Matlab Goes `doom`

Jörg J. Buchholz

13. September 2006

Einleitung

Sagt Ihnen die Tastenkombination W-A-S-D etwas? Haben Sie Ihrem halbwüchsigen Sohn schon mal über die Schulter gesehen, wenn er seinen neuesten Ego-Shooter spielt und darauf geachtet, wo die Finger seiner linken Hand liegen? Können Sie sich vorstellen, dass er sich intuitiv weitaus schneller und sicherer durch dreidimensionale Datendarstellungen in MATLAB bewegen kann als Sie mit Ihren guten alten `zoom`-, `pan`- und `rotate`-Schaltflächen; und das auch noch, ohne zu wissen, dass der Begriff „Matrix“ nicht nur im Kino, sondern auch in der Mathematik eine Bedeutung besitzt?

Die kommende Generation von Ingenieuren wird bei der Bewegung durch dreidimensionale Welten die Steuerung verwenden (wollen), mit der sie aufgewachsen ist: Die Maus schwenkt die Blickrichtung nach oben, unten, rechts und links und die Tasten bewegen den Betrachter auf die in Abbildung 1 definierte Weise in den drei räumlichen Freiheitsgraden. Auf MATLAB CENTRAL [1] finden Sie eine erste MATLAB-Implementation dieses Navigationsschemas (Ende 2005 sofort Platz 1 der „Most Downloaded This Week“ [2]

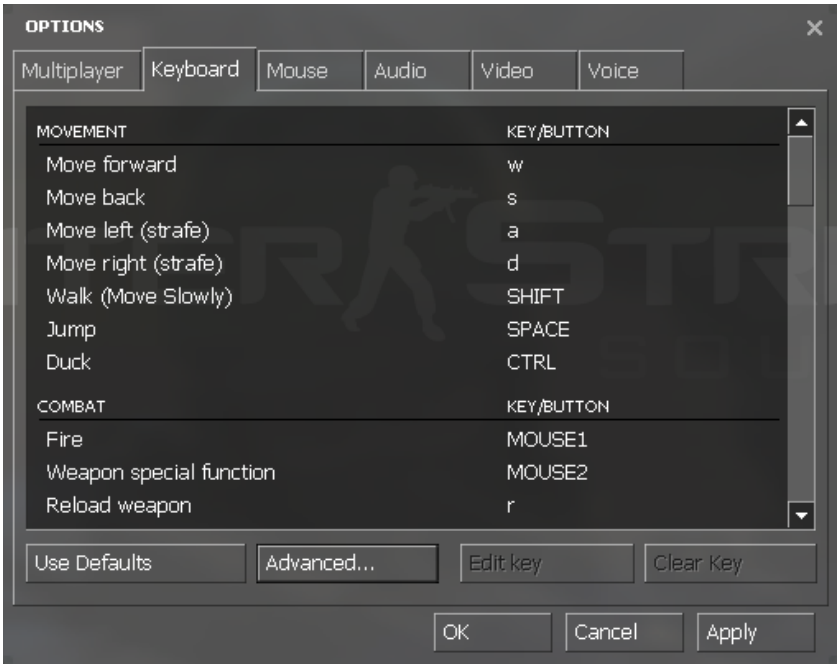

Abbildung 1: Tastenbelegung in Counter Strike Source

und „Pick of the Week“ Anfang 2006 [3]). Dieser Artikel soll nun ein wenig Licht auf die nahtlose Eleganz werfen, mit der sich `doom` in MATLAB einfügt.

Wo bin ich?

