1 | Rotation einer Schachtel

Ziel dieses Kapitels ist die Darstellung einer rotierenden Schachtel mit Hilfe von Geogebra-2D. Dazu braucht es im Wesentlichen 2 Schritte:

1. Die Darstellung der Schachtel im Kamera-System (Augen-System)
2. Die Projektion dieser transformierten Schachtel auf ein Zeichenblatt - wobei sich die Frage ergibt: welche Flächen sind sichtbar?

1.1 Transformation ins Augensystem

Die mathematischen Grundlagen, die wir benötigen, ist die Darstellung von Translationen, Spiegelungen und Drehungen von Punkten bzw. Koordinatensystemen (KS). Bei diesen Abbildungen handelt es sich um lineare Abbildungen und sie sind daher mit Matrizen darstellbar. Zuerst zu den **Drehungen**. Ein guter Überblick befindet sich in der Wikipedia.
Wichtig sind folgende Fakten:

- Passive Drehungen (also Drehung des KS) sind invers zu aktiven (siehe Anhang).
- $(R_\alpha)^{-1} = R_{-\alpha}$
- Drehmatrizen lassen das skalare Produkt (Winkel und Norm) invariant (Kongruenzabbildungen) und sind daher orthogonal!

  Es gilt nämlich:
  $$(A\,\vec{x}) \cdot \vec{y} = (a_{ij}x_j)y_i = x_j(a_{ij}y_i) = \vec{x} \cdot (A^T\vec{y})$$
  oder in Klammernnomenklatur des skalaren Produkts
  $$\langle A\,x, y\rangle = \langle x, A^T\,y\rangle$$
  Mit der Eigenschaft der Invarianz des skalaren Produkts und obiger Eigenschaft ergibt sich:
  $$\langle R\,x, R\,y\rangle = \langle x, y\rangle \Rightarrow \langle x, R^T\,R\,y\rangle = \langle x, y\rangle \Rightarrow R^T\,R = I$$
  also $(R_\alpha)^{-1} = (R_\alpha)^T$

Wir beschränken uns im Folgenden auf den $\mathbb{R}^3$.
Führt man homogene Koordinaten ein, lassen sich auch Translationen als Matrixmultiplikation darstellen. Die "Umrechnung" ist trivial - man führt eine 4.-te Koordinate ein und setzt sie 1.

$$\vec{x}_h = \begin{pmatrix} \vec{x} \\ 1 \end{pmatrix}$$

Alle Transformationsmatrizen leiten sich jetzt von (4×4) Einheitsmatrizen ab. Die für die Translation lässt sich jetzt schreiben:

$$\vec{c} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 1 \end{pmatrix} \qquad T = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \Rightarrow T\,\vec{c} = \begin{pmatrix} x+1 \\ y+2 \\ z+3 \\ 1 \end{pmatrix}$$

Auch bei den Drehmatrizen führt das zu analogen Ergebnissen (hier wird $\vec{e}_1$ um die z-Achse um α gedreht):

$$R_{z,\alpha} = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \vec{e}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \Rightarrow R_{z,\alpha}\,\vec{e}_1 = \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \\ 0 \\ 1 \end{pmatrix}$$

(Hinweis: Wer die Drehmatrizen herleiten möchte, braucht sie nur unbekannt ansetzen und auf die Basisvektoren $\vec{e}_i$ ansetzen mit bekanntem Ergebnis! Dadurch ergeben sich die 9 Variablen; Übrigens mit $R_{z,\alpha}\,R_{z,\beta} = R_{z,\alpha+\beta}$ ergeben sich die Summensätze der cos- und sin-Funktion!)
Was wir jetzt noch brauchen ist die Spiegelung um die y-z-Ebene M_{yz} - sie dreht das Vorzeichen der x-Koordinaten.
Hier noch die Zusammenfassung:

Transformationsmatrizen für homogene Koordinaten

$$T = \begin{pmatrix} 1 & 0 & 0 & x_T \\ 0 & 1 & 0 & y_T \\ 0 & 0 & 1 & z_T \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad M_{yz} = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_z = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_y = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Abb.1 : T-Translation; M_{yz}-Spiegelung um y-z-Ebene; R_i-Drehung um i-Achse

