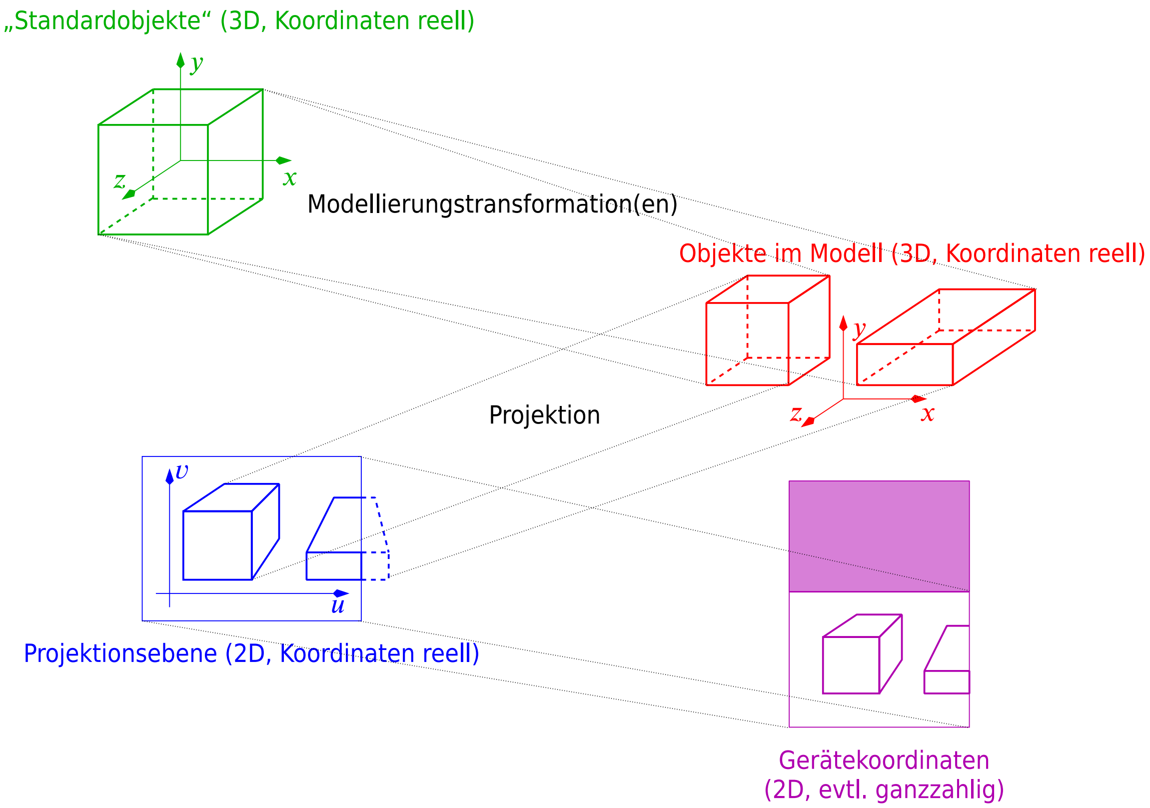
4 Koordinatentransformationen

Inhalt



Bemerkung 4.1: **OpenGL** ist ein weit verbreiteter Grafik-Standard mit verschiedenen Möglichkeiten für

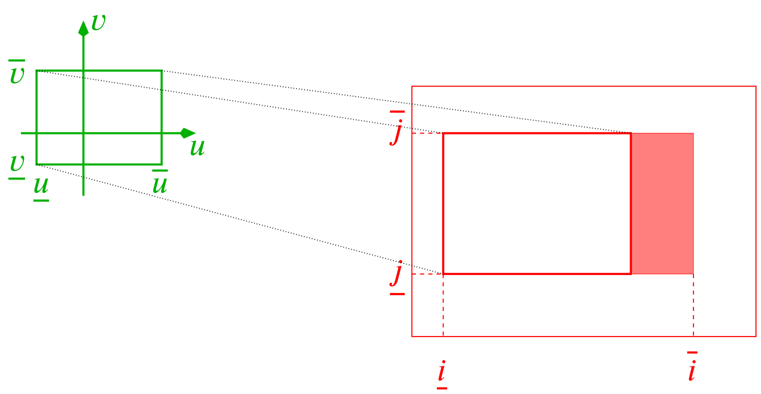
- die Beschreibung von Objekten,
- Handhabung hierarchischer Szenen,
- Festlegung von Projektionen und anderen Transformationen,
- Clipping (⇝ Kapitel 5) und Verdeckungsbehandlung (⇝ Kapitel 6),
- Färbung (⇝ Kapitel 7),
- ...

In der Vorlesung werden wir an geeigneten Stellen auf OpenGL verweisen.

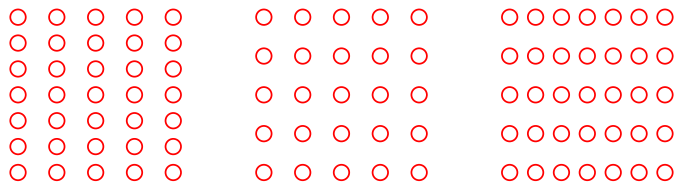
Im Februar 2016 wurde Version 1.0 des Nachfolgers **Vulkan** veröffentlicht.

4.1 Von der Projektionsebene zu Gerätekoordinaten

Ziel: Bilde den Bereich $[\underline{u};\overline{u}] \times [\underline{v};\overline{v}]$ „mit möglichst geringer Verzerrung“ in den Pixelbereich $[\underline{i},\dots,\overline{i}] \times [\underline{j},\dots,\overline{j}]$ ab.



