

# Computer-Graphik 2

## Visibility Computations II – Culling

G. Zachmann

Clausthal University, Germany

[zach@in.tu-clausthal.de](mailto:zach@in.tu-clausthal.de)



# Klassifikation (Erinnerung)



- Problemklassen innerhalb des Bereichs "Visibility Computations":
  1. *Hidden Surface Elimination (Verdeckungsrechnung)*: 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



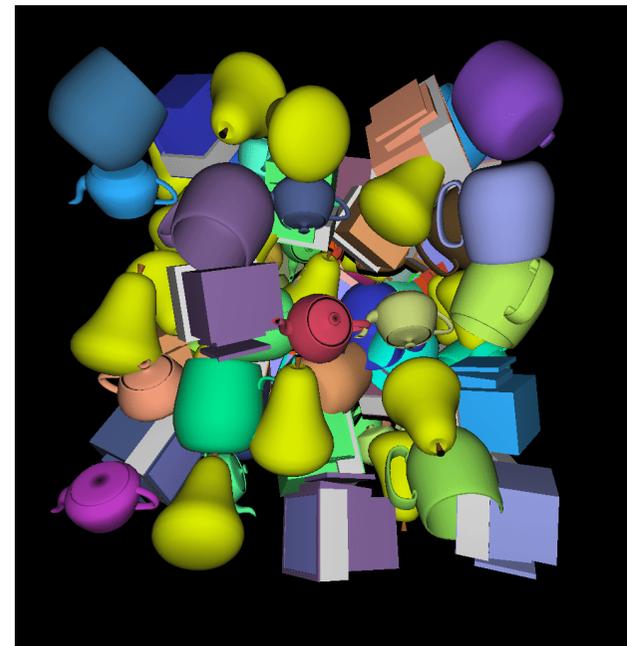
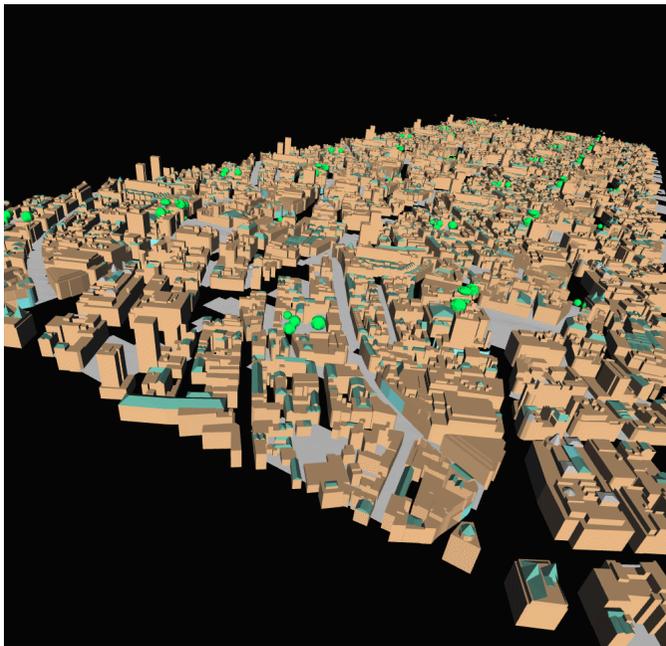
# Culling



- 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 ( $|S| \ll |A|$ ) !



