const viewer = new Cesium.Viewer("cesiumContainer", {

infoBox: false,

selectionIndicator: false,

shadows: true,

shouldAnimate: true,

});

let entity;

function getColorBlendMode(colorBlendMode) {

return Cesium.ColorBlendMode[colorBlendMode.toUpperCase()];

}

function getColor(colorName, alpha) {

const color = Cesium.Color[colorName.toUpperCase()];

return Cesium.Color.fromAlpha(color, parseFloat(alpha));

}

// The viewModel tracks the state of our mini application.

const viewModel = {

color: "Red",

colors: ["White", "Red", "Green", "Blue", "Yellow", "Gray"],

alpha: 1.0,

colorBlendMode: "Highlight",

colorBlendModes: ["Highlight", "Replace", "Mix"],

colorBlendAmount: 0.5,

colorBlendAmountEnabled: false,

silhouetteColor: "Red",

silhouetteColors: ["Red", "Green", "Blue", "Yellow", "Gray"],

silhouetteAlpha: 1.0,

silhouetteSize: 2.0,

};

// Convert the viewModel members into knockout observables.

Cesium.knockout.track(viewModel);

// Bind the viewModel to the DOM elements of the UI that call for it.

const toolbar = document.getElementById("toolbar");

Cesium.knockout.applyBindings(viewModel, toolbar);

Cesium.knockout.getObservable(viewModel, "color").subscribe(function (newValue) {

if (entity) {

entity.model.color = getColor(newValue, viewModel.alpha);

}

});

Cesium.knockout.getObservable(viewModel, "alpha").subscribe(function (newValue) {

if (entity) {

entity.model.color = getColor(viewModel.color, newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "colorBlendMode")

.subscribe(function (newValue) {

const colorBlendMode = getColorBlendMode(newValue);

if (entity) {

entity.model.colorBlendMode = colorBlendMode;

}

viewModel.colorBlendAmountEnabled =

colorBlendMode === Cesium.ColorBlendMode.MIX;

});

Cesium.knockout

.getObservable(viewModel, "colorBlendAmount")

.subscribe(function (newValue) {

if (entity) {

entity.model.colorBlendAmount = parseFloat(newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteColor")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteColor = getColor(newValue, viewModel.silhouetteAlpha);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteAlpha")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteColor = getColor(viewModel.silhouetteColor, newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteSize")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteSize = parseFloat(newValue);

}

});

function createModel(url, height) {

viewer.entities.removeAll();

const centerPosition = Cesium.Cartesian3.fromDegrees(-122.4194, 37.7749, height); // San Francisco coordinates

const radius = 1000; // Radius of the circular path

entity = viewer.entities.add({

name: url,

position: centerPosition, // Initial position at the center of the circle

model: {

uri: url, // The CesiumDroid model URI from the Cesium asset library

minimumPixelSize: 128,

maximumScale: 20000,

color: getColor(viewModel.color, viewModel.alpha),

colorBlendMode: getColorBlendMode(viewModel.colorBlendMode),

colorBlendAmount: parseFloat(viewModel.colorBlendAmount),

silhouetteColor: getColor(viewModel.silhouetteColor, viewModel.silhouetteAlpha),

silhouetteSize: parseFloat(viewModel.silhouetteSize),

},

});

viewer.trackedEntity = entity;

flyInCircles(centerPosition, radius);

}

function flyInCircles(centerPosition, radius) {

let angle = 0;

const speed = 0.005; // Change in angle per frame

// Update the position of the entity in a circular path

const updatePosition = () => {

angle += speed;

// Calculate the new position in Cartesian coordinates based on the angle

const offset = new Cesium.Cartesian3(

radius \* Math.cos(angle), // x component of the circular path

radius \* Math.sin(angle), // y component of the circular path

0 // z stays the same (flat circular motion)

);

// Apply the offset to the center position

const newPosition = Cesium.Cartesian3.add(centerPosition, offset, new Cesium.Cartesian3());

// Set the new position of the entity

entity.position = newPosition;

// Keep updating the position while the animation is running

requestAnimationFrame(updatePosition);

};

updatePosition();

}

// The CesiumDroid model URL

const cesiumDroidURL = "https://assets.cesium.com/2/models/CesiumDroid/CesiumDroid.glb";

const options = [

{

text: "CesiumDroid",

onselect: function () {

createModel(cesiumDroidURL, 500.0); // Starting height for the CesiumDroid

},

},

// You can add other options if needed

];