Um die Funktionsweise von `doom` genauer verstehen zu können, ist es sinnvoll und notwendig, zunächst ein paar Koordinatensysteme, Transformationswinkel und Kameravektoren zu definieren. Links unten in Abbildung 2 befindet sich der Ursprung des in rot dargestellten Basiskoordinatensystems, das als *geodätisches Koordinatensystem* bezeichnet wird und daher den Index g trägt. MATLAB drückt alle Vektoren in diesem Koordinatensystem aus. Beispielsweise liefert der Befehl

```
C_P = get (gca, 'CameraPosition')
```

die geodätischen Koordinaten der aktuellen Position der Kamera, mit der MATLAB Objekte „aufnimmt“. Die Kamera befindet sich also in Abbildung 2 an der Spitze des Camera-Position-Vektors $\mathbf{C}_P$ und „blickt“ in Richtung des Camera-View-Vektors $\mathbf{C}_V$, an dessen Spitze sich das darzustellende Objekt befindet. Der Befehl

```
C_T = get (gca, 'CameraTarget')
```

gibt den Camera-Target-Vektor $\mathbf{C}_T$ zurück, der die geodätischen Koordinaten der Position des betrachteten Objekts beinhaltet. Der Blickrichtungsvektor $\mathbf{C}_V$ der Kamera lässt sich daher durch eine einfache Vektordifferenz aus den anderen beiden Vektoren berechnen:

$$\mathbf{C}_V = \mathbf{C}_T - \mathbf{C}_P$$

Der Camera-View-Vektor $\mathbf{C}_V$ wiederum definiert die x_v-Achse des in Abbildung 2 blau dargestellten view-festen Koordinatensystems, dessen Achsen den Index v tragen und das die Lage (Ausrichtung) der Kamera im Raum festlegt.

