A Programmable Ray Processing Unit for Realtime Ray Tracing

vorgestellt auf der SIGGRAPH 2005
von Sven Woop, Jörg Schmittler, Philipp Slusallek

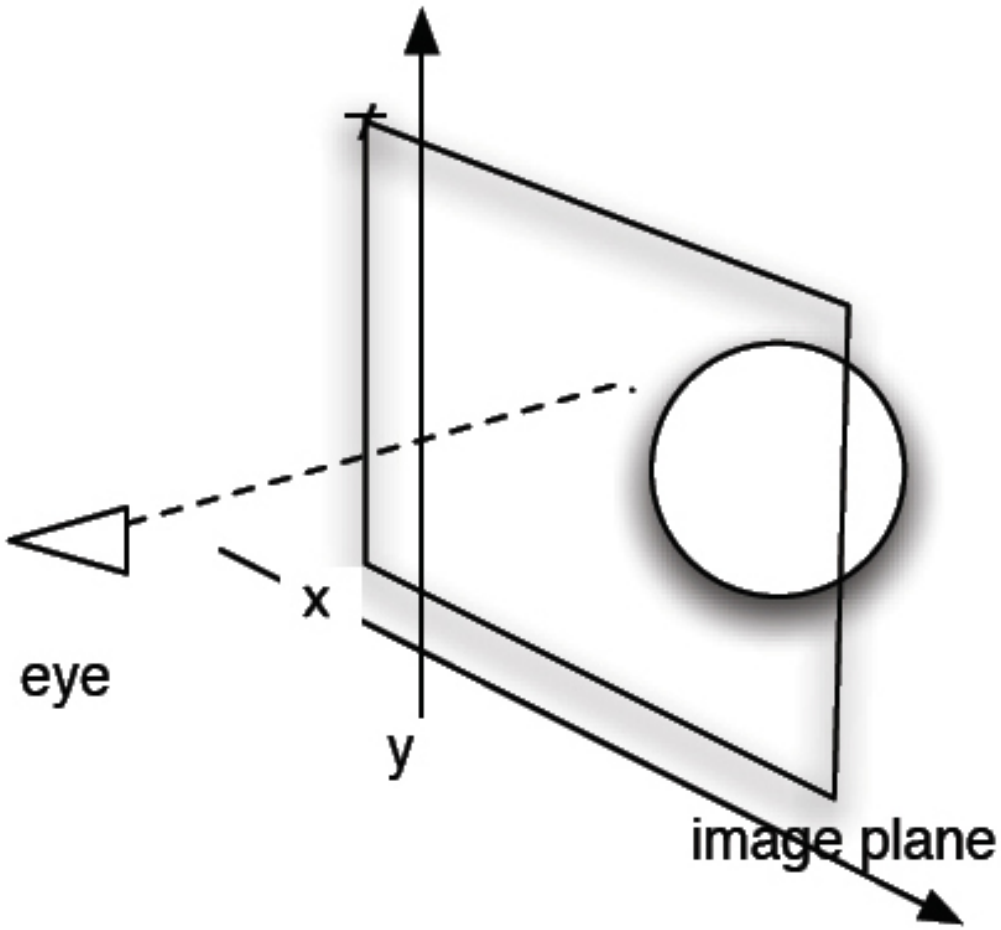
Stephan Kleber
Universität Ulm, Fakultät für Informatik
stephan.kleber@informatik.uni-ulm.de

15.11.2005 / Seminar Computergrafik

Übersicht

Gliederung

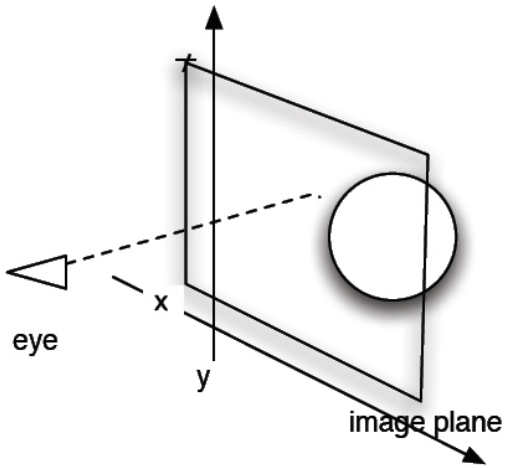
Prinzip des Raytracing I



Prinzip des Raytracing II

Gegeben

- Strahl
- Primitive
- Kameraposition
- Bildfläche



Zu Berechnen

- vom Strahl getroffene Primitive
- deren Oberfläche (Refraktion, Reflexion, Absorbtion)

Rasterung vs. Raytracing

Hardwarebasierte Rasterung

wird seit langem verwendet

und ist gut optimiert.

$$\Rightarrow \text{GPU}$$

Nachteile der Rasterung

nur lokale Berechnung

- keine globalen Effekte (Schatten, Reflexion, Transparenz, indirektes Licht)
- Emulation durch Texture- und Lightmaps, etc.

