Programmierbare Hardware-Shader

Jörn Loviscach
Hochschule Bremen

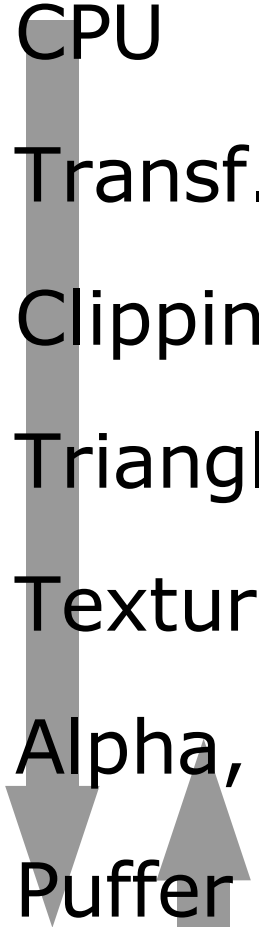
Überblick

- Vertex- und Pixel-Shader
- Anwendungsbeispiele
- fx-Dateien
- Anwendungsbeispiele
- Zusammenfassung

Vertex- und Pixel-Shader

Hardware-Renderpipeline

klassisch



CPU

Transf., Beleuchtg., Proj.

Clipping, w-Division

Triangle Setup, Rasterung

Texturierung, Nebel

Alpha, Tiefentest

Puffer

Vertex- und Pixel-Shader

Hardware-Renderpipeline

Vertex-Shader

- kein Zugriff auf andere Vertices
- kein Erzeugen eines neuen Vertex
- kein Löschen des Vertex
- kein Zugriff auf Texturen