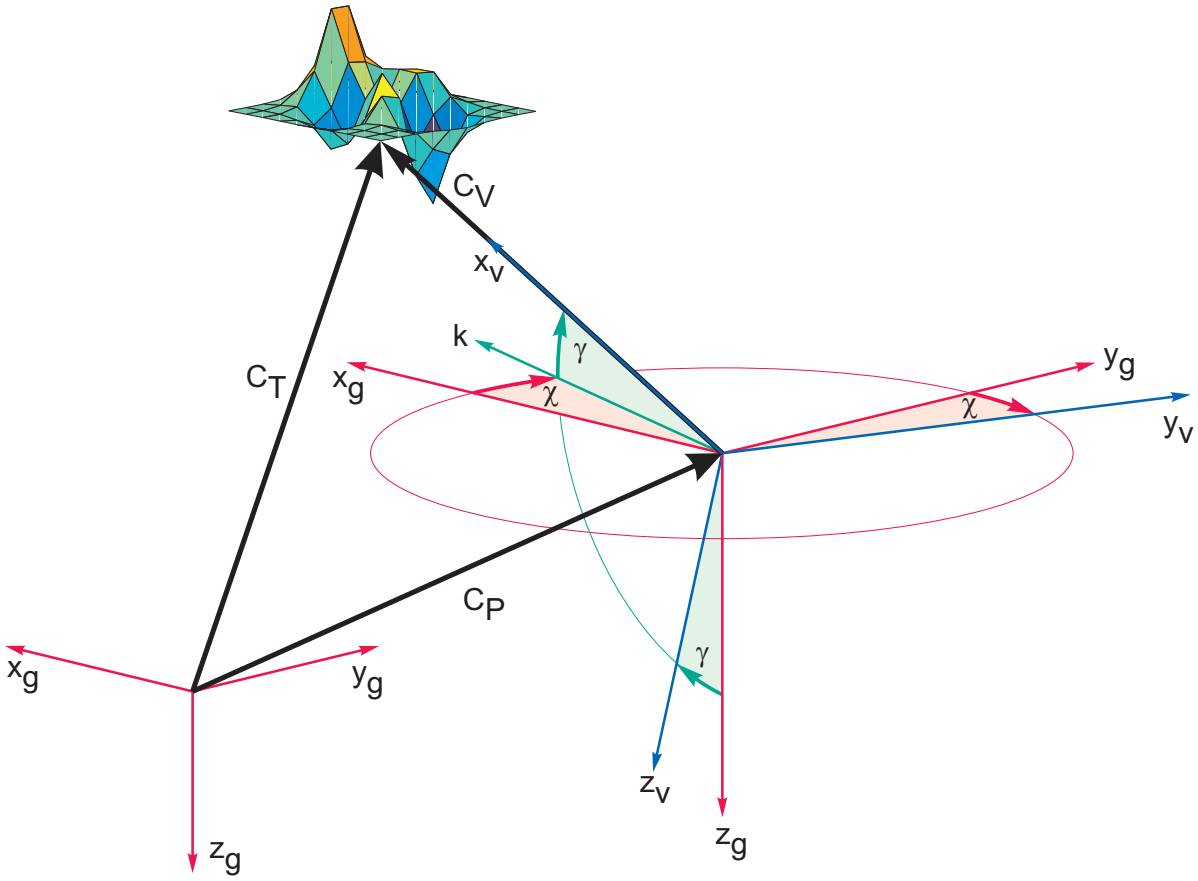

Abbildung 2: Zusammenhang zwischen dem Camera-Target-, dem Camera-Position- und dem Camera-View-Vektor ($\mathbf{C}_T = \mathbf{C}_P + \mathbf{C}_V$) und Transformation des geodätischen Koordinatensystems (Index g) in das view-feste Koordinatensystem (Index v) über die Drehwinkel χ (Azimut) und γ (Elevation)

Dabei wird die y_v-Achse des view-festen Koordinatensystems ganz bewusst in die Horizontalebene (also in die durch x_g und y_g aufgespannte Ebene) gelegt, so dass ein Rollen der Kamera um ihre Blickrichtung (x_v) nicht möglich ist. Dass dies eine sinnvolle Einschränkung ist, wird Ihnen sofort klar, wenn Sie "um Ihre eigene Blickrichtung rollen",

wenn Sie also Ihren Kopf langsam zur Seite neigen, so dass sich ein Ohr Ihrer Schulter nähert. Und - kippt während dieser Bewegung das Bild, das Sie wahrnehmen? Natürlich nicht; solange der Winkel nicht zu schnell zu groß wird (und Sie nüchtern sind), „rechnet Ihr bordeigenes steadycam-Bildverarbeitungssystem Rollwinkel in Echtzeit automatisch weg“.

Augen links! Kopf hoch!

In „wissenschaftlichen“ 3D-Steuerungen (MATLAB: Orbit Camera, `rotate3d`) bewegt sich die Kamera häufig mit konstantem Abstand um das betrachtete Objekt herum und richtet den Blick dabei immer auf das Objekt. In Abbildung 2 bleibt also $\mathbf{C}_T$ (und damit auch die Spitze von $\mathbf{C}_V$) fest, während sich der Anfangspunkt von $\mathbf{C}_V$ auf einer Kugeloberfläche bewegt. Für den Nutzer sieht das dann so aus, als ob er mit der Maus das Objekt mitsamt seinem Achsensystem dreht.

In Ego-Shootern sind translatorische und rotatorische Kamerabewegungen voneinander entkoppelt; die Translation wird über die Tastatur gesteuert, die Rotation geschieht mit Hilfe der Maus.

In Abbildung 2 bedeutet dies, dass eine Mausbewegung den Camera-View-Vektor $\mathbf{C}_V$ um seinen festen Anfangspunkt dreht. Dabei bewirkt eine links-rechts-Bewegung der Maus eine Gierbewegung von $\mathbf{C}_V$ in der Horizontalebene („Kopf blickt zur Seite“), während eine vor-zurück-Mausbewegung den Vektor $\mathbf{C}_V$ und damit den Kopf auf- bzw. abnicken lässt. Einer meiner Studenten drückte es einmal so aus: „Ich stelle mir vor, meine Hand läge (statt auf der Maus) auf dem Kopf der Person, die ich im Spiel steuere.“

