Q

Computergraphik

Vorlesung im Wintersemester 2013/14
Kapitel 6ü: Clipping, Teil 1

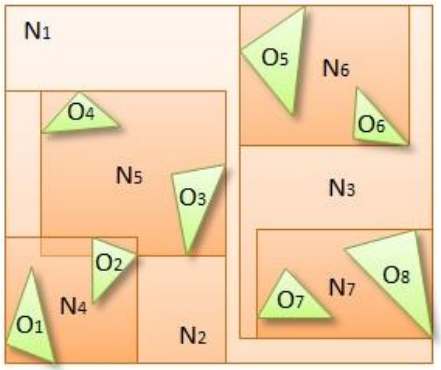
Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie

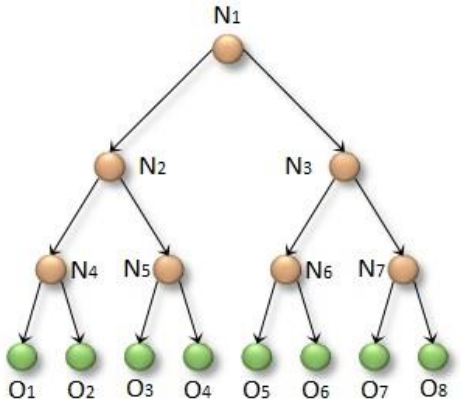
Organisatorisches

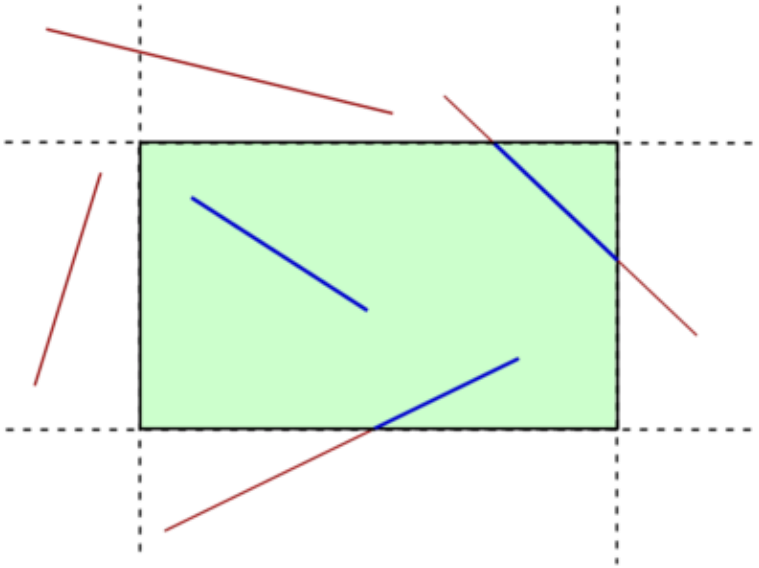
- Am 23.12. findet **keine** Übung statt
- Nächste Übung am 13.1.2014
- Übungsblatt 9 am 23.12.2013
- Übungsblatt 10 am 13.1.2014

Übersicht

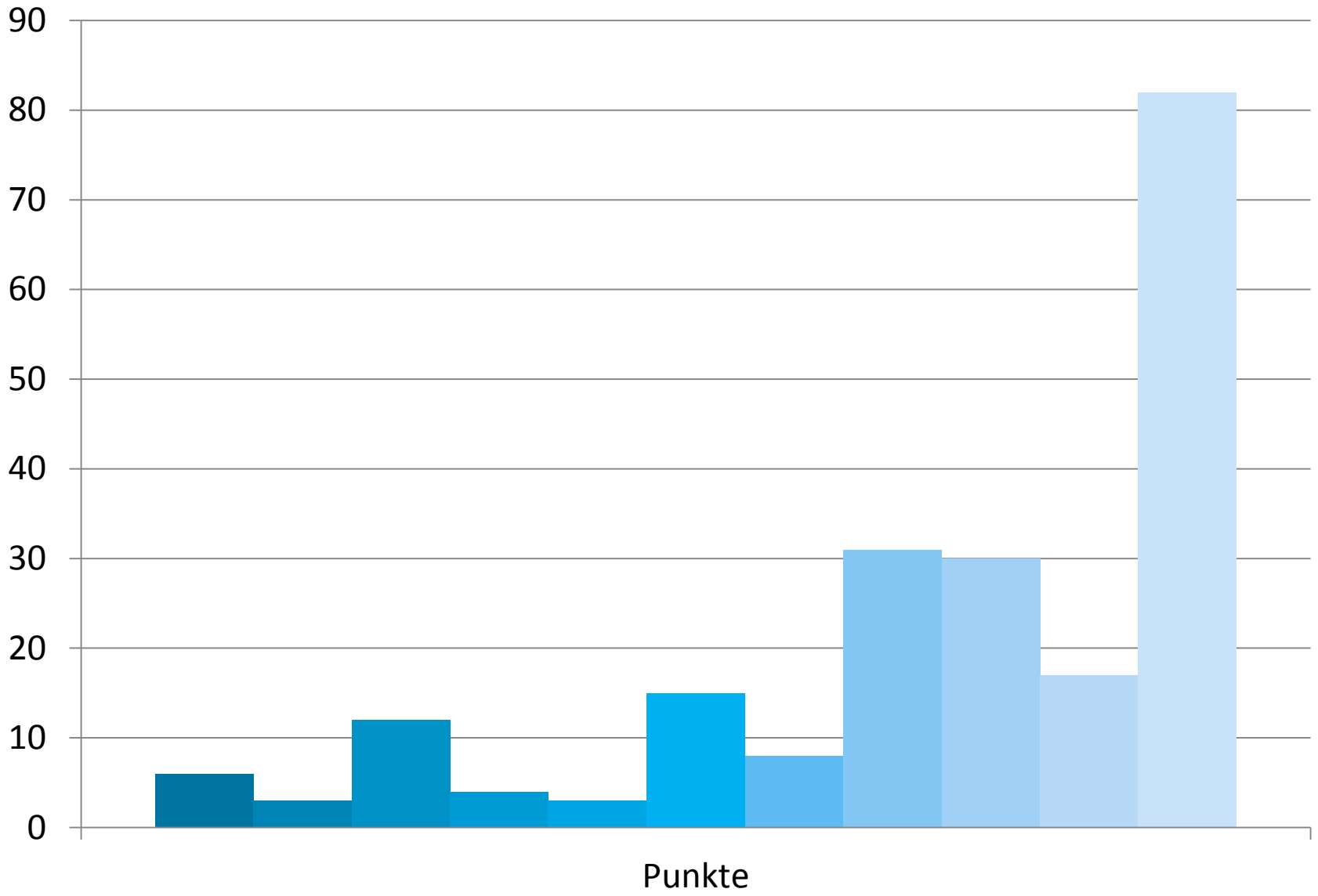
- Rückblick Blätter 6 und 7
- Übungsblatt 9 / Distributed Raytracing
- Bounding Volume Hierarchien
- Clipping Teil 1





