WEBGL İLE K.T.Ü. KAMPÜSÜNDE SANAL GEZİNTİ

Danışman: Öğr. Gör. Ömer ÇAKIR

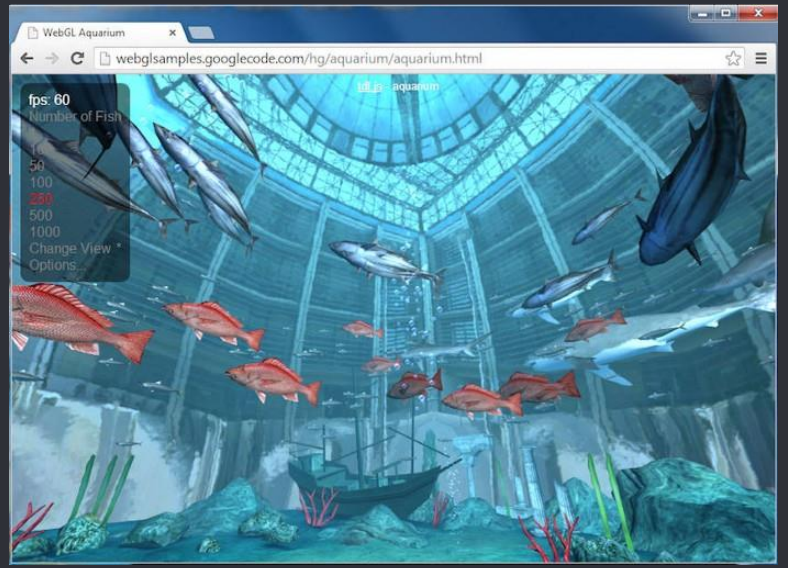
Abdullah Bayram
Berk Civelek

2021

WEBGL

WebGL

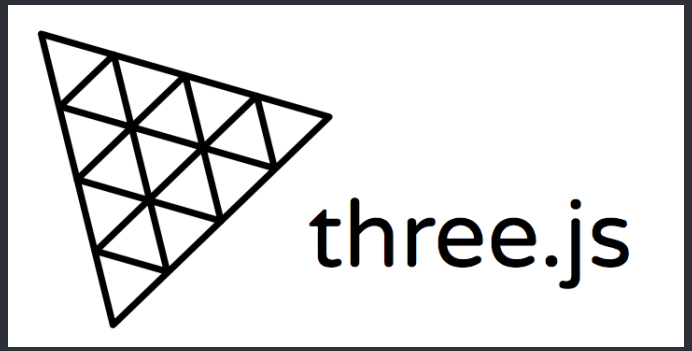
WebGL, yüksek performanslı, interaktif 3D ve 2D grafikler sunan, uyumlu olduğu tarayıcılarda eklentilere gerek duymadan çalışabilen bir grafik API'dır.