Wie in Abbildung 3 dargestellt, wird zur Interpretation einer Mausbewegung zuerst der Camera-View-Vektor $\mathbf{C}_V$ als Differenz zwischen Camera-Target-Vektor $\mathbf{C}_T$ und Camera-Position-Vektor $\mathbf{C}_P$ berechnet. $\mathbf{C}_V$ wird dann in seine Kugelkoordinaten (Azimut χ, Elevation γ und Betrag) zerlegt. Der Weg, den die Maus seit ihrer letzten Abfrage zurückgelegt hat, führt schließlich zu einer Veränderung von Azimut und Elevation ($\Delta\chi$: links-rechts, $\Delta\gamma$: vor-zurück), so dass nach einer Rücktransformation in kartesische Koordinaten der veränderte Camera-View-Vektor und - in Summe mit dem unveränderten Camera-Position-Vektor - der neue Camera-Target-Vektor berechnet werden kann.

Ein weiterer Unterschied zwischen wissenschaftlicher Anwendung und Ego-Shooter besteht in der Fixierung des Mauszeigers. In einem Ego-Shooter bewegt eine Mausbewegung nicht den Mauszeiger auf dem Bildschirm sondern nur den Blick auf das dargestellte Objekt. Der Mauszeiger (resp. das Fadenkreuz einer Waffe) bleibt dabei ortsfest - beispielsweise in der Mitte - des aktuellen Fensters. In `doom` ist dieses Verhalten realisiert, indem in jedem Aufruf des Mausbewegungsunterprogrammes der Mauszeiger aktiv in die Mitte des aktuellen Fensters „zurückgesetzt“ wird:

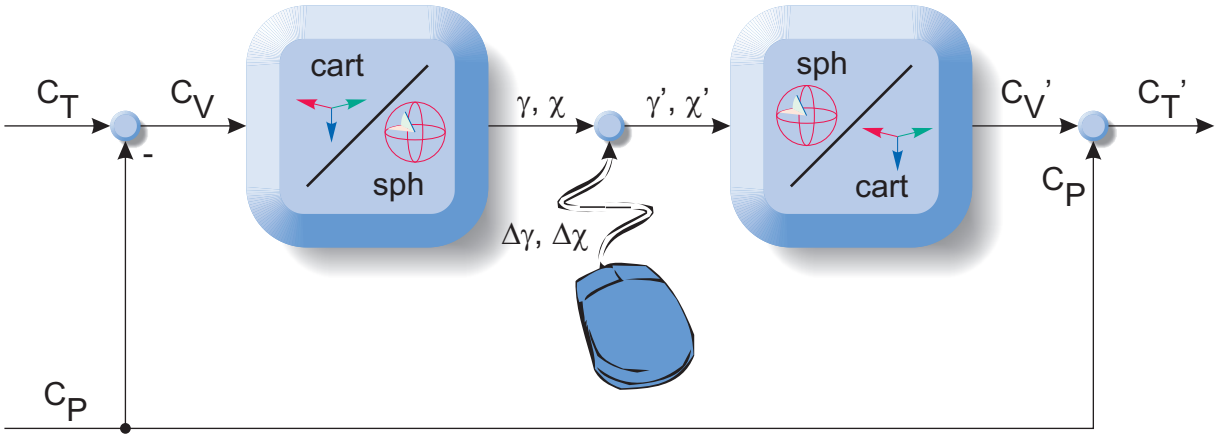

Abbildung 3: Eine Mausbewegung verändert Azimut χ und Elevation γ, damit die Lage des Camera-View-Vektors $\mathbf{C}_V$ und damit auch den Camera-Target-Vektor $\mathbf{C}_T$.

```
figure_position = get (gcf, 'position');
x_center = figure_position(1) + figure_position(3)/2;
y_center = figure_position(2) + figure_position(4)/2;
set (0, 'PointerLocation', [x_center, y_center]);
```

God Mode

Während die Maus also die Kopfdrehung eines ortsfesten Beobachters steuert, bewegt sich der Beobachter selbst in einem Ego-Shooter mit der Tastatur durch den Raum. Wie in Abbildung 1 festgelegt, bewirken dabei üblicherweise die Tasten `w` und `s` eine vor- bzw. zurück-Bewegung, während `a` und `d` den ganzen Beobachter nach links bzw. rechts verschieben. Die dritte Achse (unten/oben) wird häufig durch die Taste `Strg` und die Leertaste angesteuert. Während `Strg` in vielen Spielen dazu führt, dass sich die Spielperson duckt, also in die Hocke geht, wird die Leertaste meistens zum Springen (nach oben) verwendet.

