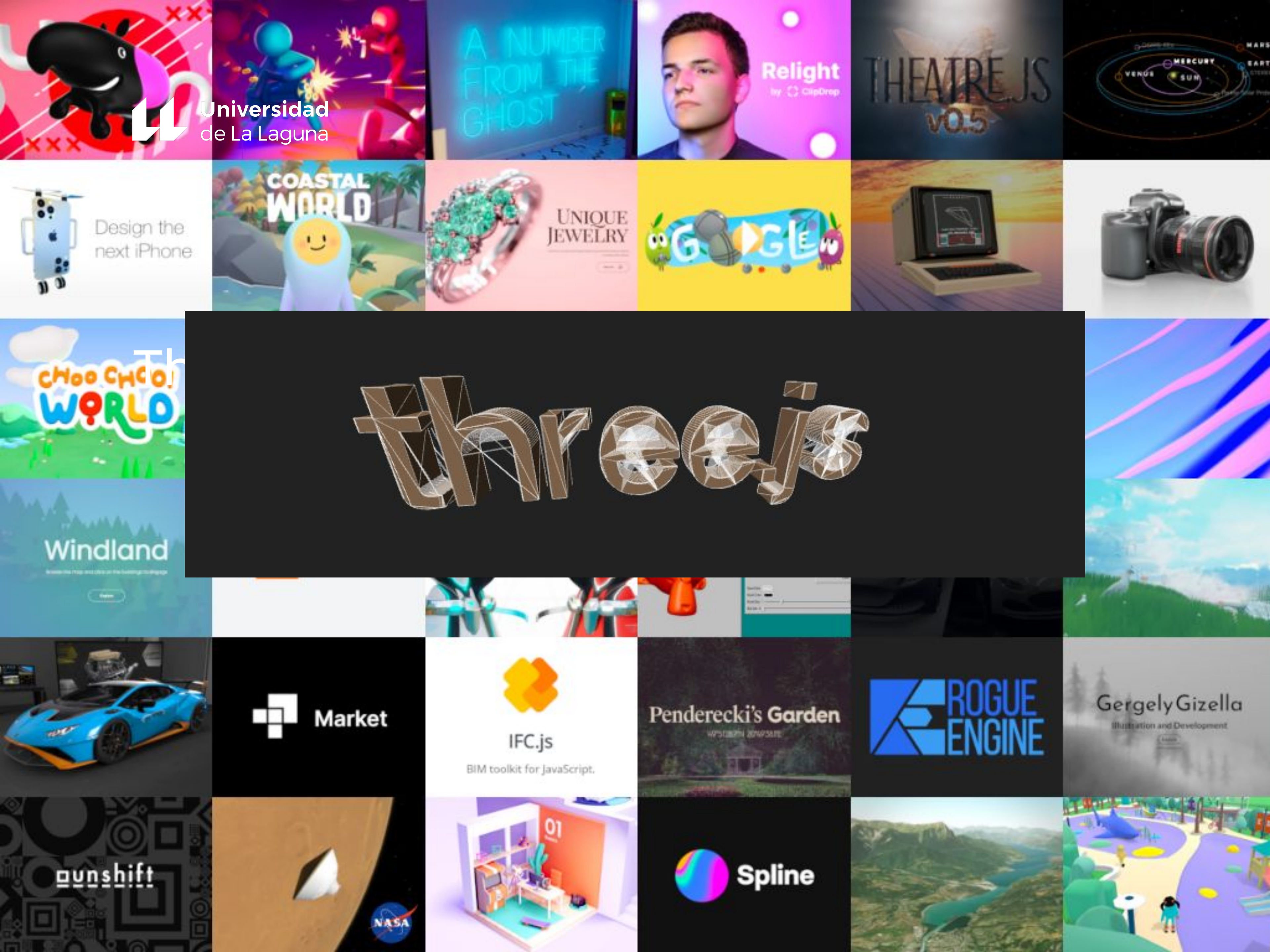


Universidad
de La Laguna
A NUMBER FROM THE GHOST
Relight
THEATRE.JS
v0.5
MERCURY
VENUS
SUN
Design the next iPhone
COASTAL WORLD
UNIQUE JEWELRY
CHOO CHOO! WORLD
threejs
Windland
Market
IFC.js
BIM toolkit for JavaScript.
Penderecki's Garden
ROGUE ENGINE
Gergely Gizella
ounshift
NASA
01
Spline

About us

Alejandro Cruz Quiralte
alejandro.cruz.30@ull.edu.es

Óscar García González
alu0101477794@ull.edu.es

Result Example

Canon Example

Index

- Introduction
- Primitives
- Scene graph
- Materials
- Textures
- Lights
- Cameras
- Set up
- Code examples

Introduction

Introduction

Three.js is a 3D library built over **WebGL** that tries to make it as easy as possible to get 3D content on a webpage

- Three.js abstracts the complexities of WebGL
- Documentation
- Quick prototyping

