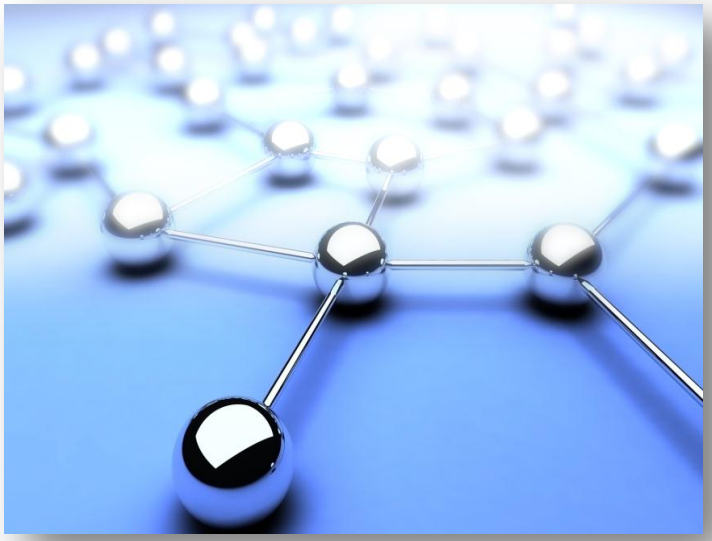
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Vorlesung Algorithmen für hochkomplexe Virtuelle Szenen

Sommersemester 2012



Matthias Fischer
mafi@upb.de

Vorlesung 14
10.7.2012

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Paralleles Rendering als Sortierproblem

- Paralleles Rendering
- Klassifizierung nach Molnar et. al.
- Sort-First, Sort-Middle, Sort-Last

Literatur

- Tomas Akenine-Möller, Eric Haines
  Real-Time Rendering
  AK Peters, 2002
- Dirk Bartz, Claudio Silva
  Rendering and visualization in parallel environments
  Tutorial of EUROGRAPHICS 2001, Tutorial 9. Eurographics Association, 1995
  http://www.gris.uni-tuebingen.de/~bartz/tutorials/eg2001tutorial
- E. Reinhard, A. G. Chalmers, F. W. Jansen
  Overview of parallel photo-realistic graphics
  STAR - State of the Art Reports, EUROGRAPHICS 1998, pages 1–25. Eurographics Association, 1998
  http://www.cs.bris.ac.uk/Publications/Papers/1000271.pdf
- Steven Molnar, Michael Cox, David Ellsworth, Henry Fuchs
  A Sorting Classification of Parallel Rendering
  IEEE Computer Graphics and Applications, 14(4):23–32, 1994
  http://citeseer.ist.psu.edu/32674.html

Paralleles Rendering als Sortierproblem

Paralleles Rendering

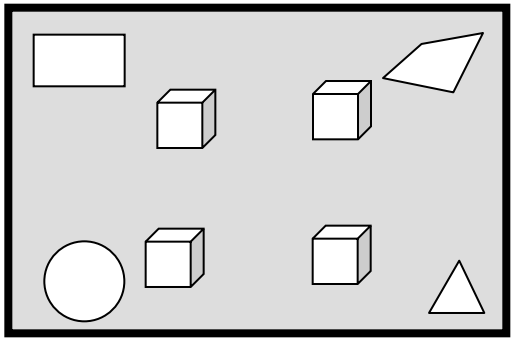
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Renderingprozess besteht aus zwei Hauptschritten

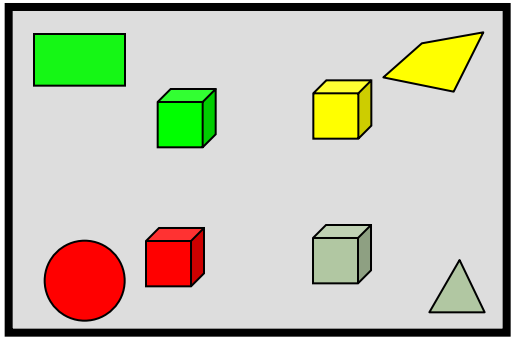
- Geometrietransformation
- Rasterung

Sequentielle Berechnung

- Polygone können in beliebiger Reihenfolge von der Grafikkarte verarbeitet werden
- Ergebnis von der Reihenfolge unabhängig



Geometrietransformation 3D→2D

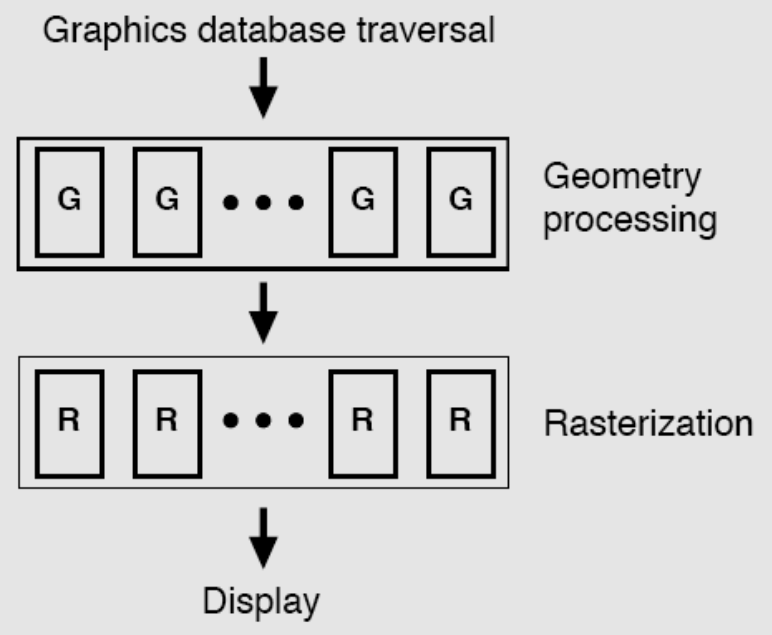


Rasterung

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Was verstehen wir unter parallelem Rendering?

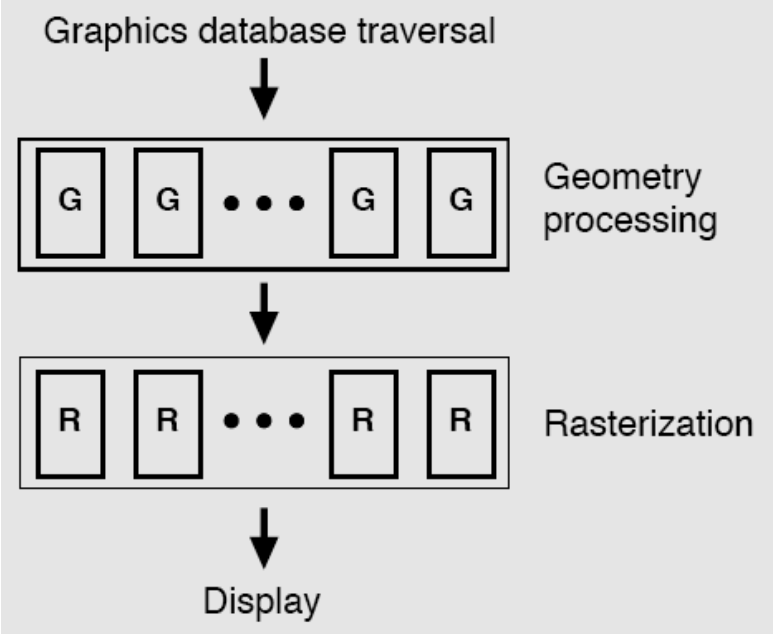
- Geometrietransformation wird parallel ausgeführt (mehrere Geometrieeinheiten)
- Rasterung wird parallel ausgeführt (mehrere Rasterungseinheiten)



HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Warum geht das mit einer Pipeline-Architektur?

- Ergebnis des Rendering ist unabhängig von der Reihenfolge, in der Dreiecke verarbeitet werden
- keine Abhängigkeit zwischen den Berechnungen einzelner Dreiecke
- Z-Buffer Algorithmus bzw. die Pipelinearchitektur erlaubt die parallele unabhängige Verarbeitung einzelner Pixel und Dreiecke



HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Wie erfolgt die parallele Verarbeitung der Geometrie?

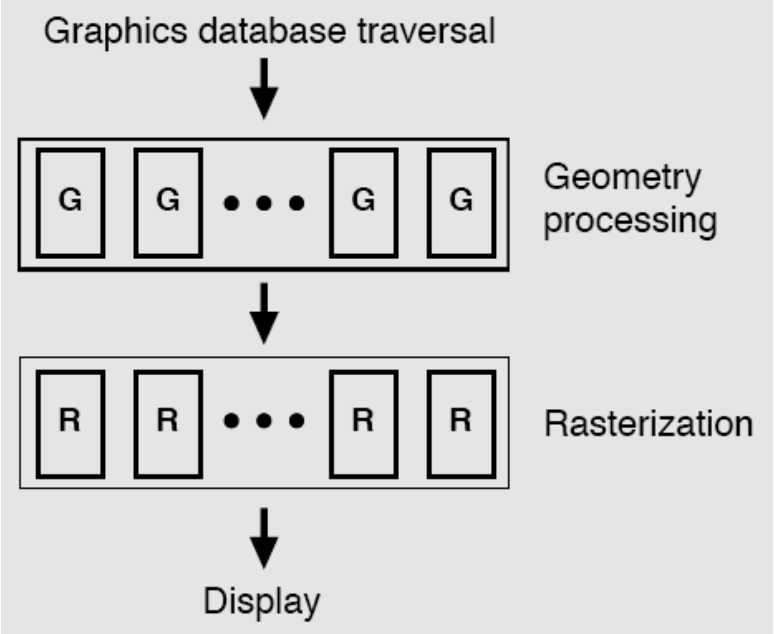
- jeder Prozessor erhält eine Teilmenge der Geometrie

Wie erfolgt die parallele Verarbeitung der Rasterung?

- jeder Prozessor erhält einen Teil der zu rendernden Pixel

Was bleibt zu klären?

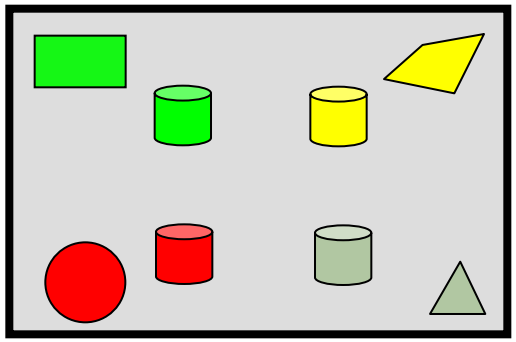
→ Wie erfolgt die Zuordnung und Aufteilung der Teilaufgaben auf die einzelnen Prozessoren!

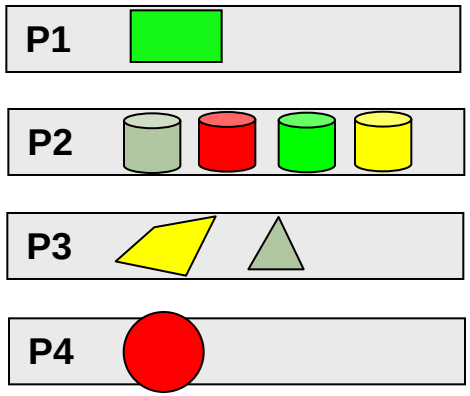


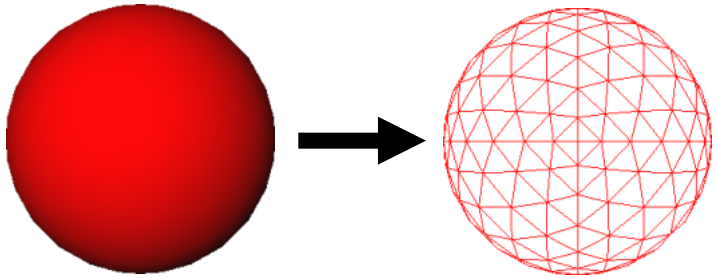
Paralleles Rendering als Sortierproblem

Annahmen

- Vor der Berechnung eines Bildes sind die Objekte beliebig über alle Prozessoren verteilt gespeichert.
- Die Objekte sind ohne Redundanz gespeichert, d.h. jedes Objekt ist auf genau einem Prozessor.
- Objekte sind beliebig im 3D-Raum platziert und erscheinen standpunktabhängig an beliebigen Stellen des Bildschirms.
- Falls die Objekte abstrakt beschrieben sind, müssen sie noch in Dreiecke zerlegt werden (tesselieren).





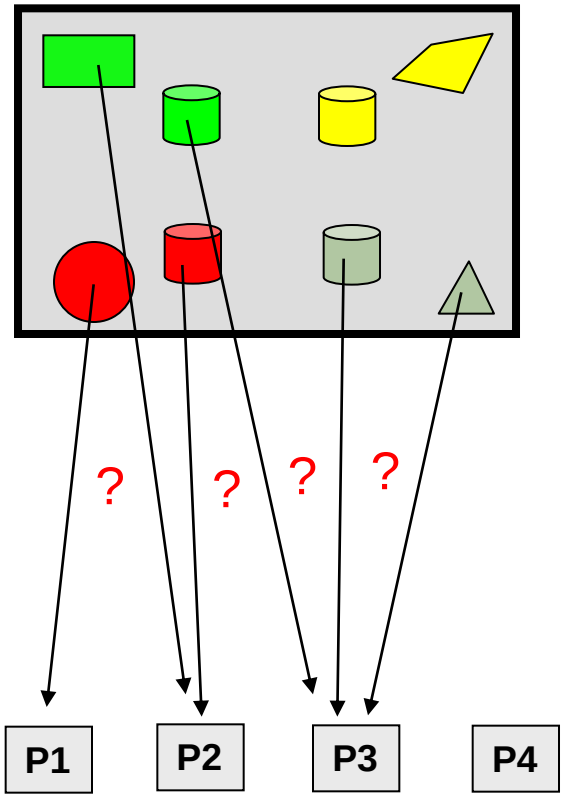


Bei mehreren Prozessoren stellen sich neue Probleme

- Welcher Prozessor berechnet welchen Teil der Geometrie?
- Wann wird die Geometrie auf die Prozessoren verteilt?
- Wann und wie wird das Bild aus den Teilbildern zusammengesetzt?