Sandcastle.addToolbarMenu(options);

Sandcastle.addToggleButton("Shadows", viewer.shadows, function (checked) {

viewer.shadows = checked;

});

const viewer = new Cesium.Viewer("cesiumContainer", {

infoBox: false,

selectionIndicator: false,

shadows: true,

shouldAnimate: true,

});

let entity;

function getColorBlendMode(colorBlendMode) {

return Cesium.ColorBlendMode[colorBlendMode.toUpperCase()];

}

function getColor(colorName, alpha) {

const color = Cesium.Color[colorName.toUpperCase()];

return Cesium.Color.fromAlpha(color, parseFloat(alpha));

}

// The viewModel tracks the state of our mini application.

const viewModel = {

color: "Red",

colors: ["White", "Red", "Green", "Blue", "Yellow", "Gray"],

alpha: 1.0,

colorBlendMode: "Highlight",

colorBlendModes: ["Highlight", "Replace", "Mix"],

colorBlendAmount: 0.5,

colorBlendAmountEnabled: false,

silhouetteColor: "Red",

silhouetteColors: ["Red", "Green", "Blue", "Yellow", "Gray"],

silhouetteAlpha: 1.0,

silhouetteSize: 2.0,

};

// Convert the viewModel members into knockout observables.

Cesium.knockout.track(viewModel);

// Bind the viewModel to the DOM elements of the UI that call for it.

const toolbar = document.getElementById("toolbar");

Cesium.knockout.applyBindings(viewModel, toolbar);

Cesium.knockout.getObservable(viewModel, "color").subscribe(function (newValue) {

if (entity) {

entity.model.color = getColor(newValue, viewModel.alpha);

}

});

Cesium.knockout.getObservable(viewModel, "alpha").subscribe(function (newValue) {

if (entity) {

entity.model.color = getColor(viewModel.color, newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "colorBlendMode")

.subscribe(function (newValue) {

const colorBlendMode = getColorBlendMode(newValue);

if (entity) {

entity.model.colorBlendMode = colorBlendMode;

}

viewModel.colorBlendAmountEnabled =

colorBlendMode === Cesium.ColorBlendMode.MIX;

});

Cesium.knockout

.getObservable(viewModel, "colorBlendAmount")

.subscribe(function (newValue) {

if (entity) {

entity.model.colorBlendAmount = parseFloat(newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteColor")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteColor = getColor(newValue, viewModel.silhouetteAlpha);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteAlpha")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteColor = getColor(viewModel.silhouetteColor, newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteSize")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteSize = parseFloat(newValue);

}

});

function createModel(url, startHeight) {

viewer.entities.removeAll();

const landingPosition = Cesium.Cartesian3.fromDegrees(-122.4194, 37.7749, 0); // San Francisco ground level

entity = viewer.entities.add({

name: url,

position: Cesium.Cartesian3.fromDegrees(-122.4194, 37.7749, startHeight), // Initial high position

model: {

uri: url, // The CesiumDroid model URI from the Cesium asset library

minimumPixelSize: 128,

maximumScale: 20000,

color: getColor(viewModel.color, viewModel.alpha),

colorBlendMode: getColorBlendMode(viewModel.colorBlendMode),

colorBlendAmount: parseFloat(viewModel.colorBlendAmount),

silhouetteColor: getColor(viewModel.silhouetteColor, viewModel.silhouetteAlpha),

silhouetteSize: parseFloat(viewModel.silhouetteSize),

},

});

viewer.trackedEntity = entity;

// Animate the model falling towards the ground

fallAndLand(entity, startHeight, landingPosition);

}

function fallAndLand(entity, startHeight, landingPosition) {

const fallSpeed = 50; // Controls how fast the object falls

const totalFrames = 300; // Total frames to animate the fall

let currentHeight = startHeight;

let frame = 0;

// Animate the object falling from the sky

const fallInterval = setInterval(() => {

if (frame < totalFrames) {

currentHeight = startHeight - (fallSpeed \* frame); // Decrease height over time

// Update the entity's position

const position = Cesium.Cartesian3.fromDegrees(-122.4194, 37.7749, currentHeight);

entity.position = position;

frame++;

} else {

// Once the object reaches the ground, stop the fall and set the final position

clearInterval(fallInterval);

// Ensure the entity reaches the landing position

entity.position = landingPosition;