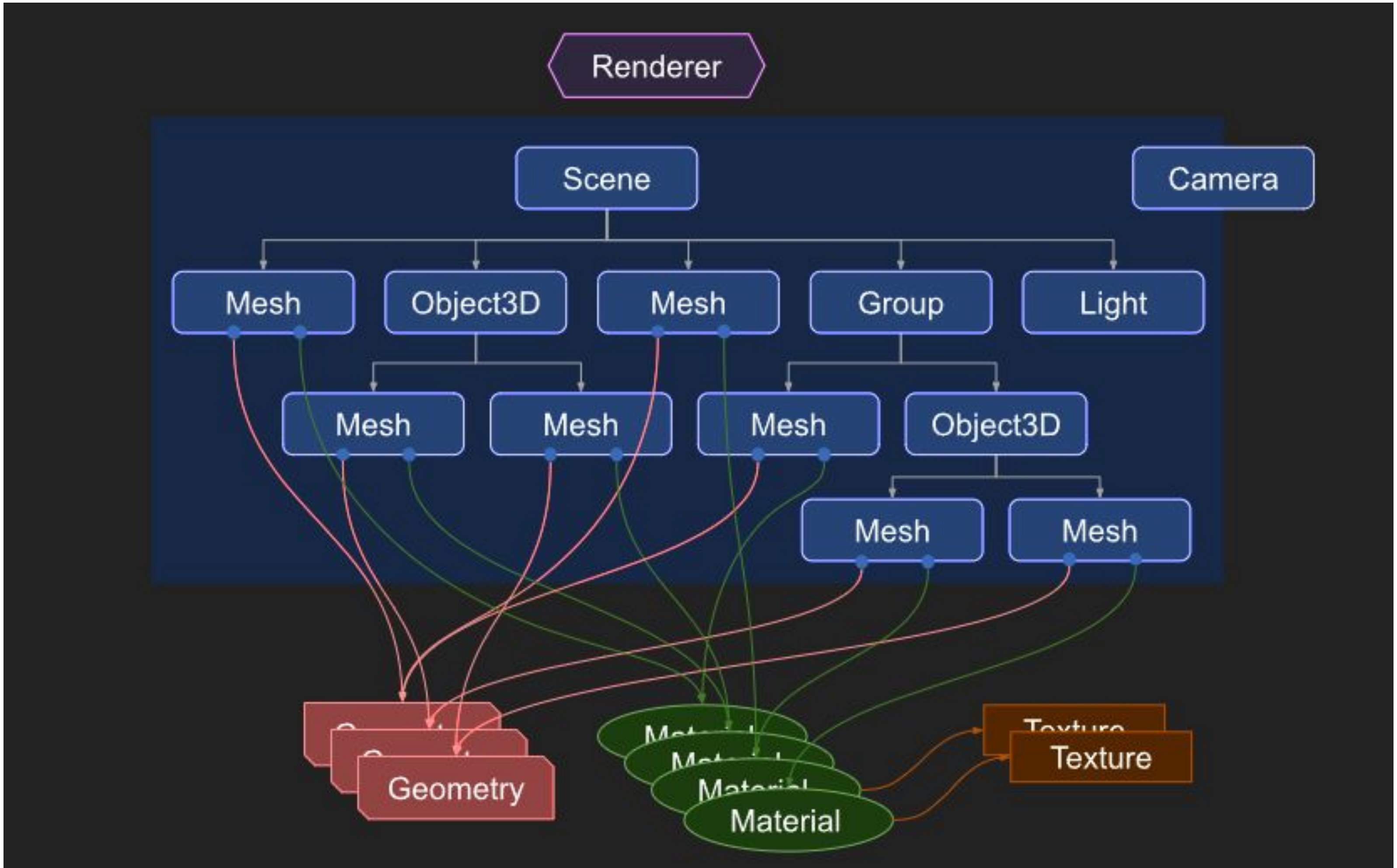
Introduction

- Overhead compared to WebGL
- Worse Low-Level Control
- Dependencies on external libraries

Introduction

Let's try to give you an idea of the structure of a three.js app

A three.js app requires you to create a series of objects and connect them together



Introduction

Renderer

Main object of three.js.
You pass a **Scene** and a **Camera** to a **Renderer** and it renders (draws) the portion of the 3D scene that is inside the field of view of the camera as a 2D image to a canvas

Scenegraph

A **Scene** object defines the root of the **scenegraph** and contains properties like the background color and fog.

These objects define a hierarchical parent/child tree like structure and represent where objects appear and how they are oriented.

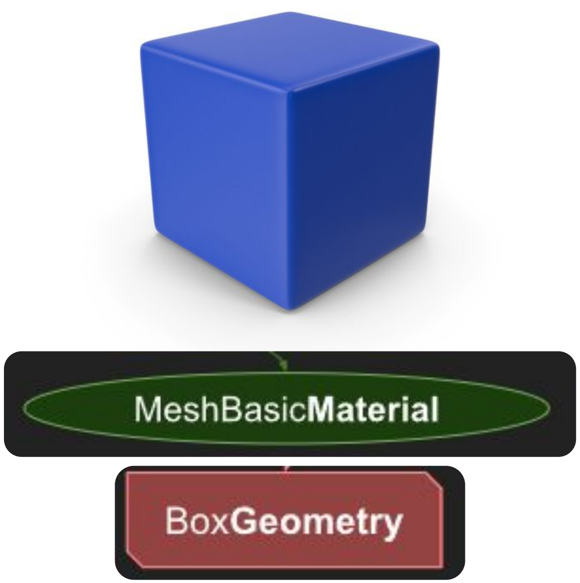
Children are positioned and oriented relative to their parent

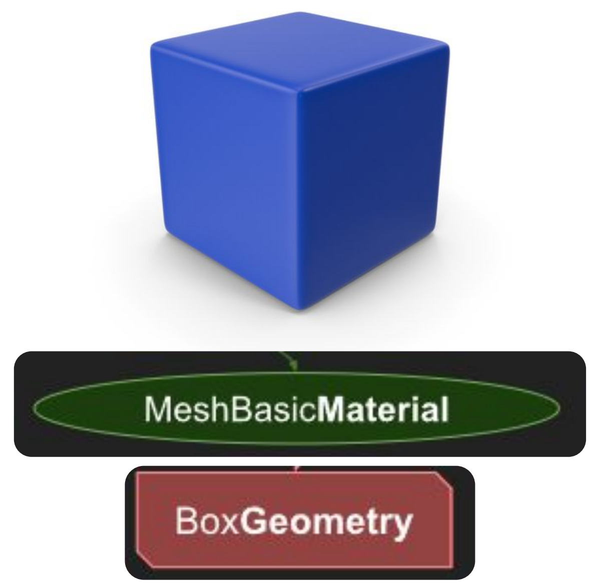
Introduction

Mesh

Mesh objects represent drawing a specific **Geometry** with a specific **Material**.

Both **Material** objects and **Geometry** objects can be used by multiple **Mesh** objects





Introduction

Material

Represents the surface properties used to draw geometry including things like the color to use and how shiny it is

