Prof. G. Zachmann
D. Mohr

Universität Bremen
Fachbereich 3
Arbeitsgruppe CGVR
18. November 2013

Wintersemester 2013/14

Übungen zu Computergraphik - Blatt 6

Abgabe am 25. 11. 2013

Aufgabe 1 (Transformationen, *1+3+3+1 Punkte*)

a) Seien $R(\phi)$ und $R(\gamma)$ Rotationen um die Winkel ϕ bzw. γ im $\mathbb{R}^2$. Zeigen Sie, dass gilt:

$$R(\phi) \cdot R(\gamma) = R(\phi + \gamma) = R(\gamma) \cdot R(\phi)$$

b) Entwickeln Sie eine Transformation, die einen achsenparallelen Quader, gegeben durch die untere linke Ecke (x_1, y_1, z_1) und die obere rechte Ecke (x_2, y_2, z_2), transformiert in einen zweiten achsenparallelen Quader, gegeben durch die untere linke Ecke (x_1', y_1', z_1') und die obere rechte Ecke (x_2', y_2', z_2'). Geben Sie die 4×4-Transformationsmatrix an.

c) Entwickeln Sie eine Transformationsfolge, die ein geometrisches Objekt an einer beliebigen Ebene im Raum reflektiert. Die Ebene sei gegeben durch einen Punkt $\mathbf{P}$ in dieser Ebene und den Ebenen-Normalenvektor $\mathbf{n}$. Geben Sie die Reihenfolge und die einzelnen 4x4- Transformationsmatrizen an.

   Tip: Sie können sich ein Koordinatensystem $\mathbf{n}, \mathbf{u}, \mathbf{v}$ aufbauen um die Spiegelung um die Ebene in ein einfacheres Problem zu überführen.

d) Zeigen Sie, dass folgende Matrizen A und B die gleiche Skalierung beschreiben.

$$A = \begin{pmatrix} s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} B = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \frac{1}{s} \end{pmatrix}$$

Aufgabe 2 (Transformationen, *6+5 Punkte*)

In dieser Aufgabe sollen einige Funktionen für ein kleines Carambolage-Billardspiel implementiert werden. Die Grundregeln des Spiels sind denkbar einfach: Der Spieler versucht mit der Spielkugel (weiße Kugel) die beiden anderen Kugeln zu berühren. Gelingt ihm dies, erhält er einen Punkt. Auf der Homepage finden Sie ein entsprechendes Framework.

a) Zuerst soll eine Kamerasteuerung implementiert werden. Dabei ist zwischen 3 verschiedenen Zuständen zu unterscheiden:

   i) Im Zustand `STATE_FREELOOK` soll sich die Kamera auf einer Halbkugel mit Radius `m_radius` und Mittelpunkt im Ursprung um den Tisch herum bewegen lassen. Der Radius lässt sich mittels der Pfeiltasten oder der mittleren Maustaste einstellen. Die Position der Kamera auf der Kugeloberfläche wird durch Kugelkoordinaten bestimmt und soll sich durch Mausbewegungen verändern lassen (Siehe Abb 1). Dabei soll die Kamera stets in Richtung des Mittelpunktes der Kugel (also des Ursprungs) schauen. Die Winkel der Kugelkoordinaten stehen Ihnen in

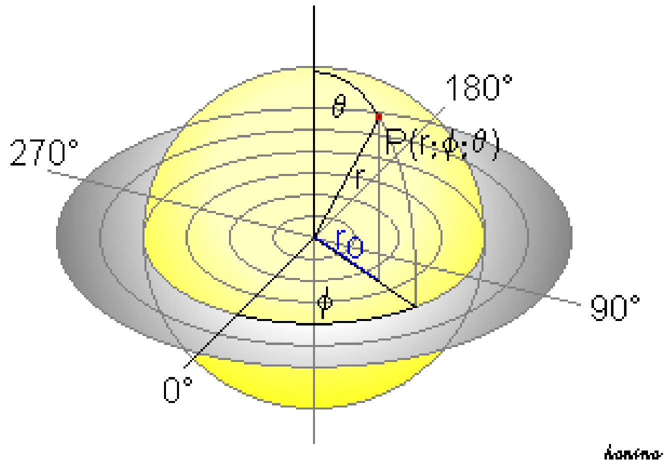


Abbildung 1: Kugelkoordinaten

den Variablen `m_phiRot` und `m_thetaRot` zu Verfügung und sind im Gradmaß angegeben. Der Tisch ist in der xy-Ebene ausgerichtet.

