

---

# Computergraphik II

Prof. Dr. Andreas Kolb

Fachgruppe Computergraphik und Multimediasysteme

Universität Siegen – Fachbereich 12

Version: 7. Januar 2005

**Klassifikation von Systemen aus Bewegungssicht:** Man unterscheidet

**Statische Systeme:** Geometrien und Beobachter sind in Lage und Größe fix

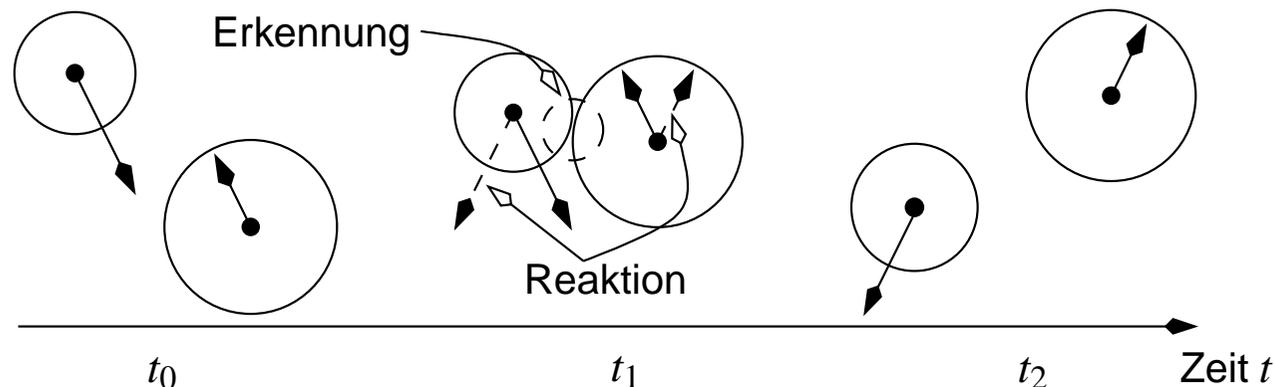
**Deterministische Systeme:** Lage/Größe variabel, aber a-priori festgelegt (Animation)

**Dynamische Systeme:** Freie Bewegung der Objekte und des Beobachters möglich

**Kollision** in deterministischen und dynamischen Systemen muß real wirken

**Kollisionsverarbeitung:** 1. *Kollisionserkennung*, d.h. Ermittlung der Kollisionsinformation

2. *Kollisionsreaktion* der (dynamischen) Objekte auf Kollisionsereignis

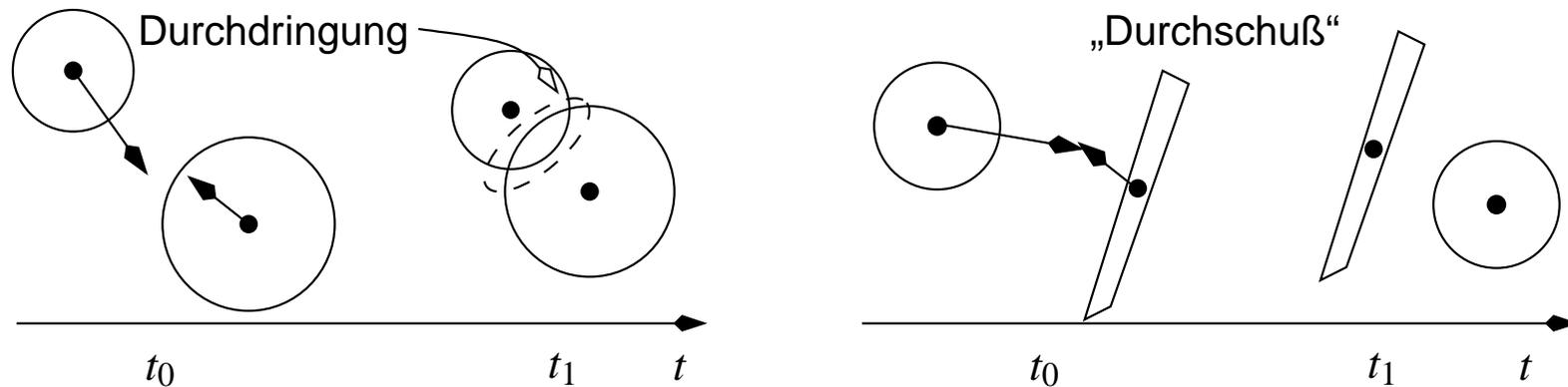


**Kollisionsinformation:** 1. Boolesche Aussage: Einfache Kollisionsreaktion

2. Kollisionspunkte und -normalen bei komplexer Kollisionsreaktion

## Grundsätzliche Konzepte zur Kollisionserkennung

**3D, zeitdiskret:** Test zu festen Zeitpunkten; exakter Kollisionszeitpunkt wird nicht erkannt!



**4D:** Echte Überprüfung in vier Dimensionen  $\{x, y, z, t\} \Rightarrow$  komplex!

**Brut Force:** Komplexität bei  $n$  Objekten:  $n(n-1)/2 \Rightarrow O(n^2)$  Kollisionsprüfungen

**Beschleunigung:** Nutzung von Koherenzen (räumlich, zeitlich) oder *Bounding Volumes*

## Abstrakter Algorithmus zur Kollisionserkennung:

```
forall moving objects m do
  forall objects o != m do
    if isClose(o, m) do           // schnelle, grobe, konserv. Abschätzung
      checkCollision(o, m);      // (aufwendige) Detailprüfung
```



## Kollisionsgeometrie

Kollisionsverarbeitung je nach geforderter Kollisionsinformation und Genauigkeit

1. auf Basis der *exakten Objektgeometrien*

- + Exaktheit, selten Ausnutzung von spez. Geometrie, z.B. Kugel-Kollision (→ schnelle Berechnung)
- Kollisionserkennung für alle Geometrie-Kombinationen, häufig sehr aufwendige Berechnung erforderlich

2. auf Basis vereinfachter, umhüllender Geometrien (*Bounding Volumes*)

- + Einheitliches Modell der Kollisionserkennung
- Ungenauigkeit

# 10.1 Einfache Kollisionsreaktionen



**Aufgabe:** Anpassung der Objekt-Geschwindigkeit, so dass keine weitere Durchdringung

**Randbedingungen:** • es kollidieren genau **zwei** Objekte  $O_1, O_2$

- von beiden Objekte ist der *Geschwindigkeitsvektor*  $\vec{v}_i$  bekannt

**Ansatz Bewegungs-Unterbindung: Kollisionsinformation:** Boolesche Aussage

**Kollisionsreaktion:** • behalte aktuelle Position bzw. letzte Nichtkollisions-Position bei

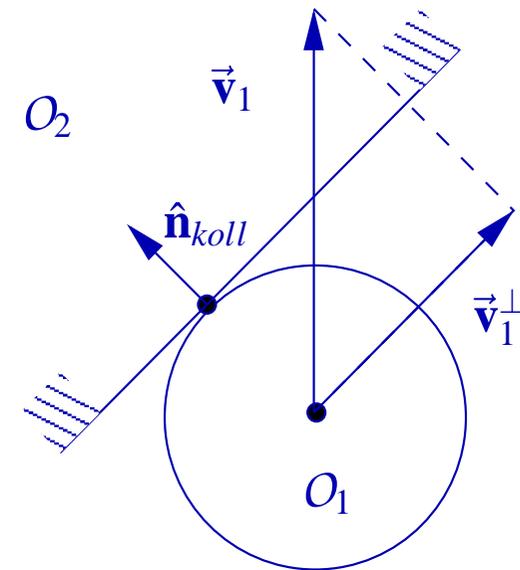
- setze aktuelle Geschwindigkeiten auf null:  $\vec{v}_1 = \vec{v}_2 = 0$

**Ansatz Entlanggleiten:**

**Kollisionsinformation:** Kollisionsnormale  $\vec{n}_{koll}$

**Kollisionsreaktion:** Entfernen des Geschwindigkeitsanteils  $\vec{v}^{\parallel}$  parallel der Kollisionsnormale:

$$\vec{v}_i^{\perp} = \vec{v}_i - (\vec{v}_i \cdot \hat{n}_{koll}) \hat{n}_{koll}$$



## Elastischer Stoß

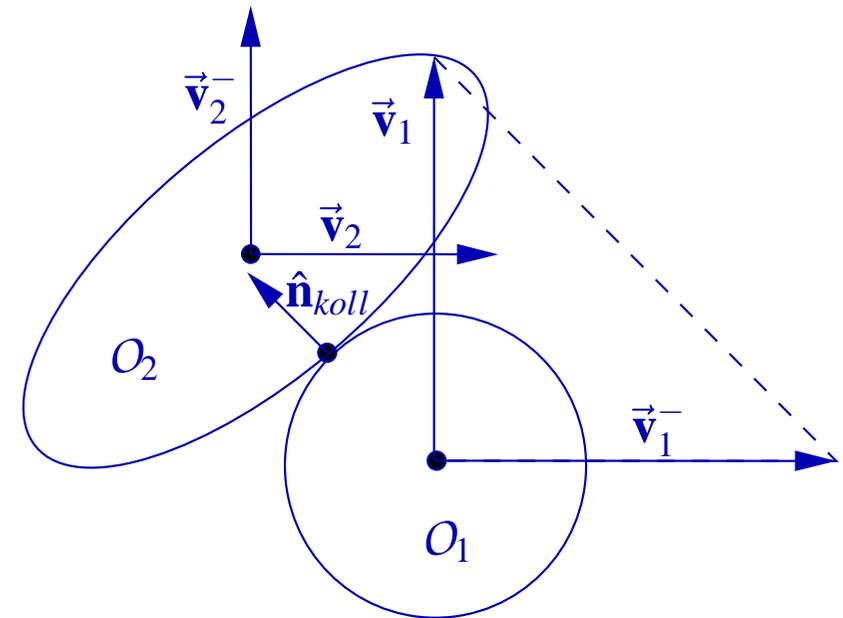
**Vereinfachung:** Vernachlässigung

- der Objektmasse
- der Umwandlung von linearer Bewegung in Rotationsbewegung

**Idee:** Reflexion des Geschwindigkeitsvektors an Kollisionsnormale:

$$\vec{v}_i^- = \vec{v}_i - 2(\vec{v}_i \cdot \hat{n}_{koll})\hat{n}_{koll}$$

**Berücksichtigung** von Masse und Rotation (siehe VR)



**Beachte:** Alle Ansätze sind physikalisch nicht korrekt, da keine Energieerhaltung

# 10.2 Boolesche Kollisionserkennung mit Bounding Volumens

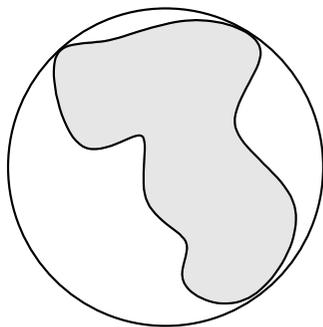
**Definition:**  $V$  ist ein *Bounding Volume* einer Geometrie  $G$  falls

1.  $V$  ist konvex
2.  $V$  „umfaßt“  $G$ , d.h.  $V \supseteq G$

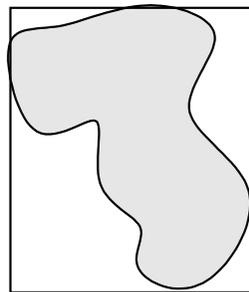
**Zielsetzung** zur Wahl von Bounding Volumen

