Hauptseminar Graphische Datenverarbeitung

Thema: Blister: GPU-based rendering of Boolean combinations of free-form triangulated shapes

(Blister: GPU-Basierendes zeichnen von booleschen Gleichungen beliebiger triangulierter Objekte)

Von Tobias Hilbrich

Betreuender Hochschullehrer: PD Dr. S. Gumhold

Betreuer: Dr. W. Mascolus

Inhalt

- Einführung in CSG Ausdrücke
- Bisherige Ansätze
- Wichtige Regeln für CSG Ausdrücke
- Blister
  - Grundidee
  - Der „DepthPeeling“ Ansatz (DEMO)
  - Ein Klassifikationsbeispiel
  - Match & Flip
  - Minimale ID`s
  - GPU Klassifikation (DEMO)
  - „Layer“ Zusammenführung (DEMO)
- Zusammenfassung und Ausblick

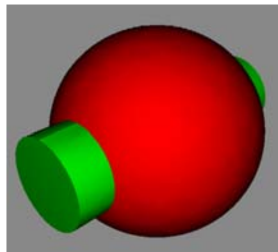
Einführung in CSG Ausdrücke (1/2)

- CSG ... Constructiv
- CSG Ausdrücke sin erknüpfte Grundobjekte(Primi
- Verwendung der O

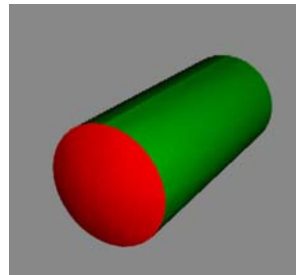
Beispiel

Primitiv A:

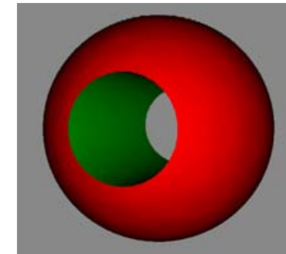
A ⋃ B: (Vereinigung)



A ⋂ B: (Schnitt)



A — B: (Differenz)

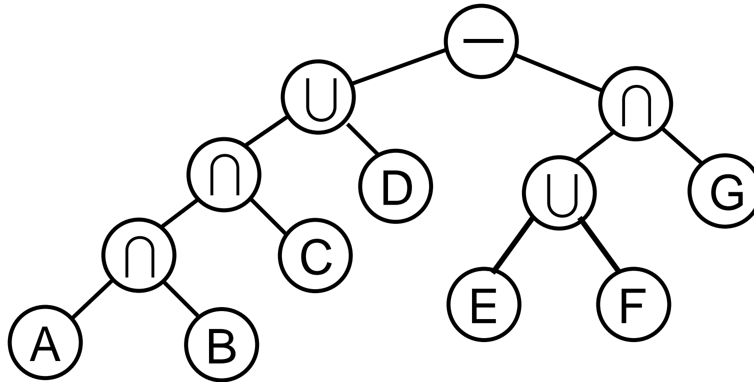


Einführung in CSG Ausdrücke (2/2)

- CSG Ausdrücke können als Bäume dargestellt werden

Beispiel

$$((((A \cap B) \cap C) \cup D) - ((E \cup F) \cap G)) = (ABC \cup D) - (E \cup F)G$$



Anwendung

- CAD z.B. Entwurf von Werkstücken, CNC Simulation
- 3D Modellierprogramme wie 3DStudio und MAYA

Bisherige Ansätze (1/2)

Allgemein

- Ansätze:
  - Berechnung der (triangulierten) Oberfläche des resultierenden Objektes
  - Darstellung ohne die Berechnung der Oberfläche (Blickwinkel abhängig)
- Auswerten von CSG Ausdrücken sehr rechenintensiv

Berechnung der Oberfläche

- Einmalige sehr komplexe Berechnung
- Günstig wenn alle beteiligten Objekte statisch
- Berechnung der Oberfläche „per Frame“ bei interaktiven Framerates bisher nicht möglich (bei komplexen CSG Ausdrücken)