Was wir jetzt noch brauchen, um mit dem Rechnen zu beginnen, ist der Punkt, wo die Kamera (Augen) steht: dazu verwenden wir Kugelkoordinaten (ρ, θ, φ).
Die Umrechnung in kartesische Koordinaten sollte durch wiederholte Anwendung des "Pythagoras" keine Schwierigkeit darstellen:

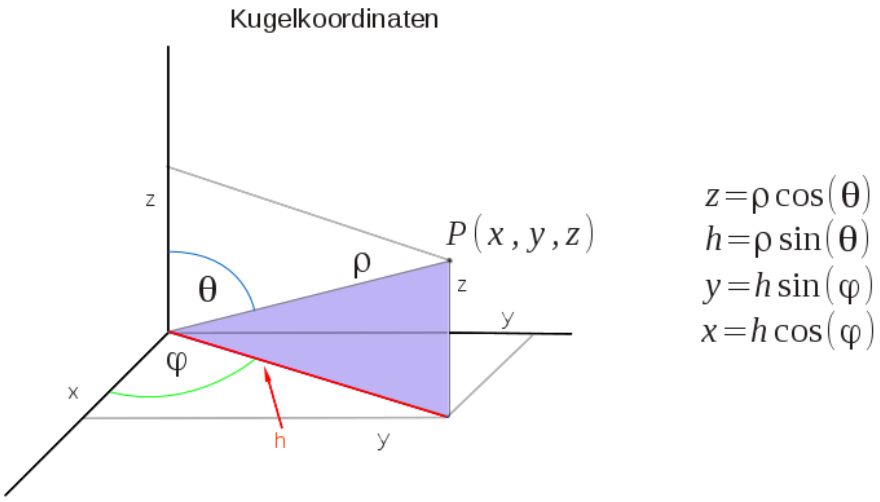


Abb.2 : Kugelkoordinaten

Das Augensystem wählen wir so, dass die x_A-Achse waagrecht nach rechts geht, die y_A-Achse nach oben (wie gewohnt) und die z_A-Achse von den Augen weg zum Ursprung des anderen Koordinatensystems.

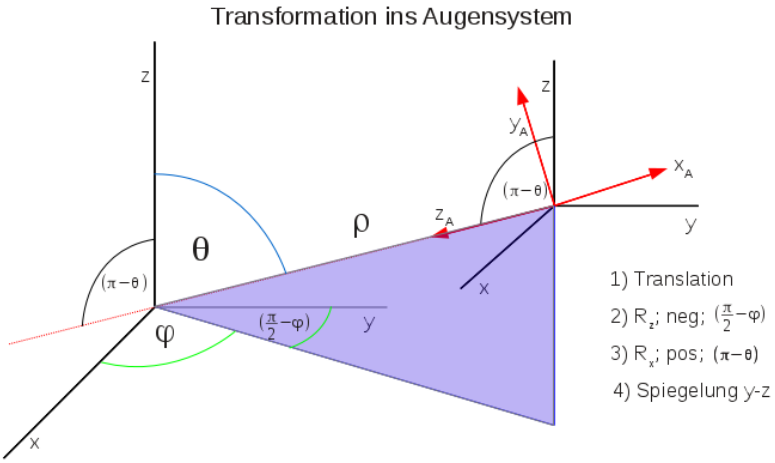


Abb.3 : Transformation ins Augensystem

Für die Transformationsmatrix T gibt es zwei Interpretationen(wie eingangs erwähnt):

- *aktiv*: $T \cdot P = P'$ P' sind die Koordinaten eines **neuen** Objekts
- *passiv*: $T^{-1} \cdot P = P'$ P' sind die Koordinaten **desselben** Objekts bei transformiertem Koordinatensystem (siehe Anhang: aktive und passive Transformation)

Hier ist einer der "springenden Punkte" in unserer Rechnung, darum schauen Sie sich die Zeichnung genau an und versuchen Sie die einzelnen Schritte nachzuvollziehen (auf den "alten" Koordinaten-Ursprung konzentrieren):

1. Translation - eh klar!
2. Drehung um die $z-$Achse im Uhrzeigersinn(math. negativ) bis $y-$Achse im blauen Dreieck liegt.
3. Jetzt Drehung um die $x-$Achse gegen den Uhrzeigersinn bis die $z-$Achse auf der Verbindungslinie der KS-Ursprünge "einrastet"
4. Zum Schluss das Umklappen der $x-$Achse (Spiegelung) - wir erhalten ein Linkssystem!