Dabei sind Effizienzprobleme zu berücksichtigen

- Wie werden die Prozessoren gleichmäßig ausgelastet „Load Balancing“ ?
- Wie wird mehrfaches Zeichnen der Primitive von mehreren Prozessoren vermieden? „Overlapping“



HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

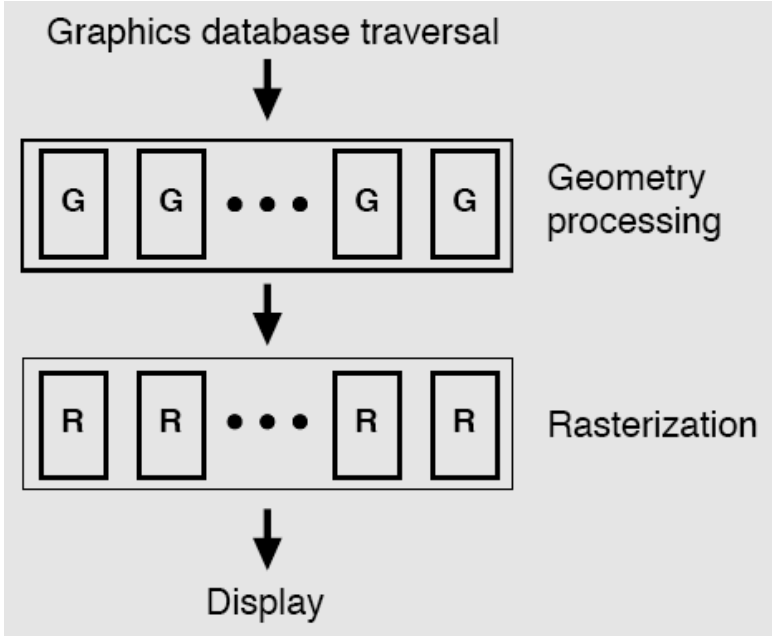
Zentrale Frage

Wann erfolgt die „Sortierung“ (= Verteilung, =Zuordnung) der Objekte an die Stelle des Bildschirms, an der sie dargestellt werden?

Geometrietransformation und Rasterung arbeiten unabhängig voneinander.

→ Sortierung kann erfolgen:

- vor der Geometrietransformation (Sort-First)
- zwischen Geometrietransformation und Rasterung (Sort-Middle)
- nach der Rasterung (Sort-Last)

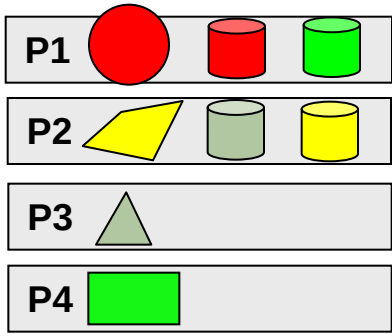


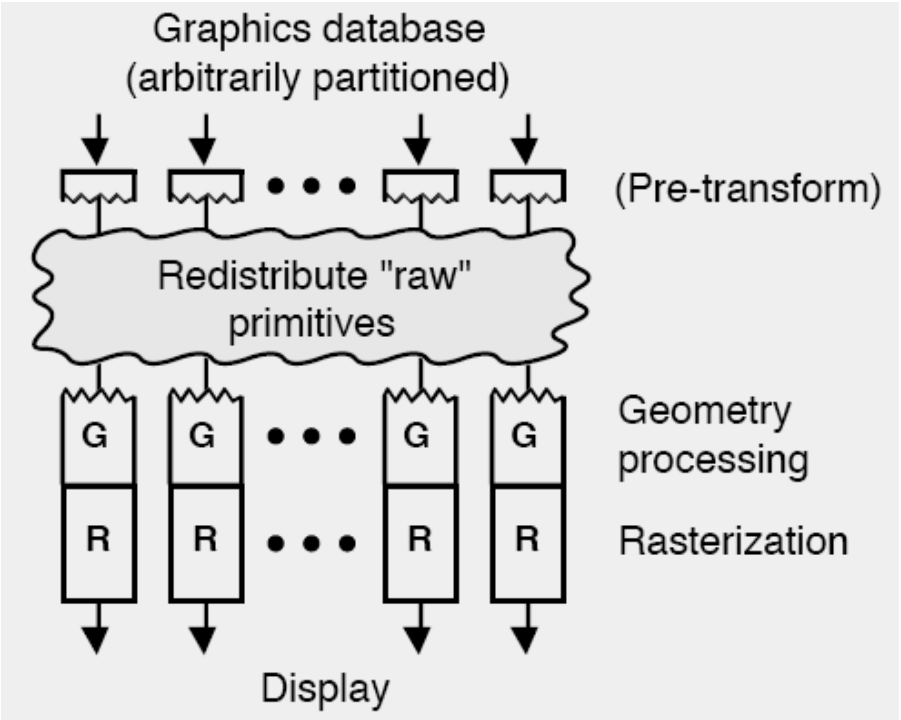
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Sort-First

Ausgangssituation

- vor der Bildberechnung sind die Objekte beliebig auf den Prozessoren verteilt





Paralleles Rendering als Sortierproblem

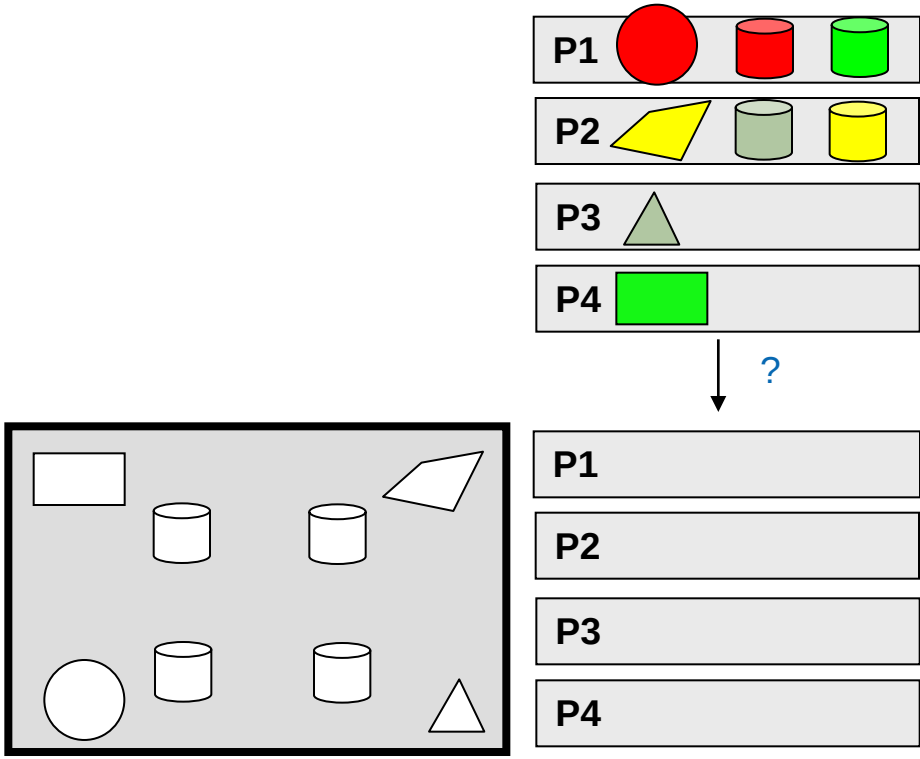
Sort-First

Schritt 1: Verteilung – „Sort“

- Verteilung (Sortierung) der Objekte erfolgt vor der Geometrietransformation auf die Prozessoren
- die Bildraumkoordinaten (2D-Werte) sind vorher unbekannt

Was wird verteilt?

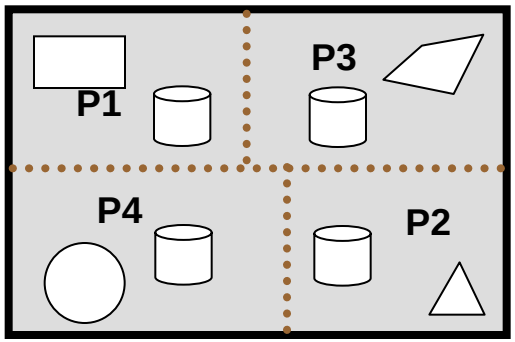
- Objekte im 3D-Objektraum



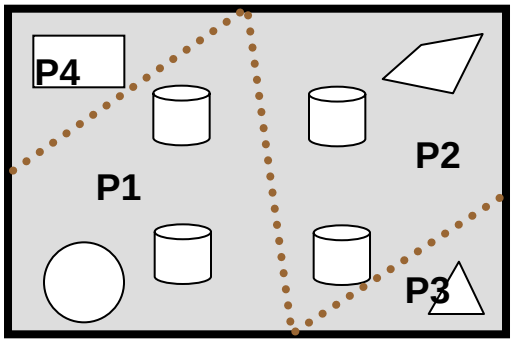
Wie werden die Objekte verteilt?

- beliebige Aufteilung des Bildschirms in disjunkte Bereiche: „Kacheln“
- bspw. gleichmäßiges Schachbrett, Rechtecke, usw.
- jeder Prozessor ist für das Rendering einer Kachel zuständig

achsenparallele Kacheln

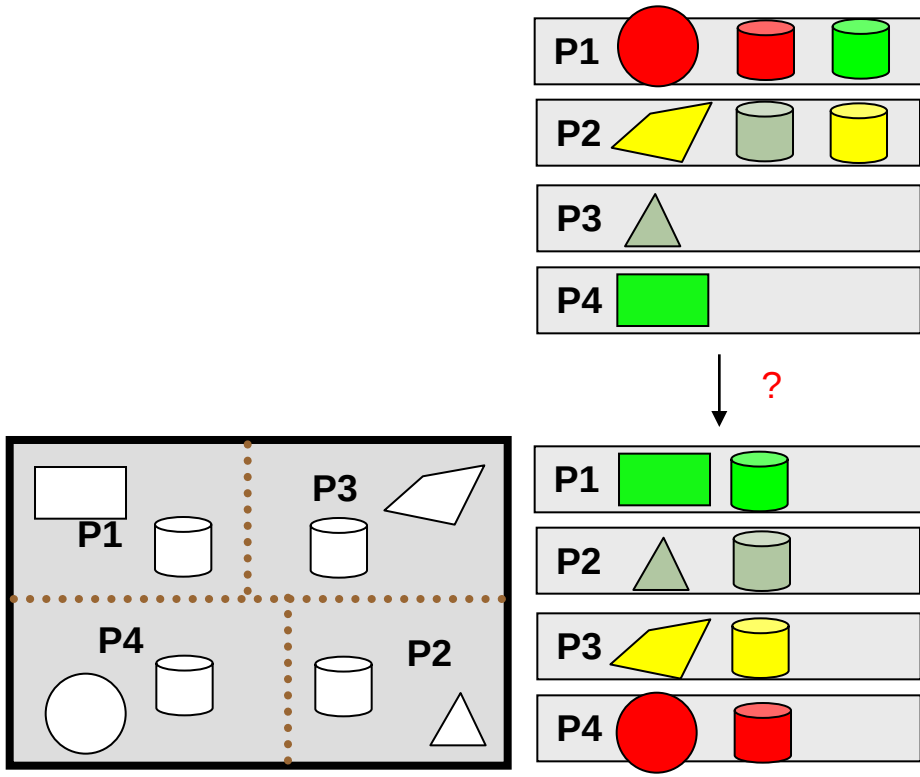


“beliebige” Kacheln möglich



Wie werden die Objekte verteilt?

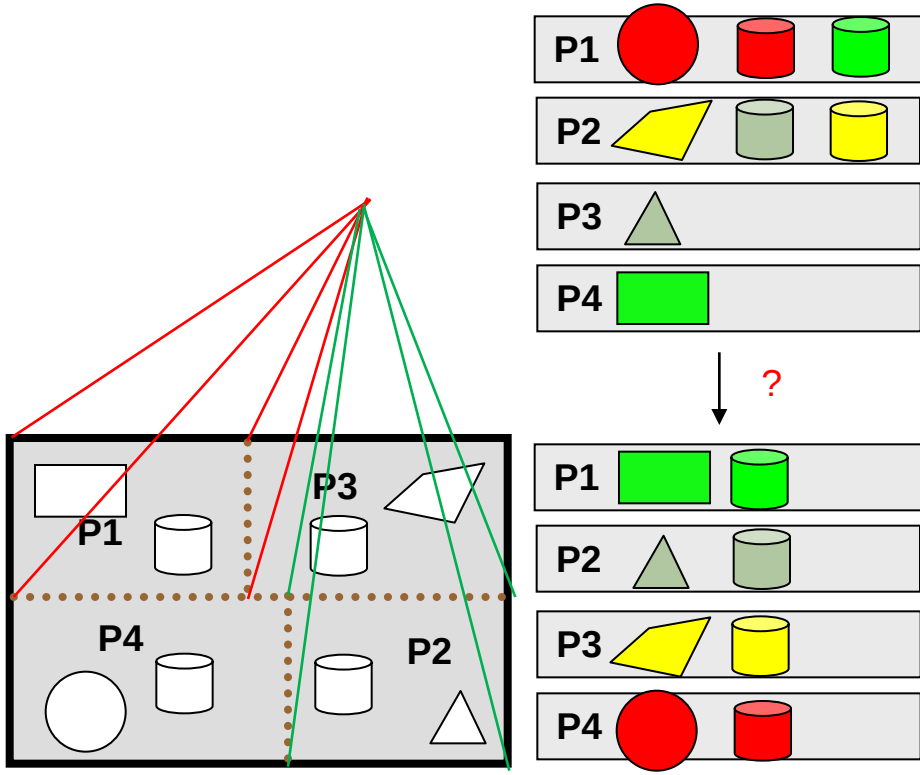
1. Transformation der Objekte in den Bildraum
   - mit Hilfe einer Transformation werden die Objekte den Kacheln zugeordnet
   - Transformation bezieht sich typischerweise auf Boundingboxen von einem oder mehreren Objekten
   - Dazu verwendet man eine räumliche Datenstruktur (z.B. Octree)



Wie werden die Objekte verteilt?