1. Effiziente Kollisionsberechnung
2. Optimale Objektanpassung, d.h.  $V \setminus G$  soll minimal sein
3. Effiziente Ermittlung für gegebene Objekte

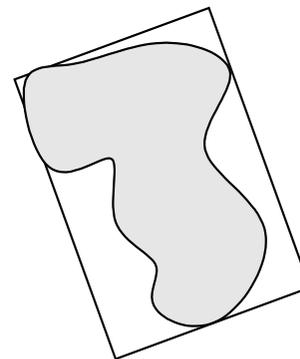
**Beispiele:**



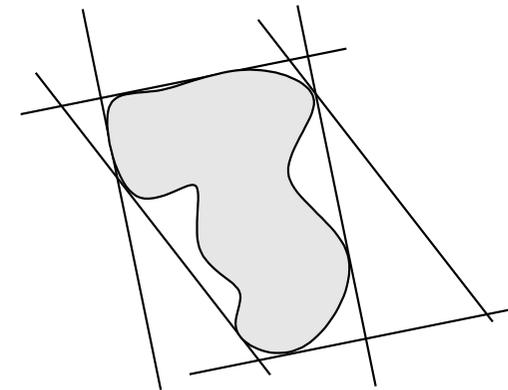
Sphäre



Axis-Aligned  
Bounding Box



Object Aligned  
Bounding Box



Discrete Orientation  
Polytop ( $k$ -dop)

## 10.2.1 Bounding Boxes



**Gegeben:** Zwei Quader  $Q^1, Q^2$  mit Mittelpunkt  $\mathbf{C}^i$ , orthonormalen Achsen  $\hat{\mathbf{e}}_j$ ,  $j = 1, 2, 3$  und Ausdehnungen  $r_j^i$ ,  $j = 1, 2, 3$ :

$$Q^i = \left\{ \mathbf{C}^i + \sum_{j=1}^3 \alpha_j \hat{\mathbf{e}}_j^i : \alpha_j \in [-r_j^i, r_j^i] \right\} \quad i = 1, 2$$

**Gesucht:** Boolesche Aussage, ob die Quader sich durchdringen

**Axis Aligned Box (AAB):**  $\hat{\mathbf{e}}_j$  sind Weltkoordinatenachsen

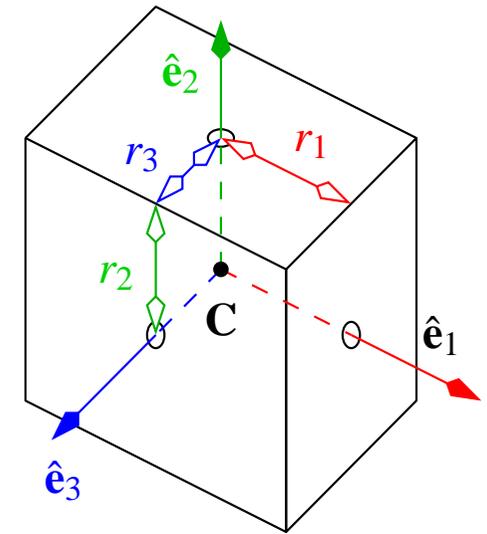
Achtung: AAB's ändern sich bei Objekt-Rotation!

**Object Aligned Box (OAB):**  $\hat{\mathbf{e}}_j$  sind am Objekt ausgerichtet

**Separierende Ebene:** Konvexe Geometrien durchdringen sich nicht, wenn es eine *separierende Ebene* zwischen ihnen gibt (Skizze!)

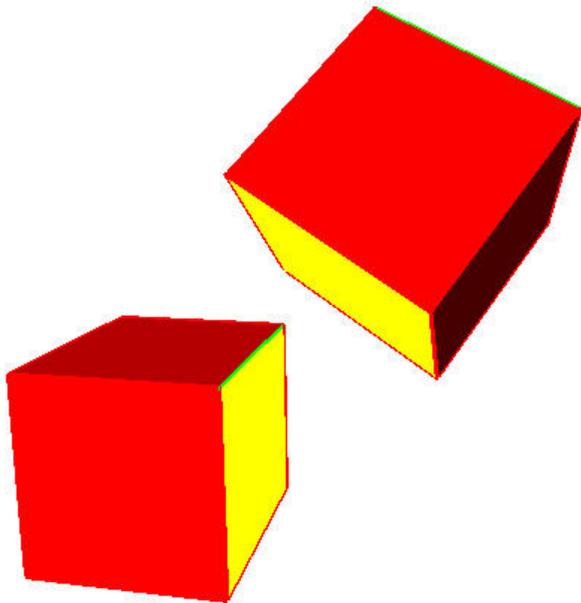
**Speziell:** Für OABs genügt es folgende Flächen auf Separation zu testen:

1. *Seitenfläche* einer Box
2. Ebene parallel zu *zwei Kantenvektoren*

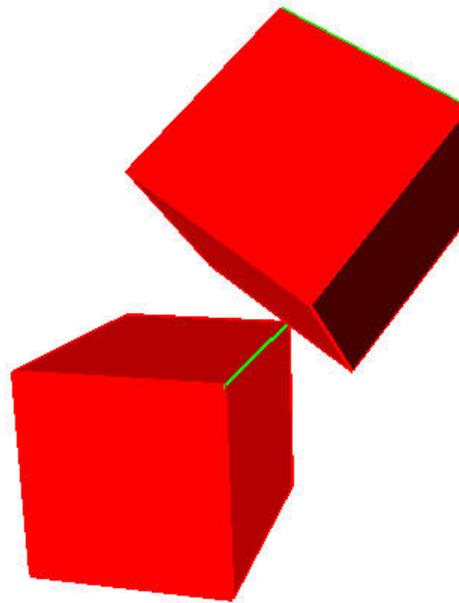


## Separierende Ebenen für OABs

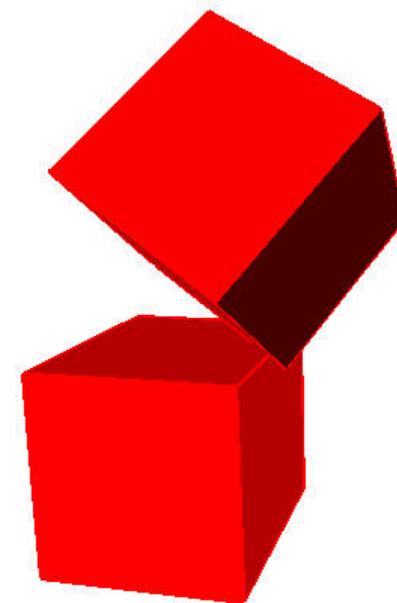
- Demo-Programm:



Kantenpaar & Seitenfläche  
separieren



Kantenpaar separiert



Durchdringung

- Es genügt **eine** separierende Ebene zu finden, um eine Kollision auszuschließen.



## Separierende Ebenen für AABs

**Beobachtung:** Kantenvektorpaare erzeugen keine neuen Ebenen  $\implies$  teste nur Seitenflächen

**Kollisions-Erkennung:** Für die Seitenflächen ergibt sich:

$$(x_{max}^1 < x_{min}^2) \vee (x_{max}^2 < x_{min}^1) \dots$$

wobei  $x_{min}^i = c_x^i - r_1^i$ ,  $x_{max}^i = c_x^i + r_1^i$  mit  $\mathbf{C}^i = (c_x^i, c_y^i, c_z^i)$

## Bemerkung:

- für zwei AABs sind maximal sechs Bedingungen zu prüfen (zwei pro Koordinatenachse)
- für OABs kommen zusätzlich maximal neun für die  $3 \times 3$  Kantenvektorpaare hinzu.
- Der **genaue Kollisionszeitpunkt** kann nicht ermittelt werden  $\implies$  arbeite mit  $\varepsilon$ -Umgebungen

$$(x_{max}^1 - x_{min}^2 < \varepsilon) \vee (x_{max}^2 - x_{min}^1 < \varepsilon) \dots$$

# 10.2.1 Bounding Boxes



## Separierende Ebenen für OABs

**Idee:** Projiziere Quader auf die Normale  $\hat{\mathbf{n}}$  der potentiell separierenden Fläche

$$\text{Ebene separiert} \iff \Delta > p^1 + p^2$$

**Projektion eines Quaders:** Für  $\vec{\mathbf{d}} = \mathbf{C}_1 - \mathbf{C}_2$  gilt:

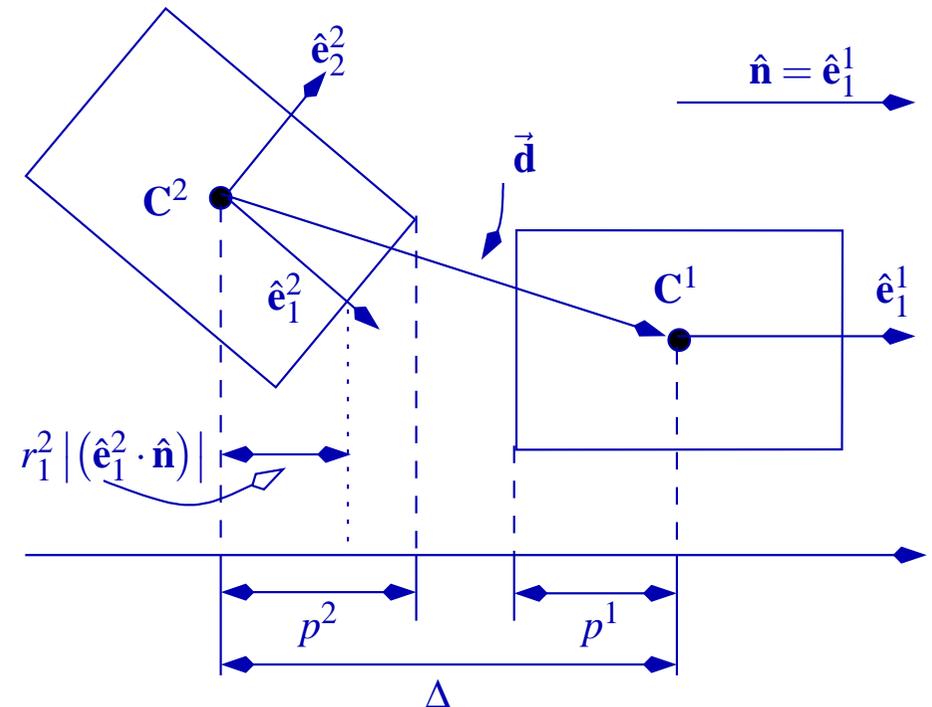
$$\Delta = \left| (\vec{\mathbf{d}} \cdot \hat{\mathbf{n}}) \right|, \quad \text{hier } \hat{\mathbf{n}} = \hat{\mathbf{e}}_1^1$$

**Fall 1:**  $\hat{\mathbf{n}}$  ist Seitenflächennormale:

$$\text{z.B. } \hat{\mathbf{n}} = \hat{\mathbf{e}}_1^1 \implies p^1 = r_1^1$$

**Fall 2:**  $\hat{\mathbf{n}}$  liegt beliebig zum Quader:

$$p^i = \sum_{j=1}^3 r_j^i \left| (\hat{\mathbf{e}}_j^i \cdot \hat{\mathbf{n}}) \right|$$



