1. Einleitung

Das vorliegende Projekt befasst sich mit der Entwicklung einer Webanwendung, die es Benutzern ermöglicht, Möbel in einer virtuellen 3D-Umgebung innerhalb eines quaderförmigen Raumes zu platzieren. Ziel dieser Anwendung ist es, eine benutzerfreundliche und intuitive Lösung zur Verfügung zu stellen, die grundlegende Funktionen der Raumplanung bietet, ohne die Benutzer durch übermäßige Komplexität zu überfordern. Besonderes Augenmerk liegt dabei auf der Funktionalität: Alle implementierten Features sollen zuverlässig und robust funktionieren. Fehleranfälligkeiten und Inkonsistenzen, wie beispielsweise fliegende oder unvollständig dargestellte Modelle, sollen vermieden werden. Der Schwerpunkt der Entwicklung liegt somit eher auf einer stabilen Basis, auch wenn dies bedeutet, dass auf zusätzliche Features zugunsten der Einfachheit und Fehlerresistenz verzichtet wird.

Für die Implementierung der Anwendung werden HTML, CSS und JavaScript verwendet. Insbesondere kommt die JavaScript-Bibliothek Three.js [1] zur Visualisierung der 3D-Modelle und Szenen zum Einsatz. Die Erstellung und das Laden von 3D-Modellen und deren Texturen erfolgt mithilfe von Open-Source-Ressourcen, wie sie von polyhaven.com und Blender bereitgestellt werden. Alle im Projekt genutzten Tools sind frei zugänglich

[1] Three.js Dokumentation zum Laden von 3D Modellen:
https://threejs.org/docs/?q=3d#manual/en/introduction/Loading-3D-models

2. Technische Architektur

Die Applikation ist in drei wesentliche Komponenten unterteilt: Struktur, Design und Logik. Die Struktur bzw. das Frontend wird durch ein HTML-File vorgegeben. In der Datei webproject.html wird die Benutzeroberfläche (UI) sowie die Verwaltung der interaktiven Elemente organisiert. Das Design basiert auf einem separaten CSS-File: styles.css. Dort werden sämtliche Positionierungen, Gestaltungsmerkmale wie Schriftarten, Farben und die Modal-Fenster (also die „Menüs“ der Applikation) definiert. Die gesamte Logik der Anwendung ist in der Datei app.js implementiert. Diese Datei ist verantwortlich für die 3D-Darstellung von Raum, Möbel und Beleuchtung. Zudem beinhaltet sie die Logik, die es dem Benutzer ermöglicht, über Maus und Tastatur mit der Anwendung zu interagieren und den Raum sowie die Möbel zu steuern.

Das Zusammenspiel zwischen den HTML-, CSS- und JavaScript-Dateien ist wie folgt strukturiert: HTML-Elemente fungieren als Auslöser für Benutzeraktionen, die über JavaScript gesteuert werden. Beispielsweise löst ein Klick auf den „Raum erstellen“-Button einen JavaScript-Event-Listener aus, der die im HTML-Formular eingegebenen Raumdimensionen verarbeitet und über Three.js in der 3D-Szene darstellt. Das CSS sorgt dafür, dass diese Elemente optisch ansprechend und benutzerfreundlich dargestellt werden.

Die statischen Ressourcen, 3D-Modelle und Texturen, sind in eigenen Verzeichnissen (models, textures) organisiert, um eine modulare Trennung zwischen der Benutzeroberfläche, der Logik und den Ressourcen sicherzustellen. Dies ermöglicht eine saubere Struktur und erleichtert außerdem zukünftige Erweiterungen.

3. Hauptfunktionen der Anwendung

In diesem Kapitel werden die Kernfunktionen der Anwendung detailliert beschrieben: die Erstellung eines Raums, die Platzierung von Möbeln und Türen sowie die interaktiven Steuerungsmöglichkeiten der Benutzer.

