

Echzeit Darstellung

Echtzeit-Typ	Eigenschaften
Harte Echtzeit	Zeitlimits zwingend, Systemfehler bei Verpassen.
Weiche Echtzeit	Kleine Abweichungen erlaubt, Leistung sinkt bei Überschreitung.

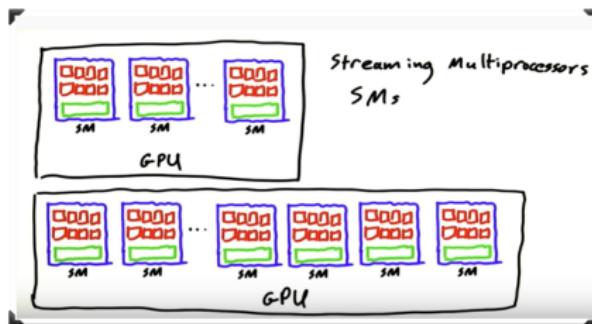
Echzeit Darstellung

Standard: 60 Frames pro Sekunde bei einer Bildschirmauflösung von $4K = 3840 \times 2160$ Pixel. D.h. 497.664.000 Farbwerte müssen pro Sekunde berechnet und an das Ausgabegerät geschickt werden. Kombination von Soft- und Hardware nötig.

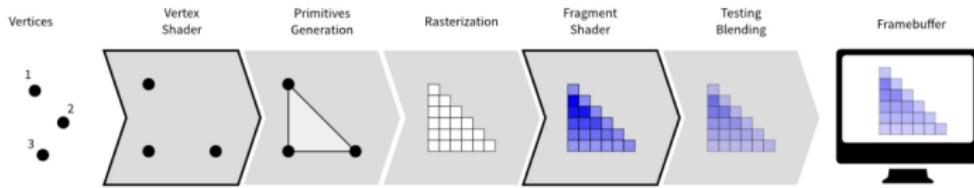
GPU-Klasse	FP32-Leistung	FP64-Leistung
Einsteiger- / Mittelklasse	2 - 10 TFLOPS	0,1 - 1 TFLOPS
High-End-Gaming	10 - 20 TFLOPS	1 - 5 TFLOPS
Workstation- / AI-GPUs	20 - 100 TFLOPS	5 - 20 TFLOPS
Spezialisierte Hochleistung	Mehrere 100 TFLOPS	Mehrere 20+ TFLOPS

- **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.