}

}, 1000 / 60); // 60 FPS

}

// The CesiumDroid model URL

const cesiumDroidURL = "https://assets.cesium.com/2/models/CesiumDroid/CesiumDroid.glb";

const options = [

{

text: "CesiumDroid Falling",

onselect: function () {

createModel(cesiumDroidURL, 1000.0); // Start height for the CesiumDroid (1 km above ground)

},

},

// You can add other options if needed

];

Sandcastle.addToolbarMenu(options);

Sandcastle.addToggleButton("Shadows", viewer.shadows, function (checked) {

viewer.shadows = checked;

});

const viewer = new Cesium.Viewer("cesiumContainer", {

infoBox: false,

selectionIndicator: false,

shadows: true,

shouldAnimate: true,

});

let entity;

function getColorBlendMode(colorBlendMode) {

return Cesium.ColorBlendMode[colorBlendMode.toUpperCase()];

}

function getColor(colorName, alpha) {

const color = Cesium.Color[colorName.toUpperCase()];

return Cesium.Color.fromAlpha(color, parseFloat(alpha));

}

// The viewModel tracks the state of our mini application.

const viewModel = {

color: "Red",

colors: ["White", "Red", "Green", "Blue", "Yellow", "Gray"],

alpha: 1.0,

colorBlendMode: "Highlight",

colorBlendModes: ["Highlight", "Replace", "Mix"],

colorBlendAmount: 0.5,

colorBlendAmountEnabled: false,

silhouetteColor: "Red",

silhouetteColors: ["Red", "Green", "Blue", "Yellow", "Gray"],

silhouetteAlpha: 1.0,

silhouetteSize: 2.0,

};

// Convert the viewModel members into knockout observables.

Cesium.knockout.track(viewModel);

// Bind the viewModel to the DOM elements of the UI that call for it.

const toolbar = document.getElementById("toolbar");

Cesium.knockout.applyBindings(viewModel, toolbar);

Cesium.knockout.getObservable(viewModel, "color").subscribe(function (newValue) {

if (entity) {

entity.model.color = getColor(newValue, viewModel.alpha);

}

});

Cesium.knockout.getObservable(viewModel, "alpha").subscribe(function (newValue) {

if (entity) {

entity.model.color = getColor(viewModel.color, newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "colorBlendMode")

.subscribe(function (newValue) {

const colorBlendMode = getColorBlendMode(newValue);

if (entity) {

entity.model.colorBlendMode = colorBlendMode;

}

viewModel.colorBlendAmountEnabled =

colorBlendMode === Cesium.ColorBlendMode.MIX;

});

Cesium.knockout

.getObservable(viewModel, "colorBlendAmount")

.subscribe(function (newValue) {

if (entity) {

entity.model.colorBlendAmount = parseFloat(newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteColor")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteColor = getColor(newValue, viewModel.silhouetteAlpha);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteAlpha")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteColor = getColor(viewModel.silhouetteColor, newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteSize")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteSize = parseFloat(newValue);

}

});

function createModel(url, startHeight) {

viewer.entities.removeAll();

const startingPosition = Cesium.Cartesian3.fromDegrees(-122.4194, 37.7749, startHeight); // San Francisco location

const startingHeading = Cesium.Math.toRadians(135); // Facing direction

const pitch = 0;

const roll = 0;

const hpr = new Cesium.HeadingPitchRoll(startingHeading, pitch, roll);

const orientation = Cesium.Transforms.headingPitchRollQuaternion(startingPosition, hpr);

entity = viewer.entities.add({

name: url,

position: startingPosition,

orientation: orientation,

model: {

uri: url,

minimumPixelSize: 128,

maximumScale: 20000,

color: getColor(viewModel.color, viewModel.alpha),

colorBlendMode: getColorBlendMode(viewModel.colorBlendMode),

colorBlendAmount: parseFloat(viewModel.colorBlendAmount),

silhouetteColor: getColor(viewModel.silhouetteColor, viewModel.silhouetteAlpha),

silhouetteSize: parseFloat(viewModel.silhouetteSize),

},

});

viewer.trackedEntity = entity;

// Start the backward flight animation

flyBackwards(entity, startingPosition);

}

function flyBackwards(entity, startPosition) {

const flightSpeed = 0.001; // Controls how fast the object moves

const maxDistance = 1.0; // Maximum distance the object will fly backward (in degrees)

let currentPosition = startPosition;

let distanceTravelled = 0;