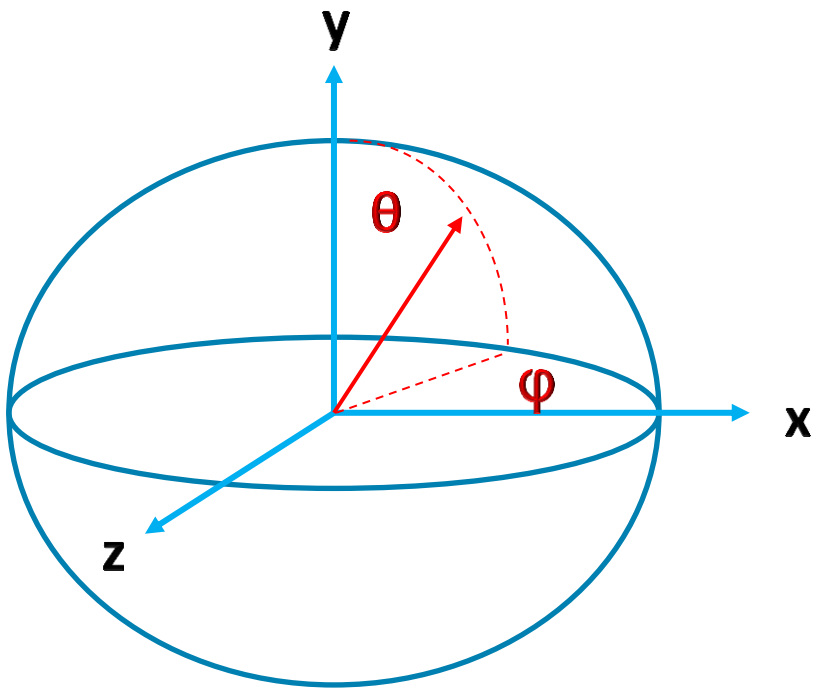


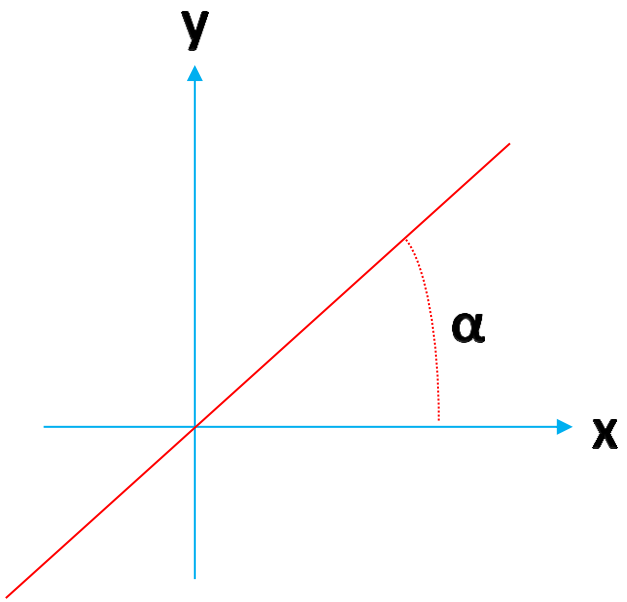
Ergebnisse ÜB 6



Übungsblatt 6

- Korrekturvorgehen leicht geändert, nicht abgegebene Dateien werden durch die leeren Dateien der Aufgabe ersetzt.
- Spherical mapping: Koordinatensysteme beachten, Beispiele im Netz (z.B. Wikipedia) haben andere Koordinatensysteme





α=atan2(y, x) α in [-π ;π]

$$u = \frac{\operatorname{atan2}(z, -x) + \pi}{2\pi}, v = \frac{\operatorname{acos} y}{\pi}$$

Übungsblatt 7

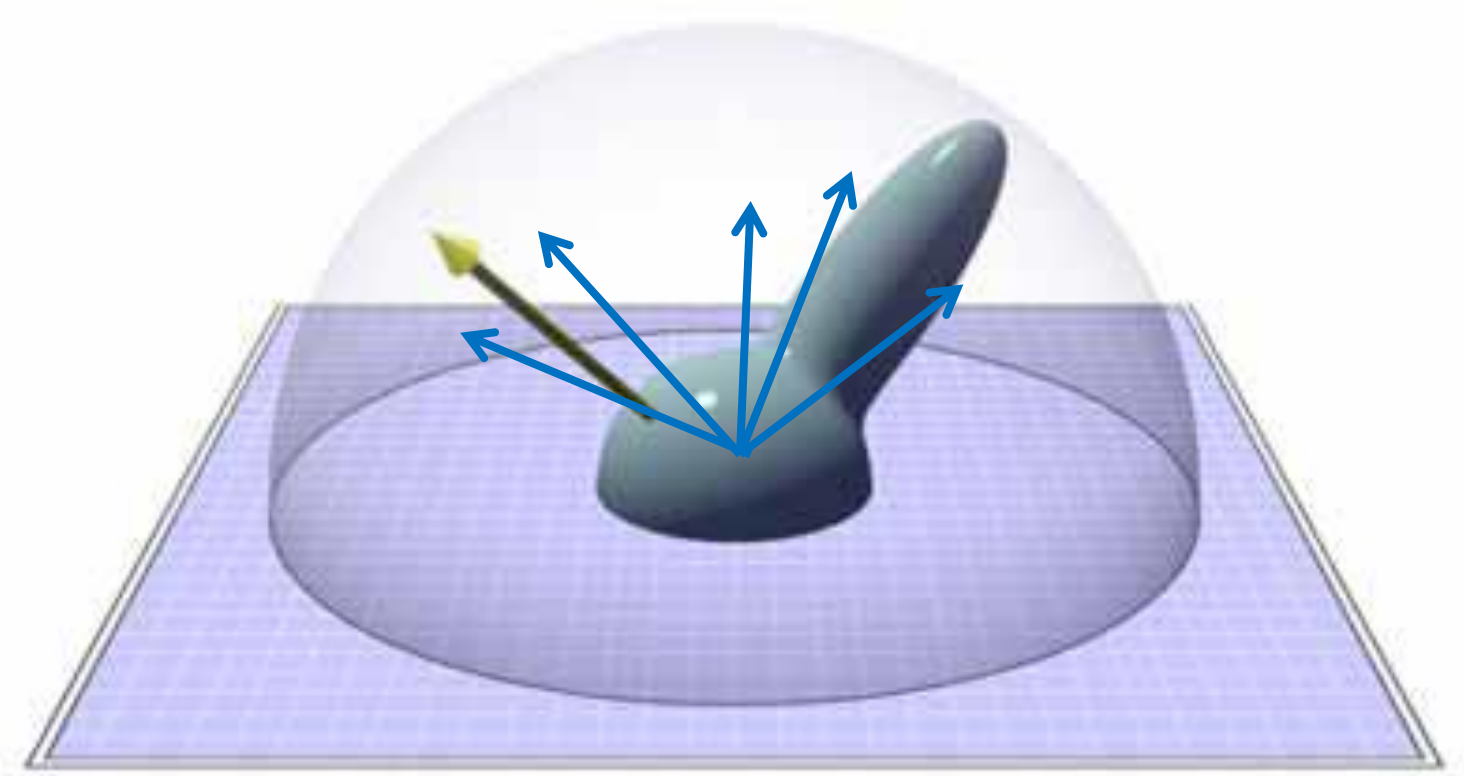
- Bestimmung des Footprints: *achsenparalleles* Rechteck
- Bestimmung des Mischfaktors beim Mip-mapping:
  - Beide Varianten volle Punktzahl (Footprint f):
    - $t = \mathrm{frac}(\log_2 f)$
    - $t = \frac{f - 2^l}{2^u - 2^l}$

Übungsblatt Nr. 9

- Am 23.12.2013
- Abgabe am 6.1.2014
- Das Blatt ist *optional,* kann aber zur Verbesserung/Ausgleich dienen:
  - Es zählen nicht zu den 100% der Punkte
  - Aufgaben mit 0 Punkten können ausgeglichen werden
- Aufgabe: Implementieren Sie *bis zu 3* der folgenden Effekte
  - Bewegungsunschärfe
  - Tiefenunschärfe
  - Weiche Schatten
  - Environment-Map Beleuchtung (Image-Based lighting)
  - Unscharfe Reflexion
  - Unscharfe Transmission
- Für jeden richtig implementierten Effekt 5 Punkt, max. 15 Punkte

Distributed Ray Tracing

- Bisher: immer nur ein Strahl weiterverfolgt
  - Ausname: (teil-)transparente Objekte
  - Kann viele Effekte nicht darstellen
- Abhilfe: mehrere zufällige Strahlen pro Schnittpunkt verfolgen (**Distributed Ray Tracing**)



Monte-Carlo-Integration

- Distributed Raytracing = Mittelwert über mehrere Strahlen
- Das ist ein Integral!

$$I = \int_{\Omega} f(\omega)\, d\omega$$

- Lösen über Monte-Carlo-Integration:

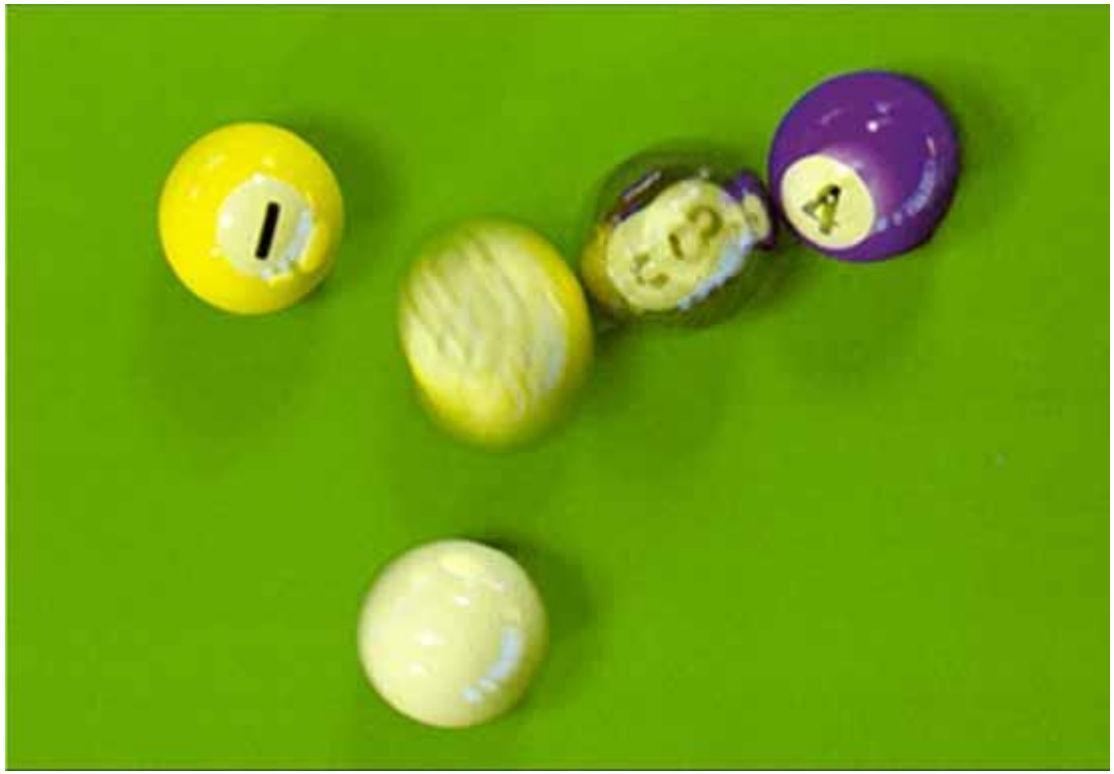
$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(\omega_i)}{p(\omega_i)}$$

