_x^2) & n_x n_y \cdot (1-\cos\varphi) - n_z \cdot \sin\varphi & n_x n_z \cdot (1-\cos\varphi) + n_y \cdot \sin\varphi & 0 \\ n_x n_y \cdot (1-\cos\varphi) + n_z \cdot \sin\varphi & n_y^2 + \cos\varphi(1-n_y^2) & n_y n_z \cdot (1-\cos\varphi) - n_x \cdot \sin\varphi & 0 \\ n_x n_z \cdot (1-\cos\varphi) - n_y \cdot \sin\varphi & n_y n_z \cdot (1-\cos\varphi) + n_x \cdot \sin\varphi & n_z^2 + \cos\varphi(1-n_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Leider kann man diese Matrix nicht auf den Vektor U verallgemeinern, da er nicht notwendigerweise die Länge 1 hat und muss so auf dessen normierte Version zurückgreifen.

Als Alternativen zu meinem Lösungsweg kommen noch Ideen auf Grundlage von *Direction of Flight* (DOF) in Frage, in meinen Augen ist dies aber aufwändiger herzuleiten (das Ergebnis hat aber die gleiche Form). Als kurze Andeutung für die durchzuführende Schritte hier die Gleichungen:

$$R(z \to DOF) = \begin{pmatrix} |y \times DOF| & |DOF \times (y \times DOF)| & |DOF| & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Am Anfang ist aber genau die inverse Rotation durchzuführen. Zusammen mit der Rotation um die z-Achse erhält man:

$$\begin{aligned} M &= R(z \to DOF) \cdot R_z(\varphi) \cdot R(DOF \to z) \\ &= R(DOF_2) \cdot R_z(\varphi) \cdot R(DOF) \end{aligned}$$

$$DOF = \begin{pmatrix} u_x \\ u_y \\ u_z \end{pmatrix}$$

$$DOF_2 = \begin{pmatrix} -u_x \\ -u_y \\ u_z \end{pmatrix}$$

$$M = \begin{pmatrix} |y \times DOF_2| & |DOF_2 \times (y \times DOF_2)| & |DOF_2| & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos\varphi & -\sin\varphi & 0 & 0 \\ \sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} |y \times DOF| & |DOF \times (y \times DOF)| & |DOF| & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Eine kleine Vereinfachung erreicht man mit der Überlegung, dass die Matrizen orthonormal sind, d.h. dass die inverse Matrix gleich der transponierten ist.

$$R(DOF)^{-1} = R(DOF)^T$$

Recht elegant ist die Rotation auf Basis von Quaternionen, allerdings erhält man nur auf Umwegen die entsprechende Transformationsmatrix. Da ich hier nicht den kompletten Beweis aufführen will, gebe ich nur die Rotationsmatrix an (die Achse selbst ist wieder normiert):

$$x_4 = n_x \cdot \sin\frac{\varphi}{2}$$

$$y_4 = n_y \cdot \sin\frac{\varphi}{2}$$

$$z_4 = n_z \cdot \sin\frac{\varphi}{2}$$

$$w_4 = \cos\frac{\varphi}{2}$$

$$Q = \begin{pmatrix} w_4^2 + x_4^2 - y_4^2 - z_4^2 & 2x_4y_4 + 2w_4z_4 & 2x_4z_4 - 2w_4y_4 & 0 \\ 2x_4y_4 - 2w_4z_4 & w_4^2 - x_4^2 + y_4^2 + z_4^2 & 2y_4z_4 + 2w_4x_4 & 0 \\ 2x_4z_4 + 2w_4y_4 & 2y_4z_4 - 2w_4x_4 & w_4^2 - x_4^2 - y_4^2 + z_4^2 & 0 \\ 0 & 0 & 0 & w_4^2 + x_4^2 + y_4^2 + z_4^2 \end{pmatrix}$$

Aufgabe 17

Als Grundvoraussetzung fordere ich erneut, dass die Normale der Spiegelebene normiert ist.
Ich zerlege die Spiegelung dann in drei Schritte:

1. Verschiebung von P_E in die xy-Ebene
2. Rotation der Spiegelebene parallel zur xy-Ebene (durch Rotation der Normalen parallel zur z-Achse)
3. Spiegelung an der xy-Ebene
4. Rotation von der xy-Ebene auf die Spiegelebene zurück (erneut Rotation der Normalen)
5. Verschiebung von P_E aus der xy-Ebene heraus (Umkehrung von 1.)

Für die Schritte 1 und 5 nutze ich die in Aufgabe 16 gewonnene Matrix und wende sie auf die normierte Normale der Ebene an, die ich auf die z-Achse transformiere. Die dazu notwendige Achse A und der entsprechende Winkel φ sind:

$$A = \frac{z \times N}{|z \times N|}$$
$$= \begin{pmatrix} -n_y \\ n_x \\ 0 \end{pmatrix} \cdot \sqrt{n_x^2 + n_y^2}$$
$$z \bullet N = |z| \cdot |N| \cdot \cos\varphi$$
$$z \bullet N = \cos\varphi$$
$$n_z = \cos\varphi$$
$$\varphi = \arccos n_z$$

Die einzelnen Matrizen sind:

1. Verschiebung von P_E in die xy-Ebene

$$T(-P_E) = \begin{pmatrix} 0 & 0 & 0 & -p_x \\ 0 & 0 & 0 & -p_y \\ 0 & 0 & 0 & -p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

