VISMO-Projekt – Entwicklerhandbuch

Handbuch für Entwickler

für das weitere Programmieren für das VISMO-Projekt

(VISMO-Projekt Version 1.1.21)

Inhaltsverzeichnis

1. Thema dieser Dokumentation

Dieses Handbuch soll zukünftigen Entwicklern des VISMO-Projekts den ersten Einstieg und das Arbeiten am bereits bestehenden Code vereinfachen.

In diesem Handbuch werden alle JavaScipt-Dateien aufgezählt, die zum Funktionieren der lauffähigen Anwendung beitragen. Dazu gehören Elemente der Three.js-Bibliothek und den durch die Projektmitglieder geschriebenen Code.

2. VISMO-Projekt Grundkonzeption

VISMO steht für „The Visualization of Structure from Motion" und bringt den Prozess der „Structure from Motion" experimentell näher. Es werden die mathematischen Grundlagen in ihrem Kern anhand eines einfachen 3D-Modells visualisiert. Die Applikation simuliert dabei, wie aus einer Konfiguration mit zwei Kameras genaue Informationen über die Struktur der Szene entnommen werden können.

3. Three.js

Die API und JavaScript-Bibliothek Three.js macht den Großteil der VISMO-Anwendung aus. Zum Verständnis der verschiedenen Klassen und Methoden, sollte die offizielle Three.js-Dokumentation hinzugezogen werden (https://threejs.org/docs/).

3.1. ThreeModule

Den als „three.module.js" auffindbare Sourcecode bildet das Fundament für die Anwendung (https://github.com/mrdoob/three.js/blob/dev/build/three.module.js).

Dieser Code ist in der VISMO-Anwendung unter „threejs" --> „ThreeModule.js" aufzufinden. Die sich darin befindende Klasse „CameraHelper" wurde von den VISMO-Projektmitgliedern modifiziert. Dies sollte beachtet werden, falls dieser Sourcecode durch eine neue Version ersetzt werden sollte.

3.2. Zusätzliche Klassen

Neben dem „three.module.js“-Sourcecode werden auch weitere Klassen, die sich nicht in diesem Sourcecode befinden, aber Teil der Three.js-Bibliothek sind, hinzugezogen, um der VISMO-Anwendung mehr Funktionen zu geben.

3.2.1. GLTFLoader

Der Sourcecode wurde 1:1 aus dem offiziellen Three.js-Repository übernommen (https://github.com/mrdoob/three.js/blob/master/examples/jsm/loaders/GLTFLoader.js). Es wurden keine Modifikationen an dem Code vorgenommen.

Die Klasse „GLTFLoader“ dient zum Laden des Giraffenmodells „Gigi“.

Die offizielle Dokumentation beinhaltet weitere Informationen (https://threejs.org/docs/#examples/en/loaders/GLTFLoader).

3.2.2. OrbitControls

Der Sourcecode wurde 1:1 aus dem offiziellen Three.js-Repository übernommen (https://github.com/mrdoob/three.js/blob/master/examples/jsm/controls/OrbitControls.js). Es wurden keine Modifikationen an dem Code vorgenommen.

Die Klasse „OrbitControls“ dient zum Steuern der Szenenkamera.

Die offizielle Dokumentation beinhaltet weitere Informationen (https://threejs.org/docs/#examples/en/controls/OrbitControls).

3.2.3. TransformControls

Der Sourcecode wurde 1:1 aus dem offiziellen Three.js-Repository übernommen (https://github.com/mrdoob/three.js/blob/master/examples/jsm/controls/TransformControls.js). Es wurden keine Modifikationen an dem Code vorgenommen.

Die Klasse „TransformControls“ dient zur Rotation der linken und rechten Kamera.

Die offizielle Dokumentation beinhaltet weitere Informationen (https://threejs.org/docs/#examples/en/controls/TransformControls).

4. JavaScript Klassen

Die folgenden JavaScript-Klassen wurden von den VISMO-Projektmitgliedern geschrieben.

4.1. Beam

Die Klasse „Beam“ erbt von der Klasse „Line“.

Die Klasse dient zur Erstellung von Strahlen durch Instanziierung. Die Strahlen sollen von dem Projektionszentrum der linken oder rechten Kamera zu einem oder allen Punkten des Gebäudes reichen.

4.2. Camera

Die Klasse „Camera“ erbt von der Three.js-Klasse „PerspectiveCamera“ (https://threejs.org/docs/index.html#api/en/cameras/PerspectiveCamera).

Die Klasse dient zur Erstellung von speziellen Kameras durch Instanziierung. Die Kameras sind in der Lage bestimmte Berechnungen und Ausgaben zu tätigen.

Das Attribut **„principalPoint“** dient zum Festhalten der Position des Hauptpunktes der Sensorebene der Kamera in Form von des Three.js-Vector3.

Das Attribut **„projectionMatrixArray“** dient zum Festhalten der Projektionsmatrix der Kamera in Form eines Arrays, das die Länge 16 besitzen sollte (4x4 Matrix).

Die Methode **„updatePrincipalPoint()“** berechnet den Hauptpunkt anhand der Kameranormalen und der Kameraposition (= Projektionszentrum).

Die Methode **„updateProjectionMatrix()“** berechnet die Projektionsmatrix der Kamera. Hierfür wurde folgende Formel verwendet:

$$[P] = [E_4] - \frac{1}{L \cdot [Z]} [Z] \cdot [L]$$

Diese Formel entspringt dem Dokument „ViSMo_Arbeitsblatt-1.pdf“. Genauere Erläuterungen zur Berechnung sind dort aufzufinden.

Die Methode **„getAKG“** berechnet die Allgemeine Koordinatengleichung der Sensorebene der Kamera und gibt diese als „string“ zurück.