1. Transformation der Objekte in den Bildraum
   ……..
   Beispiel:
   - Mit Hilfe eines Octree werden die Objekte gespeichert.
   - Berechne alle Objekte des Subfrustum; das Subfrustum bildet sich aus der Betrachterposition und einer Kachel.
   - Berechne den Inhalt für die Subfrusta der Prozessoren P1, P2, P3, P4

2. Die berechneten Objekte werden über das Netzwerk zu den Prozessoren P1, P2, P3, P4 geschickt

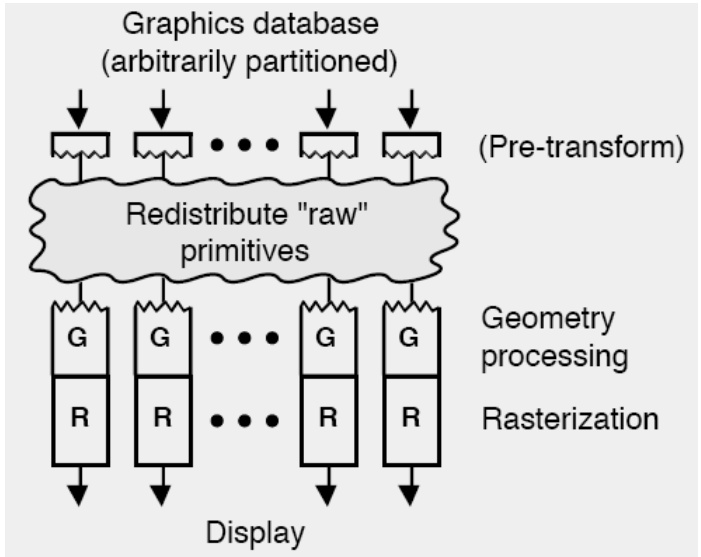


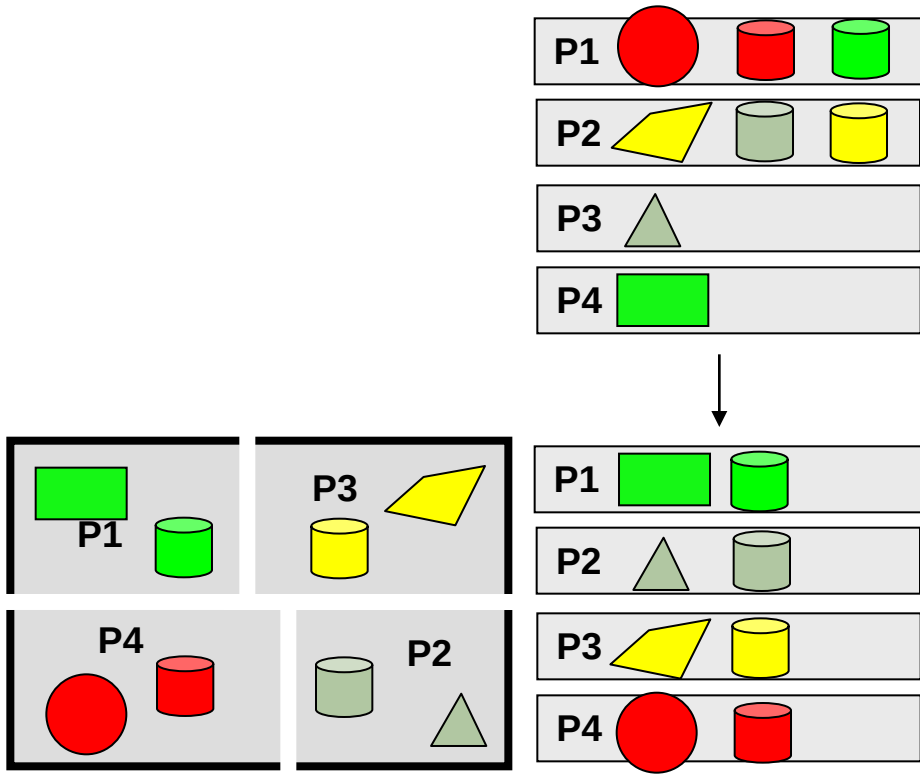
Geometrietransformation der Knoten eines Dreiecks (Renderer) findet erst im 2. Schritt statt

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Schritt 2: Rendering

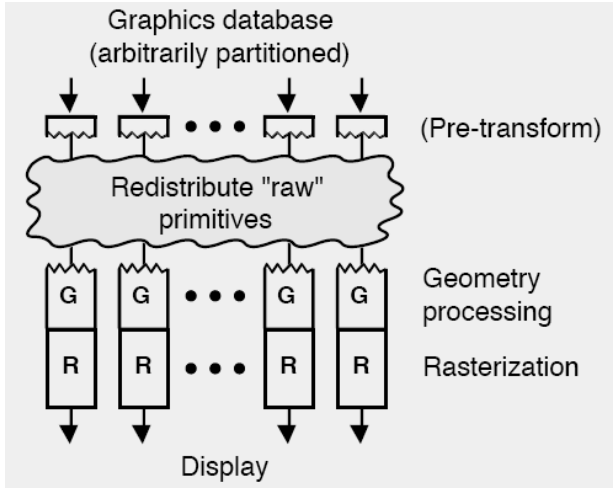
- Geometrietransformation und Rasterung erfolgen auf demselben Prozessor
- jeder Prozessor berechnet ein Teilbild / Kachel

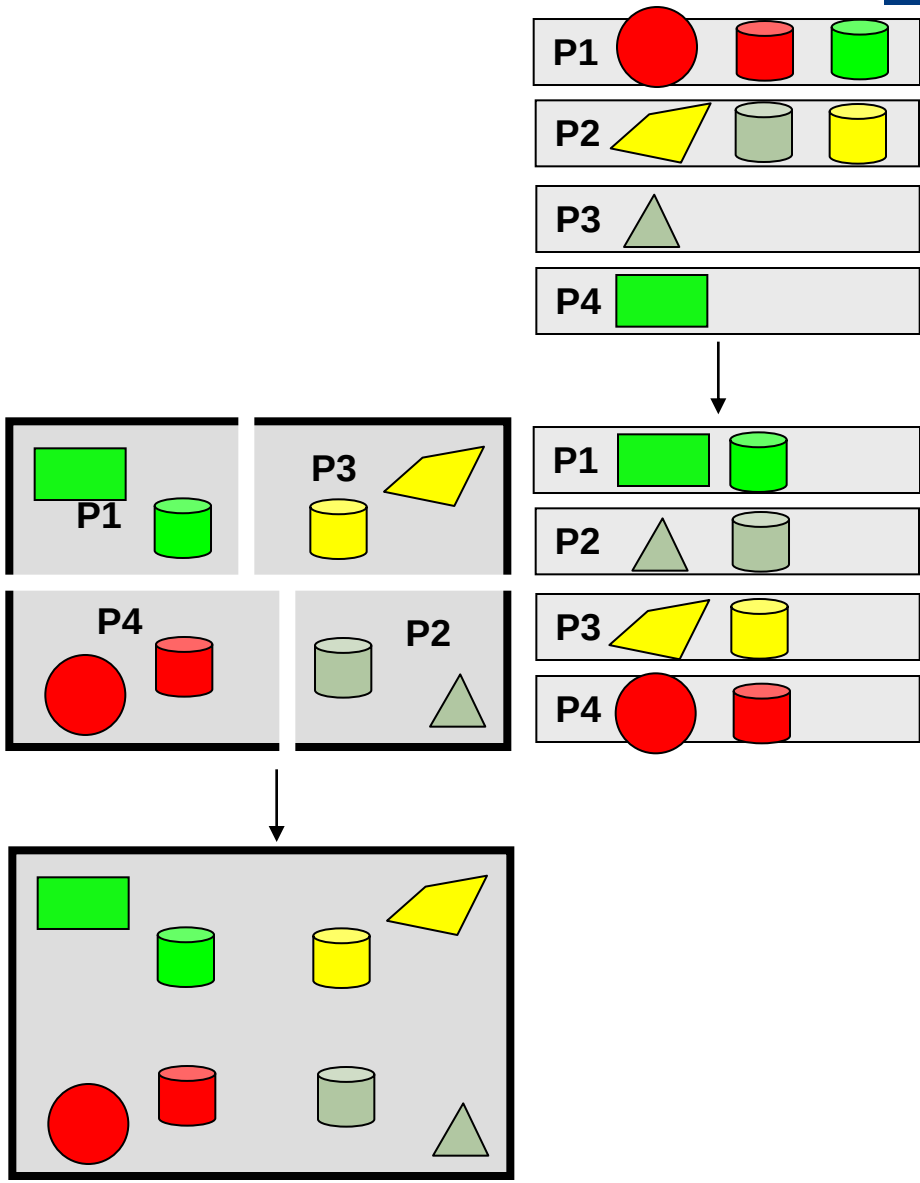




Schritt 3: Darstellung des Bildes

- Zusammensetzung der einzelnen Teilbilder nebeneinander
- dafür gibt es Hardware- und Softwarelösungen
- Softwarelösung:
  Teilbilder (Kacheln) werden über das Netzwerk zu einem Prozessor geschickt, der für die Darstellung des Gesamtbild zuständig ist.



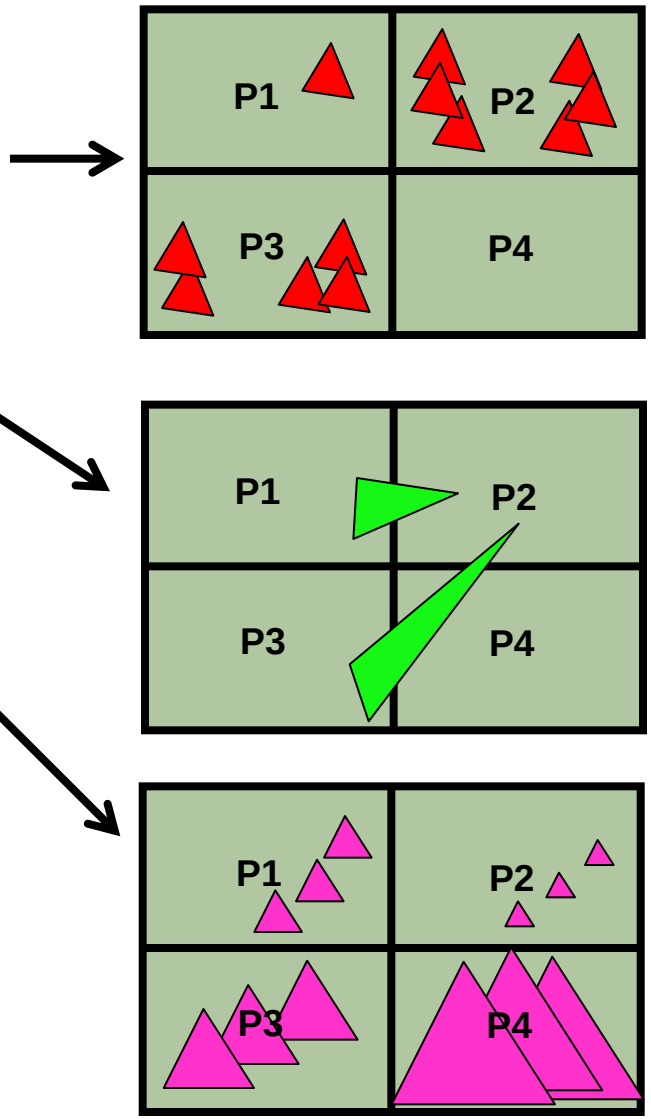


Probleme von Sort-First

- Zuteilung der Polygone auf die Geometrieprozessoren sollte gleichmäßig verteilt sein, sonst laufen einige Prozessoren leer.
- Polygone, die in mehrere Bildraumbereiche fallen, werden von mehreren Prozessoren verarbeitet.
- die Last von Geometrietransformation und Rasterung kann ungleich verteilt sein, da beides derselbe Prozessor bearbeitet

Kommunikations-Overhead gering, da nur geometrische Primitive verschickt werden (kleine Szenen)

Große Szenen?

