Architektur und Programmierung von Grafik- und Koprozessoren

Rendering Algorithmen

Stefan Zellmann

Lehrstuhl für Informatik, Universität zu Köln

SS2018

Sort-Last Compositing Algorithmen

Baumbasiertes / Rundenbasiertes Compositing

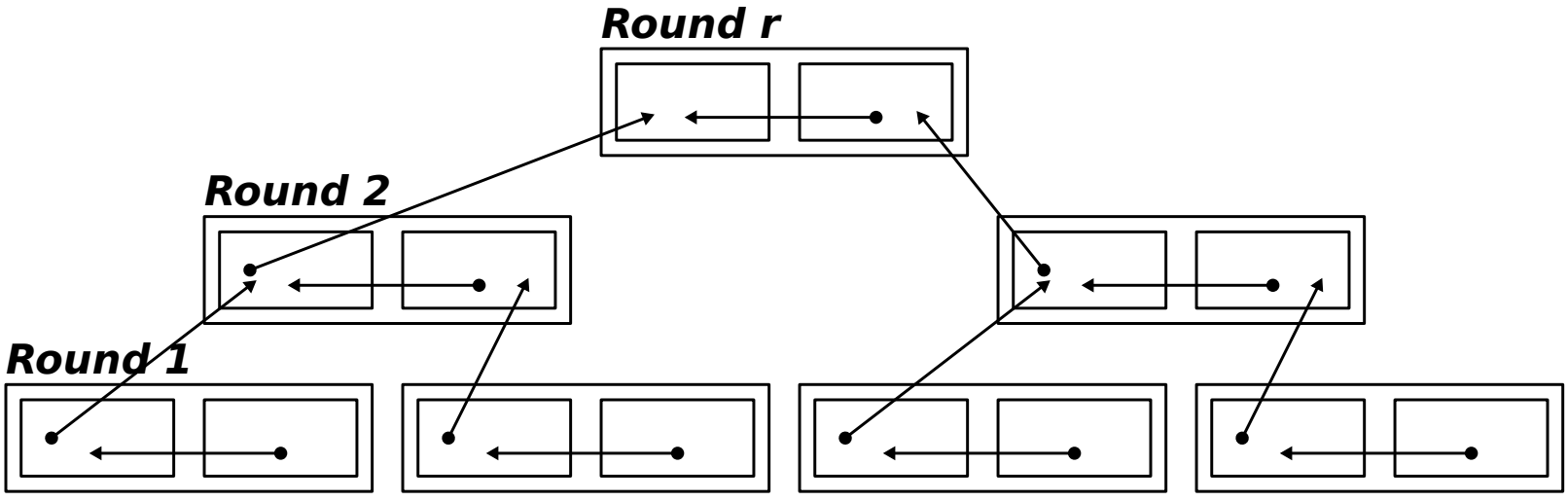


Abbildung: Problem bei naiv implementiertem rundenbasiertem Compositing (und generell bei binärbaumbasierten parallelen Algorithmen wie Reduce): Viele Prozessoren, die in frühen Runden arbeiten, sind in späteren Runden nicht beschäftigt.

Sort-Last Compositing Algorithmen

Binary-Swap

Binary-Swap Compositing löst dieses Problem,

- indem pro Runde zwei Prozessoren paarweise für eine Hälfte ihres *gemeinsamen* Bildbereichs zuständig sind (*binary*),
- und indem diese die Bilddaten, für die jeweils der andere Prozessor zuständig ist, in jeder Runde tauschen (*swap*).

In jeder Runde wird zudem der Bildausschnitt, für den zwei Prozessoren zuständig sind, halbiert. Außerdem wird die Distanz zwischen den Prozessorpaaren verdoppelt ("distance-doubling and vector-halving" Algorithmus).

Es sind stets P Prozessoren beschäftigt, wobei $P = 2^k$.

Ist der Binary-Swap Algorithmus durchgelaufen, liegt das zusammengesetzte Bild *verteilt* auf P Prozessoren vor.