Zur Veranschaulichung kann man sich das Geogebra Arbeitsblatt auf
https://www.geogebra.org/m/NnKQP7Xq anschauen bzw. runterladen.
Die "Gesamtmatrix" - die das alles macht - lassen wir uns mit einem CAS berechnen (z.B. *wxMaxima* siehe Abschnitt 1.4)

$$T = \begin{pmatrix} -\sin(\phi) & \cos(\phi) & 0 & 0 \\ -\cos(\phi)\cos(\theta) & -\sin(\phi)\cos(\theta) & \sin(\theta) & 0 \\ -\cos(\phi)\sin(\theta) & -\sin(\phi)\sin(\theta) & -\cos(\theta) & \rho \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Hier endlich das Endergebnis - 1 Matrix für Alles!
Damit wäre der erste Teil - Darstellung der Schachtel im Augensystem - erledigt.

1.2 Projektion auf das Zeichenblatt

Dies ist eigentlich nur mehr eine Angelegenheit von "ähnlichen Dreiecken". Schauen wir uns dazu die folgende Zeichnung an (siehe auch https://www.geogebra.org/m/qmk2zmKe):

- Unser "Auge" sitzt im Ursprung
- Das Zeichenblatt ist parallel zur $xy-$Ebene mit Abstand D
- Die Koordinaten (x_S, y_S) der perspektivischen Projektion von P ergeben sich aus dem blauen und grünen Dreieck (,die z_P gemeinsam haben) zu:

$$\xi_S = \frac{D}{z_P}\,\xi_P \qquad \xi \in \{x, y\}$$

$0 < D < z_P \Rightarrow$ Verkleinerung

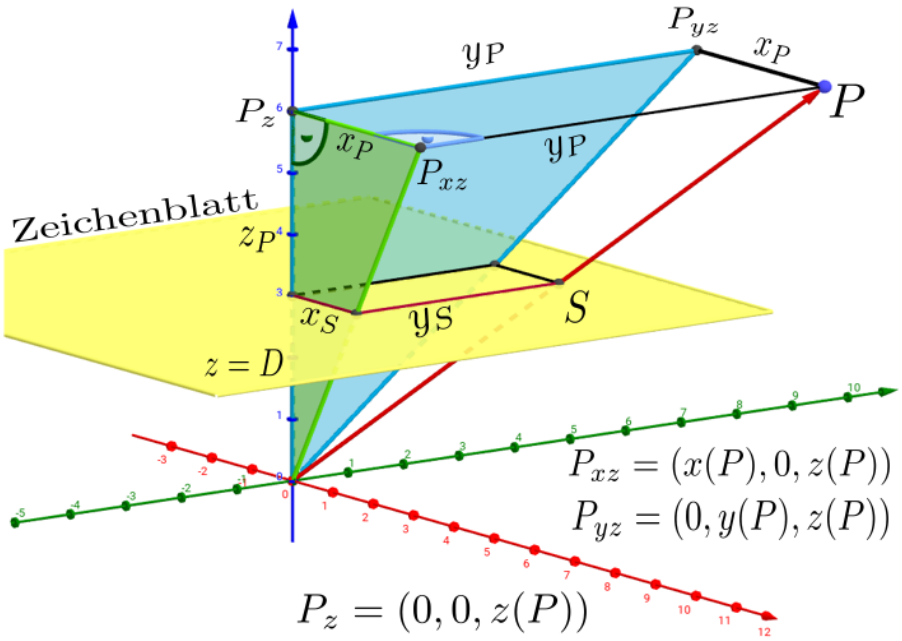


Abb.4 : perspektivische Projektion; Ursprung ist "Augenposition"

