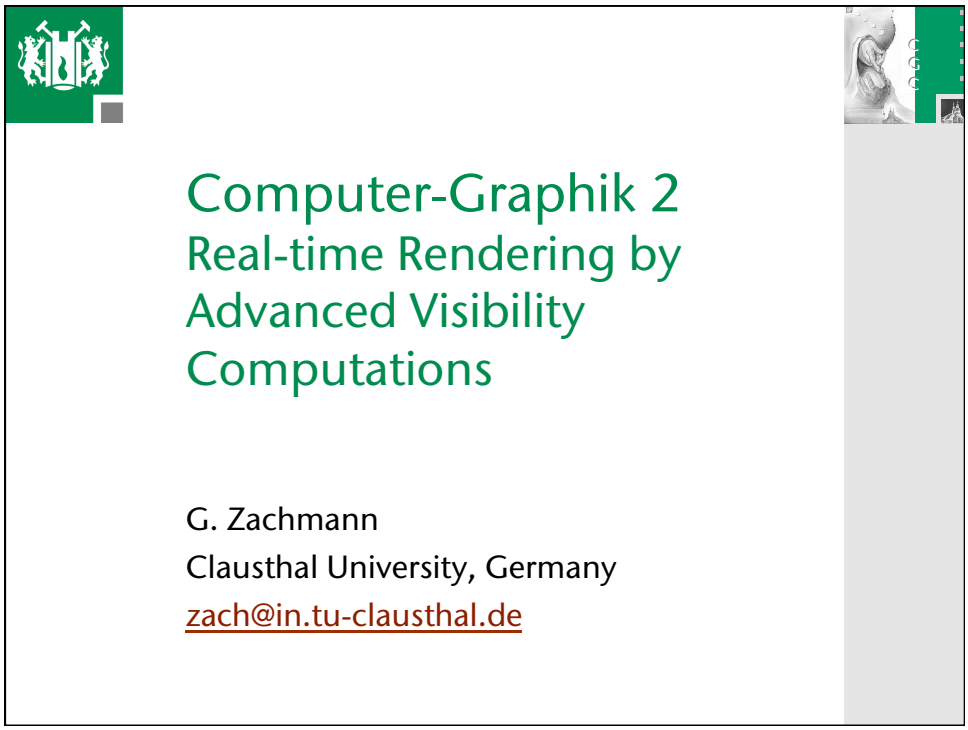
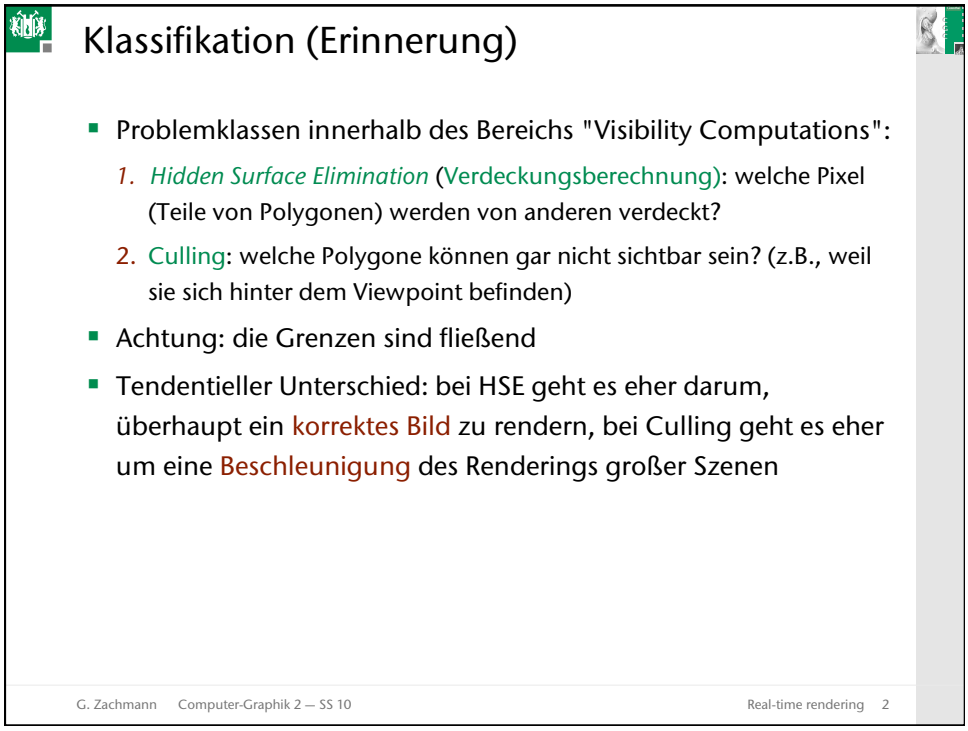