- Beispiel: Man möchte uniform über die Hemisphäre integrieren:

$$I \approx \frac{2\pi}{N} \sum_{i=1}^{N} f(\omega_i)$$

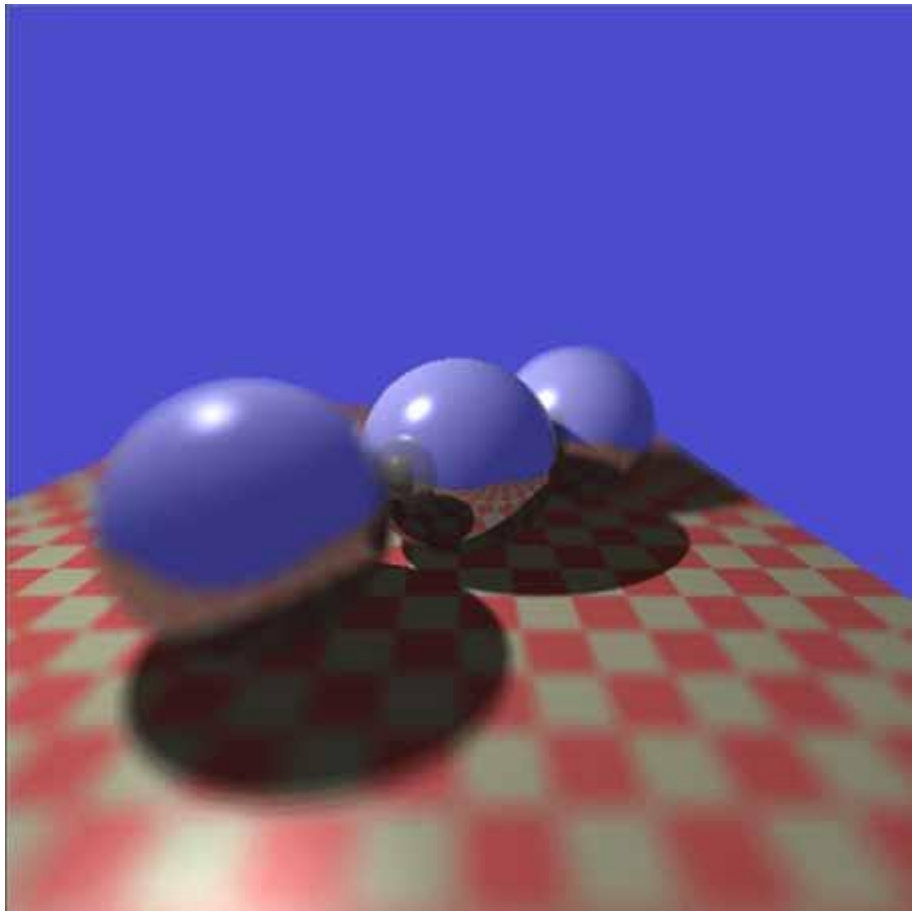
Bewegungsunschärfe

- Wichtig bei Animationen
- Berechne nicht nur Bild für festen Zeitpunkt, sondern an mehreren Zeitpunkten.
- Berechne dann Ausgabe als Mittelwert über die Zeitpunkte.
- Vgl. Belichtungszeit bei realen Kameras



Tiefenunschärfe

- Ermöglicht es, die Szenentiefe besser abzuschätzen.
- Nutze das Thin Lens Model statt dem Lochkameramodell
- Erzeuge mehrere Primärstrahlen und bilde den Mittelwert



Weiche Schatten

- Reelle Lichtquellen haben eine Ausdehnung → Weiche Schatten
- Erzeuge mehrere Schattenstrahlen zu zufällig gewählten Punkten auf der Flächenlichtquelle.
- Verschattung = #Unverschattet / #Verschossen

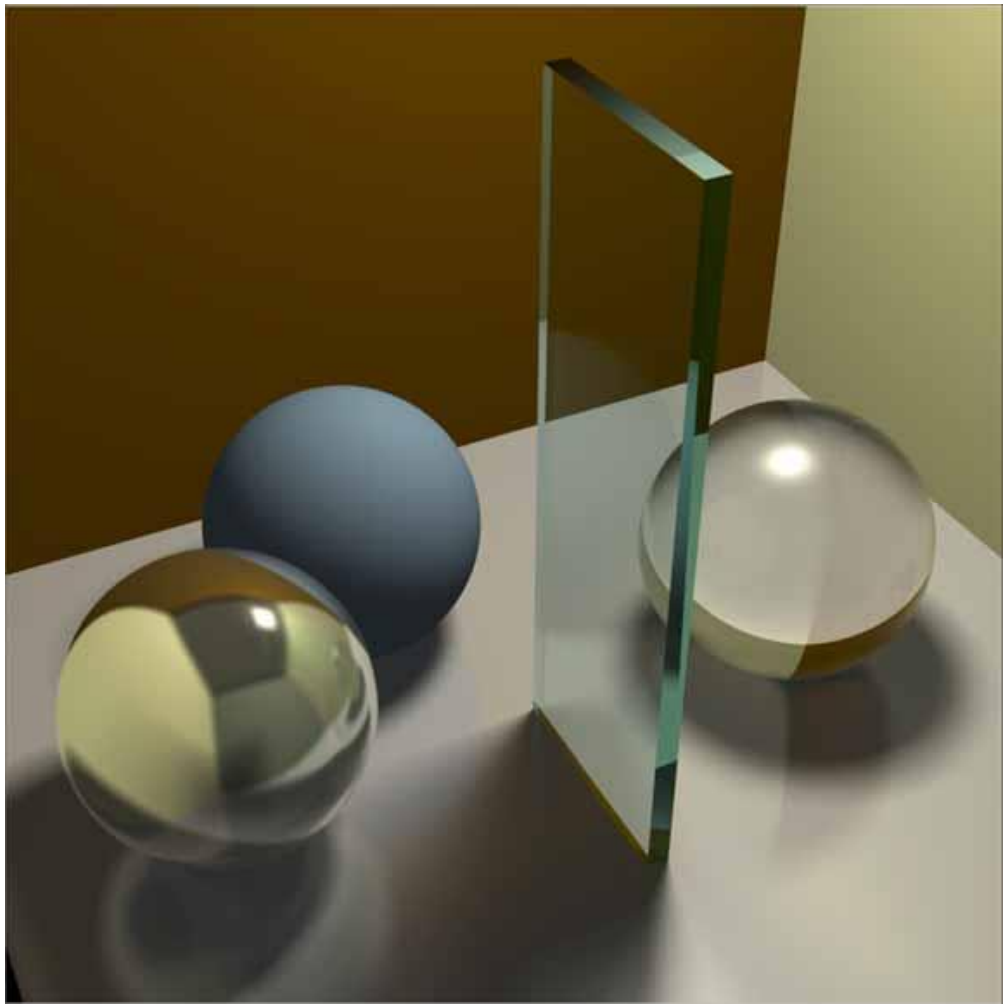
Umgebungsbeleuchtung

- Nutze Environment Map mit Kugelparametrisierung
- Erzeuge mehrere Schattenstrahlen
- Für unverschattete Strahlen greife mit der Strahlrichtung auf die Environment Map zu
- Lichtstärke ist der Wert der Environment Map

