

# Rainer Duda

⌘ presents ⌘

Zertifikat \*  
"Digital Games"  
approved

**Fun** with Echtzeit-Computergrafik → **reloaded**

WP-3P | Optimierungen

- Praktische Vertiefung
  - JS Strukturanpassung (**main.js**)
    - HTML Cleanup
    - Implementierung eines Element Buffers (*Beschreibung des Würfels mithilfe von Indices*)
    - Einbindung von glmatrix
      - Perspektivische Projektionsmatrix
      - Transformationsmatrix

- Wie können wir die HTML Datei optimieren?
  - Erstellung des Canvas in die JS Datei überführen!
    - HTML Datei benötigt keine Canvas Style Informationen.
    - HTML Datei benötigt keinen vordefinierten Canvas
  - Per Zugriff über das DOM wird ein Canvas innerhalb des Bodies erzeugt!

```
/* ===== Erzeugung eines WebGL Kontext ===== */

//const canvas = document.getElementById('my_canvas');

var canvas = document.createElement('canvas')
canvas.width = window.innerWidth
canvas.height = window.innerHeight
document.body.appendChild(canvas)

const gl = canvas.getContext('webgl2');

gl.enable(gl.DEPTH_TEST);
gl.enable(gl.CULL_FACE);

if (!gl) {
    console.log('WebGL nicht verfügbar!');
} else {
    console.log('WebGL verfügbar!');
}
```

# Strukturerweiterung (main.js) – Implementierung eines Element Buffers

- Wie können wir die Bereitstellung unseres Würfels optimieren?
  - Durch den Einsatz von Indices!
  - Wir definieren lediglich 8 Eckpunkte.
  - Wir definieren jede Würfelseite basierend auf 2 Dreiecken.
  - Wir benutzen Eckpunkte mehrmals für die Definition!

```
// Würfeloptimierung
//      v6----- v5
//      /|         |\
//      v1-----v0|
//      | |         | |
//      | |v7---|---v4
//      |/         |\
//      v2-----v3

var w_vertices = [
    1, 1, 1, 1, 1,1,1,1,
    -1, 1, 1, 1, 1,0,0,1,
    -1,-1, 1, 1, 0,1,0,1,
    1,-1, 1, 1, 0,0,1,1,
    1,-1,-1, 1, 0,1,1,1,
    1, 1,-1, 1, 1,1,0,1,
    -1, 1,-1, 1, 1,0,1,1,
    -1,-1,-1, 1, 0,0,0,1,
];

var w_indices = new Uint8Array([
    0, 1, 2, 0, 2, 3, // Front
    0, 3, 4, 0, 4, 5, // Rechts
    0, 5, 6, 0, 6, 1, // Oben
    1, 6, 7, 1, 7, 2, // Links
    7, 4, 3, 7, 3, 2, // Unten
    4, 7, 6, 4, 6, 5 // Hinten
]);
```

# Strukturerweiterung (main.js) – Implementierung eines Element Buffers

- Wie können wir die Bereitstellung unseres Würfels optimieren?
  - Wir müssen den neuen "Element" Buffer benutzen/implementieren!
  - Und den Buffer mit den Indices befüllen!

```
var w_indices = [  
    0, 1, 2, 0, 2, 3, // Vorne  
    0, 3, 4, 0, 4, 5, // Rechts  
    0, 5, 6, 0, 6, 1, // Open  
    1, 6, 7, 1, 7, 2, // Links  
    7, 4, 3, 7, 3, 2, // Unten  
    4, 7, 6, 4, 6, 5 // Hinten  
];  
  
/* ==== Definition des Element Puffers ==== */  
  
const indexBuffer = gl.createBuffer();  
  
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, indexBuffer);  
  
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(w_indices)  
    , gl.STATIC_DRAW);
```

# Strukturerweiterung (main.js) – Implementierung eines Element Buffers

- Können wir schon den Würfel rendern? Nein!
  - Wir müssen das VAO Objekt anpassen!
  - Haben wir ein neues Array im Einsatz?

```
var w_vertices = new Float32Array([
    1, 1, 1, 1, 1, 1,
    -1, 1, 1, 1, 0, 0,
    -1, -1, 1, 0, 1, 0,
    1, -1, 1, 0, 0, 1,
    1, -1, -1, 0, 1, 1,
    1, 1, -1, 1, 1, 0,
    -1, 1, -1, 1, 0, 1,
    -1, -1, -1, 0, 0, 0
]);

/* ==== Definition von Front-Face Puffern ==== */

const vertex_buffer = gl.createBuffer();

gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);

gl.bufferData(gl.ARRAY_BUFFER, w_vertices, gl.STATIC_DRAW);
```

