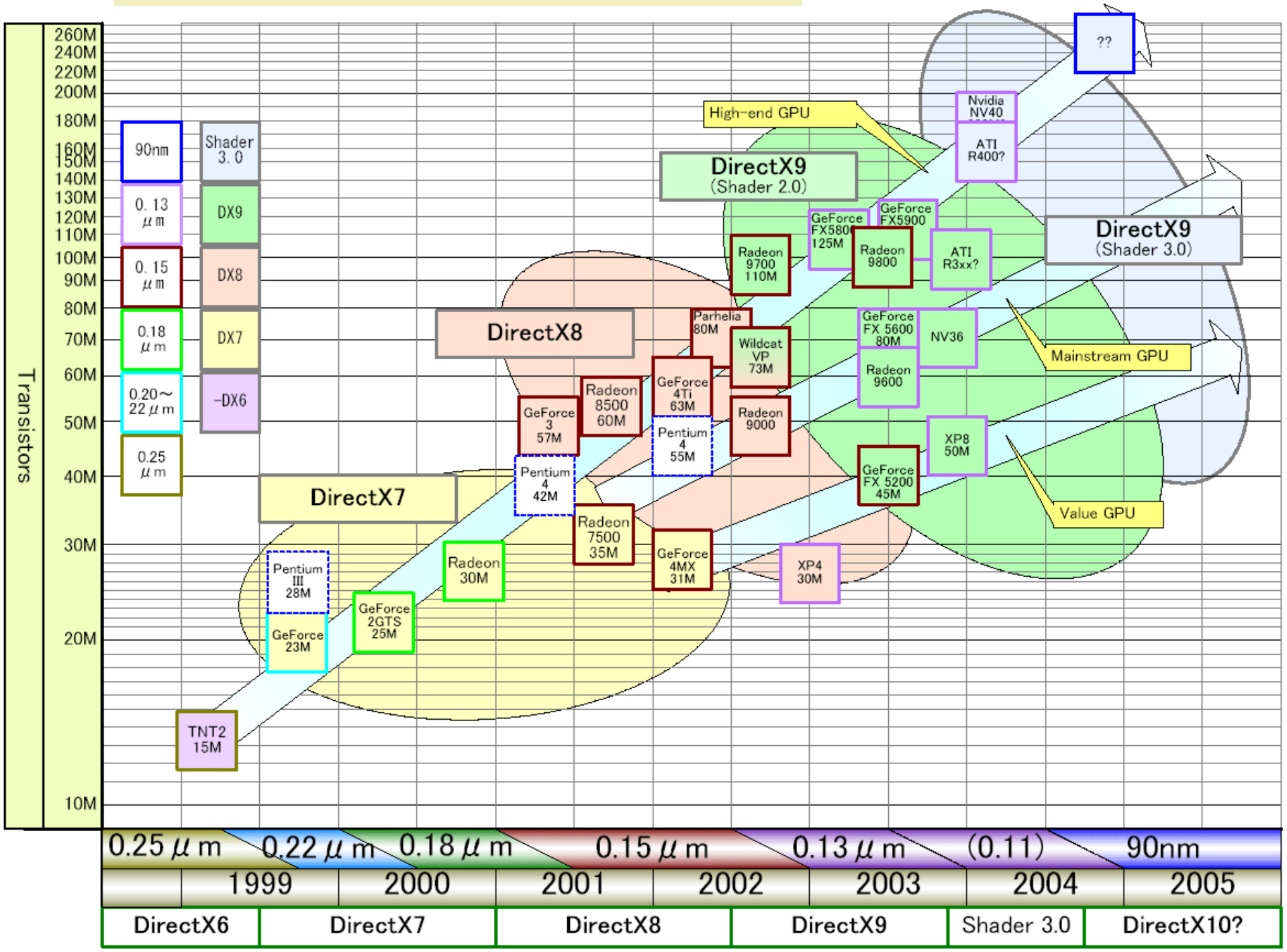
Architektur moderner GPUs

W. Sczygiol - M. Lötsch

Überblick

- Chipentwicklung
- Aktuelle Designs
  - Nvidia: NV40
  - (ATI: R420)
- Vertex-Shader
- Pixel-Shader
- Shader-Programmierung
- ROP - Antialiasing
- Ausblick
- Referenzen

GPU Transistor Count & Process Technology



Besonderheiten der Grafikberechnung

- Probleme:
  - Großes Datenvolumen
  - Hohe Berechnungskomplexität
  - Echtzeitanforderung

