



## 1.1 Einstieg in die 3D-Programmierung

### 3D-Darstellung im Browser

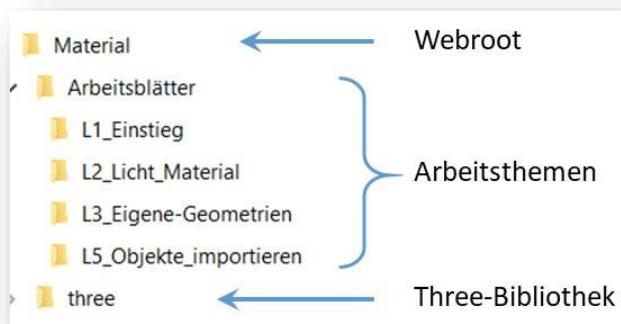
Für die Übungen und Anleitungen wurde die three.js-Bibliothek (in weiteren nur three) gewählt, da es praktisch keine Installationen benötigt. Alle benötigten Bibliotheken sind grundsätzlich auf Cloud-Servern immer in der aktuellsten Fassung verfügbar. Dann müssen lediglich eigene lokale Dateien (z.B. Texturen) auf einem Webserver angeboten werden, da moderne Webbrowser das Nachladen von lokalen Dateien verbieten. Dazu aber später mehr.

#### Voraussetzungen

Um mit three 3-D-Visualisierungen zu entwickeln, benötigt man eigentlich nur einen Editor (der vorzugsweise per Plugins erweiterbar ist wie z.B. Visual Studio Code) und einen Browser. Da aber moderne Browser das Laden lokaler Dateien verbieten (Access to ... has been blocked by CORS policy:). Daher brauchen wir einen kleinen Webserver, so dass wir die Dateien im Browser über localhost:<Port> aufrufen können/müssen.

Am einfachsten geht das Ganze, wenn auf dem Rechner Python installiert ist, dann kann das beiliegende Script server.bat verwendet werden, das einen lokalen Webserver startet, mit dem Startverzeichnis als webroot. Wird die lokale Version der three-Bibliothek verwendet, muss auf die korrekte Pfadeinstellung geachtet werden und dass alle zu verwendende Dateien unterhalb des Webroot-Verzeichnisses liegen.

Hier im rechts gezeigten Beispiel ist der Ordner Material der Ordner, in dem die Datei server.bat liegt und von wo sie auch gestartet wird. Wichtig ist, dass die three-Bibliothek unterhalb dieses Webroot-Ordners liegt. Natürlich müssen auch die HTML-Dateien oder Texturen o.ä., die importiert werden sollen, unterhalb des Webroot-Ordners liegen.



#### Fangen wir an

Starten wir mit der Vorlage „L1-1\_Vorlage.html“, in der schon ein Kommentar eingefügt wurde, an der Stelle, an der es mit dem Kodieren losgehen soll. Speichert die Datei unter dem Namen

**L1-1\_Start.html**

ab.

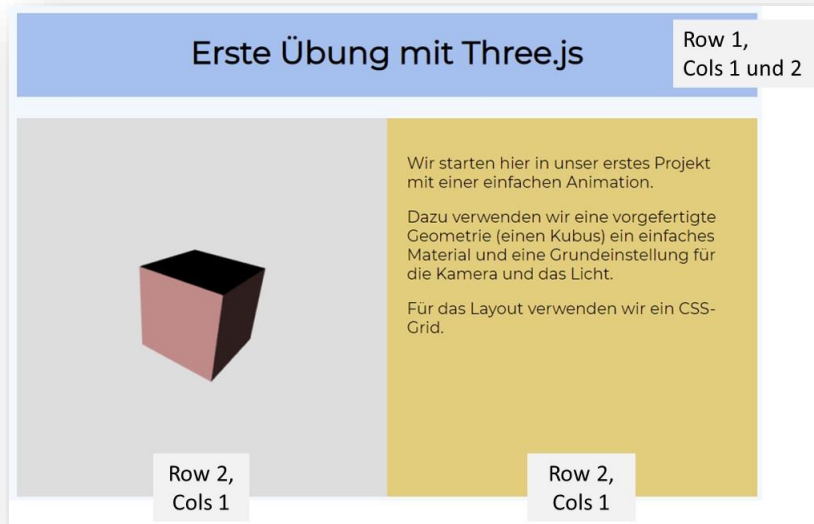
Sehen wir uns dann zunächst die HTML-Struktur an, mit der wir arbeiten. Hier werden wir nur wenig ändern, da können wir alle weiteren Dinge einbauen.