# Strukturerweiterung (main.js) – Implementierung eines Element Buffers

- Können wir schon den Würfel rendern? Nein, immer noch nicht!
  - Haben wir ein neues Array im Einsatz? – Ja!
  - Aber? Die Anzahl der Informationen hat sich reduziert!
  - Die Verknüpfung der Attribute mit den Shadern überprüfen und ggf. anpassen!

```
/* ==== Verknüpfung der Attribute mit dem Vertex Shader ==== */  
  
const positions_attribut = gl.getAttribLocation(shader_program, 'wuerfel_position');  
gl.vertexAttribPointer(positions_attribut, 3, gl.FLOAT, false, 24, 0);  
gl.enableVertexAttribArray(positions_attribut);  
  
const farb_attribut = gl.getAttribLocation(shader_program, 'wuerfel_eingabefarbwerte');  
gl.vertexAttribPointer(farb_attribut, 3, gl.FLOAT, false, 24, 12);  
gl.enableVertexAttribArray(farb_attribut);
```

# Strukturerweiterung (main.js) – Implementierung eines Element Buffers

- Können wir schon den Würfel rendern? Nein, immer noch nicht! – Aber gleich!
- **gl.drawElements** verwendet einen mit Vertex-Indizes gefüllten Buffer!
- Wir ersetzen gl.drawArrays!
- Und teilen der Funktion die Anzahl der Indices mit.

```
const mode = gl.TRIANGLES;
const first = 0;
const count = w_indices.length;
const offset = 0;

var indexType = gl.UNSIGNED_SHORT;
gl.drawElements(mode, count, indexType, offset);
```



- Können wir nicht auch die Projektionsmatrix  
nebst Transformationsmatrix vereinfachen?
- Vorerst passen wir unsere Variablen  
namenstechnisch im Vertex Shader an.

```
const vertex_source = `

    attribute vec4 wuerfel_position;
    attribute vec3 wuerfel_eingabefarbwerte;
    varying vec4 wuerfel_farbwerte;
    uniform mat4 model_view_matrix;
    uniform mat4 projection_matrix;

    void main(){

        gl_Position = projection_matrix * model_view_matrix * wue
rfel_position;
        wuerfel_farbwerte = vec4(wuerfel_eingabefarbwerte, 1);

    }

`;
```

- Können wir nicht auch die Projektionsmatrix  
nebst Transformationsmatrix vereinfachen?
  - Vorerst passen wir unsere Variablen  
namenstechnisch im Vertex Shader an.
  - Danach müssen neben dem Import der Skriptdatei  
im HTML Dokument, dass glMatrix Objekt  
dekonstruiert werden.

```
/* ==== glMatrix-Objekt Dekonstruktion ==== */  
  
const { mat2, mat3, mat4, vec2, vec3, vec4 } = glMatrix;
```

- Können wir nicht auch die Projektionsmatrix nebst Transformationsmatrix vereinfachen?
  - In der main.js Datei definieren wir nun für uns gängige Werte, die zur Bereitstellung der Projektion benötigt werden.
  - Wir fügen die Werte in die glmatrix Funktion ein und speisen die Werte in den Vertex Shader!
  - Et Voila!

```
/* ==== Definition einer perspektivischen Projektionsmatrix ====  
*/  
  
const field_of_View = 45 * Math.PI / 180;  
const aspect = gl.canvas.clientWidth / gl.canvas.clientHeight;  
const z_near = 0.1;  
const z_far = 100.0;  
const projection_mat = mat4.create();  
  
mat4.perspective(projection_mat, field_of_View, aspect, z_near, z_far);  
  
const projectUniform = gl.getUniformLocation(shader_program, "projection_matrix");  
gl.uniformMatrix4fv(projectUniform, false, projection_mat);
```

- Können wir nicht auch die Projektionsmatrix nebst Transformationsmatrix vereinfachen?
  - Wir erzeugen abermal eine Identitätsmatrix für die View-Matrix.
  - Danach definieren wir unsere Betrachtungsposition.
  - Die Position wird an den Vertex Shader übergeben.
  - Et Voila!

```
/* ==== Definition einer Transformationsmatrix ==== */  
  
const view = mat4.create();  
  
var translation = vec3.create();  
vec3.set(translation, 0, 0, -6.0);  
mat4.translate(view, view, translation);  
  
const viewUniform = gl.getUniformLocation(shader_program, "model  
_view_matrix");  
gl.uniformMatrix4fv(viewUniform, false, view);
```

