Script 04

- Code organization and the scene graph.
- Perspective Camera vs Orthographic Camera.
- Animation: querying the scene graph and applying local transformations.
- Interaction: responding to keyboard and mouse events.
- Setup – Importing libraries / modules

For some of the examples – running locally – you will need to download the latest three.js release from threejs.org.

And to run a simple local server – it can be easily done with Visual Studio Code and LiveServer.

1.1 Code organization and the scene graph

Open the folder **01_Ex_Code_Organization**.

Analyze the **four files** making up the example.

Notice the following:

- The Javascript functions have been divided into **helper functions** – which stay mostly the same for different examples – and **scene modeling functions**.

- The **scene graph** stores the various scene models and establishes hierarchical relationships between them.

Tasks:

- Add **a second spotlight** to the scene; observe the illumination and shadow effects.

- Add some **tree models** to the scene – use the **createTree()** function from the previous examples.

- Change the Perspective Camera to the **Orthographic Camera**. Pay attention to the definition of the **view-volume**. Try to spot the **differences** in the rendered scene.

1.2 Animation – Local transformations

Open the folder **02_Ex_Animation**.

Analyze the changes made in the Javascript files that animate the movement of the spotlight.

Notice the following:

- The **computeFrame()** function computes and updates **transformation parameters** for the scene elements being animated.

- The **scene graph can be queried** to access scene elements.

Tasks:

Add the following animation behavior:

- The **cylinder** rotates around its XX axis.

- The **cube** rotates around its YY axis.

- The **sphere** slides on the plane, along the ZZ direction.

1.3 Interaction – Keyboard

Open the folder **03_Ex_Interaction**.

Events are now being processed:

- See what happens when the **browser window is resized** – compare with the previous examples.

- Analyze how the **AWSD keys** are used to control the position of the cube.

Tasks:

- Use the **+ and – keys** to control the size of the sphere – suggestion: it should always touch the ground plane.

- Use the **cursor keys** to control the position of the cylinder.

Extra Task:

- In a similar way, use the **mouse buttons** to control model and scene features.

1.4 Interaction – Camera control using the mouse

Open the folder **04_Ex_Interaction**.

Three.js offers different extensions to control the camera (rotation, pan, zoom, etc.) using the mouse.

OrbitControls is commonly used:

- Left mouse button: scene rotation

- Right mouse button: scene pan
- Mouse scroll wheel: scene zoom

OrbitControls is **not directly accessible** from the three.js library and should be loaded as a Javascript script separately.

In this example, **local versions** of three.js and the OrbitControls libraries are being used.

Task:

- Identify the **simple code additions** that were made that allow controlling the camera with OrbitControls.

1.5 Setup – Importing libraries / modules

The folder **05_Ex_Setup** contains different versions of the same basic example, which differ only in the way the necessary libraries and/or modules are imported:

- **Version 1 –** including **local copies** of three.js and OrbitControls.js.
- **Version 2 –** using a **local version** of the entire **three.js package**.
- **Version 3** – using a **local server.**
- **Version 4** – using **ES6** and a **local server.**
- **Version 5** – using **ES6** and an additional library: **ConvexGeometry**.

Tasks:

- Analyze and **run the different versions** of the basic example.
- Identify and correct **possible issues** – for the **ES6** versions, you might need to define an **Import Map**.
- Use the **cursor keys** to control the position of the cylinder.