Sort-Last Compositing Algorithmen

Binary-Swap

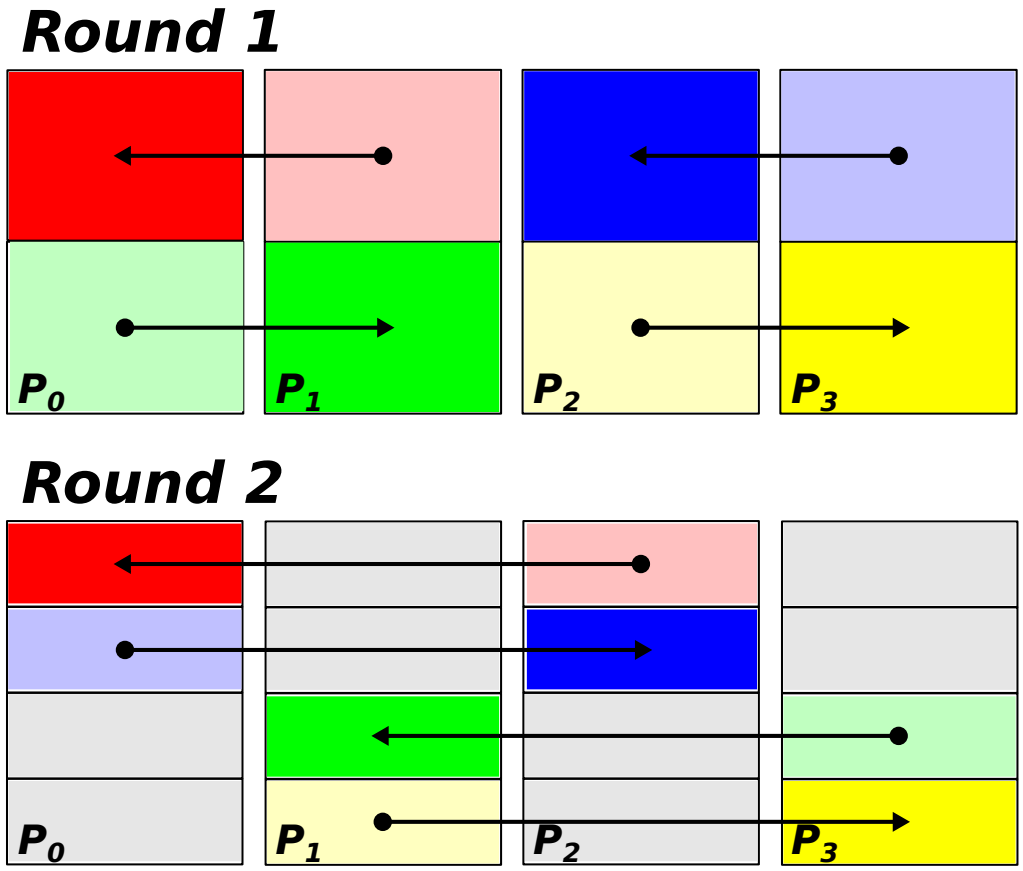


Abbildung: Kommunikationsmuster beim Binary-Swap Compositing mit vier Prozessoren, vgl. Bethel et al.: High Performance Visualization - Enabling Extreme Scale Scientific Insight (2013).

Sort-Last Compositing Algorithmen

Binary-Swap

- Bemerkung: in jeder Runde führen Paare von Prozessoren Direct-Send durch.
- Bemerkung: der Binary-Swap Algorithmus skaliert nur für $P = 2^k$ Prozessoren.

Sort-Last Compositing Algorithmen

Verallgemeinerung

Wir wollen die Algorithmen *Direct-Send* und *Binary-Swap* verallgemeinern. Dazu führen wir die folgende Notation ein.

Bezeichne r die Anzahl Runden, die der Compositing Algorithmus durchführt. Bezeichne k_i die Anzahl an Prozessoren, die in jeder Runde in jeder Kommunikationsgruppe beteiligt ist. Es ergibt sich der "k-Vektor" $\vec{k} = [k_1, k_2, ..., k_r]$.

Sort-Last Compositing Algorithmen

Verallgemeinerung

Sei P die Anzahl aller Prozessoren. Dann ergibt sich z. B.

