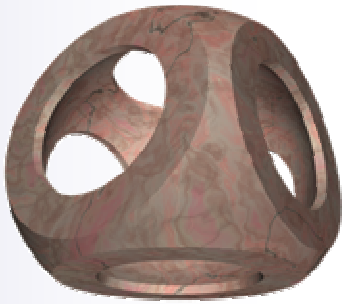
Bildbasierte CSG



Techniken zur Darstellung solider Objekte in Echtzeit

Vortrag im Seminar Computergrafik am Hasso-Plattner-Institut
gehalten von Stephan Brumme am 12.Juni 2002

Gliederung

I. Einführung

II. Übersicht CSG-Rendering-Techniken

III. Goldfeather-Algorithmus

IV. Layered Goldfeather

V. Fazit

VI. Quellen

I. Einführung

- Schaffung eines 3D-Modells:

  a) Verwendung existierender Daten

  - 3D-Scanner,
  - Nutzung mathematisch-physikalischer Gesetze
  - …

    → gut automatisierbar

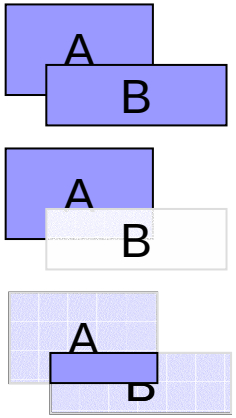
  b) interaktive Gestaltung

  - CAD-Software
  - …

    → eher schlechte Unterstützung durch Computer

I. Einführung

- Anforderungen an interaktive Gestaltung:
  - Möglichkeit des Experimentierens
  - intuitive Objektdeformationen
    - Zusammenfügen
    - Herausschneiden
    - gemeinsame Anteile von Objekten



I. Einführung

- Beispiel:

  Astro-Schale
  = (Schale – Sterne)
  – Monde

(C) David Kurlander, Columbia University

I. Einführung

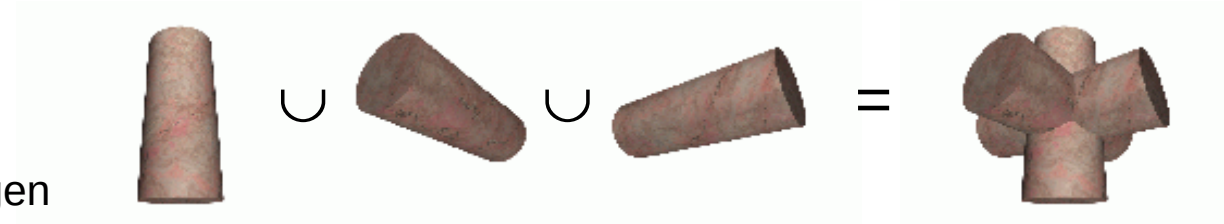
- Begriff CSG:
  - Abkürzung für Constructive Solid Geometry
  - Aufbau komplexer solider Objekte aus einfachen Primitiven
  - Verwendung boolescher Operationen:
    - Summe bzw. Vereinigung ($\cup$, ODER)
    - Produkt bzw. Durchschnitt ($\cap$, UND)
    - Differenz (–, MINUS)

I. Einführung

- Beispiele boolescher Operationen

Summe

→ Zusammenfügen



Produkt

→ gemeinsames Volumen

Differenz

→ Herausschneiden

I. Einführung

- Begriff solides Objekt:
  - geometrisches Objekt mit geschlossener Oberfläche

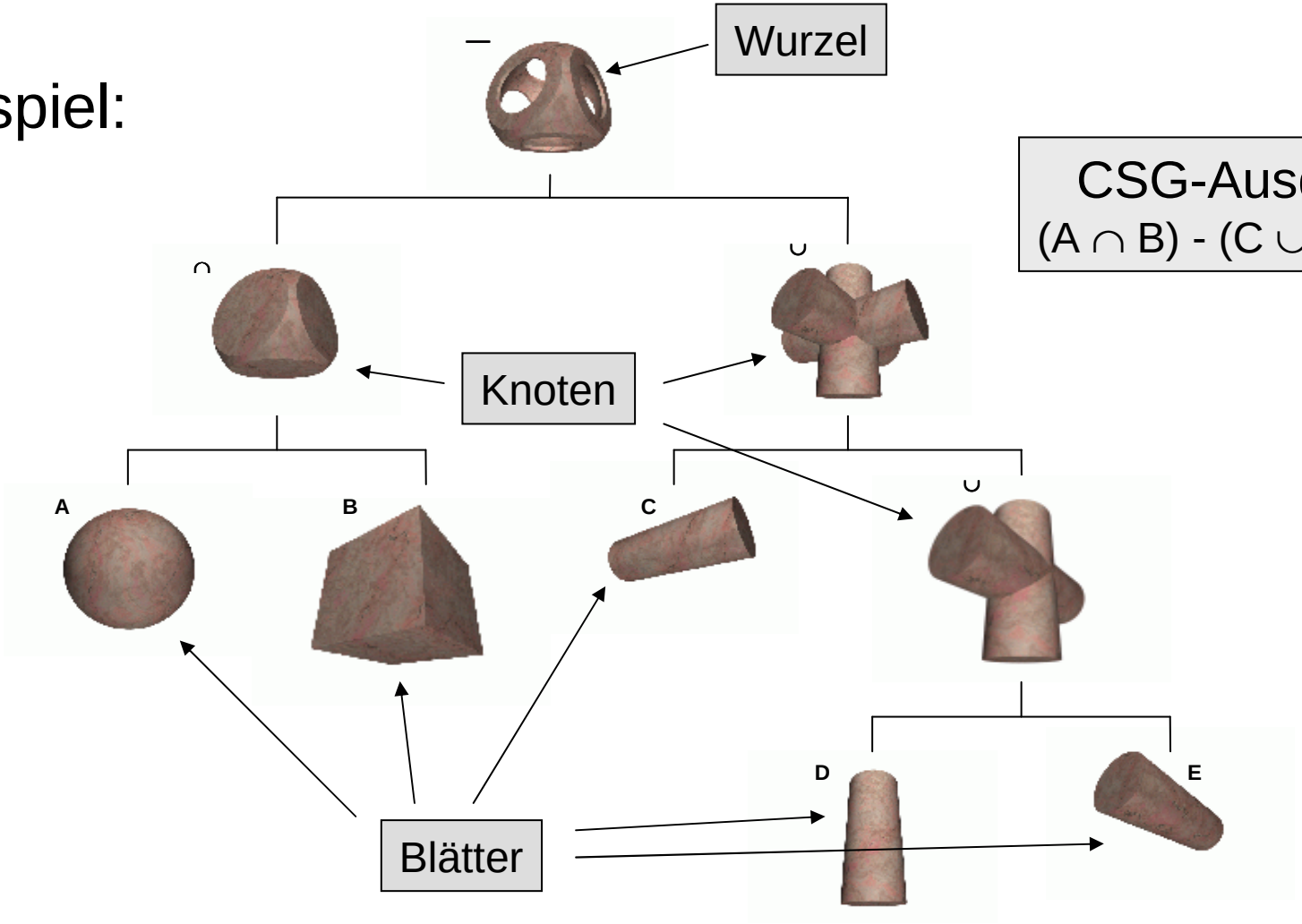
- Begriff Primitiv:
  - von der Grafikbibliothek bereitgestelltes solides Objekt
  - kann ohne weiteres Zutun korrekt dargestellt werden