➡ Günstig für CSG Ausdrücke mit statischen Objekten

Bisherige Ansätze (2/2)

Darstellung ohne Oberflächenberechnung

- Idee: Umformen des CSG Ausdruckes in spezielle Normalformen
- Ausnutzung der Eigenschaften der gewählten Normalform
- z.B. Disjunktive Normalform
- Ermöglicht Darstellung von CSG-Ausdrücken mit dynamischen Objekten
- Problem: Komplexität wächst meist kubisch (o. höher) mit Anzahl Primitive
- z.B. Disjunktive Normalform teils exponentielles Wachstum

➡ Bei ungünstigen CSG Ausdrücken nicht(kaum) anwendbar

- Blister nutzt die Blist-Normalform, welche im best case lineares Wachstum der Komplexität aufweist

➡ Erlaubt Auswertung komplexer, dynamischer CSG-Ausdrücke bei interaktiven Framerates

Wichtige Regeln für CSG Ausdrücke

Oberfläche

- Die Oberfläche eines Objektes das aus einer CSG Gleichung Y entsteht, besteht **NUR** aus Oberflächenteilen der an Y beteiligten Primitive.

Operationen und Wahrheitswerte

- Beim Auswerten eines CSG Ausdrucks Y an einem Pixel p werden die einzelnen Primitive P von Y durch W(wahr)/IN oder F(falsch)/OUT ersetzt je nachdem ob p in P liegt oder nicht.

Für alle CSG Ausdrücke Z,Z‘ gilt:

- $W \cup Z = W$
- $W \cap Z = Z$
- $Z - Z' = Z \cap \overline{Z}'$
- $F \cup Z = Z$
- $F \cap Z = F$
- $De`Morgan$

Blister

Grundidee

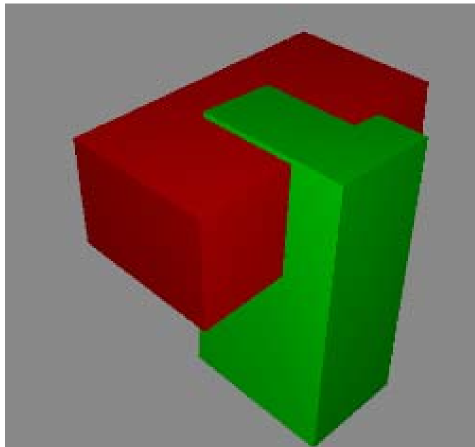
- Gegeben sei ein CSG Ausdruck Y
- Wandle Y in Positive-Normalform um
  (keine Differenz und Negationen nur über Primitiven)
- Bestimme alle Oberflächenteile der beteiligten Primitive (auf der GPU)
- Klassifiziere diese Oberflächenteile gegen Y (auf der GPU)
- Hierzu wird eine erweiterte Positive-Normalform genutzt, die sogenannte Blist-Normalform
- Füge alle Oberflächenteile zusammen die den Test gegen Y bestanden haben (auf der GPU)