Computer-Graphik 2
Real-time Rendering by Advanced Visibility Computations

G. Zachmann
Clausthal University, Germany
zach@in.tu-clausthal.de



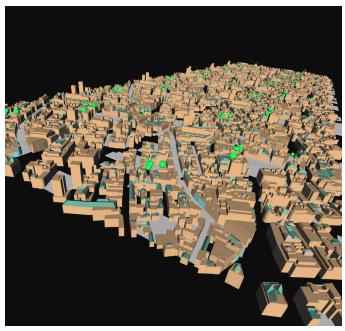
Klassifikation (Erinnerung)

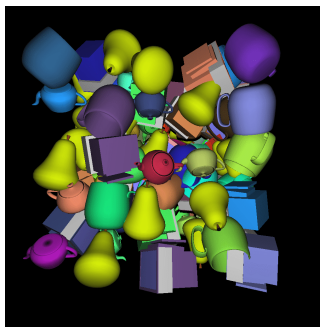
- Problemklassen innerhalb des Bereichs "Visibility Computations":
  1. *Hidden Surface Elimination* (Verdeckungsberechnung): welche Pixel (Teile von Polygonen) werden von anderen verdeckt?
  2. Culling: welche Polygone können gar nicht sichtbar sein? (z.B., weil sie sich hinter dem Viewpoint befinden)
- Achtung: die Grenzen sind fließend
- Tendentieller Unterschied: bei HSE geht es eher darum, überhaupt ein korrektes Bild zu rendern, bei Culling geht es eher um eine Beschleunigung des Renderings großer Szenen

Culling

- Sei *A* = Menge A aller Primitive;
  sei *S* = Menge der sichtbaren Primitive.
- Alle bisher betrachteten Algorithmen arbeiten auf der gesamten Menge A, d.h., sie haben einen Aufwand mindestens in $O(|A|)$.
- Unproblematisch, wenn $|S| \approx |A|$ ist.
  - Z.B., wenn Anzahl der Primitive im Vergleich zur Pixelanzahl klein ist.
  - Erinnerung: Depth Complexity
- *"to cull from"* = "sammeln [aus ...] / auslesen"
  "to cull flowers" = Blumen pflücken

- Aber: für komplexe Szenen ist die Anzahl der sichtbaren Primitive in der Regel wesentlich kleiner als die Anzahl der Primitive insgesamt (|S| << |A|) !