Da three im Browser funktioniert, brauchen wir ein HTML-Gerüst. Zugleich bekommt ihr auch eine passende css-Datei „styles.css“. Üben wir damit gleich ein bisschen HTML/CSS:



## 3D-Programmierung

### Einstieg in die 3D-Programmierung



Wir haben einen Seitenaufbau mit einem CSS-Grid, bei dem die Überschrift über zwei Spalten geht, die linke Seite muss ein Canvas-Element sein (oder allgemeiner: das HTML-Element, in dem die Szene dargestellt werden soll, muss ein Canvas-Tag sein).

Der HTML-Code ist im Kasten drunter dargestellt.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>3D mit Three.js</title>
  <link rel="stylesheet" type="text/css" href="styles.css" />
</head>
<body>
  <h1>Erste Schritte mit Three.js</h1>
  <canvas id="canvas"></canvas>
  <div class="visualization">
    <p>Wir starten hier in unser erstes Projekt mit einer einfachen Animation.</p>
    <p>Dazu verwenden wir eine vorgefertigte Geometrie (einen Kubus),
      ein einfaches Material und eine Grundeinstellung für die Kamera und das Licht.
    </p>
    <p>Für das Layout verwenden wir ein CSS-Grid.</p>
  </div>
  <script type="module">
    ... der Javascript Code
  </script>
</body>
</html>
```

Egal wie der Code aber aufgebaut ist, wir müssen zuerst die Bibliotheken, bzw. die Module einbinden, die wir benötigen.

Grundsätzlich bieten sich hier auch zwei verschiedene Verfahren: der Import aus der ‚Cloud‘ oder alle notwendigen Bibliotheken herunterzuladen und lokal bereit zu stellen. Alle Beispiele arbeiten mit der Variante, die Bibliotheken aus dem Netz zu laden, da wir hier immer die letzten Versionen nutzen können, ohne uns um die Aktualisierung zu kümmern.

Wir benötigen immer die eigentliche three-Kernbibliothek (erste Zeile) sowie bei Bedarf weitere Module.

```
import * as THREE from 'https://unpkg.com/three/build/three.module.js';
import {OrbitControls} from 'https://unpkg.com/three/examples/jsm/controls/OrbitControls.js';
```

Die Hauptbibliothek ist unter dem Verzeichnis „../build“ zu finden, besondere Module sind unter „../examples/jsm“ zu finden.



## 3D-Programmierung

### Einstieg in die 3D-Programmierung

Durch die fortlaufende Weiterentwicklung der Bibliothek kann es zu Änderungen der Cloud-Adresse kommen, deswegen wurde die Bibliothek hier lokal mitgeliefert. Alle Beispiele greifen lokal auf die Bibliotheken zu, weshalb der Webserver zwingend notwendig ist.

### Strukturierter Aufbau

Im Internet kann man natürlich viele Beispiele zu three finden, allerdings sind viele Beispiele für Ungeübte schwer zu lesen. Daher starten wir damit, dass wir uns Funktionen definieren, in denen wir den Aufbau der Szene nach und nach vornehmen und wir den Überblick behalten können, wo wir etwas am Aufbau ändern oder ergänzen.

Dabei ist es nicht wesentlich, ob die einzelnen Teile als Prozeduren (ohne Rückgabe, über globale Variablen) oder als Funktionen (mit Rückgabe des Objekts, keine globale Variable) ausgeführt werden.

Wir erzeugen zuerst den Renderer, da wir für andere Teile des Codes die Größe des Canvas brauchen. Danach erzeugen wir die Szene, damit wir alle weiteren Objekte der Szene in diese einhängen können.

Als nächstes kommen die Kamera und das Licht dran.

Zum Schluss erzeugen wir die Darstellungsobjekte, bevor wir die Szene tatsächlich anzeigen, ggfs. in einer Zeitschleife Wiederholungen erzeugen, um Bewegung zu erzeugen. Ggfs kann es in einem fortgeschrittenen Stadium hilfreich sein, weitere Hilfsfunktionen zur Erzeugung des Materials oder der Geometrie oder eben des Menüs hinzuzufügen.



Zunächst deklarieren wir lokale Variablen innerhalb unseres Moduls. Die Import-Anweisungen benötigen ebenfalls den Zusatz `,module'`. Damit wäre ein grundsätzlicher Aufbau folgendermaßen möglich:

```
<script type="module">
  import * as THREE from 'https://unpkg.com/three/build/three.module.js';

  var scene, camera, renderer, controls, canvas, cube;
  var WIDTH, HEIGHT;

  ...
</script>
```

An Stelle der Punkte kommen die Funktionen. Starten wir, wie oben beschrieben, mit der Erzeugung des Renderers.



## 3D-Programmierung

### Einstieg in die 3D-Programmierung

```
function initRender() {  
    canvas = document.querySelector('#canvas');  
    WIDTH = canvas.clientWidth;  
    HEIGHT = canvas.clientHeight;  
    renderer = new THREE.WebGLRenderer({ canvas, antialias: true });  
    renderer.setSize(WIDTH, HEIGHT);  
    renderer.setPixelRatio(window.devicePixelRatio);  
    renderer.outputEncoding = THREE.sRGBEncoding;  
    renderer.shadowMap.enabled = false;  
    document.body.appendChild(renderer.domElement);  
};
```

Wir holen uns das Node-Element aus dem Dokument-Baum über die ID. Wir setzen die Größe unseres renderer-Objekts und die verfügbare Auflösung des Bildschirms auf die Größe des canvas-Elements. Auf die beiden weiteren Anweisungen gehen wir später (Licht und Schatten) näher ein. Zum Schluss hängen wir unser renderer-Objekt in den Dokumenten-Baum ein. Der Renderer wurde mit der OpenGL-Bibliothek realisiert, um die Elemente auf dem Bildschirm darzustellen. Wir setzen die Breite und Höhe unseres Canvas im der HTML-Seite als Ausdehnung für die Darstellung.

Dann kommt die Funktion für die eigentliche Szene dran. Hier kann eine Hintergrundfarbe gesetzt werden, muss aber nicht (dann ich der Hintergrund schwarz).

```
function initScene() {  
    scene = new THREE.Scene();  
    scene.background = new THREE.Color(0xdddddd);  
};
```

Als nächstes brauchen wir die Kamera. Wir haben zwei verschiedene Typen: die perspektivische Kamera und die orthografische Kamera. Die perspektivische Kamera vermittelt den ‚echten‘ Seheindruck, während die orthografische Kamera keinen Tiefeneindruck hinterlässt. Die weiteren Beispiele arbeiten deswegen alle mit der perspektivischen Kamera.

```
function initCamera() {  
    camera = new THREE.PerspectiveCamera(75, WIDTH / HEIGHT, 0.1, 1000);  
    camera.position.set(10, 10, 15);  
    camera.lookAt(scene.position);  
};
```

Der Kamera werden als Parameter der Öffnungswinkel, das Breiten-Höhenverhältnis, die minimale und der maximale Zoombereich mitgegeben (bewegt sich ein Objekt außerhalb des Bereichs, wird es quasi ‚abgeschnitten‘).

In der Konfiguration oben wurde die Kamera rechts oberhalb des Kubus platziert und die Kamera auf den Punkt des Kubus ausgerichtet. Der Öffnungswinkel der Kamera beträgt 75°, das



## 3D-Programmierung

### Einstieg in die 3D-Programmierung

oben ermittelte Seitenverhältnis (aspect) sowie die vordere (0.1 Abstand) und hintere (100 Abstand) Clippingebene sind definiert. Setzt man die Clippingebenen auf einen Abstand, in den das Objekt hineinragt, wird es quasi abgeschnitten. Es ist jedoch nicht weg, sondern kann z.B. wieder in den sichtbaren Bereich hineinbewegt werden.

Die nächste Funktion erzeugt eine Lichtquelle, die unsere Szene erhellt. Auch dieses Thema werden wir uns später näher ansehen. Allen Lichtquellen können wir eine Lichtfarbe (hier: weiß) und eine Lichtintensität mitgeben (hier: 1).