Annahme: Die Pixel liegen in i-Richtung um einen Faktor d dichter als in j-Richtung:



$d = \frac{2}{3}$ $d = 1$ $d = \frac{3}{2}$

$\Rightarrow$ u-Koordinaten müssen d-fach stärker gestreckt werden

1. v-Bereich hat die Länge $\Delta v = \overline{v} - \underline{v}$

   j-Bereich hat die Länge $\Delta j = \overline{j} - \underline{j}$

   $\Rightarrow \quad 0 \leq$ Streckfaktor $s \leq \frac{\Delta j}{\Delta v}$

2. analog: $0 \leq d \cdot s \leq \frac{\Delta i}{\Delta u} = \frac{\overline{i} - \underline{i}}{\overline{u} - \underline{u}}$

   $\Leftrightarrow \quad 0 \leq s \leq \frac{\Delta i}{d \cdot \Delta u}$

Setze also

$$s := \min\left\{ \frac{\overline{i} - \underline{i}}{d \cdot (\overline{u} - \underline{u})}, \frac{\overline{j} - \underline{j}}{\overline{v} - \underline{v}} \right\}$$

und damit

$$\begin{aligned} i &:= d \cdot s \cdot \left(u - \underline{u}\right) + \underline{i} \\ j &:= s \cdot \left(v - \underline{v}\right) + \underline{j} \end{aligned}$$

bzw.

$$\begin{pmatrix} i \\ j \end{pmatrix} = \underbrace{\begin{pmatrix} d \cdot s & 0 \\ 0 & s \end{pmatrix}}_{=:\ M_{gp}} \cdot \begin{pmatrix} u \\ v \end{pmatrix} + \underbrace{\begin{pmatrix} \underline{i} - d \cdot s \cdot \underline{u} \\ \underline{j} - s \cdot \underline{v} \end{pmatrix}}_{=:\ t_{gp}}$$

$$= \underbrace{M_{gp} \cdot \begin{pmatrix} u \\ v \end{pmatrix} + t_{gp}}_{=:\ A_{gp}} \quad \text{(affine Transformation)}\,.$$

Bemerkung 4.2: Bei OpenGL wird der verfügbare Pixelbereich ganz genutzt:

$$M_{gp} = \begin{pmatrix} s_i & 0 \\ 0 & s_j \end{pmatrix}, \quad t_{gp} = \begin{pmatrix} \underline{i} - s_i \cdot \underline{u} \\ \underline{j} - s_j \cdot \underline{v} \end{pmatrix}$$

mit

$$s_i = \frac{\bar{i} - \underline{i}}{\bar{u} - \underline{u}}, \quad s_j = \frac{\bar{j} - \underline{j}}{\bar{v} - \underline{v}}$$

Für $\left(\bar{i} - \underline{i}\right) : \left(\bar{j} - \underline{j}\right) \neq d \cdot \left(\bar{u} - \underline{u}\right) : \left(\bar{v} - \underline{v}\right)$ treten dabei Verzerrungen auf.

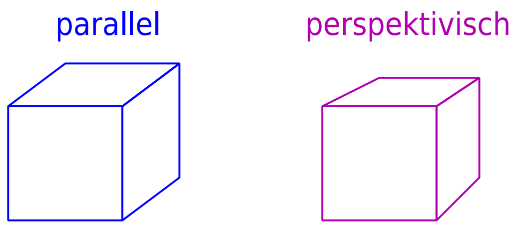
Bemerkung 4.3: Werden ganzzahlige Koordinaten benötigt, so setzt man

$$\begin{pmatrix} i \\ j \end{pmatrix} = \mathrm{round}\left(A_{gp} \begin{pmatrix} u \\ v \end{pmatrix} \right).$$

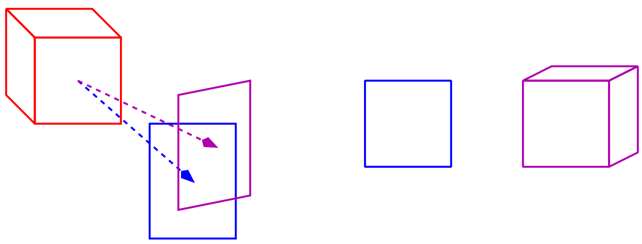
4.2 Projektion

Eine Projektion ist bestimmt durch:

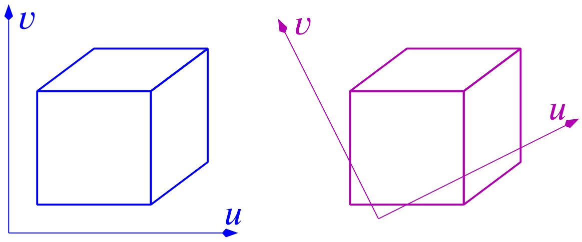
Projektionstyp:



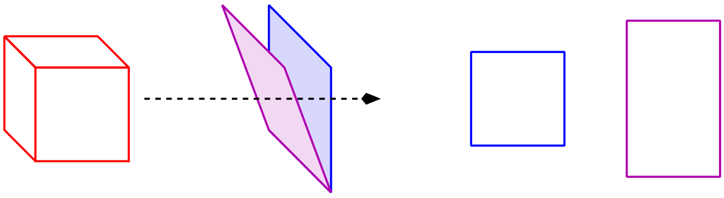
Projektionsrichtung:



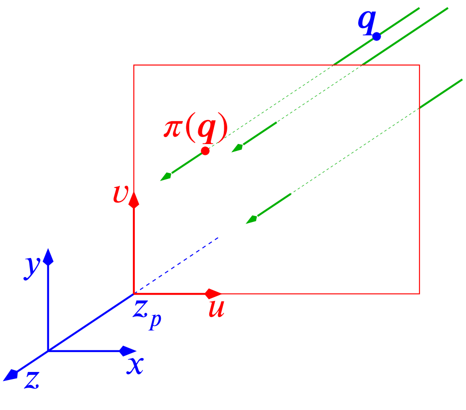
Koordinatensystem in der Projektionsebene:



(**Lage der Projektionsebene** zur Projektionsrichtung:)



4.2.1 „Standard-Parallelprojektion“



Projektionsebene: Ebene $z \equiv z_p$ (parallel zur xy-Ebene) im (rechtshändigen) „normalen“ dreidimensionalen Koordinatensystem

Projektionsrichtung: parallel zur z-Achse

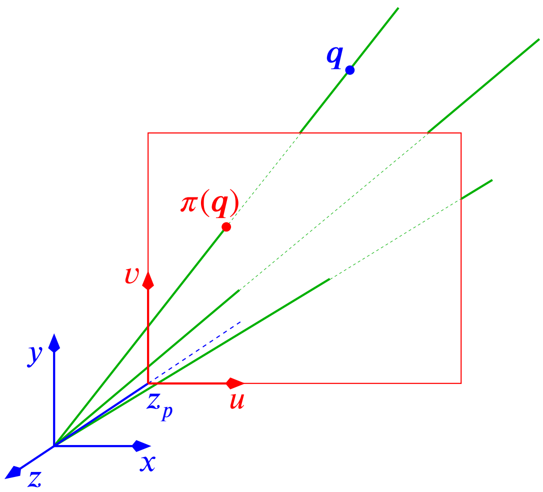
Koordinatensystem in der Projektionsebene: x, y aus dem dreidimensionalen Koordinatensystem

also:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}}_{=:\ M_{ps}} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\pi(\boldsymbol{q}) = M_{ps} \cdot \boldsymbol{q} =: A_{ps}(\boldsymbol{q})$$ (affine [sogar lineare] Transformation)

4.2.2 „Standard-Perspektivische Projektion“

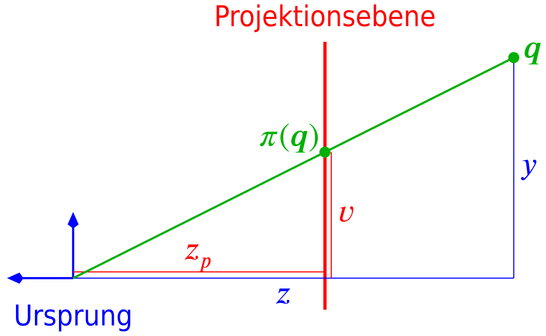


Projektionsebene: Ebene $z \equiv z_p$ im (rechtshändigen) „normalen“ dreidimensionalen Koordinatensystem

Projektionsrichtung: gegeben durch das Projektions**zentrum im Ursprung** des (x, y, z)-Systems

Koordinatensystem in der Projektionsebene: x, y aus dem dreidimensionalen Koordinatensystem

Wegen $\pi(\boldsymbol{q}) = \left(u, v, z_p\right)^T$ liefert der **Strahlensatz**



$$\frac{v}{y} = \frac{z_p}{z} = \frac{u}{x}$$

und damit

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{z_p}{z} \cdot \begin{pmatrix} x \\ y \end{pmatrix} .$$

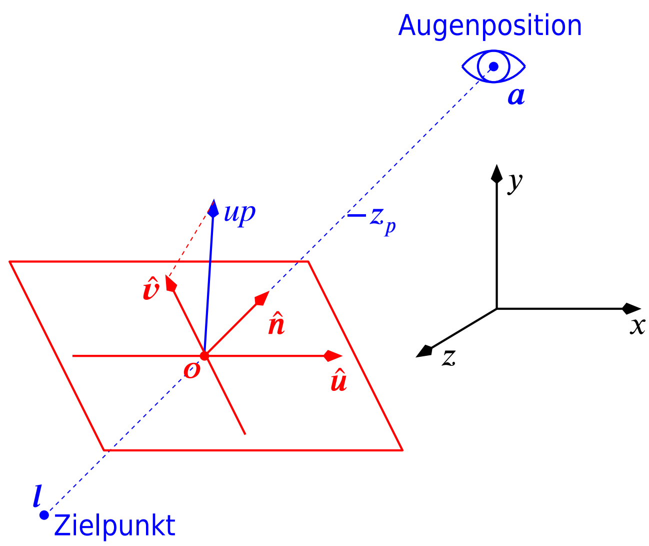
⇒ **keine affine Transformation!**

4.2.3 Allgemeine Parallel- bzw. perspektivische Projektion

Problem: Für die Standard-Projektionen müssen die Objekte so positioniert werden, dass sie bei einer bestimmten Lage der Projektionsebene ($z \equiv z_p$) sichtbar sind.

In Analogie zu einer Aufnahme mit einer Kamera sollte es möglich sein, die Objekte in einem für die Modellierung „natürlichen“ Koordinatensystem zu positionieren und die Projektionsebene geeignet zu wählen.

Dies kann etwa folgendermaßen realisiert werden (z. B. in OpenGL):



- Lege (in dem auch zur Positionierung der Objekte verwendeten (x, y, z)-Koordinatensystem)
  - eine **Augenposition** $\boldsymbol{a}$ und
  - einen **Zielpunkt** $\boldsymbol{l}$ („look at“)

  sowie den Abstand $-z_p$ der Projektionsebene fest.

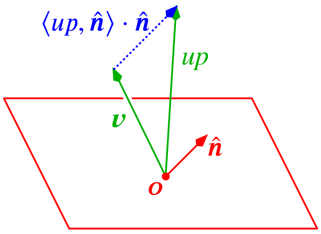
  Dann befindet sich die Projektionsebene im Abstand $-z_p$ vom Augenpunkt und ist senkrecht zur Richtung $\boldsymbol{r} = \overrightarrow{\boldsymbol{al}}$.