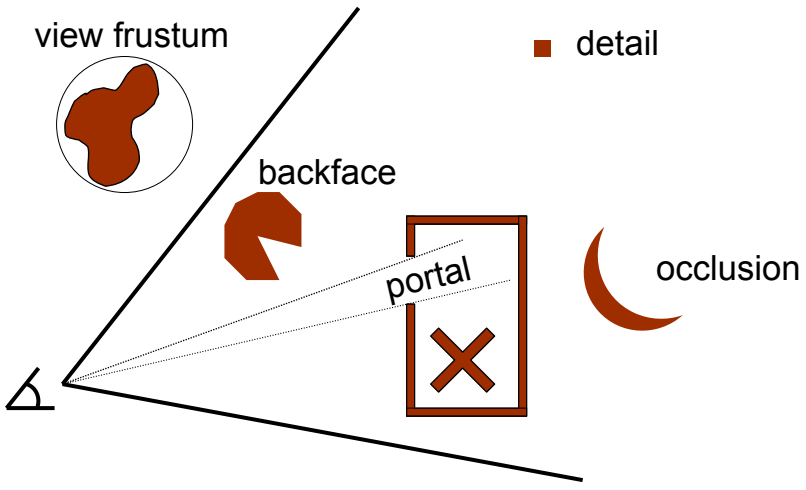




- Culling ist eine wichtige Optimierung (im Gegensatz zu Clipping)

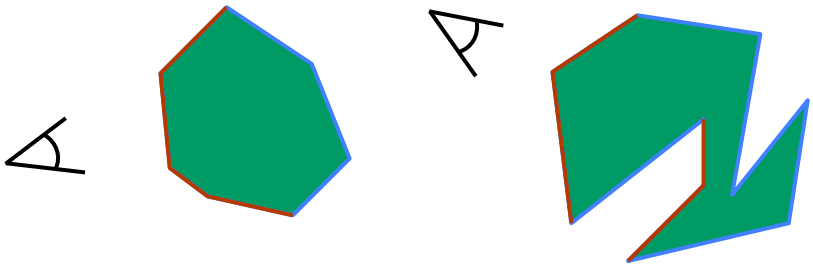
- Für $|S| << |A|$ genügen die bisherigen Algorithmen nicht
- Culling-Algorithmen versuchen, die Menge der nicht-sichtbaren Primitive $C = A \setminus S$ (oder Teilmenge davon), oder die Menge der sichtbaren Primitive S (oder Obermenge davon) zu bestimmen.
- Definition: Potentially Visible Set (PVS) = Obermenge $S' \supseteq S$
  - Ziel: möglichst kleines PVS S' mit möglichst geringem Aufwand
  - Triviales PVS (mit trivialstem Aufwand) ist natürlich A

Culling-Arten

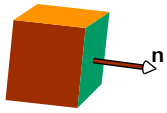


Back-Face Culling

- Definition: Solid = geschlossenes, opakes Objekt; also undurchsichtiges Objekt mit nicht-degeneriertem Volumen
- Beobachtungen:
  - Bei Solids sind die Rückseiten (back-faces) nie sichtbar
  - Bei konvexen Objekten gibt es genau 1 zusammenhängende Rückseite
  - Bei nicht-konvexen Solids eventuell mehrere

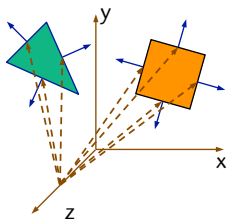


- Backface Culling = Nicht-Zeichnen der dem Viewpoint abgewandten Flächenstücke
  - Klappt nur bei Solids!



- Berechne Normale **n** des Polygons
- Berechne Vektoren **v** vom Viewpoint zu allen Punkten **p** des Polygons
  - Orthogonale Projektion: $\mathbf{v} = [0\ 0\ 1]^T$
  - Perspektivische Projektion: $\mathbf{v} = \mathbf{p} - \mathbf{eye}$
- Abgewandt (nicht zeichnen) wenn Winkel zwischen **n** und **v** < 90°

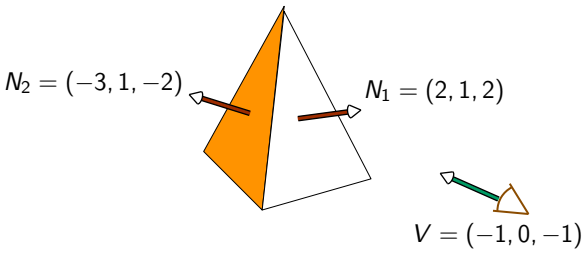
  $\leftrightarrow \quad \mathbf{n} \cdot \mathbf{v} > 0$