```
function initLight() {  
    let light = new THREE.DirectionalLight(0xffffff, 1);  
    light.position.set(0, -15, 20); // Position der Lichtquelle  
    light.target.position.set(0, 0, 0); // Position des Lichtziels  
    scene.add(light);  
    scene.add(light.target); // das Target muss der Szene hinzugefügt werden  
};
```

Das Licht ist Default auf den Ursprung (0,0,0) ausgerichtet. Dies bleibt auch im Beispiel so. Allerdings wird gezeigt, wie einerseits die Position des Lichtursprungs, andererseits auch das Lichtziel angegeben werden kann. Wird das Target gesetzt, muss dieses auch der Szene hinzugefügt werden.

Zum Schluss fügen wir unserer Szene das eigentliche Darstellungsobjekt hinzu: einen Würfel, dessen Seiten wir als Parameter übergeben und ein Oberflächenmaterial, das zu den einfachen gehört, aber dennoch bereits ein paar Effekte aufweist, wie wir später sehen werden.

```
function createObject() {  
    let geometry = new THREE.BoxGeometry(7, 7, 7);  
    let material = new THREE.MeshLambertMaterial({  
        color: 0xdd5555,  
        wireframe: false // das Wireframe zeigt die Geometrie ohne Material  
    });  
  
    cube = new THREE.Mesh(geometry, material);  
    cube.position.set(0, 0, 0); // Position des Objekts  
    scene.add(cube);  
};
```

Sind wir soweit gekommen, ist die Szene fertig. Wir brauchen nun noch die Funktionen für die Darstellung. Die Funktion requestAnimationFrame (rAF) stammt aus der Web-API und fordert den Browser auf, in regelmäßigen Intervallen eine Funktion aufzurufen (hier animate).

```
function animate() {  
    cube.rotation.x += 0.01;  
    cube.rotation.y += 0.01;  
    renderer.render(scene, camera);  
    requestAnimationFrame(animate);  
};
```



## 3D-Programmierung

### Einstieg in die 3D-Programmierung

Damit können wir eine gleichmäßige Bewegung erzeugen, wir müssen dazu nur entweder die Raumausrichtung (rotation) ändern oder die Lage des Objekts (position). Die rAF ist im Browser integriert und ändert die Aktualisierung der Oberfläche synchronisiert mit der Bildwiederholrate des Monitors. rAF ist so im Browser implementiert, dass die Bildwiederholrate langsam reduziert wird, wenn der Renderer zwischen zwei Bildern nicht fertig wird mit zeichnen.

Der Aufruf der Funktion render des renderer-Objekts zeichnet die Szene neu. Eine weitere Möglichkeit haben wir mit der setAnimationLoop des WebGLRenderers. Diese Funktion nutzt intern requestAnimationFrame, bietet aber die Möglichkeit die Animation anzuhalten, indem als Callback-Funktion null übergeben wird. Es wird empfohlen, für Virtual Reality und Augmented Reality mit dieser Funktion zu arbeiten und die Animation entsprechend zu kontrollieren.

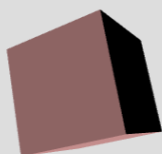
Dann setzen wir noch alles zusammen und sorgen dafür, dass alle Funktionen beim Laden der Webseite aufgerufen werden. Dazu schreiben wir eine Funktion, die unsere einzelnen Funktionen in der richtigen Reihenfolge aufruft (siehe oben).

```
function init3D() {  
  initRender();  
  initScene();  
  initCamera();  
  initLight();  
  createObject();  
  animate();  
};
```

Und letztlich muss das Szenario noch dargestellt werden, weshalb wir mit dem Seitenaufbau die Funktion init3D aufrufen müssen. Dazu fügen wir dem Fenster noch den EventListener hinzu, der ausgelöst wird, wenn die Seite geladen wurde.

