TJK-Solutions

Technische Hochschule Wildau
Technical University of Applied Sciences
WILDAU

BELEGARBEIT – VIRTUAL REALITY UND SIMULATION (SoSe 2026)

Echtzeit-3D-Digital-Twin einer Wetterstation

Reale IoT-Messdaten als interaktive 3D-Welt im Browser

Wie lässt sich eine reale IoT-Messstation als webbasierter, echtzeitfähiger 3D-Digital-Twin abbilden, dessen Live-Daten Animation, Beleuchtung und Wetter-Simulation steuern?

Verfasser	Tom Jonas Krüger	**Matrikel**	50178634
Seminargruppe	T24	**Semester**	4. Semester
Art	Projektarbeit / Belegarbeit im Modul Virtual Reality und Simulation		
Praxispartner	TJK-Solutions	**Abgabe**	2. Juni 2026
Prüfer (TH)	Prof. Dr. Alexander Kleinsorge		

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Belegarbeit eigenständig angefertigt und nur die angegebenen Quellen und Hilfsmittel verwendet habe. Verwendete KI-gestützte Werkzeuge wurden als Hilfsmittel zur Implementierung und Recherche eingesetzt; Konzeption, Entscheidungen und Bewertung liegen beim Verfasser.

Datum, Unterschrift (Verfasser/in)

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abk.	Bedeutung
API	Application Programming Interface
AR	Augmented Reality
CORS	Cross-Origin Resource Sharing
CSS	Cascading Style Sheets
DOM	Document Object Model
fps	Frames per Second (Bilder pro Sekunde)
GLB	GL Transmission Format (Binärcontainer von glTF)
glTF	GL Transmission Format
GPU	Graphics Processing Unit
HTML	Hypertext Markup Language
HTTP(S)	Hypertext Transfer Protocol (Secure)
IoT	Internet of Things
JSON	JavaScript Object Notation
JWT	JSON Web Token
LoRaWAN	Long Range Wide Area Network
MQTT	Message Queuing Telemetry Transport
OSM	OpenStreetMap
PBR	Physically Based Rendering
REST	Representational State Transfer
SVG	Scalable Vector Graphics
UI/UX	User Interface / User Experience
UV	Texturkoordinaten (U, V)
VR	Virtual Reality
WebGL	Web Graphics Library
WebXR	Web Extended Reality (Device API)

Glossar

BufferGeometry Geometrie-Repräsentation in three.js, die Vertex-Daten kompakt in typisierten Arrays hält. Ermöglicht das Rendern vieler Punkte (z. B. Regentropfen) in einem einzigen Zeichenaufruf (Draw-Call).

Canvas-Textur Ein HTML-`<canvas>`, das zur Laufzeit bezeichnet und als Textur auf eine 3D-Fläche gelegt wird. Im Projekt zur Darstellung des Live-Dashboards auf dem 3D-Bildschirm genutzt.

Digital Twin Digitales Abbild eines realen Objekts oder Systems, das über Live-Daten mit dem physischen Pendant gekoppelt ist und dessen Zustand in Echtzeit widerspiegelt.

DirectionalLight Gerichtete Lichtquelle (paralleles Licht) in three.js; im Projekt als Sonne genutzt, deren Richtung aus dem berechneten Sonnenstand folgt.

glTF / GLB Offenes Austauschformat der Khronos Group für 3D-Szenen („JPEG der 3D-Welt“). GLB ist die binäre Einzeldatei-Variante.

Import Map Browser-Mechanismus, der „bare“ Modulnamen (z. B. `three`) auf konkrete URLs abbildet.

Orthografische Kamera Kameraprojektion ohne perspektivische Verjüngung; erzeugt den typischen isometrischen Diorama-Look.

PBR Physically Based Rendering – physikalisch motiviertes Materialmodell (z. B. `MeshStandardMaterial`) mit Parametern wie Roughness und Metalness.

Prozedurale Generierung Erzeugung von Inhalten durch ein Programm statt manuellem Modellieren; hier die Blender-Szene per Python-Skript.

Public Access Token Lese-Token, das ThingsBoard für ein als *öffentlich* freigegebenes Gerät ohne persönliche Zugangsdaten ausstellt (`login/public`).

Render-Loop Kontinuierliche Zeichenschleife (`requestAnimationFrame`), die die Szene typischerweise mit 60 fps neu rendert.

Szenengraph Baumartige Anordnung von 3D-Objekten mit Eltern-Kind-Beziehungen; Transformationen vererben sich entlang der Hierarchie.

Shadow Mapping	Verfahren zur Schattenberechnung in Echtzeit über eine Tiefen-Textur aus Sicht der Lichtquelle.
ThingsBoard	Open-Source-IoT-Plattform für Geräteverwaltung, Telemetriespeicherung, Dashboards und REST-/MQTT-Schnittstellen. Im Projekt als Datenquelle (Professional Edition).
three.js	Verbreitete JavaScript-Bibliothek, die WebGL kapselt und High-Level-Konzepte (Szene, Kamera, Material, Loader) bereitstellt.
WebGL	Browser-Schnittstelle für hardwarebeschleunigtes 3D-Rendering auf der GPU.
WebXR	Browser-API für Virtual- und Augmented-Reality-Sitzungen (Headsets, mobile AR).