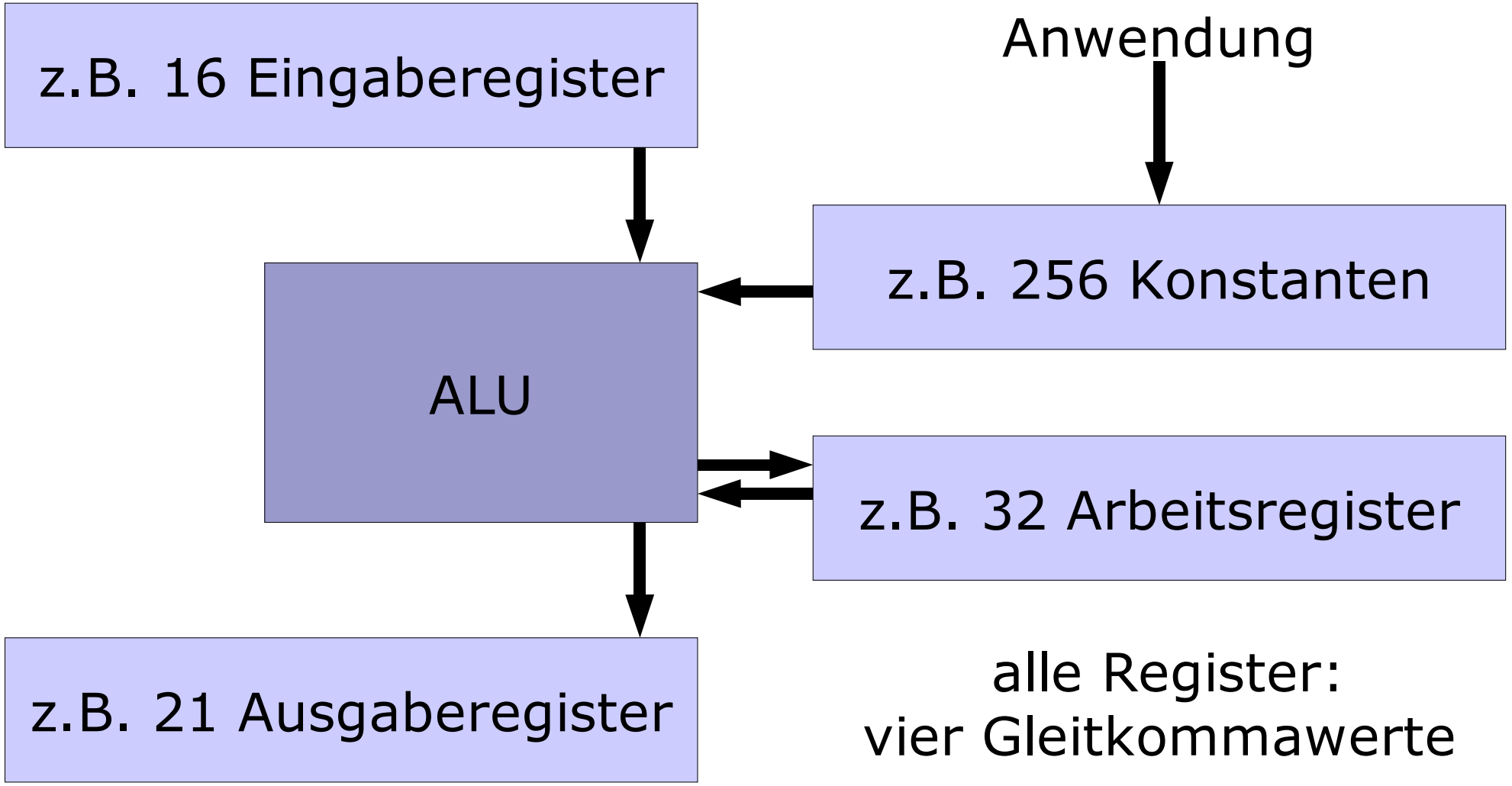
Vertex-Shader

- kein Zugriff auf andere Vertices
- kein Erzeugen eines neuen Vertex
- kein Löschen des Vertex
- kein Zugriff auf Texturen

- Verformungen
- Verzerrung von Textur-Koordinaten
- Farbänderung pro Vertex

Vertex-Shader

Ausführungsumgebung



Pixel-Shader

- kein Zugriff auf andere Pixel
- Bildschirmposition unveränderlich

Pixel-Shader

- kein Zugriff auf andere Pixel
- Bildschirmposition unveränderlich

- Zugriff auf Texturen
- Vertex-Geometriedaten als Input
- ggf. Ableitungen bzgl. Bildschirm-xy
- Farbberechnungen
- Überblendungen
- Tiefe bzgl. Bildschirm ändern

Vertex-/Pixel-Shader

Beispiele aus:
Nvidia Cg Browser

- Vertex Noise
- Procedural Terrain
- Matrix Palette Skinning
- Fresnel Demo
- Detail Normal Maps
- Bump Reflection Mapping
- Refractive Dispersion

Vertex-/Pixel-Shader

Programmierung

- maximal z.B. 256 Befehle
- meist vier float (32 Bit) als Vektor
- „Swizzle“: ... = C[95].xwzx
- „Write Mask“: R[10].xz = ...
- Schleifen, Unterprogramme, Sprünge mit Einschränkungen
- Geometriebefehle
- Texturabfragen (Pixel-Shader)

Vertex-/Pixel-Shader

Performance (nur Vertex-Shader)

- GeForce 3 Ti 200 4* 160 MFLOPS
- GeForce 4 Ti 4200 4* 470 MFLOPS
- GeForce FX 5900 4*1100 MFLOPS

Parallelität:
z.B. vier Vertex-Shader-Einheiten,
acht Pixel-Shader-Einheiten

Vertex-/Pixel-Shader

Assembler

- DirectX
- OpenGL ARB
- proprietär

Vertex-/Pixel-Shader

Assembler
- DirectX
- OpenGL ARB
- proprietär

Hochsprachen
- Nvidia Cg
- DirectX HLSL
- OpenGL SLang

Demo:
Nvidia Cg Browser

Vertex-/Pixel-Shader

Beispiele

- Qbist2:
  prozedurale Echtzeit-Videos
- genetische Suche
  nach Bild-Algorithmen
- evolutionäres Design
  von Materialien