I. Einführung

- hierarchischer CSG-Baum:
  - binärer Baum
  - Wurzel ist das zu erzeugende CSG-Objekt
  - jedes Blatt ist ein Primitiv
  - jeder Knoten enthält eine boolesche Operation
    - evtl. noch affine Transformationen

I. Einführung

- Beispiel:

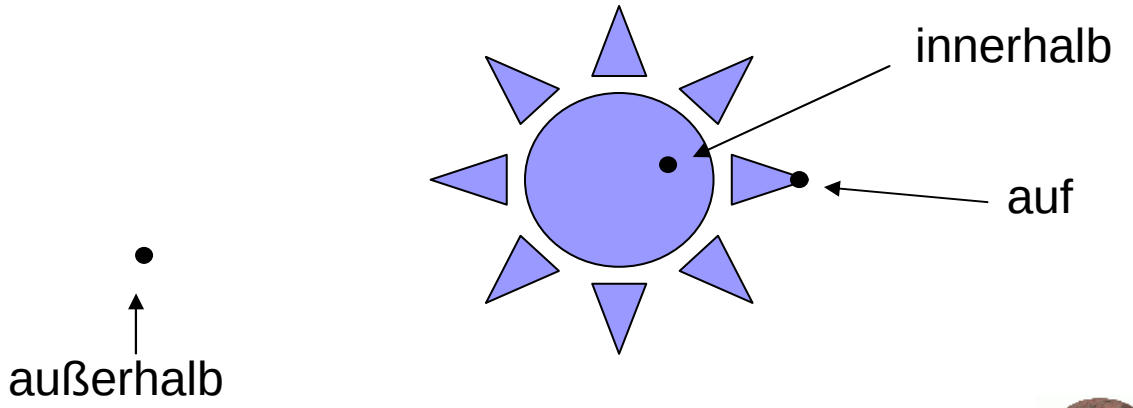


CSG-Ausdruck:
$(A \cap B) - (C \cup (D \cup E))$

I. Einführung

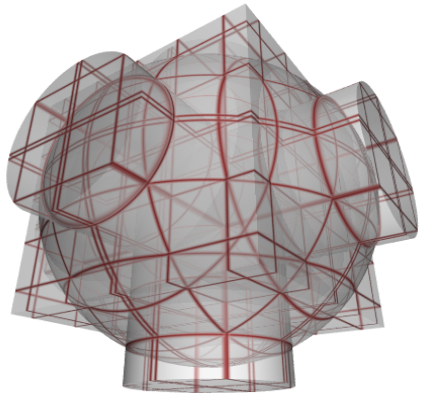
- Klassifikation eines Punktes P:
  - außerhalb, wenn $P \notin$ CSG-Objekt
  - innerhalb, wenn $P \in$ CSG-Objekt
    - auf, wenn zwar P als innerhalb gilt, aber in einer beliebig kleinen Umgebung Punkte als außerhalb eingestuft werden

      → $P \in$ Oberfläche

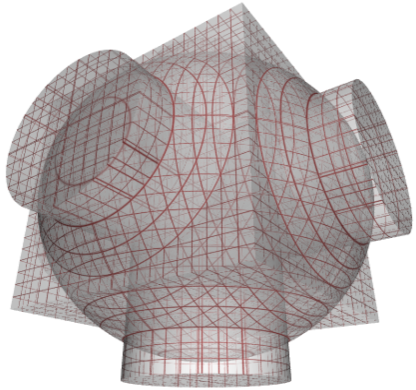


II. Übersicht CSG-Rendering-Techniken

- B-rep (boundary representation)
  - Idee:
    - Tessellation aller Primitive



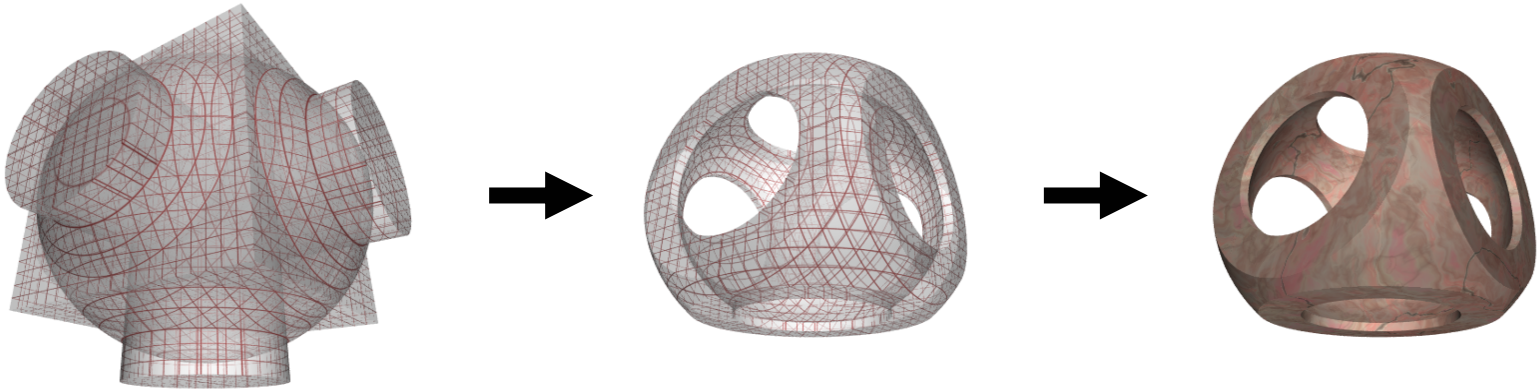
    - bei schneidenden Flächen dort stärker tessellieren



II. Übersicht CSG-Rendering-Techniken

- B-rep, Fortsetzung
  - Entfernung aller Flächen, die nach CSG-Auswertung als „innerhalb“ oder „außerhalb“ eingestuft sind

    → nur „auf“ bleibt übrig, d.h. die Oberfläche



  - Darstellung mit beliebigem Rendering-Verfahren

II. Übersicht CSG-Rendering-Techniken

- B-rep, Fortsetzung II
  - Eigenschaften:
    - objektbasiert
    - Bewertung:

      + CSG-Auswertung vor Rendering

        → unabhängig vom eigentlichen Rendering

        → unabhängig vom Betrachter

      + Neuberechnung nur dann erforderlich, wenn sich der CSG-Baum ändert

      - Tessellation oft lediglich approximativ möglich

      - keine Hardwareunterstützung

II. Übersicht CSG-Rendering-Techniken

- z-Buffer
  - Idee:
    - Nutzung von Hardware
      - Vereinfachung des CSG-Baums
    - CSG-Operationen während des Renderings durchführen
      - Multipass-Technik
      - Umsetzung boolescher Operationen durch z-Buffer-Test