Tipp: Freie Environment Maps findet man auf http://www.openfootage.net

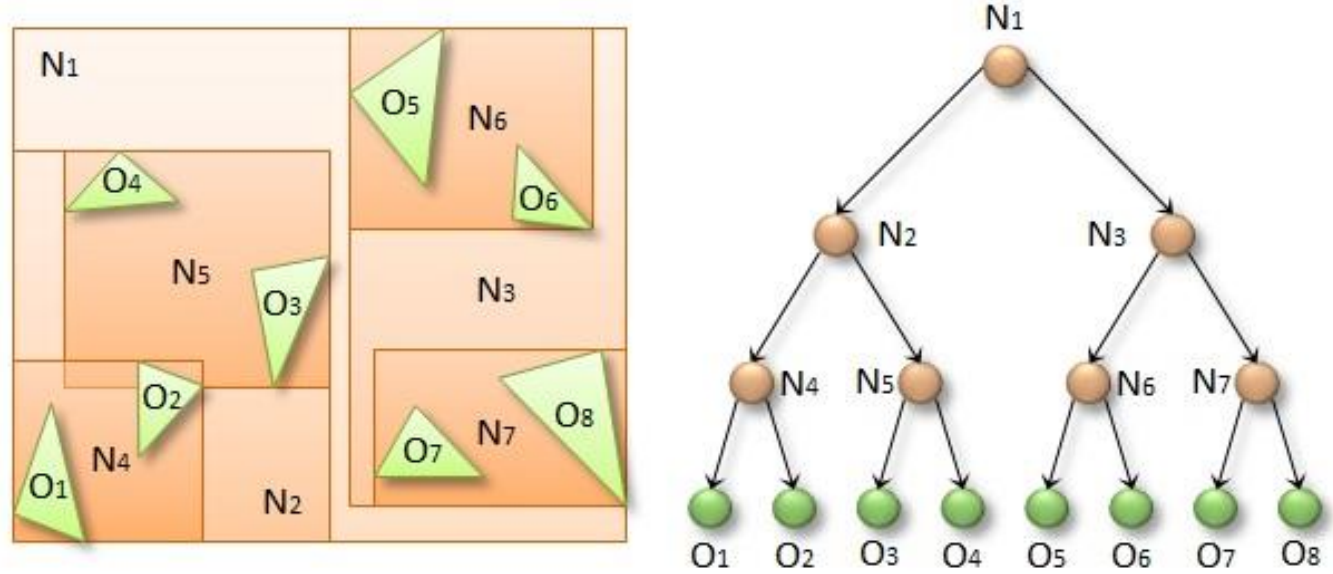
Unscharfe Reflexion / Transmission

- Erzeuge mehrere zufällige Sekundärstrahlen
- Achtung: Strahlenbaum ‘explodiert’



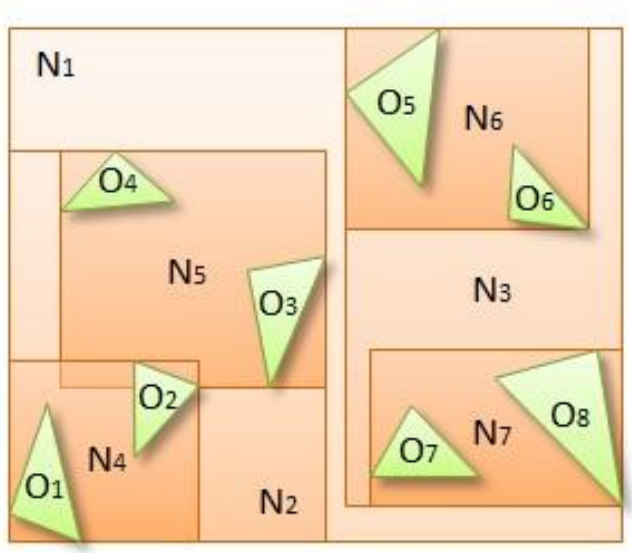
Bounding Volume Hierarchies

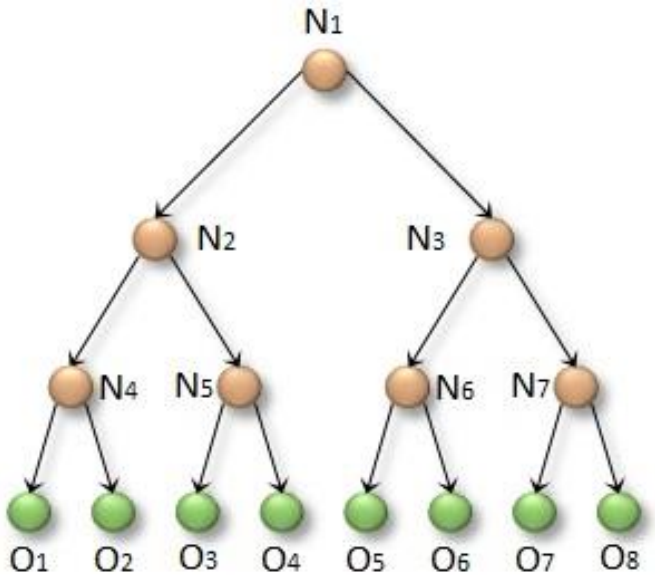
- Sichtbarkeitstests mit allen Primitiven ist langsam
- Idee:
  - Erzeuge Datenstruktur, die es ermöglicht viele Strahlschnitttests zu vermeiden
- Es gibt viele solcher Datenstrukturen
  - Kd-, BSP- oder Octree
  - Bounding Volume Hierarchie



Bounding Volume Hierarchies

- Erzeuge Hierarchie von einhüllenden Volumen, z.B.
  - AABBs
  - Kugeln
- Teile die Primitive rekursiv in Teilmengen
  - Spatial Median oder Object Median
  - Surface Area Heuristik
- Teile entlang der Achse mit der größten Ausdehnung oder zyklisch

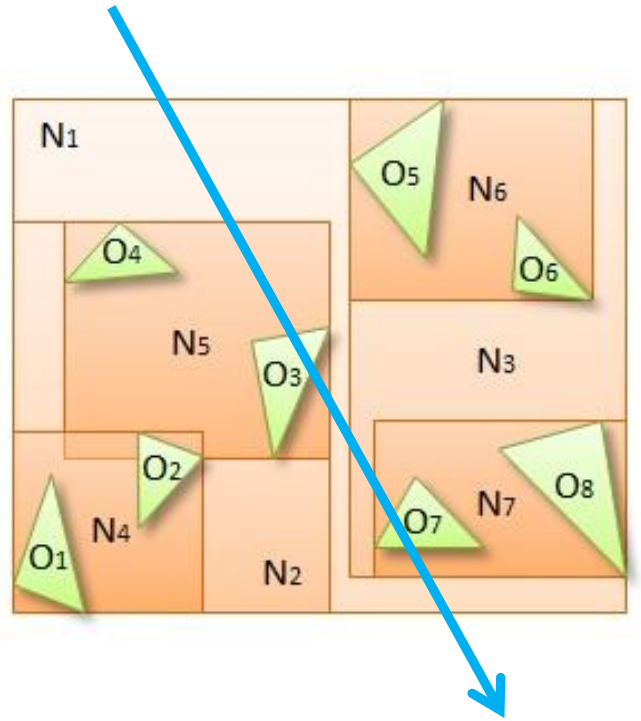


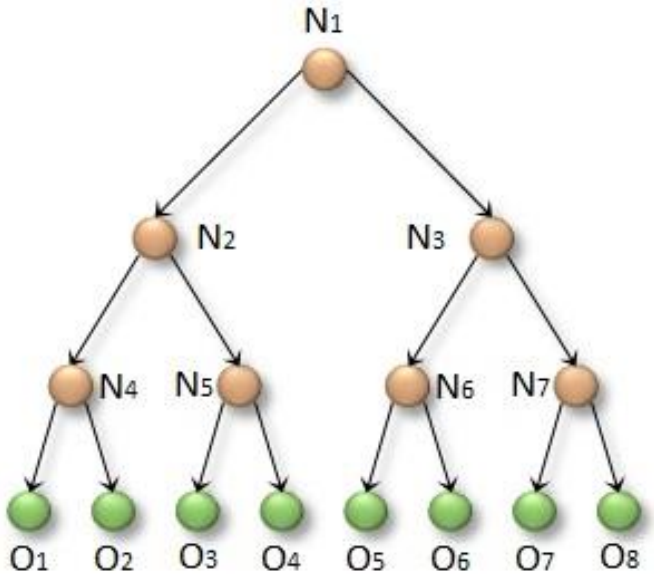


Traversierung

- Prüfe auf Schnitt mit BV der Wurzel
  - Steige rekursiv ab und prüfe BV der Kindknoten
  - Wenn Blattknoten: Prüfe alle zugehörigen Primitive

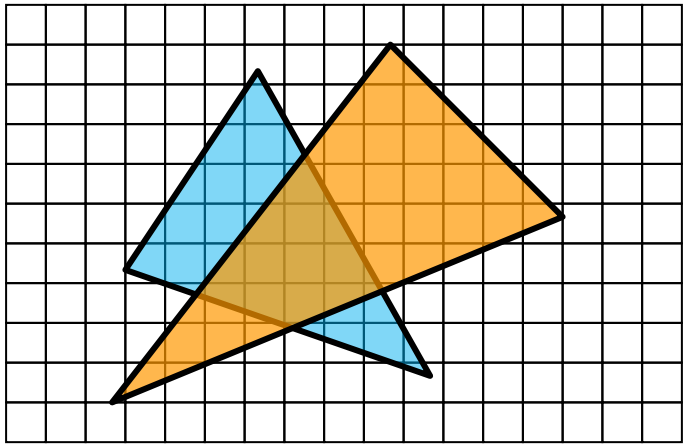
- Wichtig: Nicht den ersten gefundenen Schnittpunkt zurückgeben

