OpenGL-Praktikum

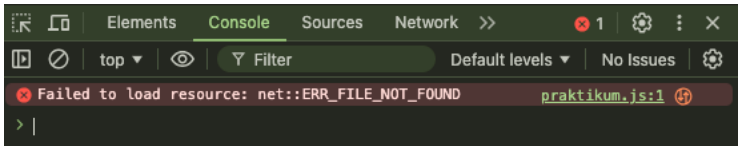
GL1:

b.

• Fehlende **Verknüpfung:**
Die HTML-Datei verweist immer noch auf `praktikum.js`, die es nach der Umbenennung nicht mehr gibt.
→ **JavaScript wird nicht geladen.**

• Fehlermeldung **im Browser:**

• **Würfel wird nicht angezeigt:**
Da das Skript für die Darstellung fehlt, wird die WebGL-Anwendung nicht ausgeführt, und der Canvas-Bereich bleibt leer.



c.

ich habe die Kommentarzeichen bei `requestAnimationFrame(render)` in der `render()`-Funktion entfernt, um die Endlosschleife für das Zeichnen zu starten.

```
function render() {
 cubeRotation();
 drawScene();
requestAnimationFrame(render); // Kommentar entfernen
}
```

d.

(d) Um die Rotation mit einem Button zu starten/stoppen, müssen wir folgende Änderungen vornehmen:

1.Am Anfang des Codes bei den anderen globalen Variablen:
var isRotating = true;

2.Die render-Funktion muss modifiziert werden, damit sie nur rotiert wenn isRotating true ist:

```
var render = function(){
    // Framebuffer und z-Buffer initialisieren
    gl.clear( gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    // Nur rotieren wenn isRotating true ist
    if(isRotating) {
        theta[axis] += 2.0;
    }

    displayScene();

    requestAnimFrame(render);
}
```

3.In der init-Funktion muss der Event-Handler für den neuen Button hinzugefügt werden:

```
document.getElementById("ButtonT").onclick = function(){
    isRotating = !isRotating; // Toggle zwischen true und false
};
```

GL2:

a. Verschieben und Drehen des ersten Würfels

Ziel: Der erste Würfel wurde aus dem Ursprung in das Weltkoordinatensystem verschoben und soll sich weiterhin mit den Buttons in der HTML-Seite drehen lassen. Zusätzlich dreht sich der Würfel um seine eigene Z-Achse.

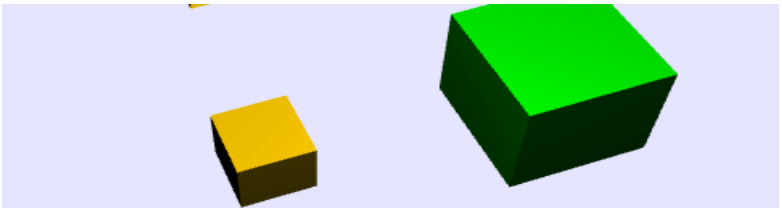
- **Verschiebung:** Der Würfel wurde um 5 Einheiten in die x-Richtung und 1 Einheit in die z-Richtung verschoben. Dies wurde durch eine **Translationstransformation** erreicht:

```
// Definiere die Translation (5.0, 1.0, 0.0)
var translationMatrix = translate(5.0, 1.0, 0.0);
```

- **Drehung:** Die Rotation des Würfels um seine eigene Z-Achse wurde durch Anwendung einer **Rotationstransformation** um die Z-Achse durchgeführt. Um den Würfel in etwa 10 Sekunden für eine vollständige Umdrehung zu drehen, wurde die Rotationsgeschwindigkeit so eingestellt, dass sie über die Zeit animiert wird.
- **Drehung mit Buttons:** Um eine manuelle Steuerung der Drehung mit den HTML-Buttons zu ermöglichen, wurde die Drehmatrix mit den Button-Events verknüpft, um die Drehwinkel zu verändern.

b. **Ziel:** Der zweite Würfel wird in Weltkoordinaten an der Position (5, 0, -3) hinzugefügt, hat eine grüne Farbe und dreht sich doppelt so schnell wie der erste Würfel um seine eigene X-Achse.