Direct-Send:
$r = 1,\ \vec{k} = [P]$.

Rundenbasiert (naiv & Binary-Swap):
$r = log(P),\ \vec{k} = [2, 2, 2, ...]$.

Sort-Last Compositing Algorithmen

Radix-k

Frage: sind andere Kombinationen von r und $\vec{k}$ möglich?

Antwort: ja. Der *Radix-k* Compositing Algorithmus erlaubt jede Faktorisierung von P, sodass

$$\prod_{i=1}^{r} k_i = P, \tag{43}$$

wobei in jeder Runde i alle Kommunikationsgruppen die gleiche Größe k_i haben. In jeder Kommunikationsgruppe selbst wird *Direct-Send* durchgeführt.

Sort-Last Compositing Algorithmen

Radix-k

Sei etwa $P = 12$. Mögliche gültige Konfigurationen für den Radix-k Algorithmus sind z. B.:

$r = 1 : \vec{k} = [12]$
$r = 2 : \vec{k} = [6, 2], \vec{k} = [2, 6], \vec{k} = [3, 4], \vec{k} = [4, 3]$
$r = 3 : \vec{k} = [2, 2, 3], ...$
$r = ...$

Sort-Last Compositing Algorithmen

Radix-k

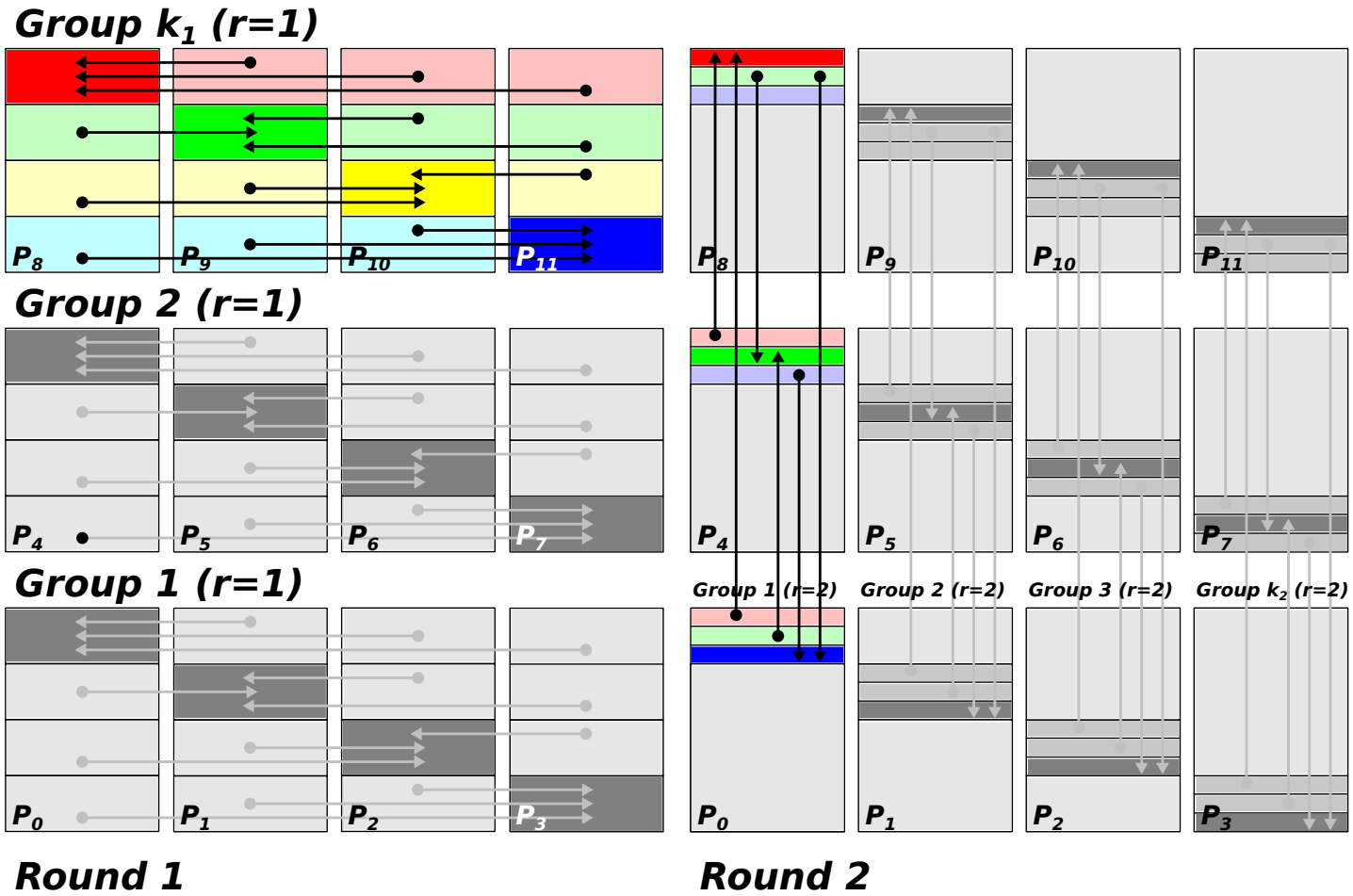


Abbildung: $P = 12, r = 2, \vec{k} = [4, 3]$, vgl. Bethel et al.: High Performance Visualization - Enabling Extreme Scale Scientific Insight (2013).