1 | Einleitung

1.1 Kontext und Auftraggeber

Die digitale Transformation von Kommunen und öffentlichen Einrichtungen stützt sich zunehmend auf das Internet of Things (IoT): Sensoren erfassen kontinuierlich Umweltdaten, die über Plattformen wie ThingsBoard gespeichert und visualisiert werden. Die Darstellung dieser Daten erfolgt heute fast ausschließlich über klassische, zweidimensionale Dashboards – Liniendiagramme, Kacheln und Tabellen. Diese sind informativ, aber wenig anschaulich und für Laien sowie in der Öffentlichkeitsarbeit nur begrenzt ansprechend.

TJK-Solutions ist ein Unternehmen mit Sitz in der Gemeinde Am Mellensee (Landkreis Teltow-Fläming), das innovative IoT-Infrastrukturen aufbaut und Kommunen bei der Digitalisierung begleitet – etwa in Smart-City- und Strandbad-Projekten [TJK1]. Für solche Vorhaben besteht der Wunsch, neben dem klassischen Dashboard auch eine *räumliche, lebendige* Darstellung der Messstandorte anbieten zu können – als Blickfang für Webseiten, Informationstafeln und Präsentationen. Die vorliegende Projektarbeit im Modul *Virtual Reality und Simulation* untersucht und realisiert einen solchen Ansatz prototypisch.

1.2 Ausgangsidee und Themenfindung

Die ursprüngliche Projektidee war ein maßstäblicher 3D-Nachbau der Stadt Jüterbog in Blender – auf Basis von OpenStreetMap-Daten und einem Geometrie-Import-Plugin. In ersten Versuchen erwies sich dieser Weg jedoch als unbefriedigend: Das Plugin erzeugte die Gebäude weitgehend automatisch, die Ergebnisqualität war schwer kontrollierbar, und der gestalterische sowie technische Eigenanteil wäre gering geblieben. Der Ansatz entsprach weder dem eigenen Anspruch noch dem Lernziel des Moduls.

Daraus entstand eine bewusste Neuausrichtung des Themas: Statt eine statische Welt aus offenen Geodaten zu importieren, sollte *etwas aus der realen Welt in die digitale Welt überführt* und dort *lebendig* gemacht werden – ein Digital Twin, der sich in Echtzeit mit seinem physischen Vorbild verändert. Als Vorbild diente eine reale, LoRaWAN-basierte Wetterstation am **Strandbad Kallinchen**, deren Telemetrie über eine ThingsBoard-Instanz von TJK-Solutions verfügbar ist.

Abbildung 1.1 Ausgangsidee der angestrebten 3D-Darstellung (Mockup).

1.3 Zielsetzung und Forschungsfrage

Ziel der Arbeit ist die Konzeption, Implementierung und kritische Bewertung eines webbasierten Echtzeit-3D-Digital-Twins einer realen Wetterstation. Die 3D-Szene wird prozedural in Blender erzeugt und im Browser mit three.js (WebGL) dargestellt; Live-Messwerte aus ThingsBoard steuern Animation (Windrad, Windfahne), die Anzeige auf einem virtuellen Bildschirm sowie eine an die realen Werte gekoppelte Wetter- und Lichtsimulation. Daraus ergibt sich die Forschungsfrage:

Wie lässt sich eine reale IoT-Messstation als webbasierter, echtzeitfähiger 3D-Digital-Twin abbilden, dessen Live-Daten Animation, Beleuchtung und Wetter-Simulation steuern?

Die Beantwortung erfolgt durch Anforderungsableitung, Technologiewahl, eine vollständige prototypische Implementierung sowie eine kritische Reflexion der aufgetretenen Probleme und ihrer Lösungen.

1.4 Struktur der Arbeit

Kapitel 2 ordnet die Begriffe VR, AR, Simulation und Digital Twin ein und beschreibt die Grundlagen des Echtzeit-3D-Renderings im Web. **Kapitel 3** erläutert die eingesetzten Technologien (Blender, glTF, three.js/WebGL, ThingsBoard, Web Audio) inklusive eines Vergleichs möglicher Darstellungsansätze. **Kapitel 4** leitet funktionale und nicht-funktionale Anforderungen ab. **Kapitel 5** beschreibt Architektur und Konzept. **Kapitel 6** dokumentiert die Implementierung. **Kapitel 7** reflektiert kritisch die aufgetretenen Probleme und ihre Lösungen. **Kapitel 8** beantwortet die Forschungsfrage und gibt einen Ausblick.

2 | Grundlagen

2.1 Virtual Reality, Augmented Reality und Simulation

Im Modulkontext sind drei Begriffe abzugrenzen. **Virtual Reality (VR)** bezeichnet eine vollständig computergenerierte, immersive Umgebung, in die der Nutzer – typischerweise per Headset – eintaucht. **Augmented Reality (AR)** überlagert die reale Wahrnehmung mit digitalen Inhalten. **Simulation** ist die rechnergestützte Nachbildung des Verhaltens eines realen Systems über die Zeit [Sim1].

Die vorliegende Arbeit liegt im Schnittfeld von Simulation und Echtzeit-3D-Visualisierung: Es entsteht eine interaktive, dreidimensionale Szene, deren Zustand kontinuierlich aus realen Messdaten abgeleitet wird. Die Anwendung läuft als plattformunabhängige Web-3D-Anwendung; da sie auf three.js/WebGL aufsetzt, ist der Übergang zu immersiver VR/AR über die WebXR-Schnittstelle technisch vorgezeichnet (vgl. Kapitel 8) [WebXR1].