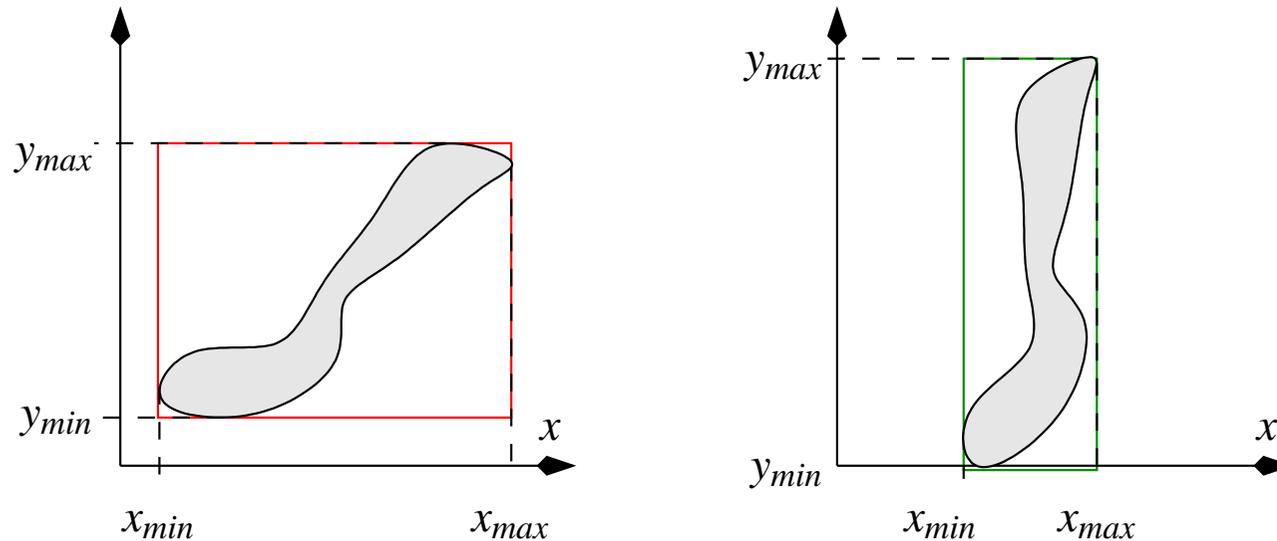
# 10.2.1 Bounding Boxes



## Bewertung von AAB

**Kollisionsberechnung:** Sehr effizient über 6 Vergleiche

**Objektanpassung:** Schlecht, da AAB von der Lage der Objekte im Raum abhängen



**Ermittlung für gegebene Polygonmodelle** mit Vertices  $\{\mathbf{V}^i\}_{i=1}^n$  sehr einfach:

$$x_{min} = \min_{i=1, \dots, n} \{v_x^i\}, \quad x_{max} = \max_{i=1, \dots, n} \{v_x^i\}, \quad \dots, \quad z_{max} = \max_{i=1, \dots, n} \{v_z^i\}$$



# 10.2.1 Bounding Boxes



## Bewertung von OAB

**Kollisionsberechnung:** Effizient über 15 Tests für potentiell separierende Ebenen

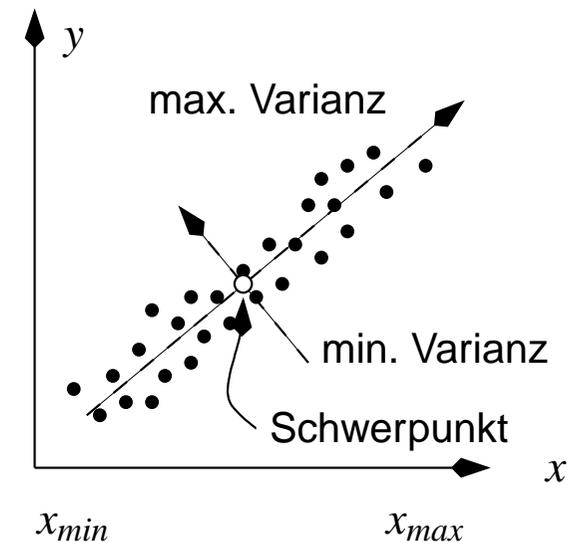
**Objektanpassung:** Gut, da OAB den Objekten lokal angepaßt werden

**Ermittlung für gegebene Polygonmodelle** aufwendiger

**Idee der Ermittlung:** Für Polygonmodell mit Vertices  $\{\mathbf{V}_i\}_{i=1}^n$ :

1. Bestimme Schwerpunkt  $\mathbf{M} = \frac{1}{n} \sum_{i=1}^n \mathbf{V}_i$   
(besser: Schwerpunkt der konvexen Hülle)
2. Bestimme Richtungen der *max.* bzw. *min.* Varianz  $\rightarrow$  Achsen  $\hat{\mathbf{e}}_1, \vec{\mathbf{e}}_2$   
(besser: nur Punkte der konvexen Hülle)
3. Wähle max. Ausdehnung z.B. für Achse  $\hat{\mathbf{e}}_j$ :

$$r_j := \max_i \left| \left\{ \left( \hat{\mathbf{e}}_j \cdot (\mathbf{V}_i - \mathbf{M}) \right) \right\} \right|$$



## 10.2.2 Bounding Volumen Hierarchien



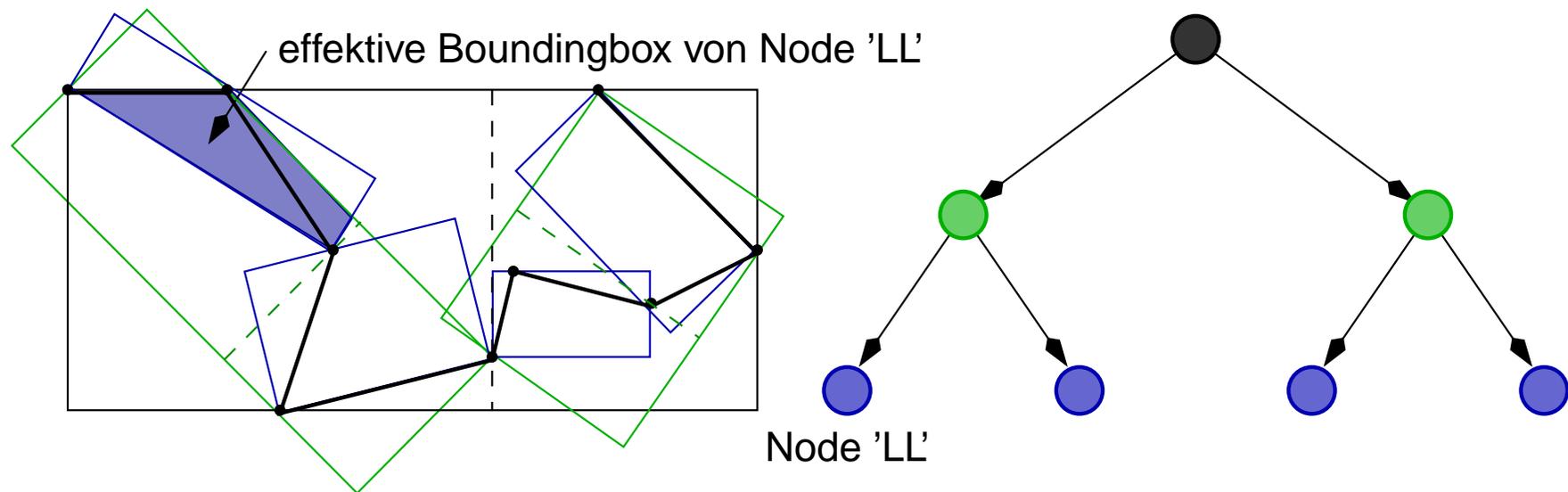
**Ziel:** Bessere Anpassung an Objekt durch mehrere, kleine Bounding Volumen

⇒ Hierarchie von Bounding Volumen

**Einschränkung:** Balance zwischen Kosten für

- Kollisionstest für Geometrien in einem Volumen
- Abarbeitung/Traversierung der Hierarchie inkl. Kollisions-Test für Bounding Volumen

**Beispiel OAB Hierarchie:** Sukzessive Unterteilung in aktuell max. Ausdehnungsrichtung



**Beachte:** Effektiv wird *Schnittmenge* der Volumen im Pfad betrachtet

# 10.2.2 Bounding Volumen Hierarchien



## Bounding Boxen und mehrgliedrige Modelle

**Ziel:** Kollisionserkennung für mehrgliedrige Modelle inkl. weiterer Geometrien (Layer-Modell)

**Ansatz:** Bestimme AAB's in *lokalen Koordinaten*

- Beachte:**
1. Kollisionstest immer in *einem* Koordinatensystem, z.B. Viewkoordinaten (VC)
  2. Translation und Rotation erhalten die rechten Winkel, Skalierung i.a. nicht!

**Explizite Transformation** der lokalen AAB in View-Koord. mit Curr. Transformationmatrix  $M$

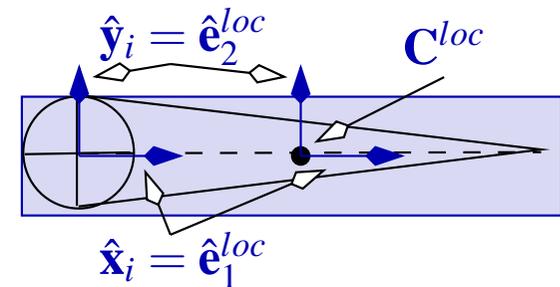
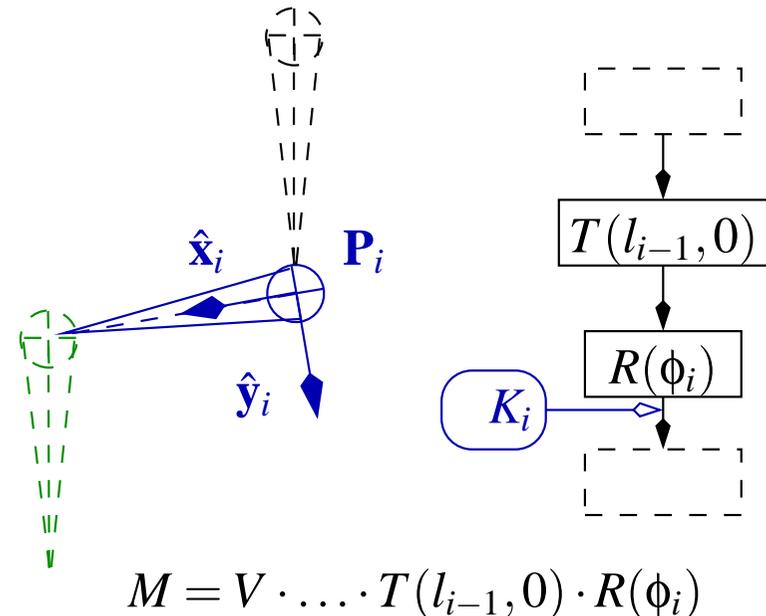
**Lokale Koordinaten:**  $C^{loc}, \hat{e}_1^{loc}, \hat{e}_2^{loc}, \hat{e}_3^{loc}$

**View-Koordinaten** falls  $M$  ohne Skalierung:

$$C^{VC} = M \cdot C^{loc}$$

$$\hat{e}_i^{VC} = M \cdot \hat{e}_i^{loc}, i = 1, 2, 3$$

**2D-Beispiel:**



## 10.3 Berechnung von Kollisionspunkten



- Beachte:** 1. zur Ermittlung von Kollisionspunkten und -normalen eignen sich *einfache implizite Geometrien* i.a. besser als Polygone
2. Kollisionspunktberechnung beruht auf *Abstandberechnung*

