# Script 03

- Code organization.
- Scene modeling: the scene graph.
- Animation: applying local transformations and global transformations to models.
- Interaction: responding to keyboard and mouse events.

### 3.1 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** – a tree structure – stores the various scene models and establishes hierarchical relationships between them.

**Tasks:**

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

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

### 3.2 Animation – Local transformations

Open the folder **02_ex_Animation**.

Analyze the changes made in the Javascript files that animate the movement of the spot-light.

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.

### 3.3 Animation – Global transformations

Open the folder **03_ex_Animation**.

Analyze the how the **scene graph** is defined and how **model rotation around the central axis** is accomplished.

Notice the following:

- An **abstract node** has been added to the scene graph that is the parent of the tree model.
- Carrying out transformations for a given scene graph node, applies those transformations to its children nodes.

**Tasks:**

- Add **three trees** to the scene graph.
- Assign **different rotation movements** to them, so that rotate around the central axis with different speeds and directions, but do not collide with each other.
- Implement a **createForest()** function, that creates a forest made up of three trees. It must be possible to **rotate the trees around the central axis of the forest**.
- Create a scene with **four forests**, each **rotating around its central axis** and all of them **rotating around the central axis of the scene**.

### 3.4 Interaction – Keyboard

Open the folder **04_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.

### 3.5 Interaction – Camera control using the mouse

Open the folder **05_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.