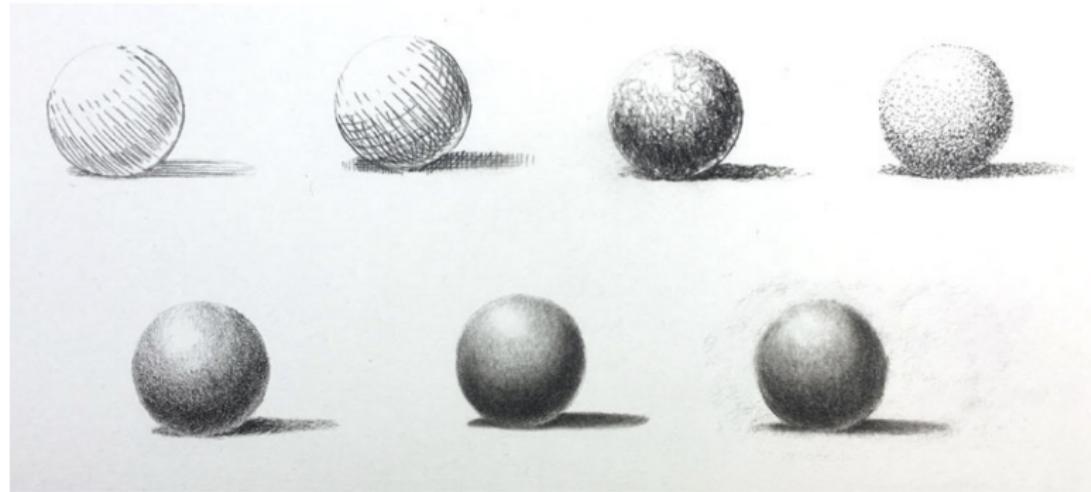
GPU



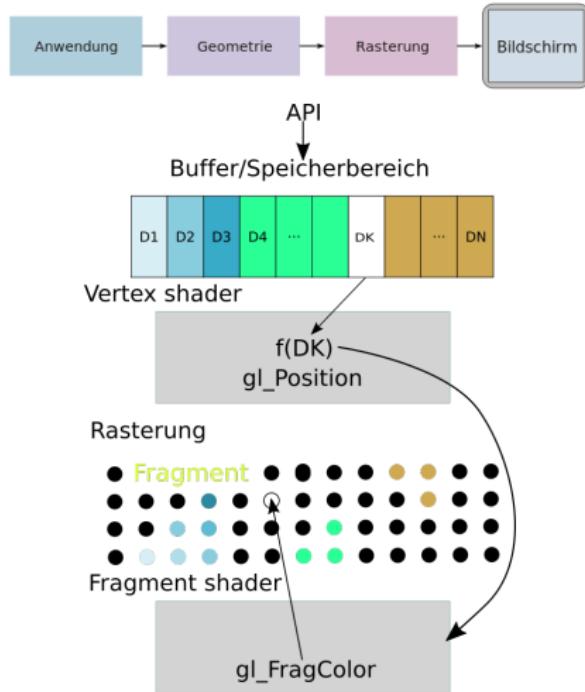
Vertex und Fragments shader



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>
```

Beispielprogramm minimal_fragmentshader.html

Beispielprogramm 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.

- 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>
```

- 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-fragment">
    precision mediump float;
    void main() {
        gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);
    }
</script>
```

- Nach dem Laden des Dokuments wird der WebGL-Kontext erstellt.
- Die Shader werden kompiliert und das Programm wird verknüpft.

```
gl = canvas.getContext("webgl");
shaderScript = document.getElementById("2d-vertex-shader");
vertexShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertexShader, shaderScript.text);
gl.compileShader(vertexShader);
```

Erstellen und Binden des Buffers

- Ein Buffer wird erstellt, um die Vertex-Daten zu speichern.
- Die Positionen der Eckpunkte des Dreiecks werden in den Buffer geladen.