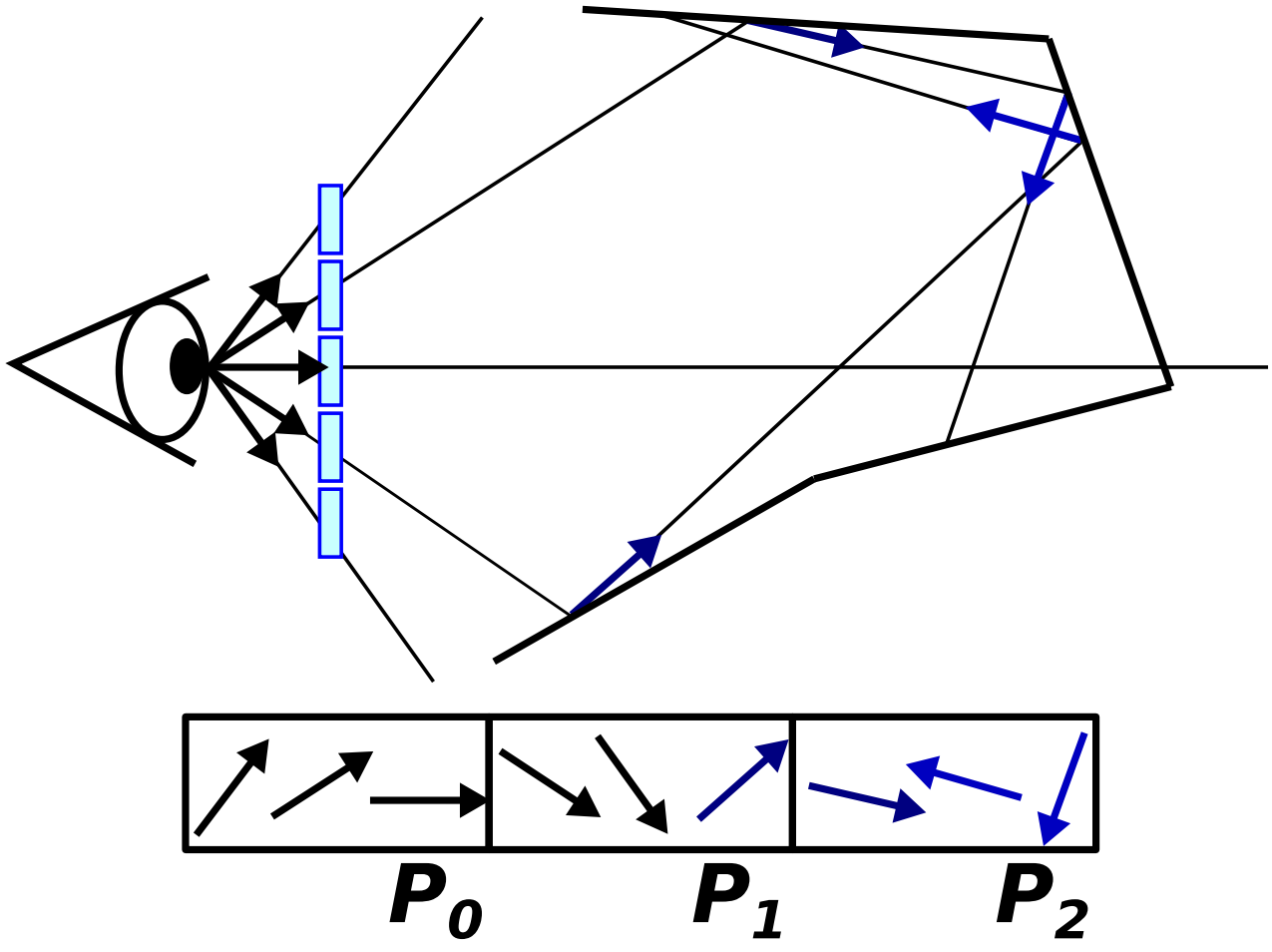
Sort-Last Compositing Algorithmen

Radix-k

Bemerkung: sowohl Direct-Send als auch Binary-Swap sind gültige Radix-k Konfigurationen (naiv rundenbasiertes Compositing nicht, da innerhalb der Kommunikationsgruppen kein Direct-Send).

Andere Parallelisierungsarten

Strahlverfolgung mit Wavefronts



Andere Parallelisierungsarten

Strahlverfolgung mit Wavefronts

- Halte mit jedem Bounce die gesamte Pipeline an.
- Verteile alle Strahlen für nächsten Bounce auf P Prozessoren.
- Vorher: sortiere ggf., z. B. nach Material-ID oder so, dass Strahlen mit ähnlichem Ursprung und ähnlicher Richtung nah beieinander.
- Kombiniere mit “compaction”: schiebe inaktive Strahlen “nach rechts” in der Liste, prozessiere nur den aktiven Teil der Liste, z. B. mit SIMD.
- Wavefronts dann besonders nützlich, wenn Strahlen sehr inkohäherent.

Andere Parallelisierungsarten

Distributed Shared Memory

- Verteiltes Speichersystem.
- Software Layer, das z. B. Cache Kohärenz Protokoll implementiert.
- Nachrichtenversand auf Basis von MPI.
- z. B. anwendbar bei Sort-First: Geometrie verteilt sich auf Speicher, dieser wirkt aber, als wäre er geteilt.

Recap (1)

- Lichttransportgleichung, Computergrafik Grundlagen: Scan Konvertierung, Texturierung, Tiefenpuffer, Alpha Blending.