3.1 Raumerstellung

Die Funktion createRoom ist verantwortlich für die Erzeugung des Raumes innerhalb der 3D-Szene. Sie verwendet die Three.js-Bibliothek, um den Raum dreidimensional darzustellen und mit Lichtquellen auszustatten. Der Raum besteht aus Boden, Wänden und Beleuchtung, die durch den Benutzer über ein Formular definiert werden. Die Funktion akzeptiert drei Parameter: Länge, Breite und Höhe des Raums (Abb.1). Diese Dimensionen werden vom Benutzer über das HTML-Formular eingegeben und in der Funktion verarbeitet.

```
const length = parseFloat(document.getElementById('length').value);
const width = parseFloat(document.getElementById('width').value);
const height = parseFloat(document.getElementById('height').value);
```

Abb. 1 Teil der EventListener mit ID „roomForm“

Sobald die Eingaben vorliegen, erstellt createRoom den 3D-Raum inklusive Beleuchtung durch die folgende Three.js Logik:

Der Boden (ein Rechteck oder „plane“) wird mit den angegebenen Maßen für die Länge und Breite des Raums erzeugt und horizontal ausgerichtet. Anschließend werden die Wände aus vier weiteren planes erstellt, die den Raum umschließen. Sie werden entlang der Höhe und den Seiten positioniert. Die Wände werden leicht transparent gestaltet, um eine bessere Sichtbarkeit innerhalb des Raums zu gewährleisten.

Um eine realistische Beleuchtung zu gewährleisten, wird eine Lichtquelle und ein sichtbares Objekt, dass die Lichtquelle repräsentiert, im Zentrum der unsichtbaren Decke hinzugefügt. Ein zusätzliches Umgebungslicht wird verwendet, um die Helligkeit auszugleichen und scharfe Schatten abzumildern, die ansonsten völlig schwarz wären, was unrealistisch wirkt. Die Abbildung von Schatten wird durch das Three.js Attribut recieveShadow ermöglicht.

3.2 Kamera und Interaktion

Nach der Raumerstellung wird die Kamera entsprechend positioniert, um eine optimale Perspektive auf den Raum zu bieten. Die Kamera kann mit Hilfe der OrbitControls-Funktion von Three.js durch den Benutzer gedreht und bewegt werden. Dies ermöglicht eine einfache Navigation durch den Raum per Links- und Rechtsklick bzw. ziehen der Maus.

Über Event-Listener werden die Benutzeraktionen erfasst, wie etwa die Eingabe der Raumdimensionen und das Bestätigen des Formulars. Sobald der Raum erstellt wurde, wird das Formular geschlossen und der Raum in der Szene dargestellt. Der Renderer und die Kamera werden dabei in der Szene aktualisiert und sichtbar gemacht (Abb.2).

```
scene = new THREE.Scene();
camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
scene.background = null;
renderer = new THREE.WebGLRenderer({ alpha: true });
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.shadowMap.enabled = true; // Enable shadows!!!
renderer.shadowMap.type = THREE.PCFSoftShadowMap;
document.getElementById('3d-container').innerHTML = '';
document.getElementById('3d-container').appendChild(renderer.domElement);
```

Abb. 2 Teil der Funktion createRoom

3.3 Objekterstellung: Möbel und Türen

Ähnlich wie die Funktion createRoom erzeugt createObject ebenfalls ein Objekt im 3D Raum. Hierbei ist jedoch der Unterschied, das sich diese Objekte innerhalb einer Grenze, der sogenannten „bounding Box“ befinden, die durch die Wände, Boden und Decke des Raums beschrieben werden. createObject verweist je nach Typ des zu erzeugenden Objekts auf eine von zwei Funktionen: Im Fall, dass ein texturiertes 3D Modell erschaffen werden soll (Stuhl, Tisch, Bett, Schrank) so wird loadModel aufgerufen, im Fall von Türen die Funktion placeDoor.

Texturierte 3D Modelle enthält einen Modell-„Loader“ und alle Zugriffe auf die Ressourcen zum tatsächlichen Modell (als .glb-Datei gespeichert) und dessen Texturen (in weiterem Sub-Verzeichnis als .jpg- und .exr-Datei vorhanden). Dagegen werden Türen nur auf Basis ihrer übergebenen Höhe und Breite erschaffen, Die Tiefe ist um einen realistischen Standard zusetzen statisch auf 0.1m festgelegt. Tür-Objekte besitzen keine spezielle Textur, sondern sind einfachheitshalber auf einen hölzernen Braunton (Hex: 8B4513) standardisiert.