```
gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
gl.bufferData(gl.ARRAY_BUFFER,
    new Float32Array([
        -1.0, -1.0,
        1.0, -1.0,
        -1.0, 1.0,
        -1.0, 1.0,
        1.0, -1.0,
        1.0, 1.0]),
    gl.STATIC_DRAW);
```

Zeichnen des Dreiecks

- Die Vertex-Attribute werden aktiviert und an den Shader übergeben.
- Schließlich wird das Dreieck mit `gl.drawArrays()` gezeichnet.

Animation



Animation

Bewegungsabläufe werden durch eine Abfolge von statischen Bildern erzeugt, die so schnell nacheinander angezeigt werden, dass im Gehirn der Eindruck einer kontinuierlichen Bewegung entsteht. Diese Abfolge von Bildern wird als Frames bezeichnet. Die Illusion einer flüssigen Bewegung wird durch ein Phänomen in unserem visuellen System erzeugt, das als "Bewegungs-Nachbilder" oder auch "Persistence of Vision" bekannt ist.

- **Illusion im Gehirn:**
 - **Verarbeitung im Gehirn:**
 - Aufeinanderfolgende Bilder werden kombiniert.
 - Übergänge werden als kontinuierliche Bewegung interpretiert.
 - **Bewegungswahrnehmung:** Spezialisierte Neuronen erkennen Bildunterschiede als Bewegung.
- **Bildrate (Frames per Second, FPS):**
 - **Unter 12 FPS:** Einzelbilder werden als ruckartig wahrgenommen.
 - **Ab 24 FPS:** Flüssige Bewegung, typisch für Filme.
 - **60 FPS:** Realistische Darstellung, z.B. in Computerspielen.

Chronofotografie

Die Chronofotografie (auch Fotochronografie) bezeichnet die fotografische Dokumentation von Bewegungen oder Prozessen, heute hauptsächlich als Hochgeschwindigkeitsfotografie.

Chronofotografie

In der Entwicklung der Fotografie wurden in den 1870er und 1880er Jahren durch empfindliche Photomaterialien und schnelle Kameraverschlüsse sogenannte Augenblicks- oder Momentfotografien möglich, Aufnahmen bewegter Objekte.

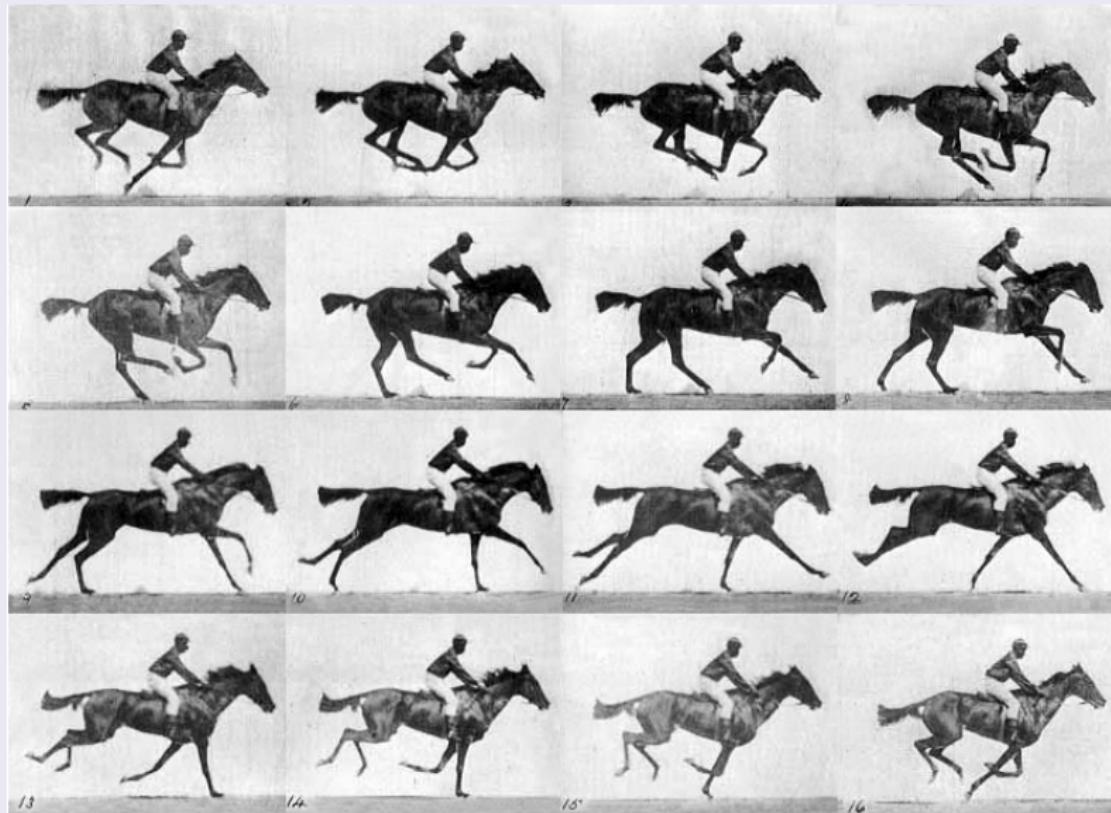
Chronofotografie

Muybridge gelang 1878 der Nachweis, dass ein Pferd im Galopp kurzzeitig mit allen vier Hufen vom Boden abhebt. Diese frühen Serienaufnahmen lieferten wichtige Impulse für die Entwicklung der bewegten Bilder und waren damit auch Vorläufer des Kinofilms.



Echzeit Darstellung

Sprite Animation



Was ist Sprite Animation?

- Sprite-Animation ist eine Technik, bei der Bilder (Sprites) in einer Sequenz angezeigt werden, um Bewegung darzustellen.
- Ein Sprite-Sheet ist ein Bild, das mehrere Sprites in einem Raster enthält.
- Jedes Sprite repräsentiert eine einzelne Bildsequenz (Frame) der Animation.
- Die Bilder werden schnell hintereinander gezeigt, um die Illusion von Bewegung zu erzeugen.

- Ein Sprite-Sheet wird in Reihen und Spalten unterteilt, wobei jeder Abschnitt einen Frame der Animation darstellt.
- Eine Schleife wechselt durch die Frames des Sprite-Sheets basierend auf einer Bildrate (fps).
- Die Position und Skalierung des Sprites können ebenfalls animiert werden.

Vorteile der Sprite Animation

- Spart Speicherplatz, da mehrere Animationen in einem Bild gespeichert werden können.
- Effizient für Spiele und Anwendungen, die viele kleine Animationssequenzen benötigen.
- Unterstützt einfache Bewegungsanimationen ohne aufwendige Rechenleistung.

Beispielprogramm SpriteAnimator

Vergleiche dazu das Programm SpriteAnimator

- Das Skript verwendet die HTML5 <canvas> API, um eine Sprite-Animation auf einer Leinwand zu zeichnen.
- Der <canvas> wird über CSS mit einer Breite von 280px und einer Höhe von 133px gestylt.
- Ein Tag lädt das Bild, das als Sprite-Sheet für die Animation verwendet wird.

- Das Canvas-Element hat eine festgelegte Breite und Höhe sowie eine Randlinie.