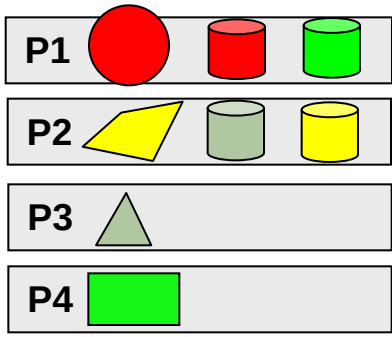


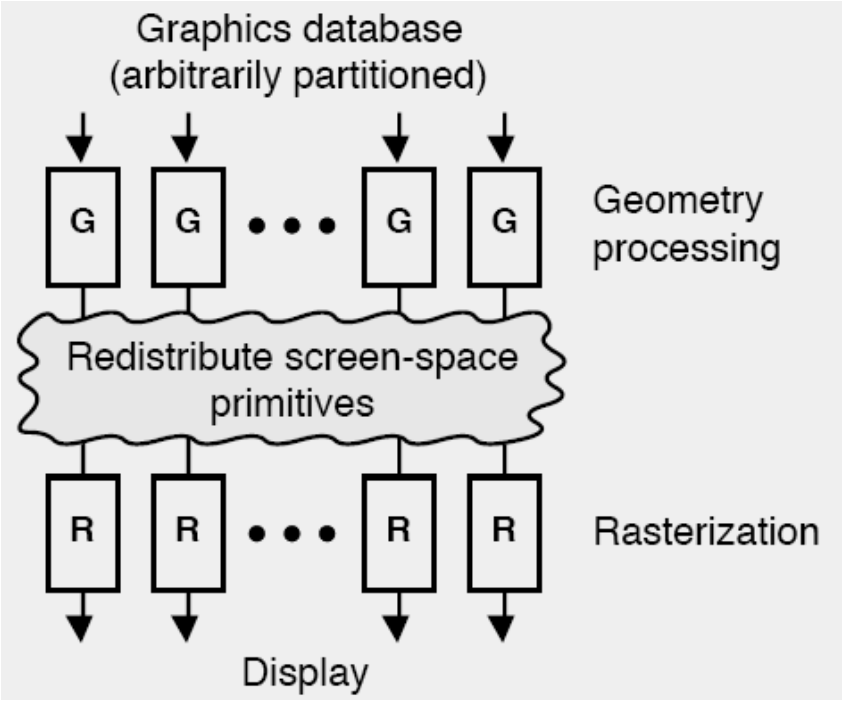
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Sort-Middle

Ausgangssituation

- vor der Bildberechnung sind die Objekte beliebig auf den Prozessoren verteilt



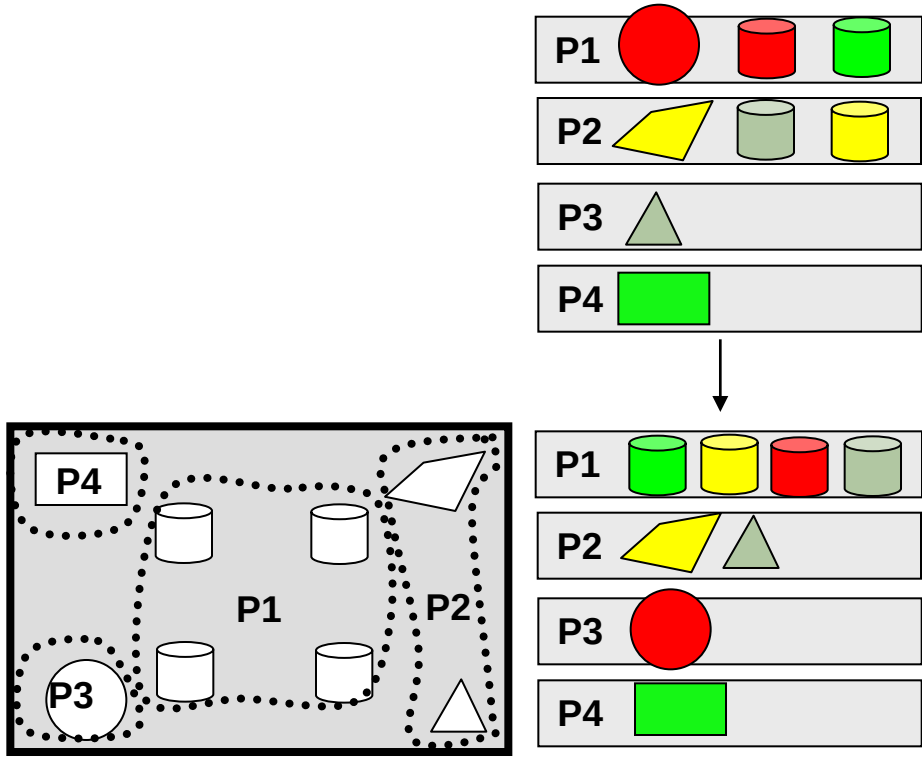


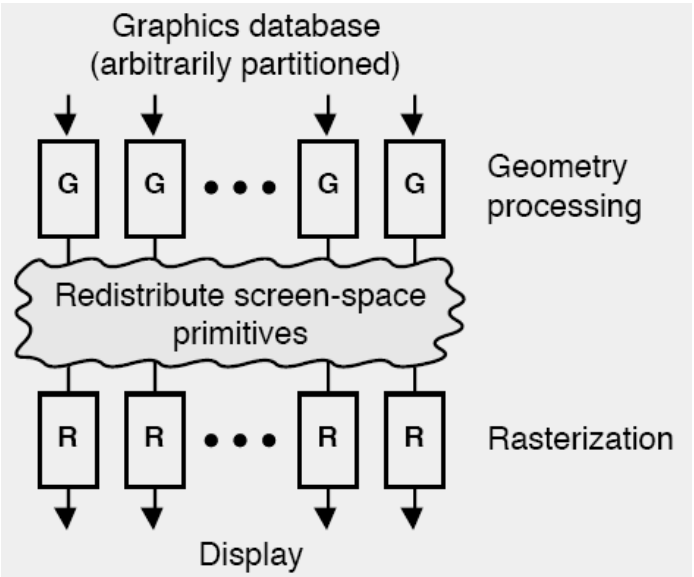
Paralleles Rendering als Sortierproblem

Sort-Middle

Schritt 1: Geometrietransformation

- die 3D-Objekte werden in den 2D-Bildraum transformiert
- zum Zwecke der Lastverteilung können die Objekte vorher ggfs. unter den Prozessoren umverteilt werden





Paralleles Rendering als Sortierproblem

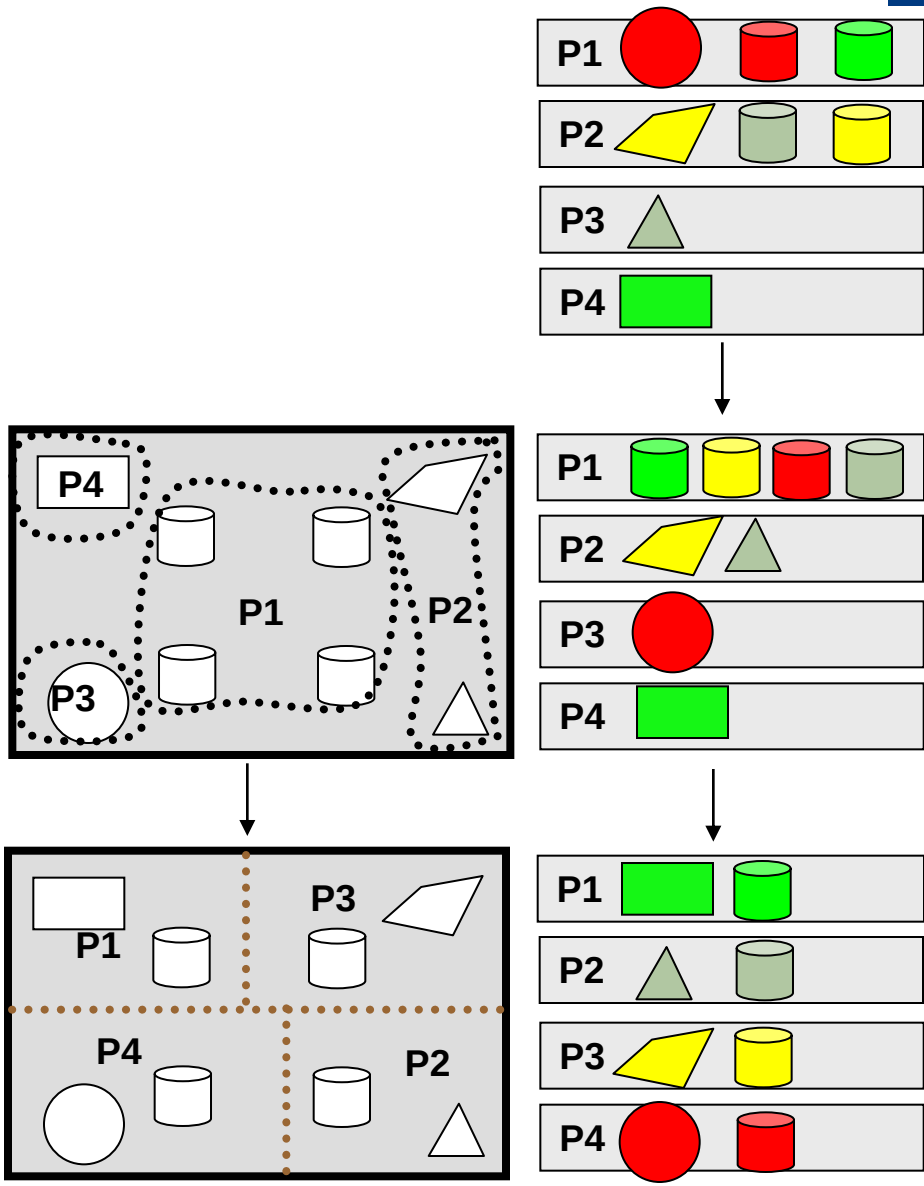
Sort-Middle

Schritt 2: Verteilung - „Sort“

- Verteilung (Sortierung) erfolgt nach der Geometrietransformation vor der Rasterung
- verteilt werden die in den 2D-Bildraum projizierten Polygone

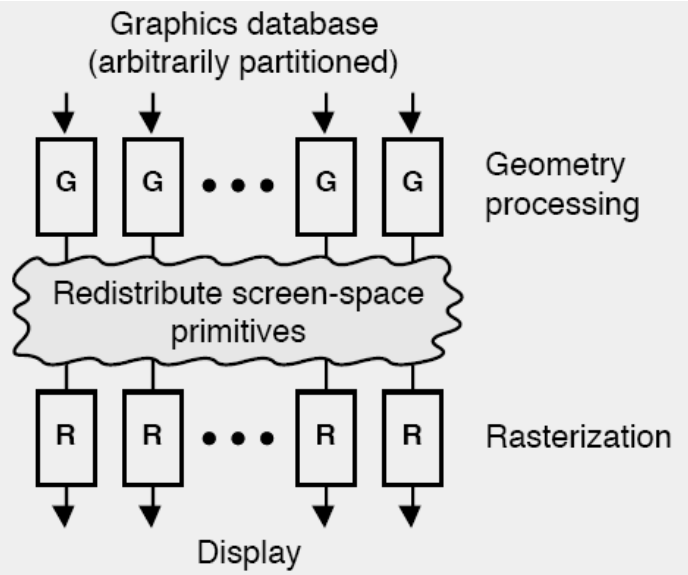
Wie werden 2D-Polgone verteilt?

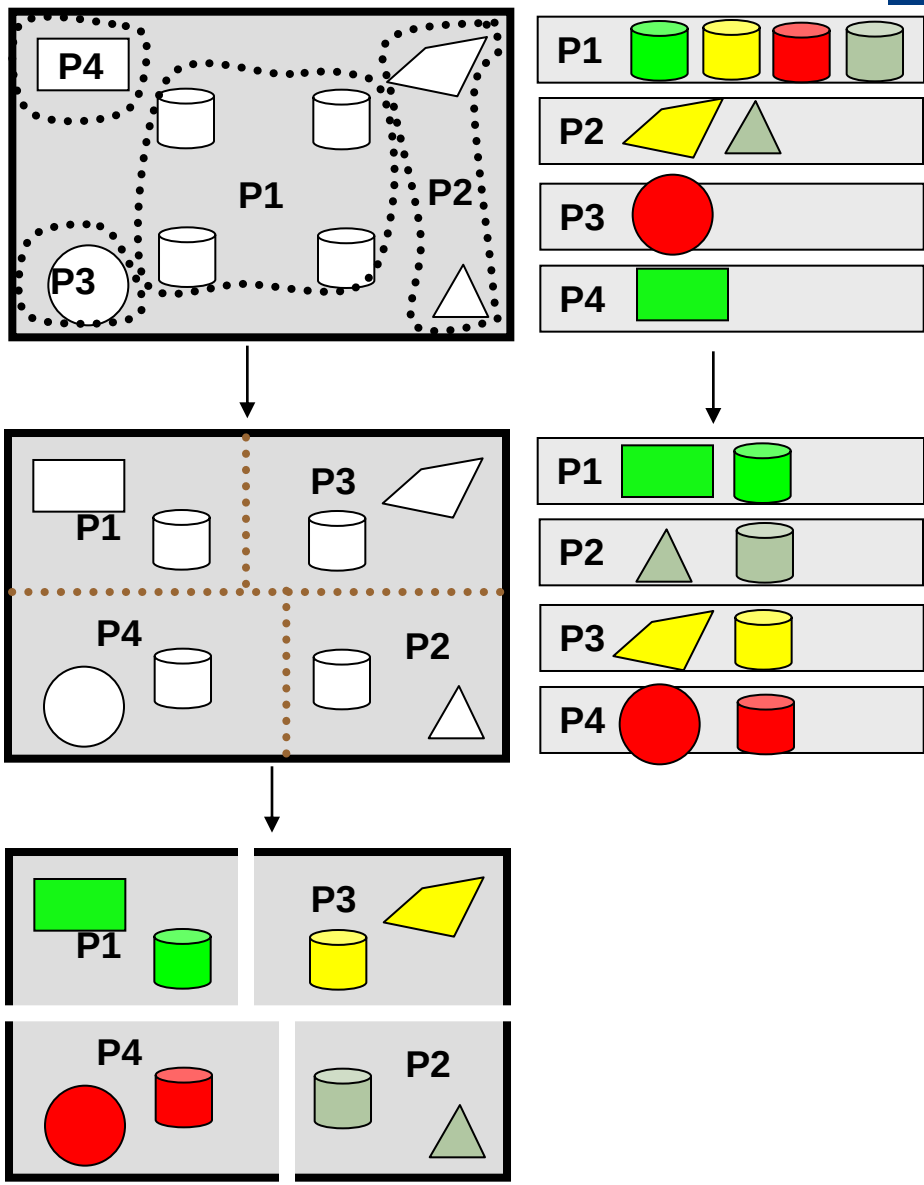
- so wie bei Sort-First: „Kachelung“
- statt der 3D-Objekte werden jedoch 2D-projizierte Polygone über das Netzwerk geschickt und in Kacheln verteilt