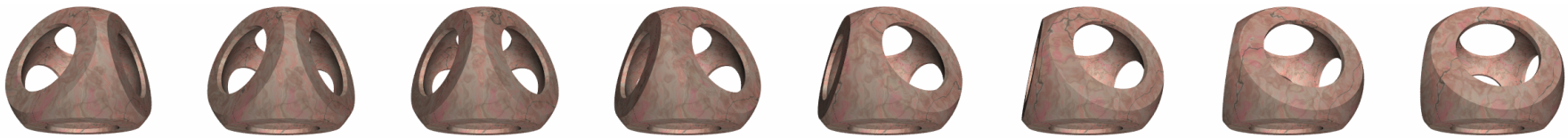
II. Übersicht CSG-Rendering-Techniken

- z-Buffer, Fortsetzung
  - Eigenschaften von z-Buffer-Techniken:
    - bildbasiert
    - Bewertung:

      + pixelpräzise

      + durch Hardware unterstützt

      - CSG-Neuberechnung für jedes einzelne Bild



II. Übersicht CSG-Rendering-Techniken

- Andere Verfahren:
  - Ray Casting (bildbasiert)
    - nicht echtzeitfähig
  - Spatial Enumeration (objektbasiert)
    - hoher Speicherbedarf
  - Hybride Verfahren
  - ...

→ im weiteren Vortrag werden nur z-Buffer-basierte Algorithmen präsentiert

III. Goldfeather-Algorithmus

- 1989 entwickelt von Goldfeather, Molnar, Turk und Fuchs [1]

- Ziel:
  - Nutzung von Spezialhardware
- Problem:
  - CSG nicht direkt mit Hardware evaluierbar

  → eine komplexe und schwierige Aufgabe auf mehrere kleine, aber lösbare abbilden

III. Goldfeather-Algorithmus

- Idee:
  - Vereinfachung des CSG-Baums durch Umformung
    - Normalisierung
  - Clipping der Primitive auf Fragmentebene
    - Klassifikation „innerhalb“ / „außerhalb“
    - Tiefentest
  - zunächst Beschränkung auf konvexe Primitive

III. Goldfeather-Algorithmus

- Normalisierung:
  - Summe von Teilbäumen mit z-less-Test
  - Produkt von Teilbäumen aber nicht-trivial
  - daher Überführung in eine Summe von Produkten
  - nur Produkte der Form Teilbaum$\cap$Primitiv
  - es gilt auch: $\boxed{A - B = A \cap \overline{B}}$
    - lies: Produkt von A mit dem Komplement von B
    - d.h. Differenz ist auch ein Produkt !

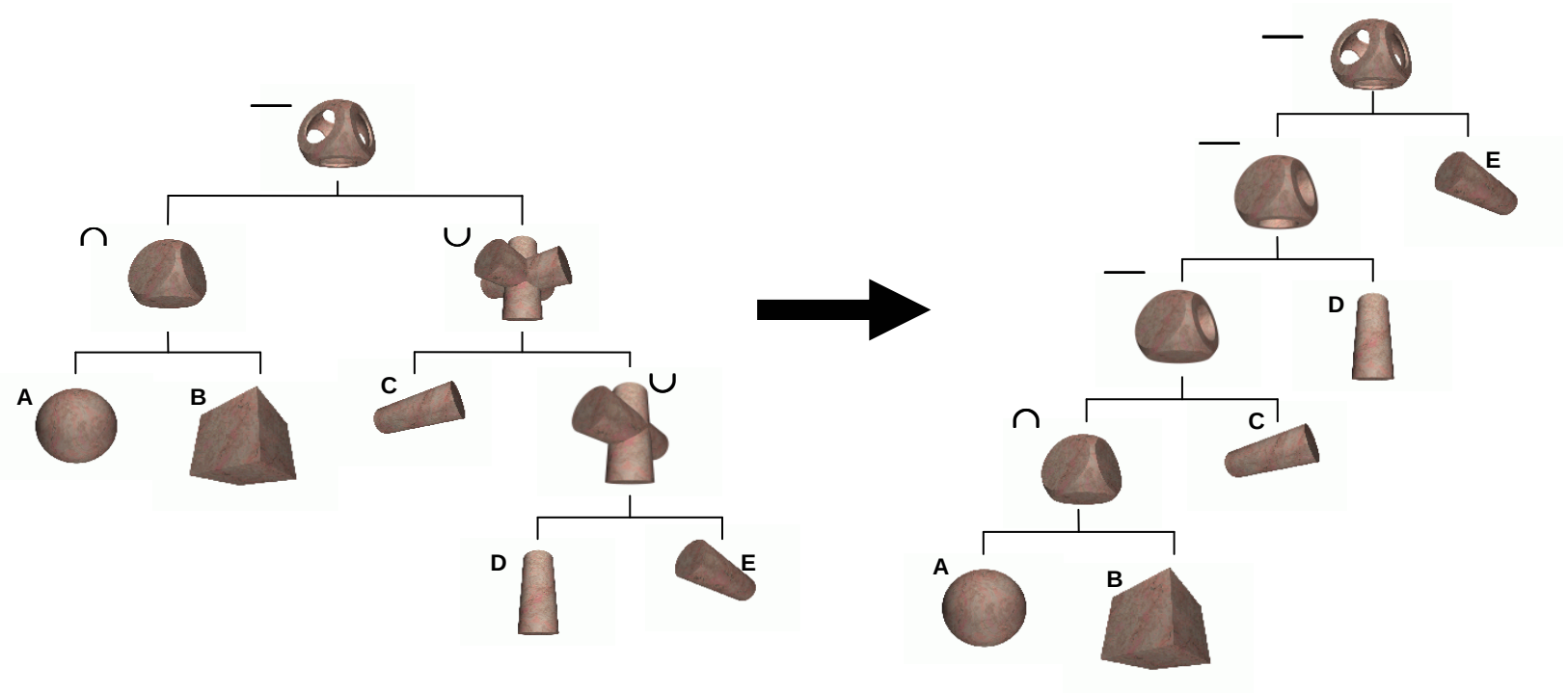
III. Goldfeather-Algorithmus

- Normalisierungs-Regeln:

1. $X - (Y \cup Z) = (X - Y) - Z$
2. $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z)$
3. $X - (Y \cap Z) = (X - Y) \cup (X - Z)$
4. $X \cap (Y \cap Z) = (X \cap Y) \cap Z$
5. $X - (Y - Z) = (X - Y) \cup (X \cap Z)$
6. $X \cap (Y - Z) = (X \cap Y) - Z$
7. $(X - Y) \cap Z = (X \cap Z) - Y$
8. $(X \cup Y) - Z = (X - Z) \cup (Y - Z)$
9. $(X \cup Y) \cap Z = (X \cap Z) \cup (Y \cap Z)$