A **Material** can reference one or more **Texture** objects

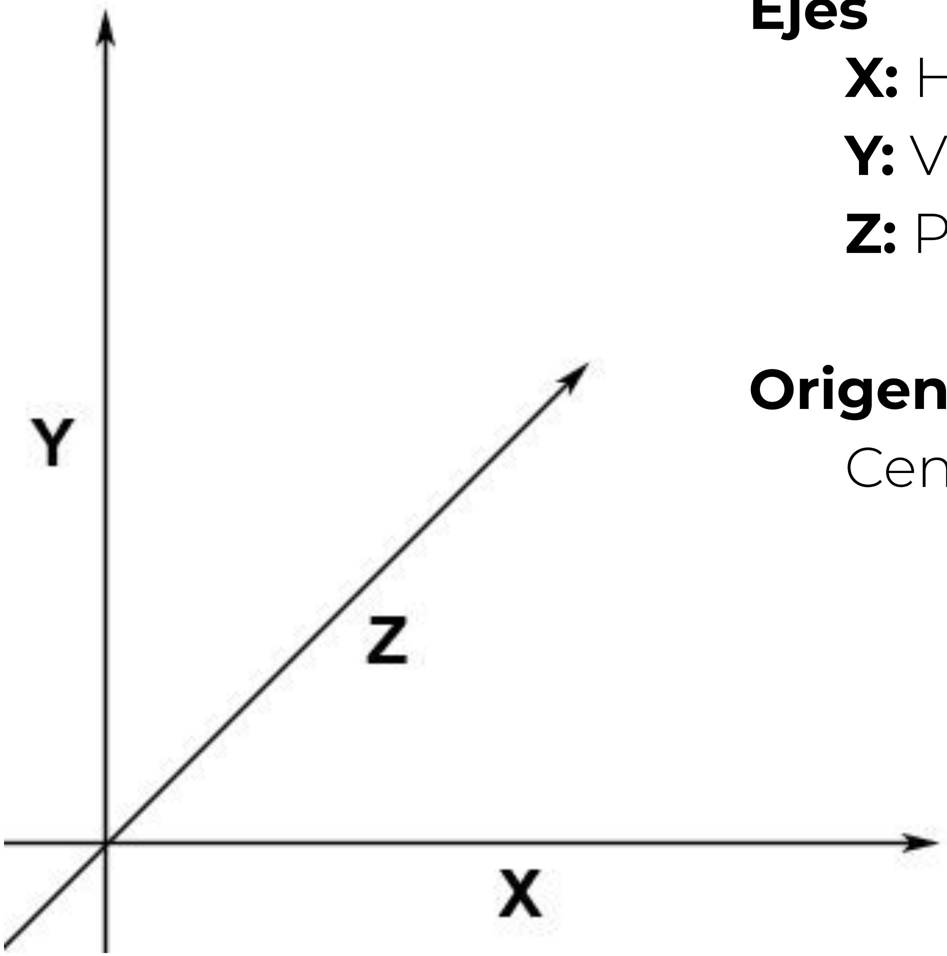
Texture

Texture objects generally represent images either loaded from image files, generated from a canvas or rendered from another **scene**

Light

These objects represent different kinds of lights

Introduction



Coordinate System

Ejes

X: Horizontal, izquierda a derecha.

Y: Vertical, abajo hacia arriba.

Z: Profundidad, adelante hacia atrás.

Origen:

Centro del plano [0,0,0]

Universidad de La Laguna

Program Example

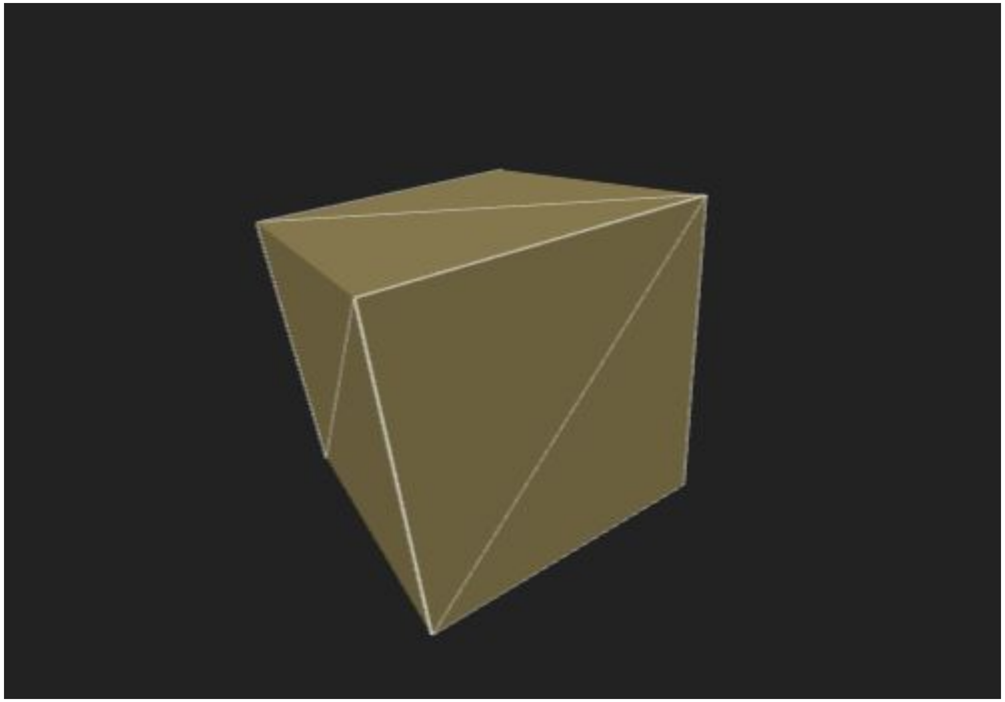
Universidad de La Laguna

Primitives

Primitives

Box

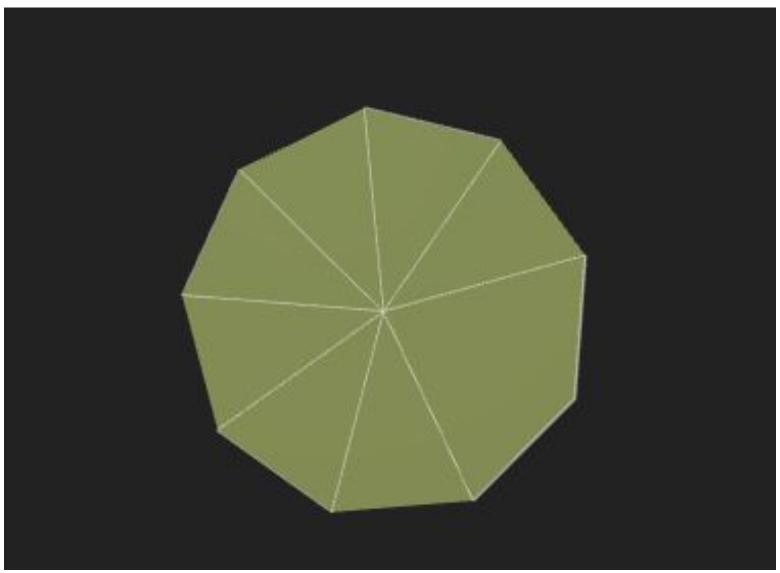
```
const geometry = new THREE.BoxGeometry(width, height, depth)
```

