Grafikprozessor (GPU)

Ein Grafikprozessor (Graphics Processing Unit, GPU) ist eine spezialisierte elektronische Schaltung, die entwickelt wurde, um die Berechnung und Darstellung von Bildern und Grafiken auf einem Computerbildschirm zu beschleunigen. GPUs sind besonders gut geeignet für parallele Verarbeitung großer Datenmengen, was sie ideal für grafikintensive Anwendungen wie Videospiele, 3D-Modellierung und wissenschaftliche Simulationen macht.

Leistungsfähigkeit von GPUs

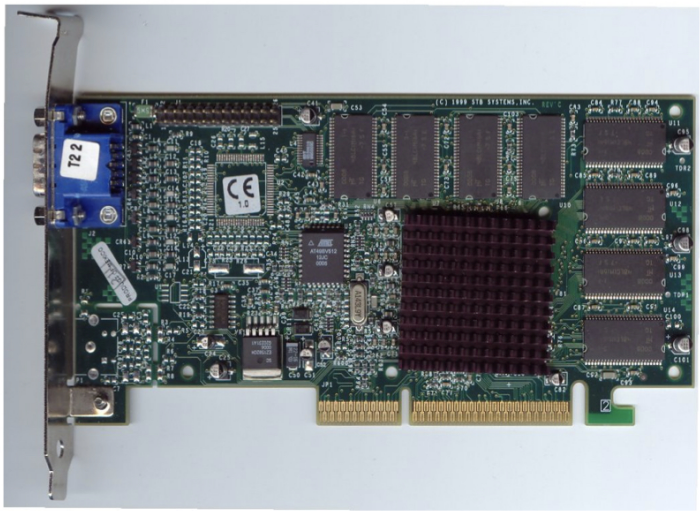
Die Leistungsfähigkeit von GPUs wird oft in Teraflops (TFLOPS) gemessen, was die Anzahl der Billionen Gleitkommaoperationen pro Sekunde angibt, die sie ausführen können. Moderne GPUs können je nach Modell und Anwendungsbereich unterschiedliche Leistungsstufen erreichen.

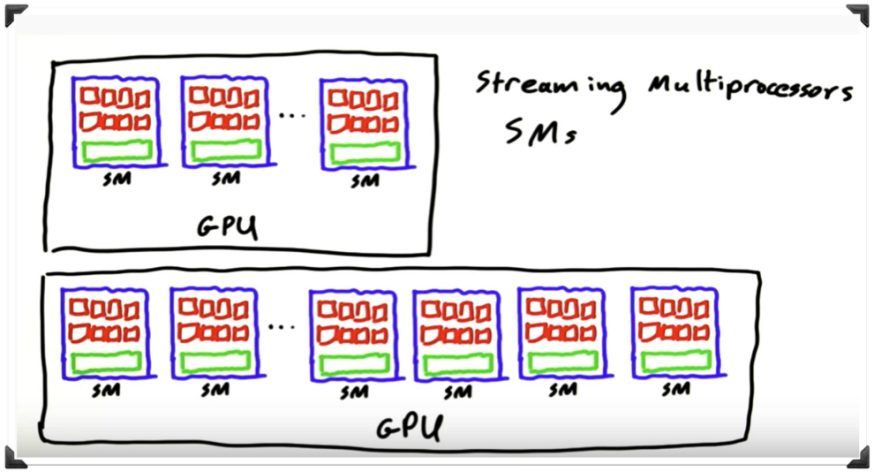
GPU-Klasse	FP32-Leistung	FP64-Leistung

Aufbau und Funktionsweise einer GPU

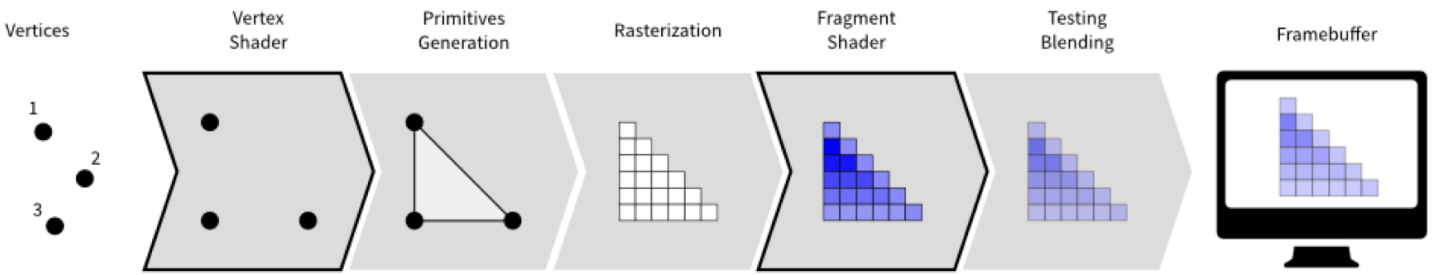
- **Architektur:** Viele einfache Rechenkerne in *Streaming Multiprocessors (SM)*, optimiert für parallele Berechnung.
- **Speicherhierarchie:**
  - *Global Memory (VRAM)*: Großer, langsamer Speicher für die gesamte GPU.
  - *Shared Memory*: Schneller Zwischenspeicher pro SM.
  - *Register*: Klein, extrem schnell, lokal pro Kern.
- **SIMD-Prinzip:** Gleiche Operation auf mehrere Daten gleichzeitig (Single Instruction, Multiple Data).
- **Thread-Modell:** Threads in *Thread-Blocks* organisiert, diese in einem *Grid*.
- **Spezialisierte Hardware:** Rasterisierung, Texturierung und Shading für Grafikoperationen.