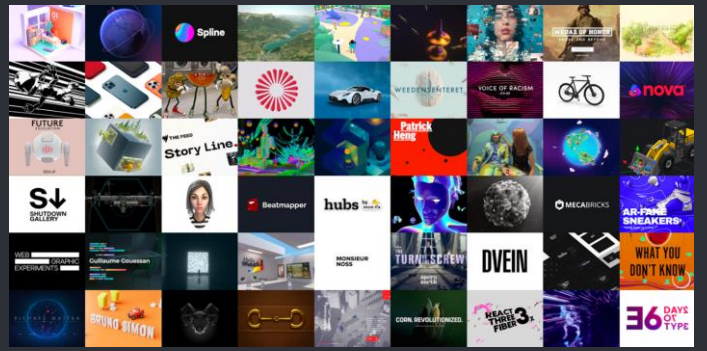


THREE.JS

Three.js

- Kullanımı kolay, hafif, 3 boyutlu bir kütüphane
- 3D matematik <-> 3D grafik





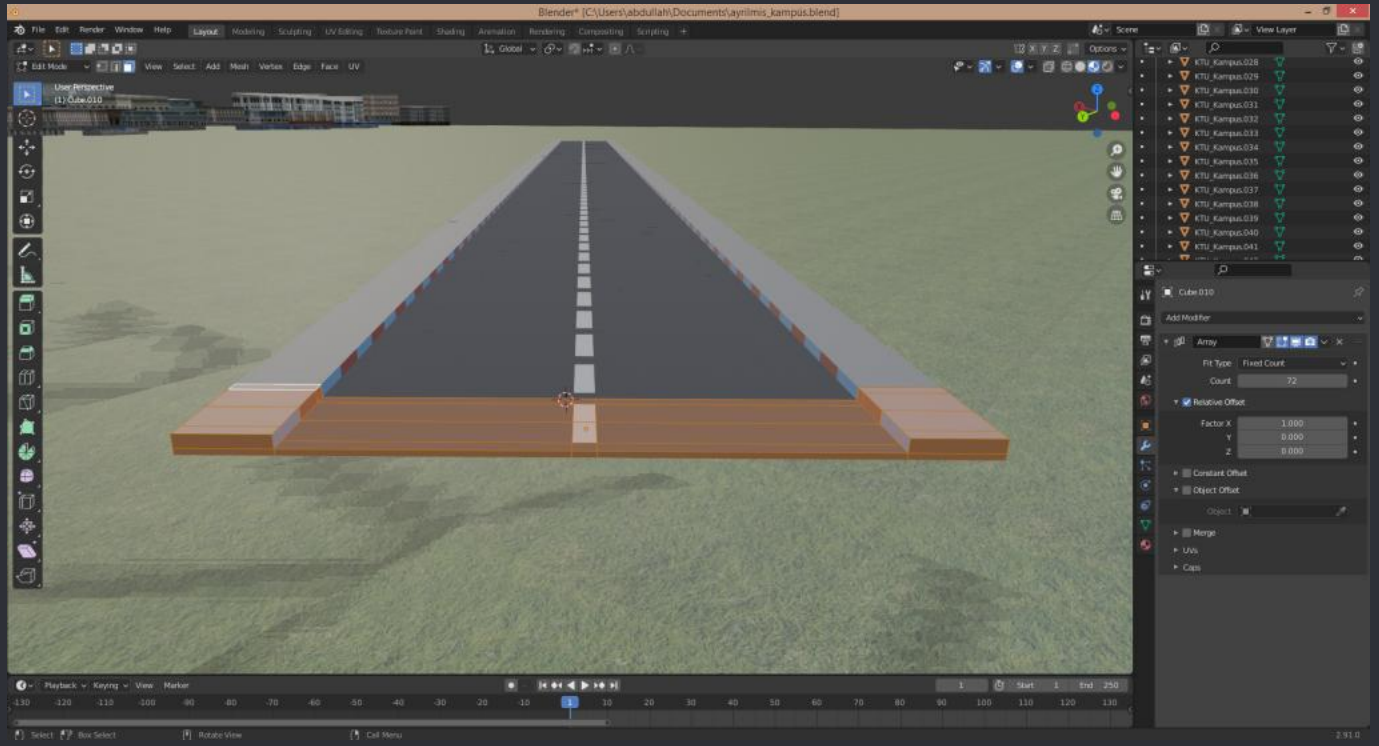
Blender

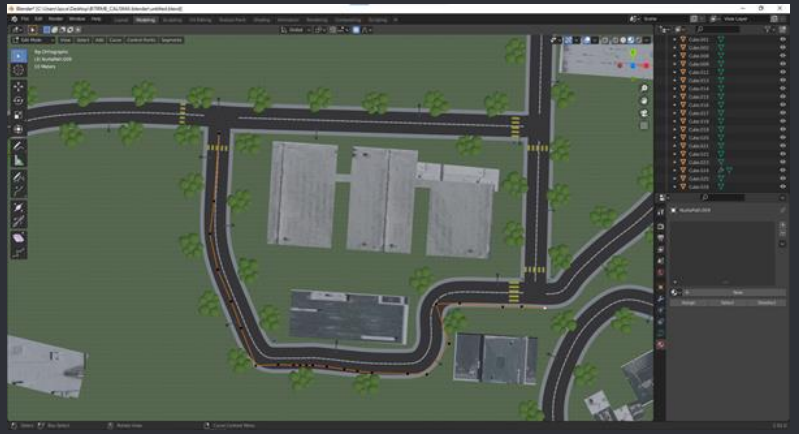
- Ton Roosendaal
- Ücretsiz
- Açık Kaynak Kodlu
- Geniş Uyumluluk
- Kullanıcı Dostu
- Geniş Kullanıcı Kitlesi ve Rehberleri