2. Rotation der Spiegelebene parallel zur xy-Ebene (durch Rotation der Normalen parallel zur z-Achse)

$$R(A,\varphi) = \begin{pmatrix} n_x^2 + \cos\varphi(1-n_x^2) & -n_x n_y \cdot (1-\cos\varphi) & -n_y \cdot \sin\varphi & 0 \\ -n_x n_y \cdot (1-\cos\varphi) & n_y^2 + \cos\varphi(1-n_y^2) & -n_x \cdot \sin\varphi & 0 \\ n_y \cdot \sin\varphi & n_x \cdot \sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Spiegelung an der xy-Ebene

$$S(-z)=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & -1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

4. Rotation von der xy-Ebene auf die Spiegelebene zurück (erneut Rotation der Normalen)

$$R(A,-\varphi)=\begin{pmatrix}n_x^2+\cos\varphi(1-n_x^2) & -n_x n_y\cdot(1-\cos\varphi) & n_y\cdot\sin\varphi & 0\\ -n_x n_y\cdot(1-\cos\varphi) & n_y^2+\cos\varphi(1-n_y^2) & n_x\cdot\sin\varphi & 0\\ -n_y\cdot\sin\varphi & -n_x\cdot\sin\varphi & \cos\varphi & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$$

5. Verschiebung von P_E aus der xy-Ebene heraus (Umkehrung von 1.)

$$T(P_E)=\begin{pmatrix}0 & 0 & 0 & p_x\\ 0 & 0 & 0 & p_y\\ 0 & 0 & 0 & p_z\\ 0 & 0 & 0 & 1\end{pmatrix}$$

Die Gesamt-Transformationsmatrix ergibt sich durch Multiplikation von:

$$M = T(P_E)\cdot R(A,-\varphi)\cdot S(-z)\cdot R(A,\varphi)\cdot T(-P_E)$$

Leider ist diese Matrix nur sehr aufwändig zu berechnen, viel einfacher geht es, die altbekannten Gesetze der Vektorarithmetik von Punkten und Ebenen zu benutzen. Diese definieren den Abstand Punkt-Ebene bei normierter Normalen der Ebene als (entstanden aus der impliziten Form):

$$d(E,X) = N \bullet (D - P_E)$$

Zu erwähnen ist, dass das Vorzeichen dabei sehr wichtig ist und man daher nicht unbedingt das Wort „Abstand“ verwenden sollte. Viel mehr ist d ein Skalar, der angibt, wie oft der Normalenvektor N der Ebene von dem Punkt zu subtrahieren ist, damit der Punkt in der Ebene D liegt.
Die Spiegelung des Punktes X an der Ebene D genügt dann der einfachen Gleichung:

$$D' = D - 2 \cdot d(E,D) \cdot N_E$$

Teilt man die Gleichung auf die einzelnen Komponenten der Vektoren auf, so ergibt sich:

$$\begin{pmatrix} d_x' \\ d_y' \\ d_z' \end{pmatrix} = \begin{pmatrix} d_x - 2 \cdot n_x \cdot (n_x \cdot (d_x - p_x) + n_y \cdot (d_y - p_y) + n_z \cdot (d_z - p_z)) \\ d_y - 2 \cdot n_y \cdot (n_x \cdot (d_x - p_x) + n_y \cdot (d_y - p_y) + n_z \cdot (d_z - p_z)) \\ d_z - 2 \cdot n_z \cdot (n_x \cdot (d_x - p_x) + n_y \cdot (d_y - p_y) + n_z \cdot (d_z - p_z)) \end{pmatrix}$$

Durch Umordnen bekommt man:

$$\begin{pmatrix} d_x' \\ d_y' \\ d_z' \end{pmatrix} = \begin{pmatrix} d_x \cdot (1 - 2n_x^2) + d_y \cdot (-2n_x n_y) + d_z \cdot (-2n_x n_z) + 2n_x \cdot (n_x p_x + n_y p_y + n_z p_z) \\ d_x \cdot (-2n_x n_y) + d_y \cdot (1 - 2n_y^2) + d_z \cdot (-2n_y n_z) + 2n_y \cdot (n_x p_x + n_y p_y + n_z p_z) \\ d_x \cdot (-2n_x n_z) + d_y \cdot (-2n_y n_z) + d_z \cdot (1 - 2n_z^2) + 2n_z \cdot (n_x p_x + n_y p_y + n_z p_z) \end{pmatrix}$$

$$= \begin{pmatrix} 1 - 2n_x^2 & -2n_x n_y & -2n_x n_z & 2n_x \cdot (n_x p_x + n_y p_y + n_z p_z) \\ -2n_x n_y & 1 - 2n_y^2 & -2n_y n_z & 2n_y \cdot (n_x p_x + n_y p_y + n_z p_z) \\ -2n_x n_z & -2n_y n_z & 1 - 2n_z^2 & 2n_z \cdot (n_x p_x + n_y p_y + n_z p_z) \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Aufgabe 18

Eine altbekannte Weisheit sagt, dass Menschen im hohen Alter in ihrem Verhalten sich immer mehr wie Kleinkinder verhalten. Letztere sind sehr glücklich, wenn sie zur Auslebung ihres Bewegungsdranges einen Tretroller geschenkt bekommen. Man kann sehr gut beobachten, dass heutzutage 30-Jährige Businessmen einem starken geistigen Verfall aufgrund des Arbeitsstresses unterliegen, da auch sie mit derart unmöglichen Gefährten sich fortbewegen. Ich kreide diesen Missstand dem Turbo-Kapitalismus an, der in geradezu okkulter Weise die Emotionalität des Menschen auf Steinzeitniveau herabsetzt. Um hier Abhilfe zu schaffen, rufe ich zur Gründung des