Schritt 3: Rasterung

- jeder Prozessor berechnet ein Teilbild / Kachel
- Prozessoren für Geometrie- und Rasterisierung müssen nicht gleich sein; getrennte Spezialhardware für beide Komponenten ist möglich

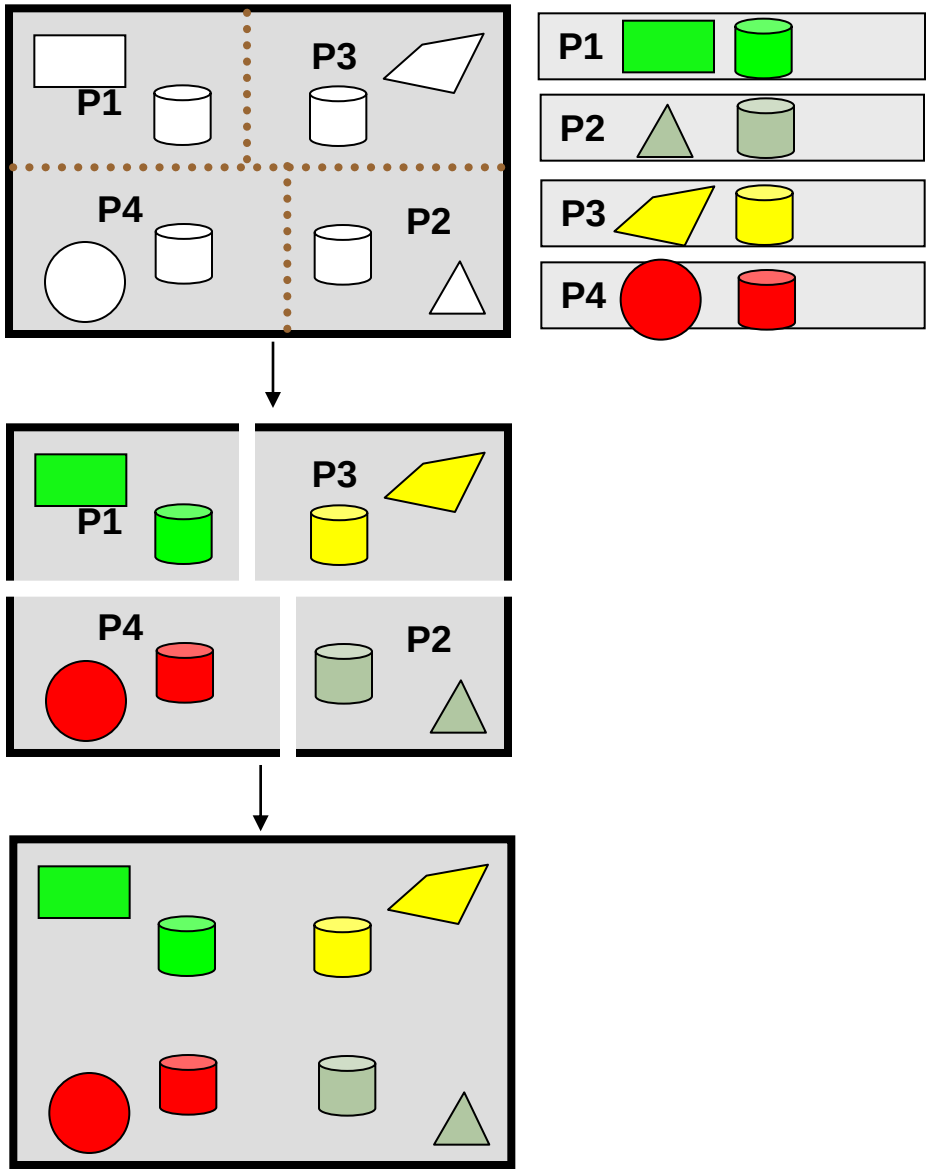


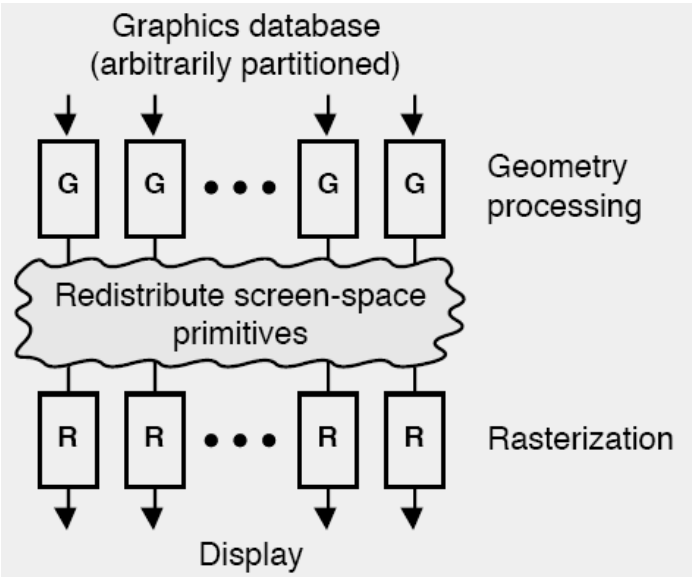


HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Schritt 4: Darstellung des Bildes

- Zusammensetzung der einzelnen Teilbilder nebeneinander wie bei Sort-First





Paralleles Rendering als Sortierproblem
Sort-Middle

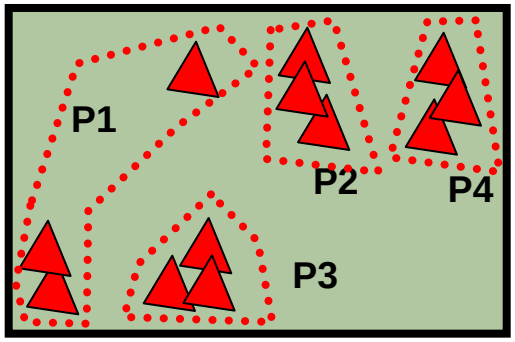
Vorteile von Sort-Middle

- Last der Geometrietransformation kann durch eine gleichmäßige Verteilung der Geometrie über die Prozessoren gut balanciert werden

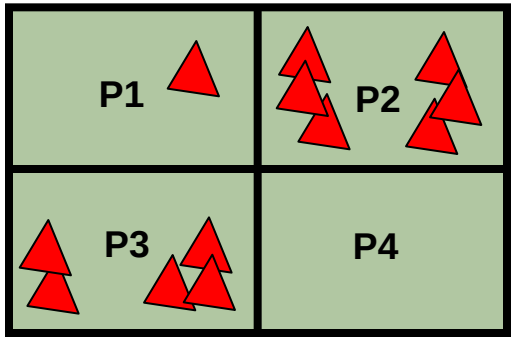
Probleme von Sort-Middle

- Hardware muss Lesen der Daten zwischen Geometrietransformation und Rasterung erlauben
- daher im PC-Cluster-System mit heutiger Standardgrafikhardware schwierig
- bei der Rasterung dieselben Balancierungsprobleme wie bei Sort-First

Geometrietransformation gleichmäßig verteilt → gute Balancierung



danach Verteilung der projizierten Polygone im 2D-Bildraum

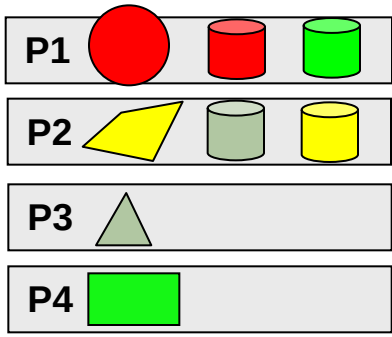


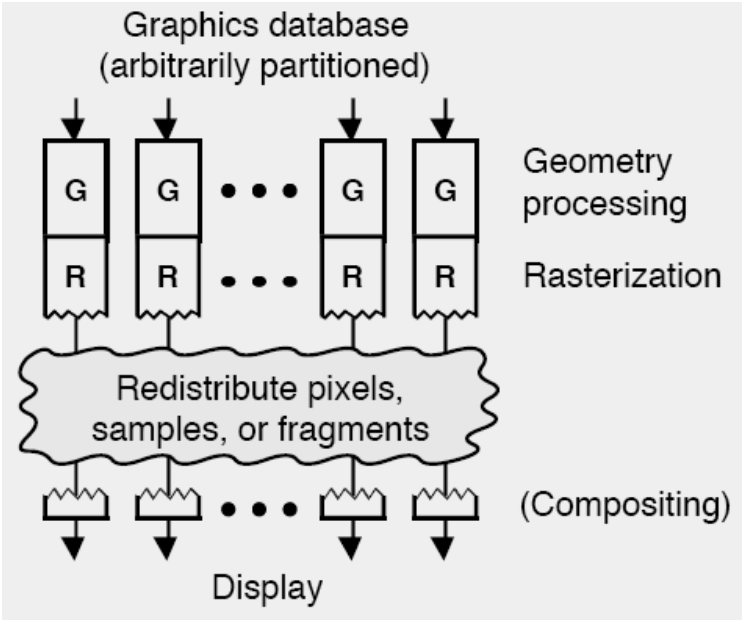
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Sort-Last

Ausgangssituation

- vor der Bildberechnung sind die Objekte beliebig auf den Prozessoren verteilt



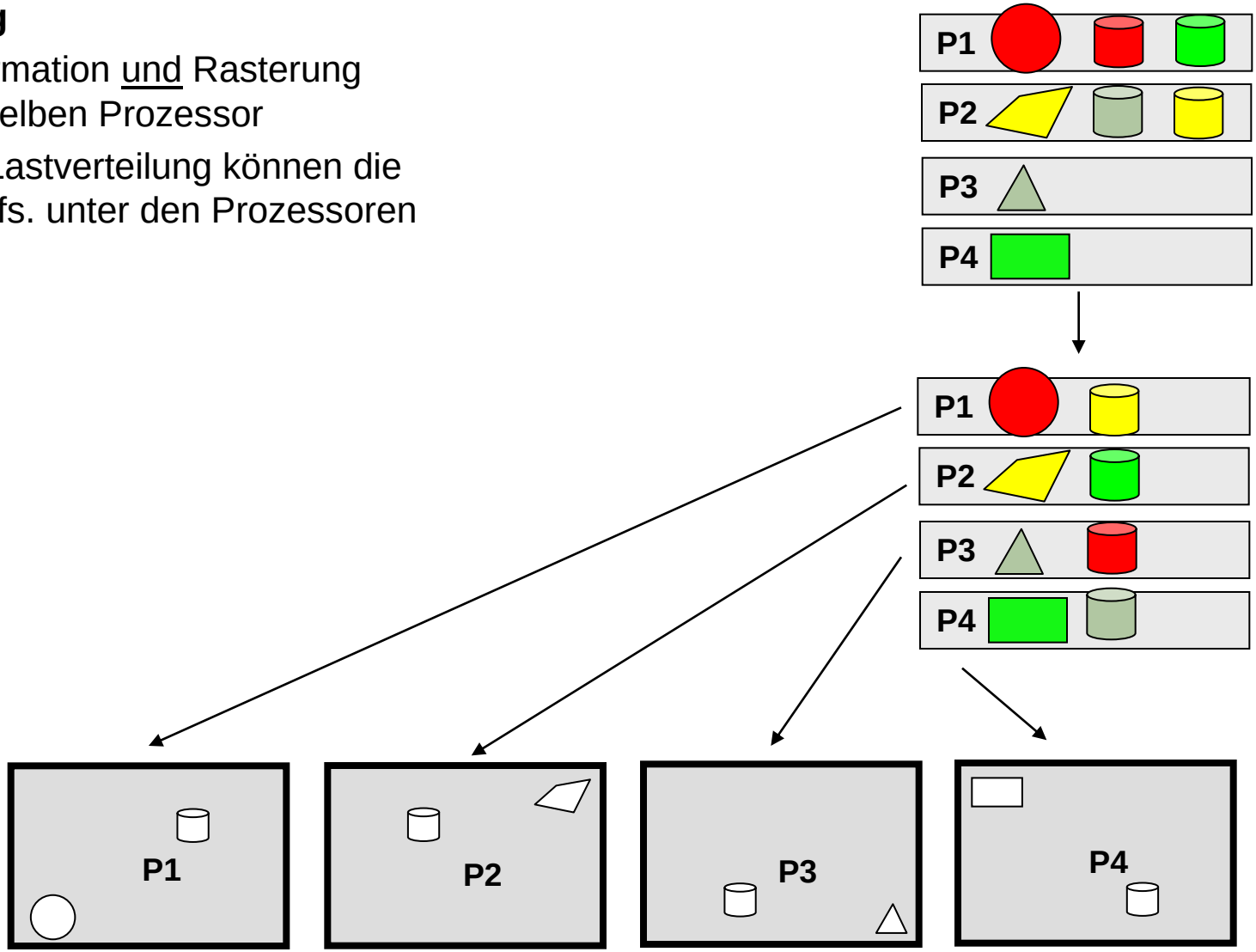


Paralleles Rendering als Sortierproblem

Sort-Last

Schritt 1: Rendering

- Geometrietransformation und Rasterung erfolgen auf demselben Prozessor
- zum Zwecke der Lastverteilung können die Objekte vorher ggfs. unter den Prozessoren umverteilt werden

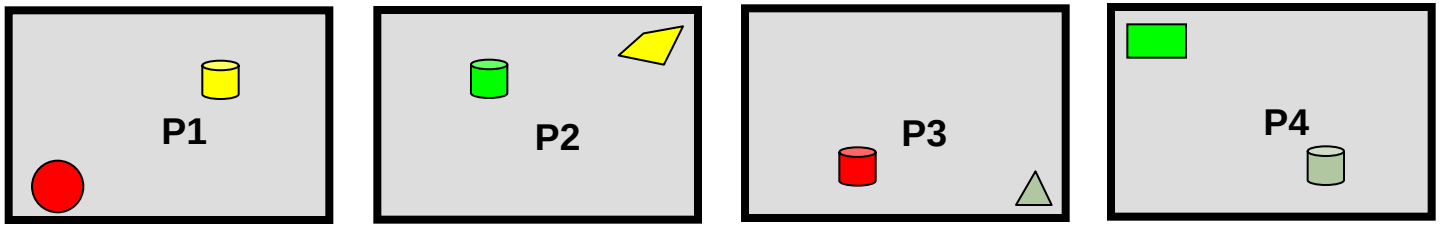


Paralleles Rendering als Sortierproblem
Sort-Last

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Was berechnet jeder einzelne Prozessor?