1.3 Sichtbarkeit

Wie man auf der Zeichnung sehen kann, ist eine Fläche nur dann sichtbar, wenn der Winkel zwischen dem Vektor $\vec{\sigma}$ (von einem Flächenpunkt zum Auge) und dem Normalvektor der Fläche (nach außen) spitz ist! Bilden die Schachtelkanten ein Basis-Dreibein gilt $\vec{n}_0 = \pm\vec{e}_i$ außerdem reichen für die 6 Flächenpunkte 2 Eckpunkte (Endpunkte der Raumdiagonalen) $\vec{p}_1$ und $\vec{p}_2$ aus:

$$\pm\vec{e}_i \cdot \underbrace{\left[\rho \begin{pmatrix} \cos\theta\cos\varphi \\ \sin\theta\sin\varphi \\ \cos\theta \end{pmatrix} - \vec{p}_k\right]}_{\vec{\sigma}} > 0$$

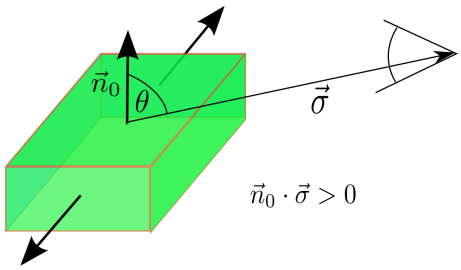


Abb.5 : Sichtbarkeit einer Fläche

Da bei den Basisvektoren $\vec{e}_i$ nur jeweils 1 Koordinate nicht verschwindet, entsteht eine skalare Sichtbarkeitsbedingung, die man in Geogebra bei den Eigenschaften der Polygonfläche eintragen kann!

Hier die "fertige" Datei zum runterladen auf
http://www.angsuesser.at/docs/math/pdf/3dbox-rotating/rotation.ggb.

1.4 Berechnung der Transformationsmatrix

Die Multiplikation einer Matrix mit einem Vektor kann als Linearkombination der Spaltenvektoren gedeutet werden, also

$$\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ \vec{a}_1 & \vec{a}_2 & \vec{a}_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{A} \cdot \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \lambda_1\vec{a}_1 + \lambda_2\vec{a}_2 + \lambda_3\vec{a}_3 = A \cdot \vec{\lambda}$$

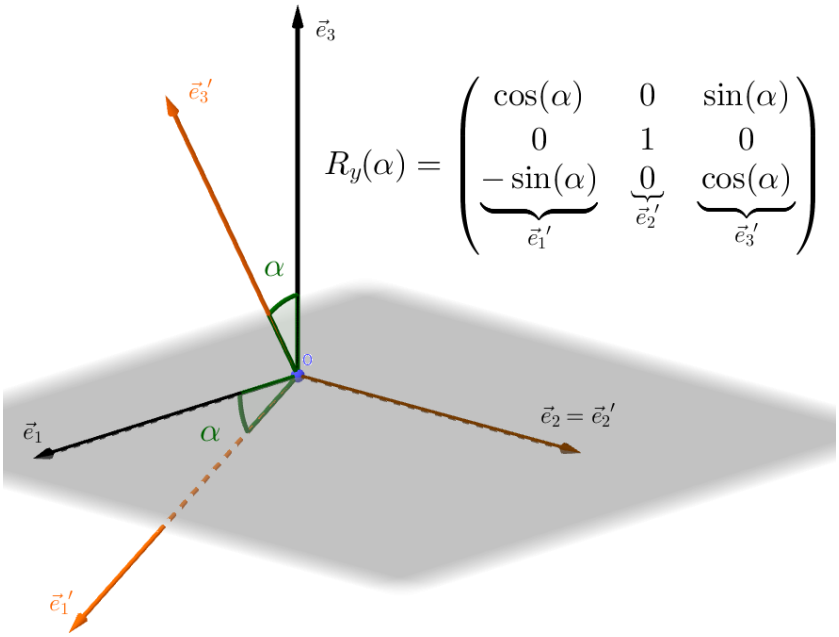
Damit ergibt sich folgender Sachverhalt:

Theorem 1.1 **Bedeutung der Spaltenvektoren**

A sei eine lineare Transformation (Matrix) von$\mathbb{R}^3 \to \mathbb{R}^3$
und
$\vec{e}_i$ seien die Standardbasisvektoren

$$\Rightarrow \begin{pmatrix} \vdots & \vdots & \vdots \\ A\vec{e}_1 & A\vec{e}_2 & A\vec{e}_3 \\ \vdots & \vdots & \vdots \end{pmatrix} = A$$