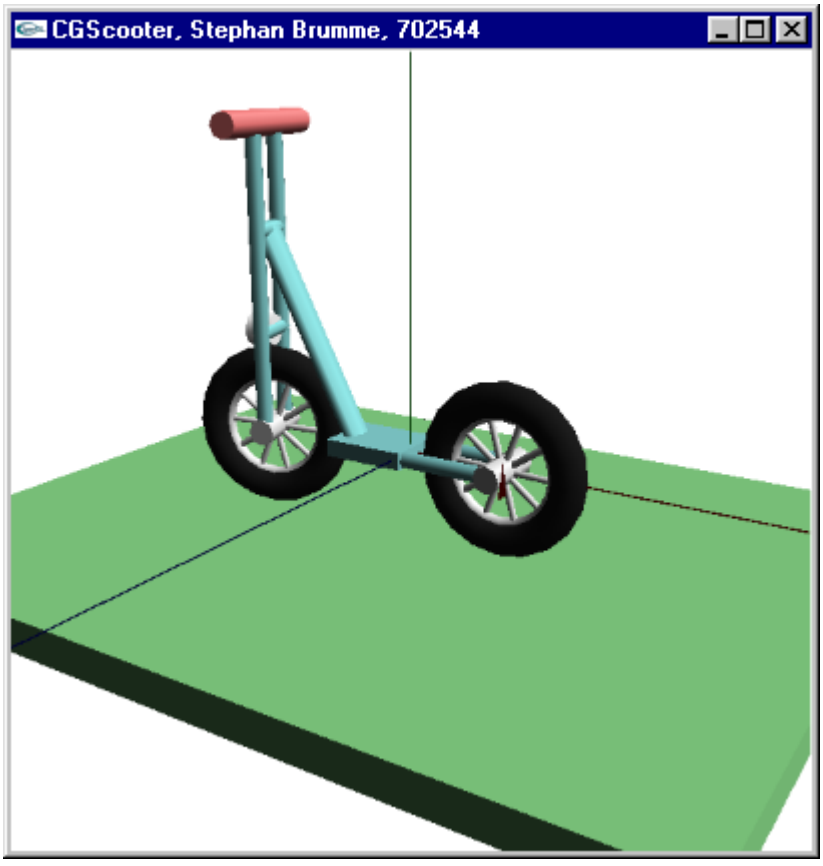
Vereins zur Rettung des gesunden Menschenverstandes e.V.

auf. Ich bitte insbesondere die Mitarbeiter des Lehrstuhls für Computergraphische Systeme um ihre tatkräftige Unterstützung.

Mein erster Beitrag besteht darin, dass ich - quasi als Ersatzdroge - einen virtuellen Tretroller kostenlos der Computerwelt zur Verfügung stelle. Dies soll den Ausstieg von Süchtigen und Abhängigen aus diesem Teufelskreis dramatisch vereinfachen und so aus unserer Erde einen glücklicheren Planeten machen. Da der Tretroller überwiegend für den weiblichen Teil der Bevölkerung gedacht ist (wo die Sucht oft exzessivste Formen annimmt), taufe ich ihn auf den Namen:

Rock `n` Roll, Baby !

Bei aller jugendlichen Ausgelassenheit, die sich nun eventuell in spontanen Freudentänzen und wilden Orgien ausdrückt, muss ich dennoch auf ein paar technische Features hinweisen, die jeden eine Etage höher, also in den 8. Himmel, katapultieren werden: Ausrüstung mit StVO- und StVZO-konformer Lichtinstallation, vorne als Lampe, hinten als Katzenauge. Um die Batterien nicht vorzeitig zu entladen, ist die Beleuchtung im Programm nicht anschaltbar, sondern es muss das Sonnenlicht, das vom Punkt $(1,1,1)^T$ ausgeht, genügen.



Sicherlich gibt es viele, sehr viele, oder sogar unwahrscheinlich viele Leser, die jetzt mehr verlangen. Und dieser Wissensdurst wird sofort gestillt:

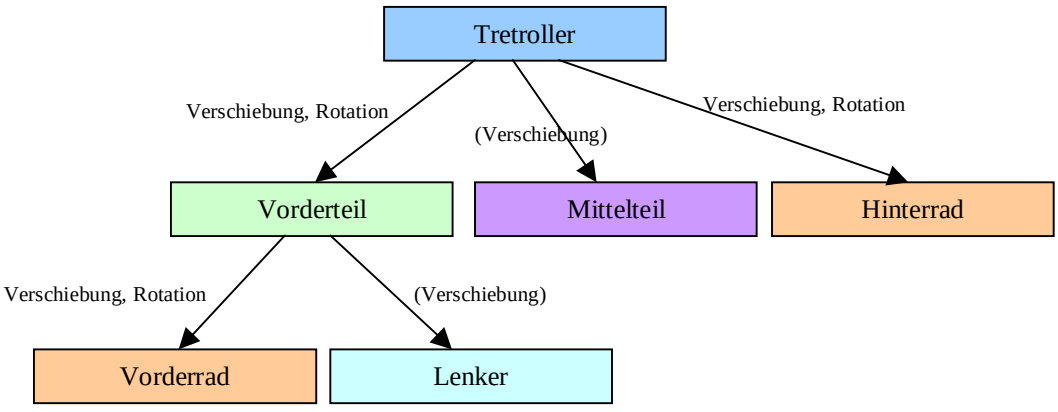
Feature	Details
Rahmenmaterial	Chrom-Molybdän
Lenkradius	45°
Höchstgeschwindigkeit	Warp 3 (in 29,2 Sekunden)
max. Zuladung	60 kg im nüchternen Zustand
Lieblingsfarbe	Mitternachts-Lila
Fun-Faktor	über 100% !!!