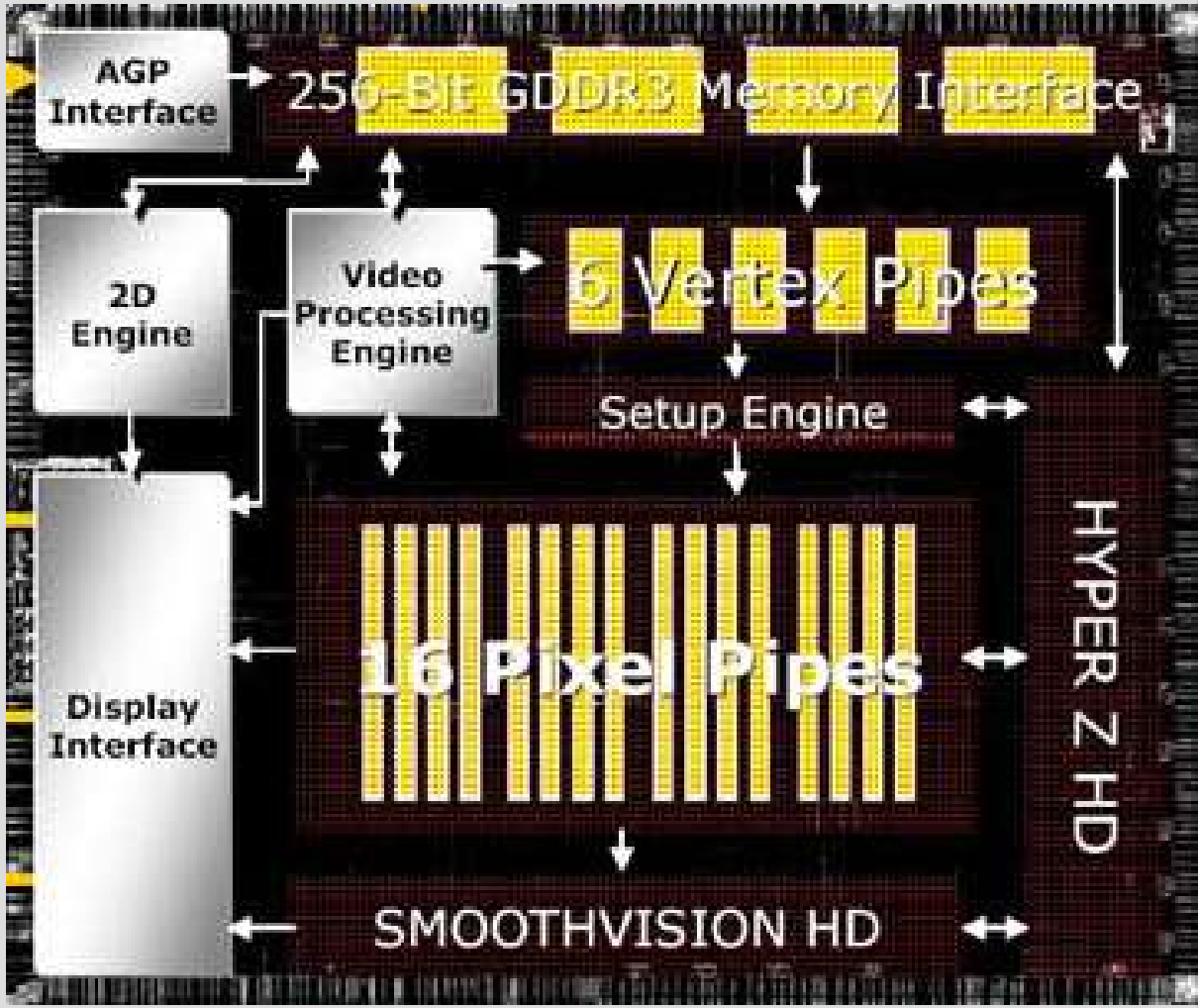
- Erleichterungen:
  - Geringe Datenabhängigkeit
  - Viele Berechnungen nicht notwendig
  - Gute Parallelisierbarkeit

NVidia NV40 Design

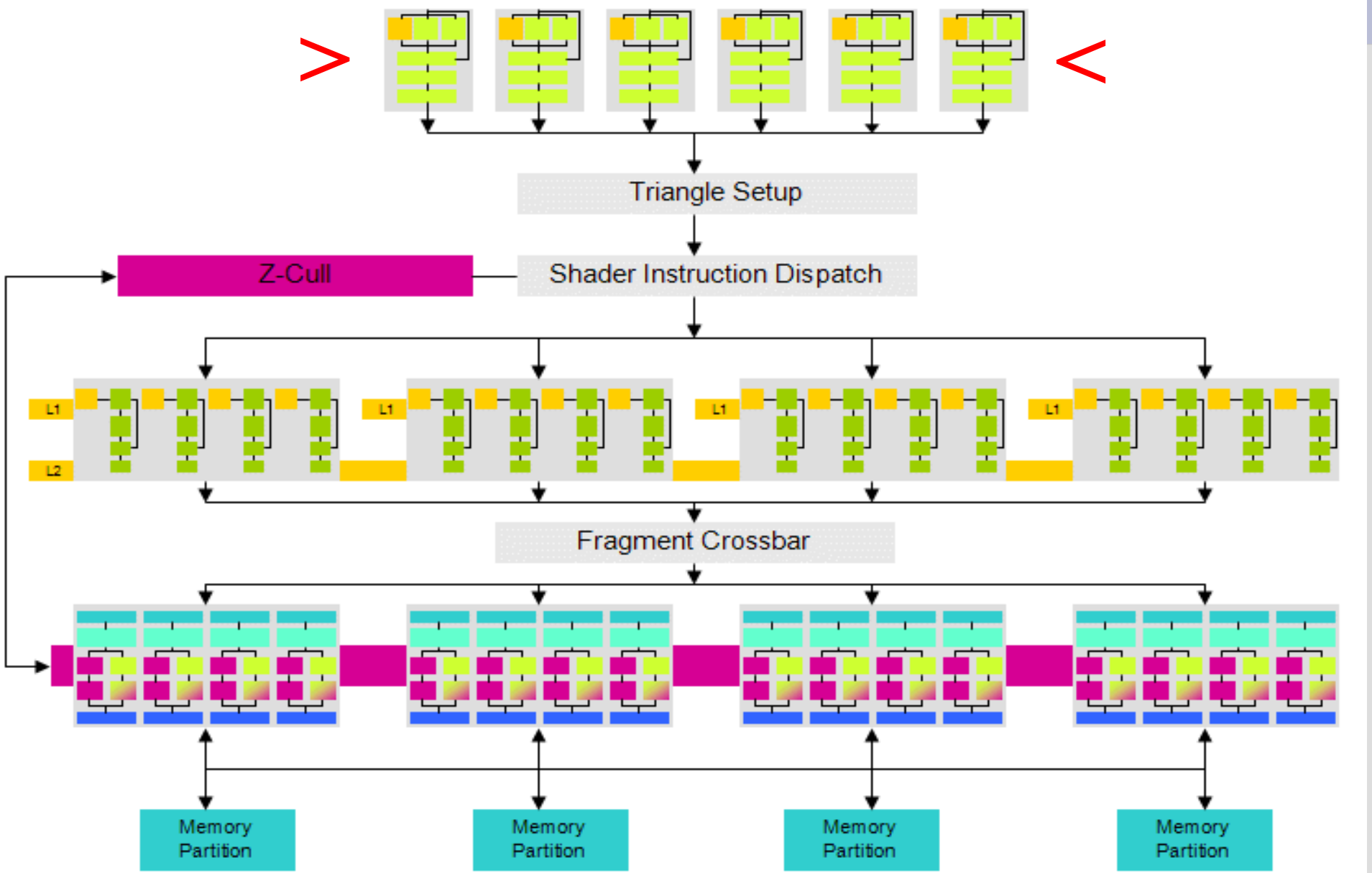
- GPU Features:
  - Chiptakt: 400 MHz
  - Speichertakt: 550 MHz
  - Transistoren: 220 Mio
  - Fertigungsprozess: 0,13µm
  - Vertexpipelines: 6
  - Pixelpipelines: 16
  - Leitungsaufnahme: ca. 120 W
  - Schnittstelle: AGP 8x oder PCI-Express
  - Programmierbarer Video-Processing Chip