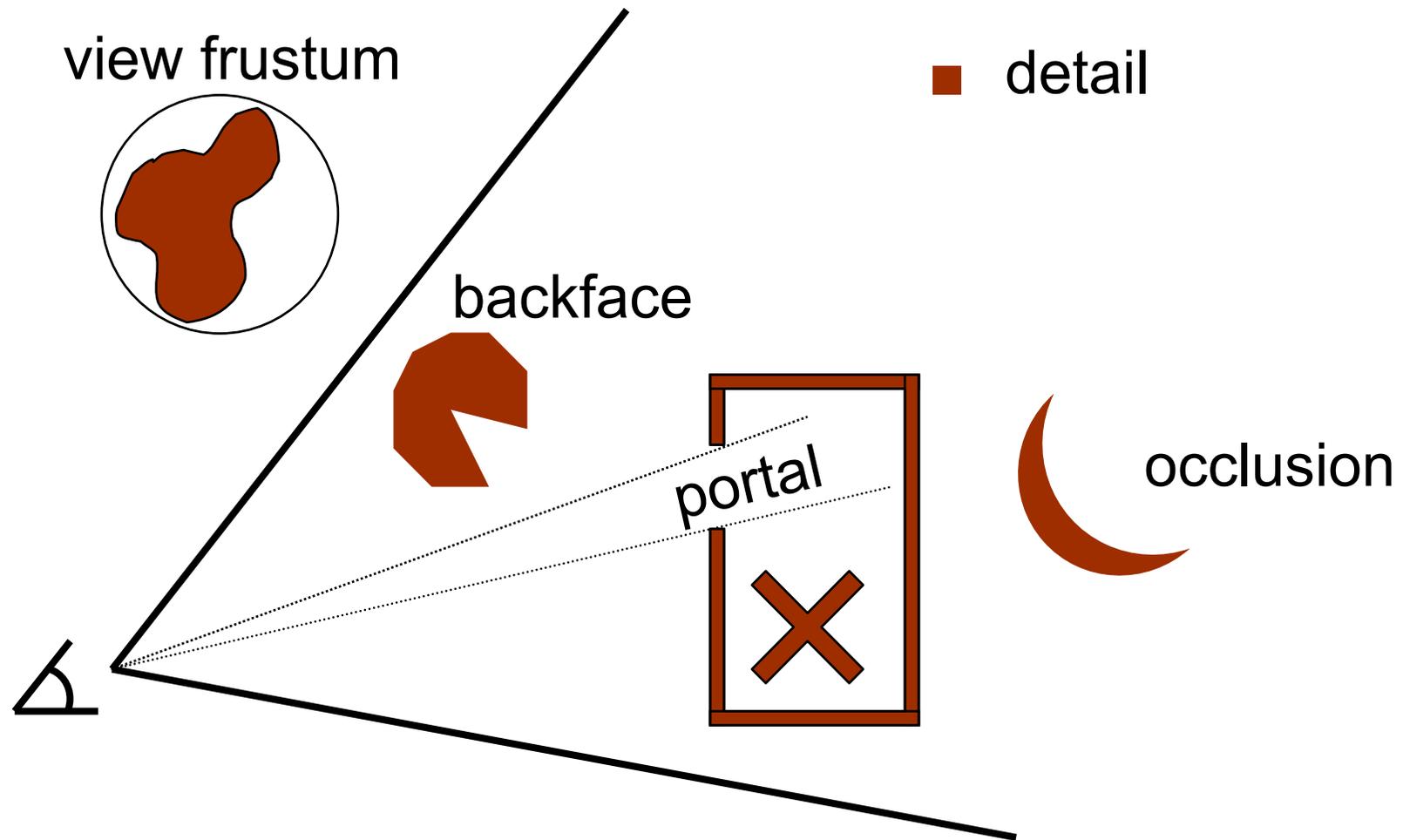
- Culling ist eine wichtige Optimierung (im Gegensatz zu Clipping)



