GPGPU

General Purpose Computation On Graphics Processing Units

(Grafikkarten-Programmierung)

Von:
Marc Blunck

Ablauf

- Einführung GPGPU
- Die GPU
- GPU Architektur
- Die Programmierung
- Programme
- Fazit

Was ist *GPGPU* ?

- GPU zweckentfremden

Anwendungsfelder:

→ nicht nur auf graphische Probleme beschränken sondern Nutzung für allgemeine rechenintensive Vorgänge.

Warum *GPGPU* ?

1. Performance

- Parallelität ist die Zukunft
- CPU hat 2 Prozessoren (dual core)

→wenig Parallelität → **serielles** Modell

- GPU hat > 24 Prozessoren (GeForce 6 Serie)

→hochgradig **parallel**

Unterschied:

- **serielle** Prozessoren arbeiten auf beliebigen Datenelementen, geringe Latenz.
- **parallele** „Stream-Prozessoren“ arbeiten auf großen homogenen Datenmengen, auf hohen Durchsatz optimiert.

2. Lastenverteilung

- GPU übernimmt eigene Aufgaben und muss nicht mehr auf die CPU warten.

Performance Vergleich

- nVIDIA GeForce 6800 → 60 GFlops
- Pentium 4 → 12 GFlops

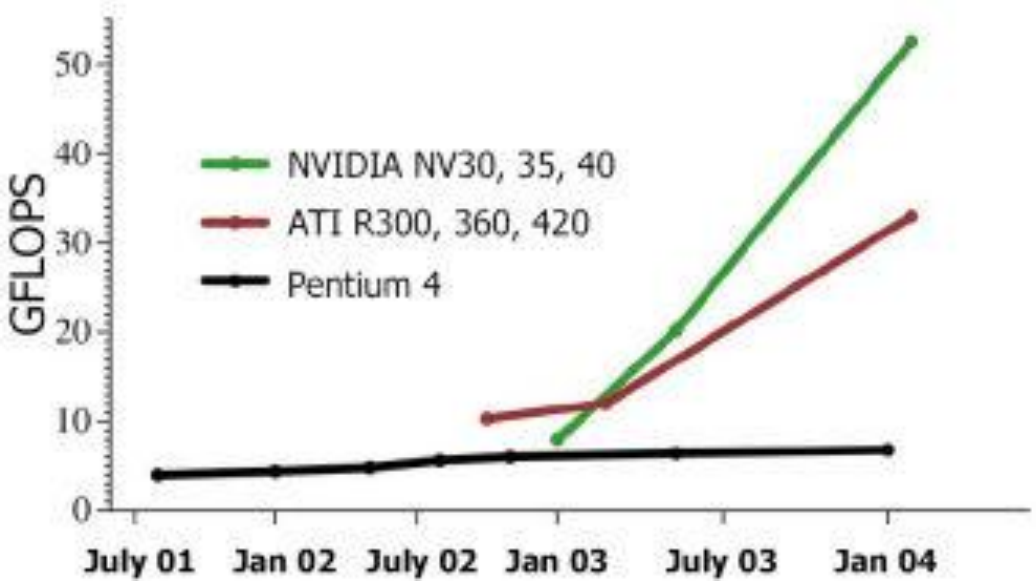


Abbildung 1: Vergleich GFlops [2]

nVidia GeForce 6800 Ultra hat 220 Mio. Transistoren
AMD Athlon 64 hat 105,9 Mio. Transistoren

Vorteil

GPUs sind billig aber leistungsfähig

→niedrige Kosten pro GFLOP

Die GPU im System

Komponente	Bandbreite
GPU Memory Interface	35 GB/sec
PCI Express Bus (×16)	8 GB/sec
CPU Memory Interface (800 MHz Front-Side Bus)	6.4 GB/sec
CPU 1st Level Cache	max. 35 GB/sec

(Keine Angaben zu Mainboards bzw. Zeitpunkten)