2.2 Digital Twin

Ein *Digital Twin* ist ein digitales Abbild eines physischen Objekts, das über einen Datenstrom mit dem realen System verbunden ist und dessen Zustand widerspiegelt [DT1]. Charakteristisch ist die *lebende* Kopplung: Ändert sich der reale Zustand (z. B. die Windgeschwindigkeit), ändert sich auch das digitale Modell. Genau diese Eigenschaft unterscheidet den Digital Twin von einem rein statischen 3D-Modell und steht im Zentrum dieser Arbeit.

2.3 Echtzeit-3D-Rendering im Browser

Echtzeit-Rendering erzeugt Bilder schnell genug für flüssige Interaktion (Zielgröße 60 fps). Im Browser stellt **WebGL** den hardwarebeschleunigten Zugang zur GPU bereit [WebGL1]. Zentrale Konzepte sind der *Szenengraph* (hierarchische Anordnung von Objekten mit vererbten Transformationen), *Materialien* (unbeleuchtet vs. physikalisch beleuchtet), *Lichtquellen* und *Shadow Mapping* sowie eine kontinuierliche *Render-Schleife*. Transformationen werden über Translation, Rotation und Skalierung beschrieben; Rotationen lassen sich als Euler-Winkel oder Quaternionen darstellen.

3 Technologien

3.1 Blender und prozedurale Szenengenerierung

Blender ist eine quelloffene 3D-Suite mit einer vollständigen Python-API (`bpy`) [Blender1]. Statt die Szene manuell zu modellieren, wird sie im Projekt *prozedural* per Skript erzeugt: Boden, Wetterstation, Sensoren, Bildschirm, Kompass, Zaun und Dekoration werden parametrisch und mit festem Zufalls-Seed reproduzierbar aufgebaut. Dieser Ansatz ist nachvollziehbar, versionierbar und schnell anpassbar.

3.2 glTF / GLB als Austauschformat

Zur Übergabe der Szene an die Web-Anwendung dient **glTF** der Khronos Group – ein offenes, schlankes Laufzeitformat für 3D-Szenen [glTF1]. Die binäre Variante **GLB** fasst Geometrie, Materialien und Szenenhierarchie in einer Datei zusammen. Wichtig für das Projekt: Objektnamen bleiben erhalten, sodass die Web-Anwendung gezielt auf einzelne Teile (z. B. `windRotor`) zugreifen kann.

3.3 three.js und WebGL

three.js kapselt WebGL und stellt High-Level-Bausteine bereit: `Scene`, `Camera`, `Light`, `Material`, Loader sowie Steuerungen [Three1]. Im Projekt werden der `GLTFLoader` (Laden der Szene), `OrbitControls` (Navigation), eine `OrthographicCamera` (Diorama-Look) sowie `DirectionalLight` und `AmbientLight` (Beleuchtung) eingesetzt.

3.4 ThingsBoard und Datenanbindung

ThingsBoard (Professional Edition) ist die IoT-Plattform, auf der die reale Wetterstation ihre Telemetrie ablegt [TB1]. Der lesende Zugriff auf Zeitreihen erfolgt über die REST-Schnittstelle. Die sichere, frontend-taugliche Anbindung über ein *öffentliches Gerät* (Public Access Token) wird in Kapitel 6 ausführlich behandelt [TBpub].

3.5 Web Audio API

Die **Web Audio API** erlaubt die Klangsynthese im Browser ohne externe Audiodateien [Audio1]. Im Projekt werden Wind- und Regengeräusche aus gefiltertem Rauschen sowie Vogelgezwitscher aus frequenzmodulierten Oszillatoren erzeugt.

3.6 Vergleich möglicher Darstellungsansätze

Für die räumliche Live-Darstellung wurden mehrere Ansätze gegenübergestellt (Tabelle 3.1).

Tabelle 3.1 Vergleich der Darstellungsansätze

Kriterium	2D-Dashboard	Web-3D (gewählt)	Native VR-App
Zugang	Browser	Browser	Headset/Store
Installation	keine	keine	hoch
Immersion	gering	mittel	hoch
Anschaulichkeit	mittel	hoch	sehr hoch
Aufwand	gering	mittel	hoch
VR/AR-Pfad	nein	ja (WebXR)	ja
Eignung Webseite	ja	ja	nein

Der webbasierte 3D-Ansatz verbindet niedrige Zugangshürde (kein Headset, keine Installation) mit hoher Anschaulichkeit und hält zugleich über WebXR den Weg zu immersiver VR/AR offen. Für den Einsatz auf Webseiten von Kommunal- und Strandbad-Projekten ist er damit am besten geeignet.

4 Anforderungen

4.1 Anwendungsfälle

Tabelle 4.1 Identifizierte Anwendungsfälle

Als (Akteur)	möchte ich	Priorität
Bürger / Badegast	den aktuellen Wetterzustand am Strandbad anschaulich sehen	Hoch
Webseiten-Besucher	eine ansprechende 3D-Darstellung statt nur Diagramme erleben	Hoch
TJK-Solutions	Live-Telemetrie eines Standorts räumlich präsentieren	Hoch
TJK-Solutions	die Lösung als Zusatzprodukt neben dem Dashboard anbieten	Mittel
Betrachter	den Tagesverlauf (Sonne, Wetter) als Demo nachvollziehen	Mittel