- Für  $|S| \ll |A|$  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  $A$



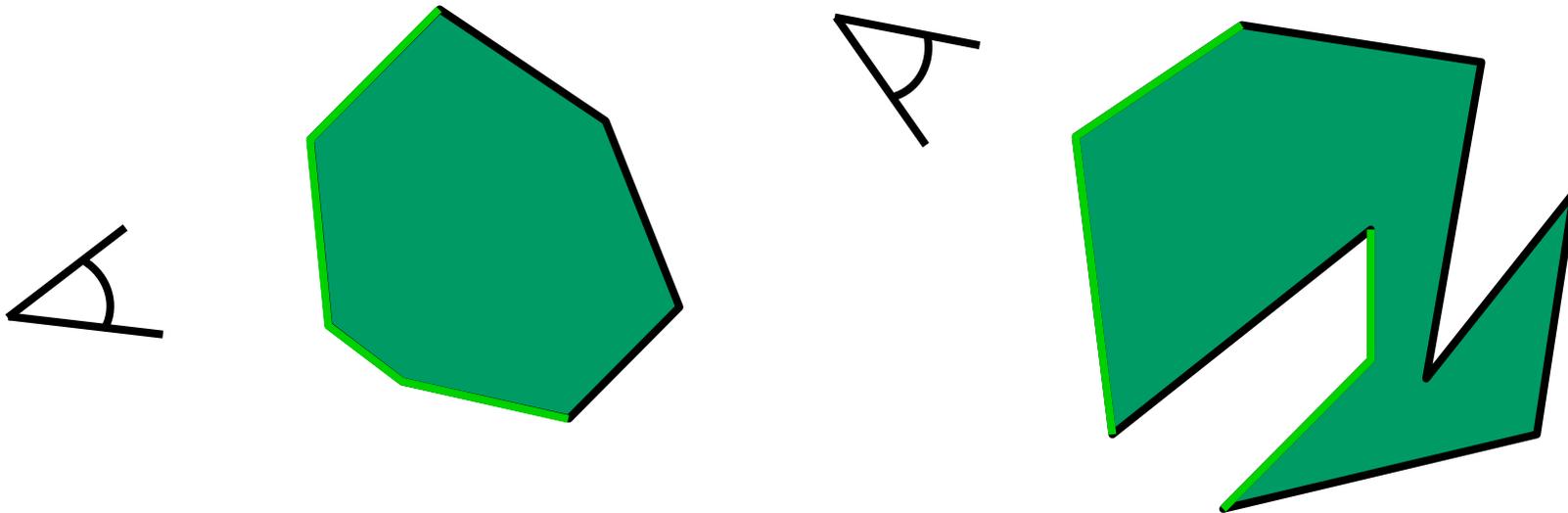
# Culling-Arten





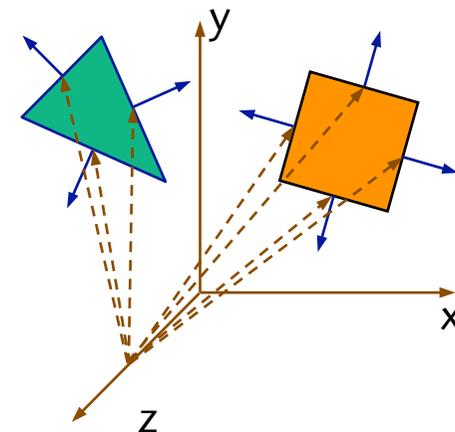
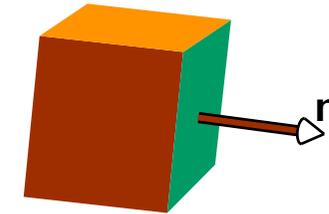
# Back-Face Culling

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





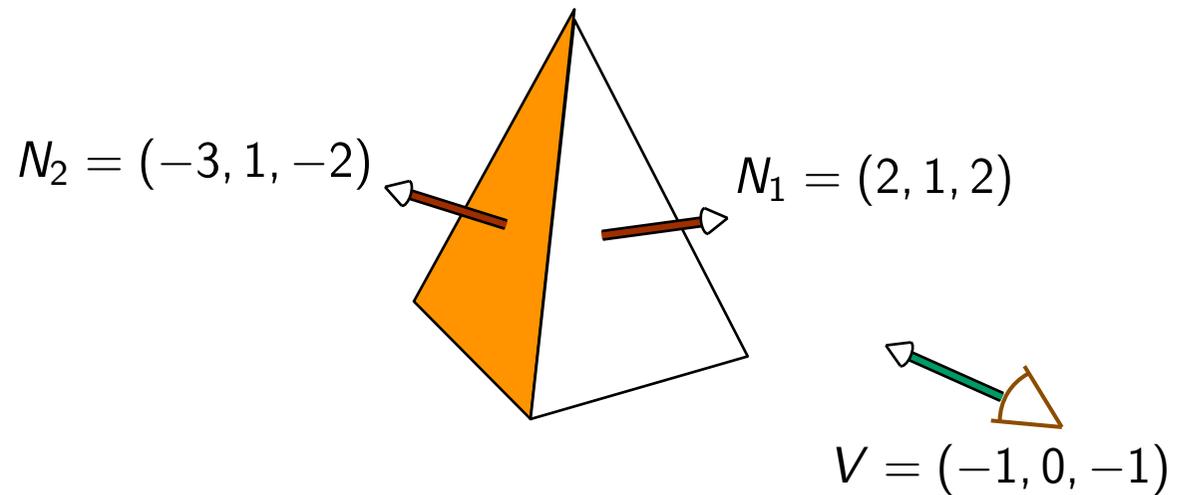
# Beispiel

$$\begin{aligned} N_1 \cdot V &= (2, 1, 2) \cdot (-1, 0, -1) \\ &= -4 < 0 \end{aligned}$$