- Grafik Pipeline mit Rasterisierung, Algorithmus $O(V \times VP \times L)$.
  - Einteilbar in mehrere parallele *Stages*:
  - Vertex Stage.
  - Rasterisierung.
  - Fragment Stage.
- Deferred Shading für viele Lichter: $O(V \times VP + L \times VP)$.
  - GPU Implementierung: speichere g-Buffer in fensterfüllenden Texturen.
  - Erhöhter Speicher- und Bandbreitenbedarf.

Recap (2)

- Strahlverfolgung: nicht dediziert in GPUs implementiert, aber auf GPUs parallel implementierbar.
- Flexibler: mehr Strahlen für höheren Realismus.
  Suchdatenstrukturen: Komplexität $O(log(V))$.
- Paralleles Rendering als Vorbereitung auf GPU Architekturen.

Literaturempfehlungen

- Peter Shirley, Steve Marschner: Fundamentals of Computer Graphics, 3rd ed. (2009)
- Matt Pharr, Wenzel Jakob, Greg Humphreys: Physically Based Rendering - From Theory to Implementation, 3rd ed. (2017)
- E. Wes Bethel, Hank Childs, Charles Hansen: High Performance Visualization - Enabling Extreme-Scale Scientific Insight, 1st ed. (2013)
- Juan Pineda: A Parallel Algorithm for Polygon Rasterization, Siggraph (1988)
- Turner Whitted: An Improved Illumination Model for Shaded Display, Communications of ACM (June 1980)