- es entstehen bei p Prozessoren p einzelne Bilder
- im Gegensatz zu Sort-First / Sort-Last sind die Teilbilder keine Kachelung des Gesamtbildes!
- jeder Prozessor ist für die gesamte Bildschirmauflösung zuständig
- vergleiche Sort-First / Sort-Last:
  die Summe der Kacheln ist die gesamte Bildschirmauflösung!

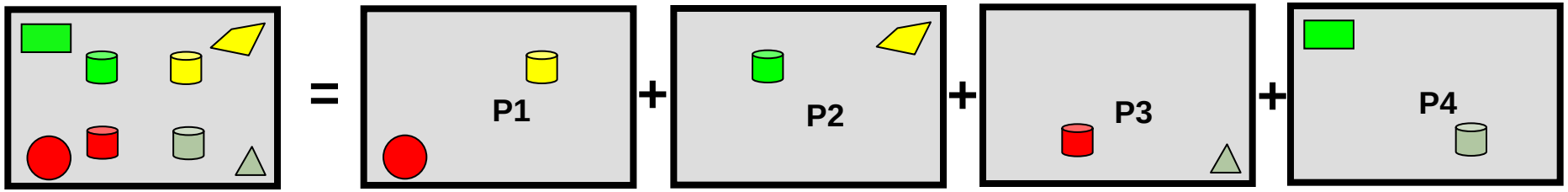


Schritt 2: Verteilung - „Sort“

- Verteilung (Sortierung) erfolgt nach der Rasterung
- verteilt (= in die richtige Reihenfolge bringen) werden die Pixelfragmente des 2D-Bildraums

Schritt 3: Darstellung des Bildes

- Zusammensetzung der einzelnen Pixel-Fragmente
- Tiefenwerte müssen bekannt sein, da sich Teile der Polygone überlappen können und die Sichtbarkeit unter den Teilbildern zu bestimmen ist

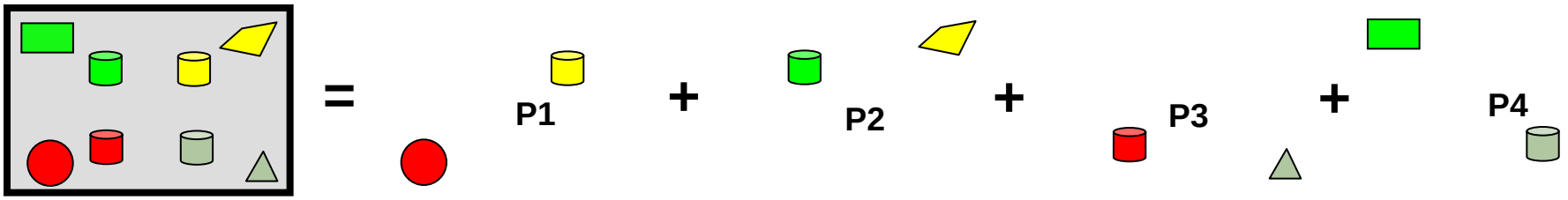


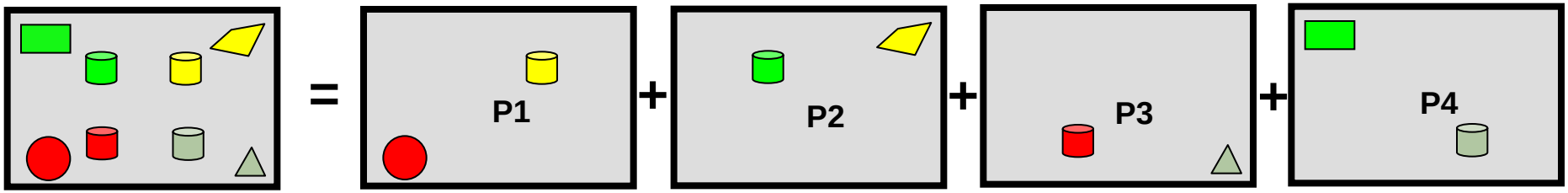
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Sort-Last: „2 Varianten“

Wie speichert jeder Prozessor seine gerenderten Pixelfragmente?

- SL-sparse: wir speichern nur die tatsächlich gerasterten Pixel
  - geringe Netzwerklast bei wenigen kleinen Objekten
  - problematisch bei vielen Objekten
- SL-full: wir speichern immer die vollständige Bildschirmauflösung
  - hohe Netzwerklast





HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Hybrides Sort-First /Sort-Last Rendering

- Motivation
- Idee
- Architektur
- 3-Phasen-Pipeline
  - Partitionierung
  - Rendering
  - Bildzusammensetzung
- Partitionierungs-Algorithmus
- Kommunikations-Overhead

Literatur

- Tomas Akenine-Möller, Eric Haines
  Real-Time Rendering
  AK Peters, 2002
- Dirk Bartz, Claudio Silva
  Rendering and visualization in parallel environments
  Tutorial of EUROGRAPHICS 2001, Tutorial 9. Eurographics Association, 1995
  http://www.gris.uni-tuebingen.de/~bartz/tutorials/eg2001tutorial
- E. Reinhard, A. G. Chalmers, F. W. Jansen
  Overview of parallel photo-realistic graphics
  STAR - State of the Art Reports, EUROGRAPHICS 1998, pages 1–25. Eurographics Association, 1998
  http://www.cs.bris.ac.uk/Publications/Papers/1000271.pdf
- Rudrajit Samanta, Thomas Funkhouser, Kai Li, Jaswinder Pal Singh
  Hybrid Sort-First and Sort-Last Parallel Rendering with a Cluster of PCs
  Proc. ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware, p. 97-108, 2000
  http://doi.acm.org/10.1145/346876.348237

Hybrides Sort-First /Sort-Last Rendering
Motivation

Es wird ein paralleles Rendering-System aus PC-Komponenten aufgebaut.

Randbedingungen durch Hardwarevorgaben

- jeder Prozessorknoten besteht aus einem PC
- es werden handelsübliche Grafikkarten von PCs verwendet
- PCs sind durch ein Netzwerk miteinander verbunden
- keine beschleunigte Zusammenführung der Bilder durch Hardware!

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Sort-Middle

Ist für ein PC-Cluster-System schwer umzusetzen

- Standardgrafikkarten unterstützen meistens nur den indirekten Zugriff auf die Ergebnisse der Geometrietransformationen.
- Schnelles Netzwerk notwendig, da viele Primitive umverteilt werden müssen.

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Sort-First

Vorteil:

- geringer Kommunikations-Overhead

Nachteil:

- zusätzliche Transformation der Grafik-Primitive für die Verteilung
- redundantes Rendering durch Überlappungen der Buckets
- da das Ausmaß der Überlappung mit der Anzahl der Prozessoren steigt (Kacheln werden immer kleiner), skaliert Sort-First schlecht

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Sort-Last

Vorteil:

- keine Überlappungsprobleme durch die Buckets
- daher sehr gut mit der Anzahl der Prozessoren skalierbar

Nachteil:

- Neben Farbwerten müssen auch Tiefenwerte verschickt werden
- schnelles Netzwerk notwendig

Hybrides Sort-First /Sort-Last Rendering
Idee

Ansatz

- Hybrides paralleles Rendering
- Kombination aus „Sort-First“ und „Sort-Last“

Der Partionierungs-Algorithmus (Heuristik)

- ist sichtpunktabhängig
- partitioniert dynamisch zur Laufzeit

Hybrides Sort-First /Sort-Last Rendering
Idee

Was wird partitioniert?

- der 2D-Bildraum in Buckets
- die 3D-Polygone in Gruppen

Warum wird partitioniert?

- um die Last der Prozessoren zu balancieren
- um die Kommunikationskosten für das Verschicken von überlappenden Buckets zu minimieren

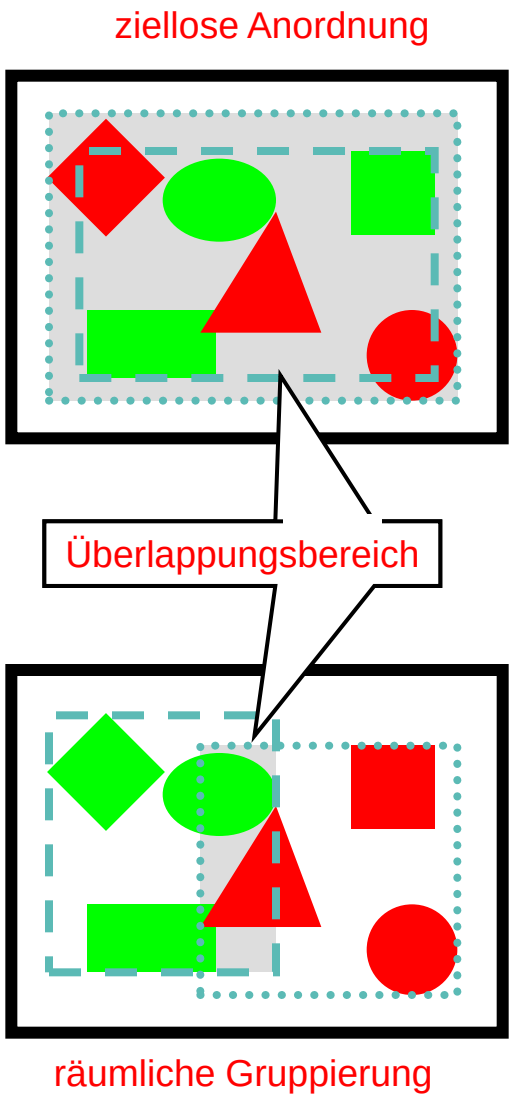
Hybrides Sort-First /Sort-Last Rendering
Idee

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Grundlegende Idee Partitionierungs-Algorithmus

- gruppiere 3D-Polygone in Gruppen
- die 2D-Boundingboxen der Gruppen im Bildraum bilden ein Bucket
- die Buckets überlappen sich
- wähle die Gruppen so, dass die sich überlappenden Bereiche der Buckets möglichst klein sind

Die Größe der überlappenden Bereiche ist entscheidend für die Höhe der Kommunikationskosten!