$\Rightarrow N_1$  front facing

$$\begin{aligned} N_2 \cdot V &= (-3, 1, -2) \cdot (-1, 0, -1) \\ &= 5 > 0 \end{aligned}$$

$\Rightarrow N_2$  back facing





## Backface Culling in OpenGL

- Muß man nur einschalten:

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

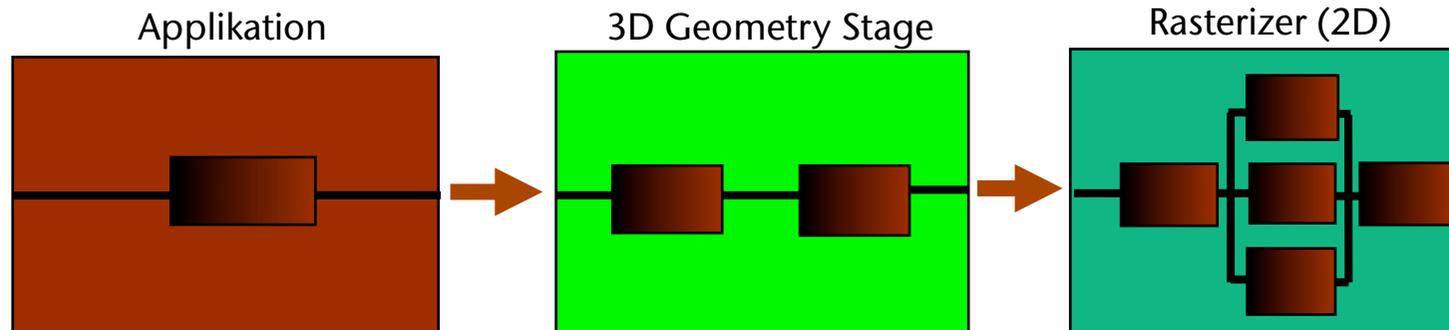




# Wann lohnt sich Backface Culling?



- 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)

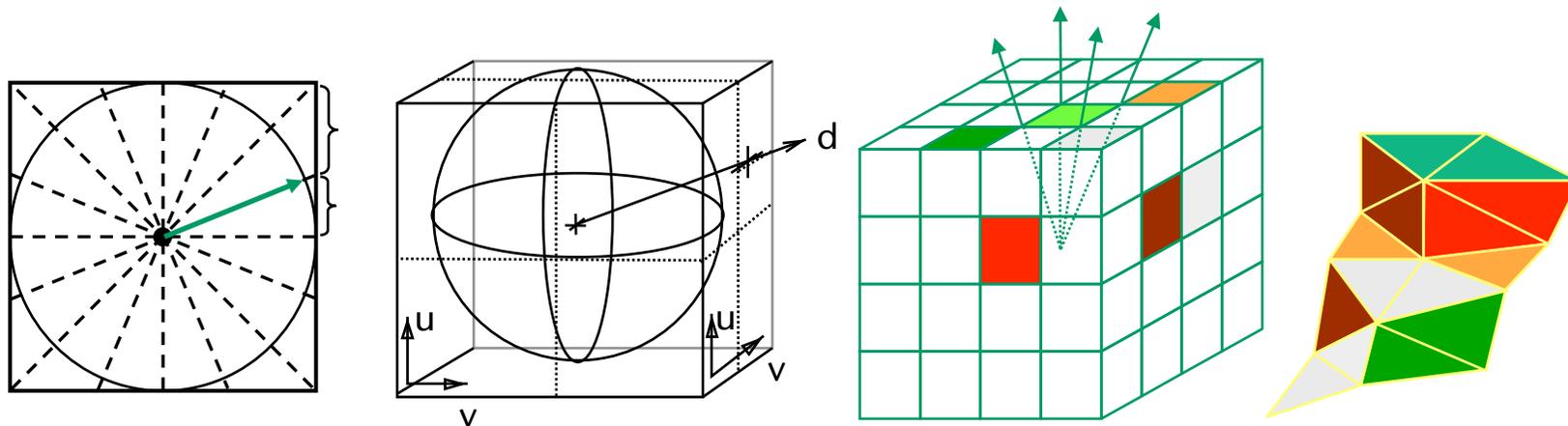


# Normalenmasken

[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")