  $\Rightarrow$ $\hat{\boldsymbol{n}} := -\frac{\boldsymbol{r}}{\|\boldsymbol{r}\|_2}$ ist ein (zum Augenpunkt zeigender, normierter) Normalenvektor zur Projektionsebene.
- Der Ursprung $\boldsymbol{o}$ des (u, v)-Koordinatensystems liegt auf dem Strahl $\overrightarrow{\boldsymbol{al}}$.

- Die positive v-Achse ergibt sich aus der Orthogonalprojektion einer (in x, y, z gegebenen) „Aufwärtsrichtung“ $up \in \mathbb{R}^3$.

  (up bestimmt, ob die [in Richtung $\boldsymbol{l}$ zeigende] Kamera gedreht ist [z. B. für eine „Hochformat“-Aufnahme].)

$$\begin{aligned}\boldsymbol{v} &:= \text{Orthogonalprojektion von } up \text{ auf Projektionsebene}\\ &= up - \text{Komponente von } up \text{ in Richtung } \hat{\boldsymbol{n}}\\ &= up - \langle up, \hat{\boldsymbol{n}}\rangle \cdot \hat{\boldsymbol{n}}\end{aligned}$$



$$\hat{\boldsymbol{v}} = \frac{\boldsymbol{v}}{\|\boldsymbol{v}\|_2}$$

- Die u-Achse wird so gewählt, dass $(\boldsymbol{o}; \hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}, \hat{\boldsymbol{n}})$ (normierte Vektoren) ein **rechtshändiges** 3D-Koordinatensystem bilden:

$$\begin{aligned}\hat{\boldsymbol{u}} &= \hat{\boldsymbol{v}} \times \hat{\boldsymbol{n}} \\ &= \begin{pmatrix} \hat{v}_y\hat{n}_z - \hat{v}_z\hat{n}_y \\ \hat{v}_z\hat{n}_x - \hat{v}_x\hat{n}_z \\ \hat{v}_x\hat{n}_y - \hat{v}_y\hat{n}_x \end{pmatrix} = \det \begin{pmatrix} \hat{\boldsymbol{x}} & \hat{\boldsymbol{y}} & \hat{\boldsymbol{z}} \\ \hat{v}_x & \hat{v}_y & \hat{v}_z \\ \hat{n}_x & \hat{n}_y & \hat{n}_z \end{pmatrix}\end{aligned}$$

($\hat{\boldsymbol{x}}$, $\hat{\boldsymbol{y}}$, $\hat{\boldsymbol{z}}$: Einheitsvektoren in x-, y- bzw. z-Richtung)

Bemerkung: Es gilt

$$\begin{aligned}\langle \hat{\boldsymbol{u}}, \hat{\boldsymbol{n}} \rangle = \langle \hat{\boldsymbol{u}}, \hat{\boldsymbol{v}} \rangle &= 0 \\ \|\hat{\boldsymbol{u}}\|_2 &= 1 \\ \det(\hat{\boldsymbol{u}}|\hat{\boldsymbol{v}}|\hat{\boldsymbol{n}}) &= +1 \qquad \text{(rechtshändig)}\end{aligned}$$

Für

- Parallelprojektion entlang der Richtung $\pm\boldsymbol{r}$ bzw.
- perspektivische Projektion mit Zentrum $\boldsymbol{a}$

liegt damit bezüglich (u, v, n)-Koordinaten „fast die Standardsituation“ vor. (Der Augenpunkt liegt nicht im Ursprung, sondern an Position $(0, 0, -z_p)$.)

Die Projektion eines Punktes $\boldsymbol{q} = (x, y, z)^T$ (in **„Weltkoordinaten“**) erfolgt also mit der Standard-Projektion, **nachdem** die folgenden Schritte durchgeführt wurden:

1. Verschiebung in den Ursprung des $(\boldsymbol{o}; \hat{\boldsymbol{u}}, \hat{\boldsymbol{v}}, \hat{\boldsymbol{n}})$-Systems:

$$\underbrace{\begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{pmatrix}}_{\tilde{\boldsymbol{q}}} = \underbrace{\begin{pmatrix} x \\ y \\ z \end{pmatrix}}_{\boldsymbol{q}} - \underbrace{\begin{pmatrix} o_x \\ o_y \\ o_z \end{pmatrix}}_{\boldsymbol{o}}$$

mit $\boldsymbol{o} = \boldsymbol{a} + z_p \cdot \hat{\boldsymbol{n}}$

2. alte Koordinatenrichtungen, ausgedrückt in den neuen:

$$u = \text{Länge der Komponente } \tilde{\boldsymbol{q}} \text{ in Richtung } \hat{\boldsymbol{u}} = \langle \hat{\boldsymbol{u}}, \tilde{\boldsymbol{q}} \rangle = \hat{\boldsymbol{u}}^T \cdot \tilde{\boldsymbol{q}}$$

analog:

$$v = \hat{\boldsymbol{v}}^T \cdot \tilde{\boldsymbol{q}}\,, \qquad n = \hat{\boldsymbol{n}}^T \cdot \tilde{\boldsymbol{q}}\,,$$

also

$$\begin{pmatrix} u \\ v \\ n \end{pmatrix} = \begin{pmatrix} \hat{\boldsymbol{u}}^T \\ \hline \hat{\boldsymbol{v}}^T \\ \hline \hat{\boldsymbol{n}}^T \end{pmatrix} \cdot \tilde{\boldsymbol{q}} = (\hat{\boldsymbol{u}}|\hat{\boldsymbol{v}}|\hat{\boldsymbol{n}})^T \cdot \tilde{\boldsymbol{q}}$$