Beispiel

$$N_1 \cdot V = (2,1,2)\cdot(-1,0,-1)$$
$$= -4 < 0$$
$\Rightarrow N_1$ front facing

$$N_2 \cdot V = (-3,1,-2)\cdot(-1,0,-1)$$
$$= 5 > 0$$
$\Rightarrow N_2$ back facing

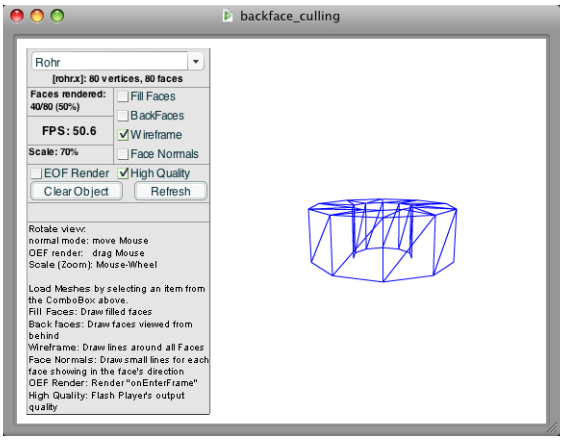


Backface Culling in OpenGL

- Muß man nur einschalten:

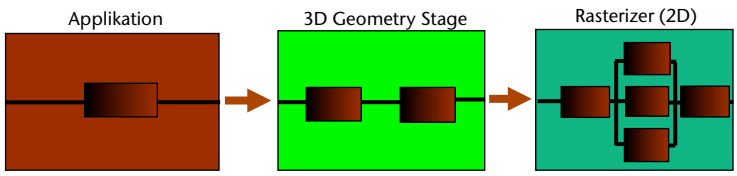
```
glCullFace( GL_BACK );
glEnable( GL_CULL_FACE );
```

Demo



Wann lohnt sich Backface Culling?

- Erinnerung: Graphik-Pipeline

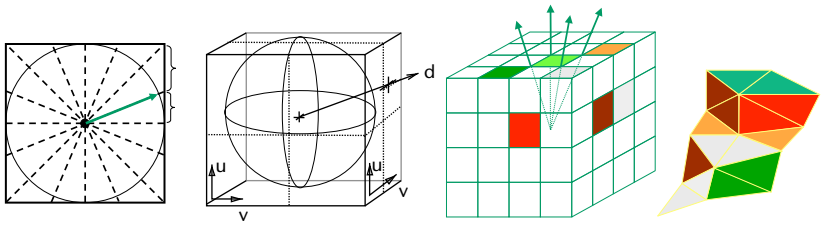


- Eine Pipeline hat immer den Durchsatz wie das langsamste Glied!
- Mögliche Bottlenecks in der Graphik-Pipeline:
  - Im Rasterizer → *"fill limited"*
  - In der Geometry-Stage → *"transform limited"*
  - Auf dem Bus zwischen App. und Graphik-Hardware → *"bus limited"*
  - Falls die Graphikkarte schneller ist als die Applikation Geometrie liefern kann → *"CPU limited"* (erkennt man an 100% CPU-Auslastung)

Normal Masks

[Zhang & Hoff, 1997]

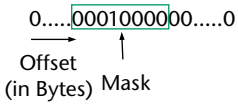
- Zentrale Idee: Skalarprodukt ersetzen durch Klassifizierung aller Normalen
- Bilde zunächst Klassen über der Menge aller Normalen
  - Umschließe Normalenkugel (Gauß'sche Kugel) mit Würfel ("Richtungswürfel" oder "direction cube")



  - Ergibt $6 \cdot N^2$ viele Klassen (N = Anzahl Gitterunterteilungen)
  - Klassifizierung einer Normalen ist sehr einfach