Einige Spiele kennen einen so genannten „God Mode“, der sich manchmal durch „Cheats“ aktivieren lässt oder nach dem virtuellen Ableben der Spielperson automatisch eingeschaltet wird. Im „God Mode“ kann sich die Spielperson in allen drei Achsen - also auch durch Wände und Decken - frei bewegen.

Dabei stellt sich natürlich die entscheidende Frage, nach einer eindeutigen Definition der Begriffe *vorne*, *hinten*, *rechts*, *links*, *oben* und *unten*. Ist *vorne* beispielsweise immer in Nordrichtung festgelegt oder zeigt meine Nase immer nach *vorne*, egal wohin ich mich

drehe? Wo ist *oben*, wenn ich im Gras liege und den Himmel betrachte? Und unter Wasser oder im Weltraum; Kann ein Raumschiff nach *links* abbiegen? Einige Spiele verwenden hier eine zuweilen recht kontra-intuitive Mischung aus geodätischen und view-festen Achsen; `doom` definiert alle translatorischen Bewegungsrichtungen konsequent im view-festen Achsensystem. Die Taste `w` bewegt die Spielperson tatsächlich in die Richtung, in die das Riechorgan momentan weist; wenn sie auf dem Rücken liegt, also himmelwärts. Mit `w` bewegt man sich also näher an das betrachtete Objekt heran und kann (God Mode sei dank) auch durch Objekte „hindurch fliegen“. Ohne Blickrichtungsänderung entfernt man sich daher dann - nach „hinterem“ Objekthüllenaustritt - mit weiterhin gedrückter `w`-Taste natürlich wieder vom Objekt weg. Die Taste `d` bewegt in Richtung des rechten Ohrs und die Leertaste zieht die Spielperson immer in Richtung seiner eigenen Schädeldecke; auch wenn dies zu einer (geodätisch gesehen) waagerechten Bewegung führt, weil die Augen gerade auf den „Erdmittelpunkt“ gerichtet sind.

In Abbildung 2 lassen sich die Bewegungen besonders anschaulich darstellen: Die Tasten `w` und `s` bewegen den gesamten Camera-View-Vektor $\mathbf{C}_V$ entlang seiner Wirkungslinie in positive bzw. negative x_v-Richtung, `a` und `d` verschieben $\mathbf{C}_V$ parallel zu sich selbst in Richtung der negativen bzw. positiven y_v-Achse, während `Strg` und Leertaste eine Bewegung des Camera-View-Vektors in positive bzw. negative z_v-Achsenrichtung bewirken.