Die Methode **„getImageCoordWorld“** berechnet die Bildpunktposition auf der Sensorebene bezüglich des Weltkoordinatensystems und gibt diesen als Three.js-Vector3 zurück. Hierfür nimmt die Methode die Koordinaten eines Punkts entgegen in Form eines Three.js-Vector3. Es sind jeweils

zwei verschiedenen Vorgehensweisen im Code vorhanden; eine durch Matrixberechnung und die andere durch mathematische Schnittpunktberechnung. Letztere ist im Code auskommentiert.

Die Methode **„getImageCoordCamera“** berechnet die Bildpunktposition auf der Sensorebene bezüglich des Kamerakoordinatensystems und gibt diesen als Three.js-Vector3 zurück. Hierfür nimmt die Methode die Koordinaten eines Punkts entgegen in Form eines Three.js-Vector3.

4.3. CameraCoordinateSystem

Die Klasse „CameraCoordinateSystem“ erbt von der Three.js-Klasse „Group“ (https://threejs.org/docs/index.html#api/en/objects/Group).

Die Klasse dient zur Erstellung der zwei Achsen des Kamerakoordinatensystems (x- und y-Achse) durch Instanziierung.

Die Methode **„update()“** aktualisiert die Achsen, indem sie ihren Abstand und ihre Größe anpasst, sodass sie direkt vor der Kamera liegt.

4.4. CameraGrid

Die Klasse „CameraGrid“ erbt von der Three.js-Klasse „LineSegments“ (https://threejs.org/docs/index.html#api/en/objects/LineSegments) und besteht größtenteils aus dem Sourcecode der Three.js-Klasse „GridHelper“.

Die Klasse dient zur Erstellung des Gitters des Kamerakoordiantensystems durch Instanziierung.

Die Methode **„update“** aktualisiert das Gitter, indem sie seinen Abstand und ihre Größe anpasst, sodass sie direkt vor der Kamera liegt.

4.5. CoordinateSystem

Die Klasse „CoordinateSystem“ erbt von der Three.js-Klasse „Group“ (https://threejs.org/docs/index.html#api/en/objects/Group).

Die Klasse dient zur Erstellung der drei Achsen des Weltkoordinatensystems (x-, y- und z-Achse) durch Instanziierung.

4.6. DefaultModel

Die Klasse „DefaultModel“ erbt von der Three.js-Klasse „Object3D“ (https://threejs.org/docs/index.html#api/en/core/Object3D).

Die Klasse dient zur Erstellung des Standardgebäudes, zusammengesetzt aus Punkten von Klasse „Point“ und Linien von Klasse „Line“ durch Instanziierung.

4.7. Epipole

Die Klasse „Epipole“ erbt von der Klasse „Point“.

Die Klasse dient zur Erstellung der Epipole durch Instanziierung.

4.8. Grid

Die Klasse „Grid“ erbt von der Three.js-Klasse „LineSegments“ (https://threejs.org/docs/index.html#api/en/objects/LineSegments) und besteht größtenteils aus dem Sourcecode der Three.js-Klasse „GridHelper“.

Die Klasse dient zur Erstellung des Gitters des Weltkoordinatensystems durch Instanziierung.

4.9. Line

Die Klasse „Line“ erbt von der Three.js-Klasse „Object3D“ (https://threejs.org/docs/index.html#api/en/core/Object3D).

Die Klasse dient zur Erstellung der Linien, die Punkte verbinden, durch Instanziierung.

Das Attribut **„startPoint“** dient zum Festhalten des Startpunktes einer Linie in Form eines „Point“-Objektes.

Das Attribut **„endPoint“** dient zum Festhalten des Endpunktes einer Linie in Form eines „Point“-Objektes.

Die Methode **„isConnectedToPoint()“** checkt, ob die Linie mit dem im Parameter übergebenen „Point“-Objekt verbunden ist und gibt dementsprechend einen „boolean“-Wert zurück.

Die Methode **„update“** aktualisiert die Linie, damit sie zu ihren Start- und Endpunkten verbunden bleibt.

4.10. Point

Die Klasse „Point“ erbt von der Three.js-Klasse „Object3D“ (https://threejs.org/docs/index.html#api/en/core/Object3D).

Die Klasse dient zur Erstellung der Punkte, die ein Gebäude ausmachen, durch Instanziierung.

4.11. UI

Die Klasse „UI“ ist eine Singleton-Klasse.

Die Klasse dient zur Erstellung von EventListeners für das Tauschen zwischen den Haupt-Tabs der UI „Gebäude“, „Kamera“ und „Einstellungen“ sowie dem Ausklappen der Sub-Tabs „Punkt“, „Linie“, „Linke Kamera“, „Rechte Kamera“ und „Modell“.

In der Methode **„init()“** werden die nicht angezeigten Haupt-Tabs versteckt und zuvor erwähnte EventListeners erstellt.

5. JavaScript Skript

Die JavaScript-Codedatei „script.js“ besteht neben zahlreichen globalen Variablen und Funktionen, aus den zwei Hauptfunktionen init() und animate().

Bei gutem Verständnis von objektorientierter Programmierung und Three.js, sollte der Programmcode durch klare Variablen- und Methodenbenennung sowie Kommentaren selbsterklärend sein.

6. Notiz

Zur kurzen Übersicht über den Code sind im VISMO-Repository ein Klassendiagramm und eine Szenenhierarchie hinterlegt („documents“ --> „UMLs“).

Bei dringenden Rückfragen setzten Sie sich am besten mit dem damaligen Projektleiter und Dozent Prof. Uwe Hahne in Verbindung oder mit einen der damaligen Projektmitgliedern.