4.2 Funktionale Anforderungen

Tabelle 4.2 Funktionale Anforderungen

ID	Komponente	Anforderung	Quelle
F001	3D-Szene	Prozedural in Blender erzeugte Diorama-Szene, Export als GLB	TJK
F002	Datenanbindung	Lesen der Live-Telemetrie der realen Station aus ThingsBoard	TJK
F003	Sicherheit	Zugriff ohne persönliche Zugangsdaten im Browser-Code	TJK
F004	Animation	Windrad dreht proportional zur Windgeschwindigkeit	TJK
F005	Animation	Windfahne zeigt korrekt die reale Windrichtung (Kompassbezug)	TJK
F006	Anzeige	Live-Dashboard (Werte + Verlauf) auf dem 3D-Bildschirm	TJK
F007	Verlauf	Auswählbarer Zeitbereich (2 h/12 h/24 h/7 d)	TJK
F008	Simulation	Beleuchtung folgt dem realen Sonnenstand (Tag/Nacht)	TJK
F009	Simulation	Wolken/Regen an reale Messwerte gekoppelt	TJK
F010	Audio	Wetter-/Umgebungsgeräusche, lautstärkegekoppelt	TJK
F011	Demo	Zeitraffer-Modus zur Vorführung des Tagesverlaufs	TJK

4.3 Nicht-funktionale Anforderungen

Tabelle 4.3 Nicht-funktionale Anforderungen

ID	Kriterium	Anforderung	Quelle
NF001	Performance	Flüssige Darstellung (Zielgröße ca. 60 fps)	TJK
NF002	Plattform	Läuft im Browser ohne Installation/Plugins	TJK
NF003	Robustheit	Bei Verbindungsfehler letzter Wert halten, kein Absturz	TJK
NF004	Wartbarkeit	Konfiguration (URL, Keys, Parameter) zentral gepflegt	TJK
NF005	Erweiterbarkeit	VR/AR-Pfad (WebXR) offenhalten	TJK
NF006	Reproduzierbar	Szene durch Skript jederzeit identisch erzeugbar	TJK

5 Konzept und Architektur

5.1 Überblick

Die Architektur trennt klar zwischen *Erzeugung*, *Daten* und *Laufzeit* (Abbildung 5.1). Ein Leitgedanke prägt das gesamte Konzept: *Blender baut den Körper, ThingsBoard liefert die Daten, die Web-Anwendung macht ihn lebendig.* Blender erzeugt die statische Szene (GLB); die Web-Anwendung lädt diese, fragt die Live-Telemetrie ab und steuert daraus Animation, Beleuchtung und Wetter-Simulation.

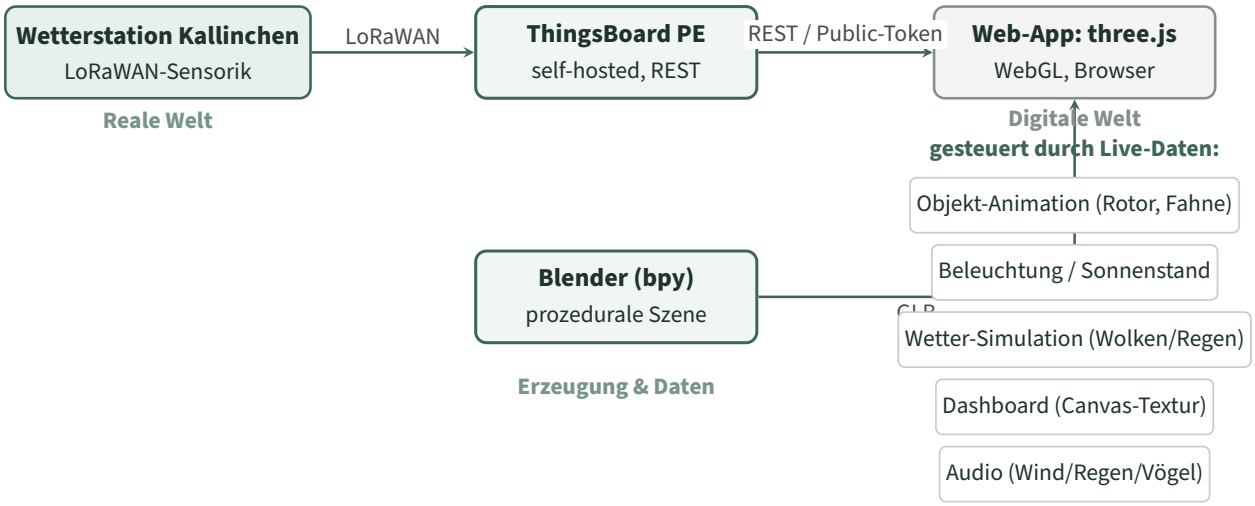

Abbildung 5.1 Systemarchitektur: Die reale Station speist ThingsBoard; Blender liefert die GLB-Szene; die Web-Anwendung lädt das Modell und steuert mit den Live-Daten Animation, Beleuchtung, Wetter und Audio.

5.2 Szenengraph und Steuerlogik

Alle beweglichen Teile sind im GLB *benannt* und werden zur Laufzeit über ihren Namen gefunden. Die Wetterstation ist als Eltern-Objekt (`weatherStation`) modelliert, die Sensoren sind Kinder – eine Drehung des Elternobjekts dreht das gesamte Sensorkreuz mit. Tabelle 5.1 fasst die datengesteuerten Objekte zusammen.