Chiplayout

- AGP Interface
- 2D Engine
- Video Processing Engine
- Display Interface
- Memory Interface
- 6 Vertex Pipelines
- 16 Pixel Pipelines



GPU Architektur



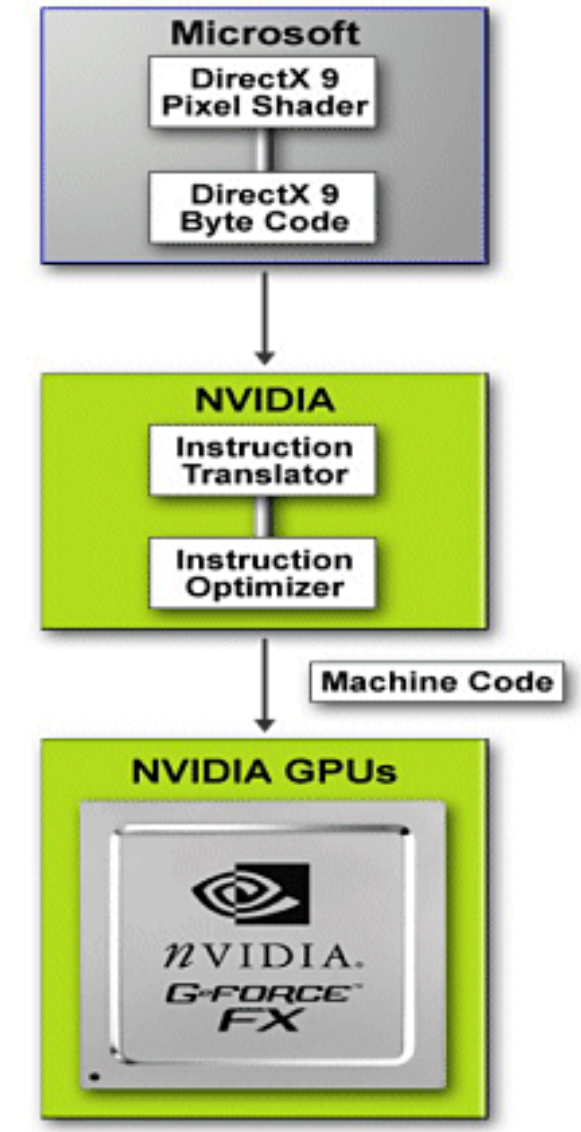
Der Vertexshader

- Vertexshader
  - Führt geometrische Berechnungen durch
    - Rotation
    - Skalierung
    - Translation
  - Kann Effekte produzieren
    - Verdrehungen
    - Verformungen
  - Ist bei aktuellen GPUs relativ frei programmierbar

Programmierung

- Low Level:
  - Assembler
- High Level:
  - OpenGL Shader Language (GLSL)
  - Microsoft's “High Level Shader Language” (HLSL)
  - NVidia's Cg

- Resultat in allen Fällen: Byte-Code
  - Wird erst vom Treiber in Maschinencode übersetzt



Programmierung - Beispiele

- Assembler:

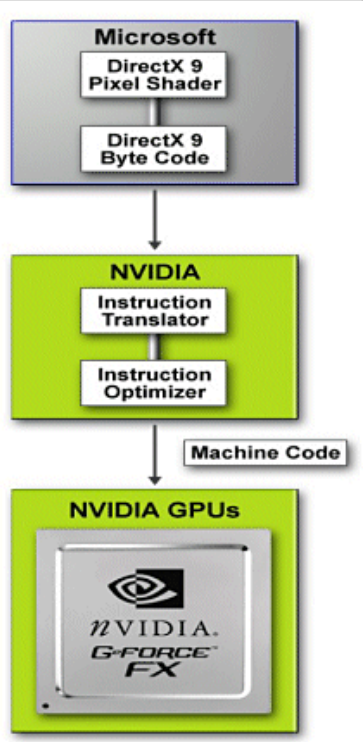
```
vs_1_1
mov a0.x,r1.x;
m4x3 r4,v0,c[a0.x + 9];
add r5,v3.xyzw,r7.yxwz;
```

- HLSL:

```
void main(in a2v IN, out v2p OUT) {
   OUT.Position = mul(IN.Position,ModelMatrix);
   OUT.Color    = IN.Color * diffuse;
 }
```

Shader Modelle und DirectX

- Shader Modelle spezifizieren Anforderungen an GPUs.
- Sind Teil der DirectX-API
- Bestimmen Mindestanforderungen an:
  - Registerbreite und Anzahl
  - Programmlängen
  - Befehlssatz
  - Kontrollflussmöglichkeiten
- Aktuell ist Modell 2.0