```
<style>
    #myCanvas {
        width: 280px;
        height: 133px;
        border: 2px solid #000000;
    }
</style>
```

- Die Animation-Klasse wird erstellt, um die Animation zu handhaben.
- Sie nimmt mehrere Parameter entgegen: das Bild, die Bildrate (fps), die Breite und Höhe des Sprites, die Skalierung, die Anzahl der Zeilen und Spalten im Sprite-Sheet, sowie die Position des Sprites auf dem Canvas.

- Die update-Methode aktualisiert den aktuellen Sprite-Index basierend auf der verstrichenen Zeit.
- Sie berechnet, wann der nächste Frame basierend auf den fps angezeigt werden soll.

```
Animation.prototype.update = function(dt) {  
    this.elapsedTime += dt;  
    if (1000.0 / this.fps - this.elapsedTime < 0.0) {  
        this.index += 1;  
        this.index = this.index % (this.nRows * this.nC  
        this.elapsedTime = 0.0;  
    }  
};
```

Draw-Methode der Animation

- Die draw-Methode zeichnet den aktuellen Frame der Animation.
- Sie berechnet die Position des aktuellen Frames auf dem Sprite-Sheet und zeichnet das entsprechende Bild auf das Canvas.

```
Animation.prototype.draw = function(ctx) {  
    var i = Math.floor(this.index / this.nColls);  
    var j = this.index % this.nColls;  
    ctx.drawImage(this.spriteSheet,  
        j * this.width,  
        i * this.height,  
        this.width,  
        this.height,  
        this.posX,  
        this.posY,  
        this.scale * this.width,  
        this.scale * this.height);
```

- Der gameLoop wird kontinuierlich ausgeführt, um die Animation zu aktualisieren und zu rendern.
- Der Loop nutzt `requestAnimationFrame`, um die Bildrate zu steuern und die Animation flüssig darzustellen.
- Jede Iteration aktualisiert die Animation und zeichnet den neuen Frame auf das Canvas.

Code des Game Loops

```
function gameLoop() {  
    window.requestAnimationFrame(gameLoop);  
    ctx.clearRect(0, 0, canvas.width, canvas.height);  
  
    d = new Date();  
    var currentTime = d.getTime();  
    var dt = currentTime - prevTime;  
    anim.update(dt);  
    anim.draw(ctx);  
    prevTime = currentTime;  
}
```