**Beispiel:** Kollisionspunkte zweier Kugeln

**Gegeben:** Kugeln  $S_i$  mit Basisgeometrie Punkt  $\mathbf{M}_i$  und Radius  $s_i$ :

$$S_i = \{\mathbf{P} : \|\mathbf{P} - \mathbf{M}_i\| = s_i\}, i = 1, 2$$

**Gesucht:** Kollisionserkennung, Kollisionspunkt(e) und Kollisionsnormale

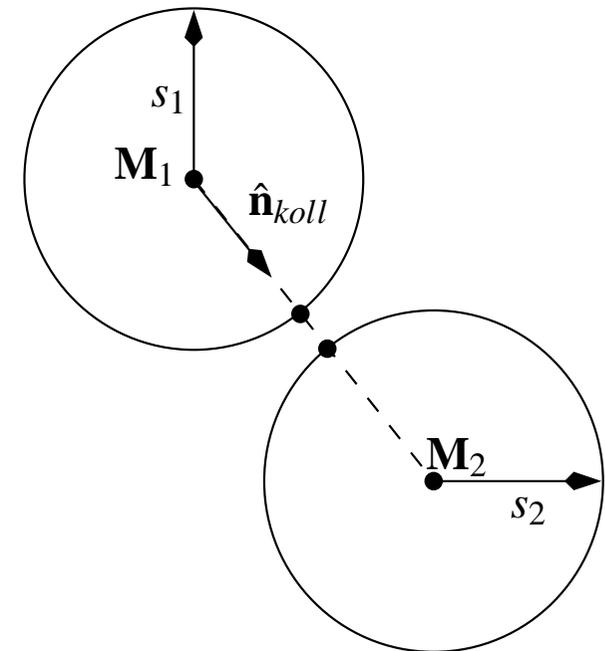
**Lösung:** 1. Abstand der beiden Kugeln:

$$d(S_1, S_2) = \|\mathbf{M}_1 - \mathbf{M}_2\| - (s_1 + s_2)$$

2. Kollisionserkennung:  $d(S_1, S_2) \stackrel{?}{\approx} 0$

3. Kollisionsnormale:  $\vec{\mathbf{n}}_{koll} = (\mathbf{M}_2 - \mathbf{M}_1) / \|\mathbf{M}_2 - \mathbf{M}_1\|$

4. Kollisionspunkt:  $\mathbf{K} = \mathbf{M}_1 + s_1 \vec{\mathbf{n}}_{koll} \approx \mathbf{M}_2 - s_2 \vec{\mathbf{n}}_{koll}$



# 10.3.1 Abstand zweier linearer Komponenten



**Aufgabe:** Bestimme Abstand zweier *linearer Komponenten*, d.h. Gerade, Strecke, Strahl:

$$L_i : \mathbf{L}_i(\alpha) = \mathbf{V}_i + \alpha_i \vec{\mathbf{l}}_i, \quad \alpha_i \in I_i \subset \mathbb{R}, \quad I_i \in \{\mathbb{R}, \mathbb{R}_+, [0, 1]\}$$

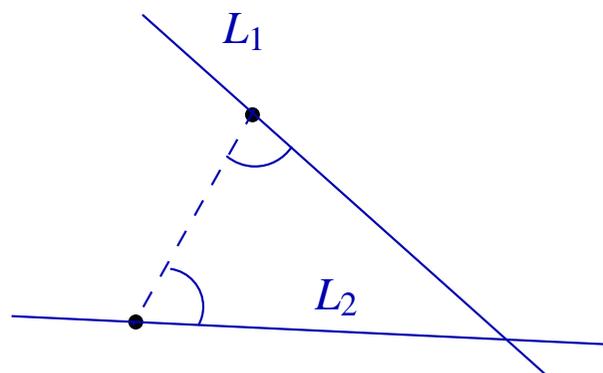
**Gegeben:** Lineare Komponenten  $L_1, L_2$

**Gesucht:** Punkte mit kürzestem Abstand:

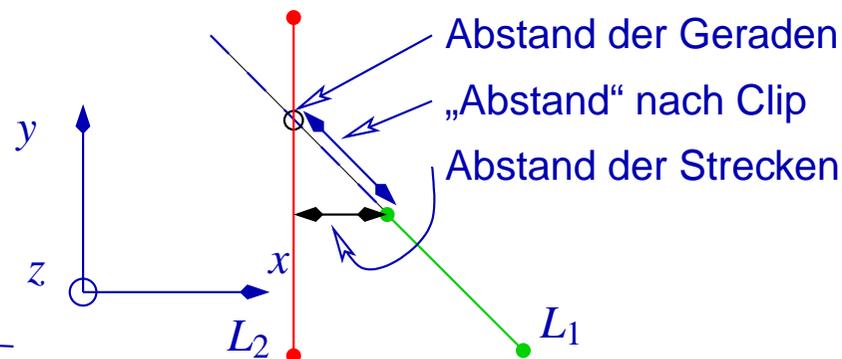
$$(\alpha_1^P, \alpha_2^P) \in I_1 \times I_2 \text{ mit } \|\mathbf{L}_1(\alpha_1^P) - \mathbf{L}_2(\alpha_2^P)\| = \min_{\alpha_1 \in I_1, \alpha_2 \in I_2} \{\|\mathbf{L}_1(\alpha_1) - \mathbf{L}_2(\alpha_2)\|\}$$

**Beobachtung:** •  $L_1, L_2$  Geraden ( $I_1 = I_2 = \mathbb{R}$ )  $\Rightarrow (\mathbf{L}_1(\alpha_1^P) - \mathbf{L}_2(\alpha_2^P)) \perp L_i$

•  $L_1$  oder  $L_2$  keine Gerade: Einfaches Clipping liefert i.a. falsches Ergebnis



Abstand zweier Gerade



Strecken in  $x - y$ -Ebene

# 10.3.1 Abstand zweier linearer Komponenten



## Abstand zweier Geraden

**Gegeben:** Lineare Komponenten  $L_1, L_2$  mit  $I_1 = I_2 = \mathbb{R}$

**Lösung:** Die beiden „Senkrecht“-Bedingungen liefern:

$$\begin{aligned} \left( (\mathbf{L}_1(\alpha_1^P) - \mathbf{L}_2(\alpha_2^P)) \cdot \vec{\mathbf{l}}_i \right) &= \left( (\mathbf{V}_1 + \alpha_1^P \vec{\mathbf{l}}_1 - \mathbf{V}_2 - \alpha_2^P \vec{\mathbf{l}}_2) \cdot \vec{\mathbf{l}}_i \right) = 0, \quad i = 1, 2 \\ \iff \alpha_1^P (\vec{\mathbf{l}}_1 \cdot \vec{\mathbf{l}}_i) - \alpha_2^P (\vec{\mathbf{l}}_2 \cdot \vec{\mathbf{l}}_i) &= ((\mathbf{V}_2 - \mathbf{V}_1) \cdot \vec{\mathbf{l}}_i), \quad i = 1, 2 \end{aligned}$$

Dies ergibt ein *lineares Gleichungssystem* für die  $\alpha_i^P$ :

$$\text{Mit } b_{ij} = (\vec{\mathbf{l}}_i \cdot \vec{\mathbf{l}}_j) : \quad \begin{pmatrix} b_{11} & -b_{12} \\ -b_{12} & b_{22} \end{pmatrix} \begin{pmatrix} \alpha_1^P \\ \alpha_2^P \end{pmatrix} = \begin{pmatrix} ((\mathbf{V}_2 - \mathbf{V}_1) \cdot \vec{\mathbf{l}}_1) \\ ((\mathbf{V}_1 - \mathbf{V}_2) \cdot \vec{\mathbf{l}}_2) \end{pmatrix}$$

Die Matrix ist regulär, wenn  $\vec{\mathbf{l}}_i$  nicht parallel sind (sonst ein  $\alpha_i^P$  frei wählbar)

$$\text{Lösung:} \quad \begin{pmatrix} \alpha_1^P \\ \alpha_2^P \end{pmatrix} = \frac{1}{b_{11}b_{22} - b_{12}^2} \begin{pmatrix} b_{22} & b_{12} \\ b_{12} & b_{11} \end{pmatrix} \begin{pmatrix} ((\mathbf{V}_2 - \mathbf{V}_1) \cdot \vec{\mathbf{l}}_1) \\ ((\mathbf{V}_1 - \mathbf{V}_2) \cdot \vec{\mathbf{l}}_2) \end{pmatrix}$$



# 10.3.1 Abstand zweier linearer Komponenten



## Abstand zweier Strecken

**Gegeben:** Lineare Komponenten  $L_1, L_2$  mit  $I_1 = I_2 = [0, 1]$

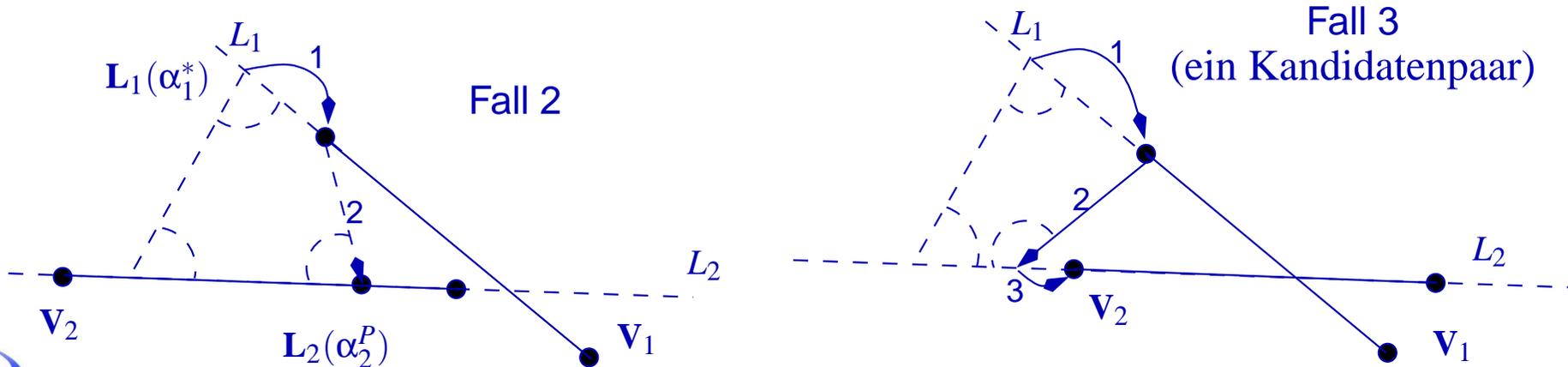
**Lösung:** Abstandsberechnung der zugehörigen **Geraden** liefert  $\alpha_1^*, \alpha_2^* \in \mathbb{R}$

**Fall 1:**  $\alpha_i^* \in [0, 1]$ : dann  $\alpha_i^P = \alpha_i^*$  und  $\mathbf{L}_i(\alpha_i^*)$  sind Punkte kürzesten Abstands.

**Fall 2:** nur ein  $\alpha^*$  außerhalb: sei z.B.  $\alpha_1^* \notin [0, 1]$ , dann gilt:

- $\alpha_1^P = \text{clip}(\alpha_1^*, [0, 1])$
- $\alpha_2^P$  ergibt sich aus Projektion von  $\mathbf{L}_1(\alpha_1^P)$  auf  $L_2$  und evtl. anschließendem Clip.