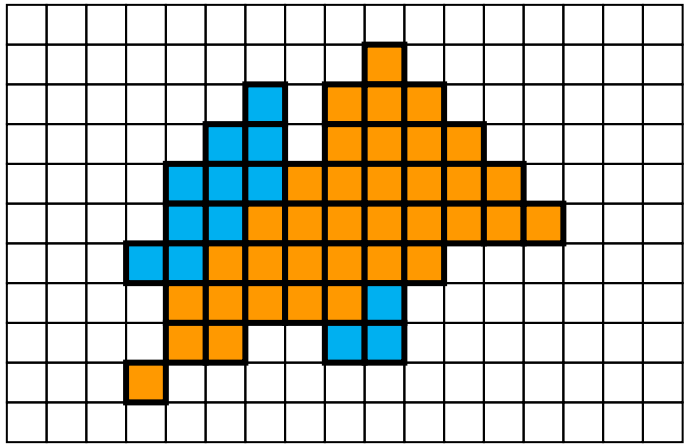




Clipping

- Clipping von Linien und Polygonen
  - Cohen-Sutherland und α-Clipping (Teil 1)
  - Sutherland-Hodgeman (Teil 2)



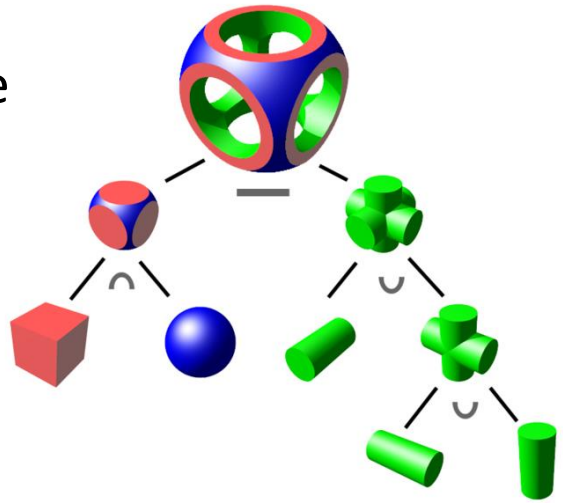


Clipping

- Clipping: Abschneiden von Linien/Polygon-Teilen die außerhalb des Bildschirms liegen (oder allg. außerhalb eines gewünschten Bereichs)

- warum ist Clipping wichtig?
  - Effizienz: zeichne nichts außerhalb des Bildschirms/View Frustums, keine Objekte hinter der Kamera (in 3D)
  - vermeide problematische Fälle bei 3D Projektionen (später mehr)

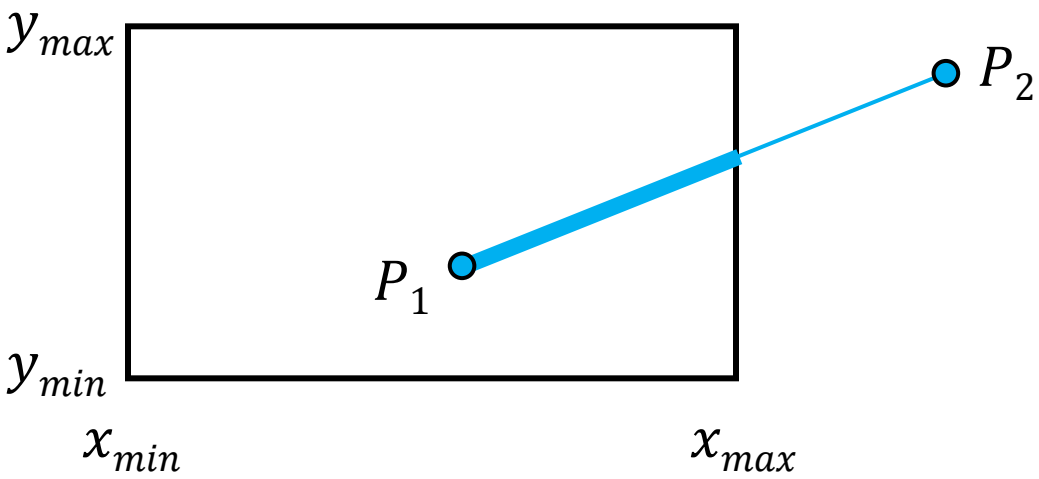
- wichtig nicht nur für Rasterisierung, beispielsweise auch für Constructive Solid Geometry, Boolsche Operationen in 2D und 3D



- unterschiedliche Ansätze
  - Clipping on-the-fly während der Rasterisierung
    - z.B. Laufvariable für x-Achse beschränken auf $0 \leq x \leq$ Breite
  - besser: analytisches Clipping (dieses Kapitel)

Clipping von Linien

- gegeben
  - eine 2D Linie von $P_1 = (x_1, y_1)$ nach $P_2 = (x_2, y_2)$
  - ein Rechteck definiert durch $(x_{min}, x_{max}, y_{min}, y_{max})$
- bestimme den Teil der Linie innerhalb des Rechtecks **effizient**
  - vermeide tatsächliche Schnittpunktberechnung soweit möglich



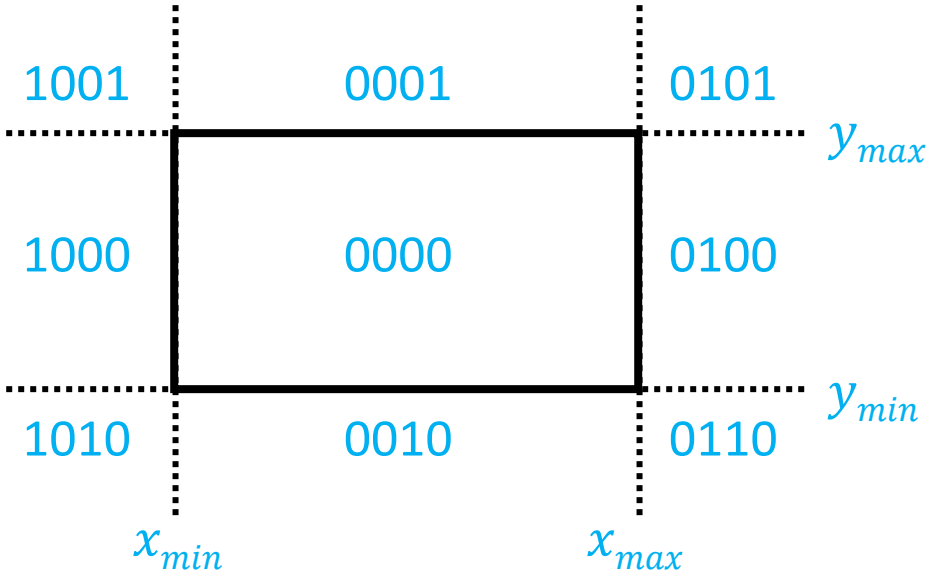
Clipping Algorithmus von Cohen-Sutherland

Clipping von Linien gegen Rechtecke

- Idee:
  - vermeide tatsächliche Schnittberechnung (soweit möglich)
  - teste Endpunkte auf trivial accept (Liniensegment im Rechteck)
  - teste Lage der Linie auf trivial reject
    (Segment außerhalb bzgl. einer Kante des Rechtecks)
  - wenn weder trivial accept noch trivial reject eintritt
    - unterteile das Liniensegment
    - teste auf trivial accept/reject

Clipping Algorithmus von Cohen-Sutherland

- unterteile die Ebene in 9 Bereiche



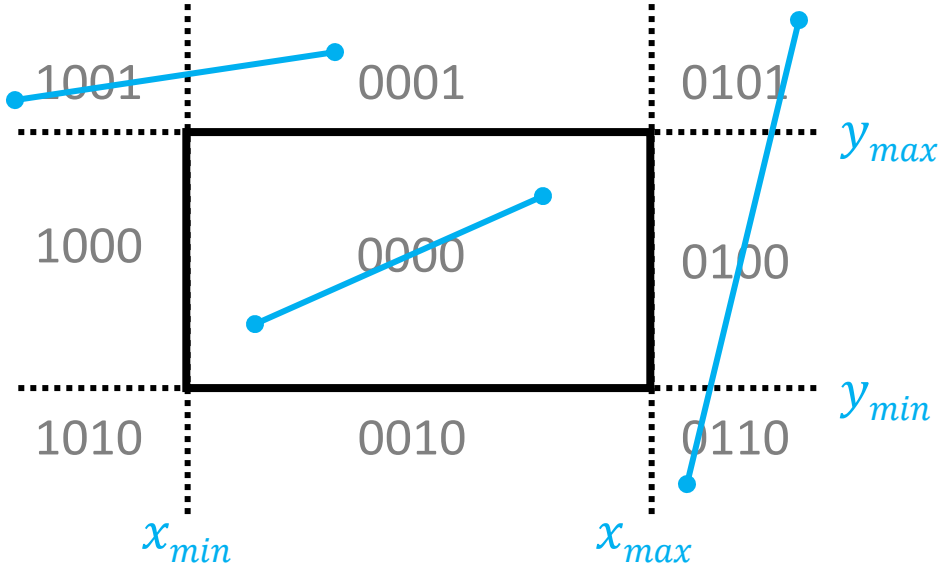
- $\Rightarrow$ “Outcodes”
  - die Lage eines Punktes wird durch einen 4 Bit Code beschrieben
  - jedes Bit entspricht einer Kante des Clipping Rechtecks
    - Bit gesetzt $\leftrightarrow$ Punkt liegt ausserhalb bzgl. dieser Kante