- **Positionierung:** Der grüne Würfel wird an der Position (5, 0, -3) im Weltkoordinatensystem gesetzt. Dies erfolgt durch eine **Translationstransformation**:
- **Rotation:** Der grüne Würfel rotiert um seine eigene X-Achse, wobei er die doppelte Geschwindigkeit des ersten Würfels hat. Dies wurde durch eine **Rotationsmatrix** um die X-Achse umgesetzt, und der Rotationswinkel wurde entsprechend verdoppelt:
- **Größe:** Die Größe des grünen Würfels wurde durch eine Skalierung verdoppelt:



c. Zeichnen der ersten Pyramide

Ziel: Eine Pyramide mit rechteckiger Grundfläche wurde hinzugefügt, die die gleiche Farbe wie der erste Würfel hat. Sie soll sich ebenfalls mit den Buttons auf der HTML-Seite drehen lassen.

- **Geometrie:** Die Pyramide hat eine rechteckige Grundfläche mit den Abmessungen 4x2 in der XZ-Ebene. Die Spitze befindet sich an der Position (0, 4, 0).
- **Farbe:** Die Pyramide wurde mit der gleichen Farbe wie der erste Würfel (goldgelb) versehen.
- **Drehung:** Die Drehung der Pyramide wurde ebenfalls durch eine **Rotationsmatrix** realisiert, die in Verbindung mit den Buttons auf der HTML-Seite die Rotation ermöglicht.

d. Zweite Pyramide – Position und Farbe

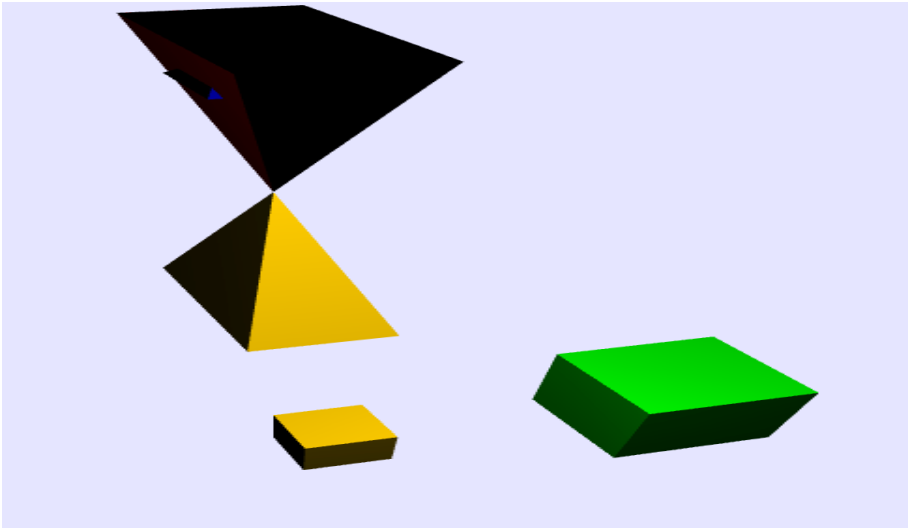
Ziel: Eine zweite Pyramide in roter Farbe wird auf die erste Pyramide gesetzt, sodass ihre Spitze die Spitze der ersten Pyramide berührt.

- **Positionierung:** Die zweite Pyramide wurde so positioniert, dass ihre Spitze genau auf der Spitze der ersten Pyramide sitzt. Die genaue Transformation erfolgte durch eine **Translationstransformation**.
- **Farbe:** Die zweite Pyramide wurde rot gefärbt.

e. Dritte Pyramide – Größe und Position

Ziel: Eine dritte, kleinere Pyramide wird auf der Spitze der roten Pyramide positioniert, mit der Basis genau auf dem Dreieck der roten Pyramide, dass der Kamera zugewandt ist.