// Animate the model flying backwards

const flightInterval = setInterval(() => {

if (distanceTravelled < maxDistance) {

// Move backwards along the longitude

const newLongitude = Cesium.Math.toRadians(Cesium.Cartesian3.toDegrees(currentPosition).longitude) - flightSpeed;

const newLatitude = Cesium.Cartesian3.toDegrees(currentPosition).latitude;

const newAltitude = Cesium.Cartesian3.toDegrees(currentPosition).height;

currentPosition = Cesium.Cartesian3.fromDegrees(newLongitude, newLatitude, newAltitude);

// Update the entity's position

entity.position = currentPosition;

distanceTravelled += flightSpeed;

} else {

// Stop the movement once the object has flown the maximum distance

clearInterval(flightInterval);

}

}, 1000 / 60); // 60 FPS

}

// The CesiumDroid model URL

const cesiumDroidURL = "https://assets.cesium.com/2/models/CesiumDroid/CesiumDroid.glb";

const options = [

{

text: "CesiumDroid Flying Backwards",

onselect: function () {

createModel(cesiumDroidURL, 1000.0); // Start height for the CesiumDroid (1 km above ground)

},

},

// You can add other options if needed

];

Sandcastle.addToolbarMenu(options);

Sandcastle.addToggleButton("Shadows", viewer.shadows, function (checked) {

viewer.shadows = checked;

});

const viewer = new Cesium.Viewer("cesiumContainer", {

infoBox: false,

selectionIndicator: false,

shadows: true,

shouldAnimate: true,

});

let entity;

// Predefined Wi-Fi hotspots (for simulation purposes)

const wifiHotspots = [

{ location: Cesium.Cartesian3.fromDegrees(-122.4194, 37.7749, 0), signalStrength: 100 }, // San Francisco

{ location: Cesium.Cartesian3.fromDegrees(-73.935242, 40.73061, 0), signalStrength: 80 }, // New York

{ location: Cesium.Cartesian3.fromDegrees(2.3522, 48.8566, 0), signalStrength: 90 }, // Paris

];

function getColorBlendMode(colorBlendMode) {

return Cesium.ColorBlendMode[colorBlendMode.toUpperCase()];

}

function getColor(colorName, alpha) {

const color = Cesium.Color[colorName.toUpperCase()];

return Cesium.Color.fromAlpha(color, parseFloat(alpha));

}

// The viewModel tracks the state of our mini application.

const viewModel = {

color: "Red",

colors: ["White", "Red", "Green", "Blue", "Yellow", "Gray"],

alpha: 1.0,

colorBlendMode: "Highlight",

colorBlendModes: ["Highlight", "Replace", "Mix"],

colorBlendAmount: 0.5,

colorBlendAmountEnabled: false,

silhouetteColor: "Red",

silhouetteColors: ["Red", "Green", "Blue", "Yellow", "Gray"],

silhouetteAlpha: 1.0,

silhouetteSize: 2.0,

gpsData: "", // Display GPS data

wifiStrength: "", // Display Wi-Fi signal strength

};

// Convert the viewModel members into knockout observables.

Cesium.knockout.track(viewModel);

// Bind the viewModel to the DOM elements of the UI that call for it.

const toolbar = document.getElementById("toolbar");

Cesium.knockout.applyBindings(viewModel, toolbar);

Cesium.knockout.getObservable(viewModel, "color").subscribe(function (newValue) {

if (entity) {

entity.model.color = getColor(newValue, viewModel.alpha);

}

});

Cesium.knockout.getObservable(viewModel, "alpha").subscribe(function (newValue) {

if (entity) {

entity.model.color = getColor(viewModel.color, newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "colorBlendMode")

.subscribe(function (newValue) {

const colorBlendMode = getColorBlendMode(newValue);

if (entity) {

entity.model.colorBlendMode = colorBlendMode;

}

viewModel.colorBlendAmountEnabled =

colorBlendMode === Cesium.ColorBlendMode.MIX;

});

Cesium.knockout

.getObservable(viewModel, "colorBlendAmount")

.subscribe(function (newValue) {

if (entity) {

entity.model.colorBlendAmount = parseFloat(newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteColor")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteColor = getColor(newValue, viewModel.silhouetteAlpha);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteAlpha")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteColor = getColor(viewModel.silhouetteColor, newValue);

}

});

Cesium.knockout

.getObservable(viewModel, "silhouetteSize")

.subscribe(function (newValue) {

if (entity) {

entity.model.silhouetteSize = parseFloat(newValue);