Um den geneigten Leser nicht länger auf die Folter zu spannen, erkläre ich zuerst die Bedienung, im Anschluss folgen dann die technischen Details.

Taste	Aktion
`l`	Drahtgittermodell
`f`	Oberflächenmodell
`c`	Entfernung verdeckter (vom Betrachter wegweisender) Flächen
`n`	Darstellung aller Flächen
`a`	Darstellung der Koordinatenachsen ein/aus
Leertaste	Rotation des Tretrollers um die y-Achse ein/aus
`r`	Rotation der Räder und des Lenkers ein/aus
ESC bzw. `q`	Programm beenden

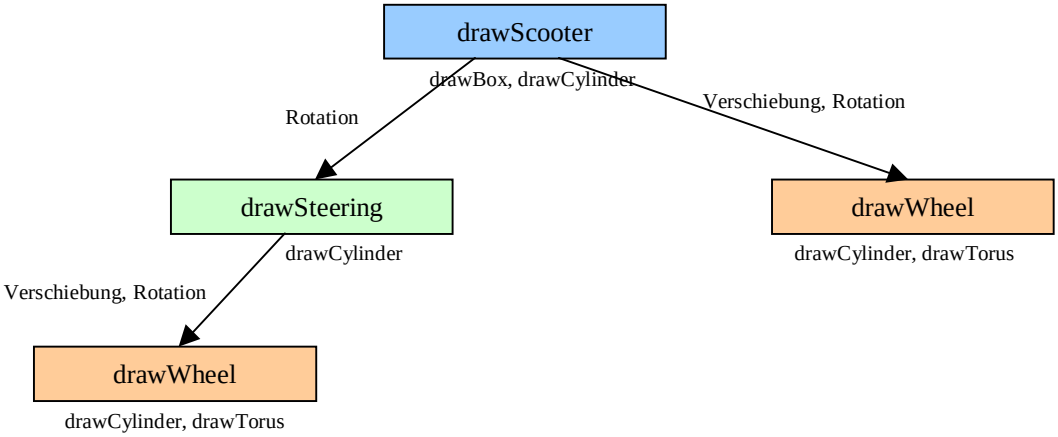
Standardmäßig sind die rot markierten Einstellungen aktiv. Um den Roller besser betrachten zu können, habe ich den Standort der Kamera leicht verschoben, er liegt jetzt bei $(0,1,3)^T$. Mit gedrückter linker Maustaste kann das Modell um die y-Achse rotiert werden.

Eine mögliche hierarchische Darstellung des Tretrollers, die ich auch dann in OpenGL umgesetzt habe, nutzt aus, dass das Vorderrad die gleiche Geometrie aufweist wie das Hinterrad und daher auch in einer separaten Funktion modelliert werden. Beide sollen die Fähigkeit besitzen, auf Tastendruck derart um ihre eigene z-Achse zu rotieren, dass eine Vorwärtsbewegung simuliert wird. Das Vorderrad steht in einer engen Beziehung zum Lenker: dieser kann sich um seine eigene y-Achse drehen und muss daher auch das verknüpfte Rad entsprechend mitrotieren. Unabhängig von all diesen Bewegung ist das starre Mittelteil, das aus einem Trittbrett und verbindenden Streben zu Lenker bzw. Hinterrad besteht. Der gesamte Roller soll zusätzlich im Raum um die y-Achse drehbar sein.



Die geklammerten Transformationen sind nicht explizit umgesetzt worden, sondern wurden in der übergeordneten Methode durch entsprechende Definition der Grafik-Primitive erreicht. Das heißt also, dass z.B. der Lenker in Wahrheit schon beim Vorderteil konstruiert wird.

Um noch einmal eine Zuordnung zum Quelltext herstellen zu können, folgt die obige Grafik noch einmal, diesmal aber mit den entsprechenden Methodennamen und den darin verwendeten Primitiven:



Die Steuerung über die Tastaturkommandos verändert 3 boolesche Variablen, die als Attribute der Klasse `CGScooter` definiert sind:

Variable	Bedeutung
`axis_`	Koordinatenachsen anzeigen
`rotate_`	Räder bewegen und Lenker drehen
`run_`	gesamten Tretroller drehen

Der Einfachheit halber sind alle 3 beim Programmstart auf `true` gesetzt. Die Rotationswinkel werden als `static`-Variablen gespeichert und unterscheiden sich für die Räder und den Lenker.

Wenn ich nichts im Wettbewerb um den schönsten Tretroller gewinne,
dann schmeiße ich aus Protest den überlagerten Joghurtbecher aus dem Fenster,
der in meinem Kühlschrank liegt.

Ich wohne im Erdgeschoss !