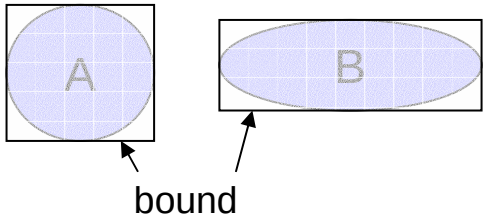
III. Goldfeather-Algorithmus

- Beispiel:
  - zweimal Regel 1 benutzt



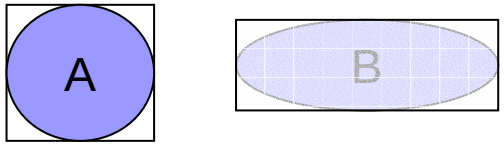
III. Goldfeather-Algorithmus

- CSG-Baum ausdünnen (pruning):
  - Normalisierung vergrößert u.U. den Baum
  - Bounding boxes bilden
    - wenn bound(A) nicht bound(B) durchdringt:
      - □ $A \cap B = \emptyset$



→ A und B entfernen

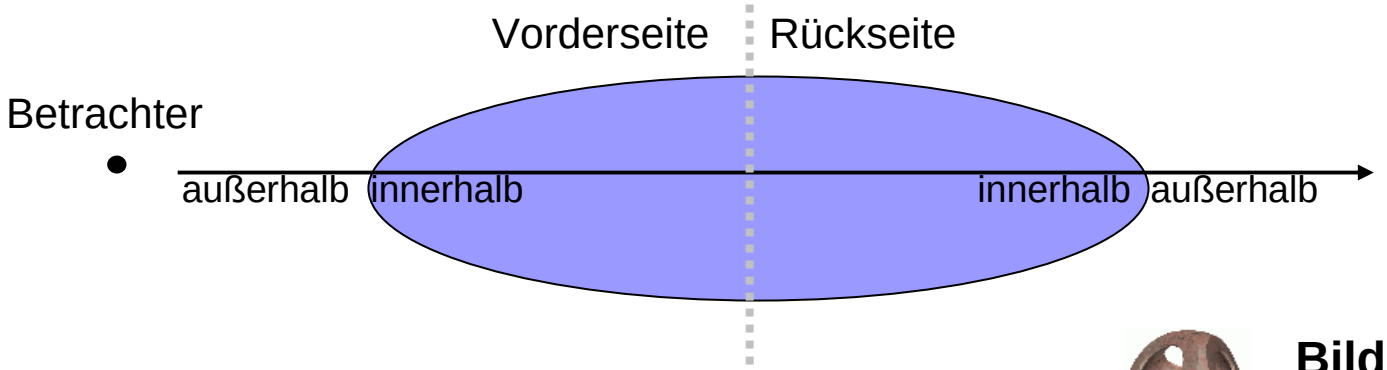
      - □ $A - B = A$



→ B entfernen

III. Goldfeather-Algorithmus

- Klassifikation „innerhalb“ / „außerhalb“
  - jeder Punkt ist immer entweder innerhalb oder außerhalb bzgl. eines Objekts
  - Betrachter ist stets „außerhalb“ (Voraussetzung)
  - für jedes Objekt:
    - Sichtstrahl ist nach Schnitt mit Vorderseite „innerhalb“
    - Sichtstrahl ist nach Schnitt mit Rückseite „außerhalb“

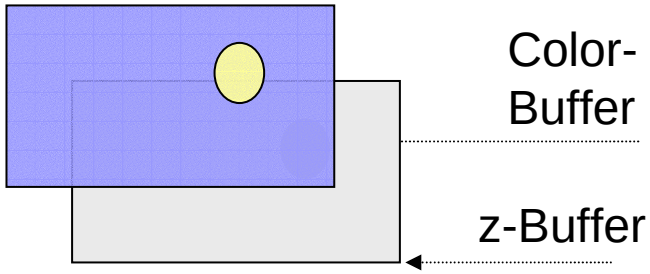


III. Goldfeather-Algorithmus

- Klassifikation mit Paritätstest P(Y)
  - Clipping von Fragmenten gegen ein Primitiv Y
    - bereits gezeichnete Fragmente setzten z-Buffer
      → nun Test, ob sie innerhalb von Y liegen
    - Vorder- und Rückseite von Y zeichnen
      - aber z- und Color-Buffer schreibgeschützt
      - nur Test auf z-less:
        - wenn erfolgreich, dann das Paritätsbit invertieren
    - bei gesetzter Parität: Fragment liegt innerhalb von Y

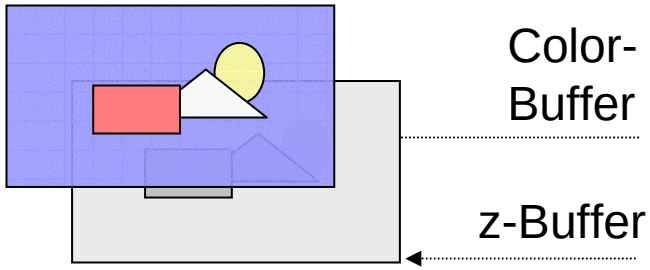
III. Goldfeather-Algorithmus

- Renderingfunktion D für konvexe Primitive
  - D(X)
    - z-less als Tiefenfunktion
    - Culling aktivieren
    - Primitiv X zeichnen



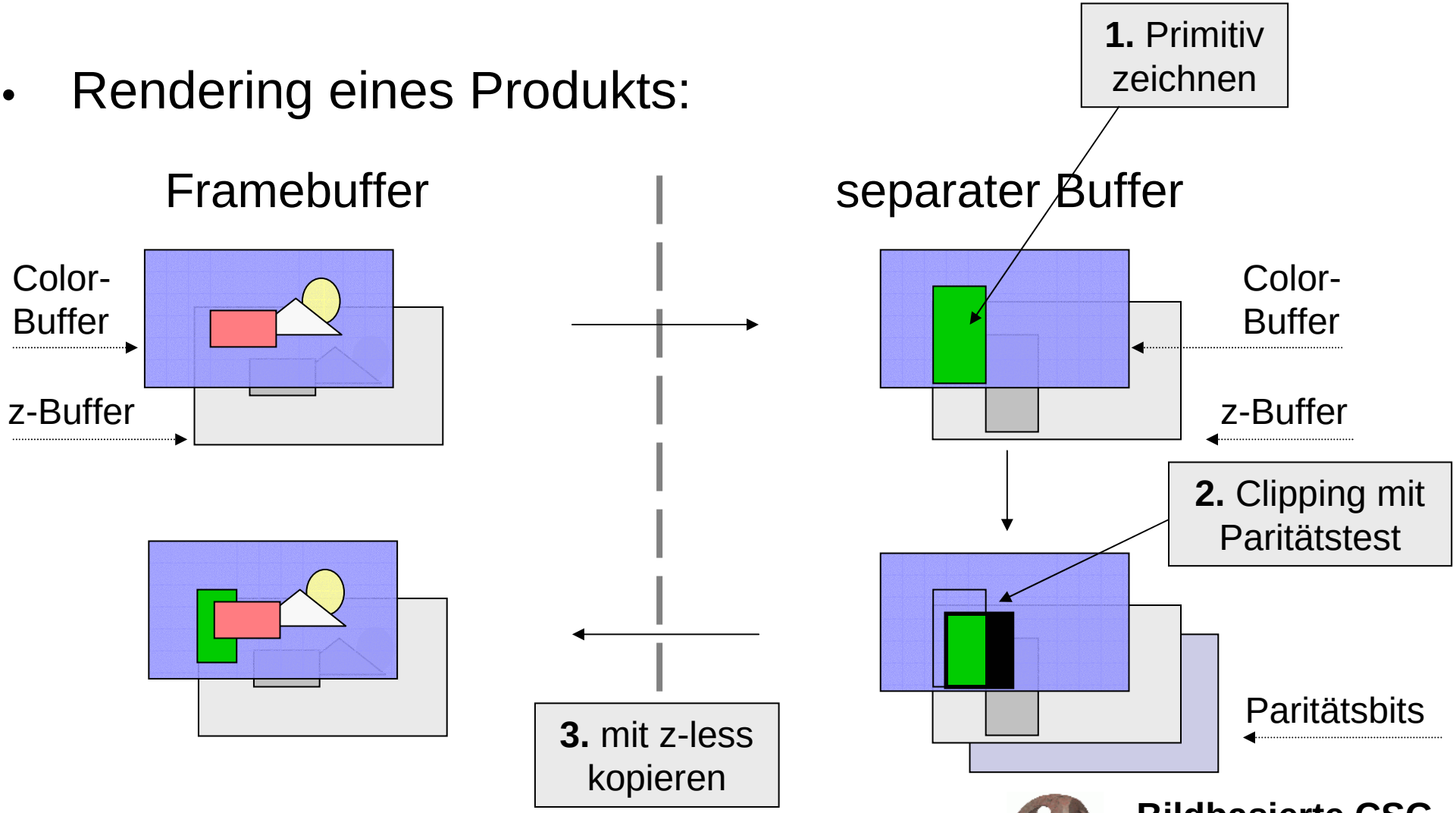
  - Summe D(X ∪ Y)
    - z-less als Tiefenfunktion
    - D(X) und D(Y) nacheinander