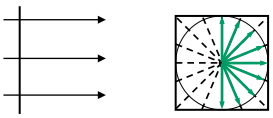
- Kodierung einer Normalen (Preprocessing):
  - Der gesamte Normalenwürfel $\hat{=}$ Bitstring der Länge $6 \cdot N^2$
  - Eine Normale $\hat{=}$ nur eine 1, sonst 0-en
  - Kodierung als Offset + Teil des Bitstrings, der die 1 enthält
  - Z.B.: unterteile Bitstring in Bytes, Offset = 1 Byte, ergibt 256x8 = 2048 Bits

```
typedef struct PolygonNormalMask
{
   Byte offset, bitMask;
};
```

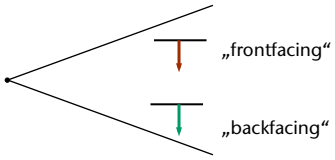


  - Speichere diese 2 Bytes zu jedem Polygon
  - Z.B.: wähle N = 16
  - Ergibt Unterteile die Normalenkugel in $6 \cdot 16 \cdot 16 = 1536$ Klassen

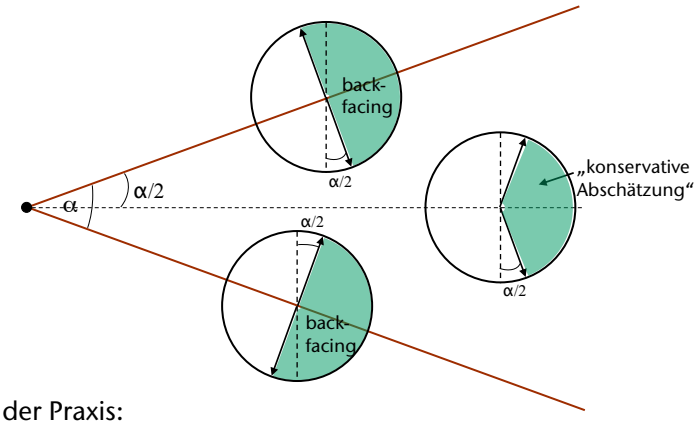
- Culling (Initialisierung):
  - Identifiziere alle diejenigen Normalenklassen, deren Normalen alle backfacing sind
  - Orthographfische Projektion:



  - Perspektivische Projektion: welche Normalen backfacing sind hängt von Normalenrichtung **und Position** des Polygons ab!



  - Deswegen: berechne "konservative" Menge von Klassen, die – unabhängig vom Ort des Polygons – backfacing sind:



- In der Praxis:
  - Teste jede Klasse in allen vier Ecken des View-Frustums
  - Test einer Klasse = Test der 4 Normalen, die in die Ecken des Quadrats zeigen

- Stelle diese Menge als Bitstring (z.B. 2048 Bits = 256 Bytes) in einem Byte-Array dar:

```
Byte BackMask[256];
```

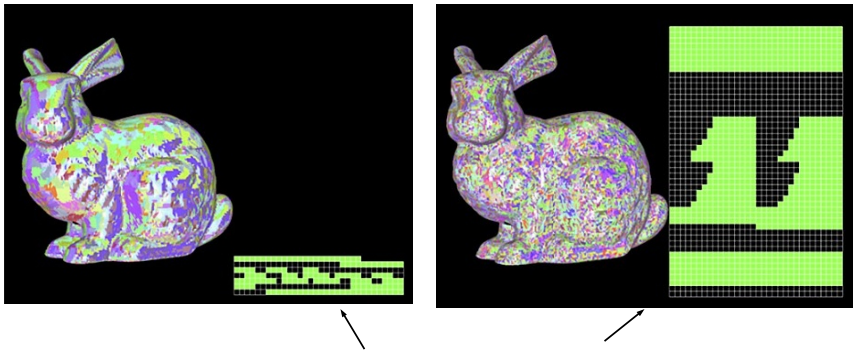
- Culling (zur Laufzeit): teste für jedes Polygon

```
BackMask[byteOffset] & bitMask
```

- Beschleunigung: rendere die Szene "sektorenweise"
  - damit ist der Winkel $\alpha/2$ in jedem Sektor kleiner;
  - für jeden Sektor eine eigene **BackMask []**.
- Resultat: die Autoren berichten Faktor 1.5 Speedup gegenüber OpenGL-Backface-Culling