Quellcode

Ausnahmsweise führe ich hier im Anschluss an die eigentliche „Arbeitsklasse“ `CGScooter` wieder die Rahmenklasse `CGApplication` auf, da ich ein paar kleine Änderungen im Maus-Handling durchführte, die die interaktive Rotation des Modell bei gedrückter Maustaste zulassen.

`"cgscooter.h"`

```
//
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 6
//

#ifndef CG_SCOOTER_H
#define CG_SCOOTER_H

#include "cgapplication.h"

class CGScooter : public CGApplication {
public:
    CGScooter();
    virtual ~CGScooter();

    // Ueberschreibe alle diese Ereignisse:
    virtual void onInit();
    virtual void onDraw();
    virtual void onIdle();
    virtual void onKey(unsigned char key);
    virtual void onSize(unsigned int newWidth, unsigned int newHeight);

    // value Methode
    unsigned char value(int x, int z) const;

private:
    void drawSteering();
    void drawScooter();
    void drawBox();
    void drawSphere();
    void drawTorus(double innerRadius, double outerRadius);
    void drawCylinder(bool caps = false);
    void drawWheel();

    bool rotate_;
    bool run_;
    bool axis_;
};

#endif // CG_SCOOTER_H
```

"cgscooter.cpp"

```
//
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 6
//

#include "cgscooter.h"

CGScooter::CGScooter() {
    rotate_ = true;
    run_ = true;//false;
    axis_ = true;//false;
}

CGScooter::~CGScooter() {
}
void CGScooter::drawBox() {
    glutSolidCube(1);
}

void CGScooter::drawSphere() {
    static GLUquadricObj* q = gluNewQuadric();
    gluSphere(q,1.0,10,10);
}

void CGScooter::drawTorus(double innerRadius, double outerRadius) {
    static GLUquadricObj* q = gluNewQuadric();
    glutSolidTorus(outerRadius-innerRadius, outerRadius, 18, 20);
}

void CGScooter::drawCylinder(bool caps) {
    static GLUquadricObj* q = gluNewQuadric();
    static GLUquadricObj* d = gluNewQuadric();

    gluCylinder(q,0.5,0.5,1,8,1);
    if (caps) {
        glPushMatrix();
        gluDisk(d,0,0.5,8,1);
        glTranslated(0,0,1);
        gluDisk(d,0,0.5,8,1);
        glPopMatrix();
    }
}

void CGScooter::drawWheel() {
    // ein Rad besteht aus Nabe, Speichen, Felge und Bereifung
    // Durchmesser des Rades: 1
    // Breite (Reifen): 0.25
    // Ausrichtung in der xy-Ebene, zentriert im Ursprung
    glPushMatrix();

    // Nabe (Durchmesser 0.4, Breite 0.5+0.5)
    glPushMatrix();
    glColor3f(0.9, 0.9, 0.9);

    glScalef(0.4, 0.4, 1);
    glTranslatef(0, 0, -0.5);
    drawCylinder(true);

    glPopMatrix();

    // Speichen (jeweils um den Mittelpunkt)
    glPushMatrix();
    glColor3f(0.9, 0.9, 0.9);

    static double angle = 0;
    if (rotate_)
        angle += 1.0;
    if (angle > 360)
        angle -= 360;
    glRotatef(angle, 0, 0, 1);
```

```
    const int nSpeichen = 10;
    for (int i=0; i<nSpeichen; i++)
    {
        glPushMatrix();
        glRotatef(i*360/nSpeichen, 0, 0, 1);
        // in die xy-Ebene bringen
        glRotatef(90, 1, 0, 0);
        // Durchmesser verkleinern
        glScalef(0.1, 0.1, 0.75);

        drawCylinder(false);
        glPopMatrix();
    }
    glPopMatrix();

    // Felge
    drawTorus(0.7, 0.8);

    // Reifen
    glColor3f(0.1, 0.1, 0.1);
    drawTorus(0.75, 1);

    glPopMatrix();
}

void CGScooter::drawSteering()
{
    // Vorderrad ist im Ursprung zentriert
    // Gesamthöhe der Lenkerkonstruktion: 1

    glColor3f(0.5, 0.8, 0.8);

    // linke Lenkerstange
    glPushMatrix();
    glTranslatef(0, 0, 0.07);
    glScalef(0.05, 0.9, 0.05);
    // auf y-Achse
    glRotatef(270, 1, 0, 0);
    drawCylinder(true);
    glPopMatrix();

    // rechte Lenkerstange
    glPushMatrix();
    glTranslatef(0, 0, -0.07);
    glScalef(0.05, 0.9, 0.05);
    // auf y-Achse
    glRotatef(270, 1, 0, 0);
    drawCylinder(false);
    glPopMatrix();

    // obere Querstange
    glPushMatrix();
    glColor3f(1, 0.5, 0.5);
    // auf Lenkerstangen auflegen
    glTranslatef(0, 0.9, 0);
    // etwas schmaler
    glScalef(0.075, 0.075, 0.5);
    // Zylinder in Mittelpunkt zentrieren
    glTranslatef(0, 0, -0.5);
    drawCylinder(true);
    glPopMatrix();

    // mittlere Querstange
    glPushMatrix();
    glColor3f(0.5, 0.8, 0.8);
    // etwa halbe Höhe des Lenkers
    glTranslatef(0, 0.6, 0);
    // etwas schmaler
    glScalef(0.05, 0.05, 0.15);
    // Zylinder in Mittelpunkt zentrieren
    glTranslatef(0, 0, -0.5);
    drawCylinder(false);
    glPopMatrix();
```

```
    // untere Querstange
    glPushMatrix();
    // knapp über dem Vorderrad
    glTranslatef(0, 0.3, 0);
    // etwas schmaler
    glScalef(0.05, 0.05, 0.15);
    // Zylinder im Mittelpunkt zentrieren
    glTranslatef(0, 0, -0.5);
    glColor3f(0.5, 0.8, 0.8);
    drawCylinder(false);
    glPopMatrix();

    // Lampe
    glPushMatrix();
    // an der unteren Querstange
    glTranslatef(-0.025, 0.3, 0);
    glRotatef(90, 0, 1, 0);
    glScalef(0.1, 0.1, 0.075);
    // Zylinder im Mittelpunkt zentrieren
    glTranslatef(0, 0, -0.5);
    glColor3f(1,1,1);