Die Spaltenvektoren von A sind die Transformation angewandt auf die Basisvektoren!



$$R_y(\alpha) = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) \\ 0 & 1 & 0 \\ \underbrace{-\sin(\alpha)}_{\vec{e}_1'} & \underbrace{0}_{\vec{e}_2'} & \underbrace{\cos(\alpha)}_{\vec{e}_3'} \end{pmatrix}$$

Abb.6 : Als Beispiel die Erstellung von $R_y(\alpha)$

Zum Beweis reicht einfaches ausrechnen!

Obiges Verfahren verwenden wir für die Darstellung von $R_z(\alpha)$ - für die homogenen Vektoren ergänzen wir noch mit der Einheitsmatrix

```
(%i1) R_z(%alpha):=matrix([cos(%alpha),-sin(%alpha),0,0],
                          [sin(%alpha),cos(%alpha),0,0],
                          [0,0,1,0],
                          [0,0,0,1]);
```

$$R_z\left(\alpha\right) := \begin{pmatrix} \cos\left(\alpha\right) & -\sin\left(\alpha\right) & 0 & 0 \\ \sin\left(\alpha\right) & \cos\left(\alpha\right) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{\%o1}$$

Dasselbe für R_x

```
(%i2) R_x(%alpha):=matrix([1,0,0,0],
                          [0,cos(%alpha),-sin(%alpha),0],
                          [0,sin(%alpha), cos(%alpha),0],
                          [0,0,0,1]);
```

$$R_x\left(\alpha\right) := \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\left(\alpha\right) & -\sin\left(\alpha\right) & 0 \\ 0 & \sin\left(\alpha\right) & \cos\left(\alpha\right) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{\%o2}$$

Wegen der Translation als Matrixmultiplikation haben wir die homeogenen Koordinaten eingeführt!

```
(%i3) T(x_T,y_T,z_T):=matrix([1,0,0,x_T],
                             [0,1,0,y_T],
                             [0,0,1,z_T],
                             [0,0,0,1]);
```

$$\mathrm{T}\left(x_T, y_T, z_T\right) := \begin{pmatrix} 1 & 0 & 0 & x_T \\ 0 & 1 & 0 & y_T \\ 0 & 0 & 1 & z_T \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{\%o3}$$

Spiegelung um die yz-Ebene - die x-Koordinate wechselt ihr Vorzeichen!

```
(%i4) M_yz:matrix([-1,0,0,0],
                  [0,1,0,0],
                  [0,0,1,0],
                  [0,0,0,1]);
```

$$\begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{M_yz}$$

Verschiebung in Kugelkoordinaten des ursprünglichen Koordinatensystems

```
(%i10) x_t:%rho * sin(%theta) * cos(%phi)$
       y_t:%rho * sin(%theta) * sin(%phi)$
       z_t:%rho * cos(%theta)$
```

Translation zum Auge anschl. Rotation

```
(%i15) R_zT:trigsimp(R_z(%pi/2-%phi) . T(-x_t,-y_t,-z_t));
```

$$\begin{pmatrix} \sin(\phi) & -\cos(\phi) & 0 & 0 \\ \cos(\phi) & \sin(\phi) & 0 & -\rho\sin(\theta) \\ 0 & 0 & 1 & -\rho\cos(\theta) \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{(R\_zT)}$$

Zum Vergleich: Rotation und anschl. Translation - die Koordinaten des Auges haben sich verändert: $\phi = \pi/2$

```
(%i18) TR_z:trigsimp( T(0,-%rho * sin(%theta),-%rho * cos(%theta)) . R_z(%pi/2-%phi) );
```

$$\begin{pmatrix} \sin(\phi) & -\cos(\phi) & 0 & 0 \\ \cos(\phi) & \sin(\phi) & 0 & -\rho\sin(\theta) \\ 0 & 0 & 1 & -\rho\cos(\theta) \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{(TR\_z)}$$