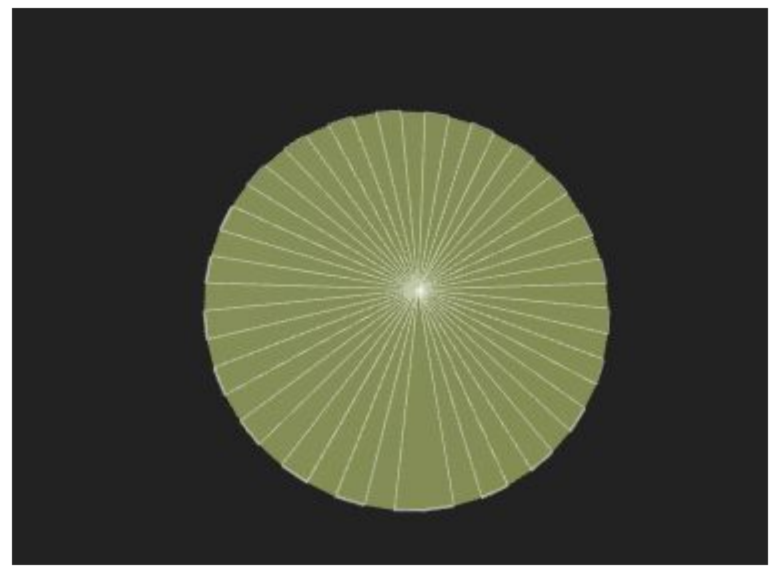


Primitives

Circle

```
const geometry = new THREE.CircleGeometry(radius, segments);
```

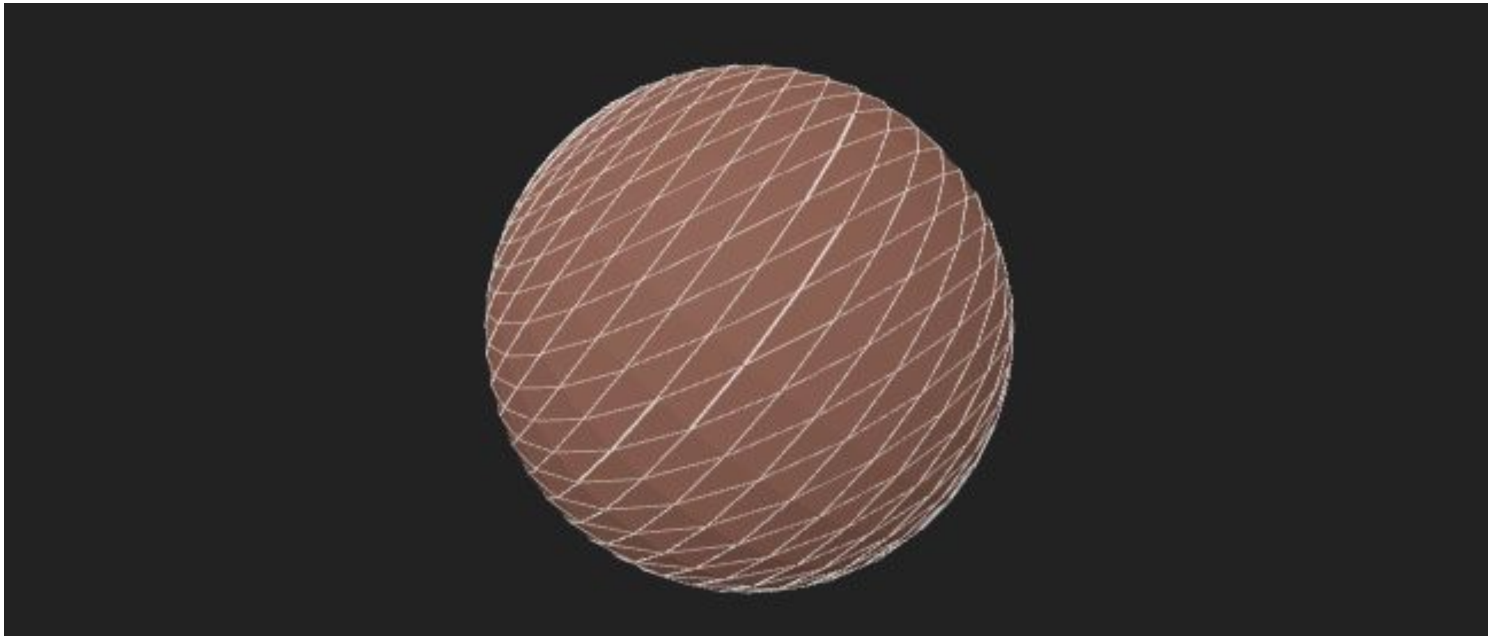




Primitives

Sphere

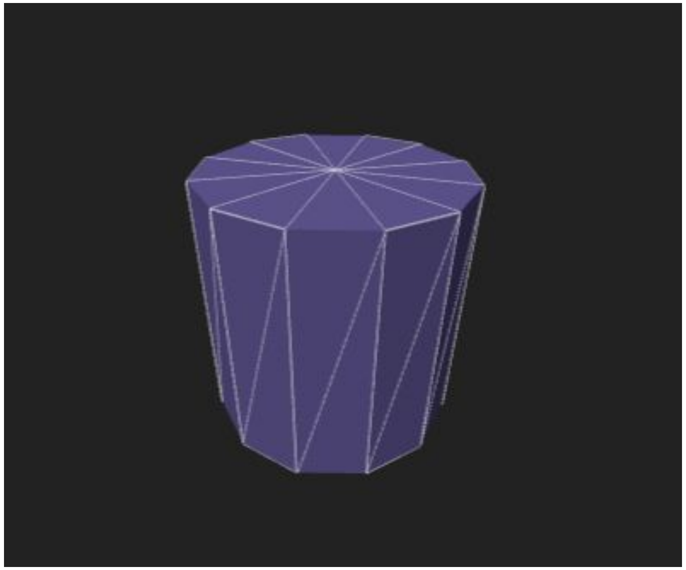
```
const geometry = new THREE.SphereGeometry(radius, widthSegments, heightSegments);
```

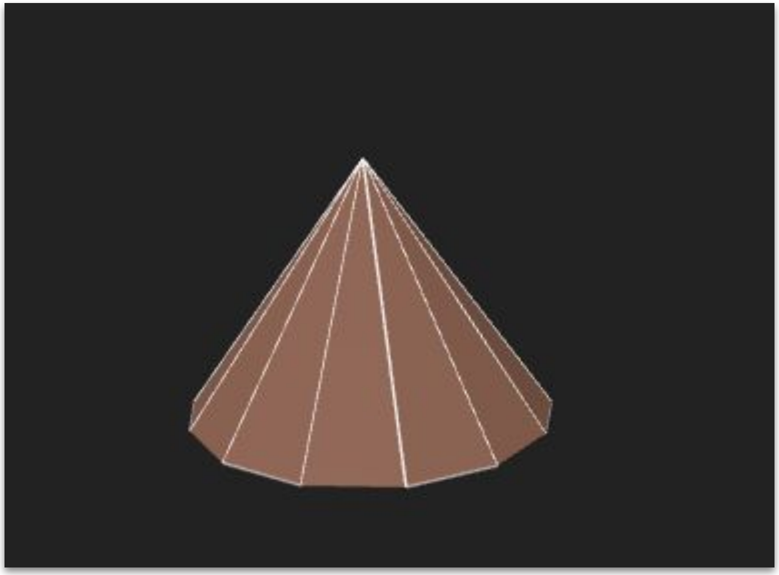


Primitives

Cylinder

```
const geometry = new THREE.CylinderGeometry(
  radiusTop, radiusBottom, height, radialSegments);
```