    drawCylinder(true);
    glPopMatrix();

    // Vorderrad
    glPushMatrix();
    glScalef(0.2, 0.2, 0.2);
    drawWheel();
    glPopMatrix();
}

void CGScooter::drawScooter()
{
    glPushMatrix();
    // rotiere Lenker, wenn nötig
    static float angle = 0.0;
    static float angle_inc = 0.5;
    if (rotate_)
    {
        angle += angle_inc;

        if (abs(angle) > 45)
            angle_inc = -angle_inc;
    }
    glRotatef(angle, 0, 1, 0);
    // zeichne Lenker
    drawSteering();
    glPopMatrix();

    // Hinterrad
    glPushMatrix();
    glTranslatef(0.8, 0, 0);
    glScalef(0.2, 0.2, 0.2);
    drawWheel();
    glPopMatrix();

    // Trittbrett
    glColor3f(0.5, 0.8, 0.8);
    glPushMatrix();
    glTranslatef(0.4, 0, 0);
    glScalef(0.25, 0.05, 0.2);
    // eigentliches Brett
    drawBox();
    glPopMatrix();

    // Streben zur Hinterachse
    glPushMatrix();
    glTranslatef(0.5, 0, -0.07);
    glScalef(0.25, 0.05, 0.05);
    // auf x-Achse
    glRotatef(90, 0, 1, 0);
    drawCylinder(true);
    glPopMatrix();

    glPushMatrix();
```

```
    glTranslatef(0.5, 0, +0.07);
    glScalef(0.3, 0.05, 0.05);
    // auf x-Achse
    glRotatef(90, 0, 1, 0);
    drawCylinder(true);
    glPopMatrix();

    // Strebe zum Lenker
    glPushMatrix();
    glTranslatef(0, 0.6, 0);
    // schräg zum Lenker hoch
    glRotatef(62, 0, 0, -1);
    // auf x-Achse
    glRotatef(90, 0, 1, 0);
    glScalef(0.1, 0.05, 0.7);
    drawCylinder(true);
    glPopMatrix();

    // rotes Katzenlicht am Hinterrad
    glPushMatrix();
    glColor3f(1, 0, 0);
    glBegin(GL_TRIANGLES);
    glVertex3f(0.84, -0.05, 0.1);
    glVertex3f(0.84, -0.05, 0.05);
    glVertex3f(0.84,  0.05,0.075);
    glEnd();
    glPopMatrix();
}

void CGScooter::onInit() {
    // OpenGL Lichtquelle
    static GLfloat light_diffuse[] = {1.0, 1.0, 1.0, 1.0};  /* diffuse light. */
    static GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};  /* Infinite light location. */
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    // automatische Normalisierung
    glEnable(GL_NORMALIZE);

    glEnable(GL_COLOR_MATERIAL);
    glColorMaterial(GL_FRONT_AND_BACK,GL_AMBIENT_AND_DIFFUSE);
    // Tiefen Test aktivieren
    glEnable(GL_DEPTH_TEST);

    // Smooth Schattierung aktivieren
    glShadeModel(GL_SMOOTH);

    // Projection
    glMatrixMode(GL_PROJECTION);
    gluPerspective(60.0,1.0,0.2,20.0);

    // LookAt
    glMatrixMode(GL_MODELVIEW);
//    gluLookAt(0.0, 0.0, 4.0,  // from (0,0,4)
    gluLookAt(0.0, 1.0, 3.0,  // from (1,1,3)
              0.0, 0.0, 0.0,  // to (0,0,0)
              0.0, 1.0, 0.0); // up

    glClearColor(1,1,1,1);
}

void CGScooter::onSize(unsigned int newWidth,unsigned int newHeight) {
    if((newWidth > 0) && (newHeight > 0))
    {
        // Passe den OpenGL-Viewport an die neue Fenstergroesse an:
        glViewport(0, 0, newWidth - 1, newHeight - 1);

        // Passe die OpenGL-Projektionsmatrix an die neue
        // Fenstergroesse an:
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluPerspective(40.0,float(newWidth)/float(newHeight),1.0, 10.0);
```

```
        // Schalte zurueck auf die Modelview-Matrix
        glMatrixMode(GL_MODELVIEW);
    }
}

void CGScooter::onKey(unsigned char key) {
    switch (key) {
    case 27 :
    case 'q': exit(0);
              break;

    case 'l': glPolygonMode(GL_FRONT_AND_BACK,GL_LINE); break;
    case 'f': glPolygonMode(GL_FRONT_AND_BACK,GL_FILL); break;

    case 'c': glEnable(GL_CULL_FACE); break;
    case 'n': glDisable(GL_CULL_FACE); break;

    case ' ': run_ = !run_; break;
    case 'a': axis_ = !axis_; break;

    case 'r': rotate_ = !rotate_; break;
    }

    glutPostRedisplay();
}

void CGScooter::onIdle() {
    if (run_)
        glRotatef(1,0,1,0);

    glutPostRedisplay();
}

void CGScooter::onDraw() {
    // Loesche den Farb- und Tiefenspeicher
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // draw coordinate system
    if (axis_) {
        glBegin(GL_LINES);
        // x
        glColor3d(1.0,0,0);
        glVertex3d(0.0,0,0);
        glVertex3d(3.0,0,0);

        // y
        glColor3d(0,1,0);
        glVertex3d(0,0,0);
        glVertex3d(0,3,0);

        // z
        glColor3d(0,0,1);
        glVertex3d(0,0,0);
        glVertex3d(0,0,3);
        glEnd();
    }

    // Matrixmodus setzen
    glMatrixMode(GL_MODELVIEW);

    // hier den Scooter zeichnen
    glPushMatrix();
    glTranslatef(-0.5, 0, 0);
    drawScooter();
    glPopMatrix();

    // Plattform
    glPushMatrix();
    glColor3f(0.5, 0.8, 0.5);
    glTranslatef(0, -0.3, 0);
    glScalef(2.5, 0.1, 2);
    drawBox();
    glPopMatrix();

    // Nicht vergessen! Front- und Back-Buffer tauschen:
```

```
    swapBuffers();
}

// Hauptprogramm
int main(int argc, char* argv[]) {
    // Erzeuge eine Instanz der Beispiel-Anwendung:
    CGScooter sample;

    // Starte die Beispiel-Anwendung:
    sample.start("CGScooter, Stephan Brumme, 702544");
    return(0);
}
```