GPU

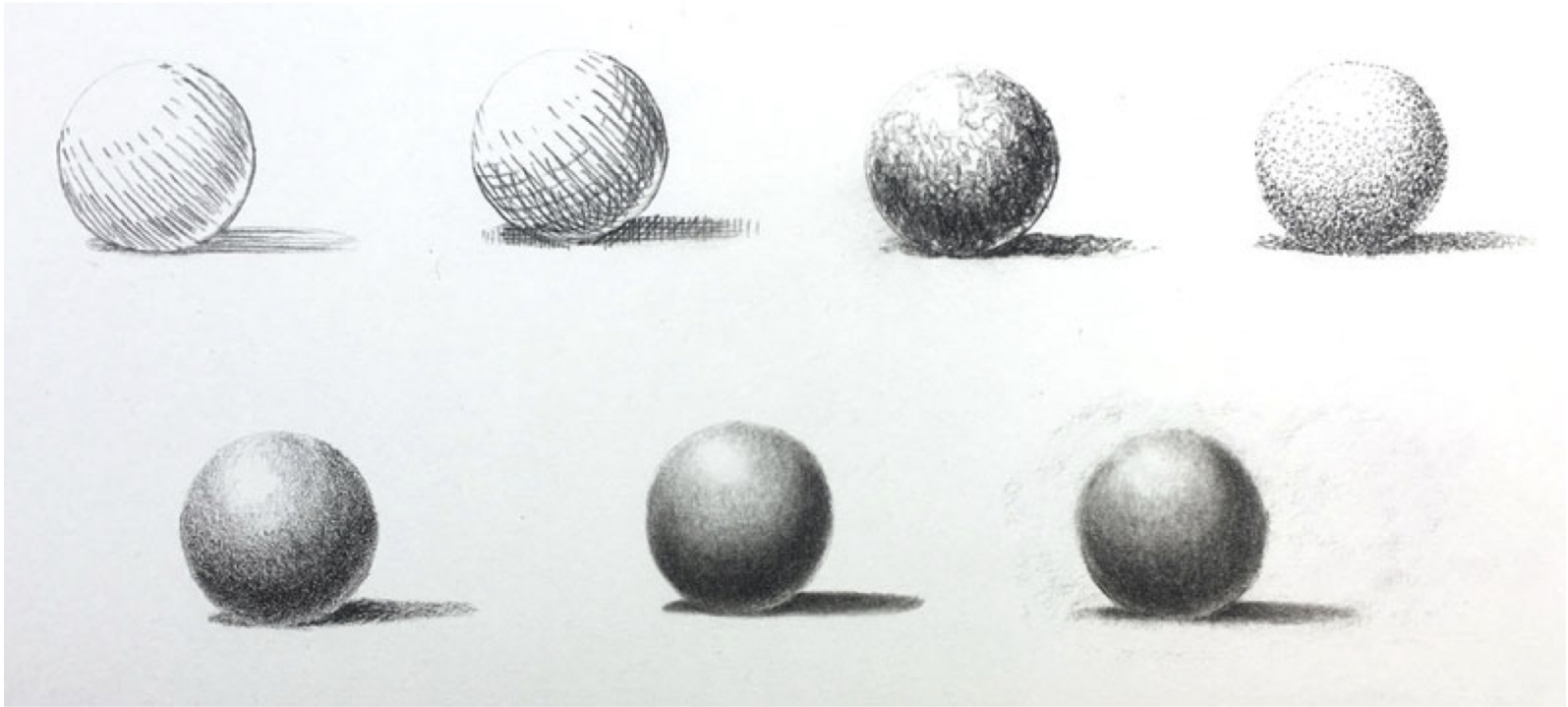



Streaming Multiprocessors
SMs
SM
SM
SM
GPU
SM
SM
SM
SM
SM
SM
GPU

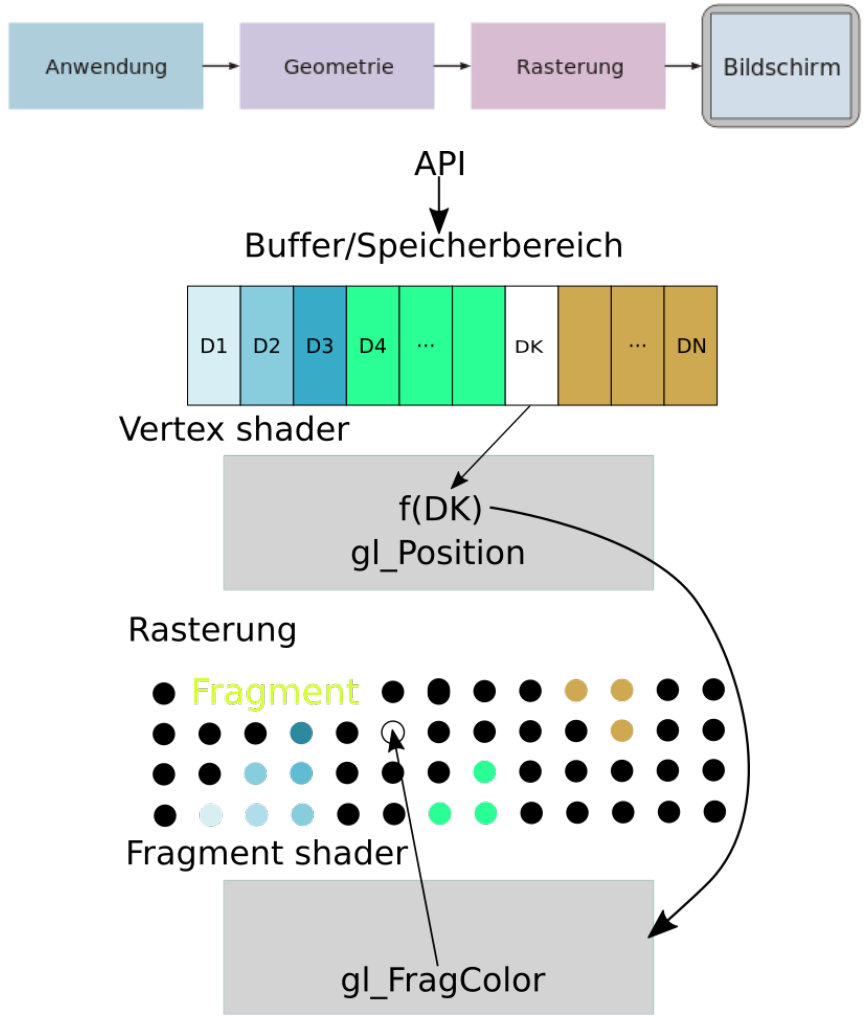
Vertex und Fragmentshader



Shader=Schattierer



Shaderprogramm



OpenGL Pipeline

```
<script id="2d-vertex-shader" type="x-shader/x-vertex">
    attribute vec2 a_position;
    uniform float t;
    varying float T;
    void main() {
    //  gl_Position = vec4(a_position, 0.0, t);
    T = t;
    gl_Position = vec4(a_position[0], a_position[1], 0.0, 1.0);
    }
</script>

<script id="2d-fragment-shader" type="x-shader/x-fragment">
    precision mediump float;
    varying float T;
    void main() {
    gl_FragColor = vec4(0.0 ,1.0,0.0,1.0);
    }
</script>
```

HTML-Grundstruktur

Beispielprogramm minimal_fragmentshader.html

Beispielrogramm zur Darstellung eines grünen Dreiecks in WebGL.
Vergleiche dazu das Programm minimal_fragmentshader.html

- Der Code beginnt mit einer einfachen HTML-Struktur.
- Das `<canvas>` Element wird verwendet, um den Zeichenbereich für WebGL zu definieren.
- Zwei `<script>` Tags enthalten den Vertex- und Fragment-Shader, die die Zeichenoperationen steuern.

Vertex Shader

- Der Vertex-Shader bestimmt die Position der Eckpunkte auf der Leinwand.
- `gl_Position` legt die Position jedes Vertex im Raum fest.

```
<script id="2d-vertex-shader" type="x-shader/x-vertex">
    attribute vec2 a_position;
    void main() {
        gl_Position = vec4(a_position.x, a_position.y,
    }
</script>
```

Fragment Shader

- Der Fragment-Shader legt die Farbe der Pixel fest, die die Dreiecke füllen.
- In diesem Fall wird die Farbe auf Grün gesetzt: `vec4(0.0, 1.0, 0.0, 1.0)`.

```
<script id="2d-fragment-shader" type="x-shader/x-fragme
    precision mediump float;
    void main() {
        gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
    }
</script>
```