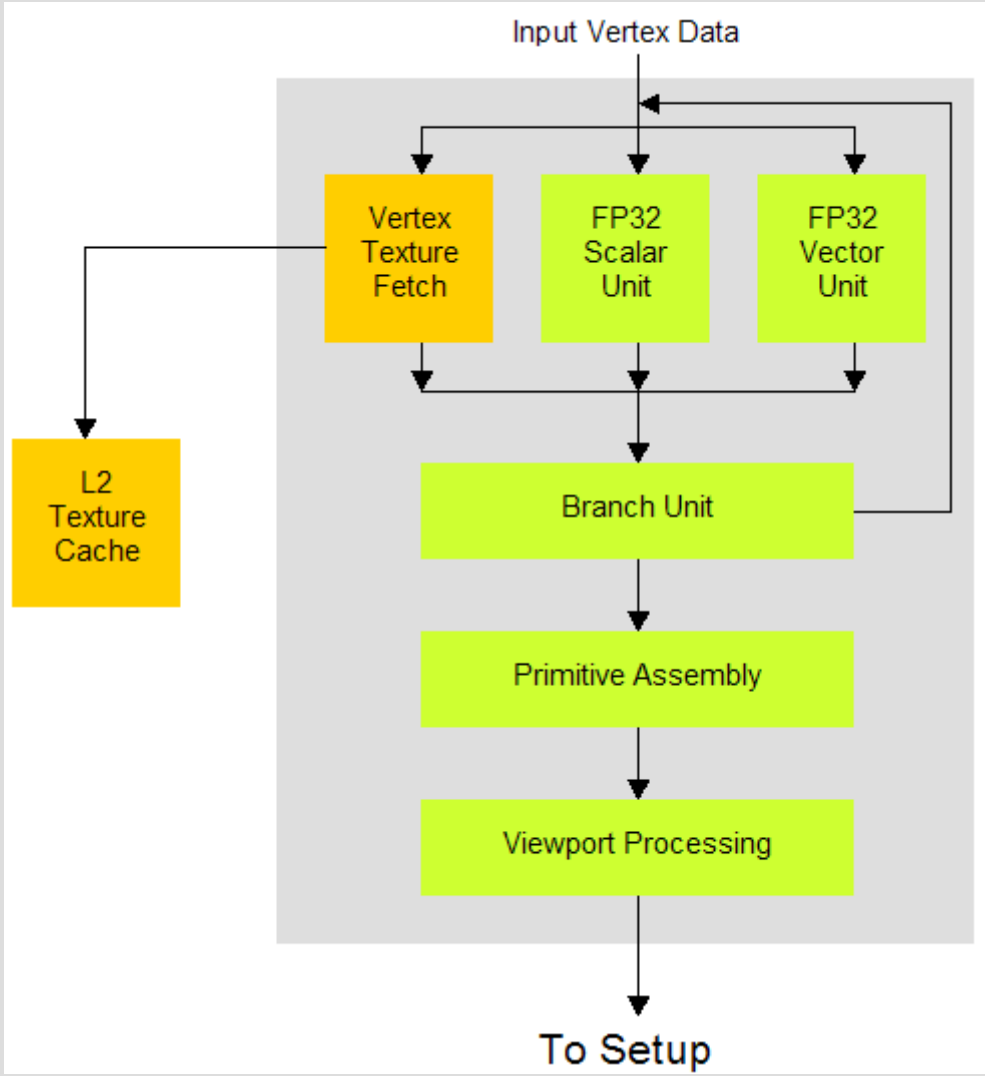


Programmierbarer Vertexshader

- NV40 implementiert als erste GPU das neue Vertex-Shader-Modell 3.0
  - Shader-Programmlänge mindestens: 512 Befehle
  - Register für Variablen: 32 x 32 Bit
  - Register für Konstanten: 256 x 32 Bit (und mehr)
  - Statische und dynamische Sprünge möglich
  - Verschachtelte Schleifen (Tiefe 24)
  - Zugriff auf Texturen während der Geometrieberechnung möglich

Vertexshader - Aufbau

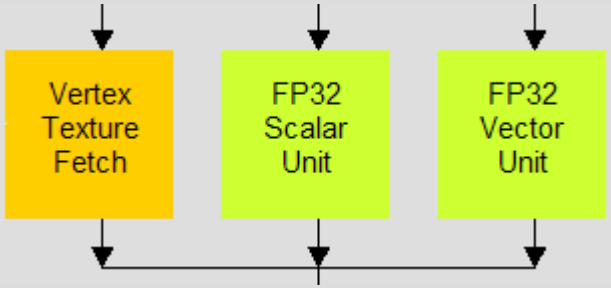
- Zugriff auf Texturen auch von der Vertexeinheit möglich
- Spezielle Branch Unit ermöglicht dynamische Sprünge
- Textur-Caching verkürzt Latenzzeiten erheblich
- Parallele Auslastung der beiden Units nicht immer gegeben
- Compiler müssen dies durch geschickte Befehlsanordnung sicherstellen



Optimierungsbeispiel

- Implementierung 1:

```
mul r0.x,r5.x,c2.x
exp r1.y,c1.y
mul r4,r3,c1
add r2,r0,r1
```