"cgapplication.h"

```
//
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 6
//

#ifndef CG_APPLICATION_H
#define CG_APPLICATION_H

#include <GL/glut.h>

// einige haeufig verwendete Header-Dateien
#include <assert.h>
#include <iostream.h>
#include <math.h>

class CGApplication {
public:

    // Die Klasse "CGApplication" stellt ein Rahmenprogramm bereit,
    // mit dem sie ihre Aufgaben loesen koennen. Dazu muessen sie
    // eine Klasse hiervon ableiten, und eine oder mehrere der
    // weiter unten deklarierten on*()-Methoden ueberschreiben.
    // Diese Klasse ist ein sogenanntes "Singleton", d.h.: Es kann
    // immer nur eine einzige Instanz von dieser (oder einer
    // abgeleiteten) Klasse erzeugt werden!
    CGApplication();
    virtual ~CGApplication();

    // Startet die Anwendung und muss im Hauptprogramm aufgerufen werden.
    void start(const char* windowTitle = "CGApplication",
               bool doubleBuffering = true,
               unsigned long windowWidth = 400, unsigned long windowHeight = 400);

    // Diese Methode wird nur einmal beim Start der Anwendung aufgerufen
    // und dient zur Initialisierung von OpenGL (hier kann z.B. die
    // Hintergrundfarbe fuer das Anwendungsfenster definiert werden). Sie
    // kann von einer abgeleiteten Klasse ueberschrieben werden.
    virtual void onInit();

    // Jedesmal wenn der Fensterinhalt neu gezeichnet werden muss,
    // wird diese Methode aufgerufen. Sie muss von einer abgeleiteten
    // Klasse ueberschrieben werden!
    virtual void onDraw() = 0;

    // Jedesmal wenn sich die Fenstergroesse geaendert hat, wird diese
    // Methode aufgerufen (die Methode onDraw() wird im Anschluss
    // automatisch aufgerufen).
    virtual void onSize(unsigned int newWidth, unsigned int newHeight) = 0;

    // Spezifiziert die beiden Konstanten "LeftMouseButton" und
    // "RightMouseButton".
    enum MouseButton { LeftMouseButton, RightMouseButton };

    // Jedesmal wenn eine Maustaste im Fenster gedrueckt wird, wird diese
    // Methode mit der entsprechenden Taste ("LeftMouseButton" oder
    // "RightMouseButton") und den zugehoerigen Koordinaten (x und y)
    // aufgerufen. Die Koordinaten werden als Pixel uebergeben, wobei der
    // Koordinaten-Ursprung die linke untere Fensterecke ist. Diese
    // Methode kann von einer abgeleiteten Klasse ueberschrieben werden.
    virtual void onButton(MouseButton button, int x, int y);

    // Jedesmal wenn die Maus mit gedrueckter Maustaste ueber das Fenster
    // bewegt wird, wird diese Methode mit den entsprechenden Koordinaten
    // aufgerufen (siehe auch Methode onButton()). Diese Methode kann von
    // einer abgeleiteten Klasse ueberschrieben werden.
    virtual void onMove(int x, int y);

    // Wird beim Ziehen der Maus aufgerufen und enthält die prozentuale
    // relative Änderung
    virtual void onDrag(double dx, double dy);
```

```
    // Jedesmal wenn eine Taste gedrueckt wird, wird diese Methode
    // aufgerufen.
    virtual void onKey(unsigned char key);


    // Wird aufgerufen, wenn nichts zu tun ist.
    virtual void onIdle();

    // Diese Methode sollte nach Beendigung aller Zeichenoperationen fuer
    // ein Bild (normaler Weise am Ende der Methode onDraw()) aufgerufen
    // werden, um den Front- mit dem Back-Buffer zu tauschen! Wird dieser
    // Aufruf ausgelassen, so werden die Zeichenoperationen nicht sichtbar!!!
    void swapBuffers();

private:
    // Verbiete den Copy-Konstruktor und den Zuweisungsoperator, da
    // diese Klasse ein Singleton ist und somit nicht kopiert werden
    // darf!
    CGApplication(const CGApplication&);
    CGApplication& operator=(const CGApplication&);
};

#endif // CG_APPLICATION_H
```