- Outcode =

$x < x_{min}$	$x > x_{max}$	$y < y_{min}$	$y > y_{max}$

Clipping Algorithmus von Cohen-Sutherland

Algorithmus zum Clipping von Linien

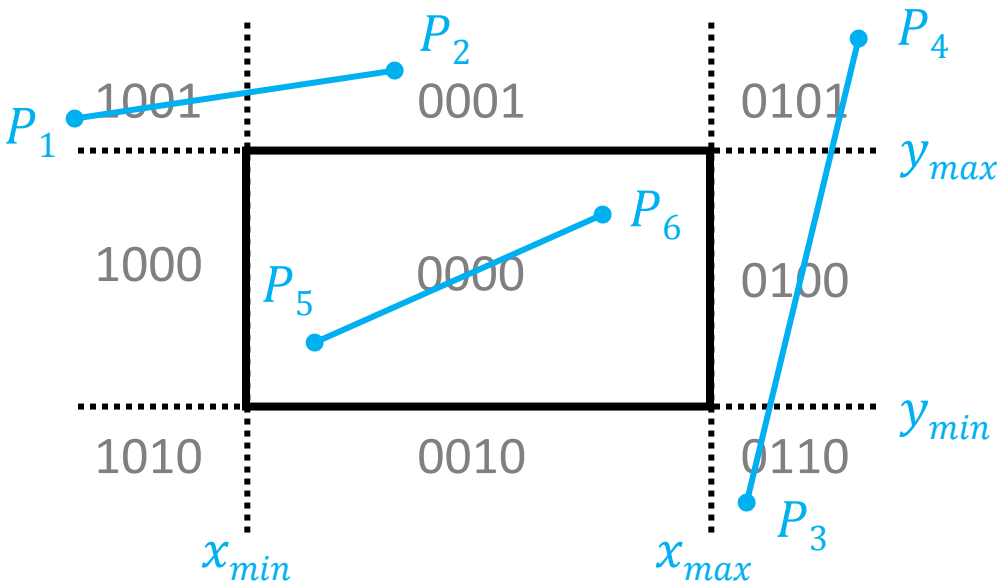


- bestimme Outcodes von P_1 und P_2
- trivial accept,
  wenn beide Punkte innen liegen $\Leftrightarrow$ Outcode(P_1) ∨ Outcode(P_2) == 0
  Ergebnis: die ganze Linie (∨ bitweises OR)
- trivial reject,
  wenn beide Punkte außerhalb sind und Outcode(P_1) ∧ Outcode(P_2) ≠ 0
  Ergebnis: keine Linie (∧ bitweises AND)

Clipping Algorithmus von Cohen-Sutherland

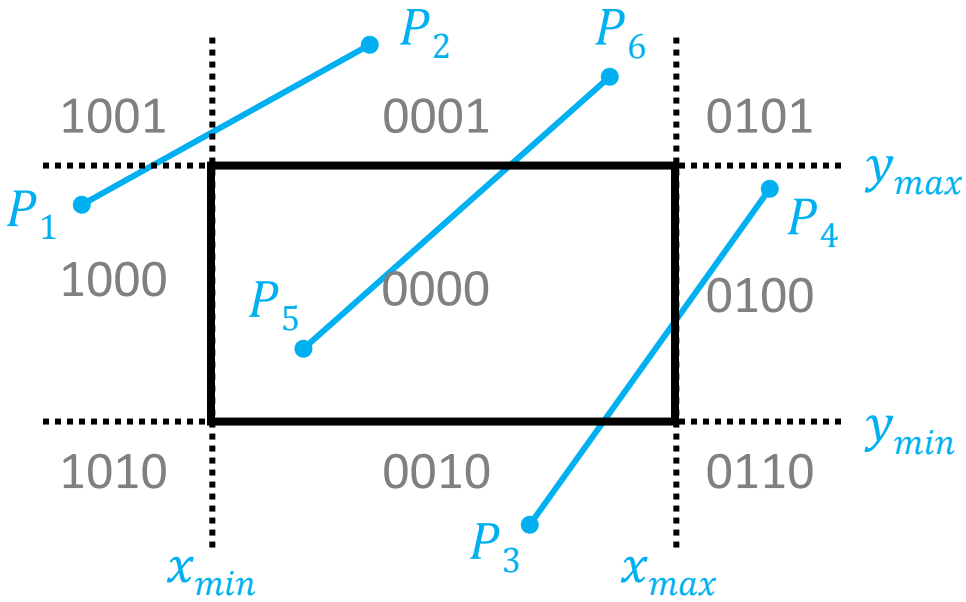
Beispiele

- Outcode (P_1) ∧ Outcode (P_2) = 0001 ≠ 0 ⇒ trivial reject
- Outcode (P_5) ∨ Outcode (P_6) = 0000 == 0 ⇒ trivial accept
- Outcode (P_3) ∧ Outcode (P_4) = 0100 ≠ 0 ⇒ trivial reject



Clipping Algorithmus von Cohen-Sutherland

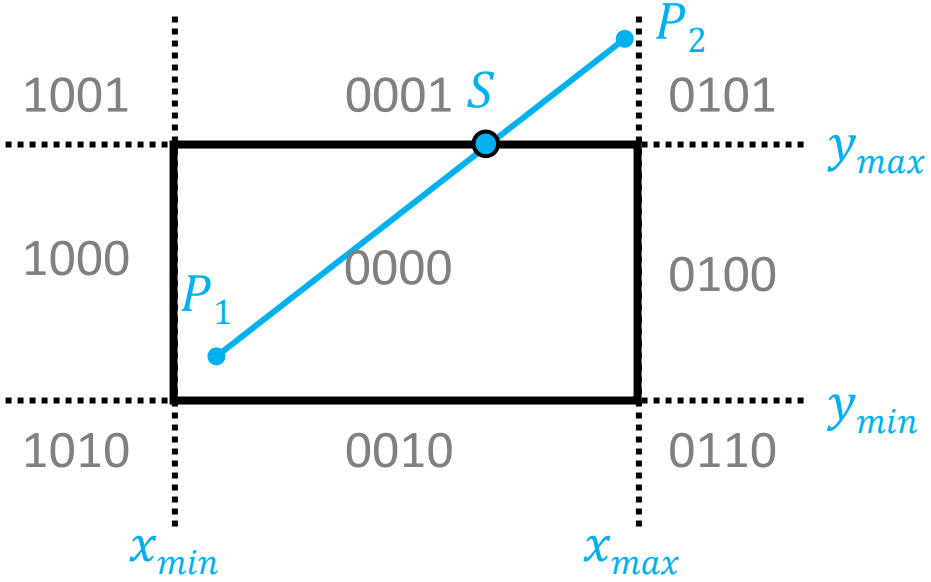
- es gibt zahlreiche Fälle die weder trivial accept noch trivial reject sind
- Achtung: es gibt Linien – z.B. $\overline{P_1 P_2}$ – die vollständig außerhalb des Rechtecks liegen, bei denen aber kein trivial reject möglich ist
- Zum Beispiel:
  - Outcode (P_1) ∧ Outcode (P_2) = 0000 == 0 $\Rightarrow$ kein trivial reject
  - Outcode (P_1) ∨ Outcode (P_2) = 1001 ≠ 0 $\Rightarrow$ kein trivial accept



Clipping Algorithmus von Cohen-Sutherland

Algorithmus zum Clipping von Linien *cont.*

- in den nicht-trivialen Fällen gibt es mindestens eine Kante deren Bit in Outcode(P_1) gesetzt und in Outcode(P_2) nicht gesetzt ist (od. umgekehrt)
  - andernfalls wäre es einer der trivialen Fälle
- die Linie P_1P_2 schneidet diese Kante
  - berechne den Schnittpunkt S
  - wenn P_1 außerhalb bzgl. dieser Kante liegt
    → verfahre weiter mit SP_2, sonst mit P_1S