Berechnungsdauer: 3 Takte

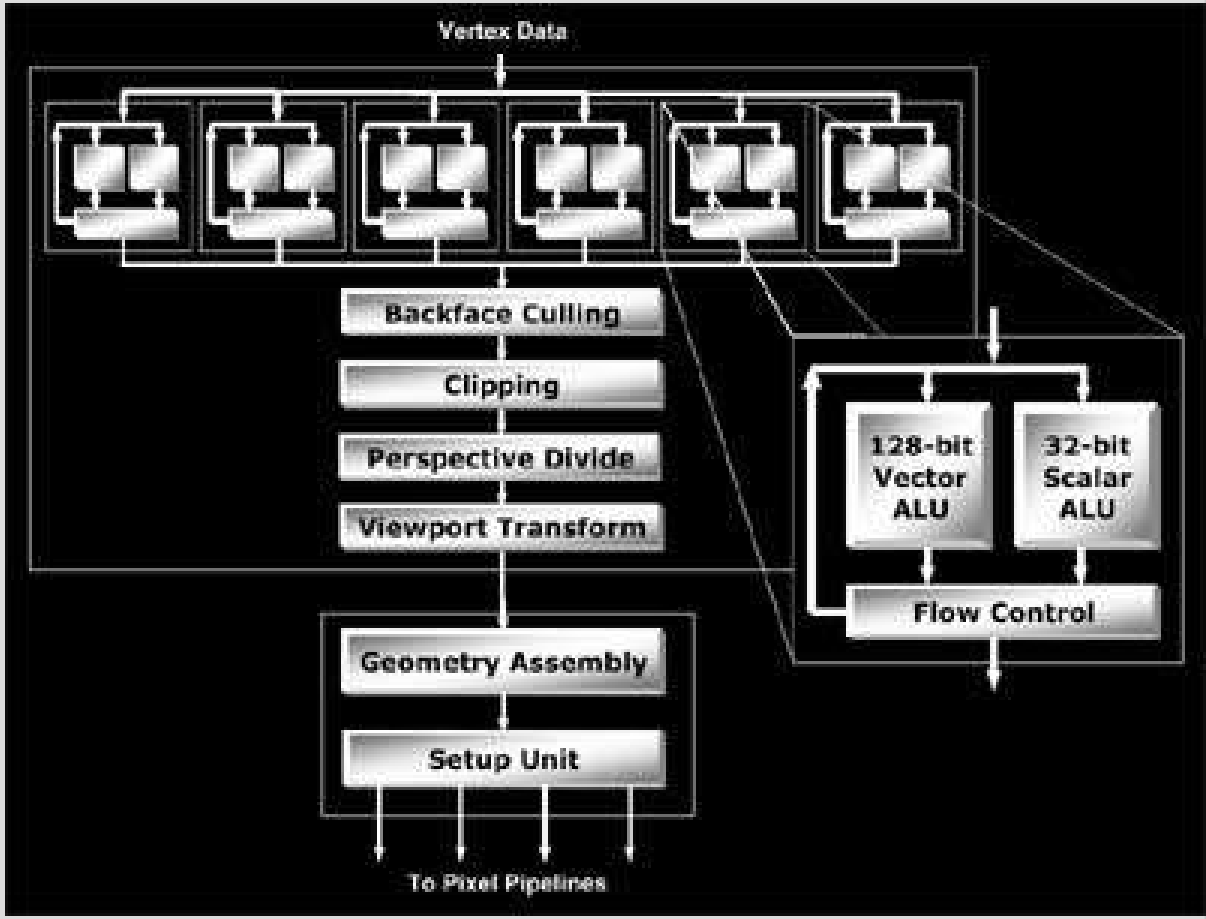
- Implementierung 2:

```
mul r0.x,r5.x,c2.x
mul r4,r3,c1
exp r1.y,c1.y
add r2,r0,r1
```

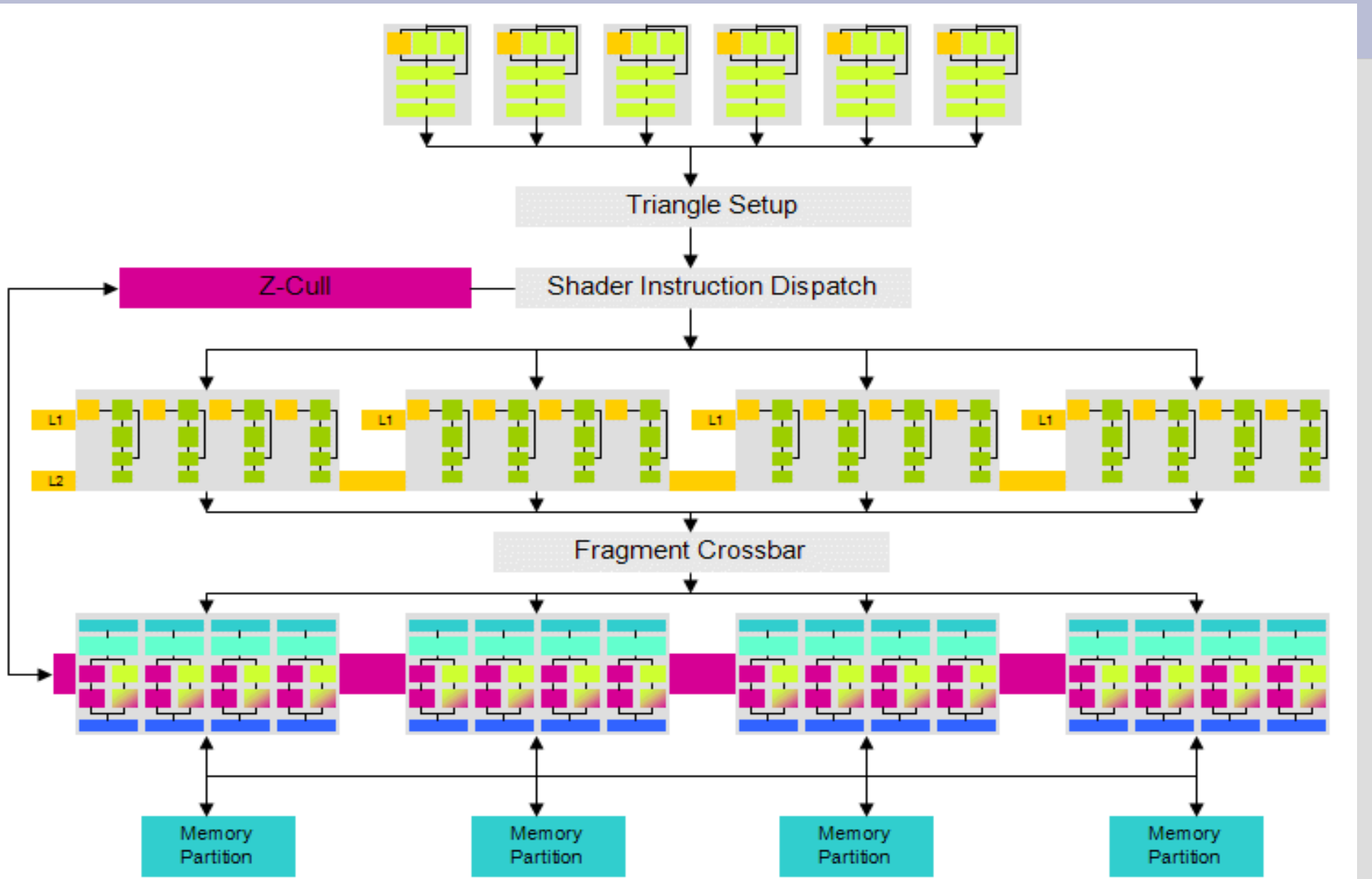
Berechnungsdauer: 2 Takte

Vergleich - Vertexshader ATI R420

- Ähnlicher Aufbau
- Unterschiede:
  - Keine dynamischen Sprünge
  - Nur konstante Sprünge möglich
  - Kein Zugriff auf Texturen in der Vertexeinheit möglich