Tobias Hilbrich
EMail: Tobias.Hilbrich@zih.tu-dresden.de

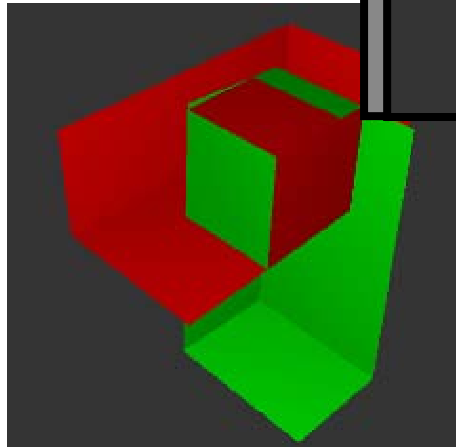
Der „DepthPeeling“ Ansatz (1/2

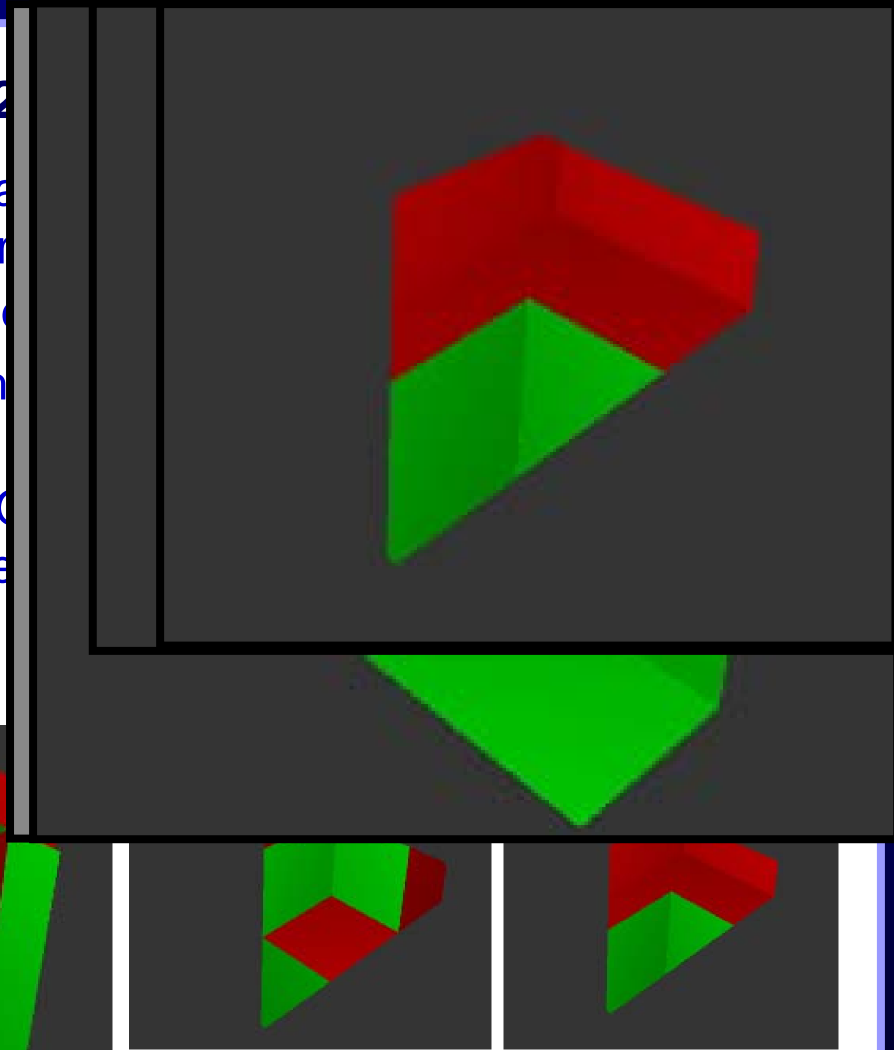
- Zum bestimmen der Oberflächenteile a kann das DepthPeeling verwendet wer
- Dieses zerlegt eine beliebige 3D Szene
- Die erste Schicht enthält die Oberfläch Nähesten gelegen sind
- Alle weiteren Schichten enthalten die C vorhergehenden Schicht liegen und de
- Beispiel:

1. Schicht



2. Schicht





Der „DepthPeeling“ Ansatz (2/2)

Umsetzung

- Verwendung von 2 Tiefenpuffern
- Erster Tiefenpuffer enthält die Tiefe der letzten Schicht (Referenz Puffer)
- Zweiter Puffer dient als üblicher Z-Puffer
- Akzeptiert werden Pixel deren Tiefe größer als der Wert im Referenz Puffer ist
- Übliche Grafikkarten haben keine 2 Tiefenpuffer !
    - GeForce FX Karten o. höher ermöglichen es Tiefenpuffer selber anzulegen und als Textur zu verwenden
    - Darstellung des Referenzpuffers als Textur
    - Mithilfe von Texturoperationen laden der entsprechenden Werte (OpenGL einfach, DirectX etwas schwieriger)
    - Sämtliche ATI und ältere GeForce Karten unterstützen kein **präzises** DepthPeeling und somit auch nicht Blister