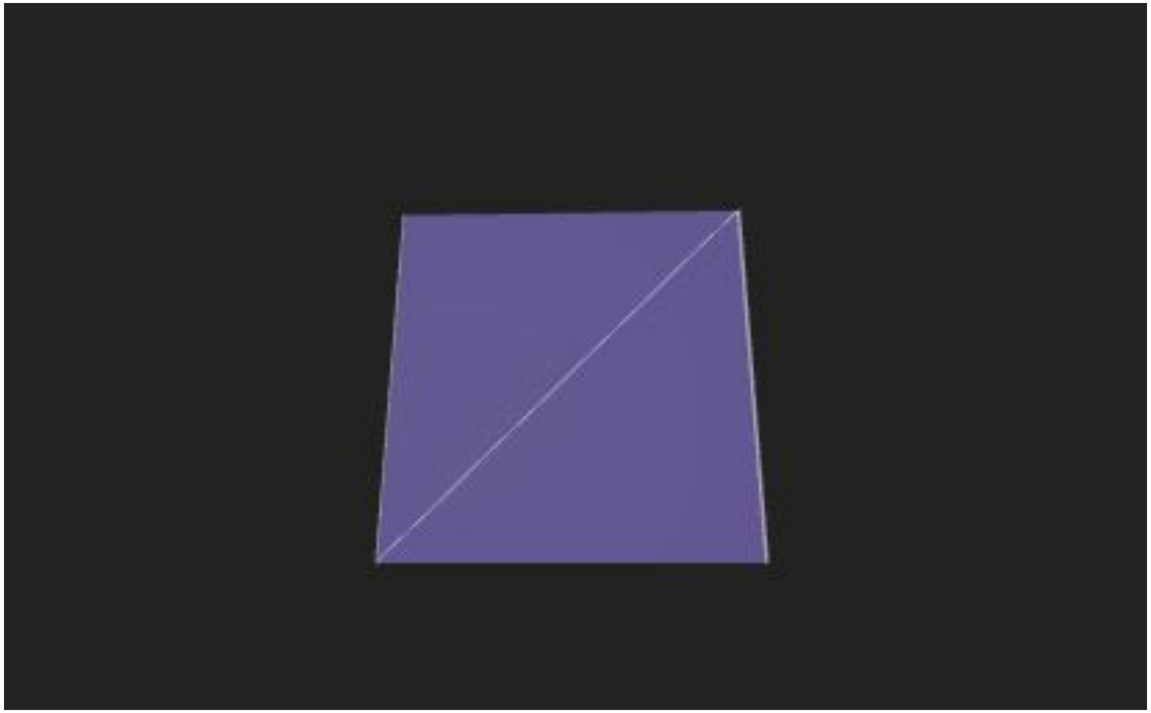




Primitives

Plane

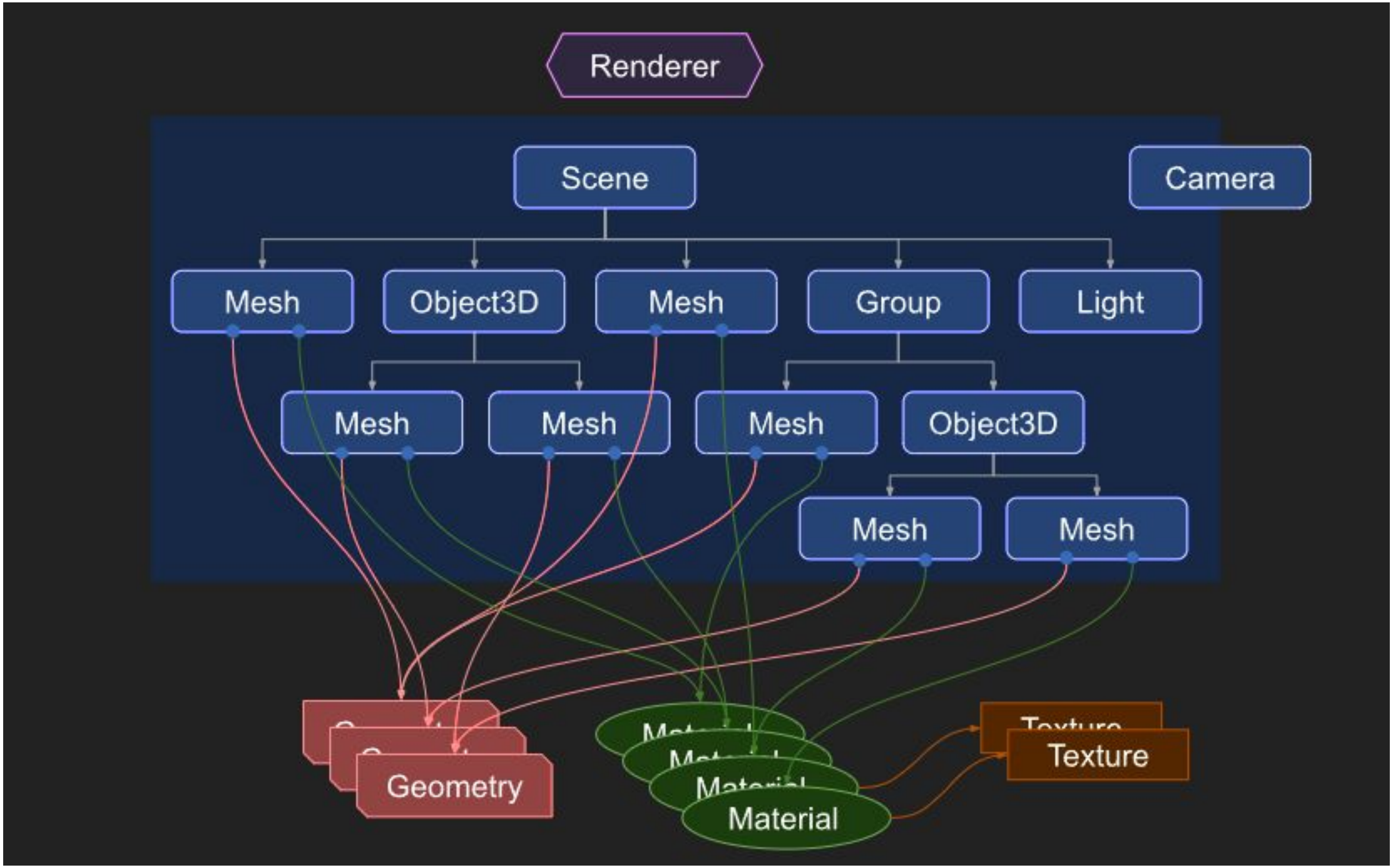
```
const geometry = new THREE.PlaneGeometry(width, height);
```