Jetzt die gesamte Transformationsmatrix

```
(%i22) Tr:M_yz . trigsimp(R_x(%theta - %pi) . TR_z) ;
```

$$\begin{pmatrix} -\sin(\phi) & \cos(\phi) & 0 & 0 \\ -\cos(\phi)\cos(\theta) & -\sin(\phi)\cos(\theta) & \sin(\theta) & 0 \\ -\cos(\phi)\sin(\theta) & -\sin(\phi)\sin(\theta) & -\cos(\theta) & \rho \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \text{(Tr)}$$

1.5 Implementation in Geogebra

Eine Matrix ist in Geogebra eine Liste von Listen - also : { {1,2,3},{4,5,6},{7,8,9}}
ein Spaltenvektor (benötigt für die Matrizenmultiplikation) ist dann
{ {1},{4},{7}}

- Die Transformationsmatrix eingeben:

```
T=  {{-sin(φ),          cos(φ),          0,        0},
     {-cos(φ) cos(θ),   -sin(φ) cos(θ),  sin(θ),   0},
     {-cos(φ) sin(θ),   -sin(φ) sin(θ),  -cos(θ),  ρ },
     {0 ,               0 ,              0 ,       1}}
```

Die 3 Schieberegler für ρ, θ, φ , die dadurch anfallen, abnicken und einstellen!

- Die 3 Zahlen für Länge (l), Breite (w) und Höhe (h) des Quaders festlegen, außerdem die Distanz des "Zeichenblatts" vor dem Auge (D) - je mehr sich D von ρ unterscheidet, umso geringer die "perspektivische Verzerrung" (Übergang in den Schrägriss)!
- Eine Liste L_1 der 4 Grundflächenpunkte erstellen:
  {(0, 0, 0), (l, 0, 0), (l, w, 0), (0, w, 0)}
- Eine Liste L_2 der 4 Deckflächenpunkte erstellen:
  $Sequence(Element(L_1, i) + (0, 0, h), i, 1, 4)$
- Eine Liste L_P aller Punkte erstellen:
  $Join(L_1, L_2)$
- Jetzt erstellen wir uns ein Werkzeug $hom[< Point >]$, welches aus einem Punkt einen homogenen Spaltenvektor erzeugt:
  (a) Punkt $P = (0, 0, 0)$ erstellen
  (b)
  ```
  P_h = {{x(P)}, {y(P)}, {z(P)}, {1}}
  ```
  (c) "Neues Werkzeug" erstellen, Eingabe P, Ausgabe P_h und Name hom
- Damit können wir eine Liste L_h der homogenen Spaltenvektoren erstellen:
  $Sequence(hom(Element(L_P, i)), i, 1, 8)$
- Jetzt wird ins "Augensystem" transformiert:
  ```
  L_T=Sequence(T * Element(L_h, i), i, 1, 8)
  ```
- Jetzt die 2D-Punkte der perspektivischen Projektion:
  ```
  L_{Prj}= Sequence( (

    D*Element(L_T, i, 1, 1) / Element(L_T, i, 3, 1),
    D*Element(L_T, i, 2, 1) / Element(L_T, i, 3, 1)

  )  ,i, 1, 8)
  ```
  Das ist sicher die schwierigste Befehlszeile, deshalb habe ich sie hier etwas strukturiert!

  Die roten Klammern erzeugen einen Punkt in Geogebra
- Jetzt die einzelnen Oberflächen (Polygone) festlegen, dazu erstellen wir eine Tabelle über die Eckpunkte und welchen Punkt wir für die Sichtbarkeit hernehmen:

Polygon	Eckpunkt-Indices	Punkt für Sichtbarkeit
P_1	1 2 3 4	1
P_2	5 6 7 8	7
P_3	1 2 6 5	1
P_4	2 3 7 6	7
P_5	3 4 8 7	7
P_6	1 4 8 5	1

- Legen wir jetzt die einzelnen Polygone fest und um die Sichtbarkeit kümmern wir uns anschließend. Das ist mühsame stupide Schreibarbeit - ich zeige das am Beispiel von P_4:
  `Polygon(Element(L_{Prj},2), Element(L_{Prj},3), Element(L_{Prj},7), Element(L_{Prj},6))`
  (Siehe auch: NACHSCLAG)

- Jetzt werden die Oberflächen(Polygone) verschieden eingefärbt (Durchsichtigkeit vermindert, opacity erhöht) und die Segmente(Kanten) auf "unsichtbar" geschaltet.