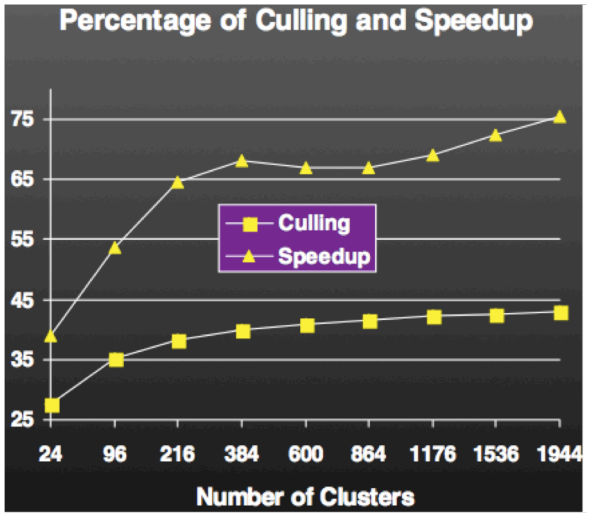
Beispiel

216 Klassen ("cluster") 1536 Klassen ("cluster")



BackMask für den aktuellen Viewpoint
(grün = backfacing)

Speedup



Clustered Backface Culling [1998]

- Erinnerung: einfache Rechenregeln zu min/max:

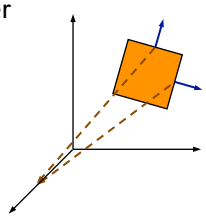
$$\max(x+y) \leq \max(x) + \max(y)$$

$$\max(x-y) \leq \max(x) - \min(y)$$

$$\max(kx, ky) = \begin{cases} k\max(x,y) & , \; k \geq 0 \\ k\min(x,y) & , \; k < 0 \end{cases}$$

- Im folgenden seien $\mathbf{n}^i$ und $\mathbf{p}^i$ die Normale bzw. ein Vertex eines Polygons aus einem Cluster (einer Menge) von Polygonen. Sei $\mathbf{e}$ der Viewpoint.
- Achtung: wir verwenden im folgenden die "umgekehrte" Definition für Backfacing!

$$\mathbf{n} \cdot (\mathbf{e} - \mathbf{p}) \leq 0$$



- Annahme: Cluster (= Menge) von Polygonen ist gegeben
- Alle Polygone sind backfacing gdw.

$$\forall i : \mathbf{n}^i \left(\mathbf{e} - \mathbf{p}^i\right) \leq 0 \quad \Leftrightarrow$$
$$\max\left\{\mathbf{n}^i \left(\mathbf{e} - \mathbf{p}^i\right)\right\} \leq 0 \qquad (1)$$

- Obere Schranke für (1) ist

$$\max\left\{\mathbf{n}^i \left(\mathbf{e} - \mathbf{p}^i\right)\right\} \leq \max\left\{\mathbf{e}\mathbf{n}^i\right\} - \min\left\{\mathbf{n}^i\mathbf{p}^i\right\} \qquad (2)$$

- Setze d := min{$\boldsymbol{n}^i \cdot \boldsymbol{p}^i$} (precomputation)
- Schreibe (2) als

$$\max\left\{\mathbf{n}^i \left(\mathbf{e} - \mathbf{p}^i\right)\right\} \leq \max\left\{e_x n_x^i + e_y n_y^i + e_z n_z^i\right\} - d$$
$$\leq \max\left\{e_x n_x^i\right\} + \max\left\{e_y n_y^i\right\} + \max\left\{e_z n_z^i\right\} - d \qquad (3)$$