Argumente für Hardware-Raytracing

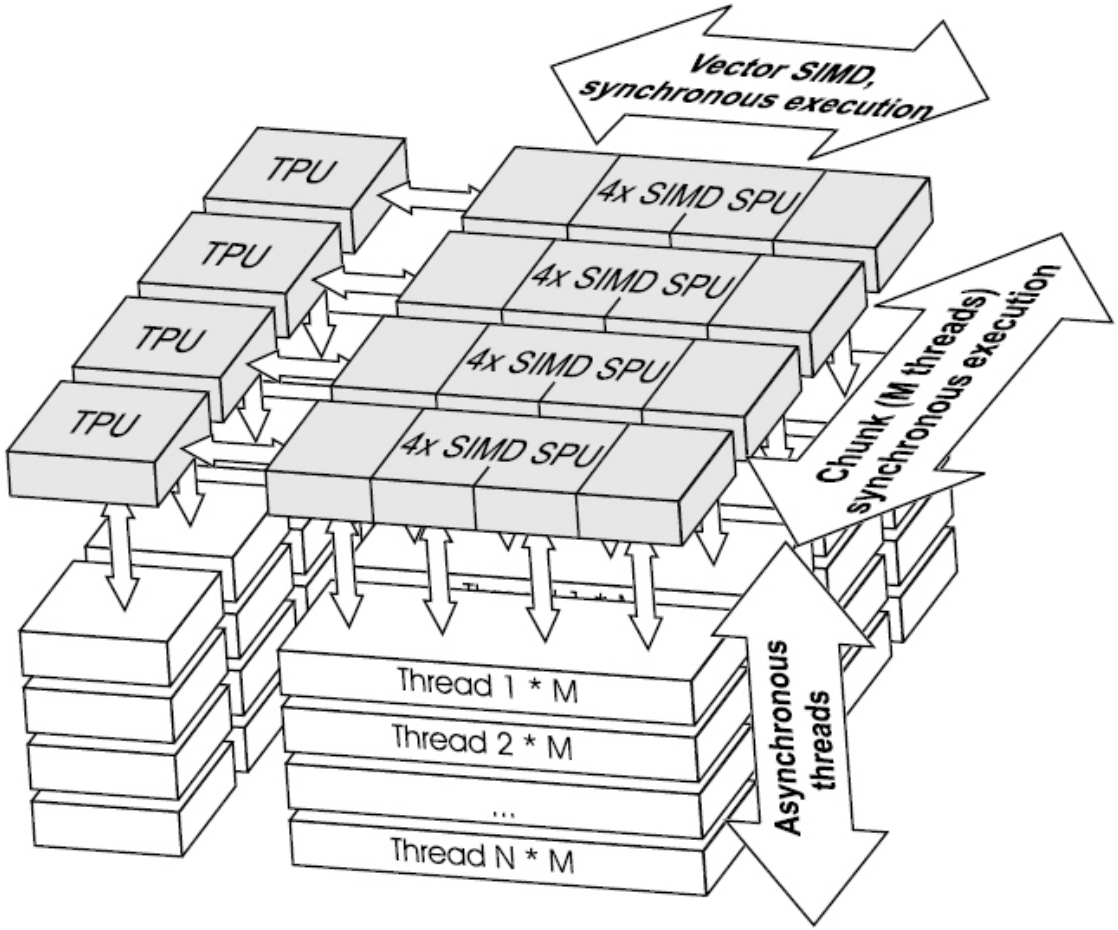
- Rendern in Echtzeit
- Softwareansatz zu langsam
- Softwareansatz zu aufwändig
- paralelle Berechnung möglich

Gliederung

Herausforderungen

- räumliche Indexstruktur
- flexibler Kontrollfluß
- komplexe Speicherzugriffsmuster
- paralelle Berechnungen

Lösungen


Vector SIMD, synchronous execution
TPU
TPU
TPU
TPU
4x SIMD SPU
4x SIMD SPU
4x SIMD SPU
4x SIMD SPU
Chunk (M threads) synchronous execution
Thread 1 * M
Thread 2 * M
...
Thread N * M
Asynchronous threads

Indexstruktur
TPU

Traversal Processing Unit

- Software-generierter *k*D-tree
- TPU traversiert *k*D-tree
- TPU wird von Shader aufgerufen

Kontrollfluß
SPU

Shader Processing Unit

- Bedingte Verzweigung (conditional braches)
- Rekursion (masked execution)

SPU besitzt einen Registerstack, der bei Sprungbefehlen implizit Registerwerte speichert und wiederherstellt.

