# Three.js Basic Scene Setup

1. **Canvas**

The canvas is the HTML element where the scene will be rendered.

1. **Scene**

The scene contains everything that is to be rendered. Cameras, objects, lights must be added to the scene before they are rendered (scene.add(x)). The Group class allows objects to be grouped and then added to the scene to allow easier management of complex scenes.

1. **Cameras**

Cameras are the POV of what is to be rendered. The CameraHelper class can be used to get a wireframe of the camera and its FOV.

* PerspectiveCamera: Mimics the way the human eye sees. Most common camera for rendering a 3D scene.
* ArrayCamera: Render a scene with a predefined set of cameras.
* CubeCamera: 360 degrees camera. Creates 6 cameras, one in each direction.
* OrthographicCamera: Constant rendering. Objects sizes stay constant regardless of the distance from the camera. Can be used for a 2D scene.
* StereoCamera: Dual PerspectiveCameras used for effects like Anaglyph 3D or Parallex Barrier. Used for VR.

1. **Renderer**

The renderer will render the scene using the given camera on the canvas. When instantiated a canvas must be given as a parameter. Then .render() is called with the scene and camera that are to be rendered. To allow dynamic scenes (i.e. animations) .render() is called in the tick() function.

1. **Meshes**

Meshes are the objects in a scene. They consist of a geometry and a material.

* MeshStandardMaterial: PBR, metallic-roughness workflow. Most used.
* MeshBasicMaterial: Simple shaded, flat.
* MeshLambertMaterial: Non-shiny, without specular highlights. More performant.
* MeshMatcapMaterial: Defined by a matcap, which encodes the material color and shading. Doesn’t receive light or shadow as that’s baked into the matcap.
* MeshNormalMaterial: Maps the normal vectors to RGB colors.
* MeshPhongMaterial: Non PBR, same as MeshLambertMaterial but can simulate shiny surfaces with specular highlights.
* MeshPhysicalMaterial: Extension of MeshStandardMaterial. Providing more advanced PBR properties. Heavy on performance.
* MeshToonMaterial: Implements toon shading.

1. **Geometries**

Representation of an object, line or point. Includes vertex positions, face indices and other information about the shape of an object. Used inside a Mesh. Some available and common used geometries are: Box, Plane, Sphere, Torus, Cone, Circle.

1. **Materials**

Materials describe the appearance of objects (meshes). They can consist of textures and contain attributes that control the way light hits the object.

1. **Textures**

Is applied to a material to display a texture on the surface of an object. Textures are loaded using TextureLoader. Are very heavy on performance. Optimization such as compression to .webp is important.

1. **Lights**

* AmbientLight: globally illuminates all object in the scene equally.
* DirectionalLight: Emitted in a specific direction. Behaves as if the light is infinitely far away and the rays produced from it are all parallel.
* HemisphereLight: Takes 2 colors. Top of objects will be illuminated with the first color and as less lights hit the objects faces the color fades to the second color.
* PointLight: Emitted from a single point in all directions. Acts like a bare lightbulb.
* RectAreaLight: Emits light uniformly across the face of a rectangular plane. Can be used to simulate light sources such as bright windows or strip lighting.
* SpotLight: Emitted from a single point in one direction, along a cone that increases in size the further from the light it gets.

1. **Shadows**

Enabled using “renderer.shadowMap.enabled = true” and “renderer.shadowMap.type = THREE.PCFSoftShadowMap”. Objects to receive and cast shadows must be defined using the castShadow and receiveShadow properties.

To increase performance, mapping must be optimized. To calculate shadows lights act as a camera. Access the camera of a light with light.shadow.camera and optimize its properties. (use a CameraHelper)

1. **Animate**

Animations can be implemented using the Timer class with a tick function. The tick function is recalled after each frame, allowing objects to be updated and then rerendered. Inside the tick function timer.getElapsed() can be used to make sure animations are uniform for all displays with different fps rates.

1. **Resizing**

Width and height of the window are defined in a sizes variable. Using an event listener on the resize property, we update the sizes properties, the camera aspect ratio and renderer size and pixel ratio.

1. **Debug**

GUI is a debug library that allows easy changing of selected properties live in the scene. Folders can be made to organize different properties.