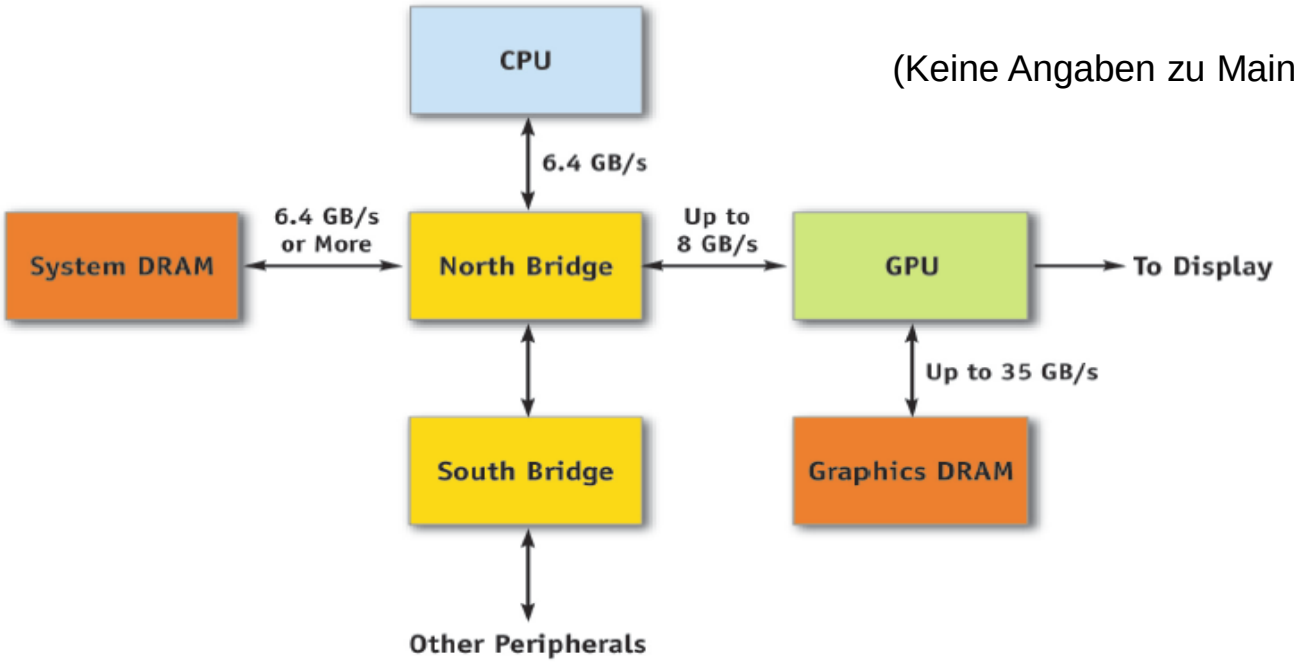


Abbildung 2: Architektur der GeForce 6 Serie [1]

GPU Architektur GeForce 6 Serie

(zu ATI kaum Infos!)