Bemerkung: Die Matrix

$$M_k := (\hat{\boldsymbol{u}}|\hat{\boldsymbol{v}}|\hat{\boldsymbol{n}})$$

drückt „neue" Koordinaten in „alten" aus, vermittelt also den Koordinatenwechsel

$$(u, v, n) \rightsquigarrow (\tilde{x}, \tilde{y}, \tilde{z}) \ .$$

Daher wird der Koordinatenwechsel

$$(\tilde{x}, \tilde{y}, \tilde{z}) \rightsquigarrow (u, v, n)$$

durch die Matrix

$$M_k^{-1} = M_k^T$$

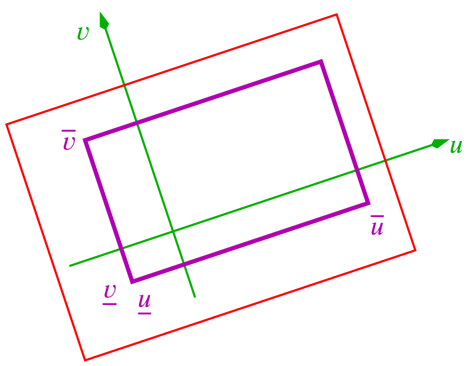
ausgedrückt.

3. Verschiebung des Augenpunktes in den Ursprung:

$$\underbrace{\begin{pmatrix} u' \\ v' \\ n' \end{pmatrix}}_{\boldsymbol{q}'} = \begin{pmatrix} u \\ v \\ n \end{pmatrix} + \underbrace{\begin{pmatrix} 0 \\ 0 \\ z_p \end{pmatrix}}_{\boldsymbol{d}}$$

Insgesamt gilt dann:

$$\begin{aligned} \boldsymbol{q}' &= M_k^T \cdot (\boldsymbol{q} - \boldsymbol{o}) + \boldsymbol{d} \\ &= M_k^T \cdot \boldsymbol{q} + \left(\boldsymbol{d} - M_k^T \cdot \boldsymbol{o}\right) \\ &=: M_{sw} \cdot \boldsymbol{q} + \boldsymbol{t}_{sw} \qquad \text{(affine Abbildung)} \end{aligned}$$



Der später im Bild **sichtbare Bereich** wird festgelegt durch ein **Projektionsfenster** $[\underline{u}, \overline{u}] \times [\underline{v}, \overline{v}]$, das den Ursprung $\boldsymbol{o}$ des Koordinatensystems nicht enthalten muss.

4.2.4 Projektive („homogene“) Koordinaten

Motivation

- Die Schachtelung affiner Abbildungen

$$\boldsymbol{q} \mapsto \boldsymbol{M}_{gp} \cdot \left(\boldsymbol{M}_{ps} \cdot \left(\boldsymbol{M}_{sw} \cdot \boldsymbol{q} + \boldsymbol{t}_{sw}\right)\right) + \boldsymbol{t}_{gp}$$

  benötigt viele Operationen. Ohne die Translationen könnten die Matrizen zu einer einzigen Matrix

$$\boldsymbol{M} := \boldsymbol{M}_{gp} \cdot \boldsymbol{M}_{ps} \cdot \boldsymbol{M}_{sw}$$

  zusammengefasst werden, und die Transformationen

$$\boldsymbol{q} \mapsto \boldsymbol{M} \cdot \boldsymbol{q}$$

  wären wesentlich effizienter.

  Kann eine Translation bei geeigneter Koordinatenwahl als lineare Transformation (Matrix) geschrieben werden?

- Perspektivische Projektion führt nicht einmal auf eine affine Transformation.

  Kann sie bei geeigneter Koordinatenwahl „affin (oder sogar linear) gemacht“ werden?

n-dimensionaler projektiver Raum:

$$\mathbb{P}_n := \text{Menge aller eindimensionalen Unterräume des } \mathbb{R}^{n+1}$$

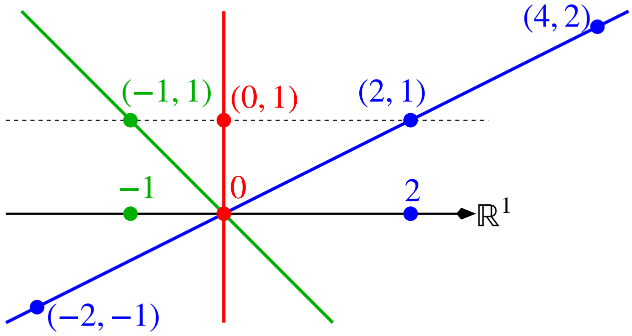
Elemente des $\mathbb{P}_n$: Ergänze die n-dimensionalen Koordinaten durch eine $(n+1)$-te mit Wert 1:

$$\left(x_1, \ldots, x_n\right) \rightsquigarrow \left(x_1, \ldots, x_n, 1\right)$$

Identifiziere im $\mathbb{R}^{n+1}$ alle Vielfachen ($\neq 0$) eines Vektors:

$$\left(x_1, \ldots, x_n, x_{n+1}\right) \equiv \left(\tau x_1, \ldots, \tau x_n, \tau x_{n+1}\right) \quad \text{für } \tau \neq 0$$

Beispiel:



- Für $x_{n+1} \neq 0$ entspricht $\left(x_1, \ldots, x_n, x_{n+1}\right)$ dem (eindeutigen) Punkt $\left(\frac{x_1}{x_{n+1}}, \ldots, \frac{x_n}{x_{n+1}}\right)$ im $\mathbb{R}^n$,
- Für $x_{n+1} = 0$ entspricht $\left(x_1, \ldots, x_n, 0\right)$ **keinem** Punkt im $\mathbb{R}^n$ **(„unendlich ferner Punkt“)**.