Tabelle 5.1 Datengesteuerte Objekte der Szene

Objekt	**Steuerung**	**Datenquelle**
`windRotor`	Drehgeschwindigkeit $\omega = v \cdot K$	wind_speed
`windVane`	Ausrichtung $\theta = 45° -$ Richtung	wind_direction
`screenSurface`	Live-Dashboard als Canvas-Textur	alle Werte
DirectionalLight	Sonnenstand (Ort + Uhrzeit)	Systemzeit
Wolken / Regen	Dichte/Intensität	light_intensity, rain

6 Implementierung

6.1 Prozedurale Szene in Blender

Das Skript `build_scene.py` räumt zunächst die Szene auf (idempotenter Lauf), definiert eine weiche Low-Poly-Farbpalette und baut die Geometrie aus Grundkörpern auf. Entscheidend für die spätere Animation ist, dass die Dreh-Objekte ihren *Origin* exakt auf der Rotationsachse erhalten. Listing 6.1 zeigt exemplarisch den Aufbau des Anemometers und das Setzen des Drehpunkts.

```
HUB = Vector((sx - 0.45, sy, GROUND_TOP + 4.05))    # Nabe = Drehachse
rotor_parts = [add_cyl("rotorHub", HUB, 0.05, 0.18, 10, M["dark"])]
for i in range(3):                                   # 3 Schalen am Kreuz
    ang = math.radians(i * 120)
    ax = HUB.x + math.cos(ang) * 0.42
    ay = HUB.y + math.sin(ang) * 0.42
    rotor_parts.append(add_ico(f"rotorCup_{i}", (ax, ay, HUB.z), 0.16, 1,
        M["dark"]))
windRotor = join(rotor_parts, "windRotor")
set_origin_to(windRotor, HUB)                        # Origin auf die Achse
parent_to(windRotor, station)
```

Listing 6.1 Anemometer mit korrektem Dreh-Origin (Auszug)

Abschließend exportiert das Skript die Szene als GLB (`bpy.ops.export_scene.gltf`, Format `GLB`, `export_yup=True`).

6.2 Datenanbindung und Sicherheit

Der lesende Zugriff auf ThingsBoard-Telemetrie ist der sicherheitskritische Kern. Hier wurde eine bewusste Designentscheidung getroffen: Der **Geräte-Zugangstoken** kann in ThingsBoard ausschließlich Daten *senden*, nicht lesen; ein **Benutzer-JWT** ist ein persönlicher Account-Schlüssel, läuft ab und darf nicht in Client-Code stehen. Da das Gerät in ThingsBoard als *öffentlich* freigegeben ist, holt sich die Web-Anwendung über `login/public` einen reinen **Lese-Token** – ohne persönliche Zugangsdaten (Listing 6.2). Damit ist Anforderung F003 erfüllt.

```
async function publicLogin() {
  const res = await fetch(`${CFG.TB_URL}/api/auth/login/public`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ publicId: CFG.PUBLIC_ID }),
```

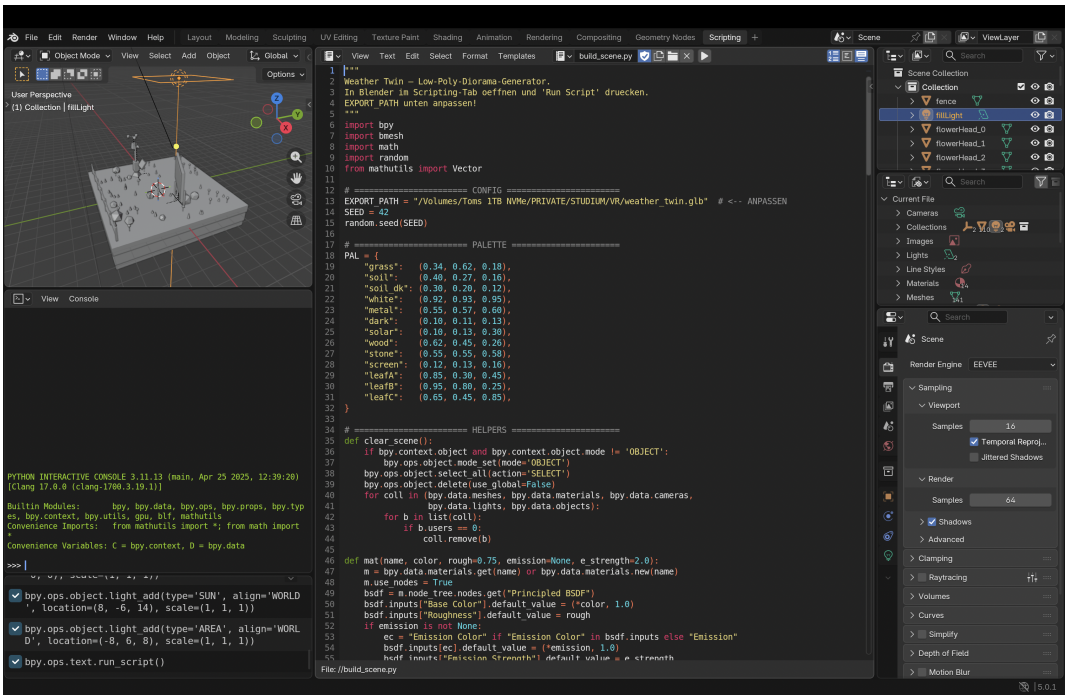

Abbildung 6.1 Ausführung des Generator-Skripts im Blender-Scripting-Tab.

```
  });
  token = (await res.json()).token;      // read-only
}
```

Listing 6.2 Public-Login: read-only Token ohne Account-Schlüssel (JavaScript)

Die Werte werden zyklisch (alle 5 s) abgefragt; zusätzlich lassen sich Zeitreihen für auswählbare Bereiche (2 h/12 h/24 h/7 d) aggregiert laden (F007). Die von der Station in *Meter pro Sekunde* gelieferte Windgeschwindigkeit wird intern in *km/h* umgerechnet. Bei Verbindungsfehlern wird der letzte gültige Wert gehalten (NF003).