GPU Architektur



Der Pixelshader

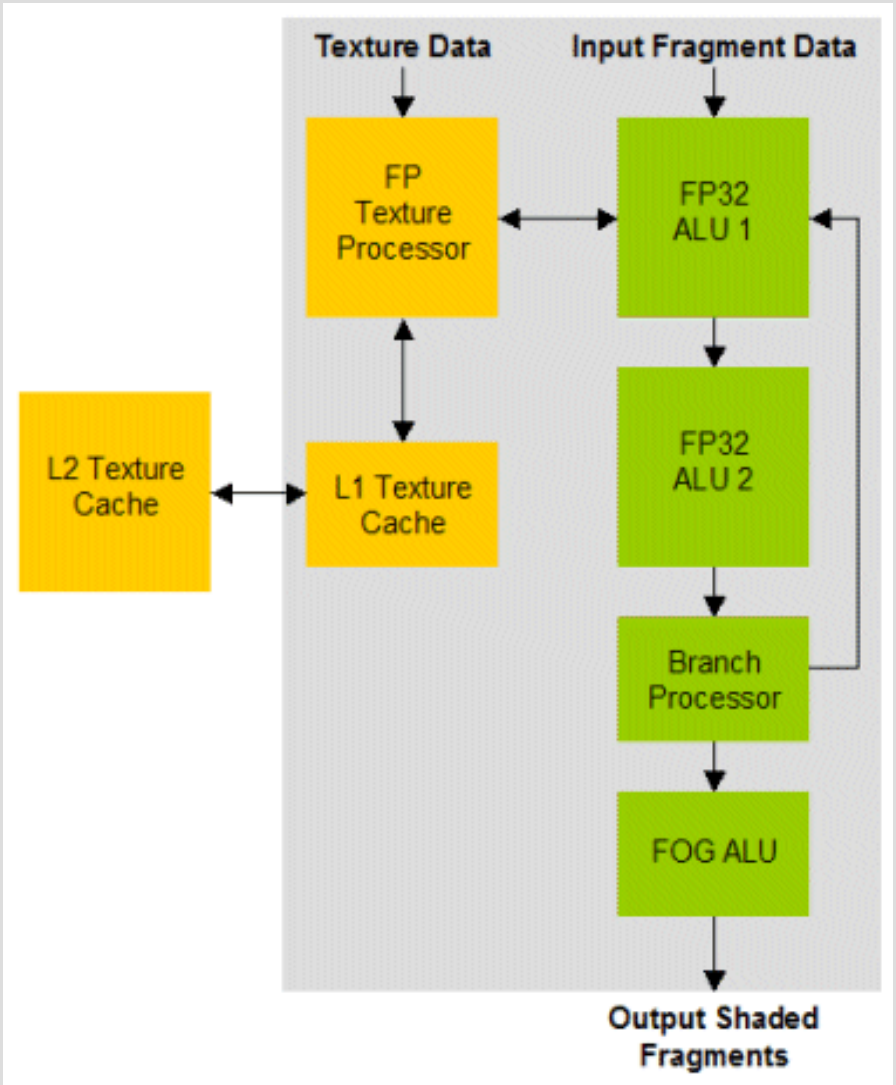
- Pixelshader
  - Versieht geometrische Formen mit Texturen
    - Färbungen
    - Maserungen
    - Oberflächenstrukturen
  - Berücksichtigt Transparenzeffekte
    - Glas
    - Wasser
  - Berechnet visuelle Auswirkung des Lichteinfalls
    - Reflexionen
    - Schatten
    - Spiegelungen
  - Relativ frei programmierbar

Programmierbarer Pixelshader

- NV40 implementiert als erste GPU das Pixel-Shader-Modell 3.0
  - Programmlänge: 512 Befehle oder mehr
  - Variablen-Register: 32
  - Konstanten-Register: 224
  - Verschachtelungstiefe von Schleifen: 24

Pixelshader - Aufbau

- ALU 1 und 2 sind unterschiedlich:
  - ALU1 führt arithmetische oder Textur-Berechnungen aus
  - ALU2 rein arithmetisch
- Optimale Auslastung nur bei geschicktem Wechsel beider Berechnungsarten gewährleistet
- Programmierer bzw. Compilerbauer sind hier stärker gefordert



Optimierungsbeispiel

- Rechenopertion:
  $a^2 * 2^b$

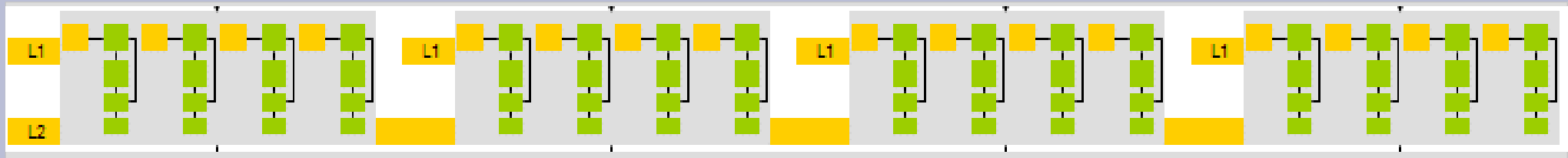
- Implementierung 1:

```
mul r0,a,a
exp r1,b
mul r2,r0,r1          Berechnungsdauer: 3(2) Takte
```

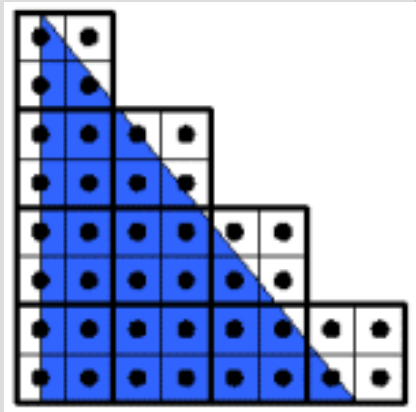
- Implementierung 2:

```
exp r1,b
mul r0,a,a
mul r2,r0,r1          Berechnungsdauer: 2(1) Takte
```