Universidad de La Laguna

Scenegraph

Scenegraph



Renderer
Scene
Camera
Mesh
Object3D
Mesh
Group
Light
Mesh
Mesh
Mesh
Object3D
Mesh
Mesh
Geometry
Material
Texture
Texture

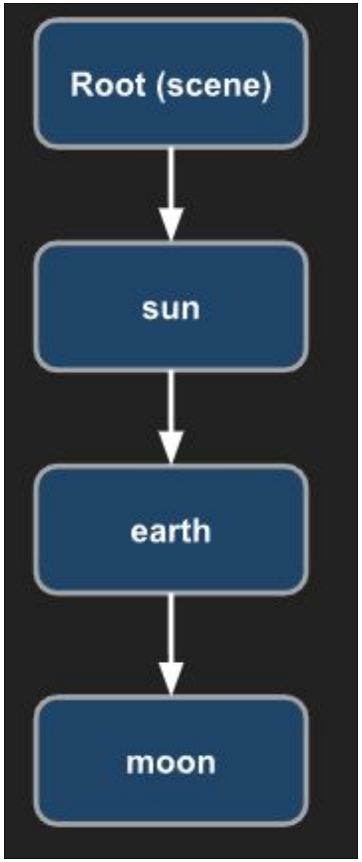
Scenegraph

Visual example: Solar system

We could think that this would be a good scene graph

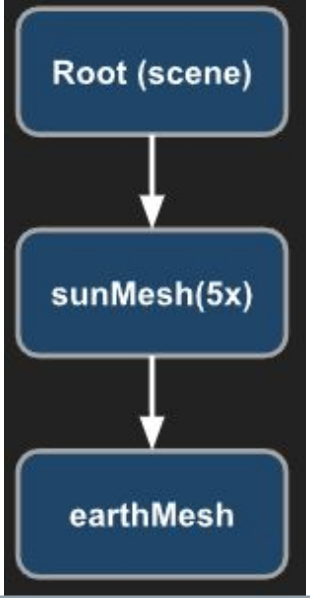
It relates sun, earth and moon

But it isn't right



Scenegraph

Visual example: Solar system



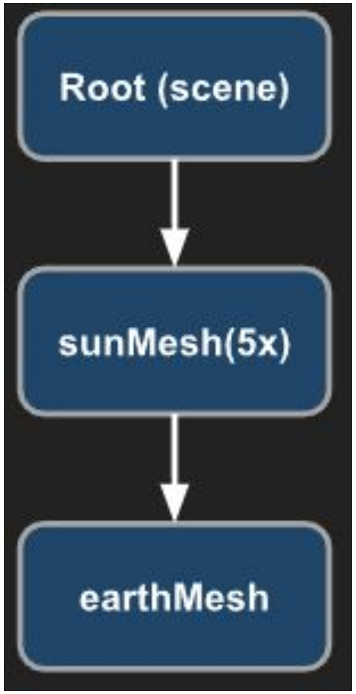

```javascript
const scene = new THREE.Scene();

const sunMesh = new THREE.Mesh(sphereGeometry, sunMaterial);
sunMesh.scale.set(5, 5, 5);
scene.add(sunMesh);

const earthMesh = new THREE.Mesh(sphereGeometry, earthMaterial);
earthMesh.position.x = 10;
sunMesh.add(earthMesh);
```