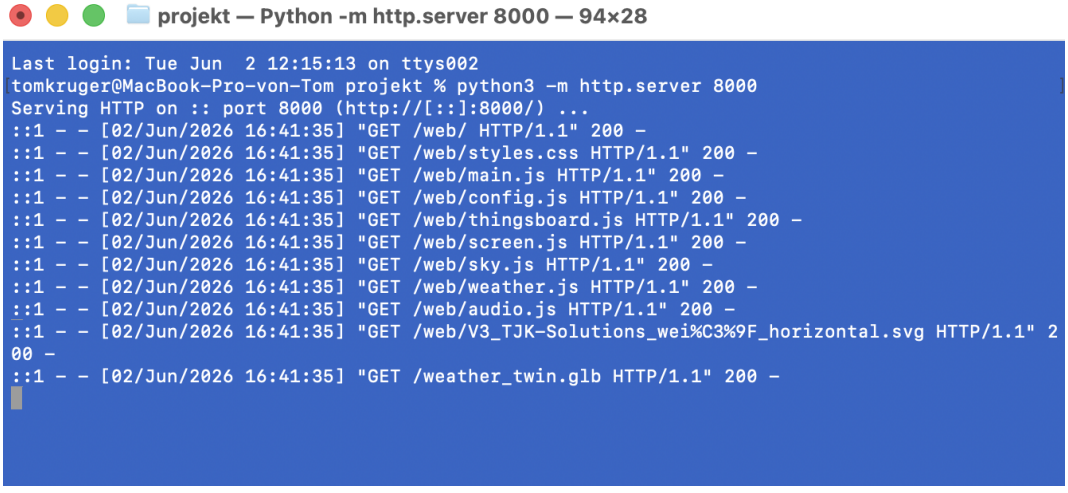

Abbildung 6.2 Start der Web-Anwendung (lokaler HTTP-Server) und Aufruf im Browser.

6.3 Objekt-Steuerung und Kalibrierung

Im Render-Loop wird das Windrad proportional zur Windgeschwindigkeit gedreht; bei nahezu Windstille steht es still. Die Windfahne wird sanft auf die Zielrichtung interpoliert und pendelt leicht (Böeneffekt). Die korrekte Nordausrichtung erforderte eine Kalibrierung über die gesamte Transformationskette (Bauausrichtung der Fahne, $-45°$-Drehung der Station, Blender-Z-up nach glTF-Y-up, three.js-Yaw), die analytisch zu $\theta = 45° -$ Richtung führt; ein Kompass im Modell dient als sichtbare Referenz (Listing 6.3).

```
if (windRotor) {
  const eff = targetSpeed < 0.1 ? 0 : targetSpeed;   // 0 -> steht still
  windRotor.rotation.y += eff * CFG.ROTOR_K * dt;
}
if (windVane) {
  const goal = targetDirRad + boeenJitter;
  let diff = ((goal - windVane.rotation.y + Math.PI) % (2*Math.PI)) - Math
     .PI;
  windVane.rotation.y += diff * Math.min(1, dt * 2);  // sanft nachfuehren
}
```

Listing 6.3 Animation: Rotor $\propto$ Wind, Fahne $\rightarrow$ Richtung (Auszug, JavaScript)

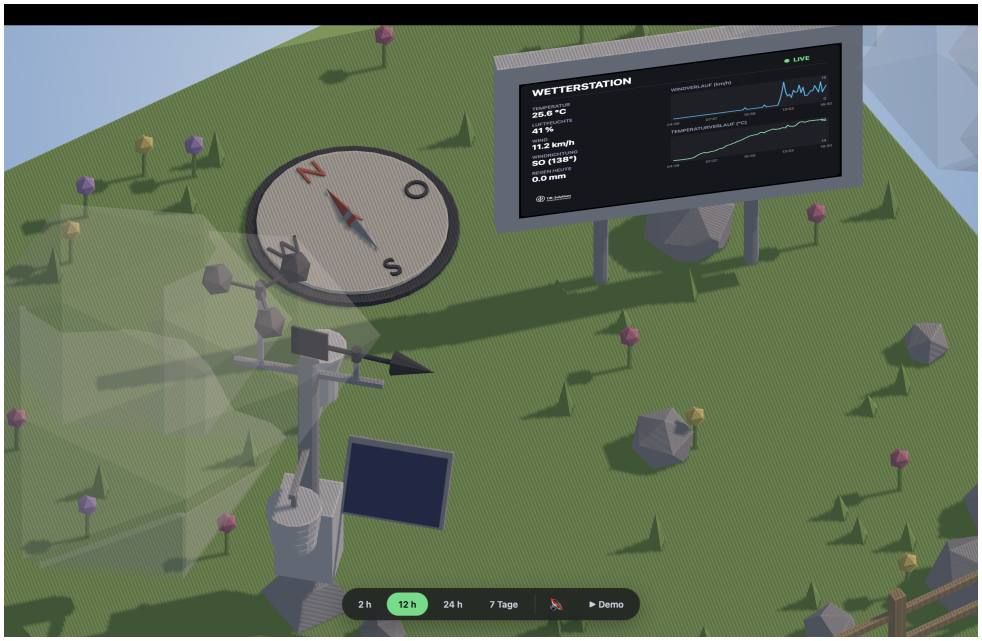

Abbildung 6.3 Kompass im Modell als Referenz für die kalibrierte Windfahne.