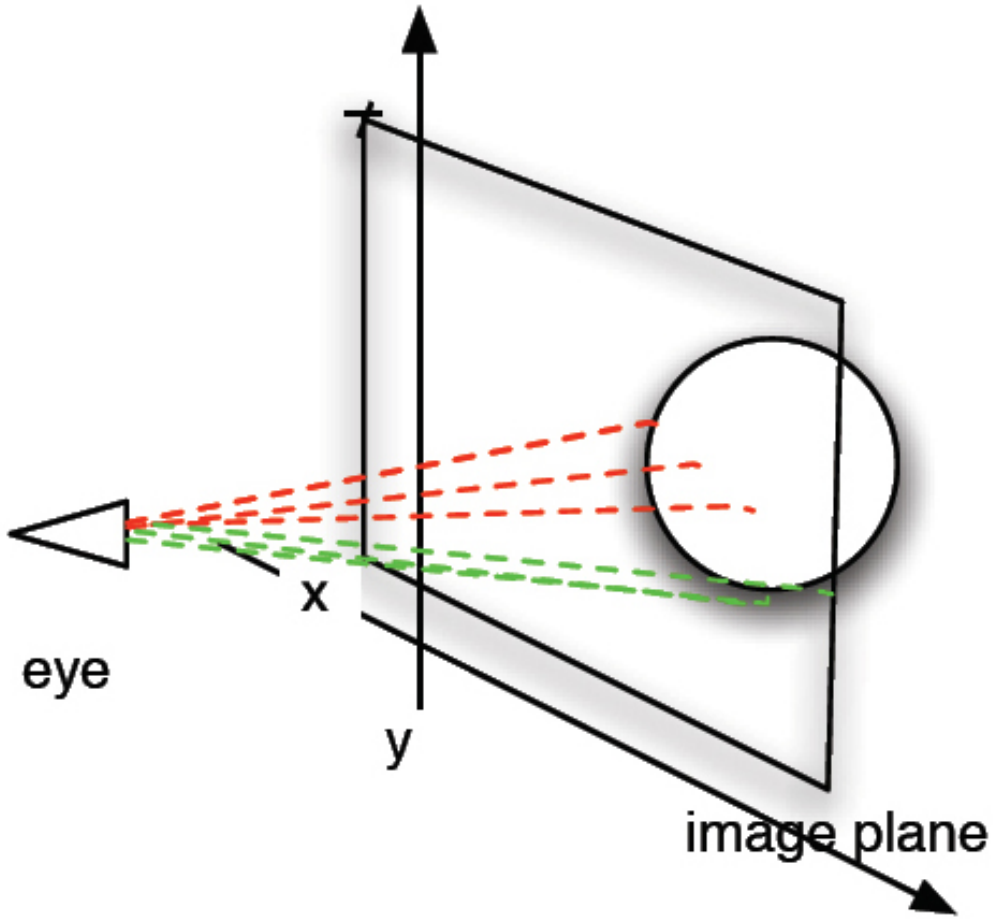
Speicherzugriffe I
SPU

typische Struktur von Szenendaten

- Kohärenz
- viele gleichartige Speicherzugriffe

$\Rightarrow$ Caching

Speicherzugriffe II
SPU

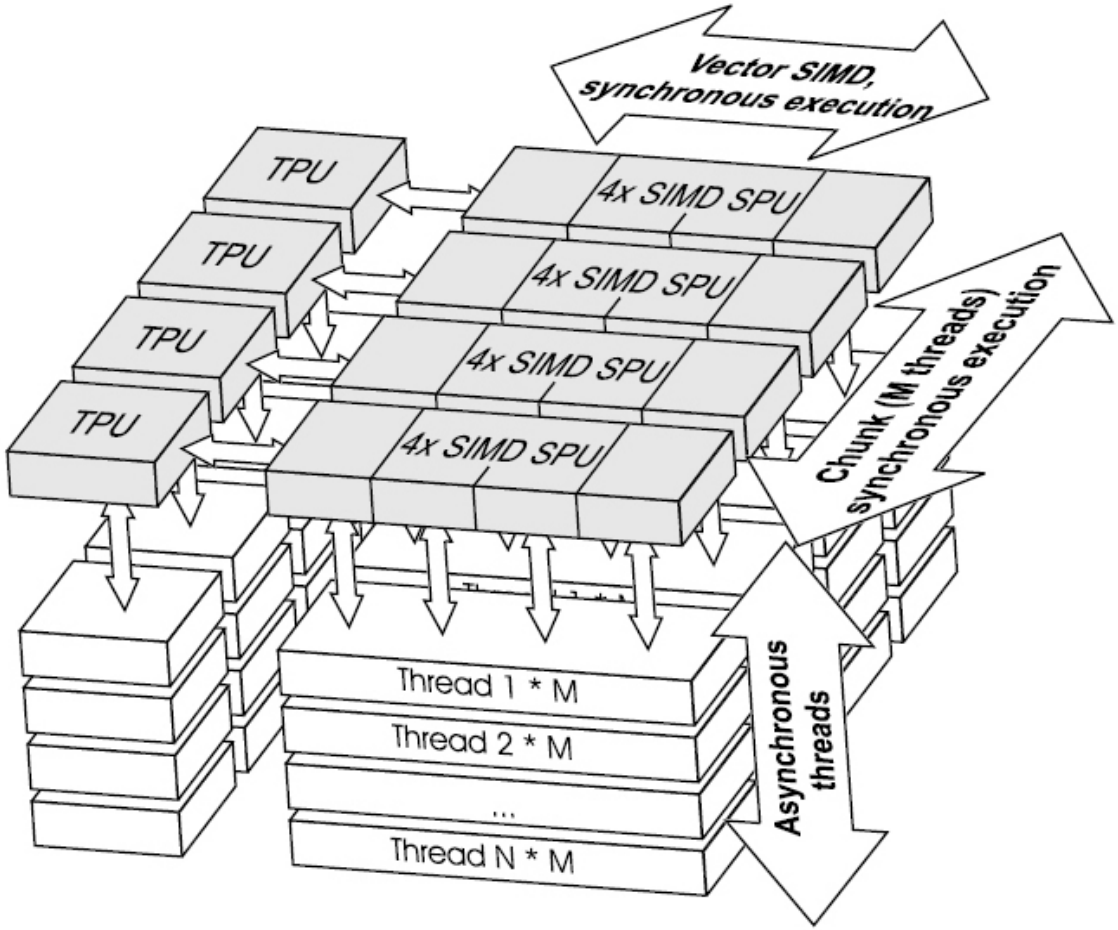


Paralellität I
SPU

- **M Chunks** bestehen aus **N Threads**
- Alle Threads eines Chunks werden paralell abgearbeitet
- Threads werden nach Bedarf (Abhängigkeiten) ausgeführt

“mixed thread and stream programming model”

Paralellität II



Paralellität III
SPU

- Ein Chunk wird von einem Satz SPUs paralell verarbeitet
- Eine SPU ist eine Vektor-ALU mit 4-Komponenten im SIMD Modell

Gliederung

weitere Eigenschaften I

- General Purpose Computing:
  Obwohl spezialisiert auf Raytracing, allgemeine Nutzung möglich
- Scaleable Design:
  Clustering von RPUs auf einem Chip; vielen Chips/Boards; PC-Clustern

Gliederung

Besonderheiten

- komplett deklarative Szenendefinition
- prozeduraler Ansatz
  - Lighting Shaders
  - Geometry Shaders
- dynamische Szenen
  - durch prozeduralen Anzatz leicht implementierbar
  - Problem: Index-Struktur