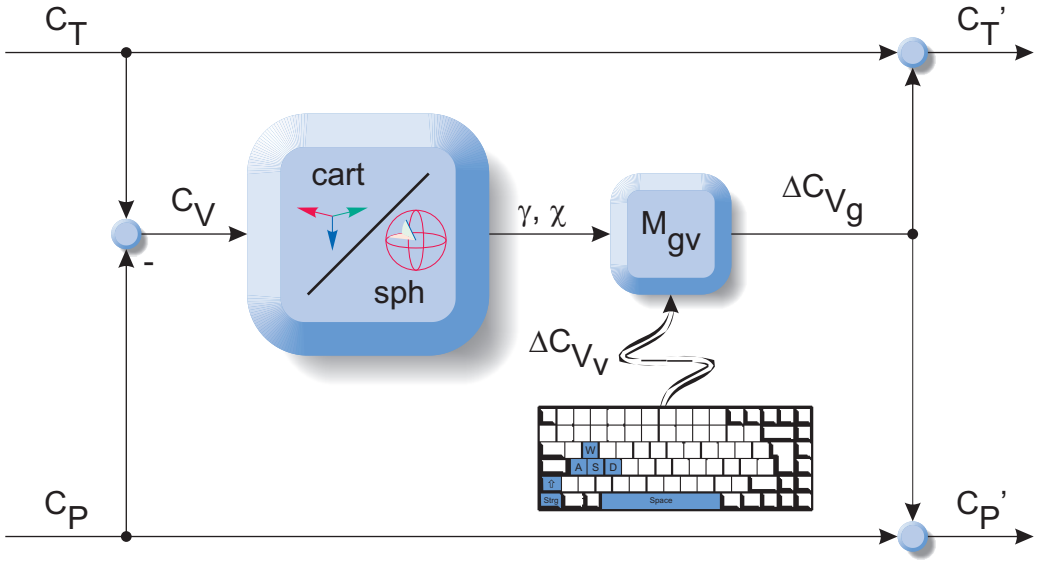

Abbildung 4: Mit der Tastatur wird eine Verschiebung des Camera-View-Vektors $\mathbf{C}_V$ in Richtung der view-festen Achsen definiert ($\Delta\mathbf{C}_{V_v}$). Diese wird mit Hilfe der Winkel χ und γ ins geodätische Koordinatensystem transformiert ($\Delta\mathbf{C}_{V_g}$) und bewirkt dort eine gleichzeitige Veränderung des Camera-Target-Vektor $\mathbf{C}_T$ und des Camera-Position-Vektor $\mathbf{C}_P$.

Abbildung 4 veranschaulicht die translatorische Bewegung nochmals in Form eines Blockschaltbildes. Wie schon bei der Rotation werden die Lagewinkel χ und γ, die die Lage des view-festen gegenüber dem geodätischen Koordinatensystems beschreiben, als Kugelkoordinaten des Camera-View-Vektors $\mathbf{C}_V$ berechnet, der sich wiederum aus der Differenz zwischen Camera-Target-Vektor $\mathbf{C}_T$ und Camera-Position-Vektor $\mathbf{C}_P$ ergibt. Aus den beiden Lagewinkel wird dann die Transformationsmatrix $\mathbf{M}_{gv}$ gebildet, mit der sich Vektoren vom view-festen ins geodätische Koordinatensystem transformieren lassen (s. Kapitel Matrix Reloaded). Jede Taste (`w a s d`...) liefert nun eine Verschiebung des Camera-View-Vektors entlang jeweils einer Achsenrichtung des view-festen Koordinatensystems ($\Delta\mathbf{C}_{V_v}$). Da Camera-Target- und Camera-Position-Vektor aber im geodätischen Koordinatensystem definiert sind, muss der Verschiebungsvektor durch Multiplikation mit der Transformationsmatrix ebenfalls im geodätische Koordinatensystem ausgedrückt werden ($\Delta\mathbf{C}_{V_g}$), bevor er zu den Camera-Vektoren addiert wird und so ihre Verschiebung bewirkt.

In vielen Spielen bewirkt die `Shift`-Taste (⇑) eine Erhöhung oder Erniedrigung der Bewegungsgeschwindigkeit (vgl. Abbildung 1). Auch in `doom` beschleunigt das gleichzeitige Drücken von `Shift` die Geschwindigkeit einer anderen Taste um das Zehnfache.

Sehr interessiert wäre der Autor von `doom` übrigens an einer Möglichkeit, unter MATLAB zwei gleichzeitig gedrückte Primärtasten (beispielsweise `w` und `a`) abzufragen.