6.4 Beleuchtung aus realem Sonnenstand

Die Sonne ist eine `DirectionalLight`, deren Position aus geographischer Breite/Länge (Strandbad Kallinchen) und der aktuellen Uhrzeit berechnet wird: aus Deklination und Stundenwinkel

ergeben sich Elevation und Azimut [Solar1]. Farbe und Intensität folgen der Sonnenhöhe (hell-weiß am Tag, warm-orange zur Dämmerung, mondblau in der Nacht); eine sichtbare Sonnenkugel und ein Mond ergänzen die Stimmung. Dichte Bewölkung verdunkelt die Szene und lässt die Sonne verschwinden (F008).

6.5 Wetter-Simulation und Audio

Wolken sind driftende Low-Poly-Körper, deren Dichte an die gemessene Lichtstärke gekoppelt ist. Regen wird als Partikelsystem realisiert: Tropfen entstehen auf Wolkenhöhe und fallen herab, am Boden erscheinen Aufprall-Spritzer. Aus Effizienzgründen liegen alle Tropfen in einer `BufferGeometry` und werden in einem einzigen Draw-Call gezeichnet (F009, NF001). Die Geräusche (Wind, Regen, Vogelgezwitscher) werden mit der Web Audio API synthetisiert und in der Lautstärke an die Messwerte gekoppelt (F010). Ein Demo-Modus rafft einen Tagesverlauf inklusive Sonnenuntergang und Regenschauer zusammen (F011).

Abbildung 6.4 Wetter-Simulation: Sonnenuntergang bzw. Regen mit gekoppelter Beleuchtung.

6.6 Live-Dashboard auf dem 3D-Bildschirm

Das Dashboard (Titel, Messwerte, zwei Verlaufs-Diagramme, Firmenlogo) wird in einem HTML-Canvas gezeichnet und als Textur auf eine dedizierte, sauber UV-abgewickelte Bildschirmfläche (`screenSurface`) gelegt (F006). So erscheint die Anzeige als Teil der 3D-Welt und bleibt beim Drehen der Kamera korrekt am Bildschirm.

Abbildung 6.5 Fertiger Digital Twin: Diorama der Wetterstation mit Live-Dashboard auf dem 3D-Bildschirm.

7 Tests und kritische Reflexion

7.1 Funktionsnachweis

Alle Anforderungen wurden im laufenden Prototyp überprüft (Tabelle 7.1). Der Digital Twin läuft im Browser mit den echten Live-Daten der Station Strandbad Kallinchen.

Tabelle 7.1 Funktionsnachweis (Auszug)

Anf.	Testfall	Ergebnis	Status
F001	Szene aus Blender exportiert (GLB)	lädt im Browser	**OK**
F002/3	Live-Telemetrie über Public-Token	Werte aktuell	**OK**
F004	Rotor dreht mit Windgeschwindigkeit	bestätigt	**OK**
F005	Fahne zeigt reale Richtung (Kompass)	kalibriert	**OK**
F006/7	Dashboard + Zeitbereiche	funktioniert	**OK**
F008	Sonnenstand Tag/Nacht	sichtbar	**OK**
F009/10	Wolken/Regen + Audio	gekoppelt	**OK**
F011	Demo-Zeitraffer	läuft	**OK**
NF001	Flüssige Darstellung	ca. 60 fps	**OK**

7.2 Aufgetretene Probleme und Lösungen

Wissenschaftliche Redlichkeit erfordert die ehrliche Dokumentation der Hürden. Tabelle 7.2 fasst die wesentlichen Probleme und ihre Lösungen zusammen; ausgewählte Punkte werden anschließend vertieft.

Tabelle 7.2 Probleme und Lösungen im Projektverlauf

Problem	Lösung
Jüterbog-Import (OSM-Plugin) unbefriedigend	Themenwechsel: realen Standort als Digital Twin lebendig machen
Geräte-Token kann nicht lesen	Öffentliches Gerät + read-only Public-Token (`login/public`)
Cloudflare blockt Test-Requests (Fehler 1010)	Browser-konforme Header; im Browser unkritisch
Wind in m/s statt km/h	Interne Umrechnung ($\times 3{,}6$) + Anpassung des Drehfaktors
Dashboard-Textur verzerrt/gespiegelt (Würfel-UVs)	Dedizierte Plane mit sauberen UVs (`screenSurface`)
Fahne zeigt falsche Richtung	Kalibrierung über Transformationskette: $\theta = 45° -$ Richtung
„bare“ Import `three` schlägt fehl	Import Map im HTML
ES-Module aus Browser-Cache	Hartes Neuladen / Cache deaktivieren
Screen-Beine nach Drehung versetzt	Drehung auf das ganze Display-Objekt statt einzelner Teile
Rotor und Fahne kollidieren	Anemometer höher gesetzt, Fahne quer ausgerichtet
Nacht zu dunkel	Mondhelles Grundlicht angehoben

7.2.1 Themenwechsel von Jüterbog zum Digital Twin

Der gescheiterte OSM-Import war rückblickend ein wertvoller Wendepunkt: Statt automatisch generierter, schwer kontrollierbarer Geometrie entstand ein Projekt mit klarem Eigenanteil und echtem Mehrwert – die Kopplung einer realen Messstation an ein lebendiges 3D-Modell. Die Entscheidung folgte dem Anspruch, *etwas aus der realen Welt in die digitale zu bringen*.