- Jetzt zur Sichtbarkeit: Wir legen uns eine Liste von Einheitsnormalvektoren für die einzelnen Polygone an:

  `NList = {(0, 0, -1), (0, 0, 1), (0, -1, 0), (1, 0, 0), (0, 1, 0), (-1, 0, 0)}`

  also `(0, -1, 0)` ist der Einheitsnormalvektor von P_3!

  Wir legen uns eine Liste von Punkten an, die auf der Fläche liegen (siehe Tabelle):

  `PInPoly = {(0, 0, 0), (4, 3, 2), (0, 0, 0), (4, 3, 2), (4, 3, 2), (0, 0, 0)}`

  Wir berechnen die "Augenkoordinaten":

  $$A = \rho * (\sin(\theta) * \cos(\phi), \sin(\theta) * \sin(\phi), \cos(\theta))$$

  Nun die Sehvektoren $\vec{\sigma}$ von der Fläche zu den Augen:

  σList = `Sequence(A - Element(PInPoly, i), i, 1, 6)`

  So nun noch die Liste der skalaren Produkte:

  `visibleL=Sequence(Element(NList, i) Element(σList, i), i, 1, 6)`
  Hier verbirgt sich das skalare Produkt, das auch mit `dot(<Vector1>,<Vector2>)` geschrieben werden kann!

- Praktisch geschafft:
  Wir tragen bei den einzelnen Polygonen die Sichtbarkeitsbedingung ein - z.B. für P_3:

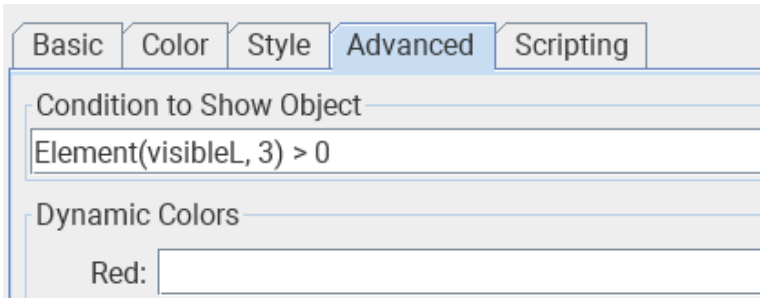


Abb.7 : Sichtbarkeitsdialog

- Animation für die Schiebregler einschalten und die Früchte der Arbeit genießen!

NACHSCHLAG:
Wie meist immer, wenn es sich um stupide Schreibarbeit handelt - wie oben, wo einfach 6 Polygone mit Eigenschaften versehen werden - bietet sich ein Programm in Javascript an. Zeitersparnis bringt das zwar ganz gewiss nicht - denn bis das Programm fehlerfrei läuft - hat man die Schreibarbeit auch erledigt. Aber es ist allemal interessanter und man lernt dabei einiges über das Javascript-Interface von Geogebra!
Daher an dieser Stelle auch die Version mit Javascript:
Der Reiter "Global Javascript" sieht dann bei mir folgendermaßen aus:

```
function ggbOnInit() {  main();  }

function main(){
 var nr=0;
    for (j=1; j<7; j++){
        initHelperLists(j);
       getPointListForPoly(j, 1);
       makePolygon(j);
     }
ggbApplet.deleteObject("PointIndices"); // not needed anymore
}
// ----------  main ends here ------------------------------

initHelperLists(j){
      ggbApplet.evalCommand("Q_{"+j+"0}={}");
      ggbApplet.setAuxiliary("Q_{" + j +"0}", true);
      ggbApplet.evalCommand("PointIndices=Element(PointList,"+j+")");
}

makePolygon(j){
  //Q_j4 ist point-list for j-th polygon denoted by P_j
  ggbApplet.evalCommand("P_"+j+"=Polygon(Q_{"+j+"4})");
  ggbApplet.evalCommand("SetConditionToShowObject(P_"+j+",
                         Element(visibleL,"+j+") > " +Number(0)+")");
  ggbApplet.setAuxiliary("P_"+j, true);
  ggbApplet.setColor("P_"+j, j*100, 250-j*30, j*10);
  ggbApplet.setFilling("P_"+j, 0.5);
}

getPointListForPoly(j, i){
   if (i < 5) {
// get point-indices for j-th polygon
     nr=ggbApplet.getListValue("PointIndices", i);
// get the current projection point as list and
//join it with its predecessors (4 points)
      ggbApplet.evalCommand("Q_{" + j + i + "}=Join(Q_{"+ j+ (i-1) +
                            "}, {Element(L_{Prj}," + nr +")})");
      ggbApplet.setVisible("Q_{" + j + i + "}", false);
      ggbApplet.setAuxiliary("Q_{" + j + i + "}", true);
    getPointListForPoly(j, i+1); //recur until 4 points
   }
}
```