Wie übertragen sich n-dimensionale Transformationen auf projektive Koordinaten?

lineare Transformationen:

$$\boldsymbol{x} \mapsto \boldsymbol{y} := M \cdot \boldsymbol{x}\,, \qquad \boldsymbol{x} \in \mathbb{R}^n$$

$$\xi = \begin{pmatrix} \tau \boldsymbol{x} \\ \tau \end{pmatrix} \mapsto \boldsymbol{\eta} = \begin{pmatrix} \tau \boldsymbol{y} \\ \tau \end{pmatrix} = \begin{pmatrix} \tau \cdot M\boldsymbol{x} \\ \tau \end{pmatrix} = \begin{pmatrix} M \cdot \tau \boldsymbol{x} \\ \tau \end{pmatrix} = \underbrace{\left(\begin{array}{c|c} M & 0 \\ \hline 0 & 1 \end{array}\right)}_{=:\,M'} \cdot \underbrace{\begin{pmatrix} \tau \boldsymbol{x} \\ \tau \end{pmatrix}}_{=\xi}$$

Fazit: Lineare Abbildungen sind auch projektiv linear.

affine Abbildungen:

$$\boldsymbol{x} \mapsto \boldsymbol{y} := M \cdot \boldsymbol{x} + \boldsymbol{t}\,, \qquad \boldsymbol{x} \in \mathbb{R}^n$$

$$\xi = \begin{pmatrix} \tau \boldsymbol{x} \\ \tau \end{pmatrix} \mapsto \boldsymbol{\eta} = \begin{pmatrix} \tau \boldsymbol{y} \\ \tau \end{pmatrix} = \begin{pmatrix} \tau \cdot (M\boldsymbol{x} + \boldsymbol{t}) \\ \tau \end{pmatrix} = \underbrace{\left(\begin{array}{c|c} M & \boldsymbol{t} \\ \hline 0 & 1 \end{array}\right)}_{=:\,M'} \cdot \underbrace{\begin{pmatrix} \tau \boldsymbol{x} \\ \tau \end{pmatrix}}_{\xi}$$

Fazit: Affine Abbildungen werden projektiv linear!

Standard-Parallelprojektion:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} \tau u \\ \tau v \\ \tau \end{pmatrix} = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \cdot \begin{pmatrix} \tau x \\ \tau y \\ \tau z \\ \tau \end{pmatrix}$$

Fazit: Die Standard-Parallelprojektion ist projektiv linear (klar, da ohnehin linear).

Standard-Perspektivische Projektion:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \frac{z_p}{z} \cdot \begin{pmatrix} x \\ y \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} z \cdot u \\ z \cdot v \end{pmatrix} = \begin{pmatrix} z_p \cdot x \\ z_p \cdot y \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} z_p & 0 & 0 & 0 \\ 0 & z_p & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} \tau x \\ \tau y \\ \tau z \\ \tau \end{pmatrix} = \begin{pmatrix} \tau z \cdot u \\ \tau z \cdot v \\ \tau z \end{pmatrix} =: \begin{pmatrix} \tau' \cdot u \\ \tau' \cdot v \\ \tau' \end{pmatrix}$$

Fazit: Auch die Standard-Perspektivische Projektion wird projektiv linear!

Bemerkungen 4.4:

1. In OpenGL können die 4×4-„view matrix“ (im Wesentlichen M'_{sw}) und die Projektion ($\triangleq M'_{ps}$) auf verschiedene Arten aufgebaut werden:
   - automatisch, z. B. durch Angabe von $\boldsymbol{a}$, $\boldsymbol{l}$, up bzw. $-z_p$, $\underline{u}$, $\overline{u}$, $\underline{v}$ und $\overline{v}$,
   - durch Zusammensetzen einfacherer Transformationsmatrizen oder
   - „von Hand“ durch Angabe der Matrixeinträge

   Die beiden letzten Möglichkeiten erlauben auch nicht-orthogonale („oblique“) Parallelprojektionen.
2. In der „modelview matrix“ wird die „view matrix“ mit den Modellierungstransformationen (Abschnitt 4.3) zusammengefasst.
3. Die „viewport transformation“ vermittelt den Übergang von der Projektionsebene zu Gerätekoordinaten.
4. Auch die n-Koordinate (Information über räumliche Tiefe) wird durch die eigentliche Projektion geführt (4×4-Matrix statt 3×4).

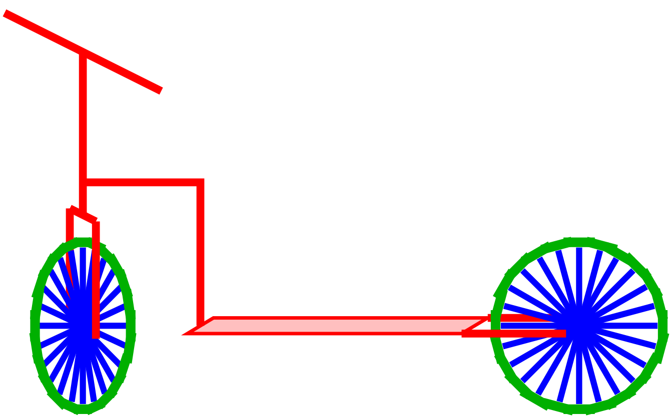
   (mehr dazu in Kapitel 5 und 6)

4.3 Transformationen in der Modellierung

hierarchische Modellierung eines Tretrollers (stark vereinfacht):

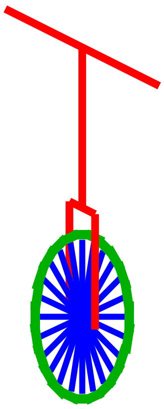
Der **Tretroller** besteht aus

- einer (um eine vertikale Achse drehbaren) „Lenkergruppe“,
- fünf starren (mit Quadern modellierten) Bauteilen und
- dem (um eine horizontale Achse drehbaren) Hinterrad.