```
window.addEventListener("load", init3D);
```

### Erste Übung mit Three.js



Wir starten hier in unser erstes Projekt mit einer einfachen Animation.

Dazu verwenden wir eine vorgefertigte Geometrie (einen Kubus) ein einfaches Material und eine Grundeinstellung für die Kamera und das Licht.

Für das Layout verwenden wir ein CSS-Grid.

Dann sollte das Ganze in etwa so im Browser aussehen, wobei sich der Kubus langsam um die x- und y-Achse drehen sollte.



#### Hinweis

Um eine bessere Orientierung im Raum zu erreichen (für den Entwickler), können wir Hilfskonstruktionen einblenden. Wenn wir der Szene den AxesHelper hinzufügen, werden die x-, y- und z-Koordinatenachsen mit eingeblendet. Der Parameter im Konstruktor gibt die Länge der Achsen an.

```
let axesHelper = new THREE.AxesHelper(10);  
scene.add(axesHelper);
```

#### Aufgabe A1

Ändern Sie die Position der Kamera indem Sie die z-Position invertieren und zusätzlich verdoppeln.

Speichern Sie Ihre Lösung in der Datei `L1-A1_Start.html`.

## 1.2 Interaktion und Bewegung

### Interaktion hinzufügen

Wir können Bewegungen/Veränderungen unserer 3D-Darstellung in 3 Bereiche einteilen:

- Veränderungen durch Berechnungen, d.h., Bewegung des Objekts oder Drehung des Objekts
- Veränderung durch Ereignisse wie Tastatur, Maus und Browser-Events
- Externe Veränderungen durch Funktionsaufrufe

Die Bewegung unserer Objekte durch Veränderung ihrer Position bzw. ihrer Rotation haben wir ja bereits im letzten Informationsblatt bzw. in der letzten Aufgabenstellung aufgenommen.

Neben der zeitgesteuerten Animation, die eine mehr oder weniger gleichmäßige Bewegung modellieren kann, können wir auch soweit die Animation beeinflussen, in dem wir Handler hinzufügen, die per Tastatur, Maus o.ä. gesteuert werden. Da die Maus zu den gebräuchlichsten Eingabegeräten gehört, fügen wir der Animation eine Drag-Steuerung hinzu, die das Objekt mit der linken Maustaste bewegt.

Wir starten wieder mit der Datei, die wir beim letzten Mal erstellt haben (`L1-1_Start.html`) und speichern die unter dem Namen

`L1-2_Start.html`

ab.

Einen Event-Handler haben wir bereits verwendet:

```
window.addEventListener("load", init3D);
```





## 3D-Programmierung

### Einstieg in die 3D-Programmierung

Eine weitere typische Interaktionsmöglichkeit wird über die OrbitControl realisiert. OrbitControls ermöglicht der Kamera um ein Ziel zu kreisen. Damit dies klappt, muss der Renderer aber auch die Sicht dieser Kamera darstellen. Es wirkt, als wenn man das Objekt drehen würde. In Wirklichkeit bewegt sich die Kamera um das Objekt.

```
let controls = new OrbitControls(camera, renderer.domElement);
```

### Control-Panel

Three bietet die Möglichkeit, Eigenschaften oder Aktivitäten über ein Control-Panel (wie ein Menü) Ein- oder Auszuschalten oder zu verändern. Dazu müssen wir die Bibliothek GUI einbinden.

```
import { GUI } from 'https://unpkg.com/three/examples/jsm/libs/dat.gui.module.js';
```

Wir müssen ein GUI-Objekt initialisieren und geben als Initialisierung mit, ob das Control-Panel am Rand des Fensters (autoplace: true) oder an einem Dokumentknoten festgemacht wird (autoplace: false). Wir verwenden letzteres, um das Menü in unserem rechten Element einzubauen.

```
function initGuiControls() {  
  // JSON-Struktur  
  var options = {  
    move: function () {  
      MOVE = !MOVE;  
    }  
  };  
  
  let gui = new GUI({ autoplace: false });  
  
  let folder = gui.addFolder('Controls');  
  folder.addColor(new ColorGUIHelper(cube.material, 'color'), 'value')  
    .name('color');  
  folder.add(camera.position, 'y', 0, 30).listen();  
  folder.open();  
  gui.add(options, 'move'); // ruft move auf  
  