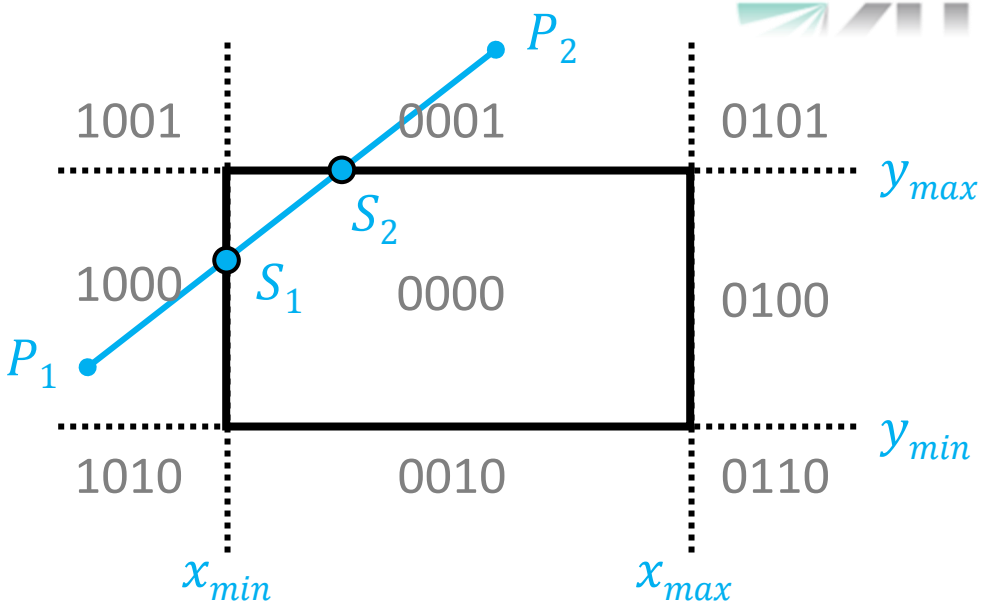


Clipping Algorithmus von Cohen-Sutherland

Beispiel

- Outcode (P_1) = 1000
- Outcode (P_2) = 0001

- Outcode (S_1) = 0000
- Outcode (S_2) = 0000



Outcode (P_1) ∨ Outcode (P_2) ≠ 0	⇒ kein trivial accept	
Outcode (P_1) ∧ Outcode (P_2) == 0	⇒ kein trivial reject	
Outcode (P_1) "left"-Bit gesetzt	⇒ berechne S_1	⇒ weiter mit S_1P_2
Outcode (S_1) ∨ Outcode (P_2) ≠ 0	⇒ kein trivial accept	
Outcode (S_1) ∧ Outcode (P_2) == 0	⇒ kein trivial reject	
Outcode (P_2) "top"-Bit gesetzt	⇒ berechne S_2	⇒ weiter mit S_1S_2
Outcode (S_1) ∨ Outcode (S_2) == 0	⇒ trivial accept (S_1,S_2)	

Clipping optimieren

α-Clipping (nach Cyrus-Beck '78, Liang-Barsky '87)

- Optimierung mittels **W**indow **E**dge **C**oordinates (WEC)
  - $WEC_E(P) < 0 \iff P$ liegt außerhalb bzgl. der Kante E
    - $WEC_{left}(P) = p_x - x_{min}$
    - $WEC_{right}(P) = x_{max} - p_x$
    - $WEC_{bottom}(P) = p_y - y_{min}$
    - $WEC_{top}(P) = y_{max} - p_y$

- Zusammenhang mit den Outcodes:
  das entsprechende Outcode-Bit ist gesetzt, wenn dazugehörige WEC negativ ist, d.h. $\text{Outcode}_E(P)$=signbit$(WEC_E(P))$

α-Clipping

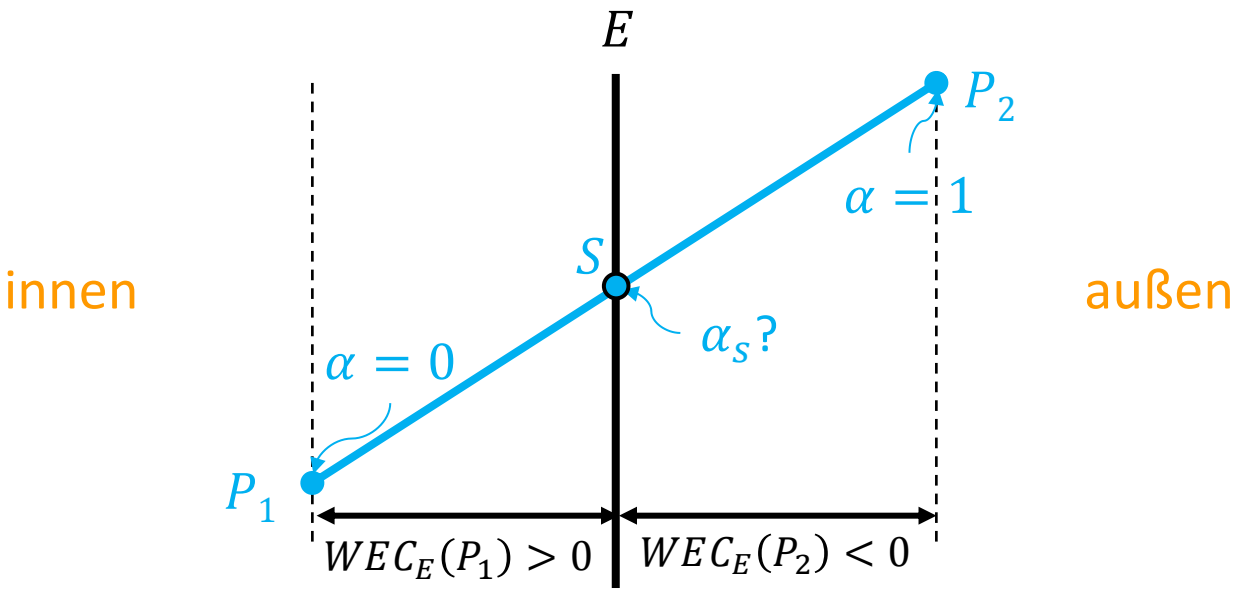
Schnittberechnung mit WECs

- parametrisiere Linie P_1P_2 mit α:

$$P = P_1 + \alpha(P_2 - P_1), \alpha \in [0;1]$$

- α-Wert für den Schnittpunkt mit einer Kante

$$\frac{\overline{P_1S}}{\overline{P_1P_2}} = \alpha_s = \frac{WEC_E(P_1)}{WEC_E(P_1) - WEC_E(P_2)}$$



α-Clipping

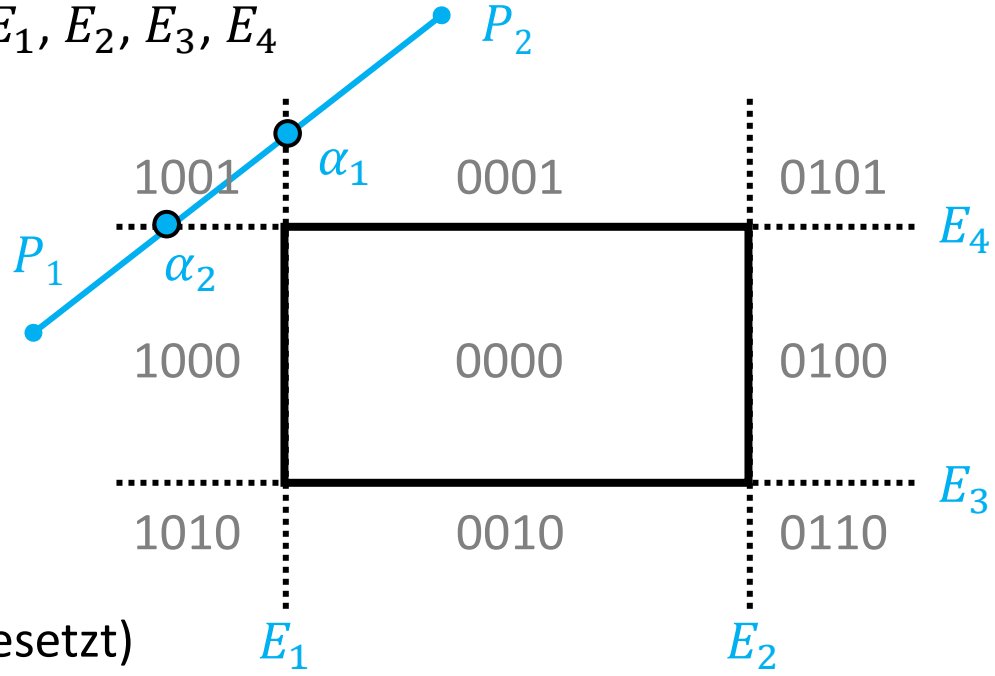
Algorithmus

- berechne WECs von P_1 und P_2
- bestimme Outcodes anhand des Vorzeichens der WECs
- teste auf trivial accept/trivial reject (wie bei Cohen-Sutherland)
- ansonsten verfahre so:
  - setze $\alpha_{min} = 0, \alpha_{max} = 1$
  - für jede Kante E für die das „Outcode$_E$“-Bit für P_1 oder P_2 gesetzt ist
    - berechne $\alpha_s = \frac{WEC_E(P_1)}{WEC_E(P_1) - WEC_E(P_2)}$
    - wenn Outcode$_E(P_1)$
      - dann $\alpha_{min} = \max(\alpha_{min}, \alpha_s)$
      - sonst $\alpha_{max} = \min(\alpha_{max}, \alpha_s)$
    - wenn $\alpha_{min} > \alpha_{max}$, dann liegt die Linie außerhalb
      - sonst gebe das folgende Liniensegment zurück: $(P_1 + \alpha_{min}(P_2 - P_1), P_1 + \alpha_{max}(P_2 - P_1))$