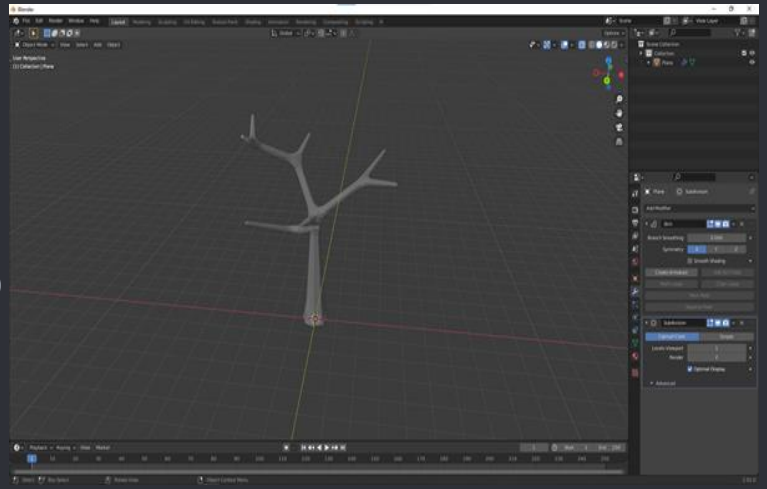
*A render of a blender that rendered in Blender. We guess...

