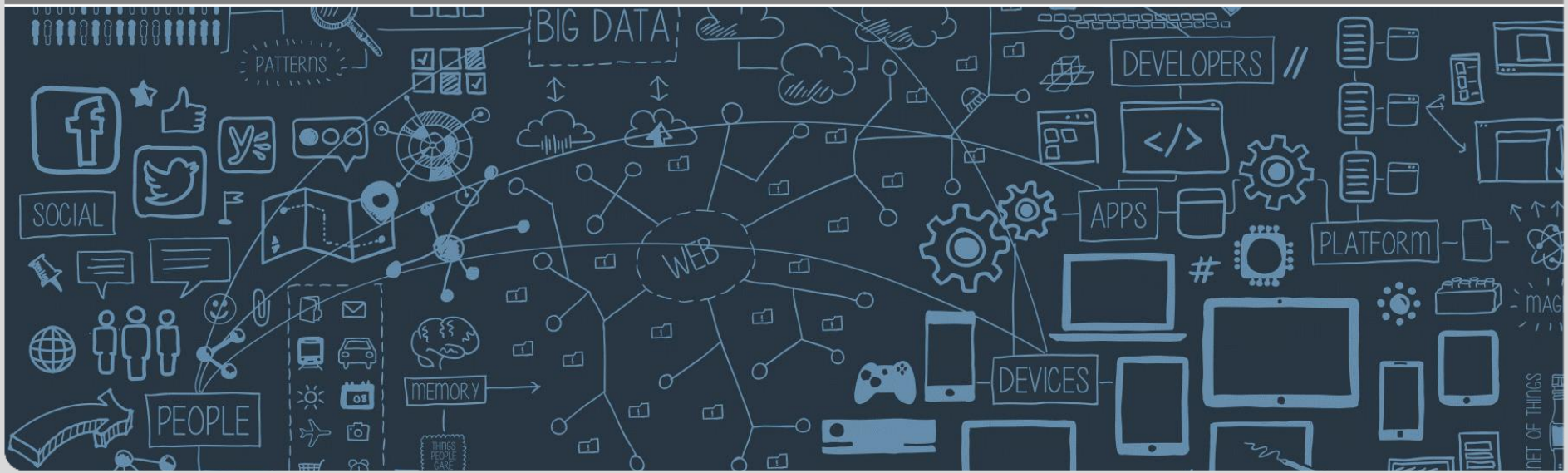
Karlsruher Institut für Technologie

Praxis der Multikernprogrammierung

Endergebnisse 06.02.2017 – Gruppe Nostrum

Manuel Karl, Dominik Kleiser, Marc Leinweber, Nico Mürdter

Institut für Programmstrukturen und Datenorganisation – Programmiersysteme



Allgemeine Implementierung

- Modularer und Objektorientierter Entwurf
  - Leicht erweiterbar
  - Kompilierbar auch auf Plattformen ohne GPU/CUDA, AVX und SSE
  - Vererbungshierarchie für räumliche Datenstrukturen
    - Ermöglicht einfachen Austausch
- Effiziente Algorithmen für geometrische Primitiven
  - Möller-Trumbore
  - Bounding Box Schnitt

Räumliche Datenstruktur

- Feststellung
  - Effiziente Datenstruktur häufig wichtiger als Parallelisierung
  - Es muss Trade-off zwischen Konstruktionszeit und Renderzeit gefunden werden

- Mehrere alternative Implementierungen
  - Vollständige Suche über alle Dreiecke
  - „naiver“ k-d-Baum (Median)
  - k-d-Baum mit Surface Area Heuristic (SAH)
  - Hybrid (erst Median, dann SAH)

- Konstruktion von SAH teuer
  - Bauzeit (in Abhängigkeit der Anzahl Dreiecke):
    - $O(nlog^2n)$
    - $O(nlogn)$

Parallelisierung des Raytracers

- Zunächst: OpenMP
  - Parallelisierung über die Bildpixel (*embarrassingly parallel*)
    - #pragma omp parallel for schedule(dynamic, 10) collapse(2)
- Danach: Strahlenbündel mit SIMD-Instruktionen
  - pc205 → SSE (4 Strahlen)
  - pc189 → AVX (8 Strahlen)
  - Erforderliche Anpassungen
    - Geometrische Algorithmen
    - Traversierung des k-d-Baums

Parallelisierung des Pathtracers

- Erster Versuch: Parallelisierung auf CPU
  - (siehe Raytracer)
  - Parallelisierung mit SIMD-Vektorregister nicht sinnvoll
    - Benachbarte Strahlen divergieren → teure Baum-Traversierung
  - Sehr hohe Laufzeiten! (exponentiell viele Strahlen)

- Idee: GPU besser geeignet
  - Überführen der vorhanden geometrischen Funktionen in CUDA-Code
  - Kopieren des k-d-Baums in eine für die GPU geeignete Struktur
  - Zwei Implementierungen
    - Ohne Rekursion (Speicherverbrauch explodiert!)
    - Rekursiv: Nicht optimal für GPU Programmiermodell

Schwierigkeiten

- Vektorisierung der geometrischen Funktionen nicht trivial
- GPU nicht gut geeignet für Rekursion
  - „Stackless“ Traversierung des k-d-Baums
    - Implementierung schwerer als gedacht
- GPU-Speicher bei manchen Szenen zu klein (vor allem auf pc205)
- SAH nicht immer beste Wahl
  - Benchmarks mit 12M Dreiecke → Lange Bauzeit
  - GPU kommt mit zu tiefem Baum nicht klar
  - Einführung einer Heuristik

Evaluation

- Getestete Szenen:

Szene	Auflösung	Dreiecke	Lichtquellen	Rekursiontiefe	Samples
Cube	8000x6000	12	2	-	-
Sponza	2048x2048	262267	1	-	-
Cornell (PT)	800x600	14	8	1	4000
Sponza (PT)	400x400	262267	12	1	32

- Ergebnisse:

Szene	naiv	k-d-Baum	+ AVX	+ SAH	GPU	Heuristik
Cube	00:03:84	00:03:73	00:03:42	00:03:21	-	00:03:50
Sponza	∞	01:41:17	00:21:51	00:06:79	-	00:06:38
Cornell (PT)	∞	01:22:01	-	01:25:76	00:22:31	00:20:05
Sponza (PT)	∞	07:17:73	-	00:17:84	01:29:12	00:17:71

Fazit

- Essentiell bevor man parallelisiert:
  - Modulare Struktur
  - Gute sequentielle Lösung
    - Effiziente Algorithmen (Möller-Trumbore)
- Gute Ergebnisse für Raytracing
  - CPU mit SSE wohl stärker als GPU
- Pathtracing akzeptabel
- OpenMP ist „toll“