Matrix Reloaded

Mathematisch lässt sich die Lage des Blickrichtungsvektors übersichtlich mittels zweier Drehmatrizen beschreiben. Man stelle sich dazu vor, das view-feste Koordinatensystem (Index v in Abbildung 2) wäre anfänglich mit dem (verschobenen) geodätischen Koordinatensystem (Index g) identisch. Die erste Drehung findet dann mit dem Azimutwinkel χ in der x_g-y_g-Ebene um die z_g-Achse statt. Dabei wird die x_v-Achse in die Zwischenachse k gedreht und die y_v-Achse landet in ihrer endgültigen Lage. Bei der zweiten Drehung wird mit dem Elevationswinkel γ in der x_v-z_v-Ebene um die y_v-Achse gedreht, um die x_v-Achse aus der Zwischenachse k und die z_v-Achse aus der z_g-Achsenrichtung in ihre jeweilige Endlage zu befördern.

Die Gesamttransformationsmatrix $\mathbf{M}_{vg}$ vom geodätischen ins view-feste Koordinatensystem ergibt sich dann als Produkt der beiden Einzeldrehmatrizen:

$$\mathbf{M}_{vg} = \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} \cos\chi & \sin\chi & 0 \\ -\sin\chi & \cos\chi & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\gamma\cos\chi & \cos\gamma\sin\chi & -\sin\gamma \\ -\sin\chi & \cos\chi & 0 \\ \sin\gamma\cos\chi & \sin\gamma\sin\chi & \cos\gamma \end{bmatrix}$$

Die Rücktransformation vom view-festen ins geodätische Koordinatensystem lässt sich sehr einfach berechnen, da die Inverse einer orthogonalen Drehmatrix freundlicherweise gleich ihrer Transponierten ist:

$$\mathbf{M}_{gv} = \mathbf{M}_{vg}^T = \begin{bmatrix} \cos\gamma\cos\chi & \cos\gamma\sin\chi & -\sin\gamma \\ -\sin\chi & \cos\chi & 0 \\ \sin\gamma\cos\chi & \sin\gamma\sin\chi & \cos\gamma \end{bmatrix}^T = \begin{bmatrix} \cos\gamma\cos\chi & -\sin\chi & \sin\gamma\cos\chi \\ \cos\gamma\sin\chi & \cos\chi & \sin\gamma\sin\chi \\ -\sin\gamma & 0 & \cos\gamma \end{bmatrix}$$

Mit $\mathbf{M}_{gv}$ lässt sich jetzt die Verschiebung der Kameravektoren auf Grund eines Tastendrucks sehr übersichtlich formulieren. Wenn der Nutzer beispielsweise die Taste `w` drückt, um die Kamera nach *vorne* zu bewegen, definiert er dadurch einen Verschiebungsvektor, der im view-festen Koordinatensystem nur eine x-Komponente besitzt $\Delta\mathbf{C}_{V_v} = \begin{bmatrix} \Delta x & 0 & 0 \end{bmatrix}_v^T$. Um die benötigte Verschiebung im geodätischen Koordinatensystem zu erhalten (vgl. Abbildung 4) muss $\mathbf{C}_{V_v}$ von links mit der Transformationsmatrix $\mathbf{M}_{gv}$ multipliziert werden, wobei erfreulicherweise nur die erste Spalte von $\mathbf{M}_{gv}$ benötigt wird:

$$\Delta\mathbf{C}_{V_g} = \mathbf{M}_{gv} \cdot \Delta\mathbf{C}_{V_v} = \begin{bmatrix} \cos\gamma\cos\chi & \cdot & \cdot \\ \cos\gamma\sin\chi & \cdot & \cdot \\ -\sin\gamma & \cdot & \cdot \end{bmatrix} \cdot \begin{bmatrix} \Delta x \\ 0 \\ 0 \end{bmatrix}_v = \begin{bmatrix} \cos\gamma\cos\chi \\ \cos\gamma\sin\chi \\ -\sin\gamma \end{bmatrix} \cdot \Delta x_v$$

Auch die anderen Tasten sprechen - bedingt durch die reine Bewegung in view-feste Achsenrichtungen - nur jeweils eine Spalte von $\mathbf{M}_{gv}$ an.