      → problemlos erweiterbar auf mehrere Primitive, d.h. D(X ∪ Y ∪ Z ∪ ...)



III. Goldfeather-Algorithmus

- Rendering eines Produkts:

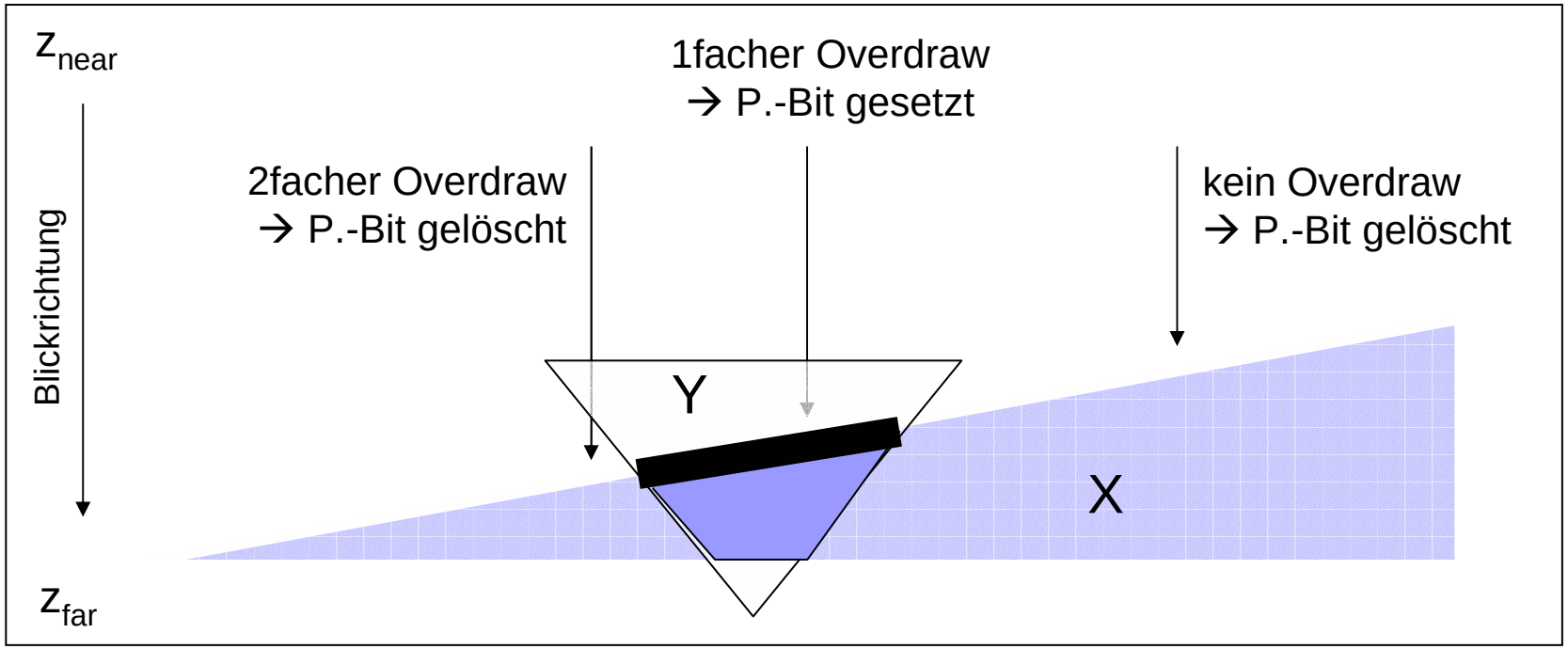


III. Goldfeather-Algorithmus

- Renderingfunktion D, Fortsetzung
  - Produkt $D(X \cap Y)$
    - separaten, leeren z- und Color-Buffer bereitstellen
    - dort D(X) zeichnen, anschließend Paritätstest P(Y)
    - Fragmente aus separaten Buffers löschen:
      - Y ist kein Komplement: löschen, wenn Paritätsbit gesetzt ist
      - Y ist Komplement: löschen, wenn Paritätsbit gelöscht ist
    - übriggebliebene Fragmente mit z-less in Framebuffer kopieren