Effizienz der Pixel-Shader

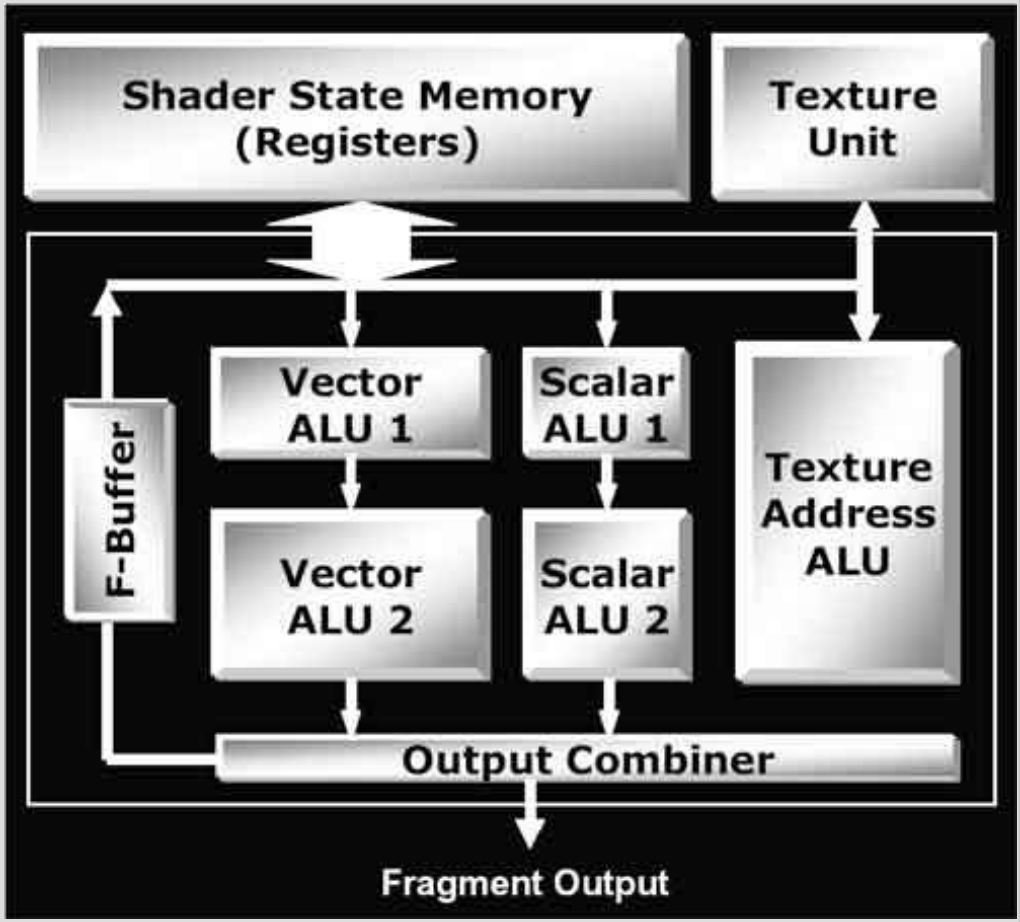


- 16 Pixel-Pipelines sind nicht unabhängig
  - Sind in vierer Blöcke unterteilt
  - Jeder Block berechnet 2x2 Pixel
- Vorteilhaft bei großen Flächen
- Nachteilhaft an Objekträndern weil unsichtbare Pixel die Auslastung verringern