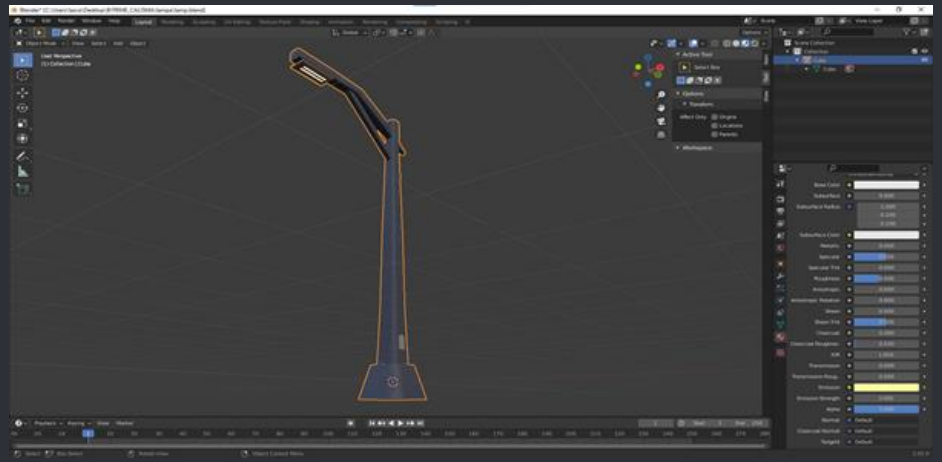
YAPILAN ÇALIŞMALAR, TESTLER

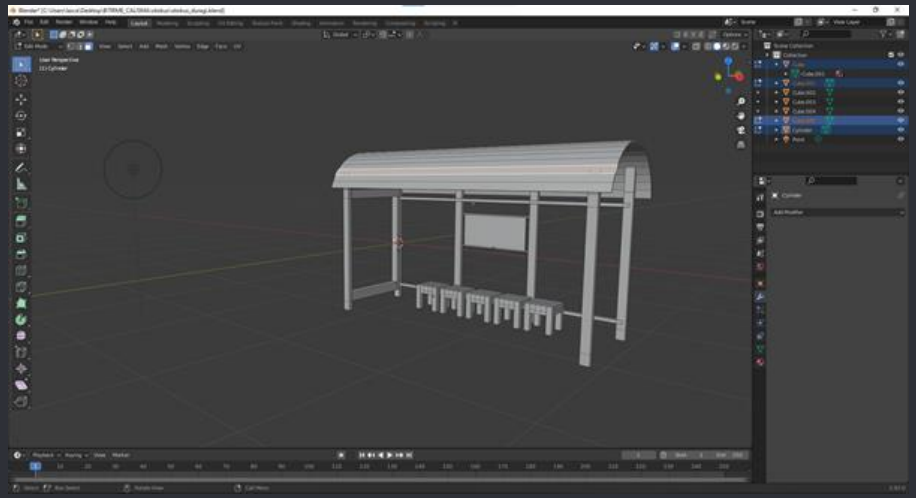
Yapılan Çalışmalar, Testler

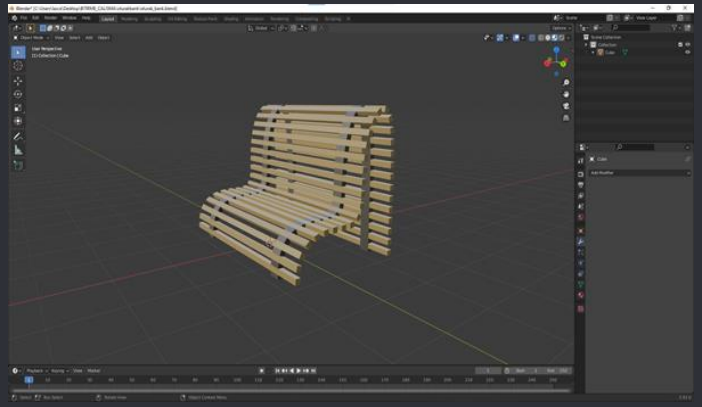


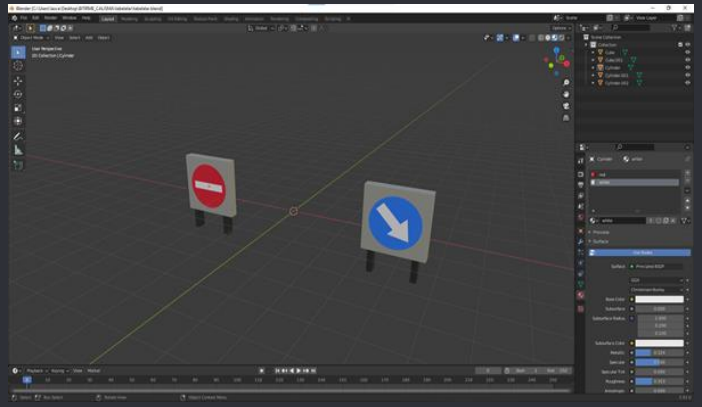


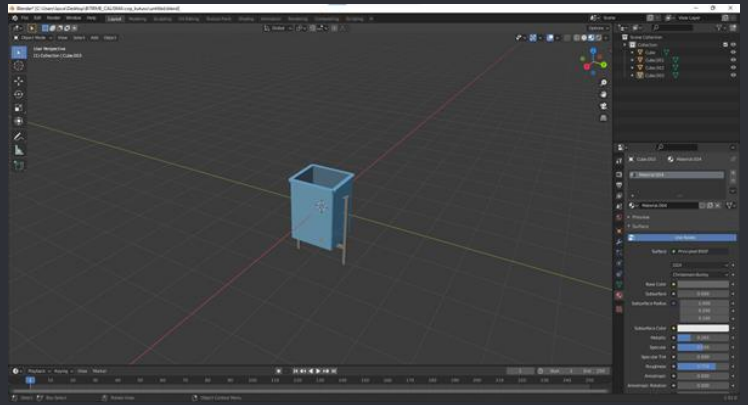


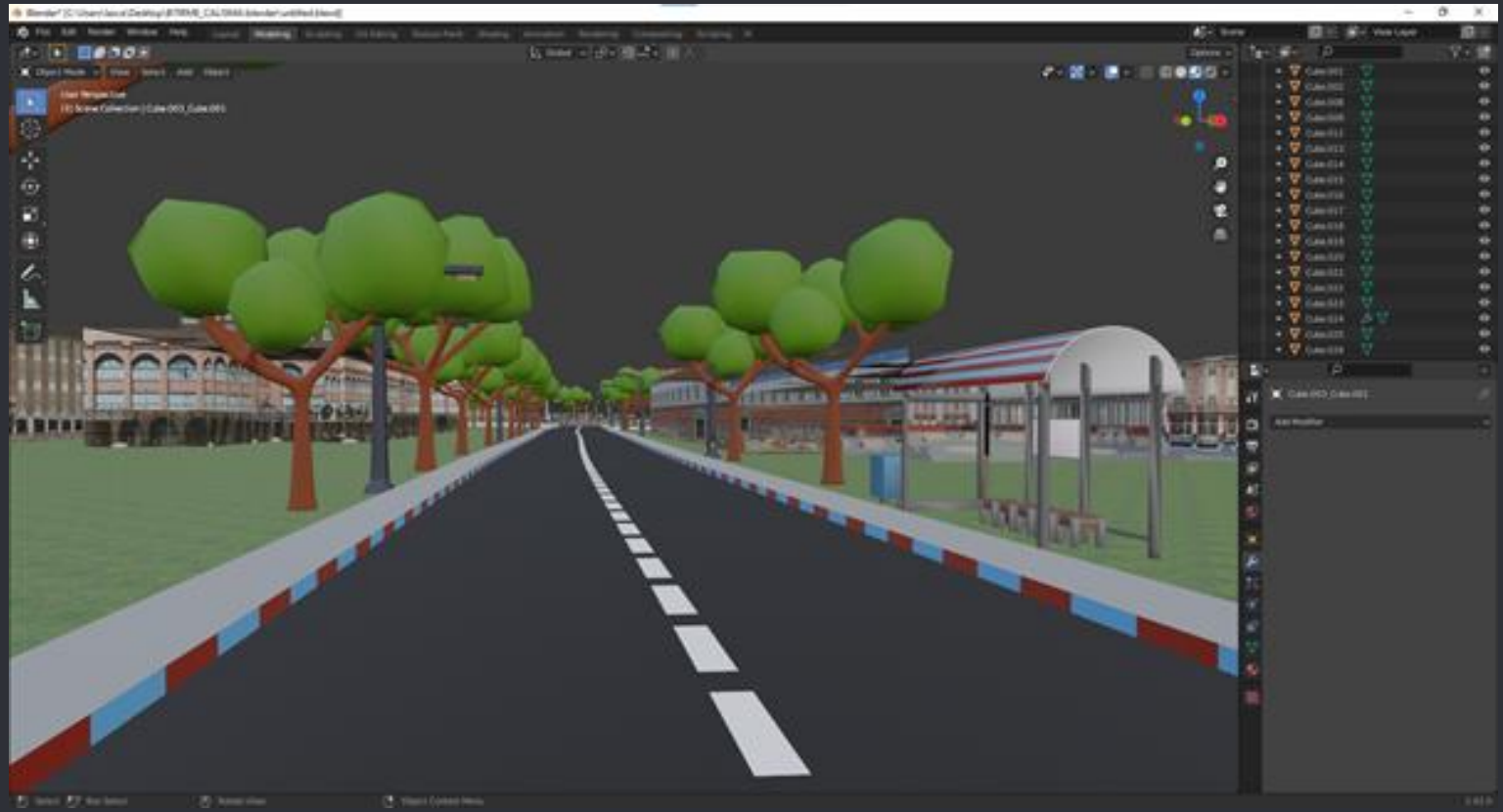












Three.js (Sahne ve Kampüs)

```
// Canvas
const canvas = document.querySelector("canvas.webgl");

// Scene
const scene = new THREE.Scene();
```

```
// GLTF Loader
const campusLoader = new GLTFLoader();

// Load Campus
campusLoader.load("/assets/objects/kampus.glb", function (glb) {
  // Make all materials' depth to true to prevent x-ray view
  glb.parser.getDependencies("material").then((materials) => {
    materials.forEach((material) => (material.depthWrite = true));
  });

  scene.add(glb.scene);
});
```

Işıklandırmalar

```
// AmbientLight
const ambientLight = new THREE.AmbientLight(0xffffff, 1);
scene.add(ambientLight);

// Directional Light
const directionalLight = new THREE.DirectionalLight(0xffffff, 0.6);