}

});

function createModel(url, startHeight) {

viewer.entities.removeAll();

const startingPosition = Cesium.Cartesian3.fromDegrees(-122.4194, 37.7749, startHeight); // San Francisco location

const startingHeading = Cesium.Math.toRadians(135); // Facing direction

const pitch = 0;

const roll = 0;

const hpr = new Cesium.HeadingPitchRoll(startingHeading, pitch, roll);

const orientation = Cesium.Transforms.headingPitchRollQuaternion(startingPosition, hpr);

entity = viewer.entities.add({

name: url,

position: startingPosition,

orientation: orientation,

model: {

uri: url,

minimumPixelSize: 128,

maximumScale: 20000,

color: getColor(viewModel.color, viewModel.alpha),

colorBlendMode: getColorBlendMode(viewModel.colorBlendMode),

colorBlendAmount: parseFloat(viewModel.colorBlendAmount),

silhouetteColor: getColor(viewModel.silhouetteColor, viewModel.silhouetteAlpha),

silhouetteSize: parseFloat(viewModel.silhouetteSize),

},

});

viewer.trackedEntity = entity;

// Start the backward flight animation

flyBackwards(entity, startingPosition);

}

function flyBackwards(entity, startPosition) {

const flightSpeed = 0.001; // Controls how fast the object moves

const maxDistance = 1.0; // Maximum distance the object will fly backward (in degrees)

let currentPosition = startPosition;

let distanceTravelled = 0;

// Animate the model flying backwards

const flightInterval = setInterval(() => {

if (distanceTravelled < maxDistance) {

// Move backwards along the longitude

const newLongitude = Cesium.Math.toRadians(Cesium.Cartesian3.toDegrees(currentPosition).longitude) - flightSpeed;

const newLatitude = Cesium.Cartesian3.toDegrees(currentPosition).latitude;

const newAltitude = Cesium.Cartesian3.toDegrees(currentPosition).height;

currentPosition = Cesium.Cartesian3.fromDegrees(newLongitude, newLatitude, newAltitude);

// Update the entity's position

entity.position = currentPosition;

// Simulate GPS data (lat, lon, alt)

viewModel.gpsData = `GPS - Latitude: ${Cesium.Cartesian3.toDegrees(currentPosition).latitude.toFixed(4)}, Longitude: ${Cesium.Cartesian3.toDegrees(currentPosition).longitude.toFixed(4)}, Altitude: ${Cesium.Cartesian3.toDegrees(currentPosition).height.toFixed(2)}m`;

// Simulate Wi-Fi signal strength

updateWiFiStrength(currentPosition);

distanceTravelled += flightSpeed;

} else {

// Stop the movement once the object has flown the maximum distance

clearInterval(flightInterval);

}

}, 1000 / 60); // 60 FPS

}

function updateWiFiStrength(currentPosition) {

let closestDistance = Number.MAX\_VALUE;

let wifiSignal = 0;

wifiHotspots.forEach(hotspot => {

const distance = Cesium.Cartesian3.distance(currentPosition, hotspot.location);

if (distance < closestDistance) {

closestDistance = distance;

wifiSignal = hotspot.signalStrength / (distance + 1); // Inversely proportional to distance

}

});

// Update Wi-Fi signal strength

viewModel.wifiStrength = `Wi-Fi Signal Strength: ${Math.max(0, wifiSignal.toFixed(0))}%`;

}

// The CesiumDroid model URL

const cesiumDroidURL = "https://assets.cesium.com/2/models/CesiumDroid/CesiumDroid.glb";

const options = [

{

text: "CesiumDroid Flying Backwards",

onselect: function () {

createModel(cesiumDroidURL, 1000.0); // Start height for the CesiumDroid (1 km above ground)

},

},

// You can add other options if needed

];

Sandcastle.addToolbarMenu(options);

Sandcastle.addToggleButton("Shadows", viewer.shadows, function (checked) {

viewer.shadows = checked;

});

EXPLANATION:

The codes simulate various moving interactive actions for a droid flying over San Francisco city, There are 5 simulations: the first one showing an object flying in reverse over the city, the second one showing a droid flying in circles, the third one showing an object falling from sky with physics and landing on the ground, the fourth and 5th showing an airplane flying with intended sensors for wifi and gps. The animations can run smoothly and without any bug and they show an approximate code simulation of a real object flying on the sky. There is only a small bug showing in the 3rd animation on the video in the landing but the code works. The animations move in a velocity from around 20- 50 km by hour average and the item backwards moves at a flight speed.