Die Erstellung jedes Objekts kann durch Benutzereingaben für die Parameter Höhe, Breite und Tiefe beeinflusst werden. Alle Möbelstücke haben eine individuelle Meshgröße [2] und sind daher bei dem initial gleich definierten height-, width- oder depth-Multiplier von 1 unterschiedlich groß: Das Modell eines Betts ist bei unveränderter Skalierung 1.5m * 1.5m * 2m groß, wobei ein Stuhl nur 1m * 1m * 1m misst. Negative Werte (im Multiplier) führen hierbei lediglich zu einer Spiegelung und sind dementsprechend unproblematisch.

Das gerenderte Objekt wird dann der Szene hinzugefügt und mit einem Label versehen (Abb.3). Labels werden pro Objekt durch die Funktion createLabelForObject erstellt und enthalten den Objektnamen, bei Türen entspricht er standardmäßig „Door“.

```
model.scale.set(width, height, depth);
scene.add(model);
objects.push(model);

createLabelForObject(name, model); // Create and attach a label to the object
createButtons(model, name); // Create interaction (select, rotate, delete) buttons for the object
```

Abb. 3 Teil der Funktion loadModel

3.4 Löschen von Objekten

Die Möglichkeit, Möbel und Türen zu löschen, ist eine wesentliche Funktion, um die Raumplanung flexibel zu halten. Die Funktion deleteObject entfernt das ausgewählte Objekt nicht nur aus der Szene, sondern kümmert sich auch um das Entfernen aller zugehörigen Elemente, wie Labels oder Buttons, die dem Objekt zugewiesen wurden. Dies stellt sicher, dass es zu keinen unerwünschten Rückständen und damit potentiellen Fehlern in der Benutzeroberfläche kommt.

3.5 Interaktive Steuerung von Raum und Objekten

Die Funktion createButtons ist verantwortlich für das Erstellen von Knöpfen, die es dem Benutzer ermöglichen, Objekte auszuwählen, zu rotieren und zu löschen (siehe Abb.3). Ein ausgewähltes Objekt kann durch die Tasteneingaben „w“, „a“, „s“ oder „d“ entlang der x- und z-Achse bewegt werden. Dabei wird auf die Funktion onKeyPress zurückgegriffen, die diese Tastenereignisse abfängt und die Bewegung entsprechend steuert. Die Funktion handleGeneralObjectMovement steuert die Bewegung des Objekts um 0,5 Meter pro Schritt entlang der x- oder z-Achse. Dabei wird sichergestellt, dass das Objekt nicht außerhalb der Raumgrenzen bewegt wird. Wenn die Bewegung ungültig ist (z. B. Kollision mit einer Wand), wird die Änderung verworfen und das Objekt bleibt an seiner ursprünglichen Position. Das Bewegen eines Möbelstücks

[2] Die Größe des Meshs in einer Software wie z.B. Blender gibt an, wie groß das Objekt beim Erstellen ist

durch ein anderes gilt nicht als Problem, da ansonsten vor allem in kleinen Räumen die Bewegung von manchen Objekten schwierig oder gar nicht möglich ist.

Die zweite Bewegungsoption ist die Rotation der Objekte. Wenn der Benutzer den Rotationsknopf drückt, wird die Funktion rotateObject aufgerufen. Diese Funktion sorgt dafür, dass das Objekt um 45 Grad gedreht wird, wobei darauf geachtet wird, dass die Objektbewegung innerhalb der durch die bounding box definierten Raumgrenzen bleibt. Durch die Funktion rotateObject kann das ausgewählte Objekt um 45 Grad um die y-Achse gedreht werden. Auch hier wird überprüft, ob die Rotation innerhalb der zulässigen Raumgrenzen bleibt. Bei einer ungültigen Rotation bleibt das Objekt unverändert.

4. Herausforderungen und Lösungswege

4.1 Technische Herausforderungen

Eine der frühesten Herausforderungen war die Implementierung von Event Listenern, welche bei fast jeder Funktionalität des aktuellen Programmes unentbehrlich sind. Dabei war es zentral, den Unterschied zwischen einem „submit“ und einem „click“ zu kennen. Es handelt sich um verschiedene Eingabetypen, die sich auf Buttons eines HTML Dokuments beziehen. W3schools.com erwies sich dabei als sehr hilfreiche Ressource. [3]

Die Konstruktion eines Renderers beschrieb einen weiteren essentiellen, aber komplex zu realisierenden Punkt der Entwicklung und lehnte sich stark an der offiziellen Dokumentation von Three.js [4] und einem sbcode.net-Beispiel [5] an. Die Implementierung von OrbitControls war aufgrund der im Überfluss vorhandenen Ressourcen vergleichsweise schnell, wobei keine herkömmlichen Tasteneingaben zur Orientierung der Kamera genutzt wurden, sondern direkt der Renderer mithilfe OrbitControls direkt auf die jeweiligen Kameraposition angewandt werden konnte.