Achten Sie darauf, dass die Kamera sich wirklich nur auf einer Halbkugel bewegt und es nicht möglich ist, unter den Tisch zu blicken.

ii) Durch Drücken der rechten Maustaste wechselt man in den sogenannten Stoß-Modus (`STATE_SHOOTING`). Auf dem Bildschirm wird eine Skala angezeigt, mit der man die Stoßstärke mit Hilfe der Pfeiltasten einstellen kann. In diesem Zustand soll die Kamera sich ähnlich wie im Zustand `STATE_FREELOOK` bewegen lassen. Allerdings ist der Mittelpunkt der Halbkugel in diesem Zustand die Spielkugel `m_balls[0]`. Durch das Drehen der Kamera legt man die Stoßrichtung fest. Das Drücken der Leertaste löst den Stoß aus.

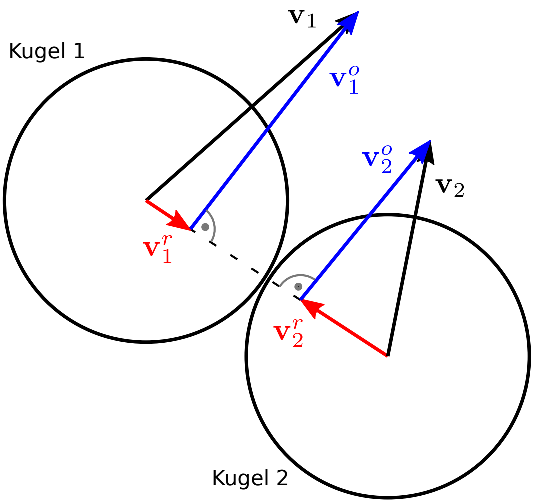
iii) Nach Auslösen des Stoßes wechselt das Spiel in den Simulations-Zustand `STATE_BALLSMOVING`. Hier werden die Kugeln entsprechend der Stoßkräfte bewegt. Beim Wechsel in diesen Zustand soll die Kamera zentral über dem Tisch positioniert werden, so dass der gesamte Tisch und damit die Kugelbewegungen zu sehen sind. Nach Beendigung der Simulation, wechselt das Spiel wieder in den Anfangs-Zustand `STATE_FREELOOK`.

Implementieren Sie die 3 Kamerasteuerungsarten in der Methode `GLWidget::moveCamera()`.

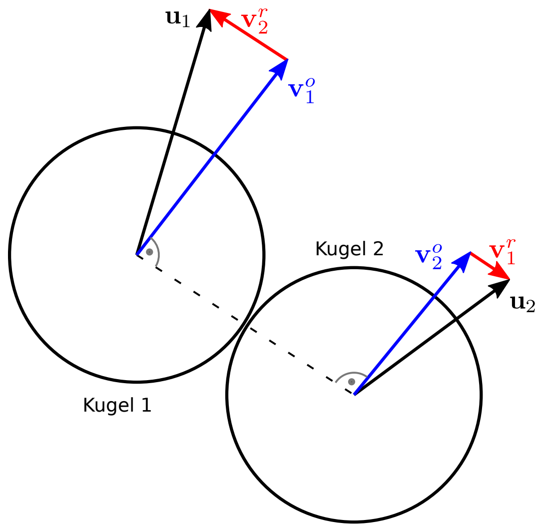
b) Im zweiten Teil der Aufgabe soll die Bewegung der Kugeln berechnet werden:

i) Dazu muss zuerst die Bewegungsrichtung des Spielballs in der Funktion `GLWidget::shoot()` berechnet werden. Wie in Aufgabenteil a) schon beschrieben, soll sich der Spielball in Blickrichtung bewegen. Setzen Sie dazu `m_balls[0].velocity` auf den entsprechenden Wert. Achten Sie darauf, dass der Geschwindigkeitsvektor normalisiert ist, da er später mit der Stoßstärke skaliert wird. Verwenden Sie nur die $x-$ und $y-$Koordinaten des Blickrichtungsvektors (da im Moment noch keine Schwerkraft simuliert wird).