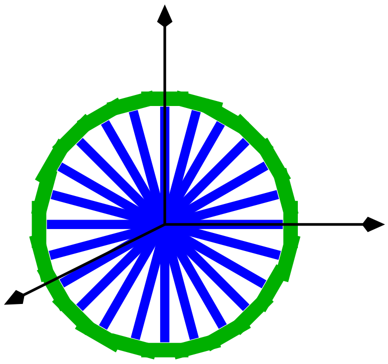


Die **Lenkergruppe** besteht aus

- dem (um eine horizontale Achse drehbaren) Rad und
- fünf starren (mit Quadern modellierten) Bauteilen für Gabel, Lenkerstange, usw.



Das **Rad** besteht aus einer vorgegebenen Anzahl von (jeweils um einen geeigneten Winkel gedrehten) „Speiche/Reifen-Einheiten".

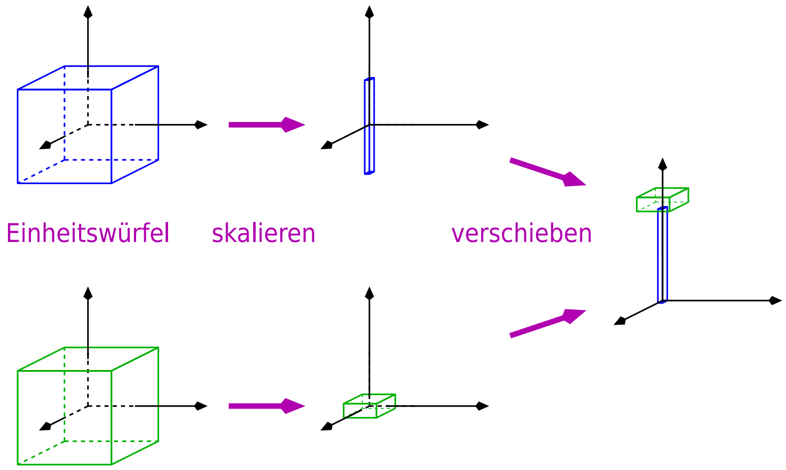


Eine **Speiche/Reifen-Einheit** besteht aus

- einer Speiche und
- einem Stück des Reifens

(beide modelliert durch Quader).

Ein **Quader** kann durch geeignete Streckung aus einem **Einheitswürfel** $[-0{,}5; 0{,}5]^3$ erzeugt werden.



Bei der Projektion auf 2D müssen die folgenden Transformationen auf die Ecken $\boldsymbol{P} \in \mathbb{R}^3$ von Einheitswürfeln angewandt werden

(ausgehend von projektiven Koordinaten $\boldsymbol{P}' = \begin{pmatrix} \tau \boldsymbol{P} \\ \tau \end{pmatrix}$ erhält man auch projektive 2D-Koordinaten $\boldsymbol{u}' = \begin{pmatrix} \tau' u \\ \tau' v \\ \tau' \end{pmatrix}$):

- wenn der Einheitswürfel dargestellt werden soll:

$$\boldsymbol{u}' = \boldsymbol{M}'_{\text{proj}} \cdot \underbrace{\boldsymbol{P}'}_{\text{Ecke des Würfels}}$$

  mit

  - $\boldsymbol{M}'_{\text{proj}}$: Projektionsmatrix 3D → 2D

- wenn der Würfel in ein Element der Speiche/Reifen-Einheit (Speiche oder Stück des Reifens) transformiert wird:

$$\boldsymbol{u}' = \boldsymbol{M}'_{\text{proj}} \cdot \underbrace{\boldsymbol{M}'_{\text{tr\_sr}_\text{s}} \cdot \boldsymbol{M}'_{\text{sc\_sr}_\text{s}}}_{\text{Transformation zur Speiche}} \cdot \underbrace{\boldsymbol{P}'}_{\text{Würfel}} \quad \text{bzw.}$$

$$\boldsymbol{u}' = \boldsymbol{M}'_{\text{proj}} \cdot \underbrace{\boldsymbol{M}'_{\text{tr\_sr}_\text{r}} \cdot \boldsymbol{M}'_{\text{sc\_sr}_\text{r}}}_{\text{Transformation zum Reifenstück}} \cdot \underbrace{\boldsymbol{P}'}_{\text{Würfel}}$$

  mit

  - $\boldsymbol{M}'_{\text{sc\_sr}_\text{s}}$ bzw. $\boldsymbol{M}'_{\text{sc\_sr}_\text{r}}$: (Diagonal-)Streckungsmatrix („scale“) für die Speiche bzw. das Reifenstück
  - $\boldsymbol{M}'_{\text{tr\_sr}_\text{s}}$ bzw. $\boldsymbol{M}'_{\text{tr\_sr}_\text{r}}$: anschließende Verschiebung („translate“)

- wenn die Speiche/Reifen-Einheit anschließend in ein Rad eingebaut wird:

$$u' = M'_{\text{proj}} \cdot \underbrace{M'_{\text{rot\_rad}}}_{\text{Einbau in Rad}} \cdot \underbrace{M'_{\text{tr\_sr}} \cdot M'_{\text{sc\_sr}} \cdot P'}_{\text{Speiche/Reifen-Einheit}}$$

  mit

  - $M'_{\text{rot\_rad}}$: Rotationsmatrix („rotate“) um die (Naben-) Achse

- wenn das Rad hinten in den Tretroller eingebaut wird:

$$u' = M'_{\text{proj}} \cdot \underbrace{M'_{\text{tr\_hr}} \cdot M'_{\text{rot\_hr}}}_{\text{Einbau in Roller}} \cdot \underbrace{M'_{\text{rot\_rad}} \cdot M'_{\text{tr\_sr}} \cdot M'_{\text{sc\_sr}} \cdot P'}_{\text{Rad}}$$

  mit

  - $M'_{\text{rot\_hr}}$: Drehung des Hinterrades
  - $M'_{\text{tr\_hr}}$: anschließende Verschiebung an die Position im Roller