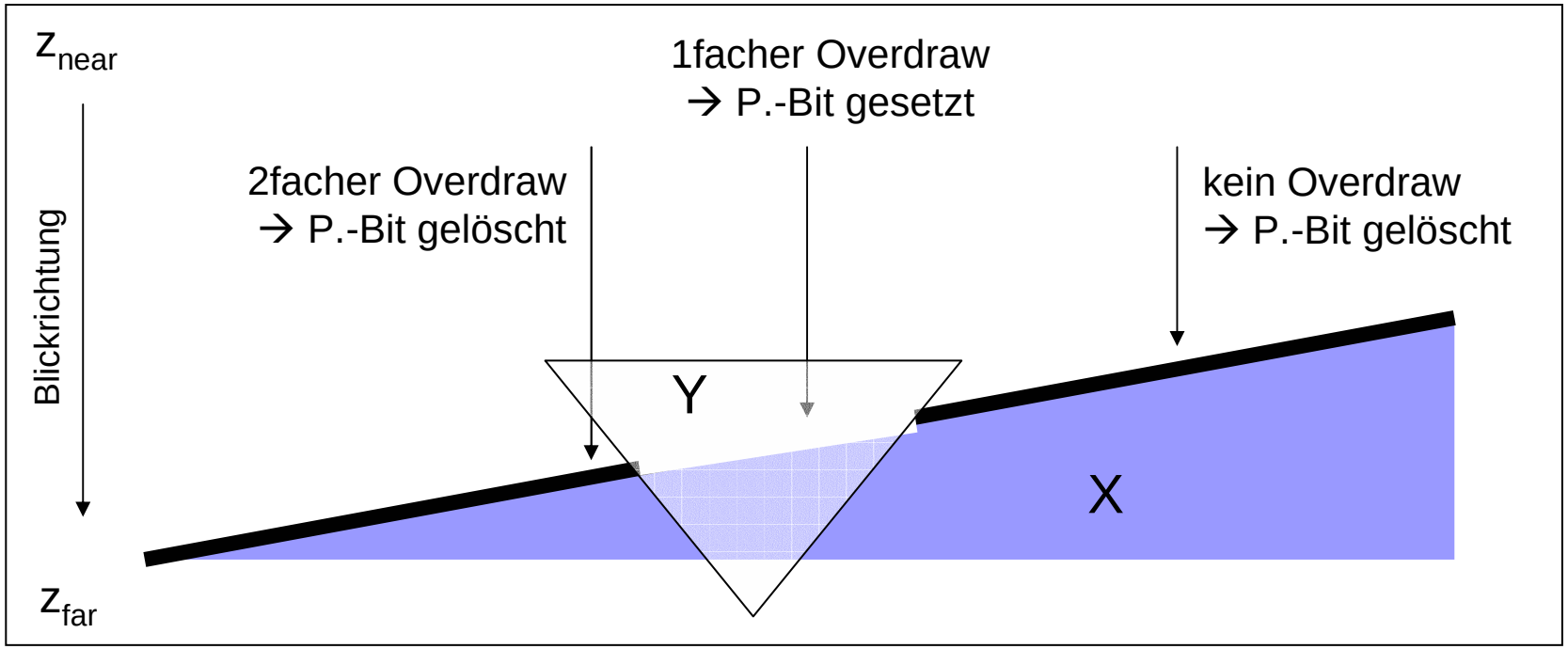
III. Goldfeather-Algorithmus

- Beispiel Renderingfunktion mit Paritätstest
  - Produkt $X \cap Y$



III. Goldfeather-Algorithmus

- Beispiel Renderingfunktion mit Paritätstest
  - Differenz $X - Y \rightarrow X \cap \overline{Y}$

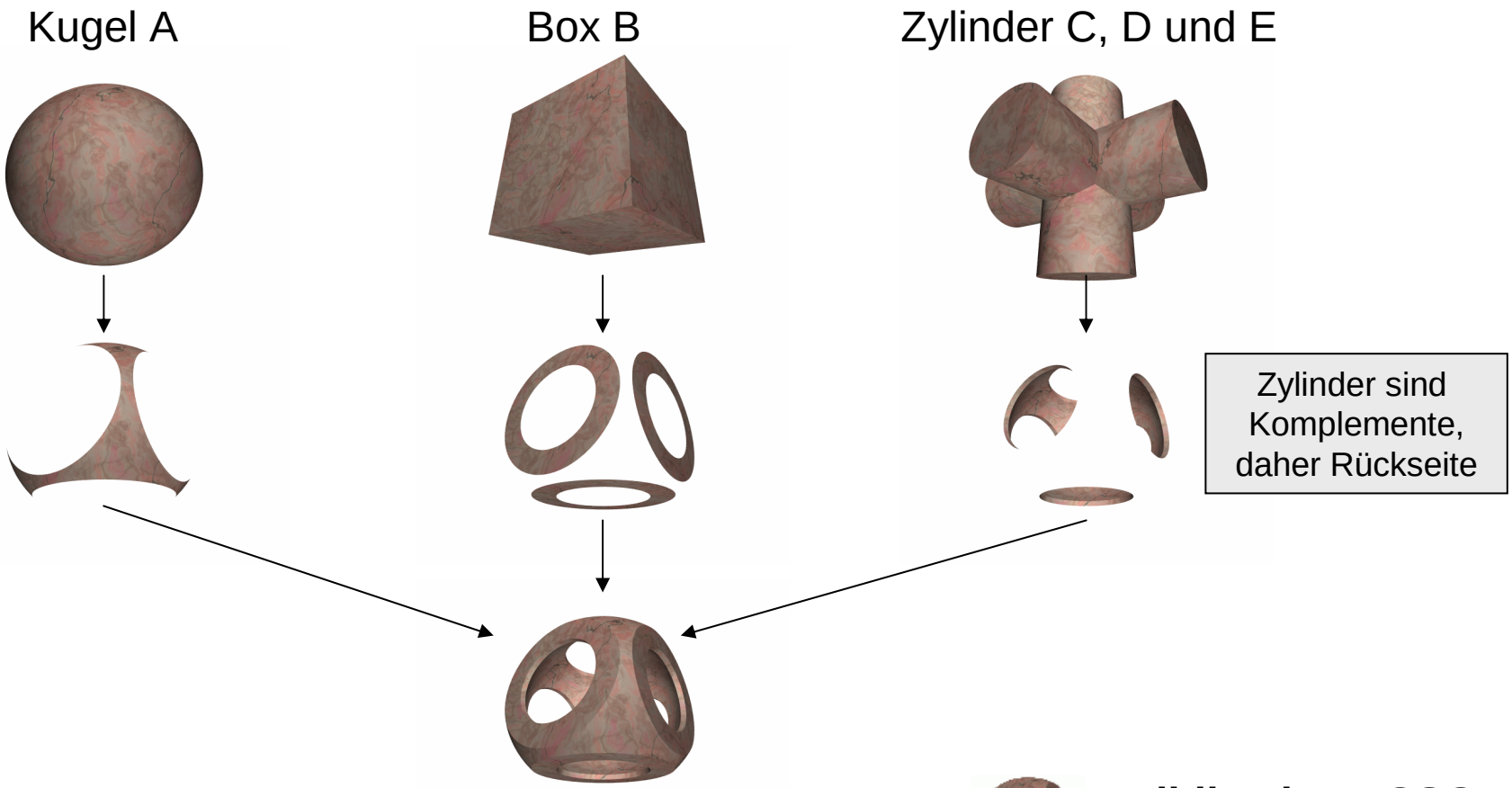


III. Goldfeather-Algorithmus

- Renderingfunktion D, Fortsetzung II
  - Produkt $D(X \cap Y \cap Z \cap \ldots)$
    - für jedes Primitiv ist einzeln durchzuführen:
      - D(Primitiv) in separaten Buffers zeichnen
      - gegen alle anderen mit Paritätstest clippen
      - ggf. Fragmente löschen
      - übriggebliebene Fragmente mit z-less in Frame-buffer kopieren