- **Größe:** Diese Pyramide hat nur 40% der Größe der anderen Pyramiden. Dies wurde durch eine **Skalierungstransformation** erreicht.
- **Positionierung:** Die dritte Pyramide wurde auf der Spitze der roten Pyramide positioniert. Sie sitzt genau auf dem Dreieck, das der Kamera zugewandt ist.
- **Farbe:** Die dritte Pyramide wurde blau gefärbt.



f. Die Drehung aller Objekte in Ihrer Szene erfolgt durch die Transformationen, die in der Funktion drawScene angewendet werden. Diese Funktion wird für jedes Objekt (Würfel, grüner Würfel, Pyramide, rote Pyramide, blaue Pyramide) aufgerufen und wendet die Rotationen und Translationen auf d ie Modell-Matrix an.

```
// Render-Funktion
var render = function() {
    // Framebuffer und Z-Buffer initialisieren
    gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

    // Nur rotieren wenn isRotating true ist
    if (isRotating) {
        theta[axis] += 2.0;
    }

    displayScene();

    // FPS Berechnung
    counter++;

    // Nur alle 20 Frames FPS berechnen
    if (counter >= 20) {
        // Aktuelle Zeit holen
        var now = Date.now() / 1000;

        // Zeitdifferenz berechnen
        var timeDiff = now - then;

        // FPS = Anzahl Frames / Zeitdifferenz
        var fps = Math.round((counter / timeDiff) * 10) / 10;

        // FPS im HTML Element anzeigen
        document.getElementById("fps").innerHTML = fps + " FPS";

        // Counter zurücksetzen
        counter = 0;

        // Aktuelle Zeit als neue Startzeit setzen
        then = now;
    }

    requestAnimFrame(render);
};

function drawScene(materialDiffuse, translateX, translateY, translateZ) {
    var lighting = true;
    gl.uniform1i(gl.getUniformLocation(program, "lighting"), lighting);

    if (lighting) {
        calculateLights(materialDiffuse);
    }

    model = mat4();
    model = mult(model, translate(translateX, translateY, translateZ));
    model = mult(model, rotate(theta[0], [1, 0, 0]));
    model = mult(model, rotate(theta[1], [0, 1, 0]));
    model = mult(model, rotate(theta[2], [0, 0, 1]));
    gl.uniformMatrix4fv(gl.getUniformLocation(program, "modelMatrix"), false, flatten(model));

    var normalMat: any
    normalMat = mult(view, model);
    normalMat = inverse(normalMat);
    normalMat = transpose(normalMat);
    gl.uniformMatrix4fv(gl.getUniformLocation(program, "normalMatrix"), false, flatten(normalMat));

    gl.drawArrays(gl.TRIANGLES, 0, numVertices);
}
```

GL3:

Erklärung der Änderungen:

- Kameraposition für die x-Achse: Die Kamera wird auf die Position (10.0, 0.0, 0.0) gesetzt, was bedeutet, dass sie 10 Einheiten vom Ursprung entfernt ist und sich auf der x-Achse befindet.
- Kamera schaut zum Ursprung: Der Blickpunkt (vrp) bleibt der Ursprung (0.0, 0.0, 0.0), sodass die Kamera in Richtung der negativen x-Achse schaut.
- Up-Vektor: Der Up-Vektor (upv) bleibt (0.0, 1.0, 0.0), was bedeutet, dass die y-Achse nach oben zeigt.

d.

Wenn du den Öffnungswinkel von 60° auf 30° reduzierst, ändert sich das Bild wie folgt:

1. **Verengung des Sichtfelds:** Das auffälligste Ergebnis ist, dass das Sichtfeld der Kamera enger wird. Du siehst einen kleineren Bereich der Szene. Es ist so, als würdest du durch ein Zoomobjektiv schauen.
2. **Zoom-Effekt:** Die Objekte in der Szene erscheinen näher und größer. Dieser Effekt entsteht, weil weniger von der Umgebung sichtbar ist und die Objekte so einen größeren Teil des Bildes einnehmen.
3. **Veränderte Perspektive:** Obwohl das Bild "herangezoomt" wirkt, bleibt die perspektivische Darstellung erhalten. Die Objekte werden nicht einfach nur vergrößert; die perspektivische Verzerrung (also die Tatsache, dass weiter entfernte Objekte kleiner erscheinen) bleibt bestehen.
4. **Verlust von Details am Rand:** Objekte, die sich zuvor am Rand des Bildes befanden, können nun aus dem Bildausschnitt verschwinden, da das Sichtfeld enger ist.

e.