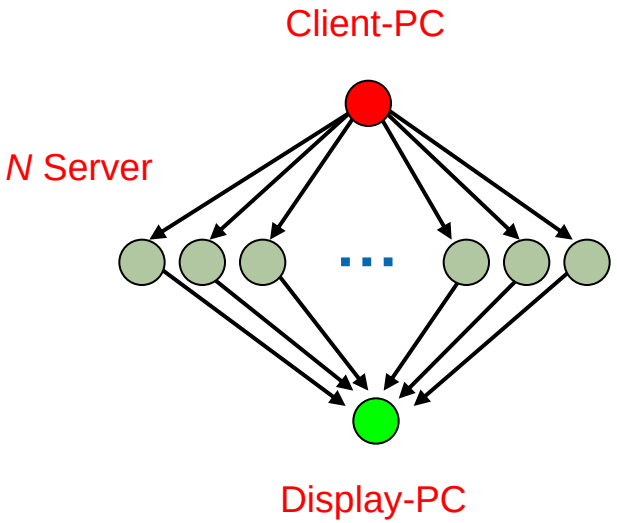
Hybrides Sort-First /Sort-Last Rendering
Architektur

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Architektur und Algorithmus

- der Algorithmus arbeitet in drei Phasen
- jede Phase wird von entsprechenden PCs bearbeitet
- das System arbeitet ein Art als „3-Phasen Pipeline“

Phase 1: ein Client-PC
Phase 2: N Server
Phase 3: ein Display-PC



Hybrides Sort-First /Sort-Last Rendering
Partitionierung

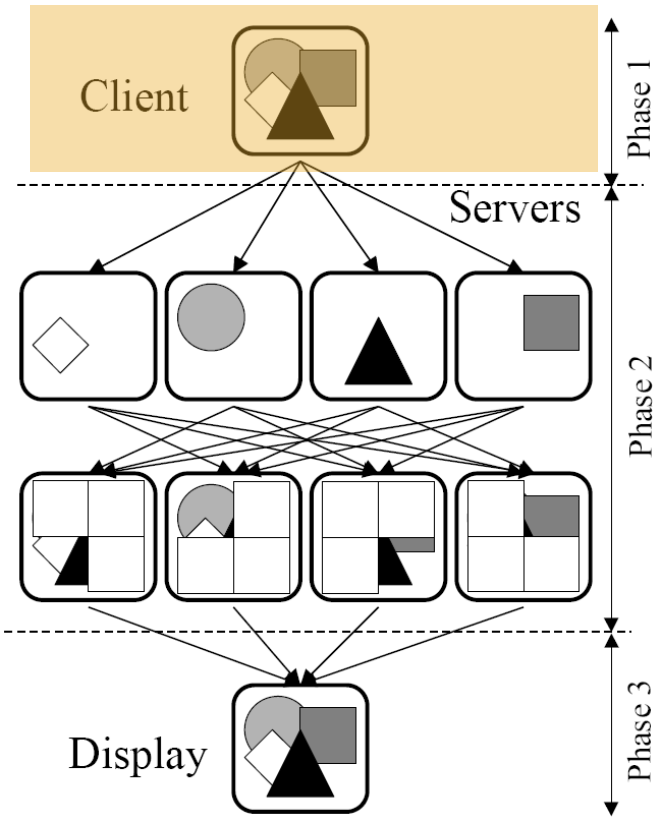
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

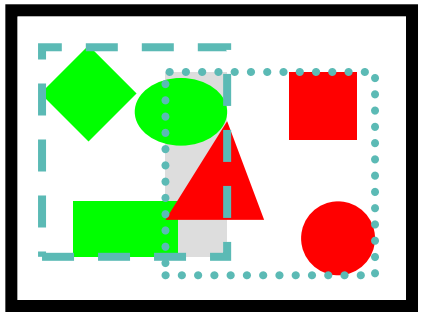
Phase 1: Partitionierung auf dem Client-PC

Ausführung des Partitionierungs-Algorithmus

Partitionierung im Objektraum

- teile die 3D-Polygone in N disjunkte Gruppen auf
- weise jede Gruppe einem Server zu





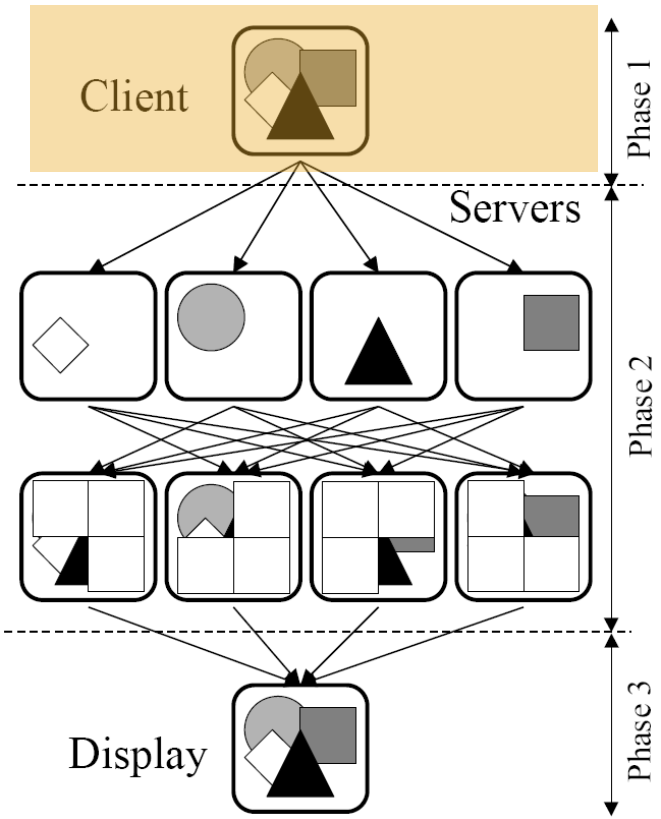
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

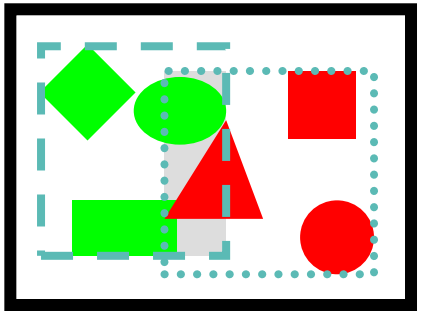
Phase 1: Partitionierung auf dem Client-PC

Ausführung des Partitionierungs-Algorithmus

Partitionierung im Bildraum

- berechne die Überlappungen der Buckets
- partitioniere die Pixel in disjunkte Überlappungsbereiche, die aus sich überlappenden und nicht überlappenden Teilen von Buckets bestehen
- wähle Server B aus, die für die Berechnung der „Überlappungsbereiche“ zuständig sind





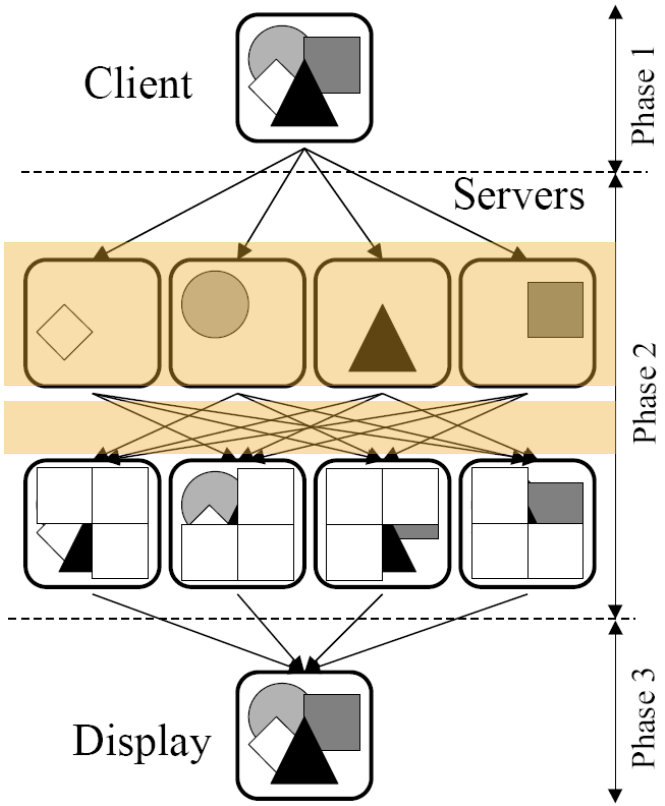
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

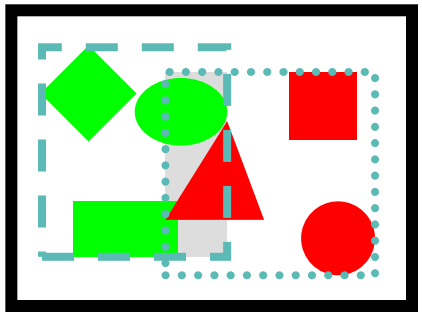
Phase 2: Rendering auf den N Servern

Rendering der Gruppen

- jeder Server A rendert die ihm zugewiesene Gruppe in den lokalen Frame-Buffer
- „read-back“: lese Farb- und Tiefenwerte aller Pixel, die in überlappenden Bereichen der Buckets liegen

- verschicke alle überlappenden Bereiche, für die ein anderer Server B zuständig ist, an den Server B





HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Phase 2: Rendering auf den N Servern

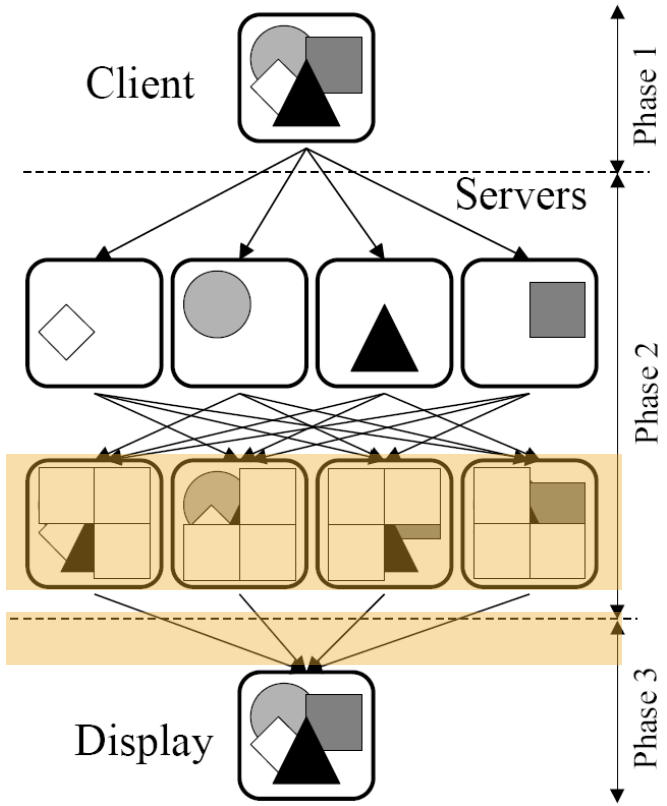
Rendering der Überlappungsbereiche

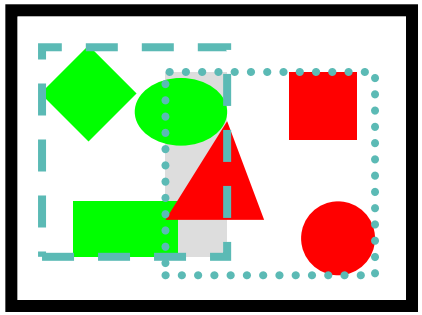
Schritt 1:

- die Server B empfangen von allen Servern A Teile der Überlappungsbereiche
- setze die Überlappungsbereiche zu einem Teilbild zusammen
- verwende dazu Farb- und Tiefenwerte mit dem lokalen Frame-Buffer

Schritt 2:

- „read-back“: lese die Pixel aus allen Überlappungsbereichen
- verschicke alle Überlappungsbereiche und die Pixel des sich nicht überlappenden Teils der Buckets zum Display-PC

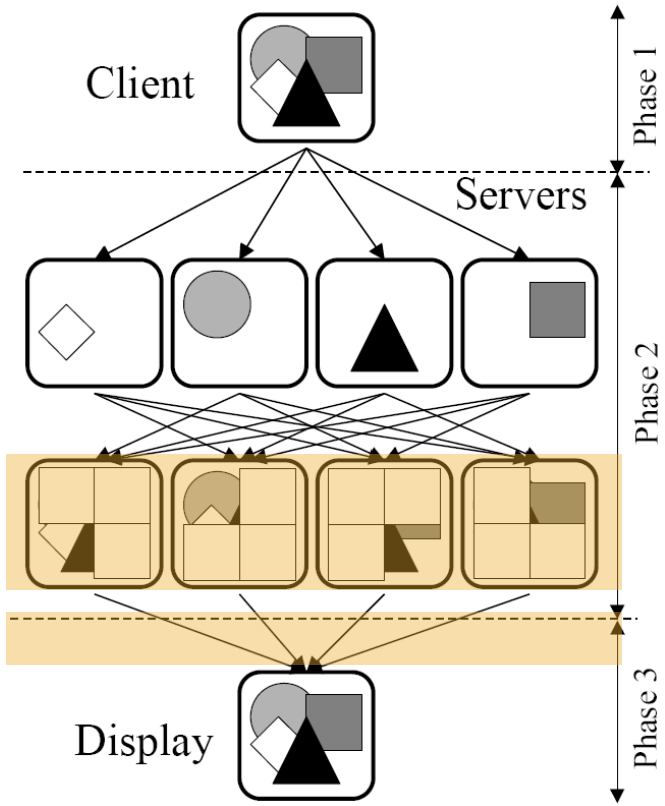


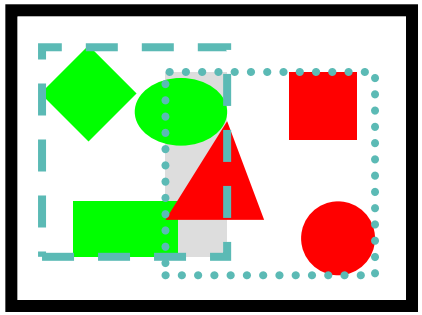


HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Phase 3:
Bildzusammensetzung auf dem Display-PC

- empfange von allen Servern Teilbilder
- setze die Überlappungsbereiche zu einem Bild zusammen
- Tiefentest ist hier nicht mehr notwendig!