In Zeile 36 wird mit verschachteltem *Join* die Punktliste `Q_j4` erstellt (ich habe die zweite Schleife hinter einer Rekursion "verborgen"). Man könnte meinen, dass man `Q_j0, Q_j1, Q_j2, Q_j3` löschen kann, da man sie ja nicht mehr braucht. Aber weit gefehlt. *Join* ist in Geogebra mit Zeigerverweisen implementiert (wie eine verkettete Liste) und es werden nicht die Werte kopiert. Mit dem Löschen von `Q_j0` verliert man den Listenanfang und damit die gesamte Liste. Man pflastert sich damit das Algebra-Fenster ganz schön zu - aber mit der Markierung als Hilfsobjekt verschwinden diese Teillisten wenigstens aus der Normalansicht!

Dazu gibt es noch einen Schaltfläche mit dem Titel "Start/Stop Animation" mit folgender Javascript Click-Methode:

```
startStop();

function startStop(){
 if (! ggbApplet.isAnimationRunning()){
   ggbApplet.setAnimating("\u03b8",true); // θ
   ggbApplet.setAnimating("\u03A6",true); // Φ
   ggbApplet.startAnimation();
}
else ggbApplet.stopAnimation();
}
```

Die utf-8 Codes für die griechischen Buchstaben in Javascript besorgt man sich aus dem Internet. Im *listing*-package von LaTeX(wie hier verwendet) ist dazu bei der Initialisierung des packages *mathescape=true* anzugeben:

```
\lstset{ %
 backgroundcolor=\color{white},
 .....
 mathescape=true
 ........
```

Dann kann man im Listing die griechischen Buchstaben wie gewohnt mit `$\theta$` schreiben!

1.6 ANHANG: Aktive und passive lineare Transformationen

Sei die lineare Transformation durch die Matrix A symbolisiert - ohne Beschränkung der Allgemeinheit sei sie hier (3×3).
Ihre Darstellung mit Spaltenvektoren lautet:

$$A = \begin{pmatrix} \vdots & \vdots & \vdots \\ \vec{a}_1 & \vec{a}_2 & \vec{a}_3 \\ \vdots & \vdots & \vdots \end{pmatrix}$$

- Zuerst die **aktive** Transformation:
  Das Bild von $\vec{p}$ unter A sei $\vec{p}\,'$, also

$$A \cdot \vec{p} = \vec{p}\,' \qquad \text{es wird ein neues Objekt erschaffen!}$$

- Jetzt die **passive** Transformation:
  Was sind die Koordinaten λ_i von $\vec{p}$ mit der Basis $\{A \cdot \vec{e}_1,\ A \cdot \vec{e}_2,\ A \cdot \vec{e}_3\}$?

$$\lambda_1 \underbrace{(A\vec{e}_1)}_{\vec{a}_1} + \lambda_2 (A\vec{e}_2) + \lambda_3 (A\vec{e}_3) = \vec{p}$$

$$\lambda_1 \vec{a}_1 + \lambda_2 \vec{a}_2 + \lambda_3 \vec{a}_3 = \vec{p}$$

$$A \cdot \vec{\lambda} = \vec{p} \Rightarrow \boxed{\vec{\lambda} = A^{-1} \cdot \vec{p}}$$

Die **passive** Transformation ist die **inverse** der **aktiven** Transformation