Das Bild ändert sich wie folgt:

1.Ausschneiden von nahen Objekten: Alle Objekte oder Teile von Objekten, die sich innerhalb eines Abstands von 15 Einheiten zur Kamera befinden, werden nicht mehr gerendert. Sie werden durch die Near-Clipping-Plane abgeschnitten und sind somit nicht mehr sichtbar.

2.Potentieller Verlust großer Teile der Szene: Da in deiner Szene die Objekte typischerweise nicht sehr weit von der Kamera entfernt sind, kann es sein, dass mit einer Near-Clipping-Plane von 15 alle oder zumindest der Großteil der Objekte abgeschnitten werden.

3.Unerwartete Effekte: Wenn du Objekte in der Szene hast, die zwar nicht direkt vor der Kamera liegen, aber sich teilweise innerhalb des Radius von 15 Einheiten befinden, werden

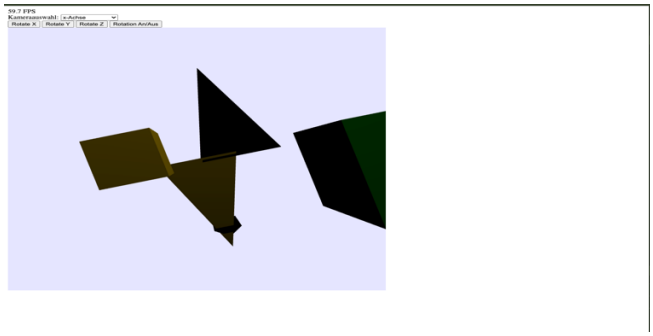
diese Teile ebenfalls abgeschnitten. Dadurch können unvollständige und verzerrte Darstellungen entstehen.

4.Erhöhte Wahrscheinlichkeit für Z-Fighting: Wenn die Near-Clipping-Plane sehr weit weg ist (wie hier 15), kann es zu Z-Fighting kommen, wenn zwei Objekte im Rendering sehr nahe beieinanderliegen. Dies führt zu flackernden oder überlappenden Darstellungen, da die Z-Puffergenauigkeit abnimmt. Das tritt hier aber mit großer Wahrscheinlichkeit nicht auf, da Objekte "geclippt" werden, bevor es zu solchen Effekten kommen kann.

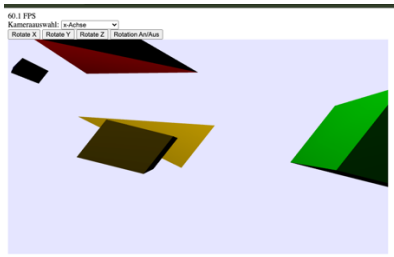
f.

```
<!-- Canvas-Bereich für die WebGL-Darstellung -->
<canvas id="gl-canvas" width="800" height="450">
    The HTML 5 Canvas is not supported by your browser. Please use another browser to view this page.
</canvas>
```

Vorher:



Nachher:



<u>Änderung der Aspect Ratio auf 16:9:</u>

Wenn du die Aspect Ratio explizit auf 16:9 setzen möchtest, müsstest du den zweiten Parameter der `perspective`-Funktion entsprechend anpassen. Da 16:9 ungefähr 1.7777 ergibt, sollte der Wert dort also etwa `1.7777` sein.

<u>Warum das angezeigte Bild im Browser immer noch ein Seitenverhältnis von 1:1 hat:</u>

Der Grund dafür, dass dein Browser das Bild immer noch mit einem 1:1 Seitenverhältnis anzeigt, obwohl du die Aspect Ratio in der Projektionsmatrix auf 16:9 geändert hast, liegt daran, dass **die Canvas selbst immer noch quadratisch ist**.