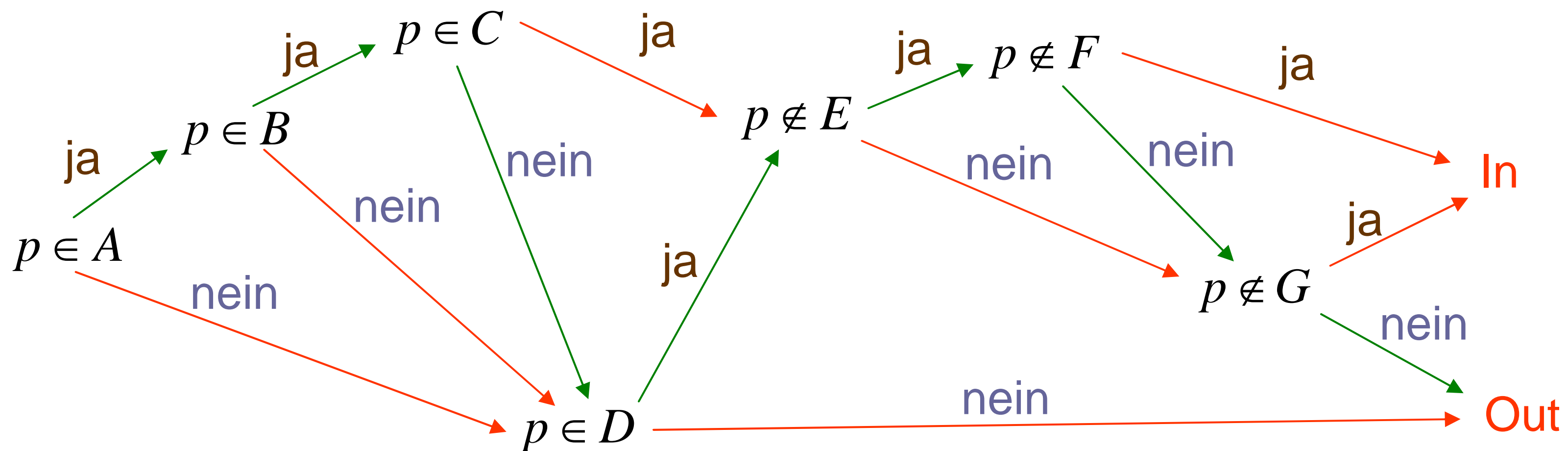
DEMO

Ein Klassifikationsbeispiel

- Es soll ein Pixel p gegen $Y = (ABC \cup D) - (E \cup F)G$ getestet werden
- Vorher bilden einer Positiven-Normalform Y` von Y
- Regeln: $X - Z = X \cap \overline{Z}$ $\overline{(X \cap Z)} = \overline{X} \cup \overline{Z}$ $\overline{(X \cup Z)} = \overline{X} \cap \overline{Z}$

$$Y' = (ABC \cup D)(\overline{E}\overline{F} \cup \overline{G})$$

- Auswertungsgraph



Match & Flip

- Beispiel für $Y' = (ABC \cup D)(\overline{E}\overline{F} \cup \overline{G})$

P	$rest(P)$	$next(P)$	$match(P)$	$flip(P)$
A	$(ABC \cup D)(\overline{E}\overline{F} \cup \overline{G})$	B	D	$false$
B	$(BC \cup D)(\overline{E}\overline{F} \cup \overline{G})$	C	D	$false$
C	$(C \cup D)(\overline{E}\overline{F} \cup \overline{G})$	D	E	$true$
D	$D(\overline{E}\overline{F} \cup \overline{G})$	E	Out	$false$
E	$\overline{E}\overline{F} \cup \overline{G}$	F	G	$true$
F	$\overline{F} \cup \overline{G}$	G	In	$false$
G	$\overline{G}$	Out	In	$false$