"cgapplication.cpp"

```
//
// Computergraphik I
// Prof. Dr. Juergen Doellner
// Sommersemester 2001
//
// Rahmenprogramm fuer Aufgabenzettel 6
//

#include "cgapplication.h"

// Diese Variable enthaelt einen Zeiger auf die aktive Anwendung.
static CGApplication* activeApplication_ = 0;

// Drag'n'Drop
static int lastDragDropX_ = 0;
static int lastDragDropY_ = 0;


CGApplication::CGApplication() {
    assert(!activeApplication_); // Es ist nur eine aktive Anwendung erlaubt!!!
    activeApplication_ = this;
}

CGApplication::~CGApplication() {
    activeApplication_ = 0;
}

// Auf diese Ereignisse wird default-maessig nicht reagiert:
void CGApplication::onInit() { }
void CGApplication::onButton(MouseButton button, int x, int y) { }
void CGApplication::onMove(int x, int y) { }
void CGApplication::onDrag(double dx, double dy) { }
void CGApplication::onKey(unsigned char key) { }
void CGApplication::onIdle() { }

// glut-Callback fuer den Zeichenaufruf
static void displayFunc() {
    assert(activeApplication_);
    activeApplication_->onDraw();
}

// glut-Callback fuer die Aenderung der Fenstergroesse
static void reshapeFunc(int newWidth, int newHeight) {
    assert(activeApplication_);
    activeApplication_->onSize(newWidth, newHeight);
}

// glut-Callback fuer das Druecken einer Maustaste
static void mouseFunc(int button, int state, int x, int y) {
    assert(activeApplication_);

    if(state == GLUT_DOWN) {
        y = glutGet(GLenum(GLUT_WINDOW_HEIGHT)) - y; // Koordinaten-Ursprung: links unten!!!
        if(button == GLUT_LEFT_BUTTON) {
            activeApplication_->onButton(CGApplication::LeftMouseButton, x, y);
        } else {
            activeApplication_->onButton(CGApplication::RightMouseButton, x, y);
        }

        lastDragDropX_ = x;
        lastDragDropY_ = y;
    }
}

// glut-Callback fuer das Ziehen bei gedrueckter Maustaste
static void motionFunc(int x, int y) {
    assert(activeApplication_);

    y = glutGet(GLenum(GLUT_WINDOW_HEIGHT)) - y; // Koordinaten-Ursprung: links unten!!!

    double distanceX = x - lastDragDropX_;
    double distanceY = y - lastDragDropY_;
```

```
    activeApplication_->onMove(x, y);

    double relativeX = distanceX / GLenum(GLUT_WINDOW_WIDTH);
    double relativeY = distanceY / GLenum(GLUT_WINDOW_HEIGHT);
    cout << "Dragging the line: " << relativeX << "/" << relativeY << endl;

    activeApplication_->onDrag(relativeX, relativeY);

    lastDragDropX_ = x;
    lastDragDropY_ = y;
}

// glut-Callback fuer einen Tastendruck
static void keyboardFunc(unsigned char key, int x, int y) {
    assert(activeApplication_);
    activeApplication_->onKey(key);
}


// glut-Callback fuer einen Tastendruck
static void idleFunc() {
    assert(activeApplication_);
    activeApplication_->onIdle();
}
void CGApplication::start(const char* windowTitle, bool doubleBuffering,
                          unsigned long width, unsigned long height) {
    // Fordert ein OpenGL-Fenster im RGB-Format mit oder ohne Double-Buffering an:
    glutInitDisplayMode(GLUT_RGB | (doubleBuffering ? GLUT_DOUBLE : GLUT_SINGLE));

    // Legt die Fenstergroesse fest:
    glutInitWindowSize(width, height);

    // Erzeugt das OpenGL-Fenster mit dem entsprechenden Namen:
    glutCreateWindow(windowTitle);

    // Registriere die oben definierten Callbacks fuer die
    // entsprechenden glut-Ereignisse:
    glutDisplayFunc(displayFunc);
    glutReshapeFunc(reshapeFunc);
    glutMouseFunc(mouseFunc);
    glutMotionFunc(motionFunc);
    glutKeyboardFunc(keyboardFunc);
    glutIdleFunc(idleFunc);

    // Zeige das OpenGL-Fenster auf dem Bildschirm an
    glutShowWindow();

    // Rufe die Methode zur Initialisierung von OpenGL auf:
    onInit();

    // Starte die Verarbeitung der glut-Ereignisse:
    glutMainLoop();
}

void CGApplication::swapBuffers() {
    glutSwapBuffers();
}
```