Die Aspect Ratio in der `perspective`-Funktion ändert nur das Seitenverhältnis der *virtuellen Kamera*, aber nicht das Seitenverh**Die Canvas-Größe ändern:** Stelle sicher, dass das HTML-Canvas-Element selbst ein 16:9-Seitenverhältnis hat. Das kannst du entweder über die HTML-Attribute oder via CSS machen.

```
<!-- Canvas-Bereich für die WebGL-Darstellung -->
<canvas id="gl-canvas" width="800" height="450">
    The HTML 5 Canvas is not supported by your browser. Please use another browser to view this page.
</canvas>
```

GL4:

a. Code-Anpassungen:

- Vertex Shader: Leitet vColor als fColor weiter.
- Fragment Shader: Nutzt fColor bei inaktiver Beleuchtung (sonst Beleuchtungsberechnung).
- JavaScript:
    - drawCube(): Definiert unterschiedliche Farben pro Seite.
    - drawScene(): Übergibt lighting und materialDiffuse
    - calculateLights(): Übergibt Lichtposition und Diffuse Produkt.
    - setCamera(): Definiert Aspekt Ratio korrekt.

```
    <!-- und hier der Quellcode für den Fragment-Shader -->
    <script id="fragment-shader" type="x-shader/x-fragment">
        precision mediump float;
        varying vec4 fColor;

        void main() {
            gl_FragColor = fColor;
        }
    </script>
</body>
```

b. **Ziel:**

Zwei gegenüberliegende Seiten des kleinen Würfels sollen schwarz und die anderen rot dargestellt werden.

Kernpunkte:

1. **Farbanordnung:** Zwei gegenüberliegende Seiten des kleinen Würfels sind schwarz, die restlichen vier Seiten sind rot.

2. **Keine Interpolation:** Es sollten keine Mischfarben wie Grau auftreten, jede Seite sollte eine klare, einheitliche Farbe zeigen.
3. **CPU-seitige Farben:** Die Farben werden direkt von der CPU definiert und nicht durch die Beleuchtungsberechnung beeinflusst.

Code-Anpassungen:

1. **JavaScript (`drawCube()`):**
    - `vertices` Definiert die 8 Eckpunkte des Würfels.
    - `vertexColors` Definiert die Farbe jedes Eckpunkts
    - `indices` Definiert die Verbindungen zwischen den Eckpunkten zu Dreiecken.
    - `gl.drawElements` benutzt das `indices` Array um die Dreiecke zu zeichnen.

d.

Vertex-Shader: Ich habe den Vertex-Shader modifiziert, um die Blinn-Phong-Berechnung zu implementieren.

- Ich habe eine Uniform-Variable `shininess` hinzugefügt.
- Ich habe den Halbierungsvektor `H` berechnet.
- Ich habe den spekularen Anteil mit `pow(max(dot(N, H), 0.0), shininess)` berechnet.
- Ich habe die spekulare, diffuse und ambiente Beleuchtung zur finalen Farbe zusammengeführt.

JavaScript: Ich habe die `calculateLights`-Funktion modifiziert.

- Ich habe die Uniform-Variable `shininess` an den Shader übergeben.
- Ich habe eine neue Lichtfarbe und Materialfarbe für die spekulare Beleuchtung eingeführt.

Ergebnis: Ich habe die Szene mit einer spekularen Beleuchtung versehen, die jetzt einen deutlichen Highlight (mit Blinn) auf der gelben Pyramide bei Betrachtung von der X-Achse zeigt.

Anmerkung: wenn ich Lightning auf True setzt, ist mein kleine Würfel nicht mehr gelb mit zwei schwarzen Seiten

e.

Änderung von `shininess`:

- **Höherer Wert:** Kleineres, intensiveres, glänzenderes Highlight.
- **Niedrigerer Wert:** Größeres, diffuseres, matteres Highlight.

f.

Wenn die Intensität des ambienten Lichts erhöht wird, wird das Bild insgesamt heller, besonders in Bereichen, die direktem Licht abgewandt sind. Eine Verringerung der ambienten Lichtintensität führt zu dunkleren, kontrastreicheren Bildern mit stärker hervorgehobenen Schatten.

60 FPS

Originalkamera
x-Achse
y-Achse
z-Achse
Pyramidenspitze

Rotate X | Rotate Y | Rotate Z | Rotate On/Off