Gliederung

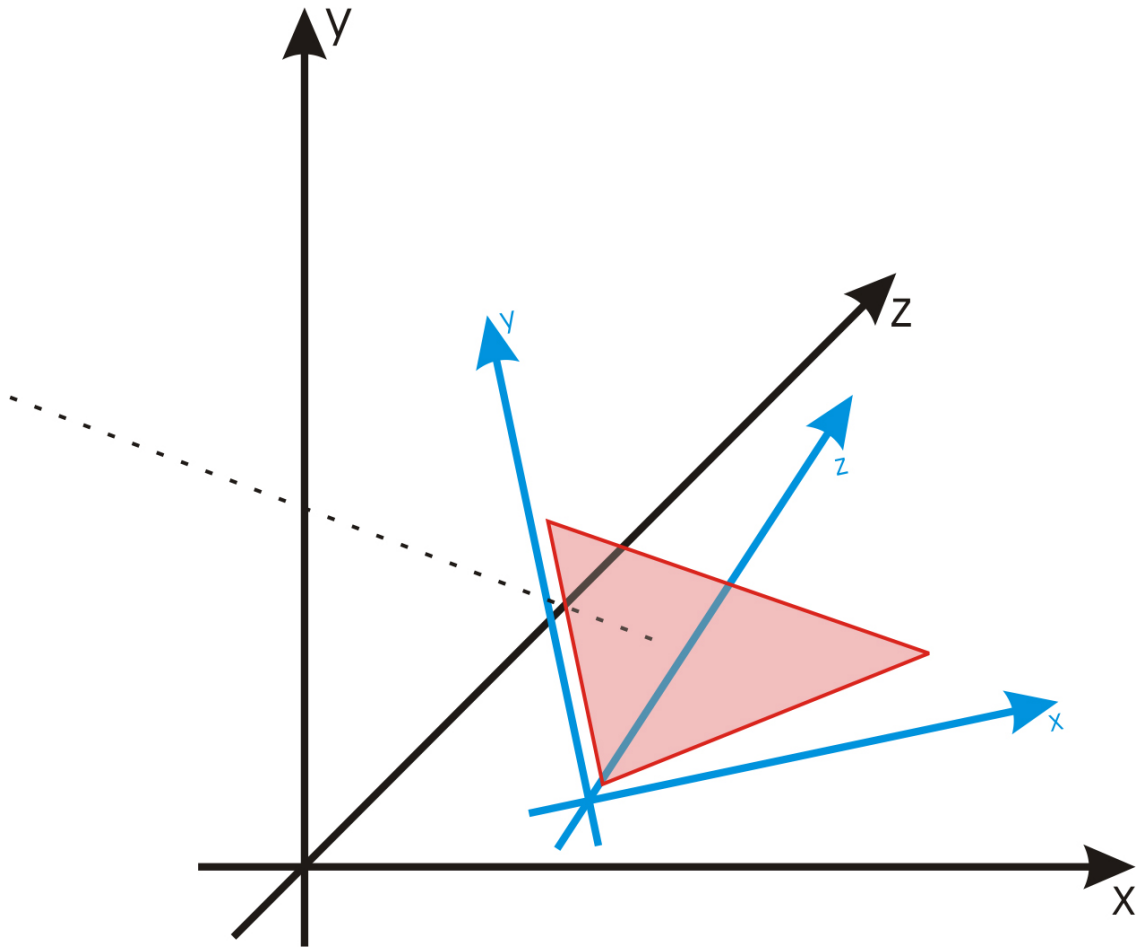
Der FPGA-Prototyp

- 4 SPUs
  - je 32 Hardware-Threads
  - in einem
    “Field Programmable Gate Array”
- betrieben mit 66 MHz

Grenzen des FPGA-Prototyps I

- keine Speicheroptimierungen $\Rightarrow$ Flaschenhals
- keine Integer-ALU $\Rightarrow$ Speicheradressen
- Shader
  - muss in der RPU resident sein
  - max. 512 x 80 bit pro Shader
  - 4 Parameter-Register pro Shader
  - max. 16 verschiedene Shader
- 24-bit Fließkomma-Präzision

Grenzen des FPGA-Prototyps II



Leistungsfähigkeit des Prototyps I

- theoretische Leistung max. 2,6 GFlops
- bis zu 20 FPS
  (nur primäre Strahlen)
- je nach Szene bessere Leistung als OpenRT auf Pentium 4 - 2,66GHz
- max. 50% der SaarCOR-Leistung
  (bei gleicher Taktung)

Leistungsfähigkeit des Prototyps II

- TPU erhöht Effizienz deutlich (fixed function)
- Vergleich mit SaarCOR zeigt 20% - 50% overhead durch FPGA-Technologie
- Gemessen an Leistung von GPUs *theoretisch* 27-fache Steigerung möglich

Schlußfolgerung

Mit aktueller Chiptechnologie ist es möglich Echtzeit-Raytracing mit vergleichsweise wenig Hardware-Aufwand zu betreiben.

Ausblick

- effizentere Lösung für Indexstrukturen mit Unterstützung von dynamischen Szenen
  $\Rightarrow$ in Hardware?
- Bildoptimierungen
  - Glanzreflexion
  - Anti-Aliasing
- weitergehende Nutzbarkeit
  - Allgemeine Sichtbarkeitstests
  - Kollisionsberechnungen
  - Nicht-optische globale Transportprobleme

Ausblick

Quo Vadis Raytracing in Hardware?