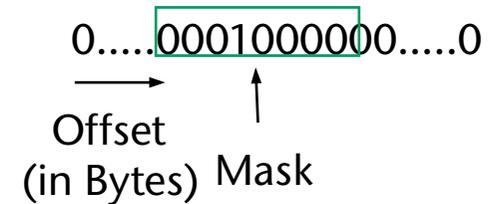


- Ergibt  $6 \cdot N^2$  viele Klassen ( $N$  = Anzahl Gitterunterteilungen)
- Klassifizierung einer Normalen ist sehr einfach



- 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  $256 \times 8 = 2048$  Bits

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



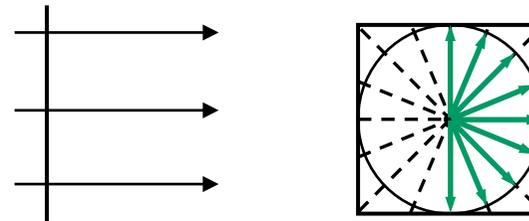
- Speichere diese 2 Bytes zu jedem Polygon
- Unterteile die Normalenkugel in  $6 \cdot 16 \cdot 16 = 1536$  Klassen



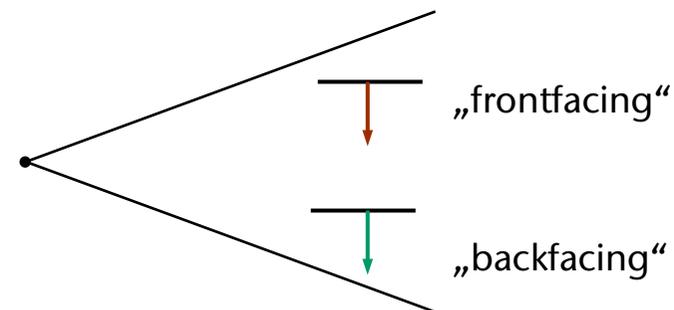
- Culling (Initialisierung):

- Identifiziere alle diejenigen **Normalenklassen**, deren Normalen alle backfacing sind

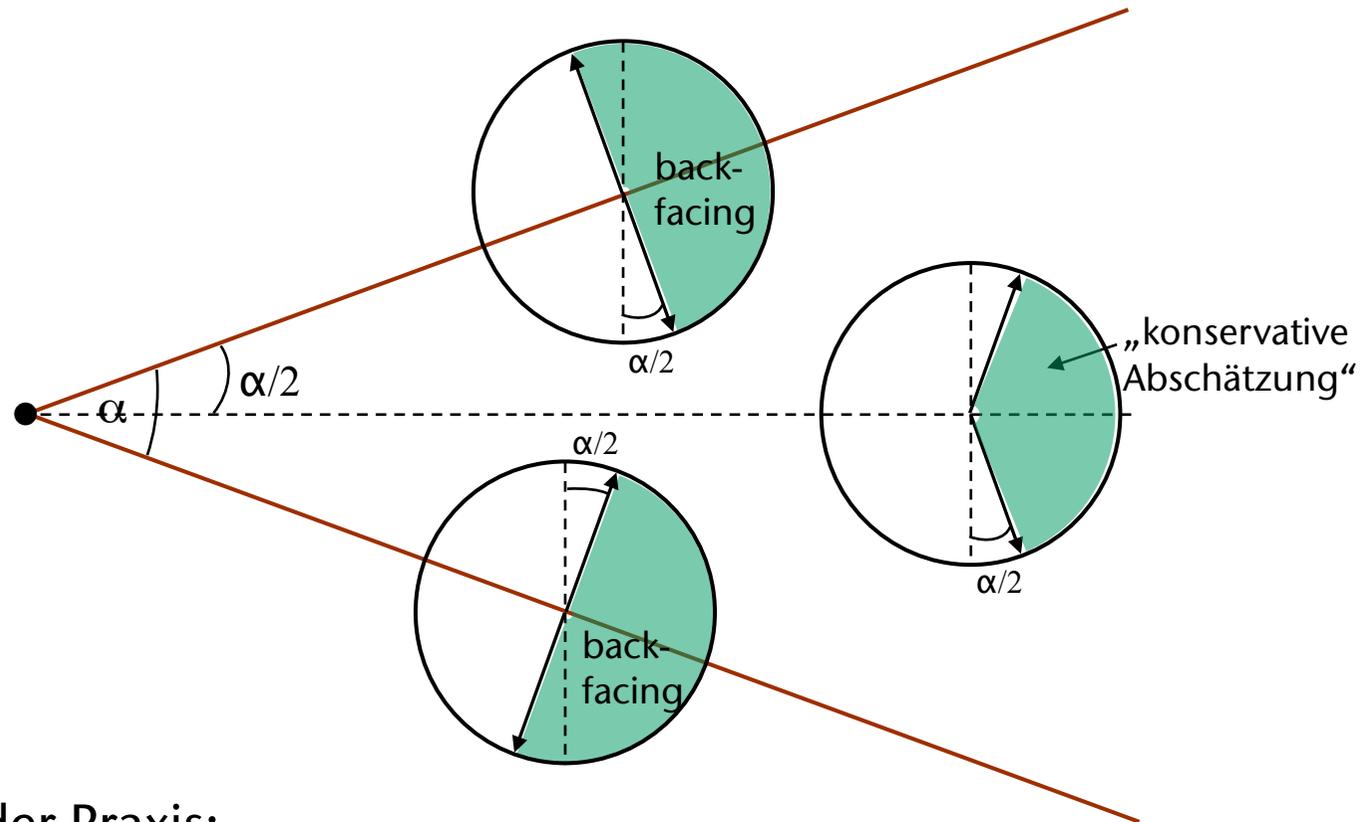
- Orthographische Projektion:



- 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 Bytearray 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



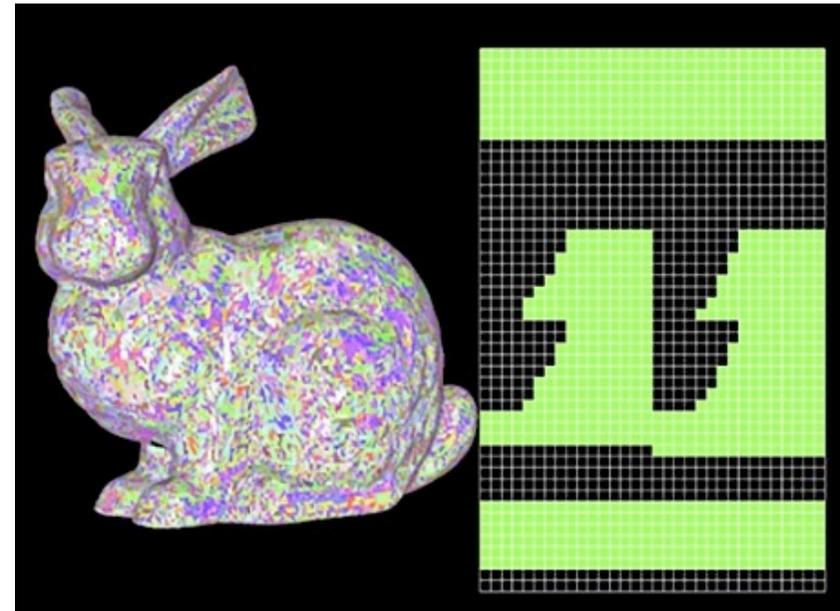
# Beispiel



216 Klassen (cluster)



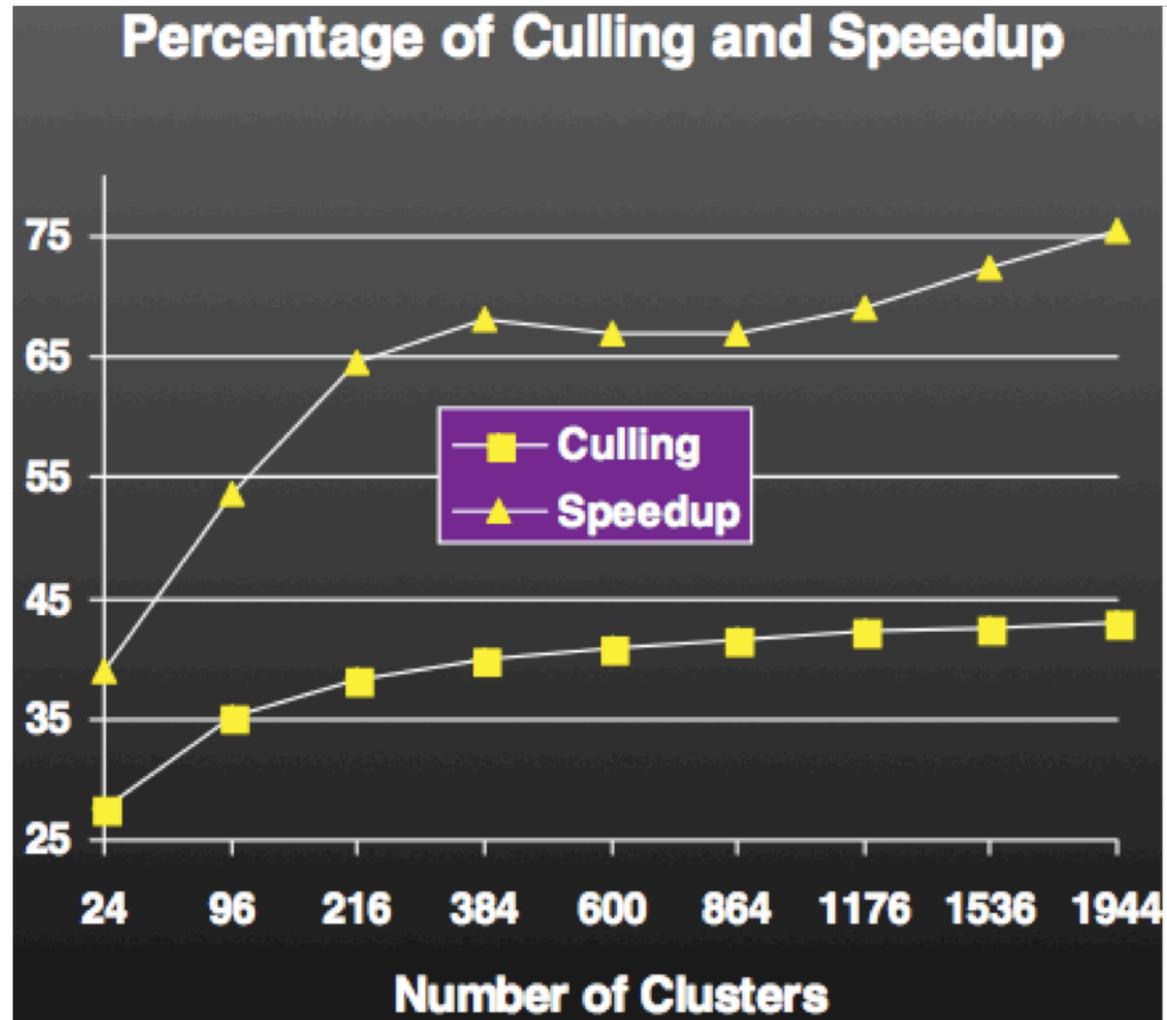
1536 Klassen (cluster)



BackMask für den aktuellen Viewpoint  
(grün = backfacing)



# Speedup





# Clustered Backface Culling

[1998]



- Erinnerung: einfache Rechenregeln zu min/max:

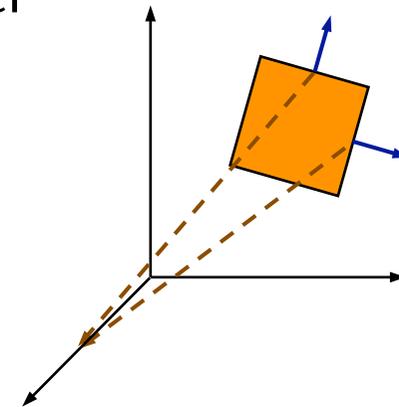
$$\max(x + y) \leq \max(x) + \max(y)$$

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

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

- 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.

$$\begin{aligned} \forall i : \mathbf{n}^i (\mathbf{e} - \mathbf{p}^i) \leq 0 & \Leftrightarrow \\ \max \{ \mathbf{n}^i (\mathbf{e} - \mathbf{p}^i) \} \leq 0 & \end{aligned} \quad (1)$$

- Obere Schranke für (1) ist

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

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

$$\begin{aligned} \max \{ \mathbf{n}^i (\mathbf{e} - \mathbf{p}^i) \} & \leq \max \{ e_x n_x^i + e_y n_y^i + e_z n_z^i \} - d \\ & \leq \max \{ e_x n_x^i \} + \max \{ e_y n_y^i \} + \max \{ e_z n_z^i \} - d \end{aligned} \quad (3)$$