.fx-Dateien

.fx-Dateien

fx: „Effects“

Materialien zuzuweisen und einzustellen
soll kein Job für Programmierer sein,
sondern für Designer!

komplette Programmierung
eines Materials als .fx-Textdatei

.fx-Dateien

- Vertex- und Pixel-Shader-Quellcode
- Alpha, z-Vergleich etc.
  als Textkommando
- ggf. mehrere Durchgänge (Passes)
- ggf. mehrere Techniken (Techniques)
- Eingabegrößen (vor allem Matrizen)
- grafische Oberfläche

Demo:
Nvidia CgFX Viewer

.fx-Dateien

Arbeitsablauf

in 3D-Software (Maya, XSI, 3ds max):

- Modelle bauen
- .fx-Materialien zuweisen
- mit Echtzeitvorschau einstellen

in Game-Engine:

- Modelle laden
- .fx-Materialien laden
- Modelle mit jeweiligen .fx rendern

.fx-Dateien

.fx-Frameworks

- Microsoft DirectX 9:
  direkt fx-Dateien verwendbar
- Nvidia CgFX: Anbindung
  an OpenGL und DirectX

Demo:
.fx in Managed DirectX

.fx-Dateien

Beispiel: C4Dfx

Besonderheiten:

- .fx in Datei rendern
- Grafikkarte auch für Offline-Rendering (siehe Maya)
- dazu Emulation des Standard-Materials als .fx-Datei im Speicher