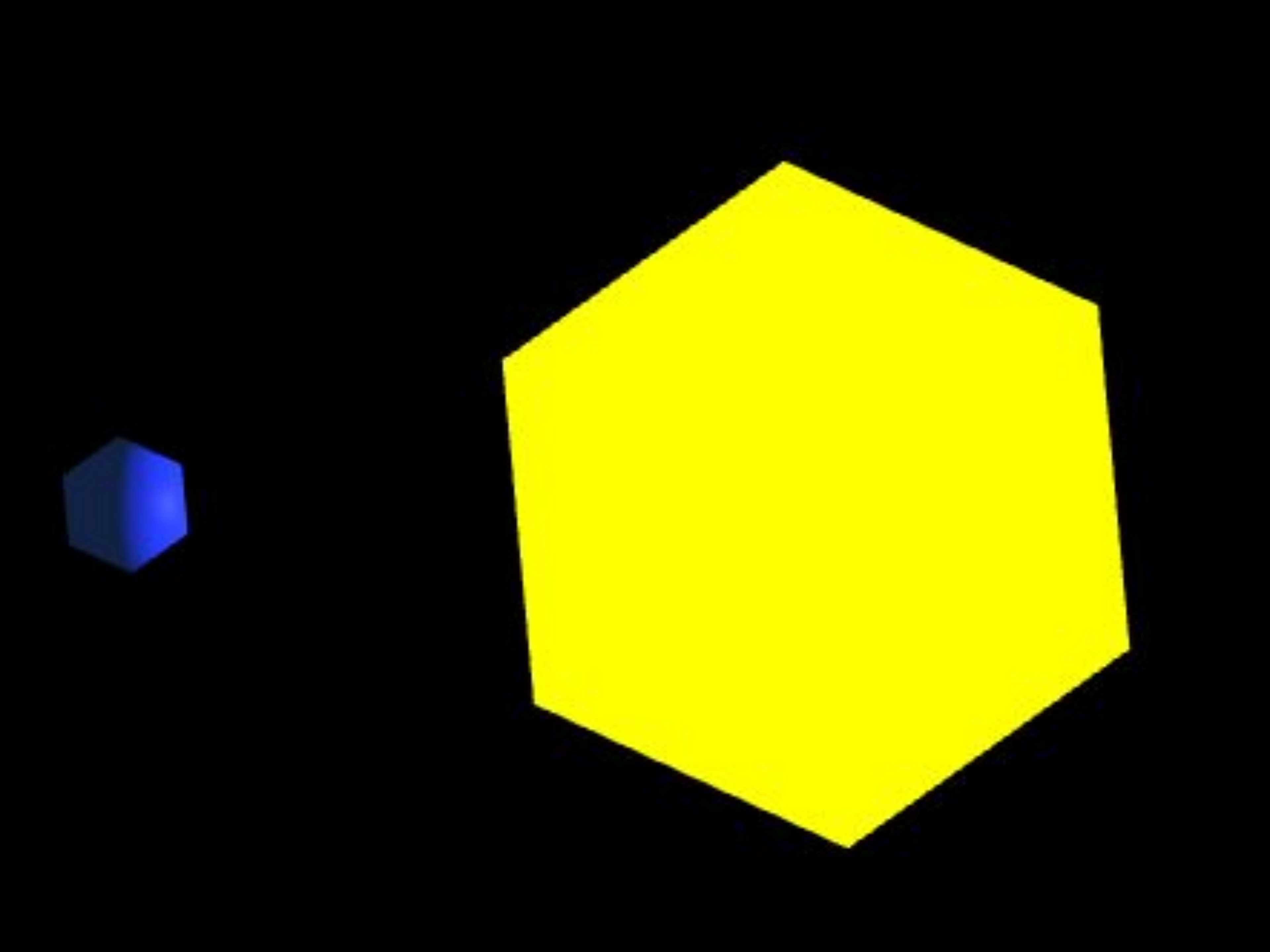
Scenegraph

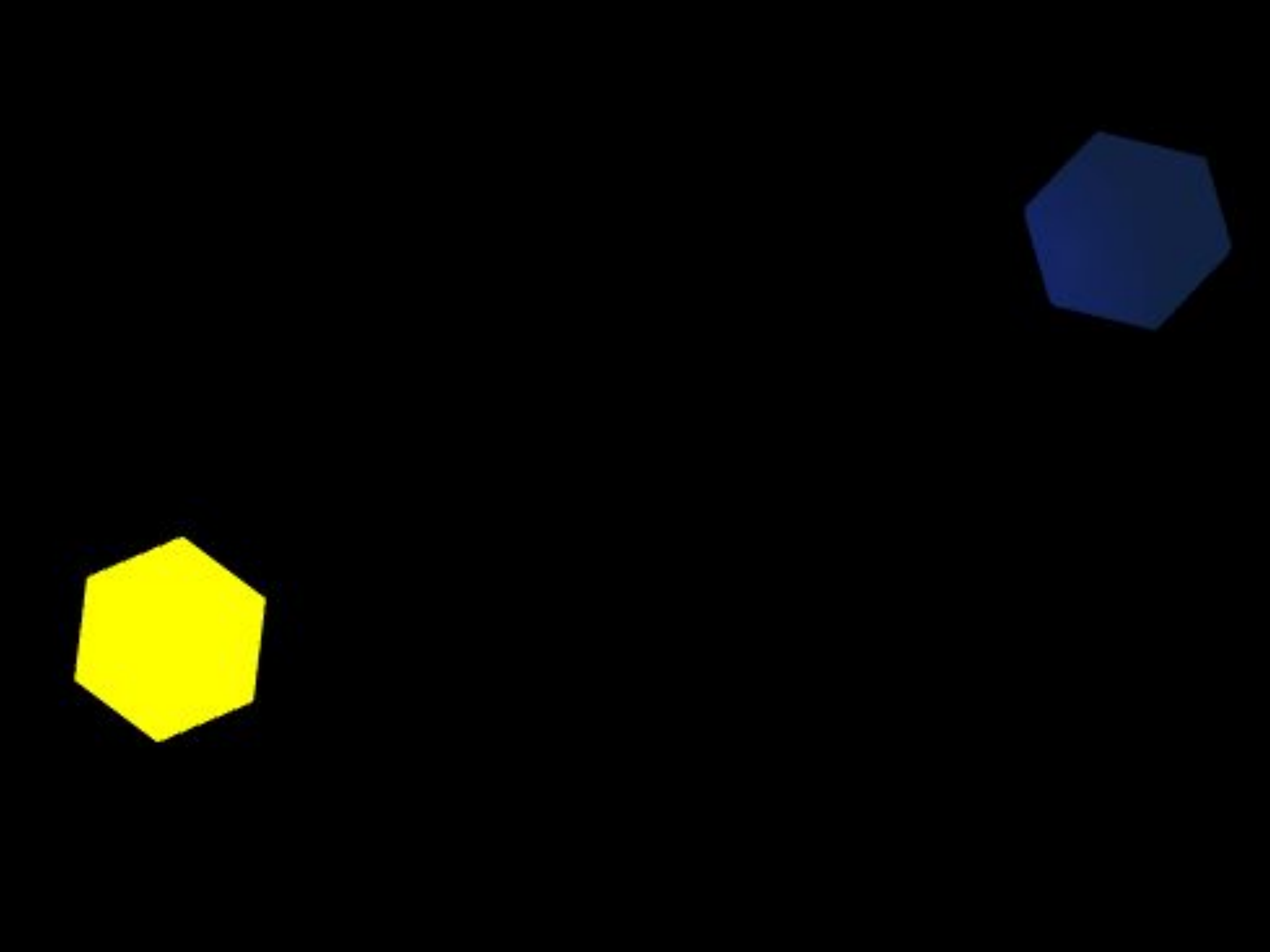
Visual example: Solar system

Following the previous scene graph, if we apply any modification to the **sunMesh** it will also apply to its childs

In this case, we scaled the **sunMesh** to 5x, but **earthMesh** will also have this scale applied