Minimale ID`s (1/2)

- Vermeidung von Spezialfällen für In und Out durch bilden von CSG Ausdruck Y` aus Ausdruck Y (in Positiver-Normalformen) wie folgt:

$$Y' = (Y \cup Out) \cap In$$

- Zur Klassifikation auf der GPU müssen die Primitive mit ID`s versehen werden, um next(P) und match(P) auswerten und darstellen zu können
- Dabei werden nur 6(7) bit für ID`s zur Verfügung stehen
  - Nur 64 (128) verschiedene ID`s
- Um CSG Ausdrücke mit mehr Primitiven darstellen zu können, müssen ID`s wiederverwendet werden
- Eine ID I kann für ein Primitiv P wiederverwendet werden, wenn für alle Primitive R links von P mit match(R) = I gilt, dass es ein Primitiv Q zwischen R und P gibt mit der ID I

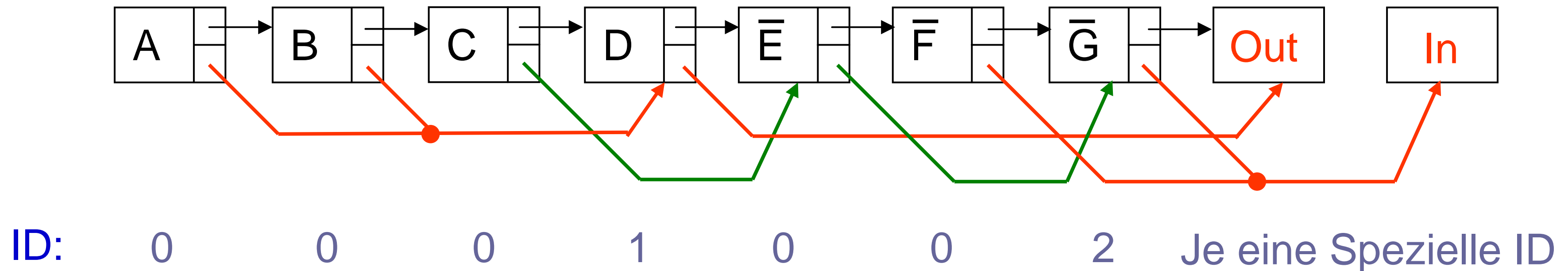
Minimale ID`s (2/2)

- Resultat für $Y = (ABC \cup D) - (E \cup F)G$

  $Y' = (ABC \cup D)(\overline{E}\,\overline{F} \cup \overline{G})$ (Positive-Normalform)

  ➡ $Y'' = ((ABC \cup D)(\overline{E}\,\overline{F} \cup \overline{G}) \cup Out)In$ (mit In und Out)