**Fall 3:** beide  $\alpha_i^*$  außerhalb: Bestimme zwei Kandidatenpaare entspr. Fall 2; wähle dasjenige mit kürzestem Abstand



## 10.3.1 Abstand zweier linearer Komponenten



### Beispiel: Abstandsberechnung zweier Strecken

**Gegeben:**  $\mathbf{L}_1(\alpha_1) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \alpha_1 \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}, \quad \mathbf{L}_2(\alpha_2) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}, \quad \alpha_i \in [0, 1]$

**Lösung:** 1. Ermittlung der nächsten Punkte auf den Geraden:

$$b_{11} = (\vec{\mathbf{I}}_1 \cdot \vec{\mathbf{I}}_1) = 4 \quad b_{22} = (\vec{\mathbf{I}}_2 \cdot \vec{\mathbf{I}}_2) = 5 \quad b_{12} = b_{21} = (\vec{\mathbf{I}}_1 \cdot \vec{\mathbf{I}}_2) = 4$$
$$\left( (\mathbf{V}_2 - \mathbf{V}_1) \cdot \vec{\mathbf{I}}_1 \right) = \left( \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} \right) = 2 \quad \left( (\mathbf{V}_1 - \mathbf{V}_2) \cdot \vec{\mathbf{I}}_2 \right) = \left( \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} \right) = -3$$
$$\implies \begin{pmatrix} \alpha_1^* \\ \alpha_2^* \end{pmatrix} = \frac{1}{5 \cdot 4 - 4^2} \begin{pmatrix} 5 & 4 \\ 4 & 4 \end{pmatrix} \begin{pmatrix} 2 \\ -3 \end{pmatrix} = \begin{pmatrix} -1/2 \\ -1 \end{pmatrix}$$

2. Da  $\alpha_1^*, \alpha_2^* \notin [0, 1]$ , müssen beide Kandidatenpaare erzeugt werden:

## 10.3.1 Abstand zweier linearer Komponenten



### Beispiel: Abstandsberechnung zweier Strecken (Forts.)

2.1. Clippe  $\alpha_1^*$ :  $\alpha_1^P = \text{clip}(\alpha_1^*, [0, 1]) = 0$ . Projiziere  $\mathbf{L}_1(0) = \mathbf{V}_0$  auf  $L_2$ :

$$\text{Es mu\ss\ gelten: } \left( (\mathbf{V}_2 + \alpha_2 \vec{\mathbf{I}}_2 - \mathbf{L}_1(0)) \cdot \vec{\mathbf{I}}_2 \right) = 0 \iff \alpha_2 = \frac{\left( (\mathbf{V}_1 - \mathbf{V}_2) \cdot \vec{\mathbf{I}}_2 \right)}{\left( \vec{\mathbf{I}}_2 \cdot \vec{\mathbf{I}}_2 \right)} = \frac{-3}{5}$$

$$\text{damit: } \alpha_2^P = 0 \text{ und } \|\mathbf{L}_2(0) - \mathbf{L}_1(0)\| = \left\| \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \right\| = \sqrt{3}$$

2.2. Clippe  $\alpha_2^*$ :  $\alpha_2^P = \text{clip}(\alpha_2^*, [0, 1]) = 0$ . Projiziere  $\mathbf{L}_2(0)$  auf  $L_1$ :

$$\text{Es mu\ss\ gelten: } \left( (\mathbf{V}_1 + \alpha_1 \vec{\mathbf{I}}_1 - \mathbf{L}_2(0)) \cdot \vec{\mathbf{I}}_1 \right) = 0 \iff \alpha_1 = \frac{\left( (\mathbf{V}_2 - \mathbf{V}_1) \cdot \vec{\mathbf{I}}_1 \right)}{\left( \vec{\mathbf{I}}_1 \cdot \vec{\mathbf{I}}_1 \right)} = \frac{2}{4}$$

$$\text{damit: } \alpha_1^P = \frac{1}{2} \text{ und } \|\mathbf{L}_2(0) - \mathbf{L}_1(1/2)\| = \left\| \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \right\| = \sqrt{2}.$$

Also sind  $\mathbf{L}_1(1/2)$  und  $\mathbf{L}_2(0)$  die n\u00e4chstliegenden Punkte beider Strecken



## Quadratische Abstandsfunktion

**Erinnerung:** Suche Punkte minimalen Abstandes, also im Falle von Geraden:

$$(\alpha_1^P, \alpha_2^P) \text{ so dass: } \|\mathbf{L}_1(\alpha_1^P) - \mathbf{L}_2(\alpha_2^P)\| = \min_{\alpha_1 \in I_1, \alpha_2 \in I_2} \{\|\mathbf{L}_1(\alpha_1) - \mathbf{L}_2(\alpha_2)\|\}$$

**Beachte:** Da  $(a < b) \iff (a^2 < b^2)$ ,  $\forall a, b \in \mathbb{R}_0^+$  kann  $(\alpha_1^P, \alpha_2^P)$  alternativ als Minimum der *quadratischen Abstandsfunktion*  $Q(\alpha_1, \alpha_2) = \|\mathbf{L}_1(\alpha_1) - \mathbf{L}_2(\alpha_2)\|^2$  gesehen werden:

$$\begin{aligned} \|\mathbf{L}_1(\alpha_1^P) - \mathbf{L}_2(\alpha_2^P)\|^2 &= ((\mathbf{L}_1(\alpha_1^P) - \mathbf{L}_2(\alpha_2^P)) \cdot (\mathbf{L}_1(\alpha_1^P) - \mathbf{L}_2(\alpha_2^P))) \\ &= \min_{\alpha_1 \in I_1, \alpha_2 \in I_2} \{\|\mathbf{L}_1(\alpha_1) - \mathbf{L}_2(\alpha_2)\|^2\} = \min_{\alpha_1 \in I_1, \alpha_2 \in I_2} \{Q(\alpha_1, \alpha_2)\} \end{aligned}$$

**Eigenschaften** von  $Q$ :

- Graph von  $Q$  über der  $(\alpha_1, \alpha_2)$ -Parameterenebene ist ein *Paraboloid*
- $Q(\alpha_1, \alpha_2)$  hat eindeutiges Minimum  $\mathbf{M} \in \mathbb{R}^2$  falls  $\vec{\mathbf{I}}_1 \nparallel \vec{\mathbf{I}}_2$
- Isolinien mit Schwellwert  $s$   $\{(\alpha_1, \alpha_2) : Q(\alpha_1, \alpha_2) = s\}$  bilden Ellipsen in  $(\alpha_1, \alpha_2)$ -Ebene

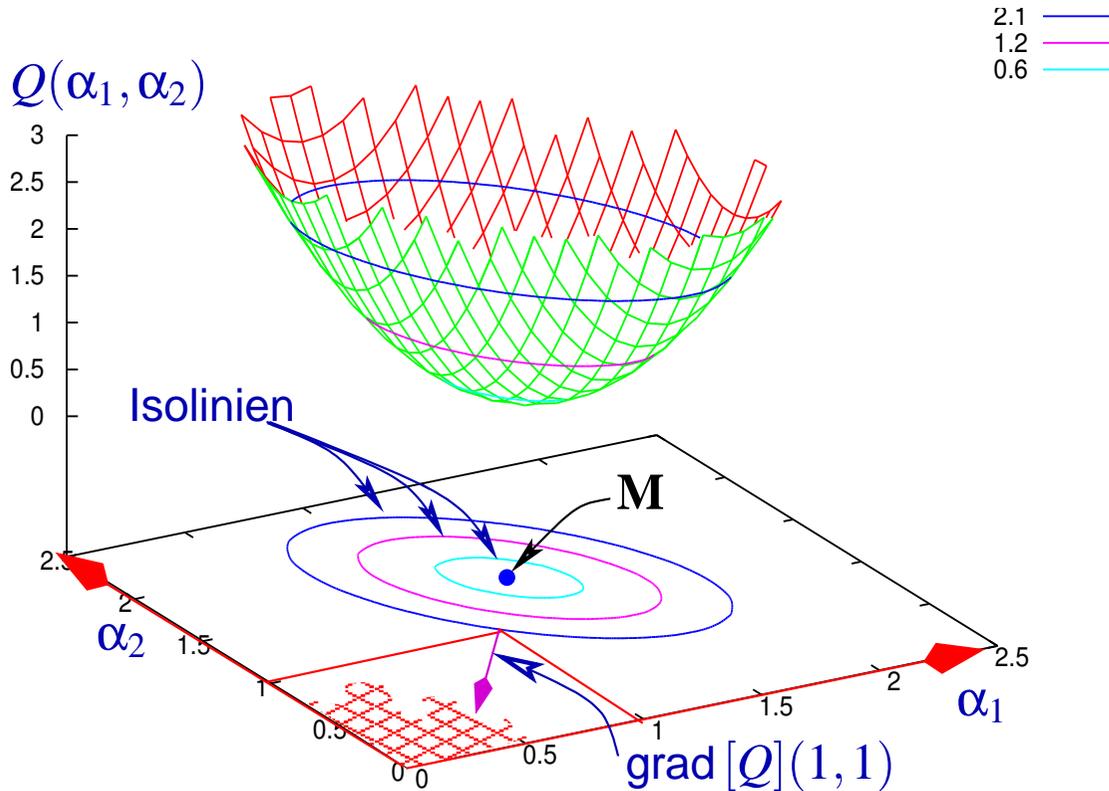
# 10.3.1 Abstand zweier linearer Komponenten



## Quadratische Abstandsfunktion (Forts.)

**Ziel:** Finde  $(\alpha_1^P, \alpha_2^P) \in I_1 \times I_2$  mit minimalem  $Q$ ;  $(\alpha_1^P, \alpha_2^P)$  liegt auf *kleinstmöglicher* Ellipse

**Gradient** mittels partieller Ableitungen:  $\text{grad}[Q](\alpha_1, \alpha_2) = \left( \frac{\partial Q}{\partial \alpha_1}(\alpha_1, \alpha_2), \frac{\partial Q}{\partial \alpha_2}(\alpha_1, \alpha_2) \right)$



### Nutzung des Gradienten:

1.  $\mathbf{M} = (\alpha_1^*, \alpha_2^*) \iff \text{grad}[Q](\mathbf{M}) = \vec{\mathbf{0}}$
2.  $\text{grad}[Q](\alpha_1, \alpha_2)$  weist in Richtung des größten Anstiegs von  $Q$  im Parameter  $(\alpha_1, \alpha_2)$  (siehe auch *Implizite Flächen*)