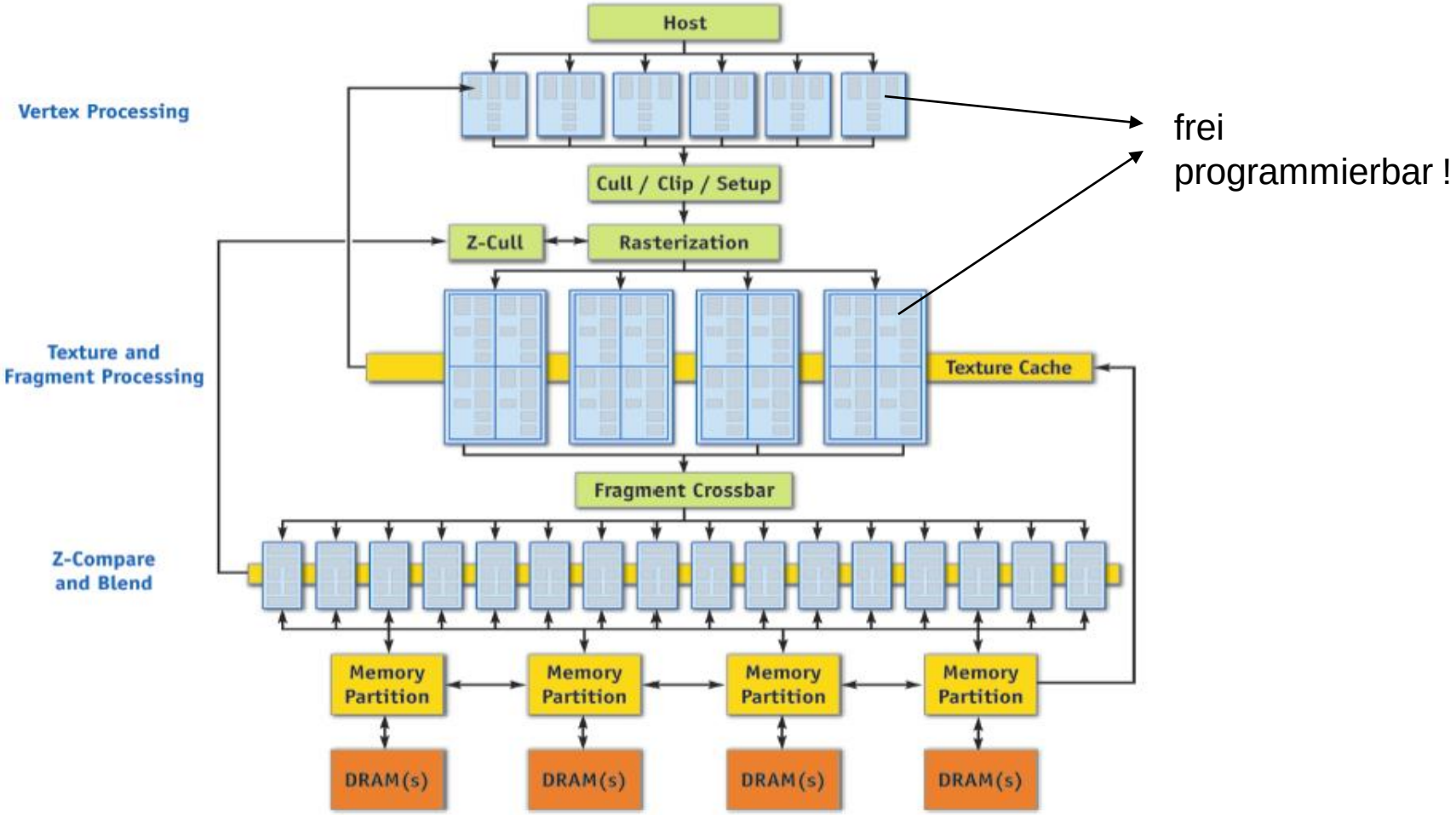
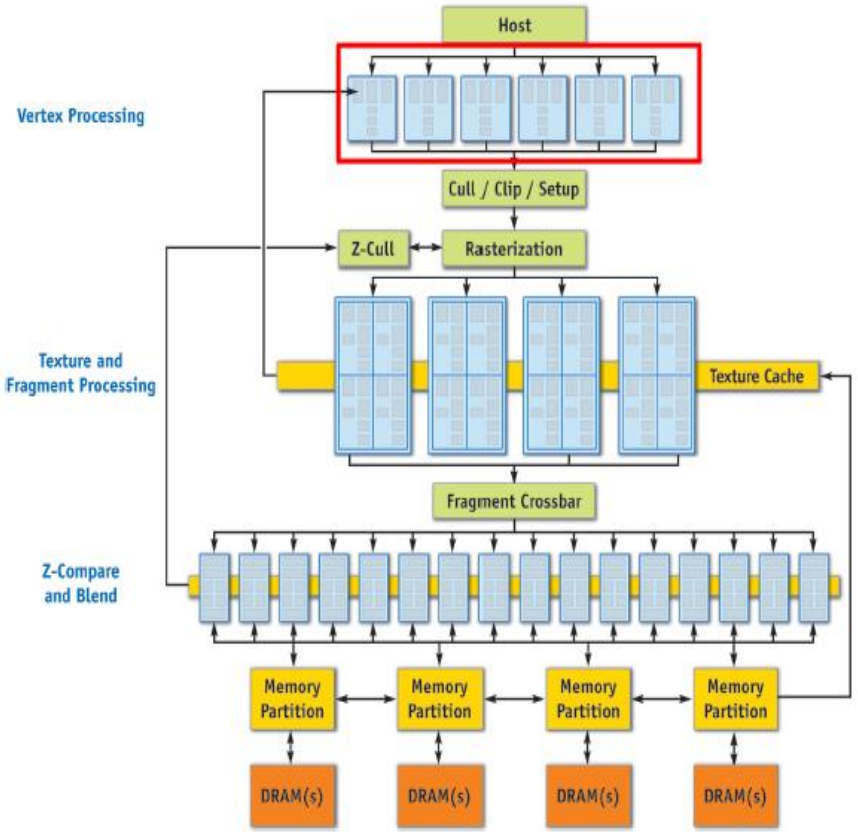