Die in Abbildung 3 benötigte Transformation von Kugel- in kartesische Koordinaten lässt sich unmittelbar an obiger Gleichung ablesen, indem Δx_v als „Radius“ r und die Komponenten von $\Delta\mathbf{C}_{V_g}$ als kartesische Koordinaten aufgefasst werden:

$$\begin{aligned} x &= \cos\gamma\cos\chi \cdot r \\ y &= \cos\gamma\sin\chi \cdot r \\ z &= -\sin\gamma \cdot r \end{aligned}$$

Die entsprechende Rücktransformation ergibt sich durch Auflösen der dritten Gleichung:

$$\gamma = -\arcsin\left(\frac{z}{r}\right)$$

Division der zweiten durch die erste Gleichung:

$$\chi = \arctan\left(\frac{y}{x}\right)$$

und natürlich Kraft des guten alten Pythagoras-3D:

$$r = \sqrt{x^2 + y^2 + z^2}$$

Ziehen oder Drücken?

Eine interessante Frage, die schon auf mancher LAN-Party zu heißen Diskussionen Anlass gegeben hat, ist die nach der „Mausumkehr“. Wohin soll der Blick gehen, wenn die Maus herangezogen wird? Nach *unten*, weil sich ja auch der Mauszeiger auf dem senkrechten Bildschirm (und damit auch der Blick des ihm folgenden Nutzers) beim Ziehen nach unten bewegt? Oder doch lieber nach *oben*, wie man es in einer Flugsimulation beim Heranziehen des Steuerknüppel gewohnt ist? Praktisch alle derzeitigen Spiele favorisieren in ihrer Standardeinstellung den ersten Fall, bieten aber die Möglichkeit, die Mausrichtung dauerhaft zu invertieren. Manche Spiele beinhalten sogar zwei Szenarien; einen Ego-Shooter und einen Flugsimulator. Konsequenterweise gibt es daher in modernen Spielen (Battlefield 2, ...) auch für jeden Teilbereich einen Einzelschalter, um die Maus zu invertieren. Auch `doom` bietet natürlich über die Taste `m` die Möglichkeit, die Mausumkehr ein- bzw. auszuschalten.

Im dualen Studiengang ILST (Internationaler Studiengang Luftfahrtsystemtechnik und -management) werden Studierende an der Hochschule Bremen zu Luftfahrtingenieuren (B.Eng.) und gleichzeitig an einer Flugschule zu Verkehrsflugzeugpiloten (ATPL) ausgebildet. Im Rahmen einer kleinen Umfrage [4] wurden diese angehenden „Piloten-Ingenieure“ zu ihren Mausumkehrpräferenzen befragt. Tatsächlich nutzen naturgemäß fast alle Befragten im Flugsimulator die Mausumkehr (Ziehen = Nase hoch). Interessanterweise schaltet aber die Hälfte der Studierenden im Ego-Shooter (und damit auch in ihrem Kopf!) die Mausumkehr wieder aus; teilweise, ohne sich darüber Gedanken zu machen. Die grenzwertige Frage, ob ein 3D-Bewegen im God Mode denn nicht eigentlich auch eine Flugsimulation darstellt, bleibt vorerst unbeantwortet...

Literatur

[1] Buchholz, J. J., *Doom.m*, MATLAB-Programm auf MATLAB CENTRAL File Exchange,
`http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=9340&objectType=FILE`

[2] The Mathworks, *Most Downloaded This Week*, MATLAB CENTRAL File Exchange,
`http://buchholz.hs-bremen.de/doom/doom_platz_1.png`

[3] Hirsch, S., *3D Navigation*, Pick of the Week auf MATLAB CENTRAL,
`http://blogs.mathworks.com/pick/?p=1632`

[4] Buchholz, J. J., *Umfrage zur Mausumkehr bei Ego-Shootern und Flugsimulatoren*,
`http://www.fbm.hs-bremen.de/sachma/ergebnis.aspx?db=doom`