directionalLight.position.set(0, 1, 0);
directionalLight.castShadow = true;
scene.add(directionalLight);
```

Window Boyutlandırmaları

```
const updateOnResize = () ⇒ {
  // Update sizes
  sizes.width = window.innerWidth;
  sizes.height = window.innerHeight;

  // Update camera
  camera.aspect = sizes.width / sizes.height;
  camera.updateProjectionMatrix();

  // Update renderer
  renderer.setSize(sizes.width, sizes.height);
  renderer.setClearColor(0×ccccff);
  renderer.setPixelRatio(Math.min(window.devicePixelRatio, 2));
};
```

Kamera Yapılandırmaları

```
// Base camera
const orbitCamera = new THREE.PerspectiveCamera(
  75,
  sizes.width / sizes.height,
  0.1,
  1000
);
orbitCamera.position.set(130, 8, -235);
orbitCamera.rotation.set(-1.5, 0, 0);
scene.add(orbitCamera);

// Free Roam Camera
const freeRoamCamera = new THREE.PerspectiveCamera(
  75,
  sizes.width / sizes.height,
  0.1,
  1000
);
const { x, y, z } = vectors[0][1];
freeRoamCamera.position.set(x, y, z);
freeRoamCamera.rotation.set(0, 4.5, 0);
scene.add(freeRoamCamera);
```