Abbildung 2: Architektur der GeForce 6 Serie [1]

Die Geometrieeinheit

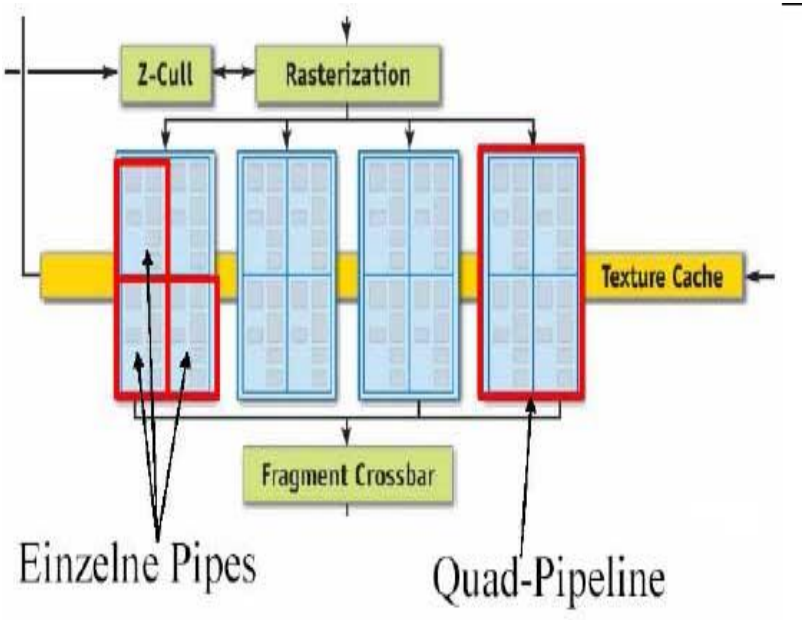


[Vertex-Prozessor / Vertex-Shader]

- mehrere Parallele Prozessoren nebeneinander
- nutzen SIMD & MIMD-Parallelisierung
- aus 3 Vertices wird Dreieck errechnet
- hat einen eigenen Befehlssatz und ist frei programmierbar!
- Vertex-Cache

Abbildung 3: Vertex-Prozessor einer GeForce 6 Serie [1]

Die Textur- und Fragmenteinheit



[Fragment-Prozessor / Pixel-Shader]

• nach SIMD Parallelisierung arbeitender Spezialprozessor in der GPU (analog Vertex Shader)

• belegt die Bildpunkte mit Eigenschaften wie Farbe, Transparenz, Reflexionsverhalten usw.