7.2.2 Datensicherheit als bewusste Entscheidung

Der naheliegende Weg über einen Benutzer-JWT wäre unsicher gewesen (Account-Schlüssel im Frontend). Die Lösung über ein öffentliches Gerät und einen reinen Lese-Token trennt Lese- und Schreibrechte sauber und ist für öffentlich einsehbare Wetterdaten angemessen.

7.2.3 Kalibrierung der Windfahne

Die korrekte Nordausrichtung war nicht trivial, da mehrere Koordinatensysteme und Drehungen ineinandergreifen. Die analytische Herleitung ($\theta = 45° -$ Richtung) und ein sichtbarer Kompass als Referenz machten das Ergebnis überprüfbar.

7.3 Grenzen

Der Prototyp bildet eine einzelne Station ab; eine Mehrstationen-/Mehrstandort-Verwaltung ist noch nicht umgesetzt. Die Sonnenstandsberechnung ist eine gute Näherung, kein voll astronomisches Modell. Wolken und Regen sind stilisiert, nicht physikalisch simuliert. Eine immersive VR/AR-Sitzung (WebXR) ist vorbereitet, aber noch nicht aktiviert.

8 Fazit und Ausblick

8.1 Beantwortung der Forschungsfrage

Wie lässt sich eine reale IoT-Messstation als webbasierter, echtzeitfähiger 3D-Digital-Twin abbilden, dessen Live-Daten Animation, Beleuchtung und Wetter-Simulation steuern?

Die Frage kann anhand des realisierten Prototyps positiv beantwortet werden. Die Trennung in *Erzeugung* (prozedurale Blender-Szene, GLB), *Daten* (ThingsBoard, sicherer Lese-Token) und *Laufzeit* (three.js/WebGL) hat sich als tragfähig erwiesen. Benannte Objekte im GLB ermöglichen die gezielte, datengesteuerte Animation; der reale Sonnenstand und an Messwerte gekoppelte Wetter-Effekte erzeugen eine glaubwürdige, *lebende* Welt. Damit ist ein echter Digital Twin entstanden – als anschauliche, browserbasierte Ergänzung zum klassischen 2D-Dashboard.

8.2 Mehrwert für TJK-Solutions

Die Lösung ist als wiederverwendbares Muster konzipiert: Sie lässt sich auf weitere TJK-Solutions-Projekte übertragen – etwa Smart-City- oder Strandbad-Vorhaben – und dort als **3D-Ergänzung neben dem klassischen Dashboard** auf Webseiten anbieten. Der räumliche, lebendige Eindruck eignet sich besonders für Öffentlichkeitsarbeit, Informationstafeln und Präsentationen.

8.3 Ausblick

WebXR (VR/AR): Da die Anwendung auf three.js basiert, ist der Schritt zu einer immersiven VR-Begehung oder einer AR-Einblendung des Standorts technisch naheliegend. **Mehrere Standorte:** Eine Verwaltung mehrerer Stationen würde aus dem Prototyp eine Produktplattform machen. **Erweiterte Sensorik:** Weitere Messgrößen (UV-Index, Luftdruck, Wasserstand) lassen sich analog visualisieren. **Realistischere Simulation:** Physikalisch fundierte Wolken-/Niederschlagsmodelle und ein vollständiges astronomisches Sonnenmodell würden die Glaubwürdigkeit weiter steigern. **Containerisierung:** Eine Bereitstellung als Docker-Container kapselt die Web-Anwendung (und optional einen Daten-Proxy) und ermöglicht eine einfache, reproduzierbare Auslieferung und Kompatibilität über verschiedene Umgebungen hinweg.

Quellenverzeichnis

[Three1] three.js Authors; *three.js Documentation – Manual und API*; https://threejs.org/docs/; zuletzt abgerufen am 02.06.2026

[glTF1] Khronos Group; *glTF 2.0 Specification*; https://registry.khronos.org/glTF/specs/2.0/glTF-2.0.html; zuletzt abgerufen am 02.06.2026

[WebGL1] Khronos Group; *WebGL Specification*; https://www.khronos.org/webgl/; zuletzt abgerufen am 02.06.2026

[WebXR1] W3C; *WebXR Device API*; https://www.w3.org/TR/webxr/; zuletzt abgerufen am 02.06.2026

[Blender1] Blender Foundation; *Blender Python API Documentation (bpy)*; https://docs.blender.org/api/current/; zuletzt abgerufen am 02.06.2026

[TB1] ThingsBoard; *ThingsBoard REST API Documentation*; https://thingsboard.io/docs/reference/; zuletzt abgerufen am 02.06.2026

[TBpub] ThingsBoard; *Public Dashboards & Devices / Public Login*; https://thingsboard.io/docs/; zuletzt abgerufen am 02.06.2026

[Audio1] Mozilla Developer Network (MDN); *Web Audio API*; https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API; zuletzt abgerufen am 02.06.2026

[Solar1] NOAA Global Monitoring Laboratory; *Solar Position Calculation (Solar Geometry)*; https://gml.noaa.gov/grad/solcalc/; zuletzt abgerufen am 02.06.2026

[DT1] Grieves, M.; Vickers, J.; *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*; in: Transdisciplinary Perspectives on Complex Systems, Springer, 2017, S. 85–113; https://doi.org/10.1007/978-3-319-38756-7_4

[Sim1] Banks, J. et al.; *Discrete-Event System Simulation*; Pearson, 5. Auflage, 2010

[TJK1] TJK-Solutions; *Unternehmenswebseite*; https://tjk-solutions.de; zuletzt abgerufen am 02.06.2026

Abbildungsverzeichnis

Tabellenverzeichnis