Sahneler için Tazeleyiciler

```
/**
 * Renderer
 */
const renderer = new THREE.WebGLRenderer({
  canvas: canvas,
  alpha: true,
});
renderer.setSize(sizes.width, sizes.height);
renderer.setPixelRatio(Math.min(window.devicePixelRatio, 2));
```

Sürekli tazeleme için loop fonksiyonu

```
/**
 * Animate
 */
function animate() {

  // Renders
  // renderer.render(scene, !isSplineCamera ? camera : splineCamera);
  renderer.render(scene, currentCamera);

  // If label renderer option is true, this must be in the animate
  if (isNodeViewsOn) {
    labelRenderer.render(scene, currentCamera);
  }

  // Call tick again on the next frame
  requestAnimationFrame(animate);
}
```

DOM Elementleri için dinleyiciler

```
// Event Listeners for global window
window.addEventListener("resize", updateOnResize, false);

// Event Listeners for DOM elements
document
  .querySelector("#toggleNodeViews")
  .addEventListener("click", toggleNodeViewsOnTheScene);

document
  .querySelector("#changeToFreeRoamCamera")
  .addEventListener("click", changeToFreeRoamCamera);

document
  .querySelector("#changeToOrbitCamera")
  .addEventListener("click", changeToOrbitCamera);
```

Kamera Kontrolleri (Serbest Gezinme)

```
// This controls provide hiding cursor and event binding to the
// first parameter (camera)
const controls = new PointerLockControls(freeRoamCamera, document.body);
```

```
/*
 * FreeRoam Management
 * Key Events
 */
const onKeyDown = function (event) {
  switch (event.keyCode) {
    case 87: // w
      moveForward = true;
      break;
    case 65: // a
      moveLeft = true;
      break;
    case 83: // s
      moveBackward = true;
      break;
    case 68: // d
      moveRight = true;
      break;
  }
};
```

```
const onKeyUp = function (event) {
  switch (event.keyCode) {
    case 87: // w
      moveForward = false;
      break;
    case 65: // a
      moveLeft = false;
      break;
    case 83: // s
      moveBackward = false;
      break;
    case 68: // d
      moveRight = false;
      break;
  }
};
```

Kuş sesleri

```
// Add Birds audio
const listener = new THREE.AudioListener();
freeRoamCamera.add(listener);

// Create a global audio source
const sound = new THREE.Audio(listener);

// Load bird sound and set it as the Audio object's buffer
const audioLoader = new THREE.AudioLoader();
audioLoader.load("/assets/audio/birds.wav", (buffer) => {
  sound.setBuffer(buffer);
  sound.setLoop(true);
  sound.setVolume(0.12);
  sound.play();
});
```

Sinematik Gezinme

```
function startCinematicTrip() {
  // Get selections
  const source = document.querySelector("#sourceSelection").value;
  const destination = document.querySelector("#destinationSelection").value;

  if (!source || !destination) {
    throw new Error("Please entry start and destination point!");
  }

  if (source === destination) {
    throw new Error("Source and destination can't be same!");
  }

  const shortestWayArray = helpers.getShortestWayPointsArray(
    source,
    destination
  );

  const curve = helpers.createCurve(shortestWayArray);

  // Set last coordinate as a look at object for cinematic camera
  lookAtObject = helpers.createLookAtObject(
    shortestWayArray[shortestWayArray.length - 1]
  );

  // Set curve to global cinematic path
  cinematicPath = curve;

  startCinematicViewClock();
  isCinematic = true;
}
```

```
function stopCinematicTrip() {
  // Get selections
  const source = document.querySelector("#sourceSelection");
  const destination = document.querySelector("#destinationSelection");

  // Reset selections
  source.value = "0";
  destination.value = "0";

  // Reset variables for path;
  lookAtObject = null;
  cinematicPath = null;

  stopCinematicViewClock();
  isCinematic = false;
}
```

```
function startCinematicViewClock() {
  clock.start();
}

function stopCinematicViewClock() {
  clock.stop();
}
```

DEMO

TEŞEKKÜRLER