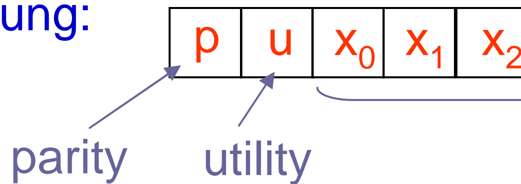
- Resultat der Linearisierung mit ID`s

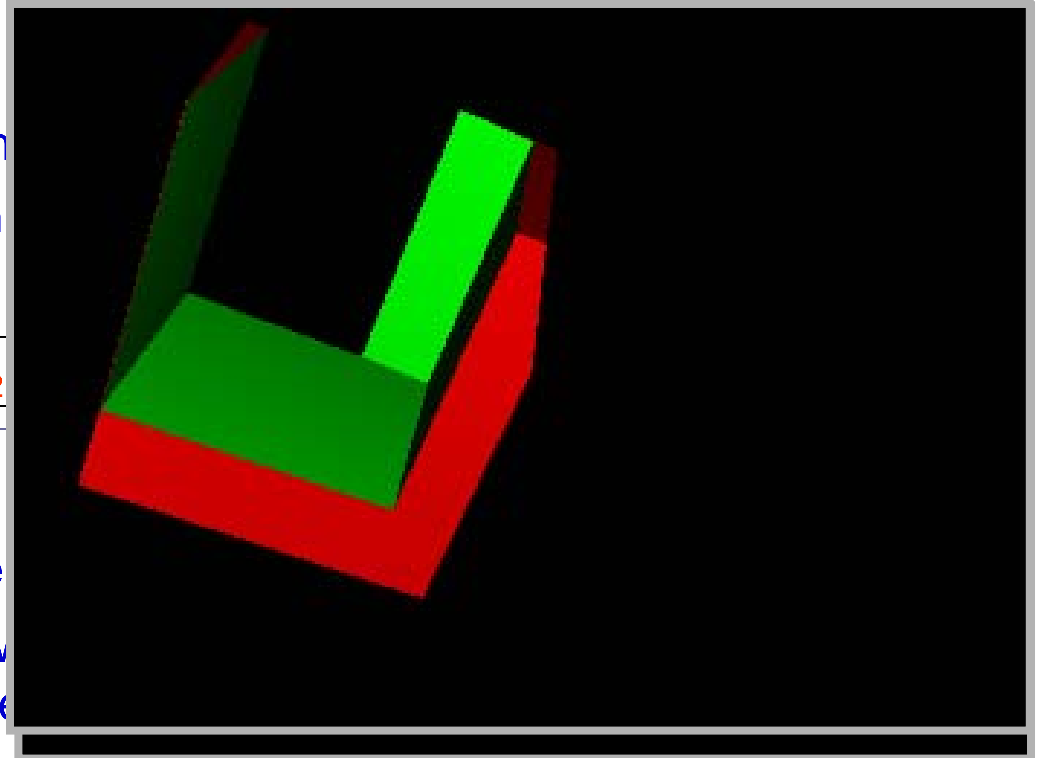


ID: 0 0 0 1 0 0 2 Je eine Spezielle ID

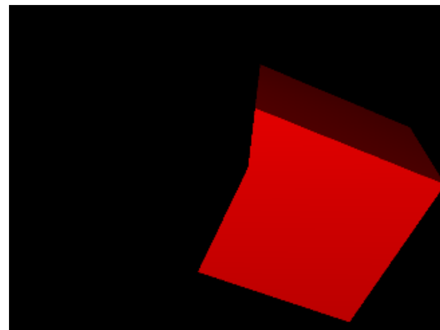
GPU Klassifikation

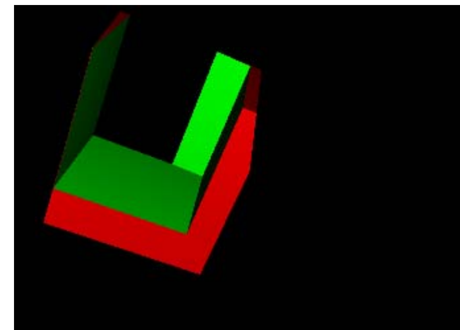
- Mithilfe des Stencilpuffers kann
- Stencilbuffer ist Teil des Tiefen
- Dieser hat bis zu 8 bit
  Aufteilung:





- ID ist die ID des nächsten für e
- Diese 8 bit können modifiziert v
  Dekrement, Ersetzung und Inve
- Nutzung der 8 bit für Vergleiche um Operationen nur auf bestimmte Pixel auszuführen
- Beispiel:

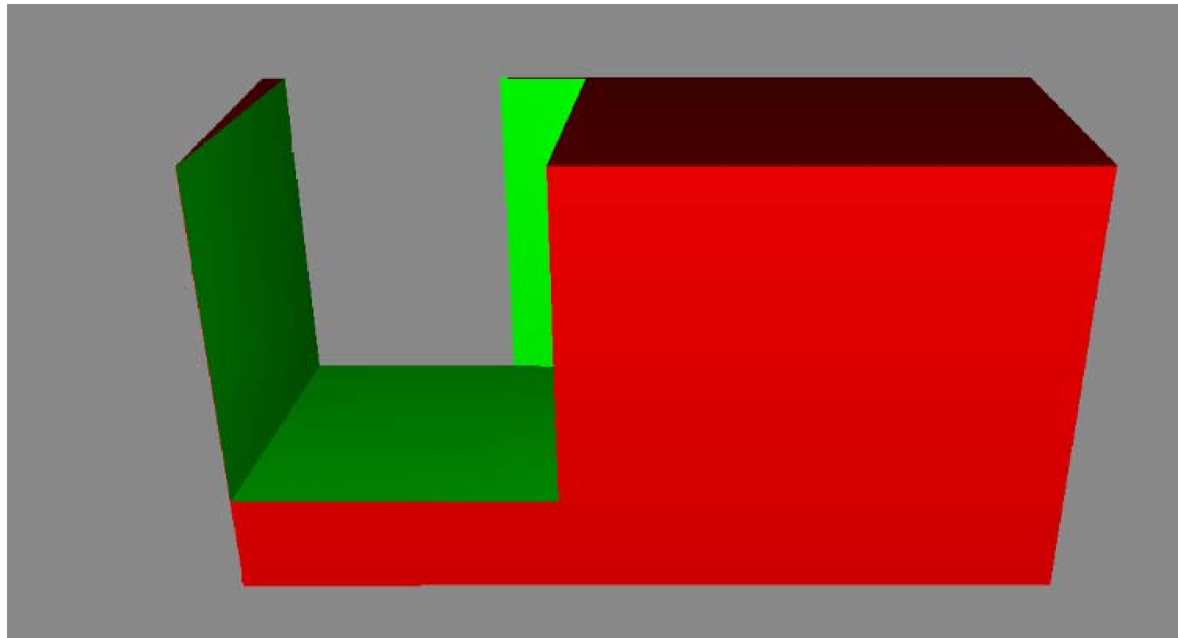




DEMO

„Layer“ Zusammenführung

- Das resultierende Objekt für den CSG Ausdruck entsteht indem alle klassifizierten layer zusammengeführt werden
- Dabei muss mit dem letzten layer begonnen werden, da dieser am weitesten hinten liegt
- Zum Zusammenführen kann eine Stenciloperation oder ein Pixelshader verwendet werden
- Beispiel



DEMO

Zusammenfassung und Ausblick

- Es wurde ein Verfahren vorgestellt, dass es ermöglicht CSG Ausdrücke in Echtzeit auf handelsüblichen Grafikkarten auszuwerten und darzustellen
- Dabei wurde die Blist-Normalform genutzt
- Diese dient dazu einen CSG Ausdruck zu linearisieren
- Blister nutzt den folgenden Ablauf

Initialisierung:
- Bilde Blist-NF und bestimme minimale ID`s

Darstellung:
- Bestimme das DepthPeel für layer l
- Klassifiziere l mithilfe von Stenciloperationen
- Füge die layer zusammen und beginne dabei mit dem letzten

- Das Verfahren hat eine Komplexität von O(n*k), wobei n Anzahl der Primitive und k Anzahl der benötigten Schichten ist
- Eine Implementierung ist mit OpenGL und mit DirectX (9.0c) möglich
- Allerdings wird eine GeForce FX Grafikkarte oder höher benötigt