- Annahme: **e** liegt im pos. Oktanten, d.h., e_x, e_y, $e_z \geq 0$; dann können wir eine obere Schranke von (3) angeben:

$$\max\left\{\mathbf{n}^i \left(\mathbf{e} - \mathbf{p}^i\right)\right\}$$

$$\leq e_x \cdot \max\{n_x^i\} + e_y \cdot \max\{n_y^i\} + e_z \cdot \max\{n_z^i\} - d$$

$$\leq \mathbf{m} \cdot \mathbf{e} - d\,, \quad \text{mit} \quad \mathbf{m} = \begin{pmatrix} \max\{n_x^i\} \\ \max\{n_y^i\} \\ \max\{n_z^i\} \end{pmatrix}$$

- Analog gilt für e_x, e_y, $e_z \leq 0$:

$$\max\left\{\mathbf{n}^i \left(\mathbf{e} - \mathbf{p}^i\right)\right\} \leq \mathbf{n} \cdot \mathbf{e} - d\,, \quad \text{mit} \quad \mathbf{n} = \begin{pmatrix} \min\{n_x^i\} \\ \min\{n_y^i\} \\ \min\{n_z^i\} \end{pmatrix}$$

- Für alle anderen Oktanten hat man die entsprechenden Kombinationen aus min & max
- Schreibweise: definiere eine Art „If“-Operator auf Vektoren

$$\text{if}\,(\mathbf{e};\mathbf{u},\mathbf{v}) := \begin{cases} u_\alpha & , e_\alpha \le 0 \\ v_\alpha & , e_\alpha > 0 \end{cases} \qquad \text{mit } \alpha \in \{x, y, z\}$$

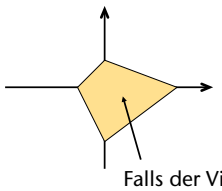
- Damit kann man den (konservativen) Test dann so schreiben:

$$\text{if}\,(\mathbf{e};\mathbf{n},\mathbf{m})\cdot\mathbf{e} - d \le 0 \quad \Rightarrow \quad \text{cluster is backfacing} \tag{4}$$

- Precomputation: pro Cluster **n**, **m** und *d* bestimmen
- Speicherbedarf pro Cluster: 28 Bytes (2 Vektoren + 1 Skalar)

Geometrische Interpretation

- Ungleichung (4) definiert 8 Ebenen (pro Oktant eine)
- Je 4 Ebenen (in angrenzenden Oktanten) schneiden sich in einem Punkt auf der dazwischenliegenden Koord.achse
  - Beispiel: betrachte die 4 Ebenen in den Oktanten mit $\boldsymbol{e}_x \ge 0$
  - Alle 4 Ebenen haben Normale der Form $\boldsymbol{n} = (m_x, \cdot, \cdot) \rightarrow$
  - sie schneiden alle die X-Achse im Punkt $\left(\frac{d}{m_x}, 0, 0\right)$.
- Diese 8 Ebenen bilden ein geschossenes Volumen, das sog. "Culling-Volumen"



Falls der Viewpoint sich hier drin befindet, ist der Cluster komplett backfacing

Weitere Optimierung: lokale Test-Koordinaten

- Problem: wenn die Polygone weit vom Ursprung entfernt liegen, und der Ursprung auf der positiven Seite der Normalen liegt, dann wird d sehr weit negativ → der Test fällt nie positiv aus
- Abhilfe: führe den Test in einem lokalen Koordinatensystem durch
- Verschiebe den lokalen Ursprung $\mathbf{c}$ so, daß

$$d = \min\left\{\mathbf{n}^i \cdot (\mathbf{p}^i - \mathbf{c})\right\}$$

möglichst weit positiv wird. Gesucht ist das optimale $\mathbf{c}$.