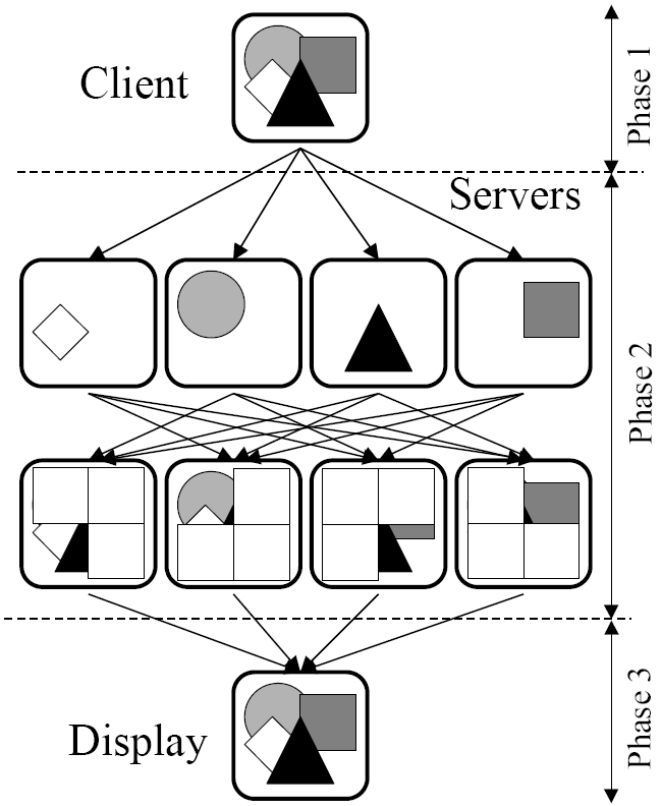


HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Display-PC

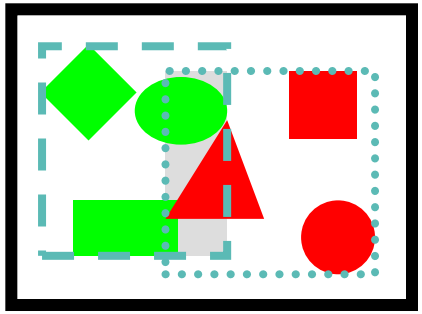
- empfängt in der Bildschirmauflösung jedes Pixel nur genau einmal
- kein Empfang von Tiefenwerten notwendig

Vergleiche dazu die Nachteile von Sort-Last- Ansätzen.



HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Anforderungen an den Partitionierungs-Algorithmus



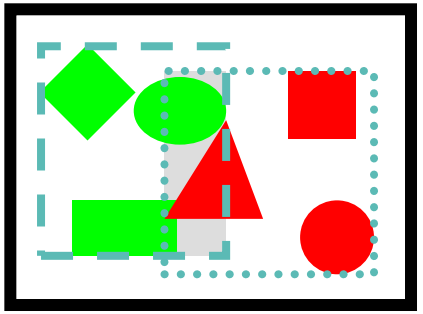
- Balancierung der Rendering-Last.
- Minimierung des Overheads durch Umverteilung der Pixel in den Überlappungsbereichen.
  (Kommunikation und Berechnungen!)
- Berechnung muss so schnell erfolgen,
  dass der Client nicht zum Bottleneck wird.

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Partitionierungs-Algorithmus

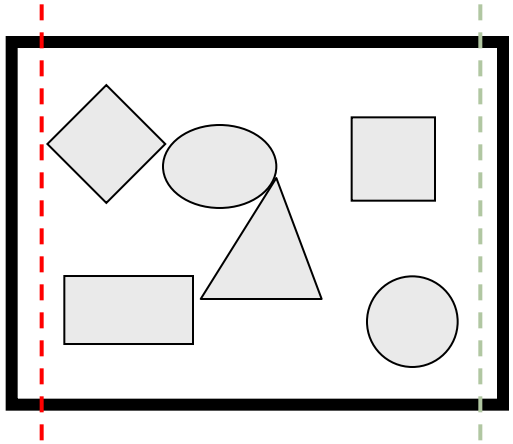
Wir wollen

- zum balancieren Gruppen bilden,
- aber gleichzeitig die Überlappungsbereiche der Buckets klein halten.



Ansatz

- Bilde rekursiv eine binäre Aufteilung der Gruppen im 3D-Raum,
- verwende dazu zwei Sweepline-Geraden, die im 2D-Bildraum aufeinander zu laufen.



Hybrides Sort-First /Sort-Last Rendering
Partitionierungs-Algorithmus

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

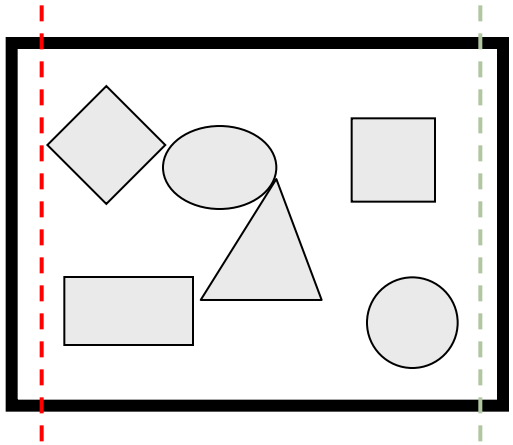
Partitionierungs-Algorithmus

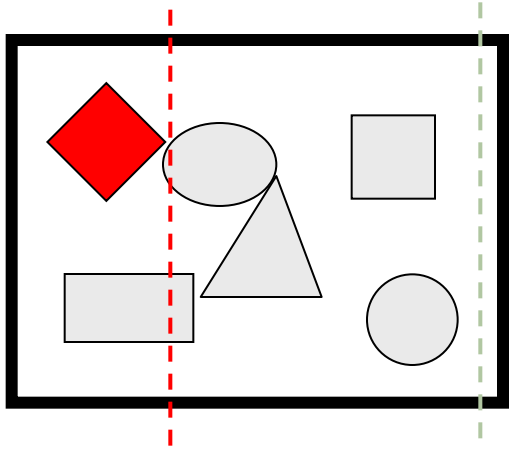
- die linke Sweep-Line läuft immer nach rechts
- die rechte Sweep-Line läuft immer nach links

- sobald ein Objekt vollständig von der Sweep-Line überlaufen wurde, kommt es zur Gruppe

- Objekte, die von der linken (rechten) Sweep-Line überlaufen wurden, kommen zur linken (rechten) Gruppe

- die Sweep-Lines laufen so lange, bis alle Objekte verteilt wurden





Hybrides Sort-First /Sort-Last Rendering
Partitionierungs-Algorithmus

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

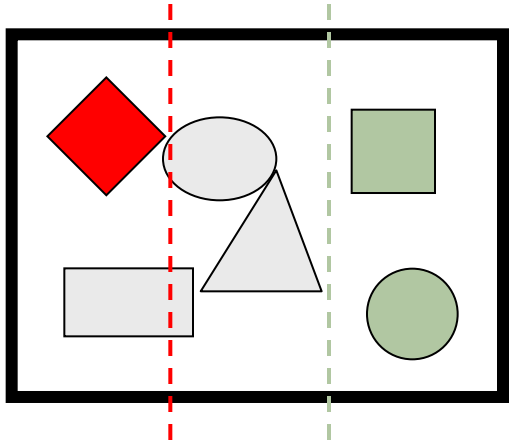
Partitionierungs-Algorithmus

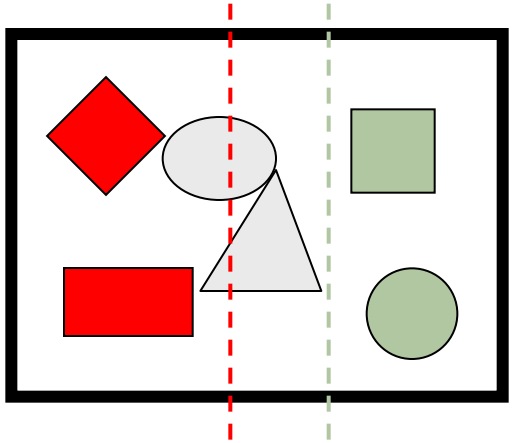
- die linke Sweep-Line läuft immer nach rechts
- die rechte Sweep-Line läuft immer nach links

- sobald ein Objekt vollständig von der Sweep-Line überlaufen wurde, kommt es zur Gruppe

- Objekte, die von der linken (rechten) Sweep-Line überlaufen wurden, kommen zur linken (rechten) Gruppe

- die Sweep-Lines laufen so lange, bis alle Objekte verteilt wurden



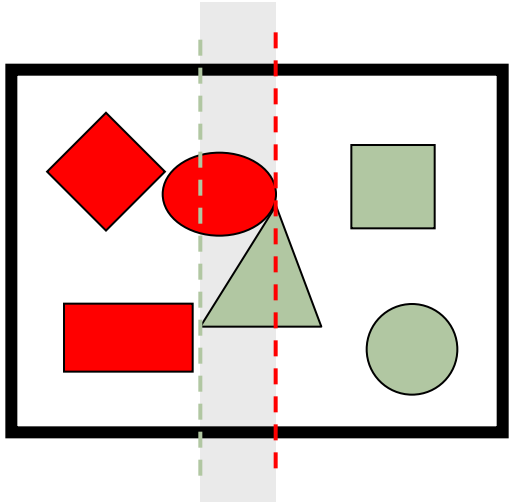


Hybrides Sort-First /Sort-Last Rendering
Partitionierungs-Algorithmus

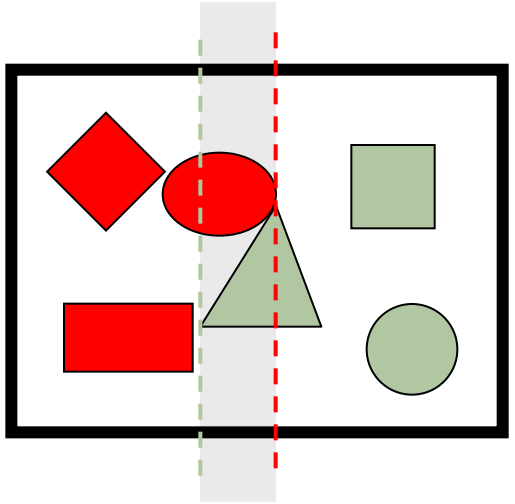
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Partitionierungs-Algorithmus

- ….
- die Sweep-Lines laufen so lange, bis alle Objekte verteilt wurden
- wenn alle Objekte zugeordnet wurden und die Sweep-Lines aneinander vorbei gelaufen sind, dann entstehen die Überlappungsbereiche der Buckets



HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität



Partitionierungs-Algorithmus

Wie erfolgt der rekursive Schritt?

- wir teilen im Bildraum immer senkrecht zur längsten Achse (Greedy)
- nach einer Aufteilung in zwei Gruppen teilen wir rekursiv weiter
- wir beenden die Aufteilungen, bis N gleichmäßige Gruppen für die N Server entstanden sind
- jede Gruppe sollte die gleiche Rendering-Last erzeugen

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Kommunikations-Overhead

- ein Teil des Kommunikations-Overhead entsteht durch das Verschicken der überlappenden Bereiche der Buckets
- wie können wir den Kommunikations-Overhead quantifizieren?

Eine exakte Berechnung ist schwierig, da die Breite der Überlappung schwer abzuschätzen ist.

Die Breite der Überlappung hängt ab von:

- Aufbau und Struktur der 3D-Szene
- aktuellem Standpunkt

HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Überlappende Bereiche der Buckets

Wir setzen vereinfachende Annahmen voraus:

- alle Objekte sind gleich verteilt
- alle Objekte sind im Bildraum (nach Transformation) Quadrate gleicher Größe (Kantenlänge sei B)
- jeder Server bekommt ein Bucket mit $\frac{P}{N}$ Pixeln und der Kantenlänge $\frac{\sqrt{P}}{\sqrt{N}}$ (Bildraum besteht aus P Pixeln, es gibt N Server)

Hybrides Sort-First /Sort-Last Rendering
Kommunikations-Overhead

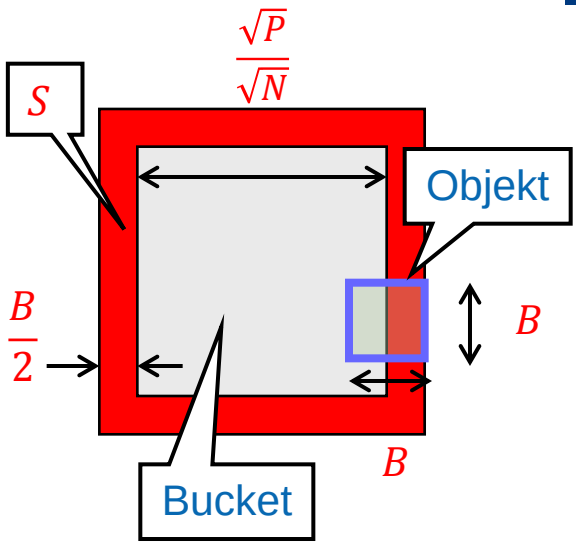
HEINZ NIXDORF INSTITUT
Universität Paderborn
Algorithmen und Komplexität

Wir wollen die Bereiche (S) der Buckets bestimmen, an denen sich die gerenderten Pixel überlappen

- wir betrachten ein Bucket
- der Bereich S ist ein „Streifen“ der Breite $\frac{B}{2}$ um das Bucket

Für den Kommunikationsaufwand gilt:

- jeder Server schickt den Bereich S zu anderen Servern
- oder erhält Pixel derselben Größe von anderen Servern



Hybrides Sort-First /Sort-Last Rendering

Kommunikations-Overhead

Die Größe des überlappenden Bereichs S setzt sich zusammen aus

- den vier Ecken, jeweils $\frac{B}{2} \cdot \frac{B}{2}$
- den vier Rechtecken an den Rändern des Buckets, jeweils $\frac{B}{2} \cdot \frac{\sqrt{P}}{\sqrt{N}}$

Zusammen ergibt sich S zu

$$S = 4 \cdot \frac{B}{2} \cdot \frac{\sqrt{P}}{\sqrt{N}} + 4 \cdot \frac{B}{2} \cdot \frac{B}{2}$$
$$= 2B \cdot \frac{\sqrt{P}}{\sqrt{N}} + B^2$$

Diese Anzahl Pixel muss jeder Server in jedem Frame an andere Server schicken!