- wenn das „Trittbrett“ in den Tretroller eingebaut wird:

$$u' = M'_{\text{proj}} \cdot \underbrace{M'_{\text{tr\_brett}} \cdot M'_{\text{sc\_brett}}}_{\text{Einbau als Trittbrett}} \cdot \underbrace{P'}_{\text{Würfel}}$$

  mit

  - $M'_{\text{sc\_brett}}$: Streckungsmatrix für das Trittbrett
  - $M'_{\text{tr\_brett}}$: anschließende Verschiebung an die Position im Roller

generelles Vorgehen: ist eine „Struktur“ (z. B. Tretroller) aus „Unterstrukturen“ (z. B. Lenkergruppe, Quader für Trittbrett usw., Hinterrad) aufgebaut, dann

- vor dem „Aufruf“ jeder Unterstruktur die zum Einbau erforderlichen Transformationen setzen,
- die Unterstruktur aufrufen (und dabei ggf. rekursiv genauso verfahren),
- anschließend die „Einbau“transformationen wieder zurücksetzen

⇒ Verwaltung der Transformationsmatrizen mittels **Stack**

Bemerkung 4.5: Damit am Ende jede Ecke der Szene mit einer einzigen Matrix-Vektor-Multiplikation transformiert werden kann, werden bei **OpenGL** stets alle „bislang bekannten“ Matrizen aufmultipliziert.

Beispielsweise liegt vor dem Einbau einer Speiche/Reifen-Einheit in das Hinterrad die Matrix

$$M' := M'_{\mathrm{proj}} \cdot M'_{\mathrm{tr\_hr}} \cdot M'_{\mathrm{rot\_hr}} \cdot M'_{\mathrm{rot\_rad}}$$

vor, innerhalb der Speiche/Reifen-Einheit wird sie durch

$$M' \cdot M'_{\mathrm{tr\_sr_s}} \cdot M'_{\mathrm{sc\_sr_s}}$$

ersetzt.

Die auf eine Unterstruktur anzuwendenden Transformationen sind daher **in umgekehrter Reihenfolge** anzugeben.

Tretroller/roller.c

```c
// eine Speiche und ein dazu gehöriges Stück des Reifens zeichnen
void zeichneSpeicheReifen(void)
{
  // für die Speiche die Farbe auf Blau setzen
  // (Beiträge an Rot, Grün, Blau)
  glColor3f(0.0, 0.0, 1.0);

  // alte Transformationsmatrix sichern
  glPushMatrix();
  // aus einem Einheitswürfel [-0,5; 0,5]³ durch
  // Skalierung eine Speiche erzeugen und danach(!)
  // verschieben, so dass sie im Ursprung anliegt
  glTranslated(0.0, 0.5, 0.0);
  glScaled(0.02, 1.0, 0.02);
  glutSolidCube(1.0);
  // alte Transformation wiederherstellen
  glPopMatrix();

  // analog für grünes Stück Reifen
  glColor3f(0.0, 1.0, 0.0);
  glPushMatrix();
  glTranslated(0.0, 1.00, 0.00);
  glScaled(0.3, 0.15, 0.1);
  glutSolidCube(1.0);
  glPopMatrix();
}

// ein aus anzahl_speichen bestehendes Rad zeichnen
void zeichneRad(int anzahlSpeichen)
{
  glPushMatrix();

  // das Rad aus anzahlSpeichen Speichen zusammensetzen
  // (jeweils um 360 / anzahlSpeichen Grad WEITERdrehen)
  for (int k = 0; k < anzahlSpeichen; k++)
    {
      glRotated(360.0 / anzahlSpeichen, 0.0, 0.0, 1.0);
      zeichneSpeicheReifen();
    }

  glPopMatrix();
}
```

```
// Die aus Vordergabel, vertikaler Stange und horizontaler
// Lenkerstange bestehende Lenkergruppe zeichnen
void zeichneLenkergruppe(void)
{
  // Vorderrad zeichnen
  glPushMatrix();
  glScaled(0.6, 0.6, 1.8);
  zeichneRad(16);
  glPopMatrix();

  // roten Rahmen dazu zeichnen
  glColor3f(1.0, 0.0, 0.0);

  // linke Strebe der Radgabel
  glPushMatrix();
  glTranslated(0.0, 0.35, -0.2);
  glScaled(0.1, 0.8, 0.05);
  glutSolidCube(1.0);
  glPopMatrix();

  // vier weitere Quader für rechte und horizontale Strebe
  // der Gabel, vertikale Stange und Lenkerstange

  // …
// Den kompletten Tretroller zeichnen; dabei die Lenkergruppe
// und das Hinterrad je nach verstrichener Zeit (angegeben in
// minuten) drehen.
void zeichneRoller(double minuten)
{
  // gedrehte Lenkergruppe zeichnen
  glPushMatrix();
  glRotated(minuten * -0.2, 0.0, 1.0, 0.0);
  zeichneLenkergruppe();
  glPopMatrix();

  // Trittbrett zeichnen
  glPushMatrix();
  glTranslated(1.8, 0.0, 0.0);
  glScaled(2.0, 0.1, 0.4);
  glutSolidCube(1.0);
  glPopMatrix();

  // vier weitere Quader für Befestigung an Lenkergruppe
  // und Hinterradgabel zeichnen

  // …
```



```
  // gedrehtes Hinterrad zeichnen
  glPushMatrix();
  glTranslated(3.65, 0.0, 0.0);
  glRotated(minuten * -1.0, 0.0, 0.0, 1.0);
  glScaled(0.6, 0.6, 1.2);
  zeichneRad(32);
  glPopMatrix();
}
```