  let guiContainer = document.querySelector('#visualization');  
  guiContainer.insertBefore(gui.domElement, guiContainer.firstChild);  
}
```

Wir können das Menü über die JSON-Struktur definieren

Oder über die Anlage einzelner Folder

Autoplace:false lässt uns die GUI in der HTML-Struktur verankern, sonst wird die GUI am rechten oberen Eck des Browser-Fensters eingehängt.

Der GUI müssen einzelne Menüpunkte hinzugefügt werden (haben wir nur die GUI, ist das ein Menü ohne Inhalt). Entweder über die Funktion addFolder oder über eine JSON-Struktur (hier: options). In der JSON-Struktur können einfach Funktionen eingebunden werden. Im Beispiel unten ist die Struktur „options“ mit einem Element definiert.

Im Rumpf des Elements wird eine lokale Variable SPEED invertiert. Wir können nun mit der Funktion „add“ der GUI die JSON-Struktur übergeben und müssen als 2. Parameter sagen, welche der Definitionen in der Struktur hier im Menü auftauchen soll.

Wir können einzelne Attribute unserer Szenenobjekte angeben (siehe Bsp. Camera.position) mit min- und max-Werten und wir können komplette Objekte übergeben wie die Instanz des ColorGuiHelper, mit dem wir den Farbwert des Materials verändern können (siehe unten).





## 3D-Programmierung

### Einstieg in die 3D-Programmierung

```
class ColorGUIHelper {  
  constructor(object, prop) {  
    this.object = object;  
    this.prop = prop;  
  }  
  get value() {  
    return `#${this.object[this.prop].getHexString()}`;  
  }  
  set value(hexString) {  
    this.object[this.prop].set(hexString);  
  }  
}
```

Die Helper-Klasse dient  
für den Color-Picker

Das Ganze sieht dann wie unten gezeigt aus, wobei in der HTML-Struktur ein Element „controls“ eingebaut wurde. Trotzdem ist zu beachten, dass das Kontrollelement nicht flexibel ist in der Größe.

Wir können unserer GUI Menüpunkte direkt anheften (siehe `gui.add(options,..)`) o-

der wir fügen unserer GUI einen Folder an, der wiederum Menüpunkte aufnehmen kann (`folder.add(..)`). Jeder Folder lässt sich auf- und zuklappen.

Das Ganze sollte dann wie unten gezeigt aussehen. Der Menüpunkt mit dem Color-Picker und der Höhenverstellung (Definierter Bereich von 0 bis 10 für den y-Wert des Kubus) lässt sich auf- und zuklappen.

Die Funktion „initGuiControls“ müssen wir natürlich in die Initialisierungsfunktion „init3D“ einbauen, bzw. dort aufrufen. Dies geschieht am besten unmittelbar vor dem Aufruf der „animate“-Funktion.

```
function animate() {  
  if (MOVE) {  
    // vorgesehene Bewegung  
    ... // hier kommt der Code für die Bewegung hin  
  }  
  renderer.render(scene, camera);  
  requestAnimationFrame(animate);  
};
```



## Erste Übung mit Gui-Controls

Höhenverstellung

Color-Picker

Bewegung an/aus

Controls

color #dd5555

y 10

move

Close Controls

E  
v  
können wir mit der Maus das Objekt drehen.

Zusätzlich erlaubt eine weitere Bibliothek - die gui-Lib - die Steuerung bzw. Einstellungen über ein Menü vorzunehmen.

### Hinweise:

Die Größe des Würfels kann über das Attribut `scale` verändert werden.

```
cube.scale.x = 1.0; // Größe bleibt konstant
```

### Aufgaben

1. Ergänzen Sie den Folder um die folgenden Funktionen:
  - a. Verändern Sie die Größe des Würfels (es reicht, die Seiten um den Faktor 2 zu vergrößern).
  - b. Schalten Sie das Wireframe ein und aus.
2. Schicken Sie den Würfel auf eine Kreisbahn um das Zentrum mit dem Radius von 5. Beachten Sie dabei, dass eine Kreisbahn um die y-Achse durch die Veränderung der x- und z-Werte erzeugt werden muss. Zudem muss das Ganze in Schritten erfolgen, damit eine sichtbare Bewegung entstehen kann. Sinnvoll ist z.B. eine Einteilung in 360 Schritte pro Sekunde (1 Grad pro Sekunde), wobei zu beachten ist, dass pro Sekunde etwa 60 Frames erzeugt werden.