# 10.3.1 Abstand zweier linearer Komponenten



## Abstand zweier Strecken mittels Quadratischer Abstandsfunktion

**Ziel:** Effiziente Behandlung der Fälle 2+3

**Suche:** Kontaktpunkt von Region 0 mit kleinster Ellipse um **M**

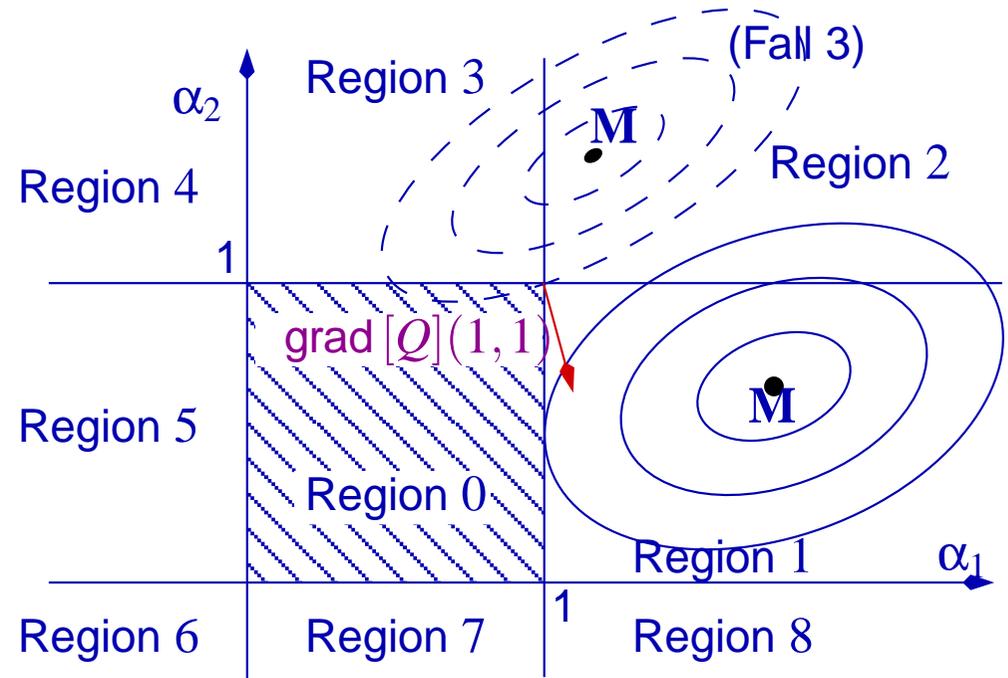
**Fall 2:** **M** liegt in Region 1, 3, 5 oder 7.  
Bsp. Region 1: Optimum liegt auf  $\alpha_1 = 1$

**Fall 3:** **M** liegt in Region 2, 4, 6 oder 8  
Bsp. Region 2: Optimum auf

$$\{(\alpha_1, \alpha_2) : \alpha_1 = 1 \vee \alpha_2 = 1\}$$

grad  $[Q]$  bei  $(1, 1)$  liefert Richtung:

$$\text{Optimum liegt } \left\{ \begin{array}{l} \text{bei } \alpha_1 = \alpha_2 = 1 \\ \text{auf } \alpha_1 = 1 \\ \text{auf } \alpha_2 = 1 \end{array} \right\} \iff \left\{ \begin{array}{l} \text{sgn}(\text{grad } [Q](1, 1)) = (-, -) \\ \text{sgn}(\text{grad } [Q](1, 1)) = (-, +) \\ \text{sgn}(\text{grad } [Q](1, 1)) = (+, -) \end{array} \right\}$$



Aufteilung der  $(\alpha_1, \alpha_2)$ -Ebene



## 10.3.1 Abstand zweier linearer Komponenten



### Beispiel: (siehe Daten von oben)

Die Fallunterscheidung von oben sieht jetzt folgendermaßen aus:

1. Gegeben:  $\mathbf{L}_1(\alpha_1) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \alpha_1 \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix}$ ,  $\mathbf{L}_2(\alpha_2) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix}$ ,  $\alpha_i \in [0, 1]$

2.  $\mathbf{M} = (\alpha_1^*, \alpha_2^*) = (-1/2, -1) \Rightarrow \mathbf{M}$  liegt in Region 6

3. Auswertung des Gradienten beim Eckpunkt des Parametergebietes  $(0, 0)$ :

$$\begin{aligned} \text{grad}[Q](0, 0) &= 2 \left( \left( (\mathbf{L}_1(0) - \mathbf{L}_2(0)) \cdot \vec{\mathbf{I}}_1 \right), - \left( (\mathbf{L}_1(0) - \mathbf{L}_2(0)) \cdot \vec{\mathbf{I}}_2 \right) \right) \\ &= 2 \left( \left( \left( \begin{pmatrix} -1 \\ -1 \\ -1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 0 \\ 0 \end{pmatrix} \right), \left( \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 0 \\ 1 \end{pmatrix} \right) \right) = (-4, 6) \end{aligned}$$

damit liegt Optimum auf  $\alpha_2 = 0$

4. Ermittle den Abstand von  $\mathbf{L}_2(0)$  zu  $L_1$  (siehe oben)



### Abstand Punkt-Rechteck

**Vorteil der quadratischen Abstandsfunktion:** Allgemeingültigkeit für ähnliche Probleme

**Gegeben:** Punkt  $\mathbf{P}$  und Rechteck  $R$ :

$$\mathbf{R}(\alpha_1, \alpha_2) = \mathbf{V} + \alpha_1 \vec{\mathbf{e}}_1 + \alpha_2 \vec{\mathbf{e}}_2, \quad \alpha_1, \alpha_2 \in [0, 1]$$

**Gesucht:**  $d_R(\mathbf{P}) = \min_{\mathbf{Q} \in R} \|\mathbf{P} - \mathbf{Q}\|$

**Quadratische Abstandsfunktion:**

$$Q(\alpha_1, \alpha_2) = \|\mathbf{R}(\alpha_1, \alpha_2) - \mathbf{P}\|^2 = ((\mathbf{R}(\alpha_1, \alpha_2) - \mathbf{P}) \cdot (\mathbf{R}(\alpha_1, \alpha_2) - \mathbf{P}))$$

**Globales Minimum:**  $\mathbf{M} = (\alpha_1^*, \alpha_2^*)$  ohne Rücksicht auf Einschränkung für  $\alpha_i$

**Regionen:** Unterteile  $(\alpha_1, \alpha_2)$ -Ebene in Regionen; es entsteht dieselbe Unterteilung wie Strecke-Strecke

**Fälle:** analog wie Strecke-Strecke



## Weitere Objekte

**Punkt/Dreieck:** Definition des Dreiecks:

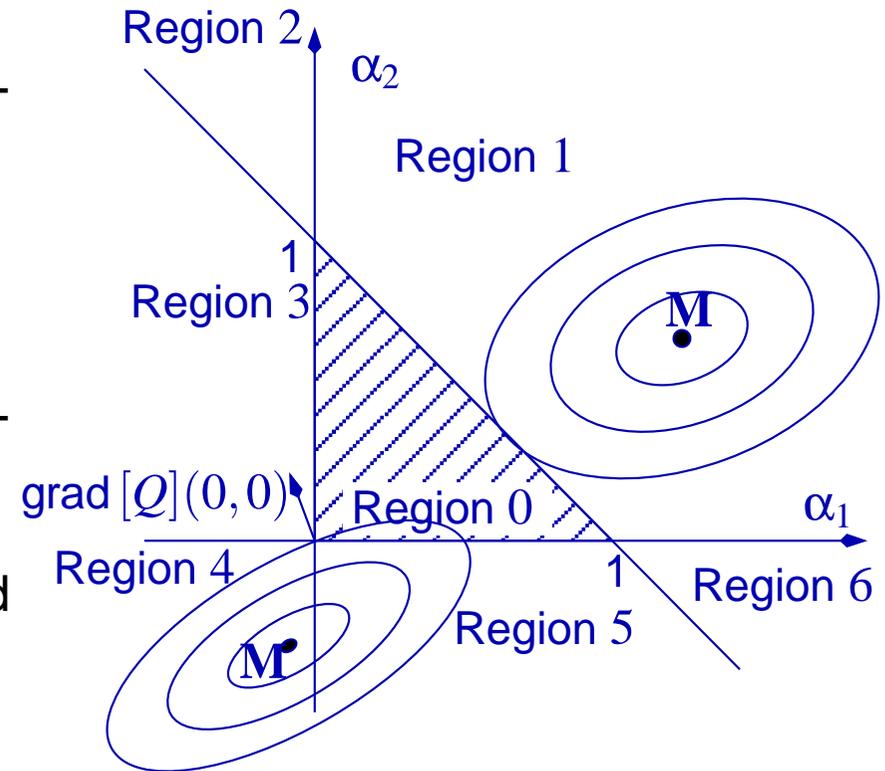
$$\mathbf{D}(\alpha_1, \alpha_2) = \mathbf{V} + \alpha_1 \vec{\mathbf{e}}_1 + \alpha_2 \vec{\mathbf{e}}_2, \quad \alpha_1, \alpha_2, \alpha_1 + \alpha_2 \in [0, 1]$$

**Abstand Lineare Komponente/Dreieck:** Drei Parameter für Abstandsfunktion:

$$Q(\alpha_1, \alpha_2, \alpha_3) = \|\mathbf{D}(\alpha_1, \alpha_2) - \mathbf{L}(\alpha_3)\|^2$$

$L$  ist Gerade/Strahl/Strecke  $\implies$  7/14/21 Regionen im  $\mathbb{R}^3$

**Dreieck/Dreieck:** Insgesamt vier Parameter und 49 Regionen.



## 10.3.3 Kollisionspunkt und -normale



### Ansatz:

- Ziel:**
- Kollisionspunkt und -normale für einfache implizite Objekte effizient berechnen
  - Ermittlung unabhängig von der konkreten Ausprägung der Basisgeometrie

**Gegeben:** Implizite Geometrie  $M_1, M_2$  mit Basisgeometrien  $B_1, B_2$  und Schwellwerten  $s_1, s_2$ .

**Abstand:** Ermittlung der Punkte kürzesten Abstands auf den Basisgeometrien:

$$\left\| \mathbf{B}_1(\alpha_{1,1}^P, \dots, \alpha_{1,n}^P) - \mathbf{B}_2(a_{2,1}^P, \dots, \alpha_{2,m}^P) \right\| = \min \{ \left\| \mathbf{B}_1(\alpha_{1,1}, \dots, \alpha_{1,n}) - \mathbf{B}_2(a_{2,1}, \dots, \alpha_{2,m}) \right\| \}$$

$$\text{Abstand: } d(M_1, M_2) = \left\| \mathbf{B}_1(\alpha_{1,1}^P, \dots, \alpha_{1,n}^P) - \mathbf{B}_2(a_{2,1}^P, \dots, \alpha_{2,m}^P) \right\| - (s_1 + s_2)$$

**Kollisionspunkte:** Falls  $d(M_1, M_2) \approx 0$ :

$$\text{Kollisionsnormale: } \vec{\mathbf{n}}_{koll} = \mathbf{B}_2(\alpha_{2,1}^P, \dots, \alpha_{2,m}^P) - \mathbf{B}_1(\alpha_{1,1}^P, \dots, \alpha_{1,n}^P)$$

$$\hat{\mathbf{n}}_{koll} = \vec{\mathbf{n}}_{koll} / \|\vec{\mathbf{n}}_{koll}\|$$

$$\text{Kollisionspunkte: } \mathbf{K}_1 = \mathbf{B}_1(\alpha_{1,1}^P, \dots, \alpha_{1,n}^P) + s_1 \hat{\mathbf{n}}_{koll}$$

$$\mathbf{K}_2 = \mathbf{B}_2(\alpha_{2,1}^P, \dots, \alpha_{2,m}^P) - s_2 \hat{\mathbf{n}}_{koll}, \quad \mathbf{K}_1 \approx \mathbf{K}_2$$



# 10.4 Kollision für Dreiecksnetze



**Idee:** Verwende Abstandsberechnung für Basisgeometrien zur Kollisionserkennung, z.B. für Dreiecken:

$$\text{Dreiecke: } \mathbf{D}_i(\alpha_1, \alpha_2) = \mathbf{V}_i + \alpha_{i,1}\vec{\mathbf{e}}_{i,1} + \alpha_{i,2}\vec{\mathbf{e}}_{i,2}, \quad \alpha_{i,1}, \alpha_{i,2}, \alpha_{i,1} + \alpha_{i,2} \in [0, 1], \quad i = 1, 2$$

$$\text{Schwellwert: } s_i = 0, \quad i = 1, 2$$

$$\text{Kollision} \iff d(D_1, D_2) \approx 0$$

**Problem:** relativ aufwendige Berechnung für Dreiecke

**Alternativer Ansatz:** Betrachte Kollision paarweise zwischen Eckpunkt, Kante und Dreieck

	<b>Punkt</b>	<b>Kante</b>	<b>Dreieck</b>
<b>Punkt</b>	degeneriert	→ Kante-Kante	!!
<b>Kante</b>	–	!!	→ Punkt-Dreieck → Kante-Kante
<b>Dreieck</b>	–	–	→ Punkt-Dreieck → Kante-Kante