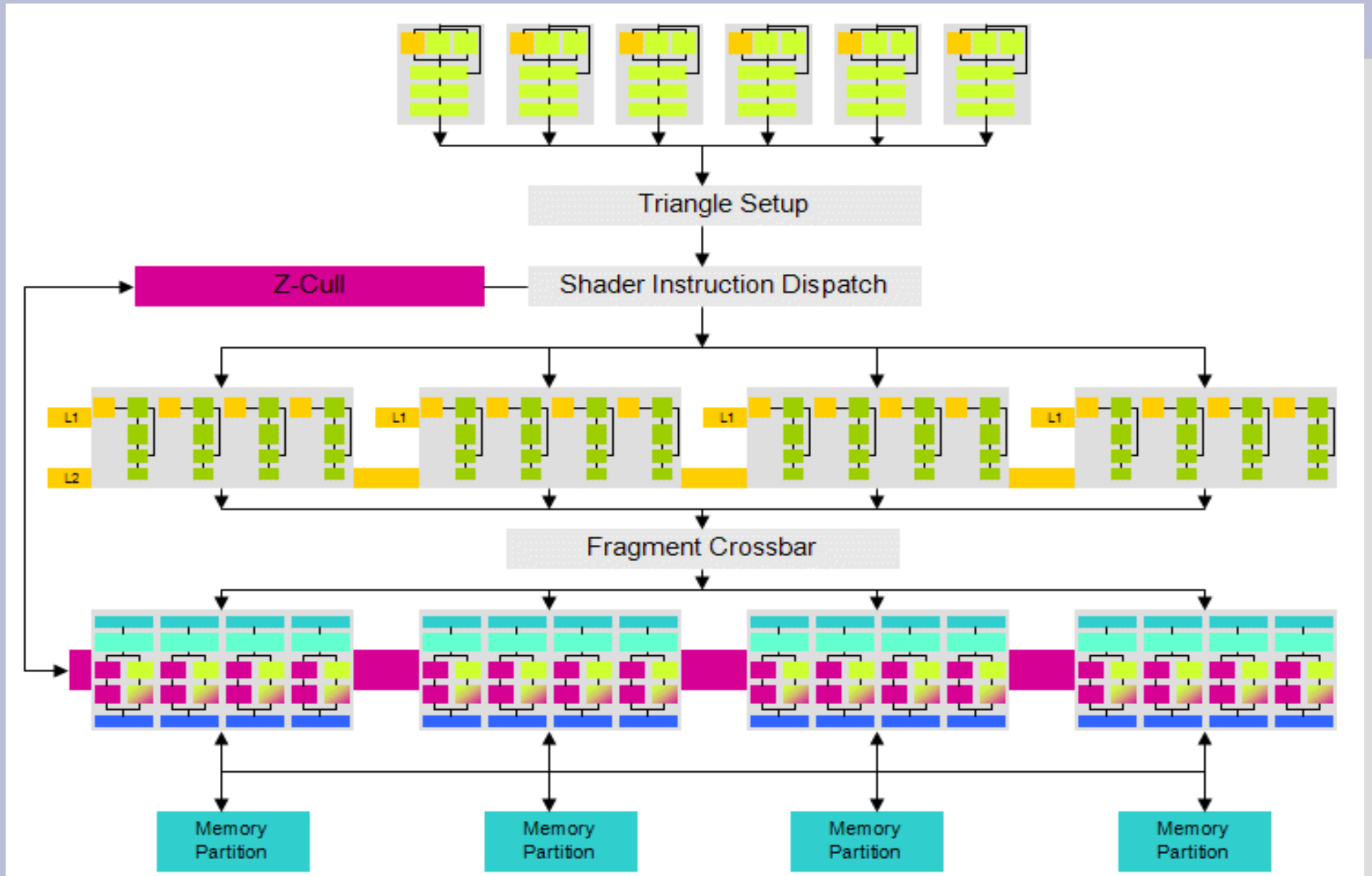


Vergleich - Pixelshader ATI R420

- Ähnlicher Aufbau
- Unterschiede:
  - Getrennte Berechnung von Textur und Arithmetik
  - Jeweils zwei ALUs für Vector- und Scalaroperationen
  - Die zweite ALU ist funktional stark eingeschränkt



GPU Architektur



Pixel Engines (ROP)

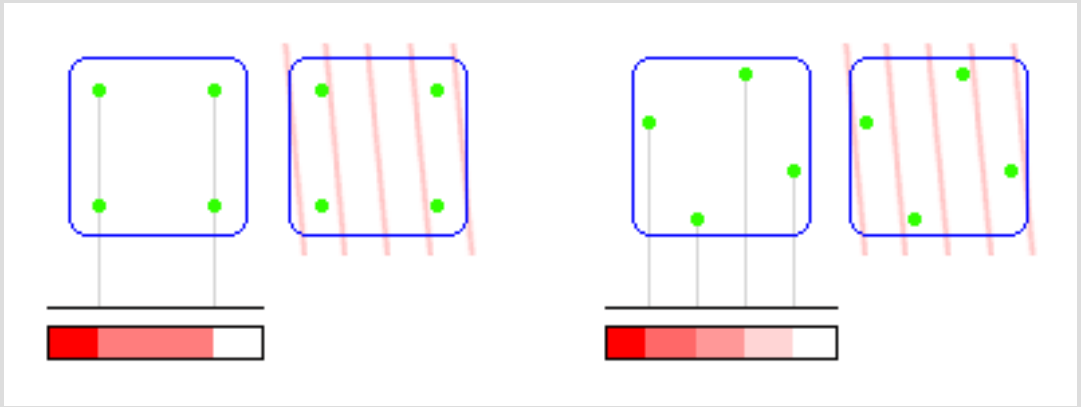
- Führen Effekte auf dem Gesamtbild aus
  - Antialiasing

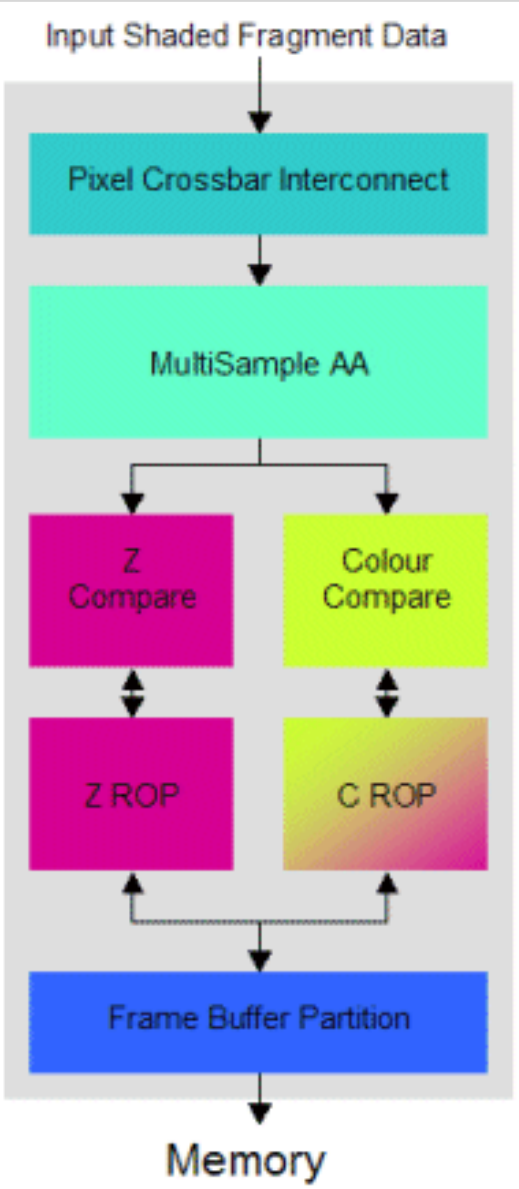
    Beseitigt den Treppeneffekt

2x 

4x 

Rotated Grid-Antialiasing





Ausblick

- Noch mehr Pipelines
- Höhere Speicherbandbreite
- Höhere Takte
- Effizientere Compiler

- Größerer Energiebedarf
- Daher aufwendige Kühlung notwendig
- Starke Netzteile erforderlich
- ggf. kurze Akkulaufzeiten

Referenzen

- www.nvidia.com
- www.ati.com

- www.beyond3d.com
- www.xbitlabs.com
- www.digit-life.com