• Texture-Cache

Abbildung 4: Fragment-Prozessor

Stream-Modell

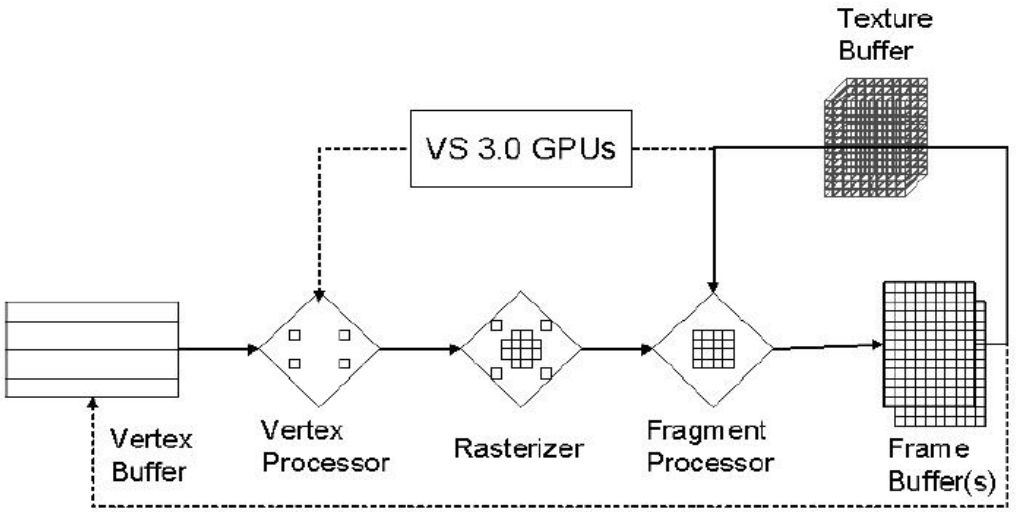


Abbildung 5: GPU Streams

- Effiziente Berechnungen durch Parallelität
- Basis für Programmierung auf GPUs
- Stream-Prozessoren arbeiten auf kompletten Datenströmen (Streams) und sind damit schnell
- Für Programmierung meist Fragment-Prozessoren

Wieso CPU nicht durch GPU ersetzen?

- GPU kann nur auf bestimmten Arten von Daten schnelle Operationen ausführen.
- Operationen müssen parallel verarbeitet werden können.

..., weil GPU darauf ausgelegt ist, auf kompletten Datenströmen (Streams) zu arbeiten!

→GPU ist als Stream-Prozessor konzipiert.

C_g - C for Graphics

- Von NVIDIA entwickelt, basiert auf C, ähnliche Syntax
- Datentypen wurden für GPGPU ergänzt:

Datentyp	Beschreibung
float	32 Bit Fließpunktzahl
half	16 Bit Fließpunktzahl
int	32 Bit Integer
fixed	12 Bit Fixpunktzahl
bool	Boolean Variable

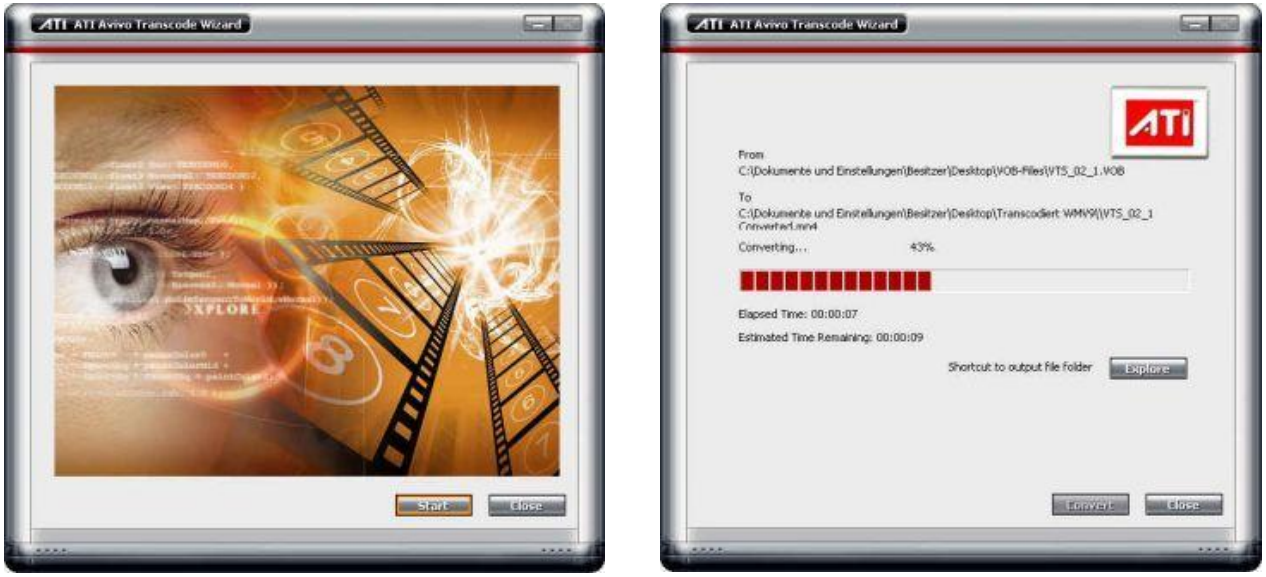
Vorteile:

- Leichter als Assembler zu erlernen, programmieren, lesen und zu verstehen.
- Auf viele verschiedene Plattformen portierbar
- Code kann von Compilern optimiert werden

Programme (1)

- **ATI Avivo Xcode**

Video-Encoder von ATI, der die Rechenpower der Grafikkarte nutzt und so das Umrechnen von Filmen in andere Formate drastisch beschleunigt.

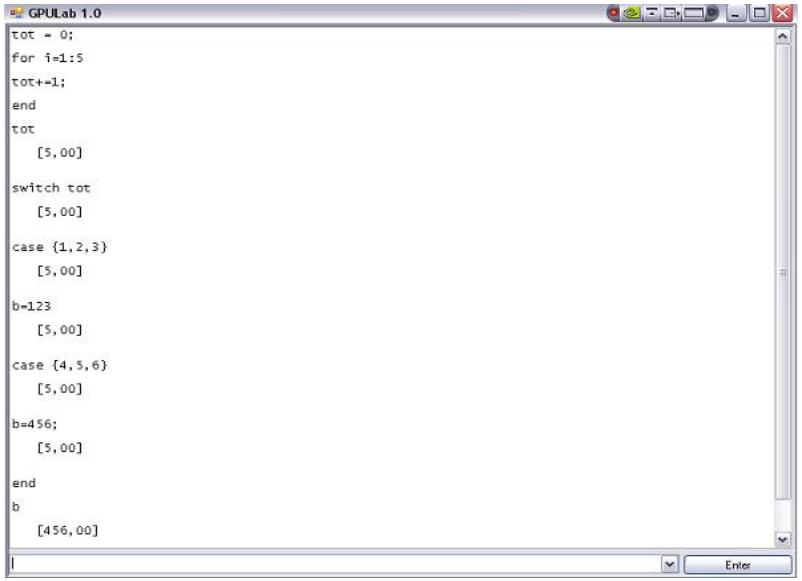


Quelle: [3]

Abbildung 3: ATI Avivo Xcode Benutzeroberfläche

Programme (2)

- **GPULab**
  - Matlab-ähnliche Sprache für lineare Algebra, welche die GPU nutzt.



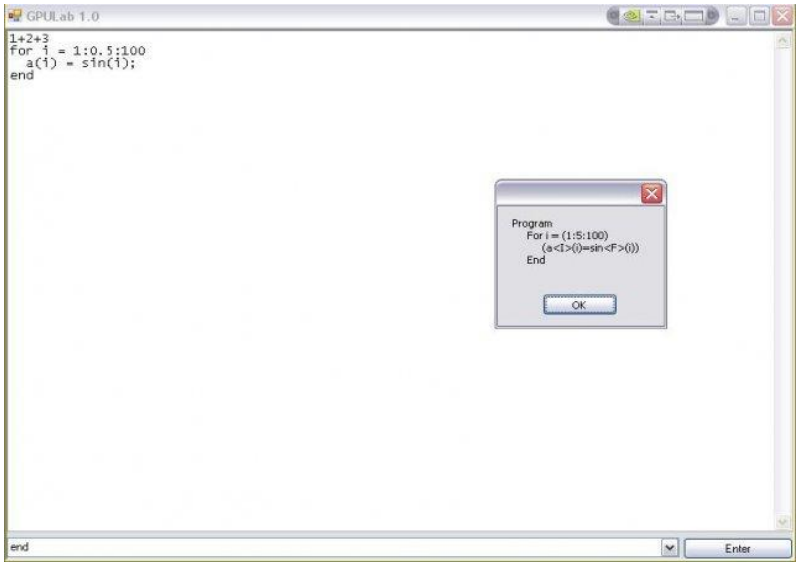


Abbildung 4: GPULab Benutzeroberfläche

Quelle: [4]

Programme (3)

- **FFFF - Fast Floating Fractal Fun**
- Verschiedene Optimierungen
- Benchmarkfunktion