Literaturverzeichnis

- Everitt, C. *Interactive order.independent transparency*. Tech Report, nVidia Corporation *http://developer.nvidia.com* 2002
- Goldfeather J., Hultquist J. und Fuchs H. *Fast constructive solid geometry display in the pixelpowers graphics system*. Annual Conference on Computer Graphics and Interactive Techniques 107-116. 1986
- Microsoft Corporation. *DirectX Documentation for C++*. DirectX 9.0 SDK Update August 2005: Stencil Buffer Techniques. 2005
- John Hable, Jarek Rossignac. *Blister: GPU-based rendering of Boolean combinations of free-form triangulated shapes*. *http://www.johnhable.com/Blister/Blister.pdf* 2005
- Requicha A, Voelcker H. *Boolean operations in solid modeling: Boundary evaluation and merging algorithms.* Procedings of the IEEE, 75, 1, 30-44. 1985
- Tilove, R.B. *A Null-Object Detection Algorithm for Constructive Solid Geometry.* Communications of the ACM 27, 7, 684-694. 1984

Wichtige Regeln für CSG Ausdrücke (2/2)

Konsequenzen

- Das resultierende Objekt eines CSG Ausdrucks besteht aus einer Teilmenge der Oberflächenteile aller beteiligten Primitive
- Welche dieser Oberflächenteile wirklich zum Objekt gehören kann durch Auswertung des CSG Ausdrucks für die Pixel der Oberflächenteile herausgefunden werden
- Es ist möglich die Differenz in einen Schnitt mit einer Negation umzuwandeln
- Der CSG Ausdruck muss für einen Pixel nur solange ausgewertet werden bis sich W oder F für den Gesamtausdruck ergibt
- Dazu müssen unter Umständen nur Teile des CSG Ausdruckes ausgewertet werden

Match & Flip (1/2)

- Beobachtungen:
  - Es genügt jedes Primitiv einmalig zu testen
  - Wann immer ein Primitiv P gegen einen Pixel getestet wird, gibt es nur 2 Fälle
    (1) Das nächste zu betrachtende Primitiv ist der rechte Nachbar von P
    (2) Das nächste zu betrachtende Primitiv ist ein weiter rechts gelegenes Primitiv
  - Das Primitiv von (1) wird als next(P) und (2) als match(P) bezeichnet
  - flip(P) entscheidet ob das nächste Primitiv der rechte Nachbar von P ist oder match(P)
- flip(P) hängt von der „rechten“ Operation von P ab und ob P negiert ist oder nicht
- Die Semantik von flip(P) muss festgelegt werden: (P ... Primitiv, p ... Pixel)
  - flip(P) == false und p in P dann betrachte next(P)
    flip(P) == true und p in P dann betrachte match(P)

GPU Klassifikation (2/2)

- Vorgehen zum Klassifizieren eines Primitives P gegen layer des DepthPeels
  - Für alle Pixel p mit stencil(p).id == P.id
    setze stencil(p).parity = 0, stencil(p).utility = 1
  - Zeichne P und führe für alle Pixel von P die vor einen Pixel p des layers liegen folgende Operation aus
    wenn (stencil(p).utility == 1) dann invertiere stencil(p).parity
  - Für alle Pixel p mit stencil(p).utility == 1 und stencil(p).parity == 1 (Pixel ist aktiv für P und innerhalb von P)

    wenn P.flip == 0 setze stencil(p).id == next(P)

    wenn P.flip == 1 setze stencil(p).id == match(P)
  - Für alle Pixel p mit stencil(p).utility == 1 und stencil(p).parity == 0 (Pixel ist aktiv für P und innerhalb von P)

    wenn P.flip == 0 setze stencil(p).id == match(P)

    wenn P.flip == 1 setze stencil(p).id == next(P)

DEMO

Tobias Hilbrich

EMail: Tobias.Hilbrich@zih.tu-dresden.de

Mögliche Erweiterungen und Optimierungen

- Mitführen von aktiven Bereichen für Klassifikation
- Darstellung von Transparenten Objekten
- Reduzierung der Anzahl der Stenciloperationen
- Nutzung von OpenGL Extension die XOR im Stencilpuffer ermöglicht damit wird utility nicht mehr benötigt -> 7 bit für ID
- Shadowmapping
- Optimierung von CSG Ausdrücken um Bäume zu erstellen die möglichst wenige ID`s benötigen