Ein später auftretendes Problem war das Finden und Einbinden von 3D-Modellen. Zunächst mussten geeignete Modelle und Texturen auf Websites wie polyhaven gefunden und in das richtige Format gebracht werden. Mit der Open-Source-Anwendung Blender [6] wurden die Modelle in das .glb-Format konvertiert, das für Three.js geeignet ist. Das Einbinden der Modelle erwies sich als kompliziert, da der GLTFLoader [7] von Three.js Modelle asynchron lädt, was anfangs schwer nachvollziehbar war. Die Funktion loadModel nahm den größten Teil der Entwicklungszeit in Anspruch, hauptsächlich aufgrund von Problemen mit der Darstellung der Texturen und Modelle.

Obwohl der Code korrekt schien und der Loader, Renderer und die Kamera funktionierten, wurde zunächst kein 3D-Modell sichtbar. Nach vielen Debugging-Versuchen, auch mithilfe GPT-4o und dessen vorgeschlagenem Einsatz von Konsolen-Logs, stellte sich heraus, dass das Problem durch CORS (Cross-Origin Ressource Sharing) verursacht wurde. Diese Sicherheitsrichtlinie der Browser blockiert standardmäßig den Zugriff auf externe Ressourcen. [8] Die Lösung bestand darin, entweder einen Proxy-Server zu verwenden oder, wie in diesem Projekt, die Website, also die HTML-Datei über die Erweiterung Live-Server in Visual Studio Code auszuführen, um das Laden externer Ressourcen zu ermöglichen. [9]

---

[3] https://www.w3schools.com/jsref/met_element_addeventlistener.asp, https://www.w3schools.com/tags/att_input_type_submit.asp

[4] https://threejs.org/docs/#api/en/renderers/WebGLRenderer

[5] https://sbcode.net/threejs/scene-camera-renderer/

[6] How to Export to .glb: https://www.myarstudio.cloud/learn/documentation/blender/exporting-glb-files-from-blender/, https://www.blender.org

[7] https://threejs.org/docs/#examples/en/loaders/GLTFLoader

[8] https://www.reddit.com/r/threejs/comments/fqkj3t/cors_policy_blocks_gltf_loader/

[9] https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer

Insgesamt war es schwierig bei der wachsenden Komplexität des Codes jeden Teil so modular zu halten wie möglich. Modularität war ein Ziel, da Erweiterungen andernfalls sehr schnell sehr aufwendig werden können. Die Aufteilung der .js Logik in Raum- und Objektsteuerung sowie das Erstellen, Löschen und Bewegen von Objekten war von Anfang an der Plan. Trotzdem stellte sich immer wieder heraus, dass die häufige Interaktion zwischen Funktionen stark auf die Komplexität der Fehlersuche Einfluss nimmt. Oft involvierte der Prozess bis zur einer Lösung mehrere Schritte, in denen ein Problem gelöst wurde, aber ein oder sogar mehrere andere (neue oder bereits bekannte) Probleme entstanden.

4.2 Herausforderungen und Hilfsmittel für intuitives Design und Styling

Die in Abschnitt 4.1 erwähnten Ressourcen, wie Teile der offiziellen Three.js-Dokumentation, Tutorials auf sbcode.net und verschiedene Forenbeiträge, konzentrierten sich vor allem auf die technische Funktionalität des Projekts. Ein weiterer entscheidender Aspekt war jedoch das Design der Webanwendung, das nicht nur visuell ansprechend sein sollte, sondern auch die Benutzerfreundlichkeit stark beeinflusst. Besonders wichtig war es, die Anwendung so zu gestalten, dass sie auch für neue Benutzer leicht verständlich und intuitiv bedienbar ist.