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

$$\begin{aligned} & \max \{ \mathbf{n}^i (\mathbf{e} - \mathbf{p}^i) \} \\ & \leq e_x \cdot \max\{n_x^i\} + e_y \cdot \max\{n_y^i\} + e_z \cdot \max\{n_z^i\} - d \\ & \leq \mathbf{m} \cdot \mathbf{e} - d, \quad \text{mit } \mathbf{m} = \begin{pmatrix} \max\{n_x^i\} \\ \max\{n_y^i\} \\ \max\{n_z^i\} \end{pmatrix} \end{aligned}$$

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

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



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

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

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

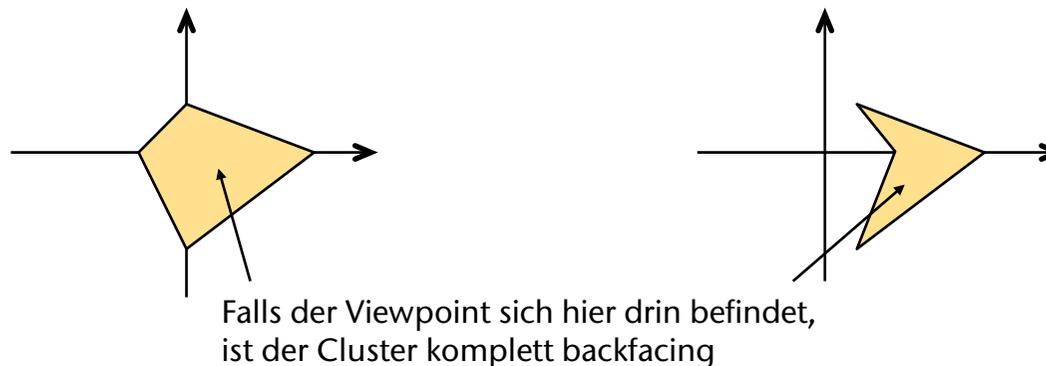
$$\text{if}(\mathbf{e}; \mathbf{n}, \mathbf{m}) \cdot \mathbf{e} - d \leq 0 \quad \Rightarrow \quad \text{cluster is backfacing} \quad (4)$$

- Precomputation: pro Cluster  $\mathbf{n}$ ,  $\mathbf{m}$  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  $e_x \geq 0$
  - Alle 4 Ebenen haben Normale der Form  $\mathbf{n} = (m_x, \cdot, \cdot) \rightarrow$
  - sie schneiden alle die X-Achse im Punkt  $(\frac{d}{m_x}, 0, 0)$ .
- Die 8 Ebenen bilden ein **geschossenes Volumen**, das sog. "Culling-Volumen"





## Hierarchical Clustered Backface Culling



- Zwei Cluster kann man zusammenfassen zu einem großen:

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

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

- 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



## 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 groß negativ  $\rightarrow$  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 groß 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

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

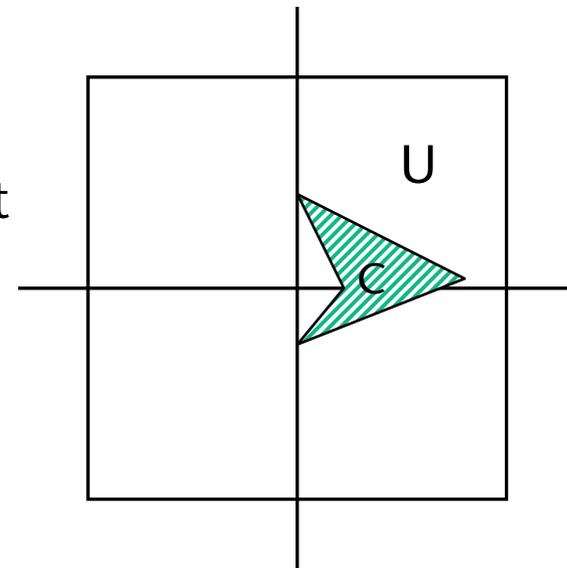


# Erzeugung der Cluster

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

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

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

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

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

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

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

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

Ann.:  
 $R(A) = n_A \cdot r$ ,  
 $r = \text{konstanter Aufwand für 1 Polygon}$

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