**Relavante Fälle:** Kante-Kante und Punkt-Dreieck



# 10.4 Kollision für Dreiecksnetze



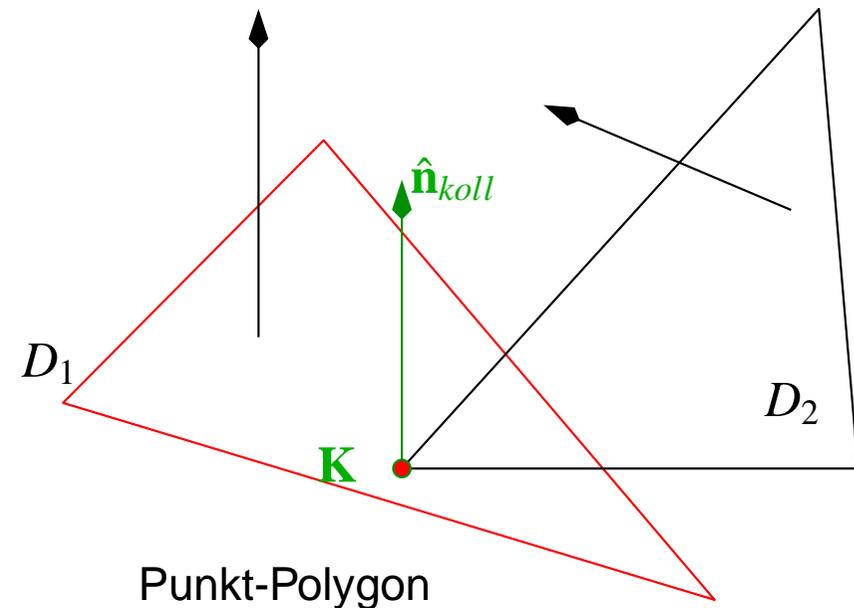
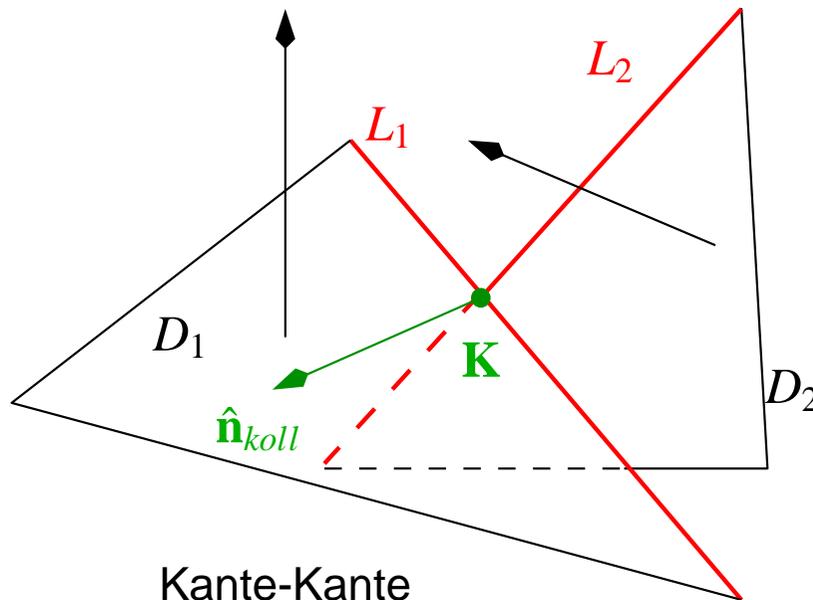
## Alternativer Ansatz für Dreiecke/Polygone (Forts.)

**Kollisionspunkt:** Gegeben durch Schnittpunkt der Kanten bzw. durch den Punkt bei Punkt-Dreieck

**Kollisionsnormale** wird aus geometrischen Primitiven ermittelt:

**Kante-Kante:** Verwende das Kreuzprodukt der Kantenrichtungen

**Punkt-Fläche:** Normale des Polygons



# 10.4 Kollision für Dreiecksnetze



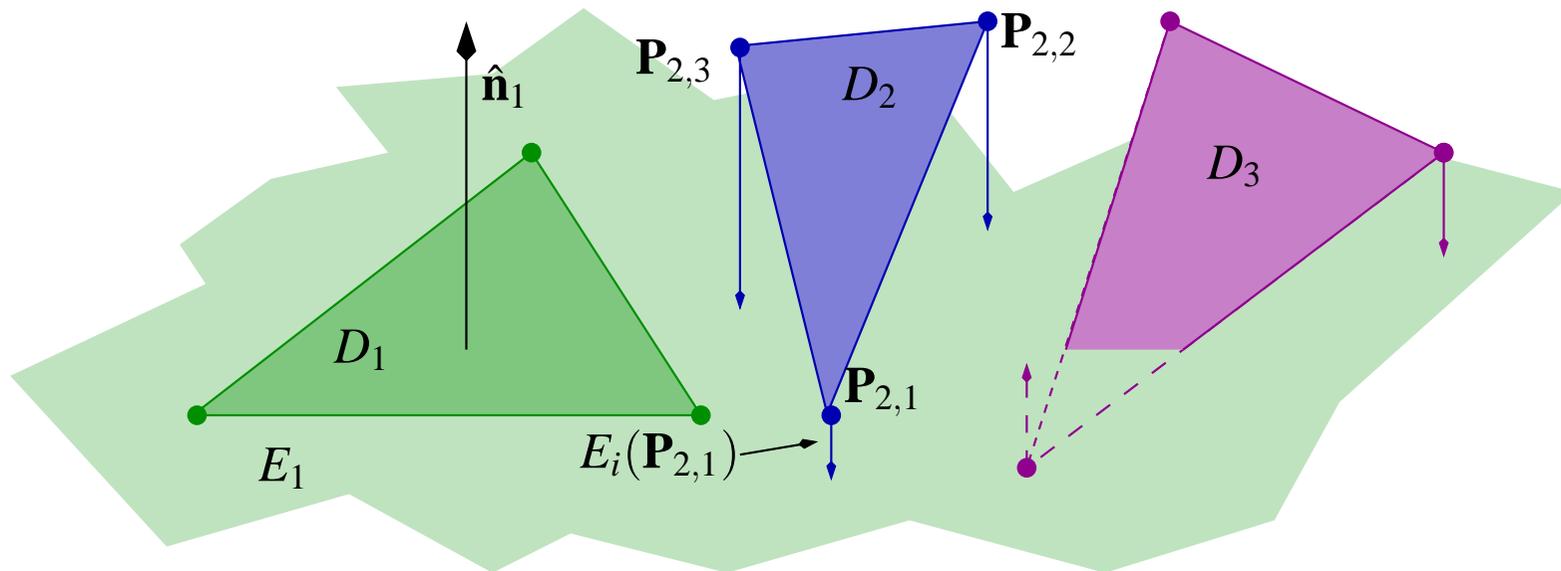
## Grobe Abschätzung für Dreieck-Dreieck-Kollision

**Beobachtung:** Keine Kollision, wenn Eckpunkte eines Dreiecks *auf einer Seite* der anderen Dreiecksebene sind

**Hessesche Normalform:**  $E_i(\mathbf{P}) = (\mathbf{P} \cdot \hat{\mathbf{n}}_i) - d_i$

**Eckpunkte des  $j$ -ten Dreiecks  $\mathbf{P}_{j,k}$ ,  $k = 1, \dots, 3$  auf einer Seite von  $E_i$ , wenn**

$$\text{sgn}(E_i(\mathbf{P}_{j,1})) = \text{sgn}(E_i(\mathbf{P}_{j,2})) = \text{sgn}(E_i(\mathbf{P}_{j,3}))$$





## Einsatzbereiche von Raumunterteilungen

**Visibilitätstest:** Welches Objekt liegt einem Beobachterpunkt am nächsten?

**Level of Detail:** Wie detailliert muß ein Objekt für einen Beobachterpunkt dargestellt werden (Objektentfernung)?

**Kollisionstest:** Welche (statischen) Objekte liegen nahe einem bewegten Objekt?

**Allgemeines Ziel:** Effizienter Zugriff auf Objekte relativ zu einer Referenzposition

## Generelle Eigenschaften

**Raumunterteilung:** *Disjunkte* Zerlegung eines (*Beobachtungs-*)Raumes  $R$  in Teilräume  $U_i$ :

$$R = \dot{\bigcup}_{i=1}^n U_i \text{ in konvexe Teilräume } U_i, i = 1, \dots, n$$

Disjunkte heißt hier:  $\forall \mathbf{P} \in R \exists_1 i : \mathbf{P} \in U_i$

**Hierarchie** von Raumunterteilung durch rekursive Unterteilung ( $\Rightarrow$  Baumstruktur)

**Zuordnung** von (meist statischen) Objekten zu den Teilräumen



**Ausgangslage:** Bounding-Box der gesamten Szene  $U^0$

**Rekursive Unterteilung** am Box-Mittelpunkt in *gleichgroße Teilbereiche*:

Quadtree:

$$\text{initial: } U^0 = \dot{\bigcup}_{k=1}^4 U_k^1$$

$$\text{rekursiv: } U_I^i = \dot{\bigcup}_{k=1}^4 U_{(I,k)}^{i+1}$$

Octree

$$\text{initial: } U^0 = \dot{\bigcup}_{k=1}^8 U_k^1$$

$$\text{rekursiv: } U_I^i = \dot{\bigcup}_{k=1}^8 U_{(I,k)}^{i+1}$$

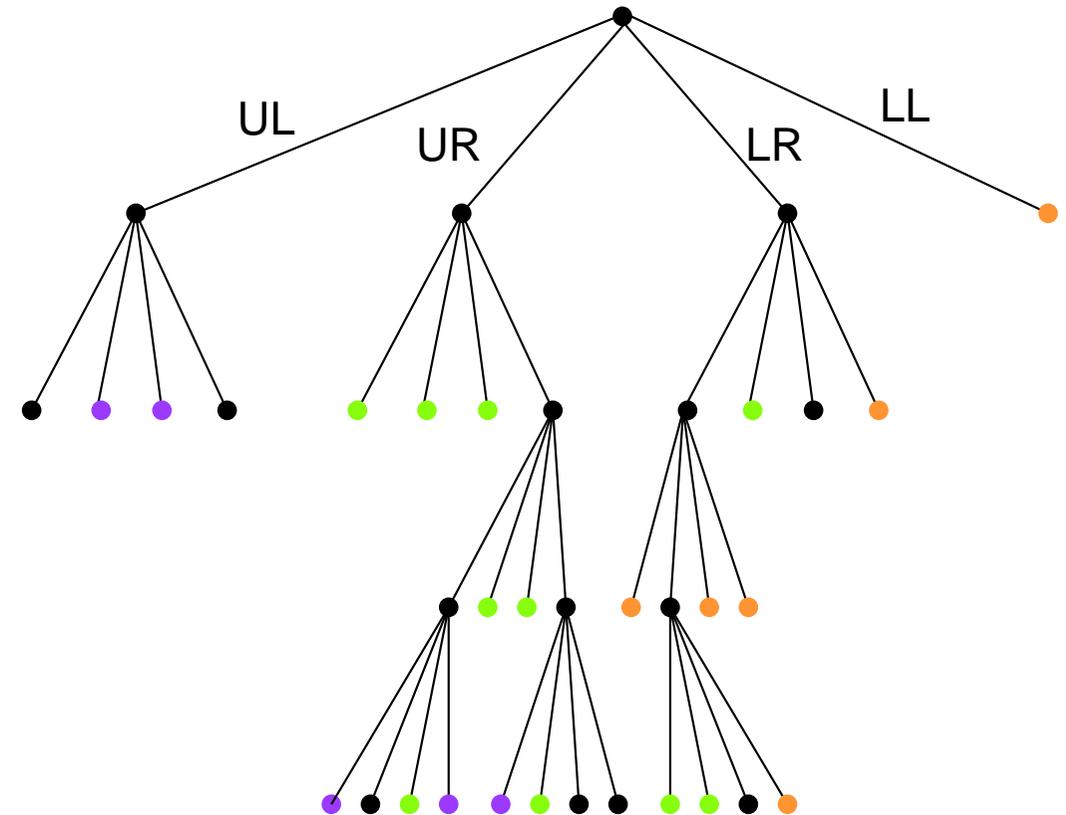
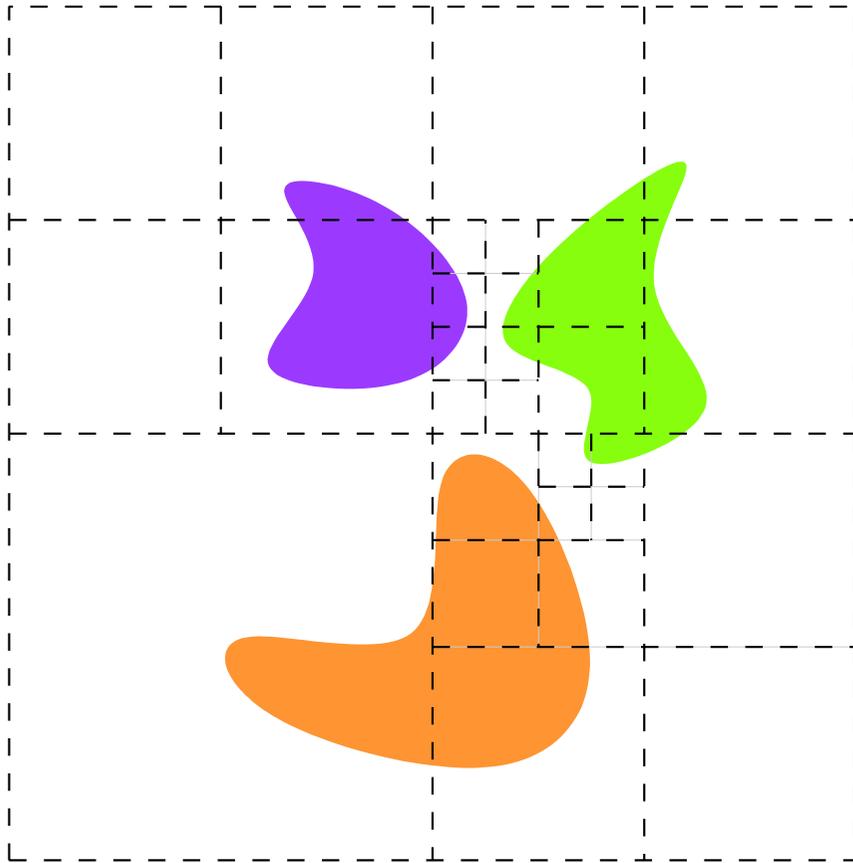
mit Multi-Index  $I = (k_1, \dots, k_i)$ ,  $k_j \in \{1, 2, 3, 4\}$  bzw.  $k_j \in \{1, \dots, 8\}$ .

**Abbruchkriterium:** Zwei gängige Alternativen

1. Teilbereich enthält (auch teilweise) max. eine Obergrenze  $n_{max}$  von Objekten
2. max. Unterteilungstiefe erreicht

**Blattknoten** können auch keine oder weniger als  $n_{max}$  Objekte (teilweise) enthalten

## Beispiel: Quadtree (2D)



### Benennung der Teilräume:

UL=„upper left“, UR=„upper right“, LR=„lower right“, LL=„lower left“



## Quadtree Search

**Ziel:** Suche Zelle, in der Punkt  $\mathbf{P} = (p_x, p_y)$  liegt

### **Pseudo-Code:**

```
NodeID searchQuadtree(Node root, Point P)
{
    if ( root.hasChildren == false ) return root.id;

    Point midPoint = 0.5 * (root.min + root.max); // subdiv. point
    if ( P.x < midPoint.x ) {
        if ( P.y < midPoint.y ) return searchQuadtree(root.ll, P);
        else return searchQuadtree(root.ul, P);
    }
    else {
        if ( P.y < midPoint.y ) return searchQuadtree(root.lr, P);
        else return searchQuadtree(root.ur, P);
    }
}
```



# 10.5.2 Binary Space Partition (BSP)



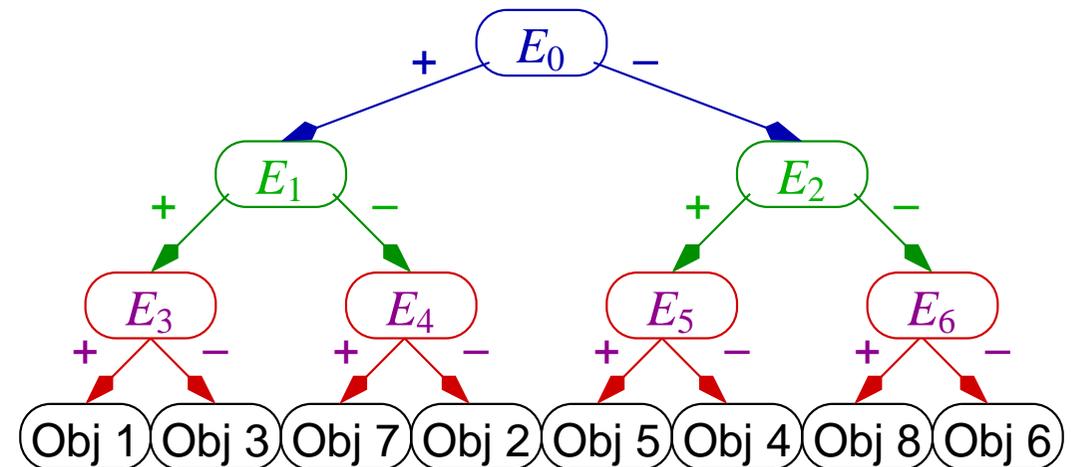
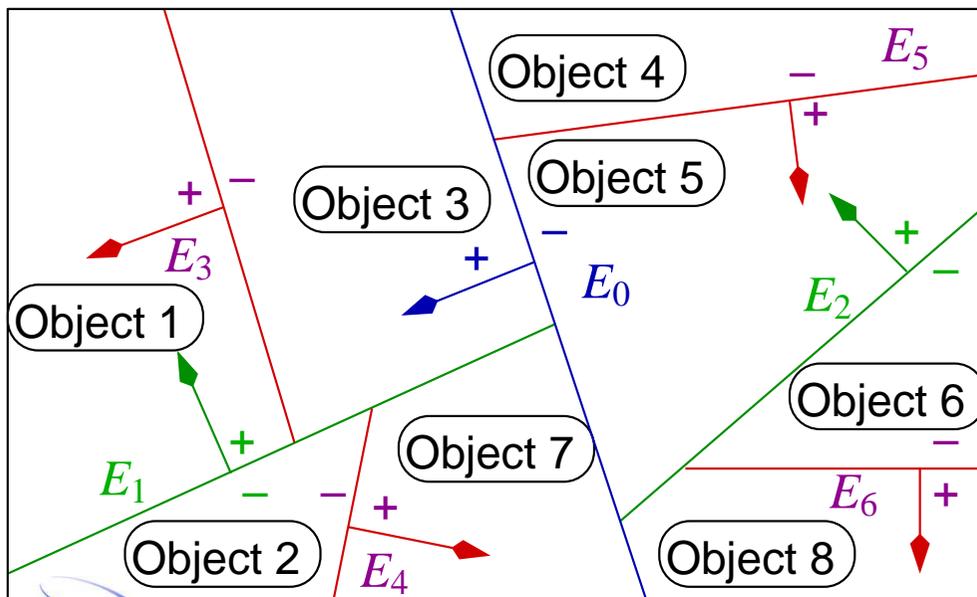
**Ansatz:** Rekursive, binäre Unterteilung

$n - 1$ -dimensionale *Hyper-Ebenen* unterteilen aktuellen Bereich in zwei Teilbereiche:

$$\text{Ebenengleichung: } E(\mathbf{P}) = (\mathbf{P} \cdot \hat{\mathbf{n}}) - d = 0$$

$$\text{Front-Bereich: } \{\mathbf{P} : E(\mathbf{P}) > 0\} \quad \text{Back-Bereich: } \{\mathbf{P} : E(\mathbf{P}) < 0\}$$

- Ziele:**
- Objekte *vollständig* im Front- oder Back-Bereich ( $\Rightarrow$  Objekte notfalls teilen!)
  - Ziel: Jeweils dieselbe Anzahl Objekte auf jeder Seite ( $\Rightarrow$  balancierter Baum)



## 10.5.2 Binary Space Partition (BSP)



### BSP-Tree Search

**Ziel:** Suche Blattknoten, in der Punkt  $\mathbf{P} = (p_x, p_y)$  liegt

### **Pseudo-Code:**

```
NodeID searchBSPTree(Node root, Point P)
{
    if ( root.hasChildren == false ) return root.id;

    if ( evalPlane(root.plane, P) > 0 ) { // positive region
        return searchBSPTree(root.posSubtree, P);
    }
    else { // negative region
        return searchBSPTree(root.negSubtree, P);
    }
}
```





### Vergleich Quadtree/Octree und BSP-Tree

#### Grundsätzlich:

- beide Baumtypen für dynamische Szenen schwer anpaßbar (Anpassung/Update teuer)
- *kd-Tree* ist ein BSP-Tree, bei dem nur achsenparallele Ebenen verwendet werden

**Octree:** + effiziente Entscheidung auf Knotenebene (zwei Vergleiche)

- Baum i.a. nicht ausbalanciert
- Objekte u.U in mehreren Blattknoten (Problem bei einigen Anwendungen)

**BSP-Tree:** + Baum i.a. gut balanciert  $\Rightarrow$  konstanter Aufwand für Zugriff

- + effizienterer Baum, da Objekte genau in einem Blattknoten
- aufwendige Konstruktion durch Objekt-Splitting

# 10.5.2 Binary Space Partition (BSP)



## Hidden-Surface Removal mit BSP-Tree

**Ziel:** Rendering mehrerer Objekten in „Back-to-Front“- Reihenfolge (korrekte Verdeckung!)

```
Algorithmus: void renderBSPTree(Node root, Camera cam)
{
  if ( root.hasChildren == false ) { /* render objects */ }
  if ( evalPlane(root.plane, cam) > 0 ) { // cam in pos. region
    renderBSPTree(root.negSubtree, cam);
    renderBSPTree(root.posSubtree, cam);
  } else { // cam in neg. region
    renderBSPTree(root.posSubtree, cam);
    renderBSPTree(root.negSubtree, cam);
  }
}
```