III. Goldfeather-Algorithmus

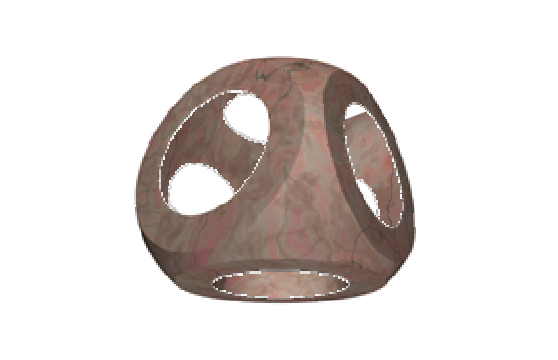
- Beispiel sichtbarer Fragmente



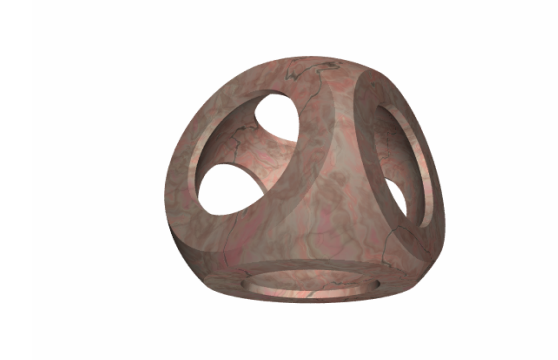
III. Goldfeather-Algorithmus

- Culling:
  - jeweils nur eine Seite eines Primitivs ist sichtbar
  - Back-Face Culling, wenn X kein Komplement
  - sonst Front-Face Culling
  - Beispiel: Inneres sind Komplemente (Zylinder C, D, E):

Rückseite nicht gezeichnet
→ falsch



Back-Face Culling
→ falsch
(oben rechts weiß,
Lichteinfall passt nicht)

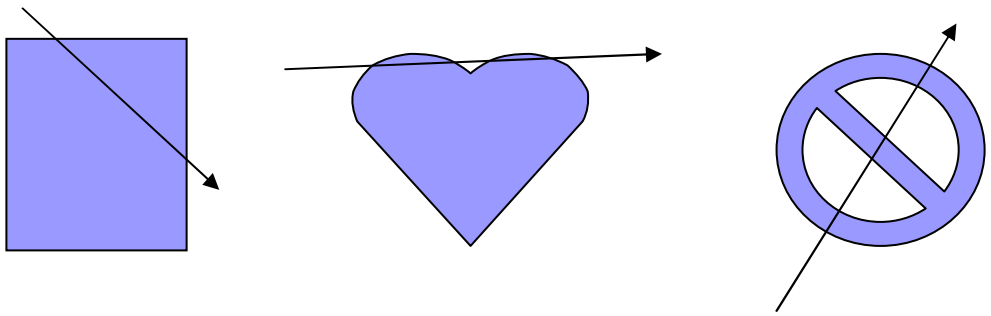


Front-Face Culling
→ korrekt

III. Goldfeather-Algorithmus

- konkave Primitive
  - mehrere Vorder- und Rückseiten pro Primitiv und Fragment möglich
  - sei *k* maximal denkbare Anzahl an Paaren Vorder-/Rückseite

    → Primitiv heißt *k*-konvex



1-konvex 2-konvex 3-konvex

III. Goldfeather-Algorithmus

- konkave Primitive, Fortsetzung
  - Primitive und Summen mit bisherigen Techniken weiterhin darstellbar
  - jedes konkave Primitiv ist als Summe von konvexen Teilprimitiven darstellbar
  - Umformung von Produkten in Teilprodukte
    - jedes Teilprodukt enthält ein Paar Vorder-/Rückseite des *n*-konkaven Primitivs
    - alle Teilprodukte werden summiert
    - Beispiel: A sei 2-konvex, B nur 1-konvex
      - $A \cap B = (A_1 \cap B) \cup (A_2 \cap B)$

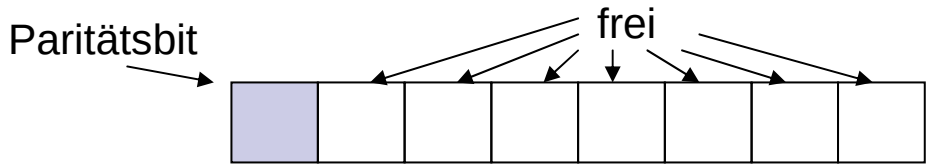
III. Goldfeather-Algorithmus

- Original-Technik verlangt:
  - 2 z-Buffer,
  - 2 Color-Buffer und
  - 1 Paritätsbit-Buffer

- Adaption von Wiegand auf OpenGL (1996) [2]
  - nur 1 z-Buffer, 1 Color-Buffer und 1 Stencil-Buffer vorhanden
    → Paritätsbit im Stencil-Buffer verwalten

III. Goldfeather-Algorithmus

- Wiegand, Fortsetzung
  - Stencil-Buffer ist oft mehrere Bits breit, z.B. 8:
    - 1 Paritätsbit, 7 noch frei verfügbar



  - geclipptes Primitiv nicht sofort in Framebuffer kopieren → Gruppierung
  - zuerst alle freien Bits für Sichtbarkeit ausnutzen
    - z.B. *n*-tes Bit für das *n*-te Primitiv setzen, wenn Fragment dort nicht geclippt wird

III. Goldfeather-Algorithmus

- Wiegand, Fortsetzung II
  - alle Primitive erneut im Framebuffer zeichnen
    - nur dort, wo das zum Primitiv passende Bit im Stencil-Buffer gesetzt ist
    - Culling und z-less verwenden
  - Problem: immer noch 1 z-Buffer für Clipping, 1 z-Buffer für Framebuffer
  - Bewertung:

    + nur 1 Color-, 2 z- und 1 Stencil-Buffer notwendig

    - zweifaches Zeichnen aller Primitive

    - zweiten z-Buffer durch Kopieren emulieren

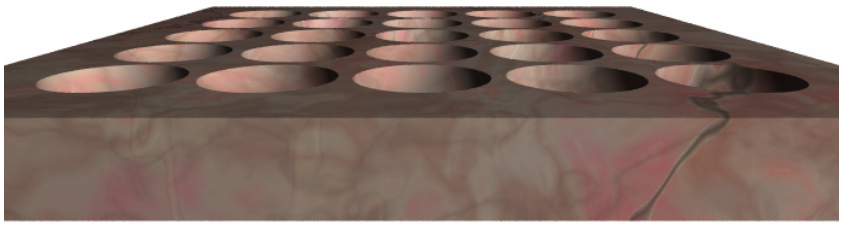
IV. Layered Goldfeather

- Wiegands Verfahren optimiert durch Stewart, Leach und John (1998) [3]
  - Beobachtung:
    - Clipping ist nicht optimal: Vergleich von jedem Primitiv mit jedem anderen des Produkts
      → Komplexität $O(n^2)$
  - aber: viele Objekte überlappen nicht einander
    - maximale Anzahl Primitive pro Fragment ist die Tiefenkomplexität k
    - abhängig von der Blickrichtung ist k oft sehr klein