  - Frage: wird Rotation noch etwas bringen?
  - In praxi: probiere Mittelpunkt und Ecken der BBox der Polygone als $\mathbf{c}$.
- Speichere $\mathbf{c}$ zusätzlich zum Cluster; teste dann

$$\text{if } (\mathbf{e} - \mathbf{c}; \mathbf{N}, \mathbf{M}) \cdot (\mathbf{e} - \mathbf{c}) - d \leq 0$$

Hierarchical Clustered Backface Culling

- Zwei Cluster kann man zusammenfassen zu einem gemeinsamen:

$$\hat{\mathbf{n}} = \begin{pmatrix} \min\left(n_x^1, n_x^2\right) \\ \min\left(n_y^1, n_y^2\right) \\ \min\left(n_z^1, n_z^2\right) \end{pmatrix} \quad \hat{\mathbf{m}} = \begin{pmatrix} \max\left(m_x^1, m_x^2\right) \\ \max\left(m_y^1, m_y^2\right) \\ \max\left(m_z^1, m_z^2\right) \end{pmatrix}$$

$$\hat{d} = \min\left(d_1, d_2\right)$$

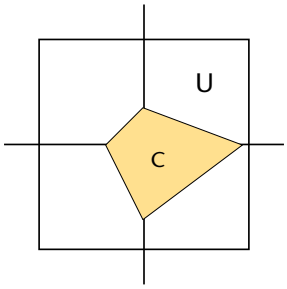
  - Diese beiden Vektoren und Ď liefern eine konservative Abschätzung
  - D.h.: falls der gemeinsame Cluster unsichtbar ist, dann auch garantiert die beiden ursprünglichen → Cluster-Hierarchie
- Falls eine Hierarchie von Clustern aufgebaut wird, definiere analog folgenden Frontface-Test:
  - Höre auf zu testen, falls kompletter Cluster front- oder backfacing
  - Sonst: teste die Kinder auf komplett front- oder back-facing

Erzeugung der Cluster

- Zur Bewertung von Cluster-Kandidaten in einem Algo benötigen wir ein Maß für die "Güte" eines Clusters
- Hier: Wahrscheinlichkeit, daß Cluster gecullt wird
- Heuristik:

$$\frac{\text{Volumen aller möglicher Viewpoint-Standort}}{\text{Volumen des Culling-Volumens}} = \frac{Vol(U)}{Vol(C)}$$

  - Vol(*C*) kann man exakt bestimmen
  - Wähle als *U* die BBox der gesamten Szene
- Falls lokale Culling-Koordinaten verwendet werden:
  wähle als $U = c \cdot \text{BBox(Cluster)}$
  („Nahfeld-Culling-Wahrscheinlichkeit“)



- Frage: gegeben zwei Cluster *A,B*;
  ist es schneller *A* und *B* getrennt zu testen und zu rendern, oder als ein Cluster $C := A \cup B$?
- Sei *T*(*A*) die erwartete(!) Zeit, um den Cluster *A* zu testen und ggf. zu rendern. Dann ist

$$T(A) = t_C + (1 - P(A))\, R(A)$$

  wobei *P*(*A*) = Wahrscheinlichkeit, daß Cluster *A* gecullt wird, und *R*(*A*) = Zeit zum Rendern von *A* (ohne weitere Tests), und t_C = Zeit zum Backface-Test eines Clusters

- A und B zusammenfassen lohnt sich gdw.

$$T(C) < T(A) + T(B) \quad \Leftrightarrow$$

$$t_C + (1 - P(C))\,R(C) < 2t_C + (1 - P(A))\,R(A) + (1 - P(B))\,R(B) \quad \Leftrightarrow$$

$$(1 - P(C))\,(R(A) + R(B)) < t_C + \ldots \quad \Leftrightarrow$$

$$P(C) > \frac{t_C + P(A)R(A) + P(B)R(B)}{R(A) + R(B)} \quad \Leftrightarrow$$

$$P(C) > \frac{\frac{t_C}{r} + P(A)n_A + P(B)n_B}{n_A + n_B}$$

Ann.: R(A) = n_A·r, r=konstanter Aufwand für 1 Polygon

- Verhältnis t_C/r ist maschinenabhängig; kann aber leicht vorab experimentell und automatisch bestimmt werden
  (Hängt ab von Graphikkarte, Anzahl Lichtquellen, Texturen, ...)