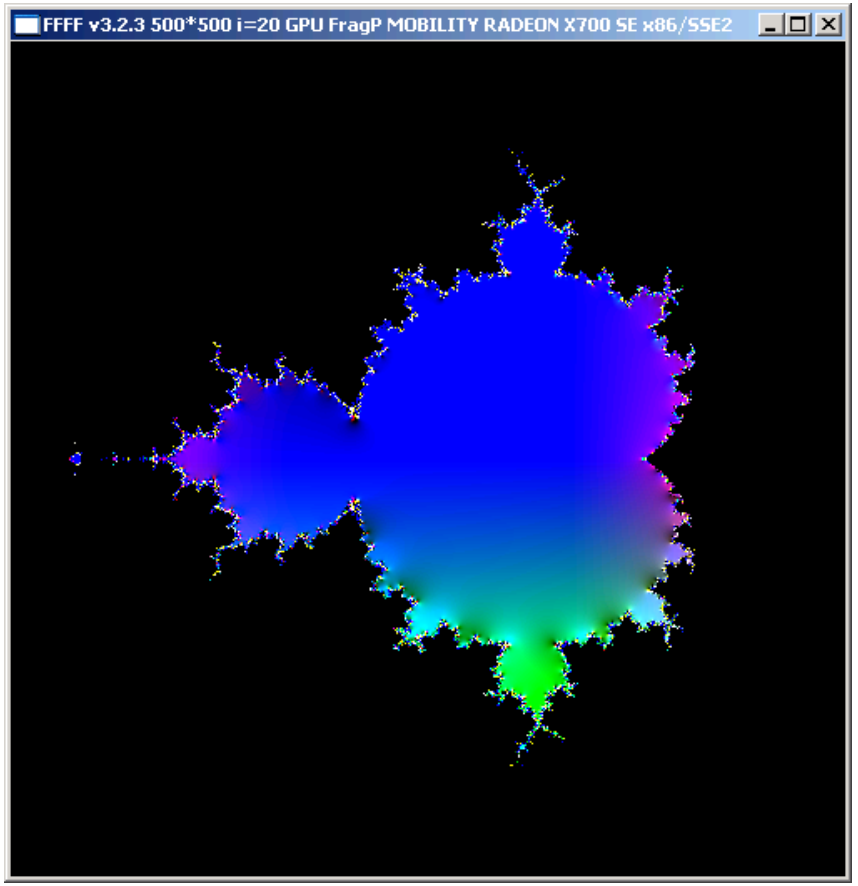


Abbildung 5: Mandelbrotmenge mit FFFF

Quelle: [6]

Programme (4)

- **GPUSort**
- Sortieralgorithmus in der GPU implementiert
- Läuft momentan nur auf NVIDIA Karten

Quelle: [7]

Programme (5)

- **Brook**

- Sprache zur GPU Programmierung
- Projekt, welches einen Compiler und eine Laufzeitumgebung bereitstellt, um moderne Grakkarten zu programmieren.
- nutzt intern Cg

Quelle: [8]

Fazit

- GPU teilweise deutlich schneller als CPU
- Programmierung nicht trivial
- Bei Vielzahl von Programmen neueste Hardware erforderlich

Quellen

[1] **The GeForce 6 Series GPU Architecture:**
http://download.nvidia.com/developer/GPU_Gems_2/GPUGems2_ch30.pdf

[2] **Wikipedia**
http://de.wikipedia.org/wiki/GPGPU

[3] **ATI Avivo Xcode**
http://www.chip.de/downloads/c1_downloads_18000750.html

[4] **GPUlab**
http://gpulab.sourceforge.net/

[5] **Dominik Göddeke -- GPGPU::Basic Math Tutorial**
http://www.mathematik.uni-dortmund.de/%7Egoeddeke/gpgpu/tutorial.html

[6] **Fast Floating Fractal Fun**
http://sourceforge.net/projects/ffff

[7] **GPUSort**
http://gamma.cs.unc.edu/GPUSORT/

[8] **Brook**
http://graphics.stanford.edu/projects/brookgpu/

[9] **GPGPU**
http://gpgpu.org