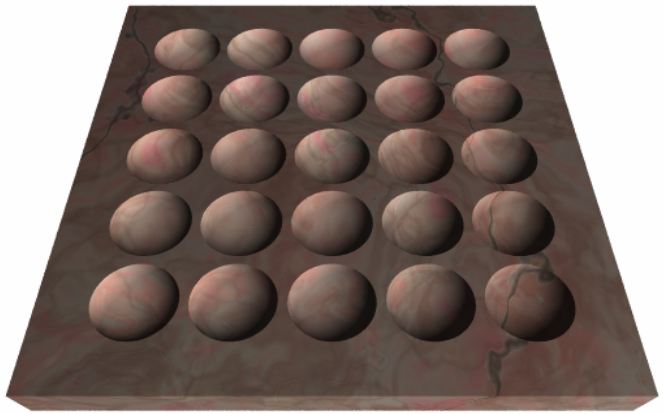
IV. Layered Goldfeather

- Beispiel:
  - 1 Box – 25 Kugeln
  - trotzdem lediglich geringe Tiefenkomplexität

k=6



k=2

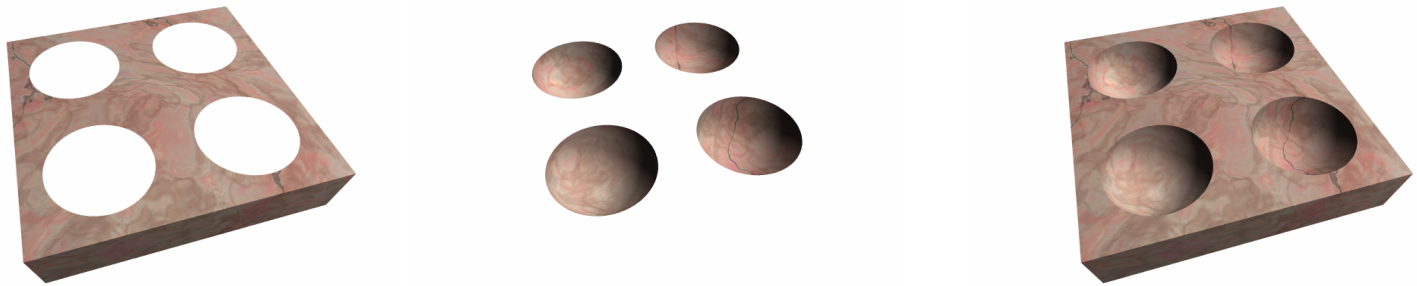


IV. Layered Goldfeather

- Idee:
  - es existieren max. *k* Schichten (Layer) je Produkt
  - jedes Fragment gehört zu genau einem Layer
  - Primitive eines Layers überlappen sich nicht

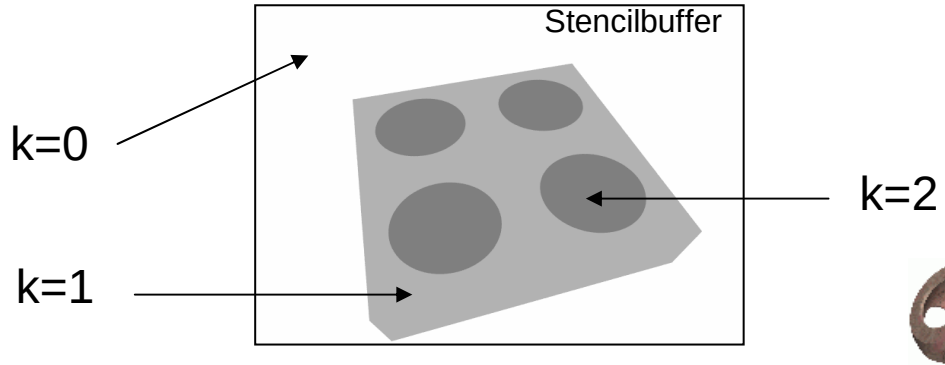
    → alle Primitive eines Layers gleichzeitig clippen

Layer 1 + Layer 2 = Resultat



IV. Layered Goldfeather

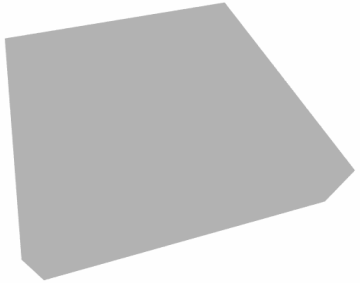
- Algorithmus:
  - Bestimmung von *k* für ein Produkt:
    - Stencil-Buffer löschen, Color- und z-Buffer sperren
    - Zeichnen aller Primitive *ohne* CSG oder z-Buffer
      - Culling einer Seite (sonst Primitive doppelt gezählt)
    - pro Fragment Stencil-Wert inkrementieren
      - maximaler Wert entspricht *k*
      - 8 Bit - 1 Paritätsbit → max. 2^7=128 Layer je Produkt



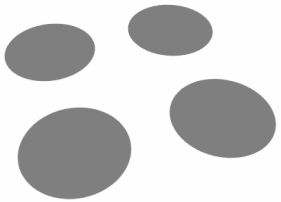
IV. Layered Goldfeather

- Algorithmus, Fortsetzung
  - Extrahieren der Layer je Produkt:
    - für jeden Layer *n*:
      - z- und Stencil-Buffer löschen
      - *alle* Primitive zeichnen und clippen
        - nur Fragmente mit Stencil = *n*
        - aber immer Stencil = Stencil + 1
      - Paritätstest
      - ggf. Fragmente löschen
      - restliche Fragmente mit z-less in den Framebuffer kopieren

Layer 1



Layer 2



IV. Layered Goldfeather

- Probleme
  - Trennung in Schichten ist kritisch:
    - exakte Reihenfolge der Primitive stets beibehalten
    - Layer wird gegen alle Primitive geclippt
      - d.h. auch gegen sich selbst
      - Lösung: Paritätstest mit z-less-or-equal
        - Rundungsfehler möglich → Artefakte im Bild
          - abhängig von Grafikkarte und Treiber
          - nVidia wenig betroffen
          - SGI u.U. große Probleme

V. Fazit

- Laufzeitkomplexität:
  - n – Anzahl Primitive
  - k – Anzahl Layer

  - Original Goldfeather: $\cong O(n^2)$
  - Layered Goldfeather: $O(2kn+2n)$ $\cong O(n \log n)$

- Geschwindigkeit:
  - Layered Goldfeather nur schneller, wenn deutlich weniger Layer als Produkte vorhanden sind
  - Juni 2002: Interaktivität nur bei ca. 100 Primitiven mit Standardhardware

V. Fazit

- Ausblick:
  - Behandlung der Clipping-Planes des View Frustums
  - CSG-Bäume ändern sich durch Interaktion:
    - Wiederverwendung normalisierter Bäume bei nur kleinen Änderungen im Baum

      → Caching
  - stärkere Ausdünnung (pruning) des Baums

Fragen

? ? ?

VI. Quellen

[1] Jack Goldfeather, Steven Molnar, Greg Turk und Henry Fuchs: „*Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning*“, IEEE Computer Graphics and Applications, Ausgabe 9(3), Seiten 20-28, 1989

[2] Tim F. Wiegand: „*Interactive Rendering of CSG models*“, Computer Graphics Forum, Ausgabe 14(4), Seiten 249-261, 1996

[3] Nigel Stewart, Geoff Leach und Sabu John: „*An improved Z-buffer CSG rendering algorithm*“, Proceedings of the Eurographics 98, Seiten 25-30, ACM Press, 1998

[4] http://www.nigels.com/research/

Mehrere Illustrationen entstanden nach Vorlage von [3] und [4].