α-Clipping

Beispiel

- berechne WECs für P_1 und P_2 bzgl. E_1, E_2, E_3, E_4
- bestimme Outcodes
  - Outcode$(P1) = 1000$
  - Outcode$(P2) = 0001$
- kein trivial accept/trivial reject
- Clipping

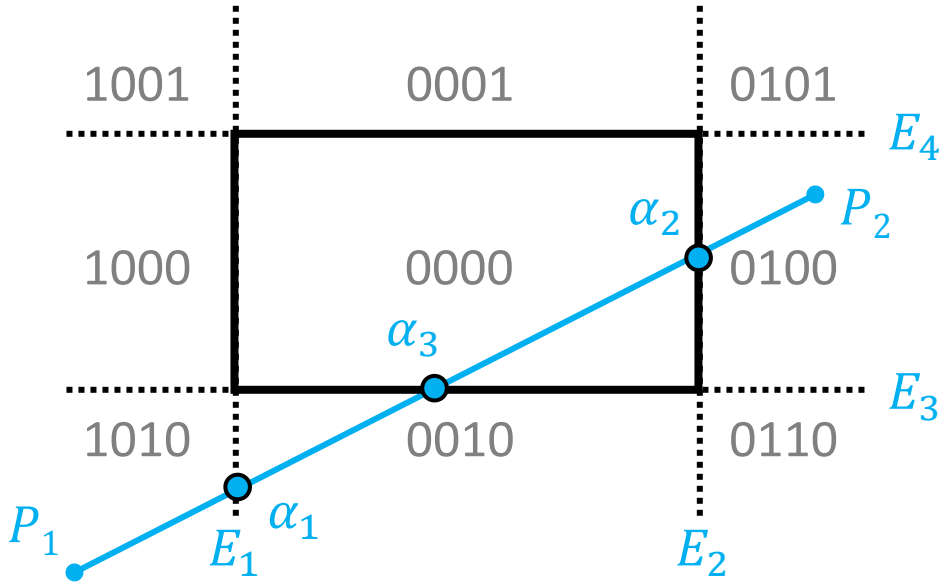


  - setze $\alpha_{min} = 0, \alpha_{max} = 1$
  - für Kante E_1 (Outcode$_{E1}(P_1)$ ist gesetzt)
    - berechne α_1 bzgl. E_1
    - setze $\alpha_{min} = \alpha_1$, weil Outcode$_{E1}(P_1)$ gesetzt ist
  - für Kante E_4 (Outcode$_{E4}(P_2)$ ist gesetzt)
    - berechne α_2 bzgl. E_4
    - setze $\alpha_{max} = \alpha_2$, weil Outcode$_{E4}(P_2)$ gesetzt ist
  - $\alpha_{max} < \alpha_{min} \Rightarrow$ keine Linie

α-Clipping

Beispiel

- Kante $E_1 \rightarrow \alpha_{min} = \alpha_1$
- Kante $E_2 \rightarrow \alpha_{max} = \alpha_2$
- Kante $E_3 \rightarrow \alpha_{min} = \alpha_3$
- gebe Liniensegment zurück, denn $\alpha_{min} < \alpha_{max}$



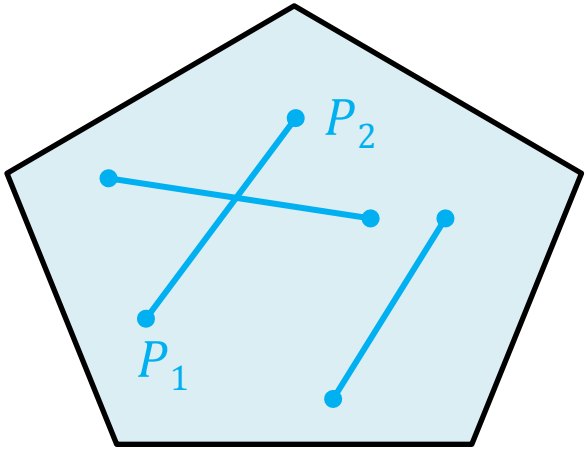
α-Clipping

Pseudocode (hier: Axis-Aligned Clipping Region)

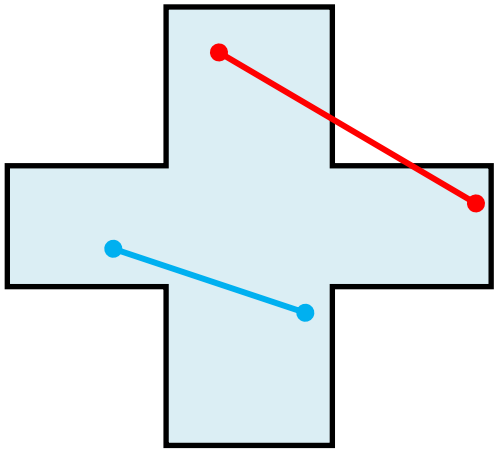
```
void drawClipLine(float2 p1, float2 p2, float xmin,float ymin,float xmax,float ymax) {
  // bestimme Outcodes
  ...

  if ( outcode( p1 ) | outcode( p2 ) == 0 )          // trivial accept
  {  drawLine( p1, p2 ); return; }

  if ( outcode( p1 ) & outcode( p2 ) != 0 ) return;  // trivial reject

  a_min = 0; a_max = 1;

  if ( Bit "links" in outcode( p1 ) gesetzt ) {      // Clipping von p1 an linker Kante
    a = WEC_left( p1 ) / ( WEC_left( p1 ) - WEC_left( p2 ) );
    a_min = max( a_min, a );
  } else
  if ( Bit "links" in outcode( p2 ) gesetzt ) {      // Clipping von p2 an linker Kante
    a = WEC_left( p1 ) / ( WEC_left( p1 ) - WEC_left( p2 ) );
    a_max = min( a_max, a );
  }

  if ( Bit "rechts" in outcode( p1 ) gesetzt ) ...   // weiter mit allen anderen Kanten

  If ( a_min < a_max ) {
    pt1 = p1 + a_min * (p2 - p1);  pt2 = p1 + a_max * (p2 - p1);
    drawLine( pt1, pt2 );
} }
```

Konvexe Clipping Regionen

- Definition
  - eine Menge von Punkten $\mathrm{M} \subseteq \mathbb{R}^2$ ist konvex, wenn für alle $P_1, P_2 \in \mathrm{M}$ alle Punkte auf der Linie $\overline{P_1 P_2}$ ebenfalls in M liegen



konvex



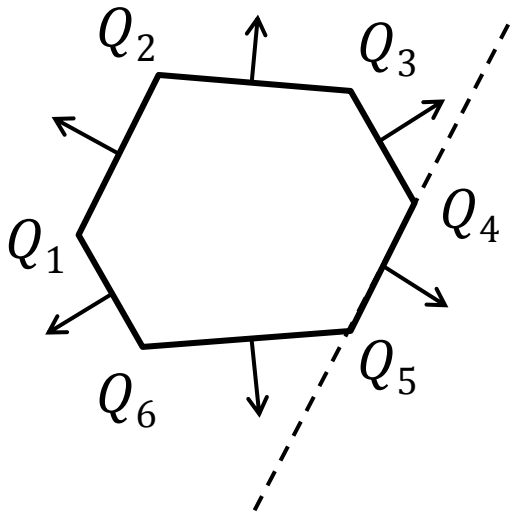
nichtkonvex

- ein Polygon ist konvex, wenn die Menge der Punkte im Innern konvex ist
- Achtung: nichtkonvex ist nicht gleichbedeutend mit konkav (manche Definitionen bezeichnen eine Teilmenge als konkav, wenn ihr Komplement konvex ist)

Konvexe Clipping Regionen

- Clipping einer Linie gegen konvexe Polygone
  - Resultat ist ein Liniensegment oder eine leere Menge von Punkten
  - bei nichtkonvexen Polygonen können mehrere Segmente entstehen
- bei konvexen Polygonen kann α-Clipping direkt übertragen werden
- für ein Polygon Q_1,…, Q_n ergibt sich dann
  - n Kanten → n WECs
  - $WEC_i(P)$: gerichteter Abstand von P zur Linie Q_iQ_{i+1} (negativ, wenn P außerhalb des Polygons liegt)
  - Konvention: $Q_{n+1} = Q_1$

- Clipping in 3D
  - analog, WEC sind die Abstände zu Clipping-Ebenen



Nichtkonvexe Clipping Regionen

- unterteile in konvexe Teilregionen
- anschließend Clipping gegen jede Teilregion…
- …und zusammensetzen der Liniensegmente