ii) Nach Festlegung der Stoßrichtung und -stärke beginnt die Simulation der Kugelbewegungen in der Methode `GLWidget::simulate()`. Hier werden zuerst die neuen Positionen der Kugeln berechnet. Anschließend wird überprüft, ob zwei Kugeln kollidieren. Falls dies der Fall ist, müssen die Geschwindigkeitsvektoren der Kugeln neu gesetzt werden. Implementieren Sie dazu die Funktion `GLWidget::calcNewVelocitiesBallBall(indexBall1, indexBall2)`. `indexBall1` und `indexBall2` enthalten die Indizes der kollidierenden Kugeln im `m_balls`-Array.



Vor dem Stoß



Nach dem Stoß

Wenn zwei Kugeln kollidieren, müssen zunächst die Geschwindigkeitvektoren $\mathbf{v}_1$ und $\mathbf{v}_2$ zerlegt werden in die Vektoren $\mathbf{v}_1^r$ und $\mathbf{v}_2^r$, die in Richtung der anderen Kugel wirken, und in die dazu orthogonalen Vektoren $\mathbf{v}_1^o$ und $\mathbf{v}_2^o$.
Aus der Physik weiß man, dass die Geschwindigkeitskomponente $\mathbf{u}_1^r$ nach dem Stoß berechnet wird durch

$$\mathbf{u}_1^r = \frac{2m_2\mathbf{v}_2^r + (m_1 - m_2)\mathbf{u}_1^r}{m_1 + m_2}$$

und analog für $\mathbf{u}_2^r$. Da unsere Billardkugeln alle die gleiche Masse haben sollen, erhalten wir $\mathbf{u}_1^r = \mathbf{v}_2^r$ und $\mathbf{u}_2^r = \mathbf{v}_1^r$. Die Komponenten $\mathbf{v}_1^o$ und $\mathbf{v}_2^o$ bleiben unverändert.

Jetzt müssen die Geschwindigkeitsvektoren addiert werden. Für die Vektoren nach dem Stoß ergibt sich damit $\mathbf{u}_1 = \mathbf{v}_1^o + \mathbf{v}_2^r$ und $\mathbf{u}_2 = \mathbf{v}_2^o + \mathbf{v}_1^r$. Diese müssen noch um den Faktor `shockloss` verkürzt werden, um Stoßverluste zu simulieren.

iii) Nach der Kollisionsberechnung der Kugeln untereinander erfolgt noch ein Test, ob die Kugeln mit den Banden kollidieren. Auch in diesem Fall müssen die Geschwindigkeitsvektoren der Kugeln neu berechnet werden. Implementieren Sie dazu die Methode `GLWidget::calcNewVelocitiesBallBank(indexBall, banknormal[])`. Die Kollision einer Kugel mit einer Bande entspricht einer Reflexion, d.h. Einfallswinkel = Ausfallswinkel. Außerdem tritt infolge der Kollision ein Geschwindigkeitsverlust ein. Dieser entspricht dem Cosinus des Einfallswinkels multipliziert mit dem Verlustfaktor `bankfactor`.

Freiwillige Aufgaben (je 1 Punkt):

- Schalten Sie bei den Wechseln zwischen den Kamera-Modi nicht einfach hin und her, sondern interpolieren Sie eine Kamerafahrt von der Ausgangsposition zur Zielposition.
- Implementieren Sie einen Punktezähler, der anzeigt, wie viele erfolgreiche und wie viele erfolglose Stöße der Spieler bisher vollbracht hat.
- ...