Die Herausforderungen beim Thema intuitives Design lagen darin, dass es aus der Sicht des Entwicklers schwer nachvollziehbar ist, welche Schwierigkeiten neue Benutzer haben könnten. Als Entwickler kennt man die Features und Möglichkeiten der Web-App genau, im Gegensatz zu einem unvoreingenommenen Benutzer. Hier halfen Testnutzer (Kommilitonen) und obwohl die Funktionen zu diesem Zeitpunkt noch nicht vollständig ausgereift waren, lieferten sie wertvolles Feedback zur Benutzerfreundlichkeit.

Als weiteres Hilfsmittel wurde hierbei GPT-4o verwendet, um bei der folgenden Frage einen Überblick zu schaffen, wobei nur die wichtigsten Antworten in kurz zusammengefasst sind:

1. Was verleiht einer Website intuitive Interaktion?

Antwort: Klares Design und Layout, Einfache Navigation, Sichtbarkeit von wichtigen Features, Feedback und Benutzerinteraktion.

Die Implementierung von Funktionalitäten wurde nach diesen Prinzipien umgesetzt, indem die Funktion von verschiedenen Buttons klar in deren Namen deklariert ist. Außerdem wurde auf Dropdown-Menüs in Form von Modals zurückgegriffen um dem User zu zeigen in welchem Teil der Website er sich gerade befindet und wo Input erwartet wird. Des weiteren wurden möglichst wenige Buttons oder andere möglicherweise visuell ablenkenden Objekte genutzt, um Überforderung des Benutzers zu vermeiden.

Für das Styling der Buttons und Eingabefelder diente eine Website, die eigentlich für Matrixberechnungen konzipiert wurde, als Vorbild. [10] Diese bot eine gute Grundlage für die Eingabe- und Darstellungsstruktur der vorliegenden Anwendung. Das Farbschema der 3D-basierten Webanwendung wurde jedoch angepasst, um ein lebendigeres und passenderes Aussehen zu schaffen, das besser zum Charakter des Projekts passt, da die Beispielwebsite sehr trist gestaltet war.

[10] https://matrixcalc.org/de/vectors.html, Rechtsklick auf eine Stelle der Seite>Element untersuchen>im HTML nach <style> suchen und aufklappen

5. Fazit und Ausblick

Die entwickelte Webanwendung bietet eine intuitive Lösung für die Platzierung von Möbeln in einer virtuellen 3D-Umgebung. Zu den wichtigsten Funktionen zählen die Raumerstellung, das Hinzufügen, Entfernen und Manipulieren (in Größe und Position) von Möbeln sowie die interaktive Steuerung dieser Objekte durch den Benutzer. Durch verständliche Buttons sowie die Verwendung von Dropdown-Menüs in Form von Modals ist die Applikation benutzerfreundlich gestaltet. Insgesamt wurde darauf geachtet, dass die Anwendung robust und frei von kritischen Fehlern bleibt, insbesondere was das korrekte Platzieren und Darstellen der Möbelmodelle betrifft. Die oben genau beschriebene Integration von HTML, CSS, JavaScript und Three.js sowie der Einsatz von Ressourcen wie polyhaven.com und Blender ermöglichten die erfolgreiche Umsetzung dieses Projekts.

Es gibt verschiedene Ansätze, um die Anwendung weiter zu verbessern: Eine mobile Version könnte entwickelt werden, um die Benutzererfahrung auf z.B. Smartphones zu optimieren und die Anwendung einer größeren Nutzerzahl zugänglich zu machen. Auch die Erweiterung der Interaktionsmöglichkeiten hat Potenzial: Funktionen wie eine präzisere Kollisionserkennung oder das automatische Ausrichten von Möbeln (Türen speziell) würden die Anwendung noch realistischer und benutzerfreundlicher machen. Zudem könnten mehr 3D-Modelle und Texturen hinzugefügt werden, um die Vielfalt der Gestaltungsmöglichkeiten zu erhöhen, bisher ist z.B. noch keine API-Lösung für diese Art von Anwendung auffindbar. Schließlich wäre es sinnvoll, dem Benutzer die Möglichkeit zu geben, Raumkonfigurationen zu speichern und später wieder aufzurufen, um die Flexibilität und Nutzbarkeit der Anwendung weiter zu steigern, was bei einem tatsächlichen Deployment von Vorteil wäre.