.fx-Dateien

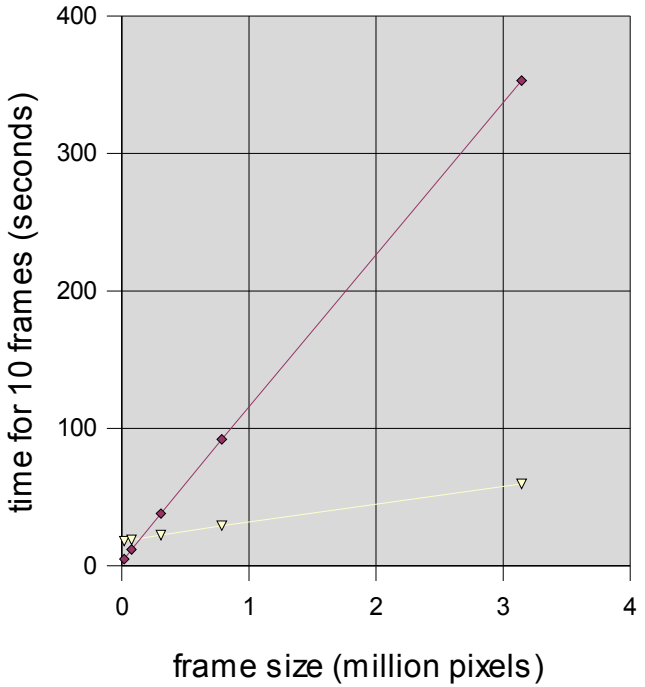
Beispiel: C4Dfx

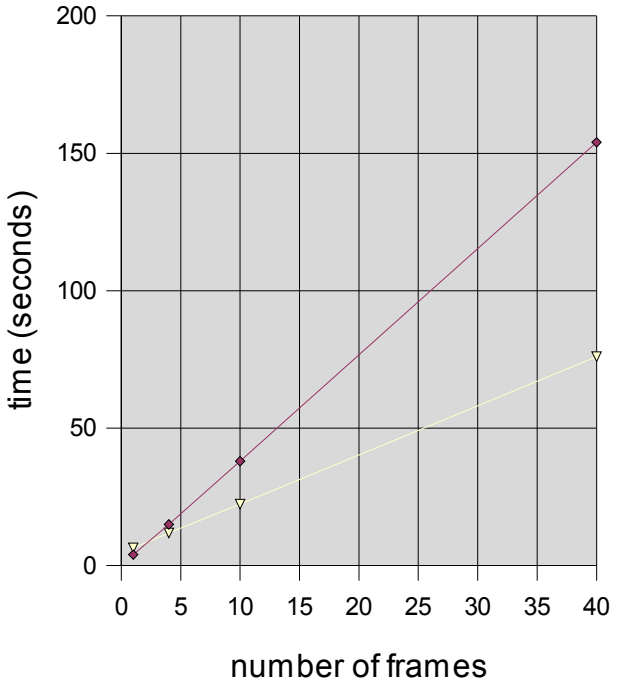
Emulation
Demo: interne .fx-Datei

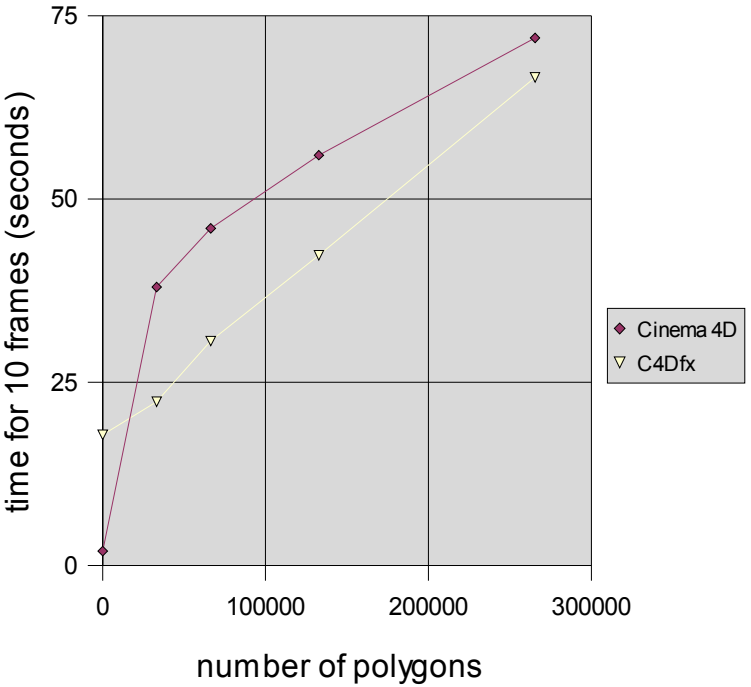
- Bump-Map zu Normal-Map
- Environment-Map von Kugel auf Würfel
- Glanzlicht-Form als Textur
- für Schatten Tiefen-Map gerendert

.fx-Dateien

Benchmarks zu C4Dfx







Zusammenfassung

Zusammenfassung

- Vertex- und Pixel-Shader machen große Teile der Render-Pipeline programmierbar

Zusammenfassung

- Vertex- und Pixel-Shader machen große Teile der Render-Pipeline programmierbar
- Programmiermodell auf Geometrie- und Farbverarbeitung gerichtet

Zusammenfassung

- Vertex- und Pixel-Shader machen große Teile der Render-Pipeline programmierbar
- Programmiermodell auf Geometrie- und Farbverarbeitung gerichtet
- ungestörte Parallelität

Zusammenfassung

- Vertex- und Pixel-Shader machen große Teile der Render-Pipeline programmierbar
- Programmiermodell auf Geometrie- und Farbverarbeitung gerichtet
- ungestörte Parallelität
- hohe Leistung

Zusammenfassung

- Vertex- und Pixel-Shader machen große Teile der Render-Pipeline programmierbar
- Programmiermodell auf Geometrie- und Farbverarbeitung gerichtet
- ungestörte Parallelität
- hohe Leistung
- .fx-Dateien als Mittel, komplexe Materialien zu bündeln, auszutauschen und für Designer zugänglich zu machen