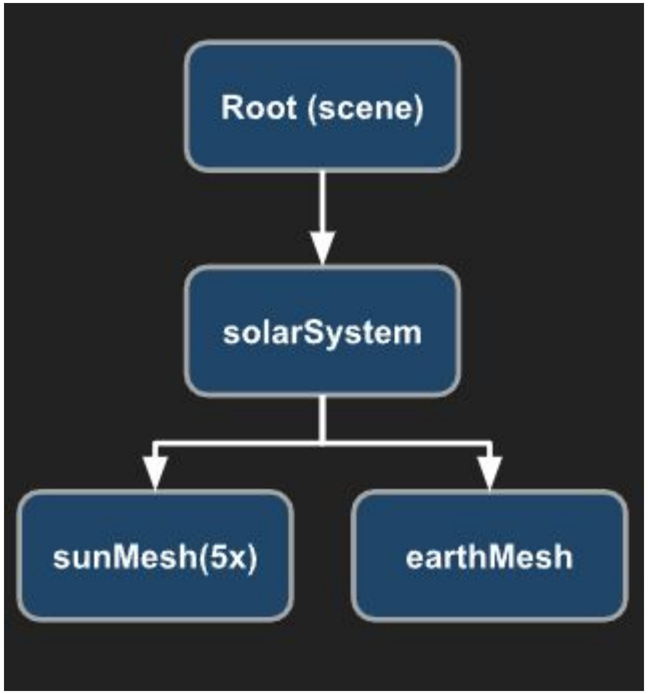


Scenegraph

Visual example: Solar system

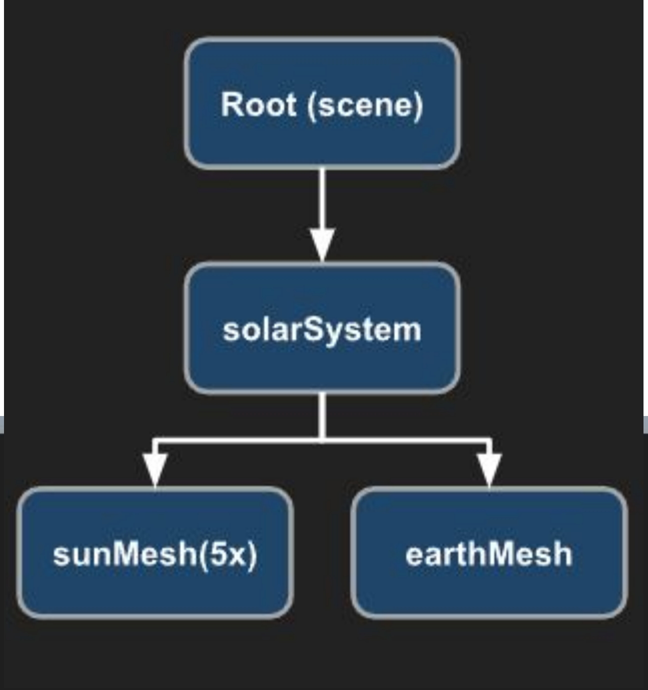
Creating an empty parent node and parenting both elements to it solves the situation

Now, any modification to **sunMesh** only affects to this node and all childs of it while **earhMesh** stays intact

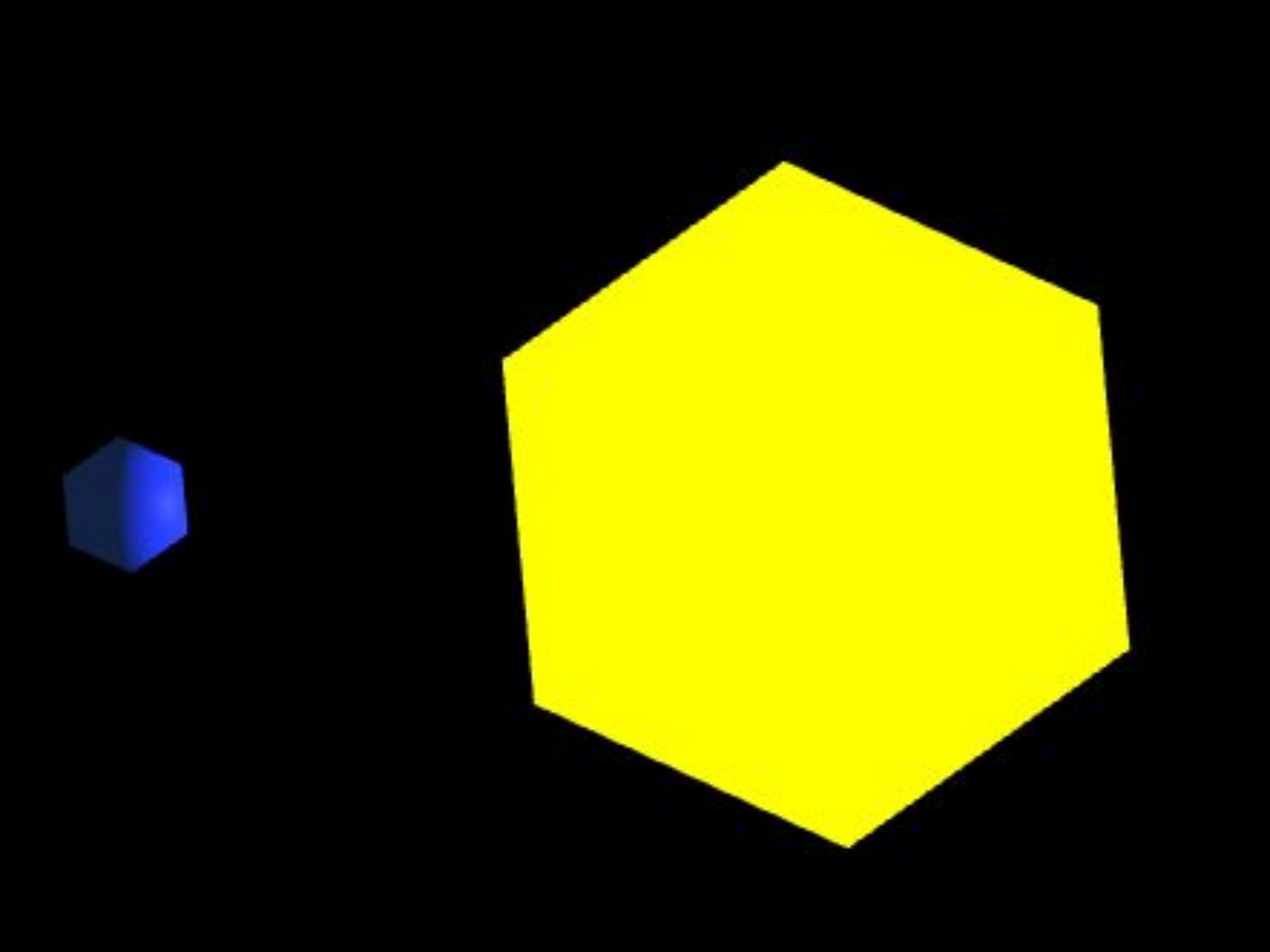


Scenegraph

Visual example: Solar system




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

const solarSystem = new THREE.Object3D();
scene.add( solarSystem );

const sunMesh = new THREE.Mesh( sphereGeometry, sunMaterial );
sunMesh.scale.set( 5, 5, 5 );
solarSystem.add( sunMesh );

const earthMesh = new THREE.Mesh( sphereGeometry, earthMaterial );
earthMesh.position.x = 10;
solarSystem.add( earthMesh );
```



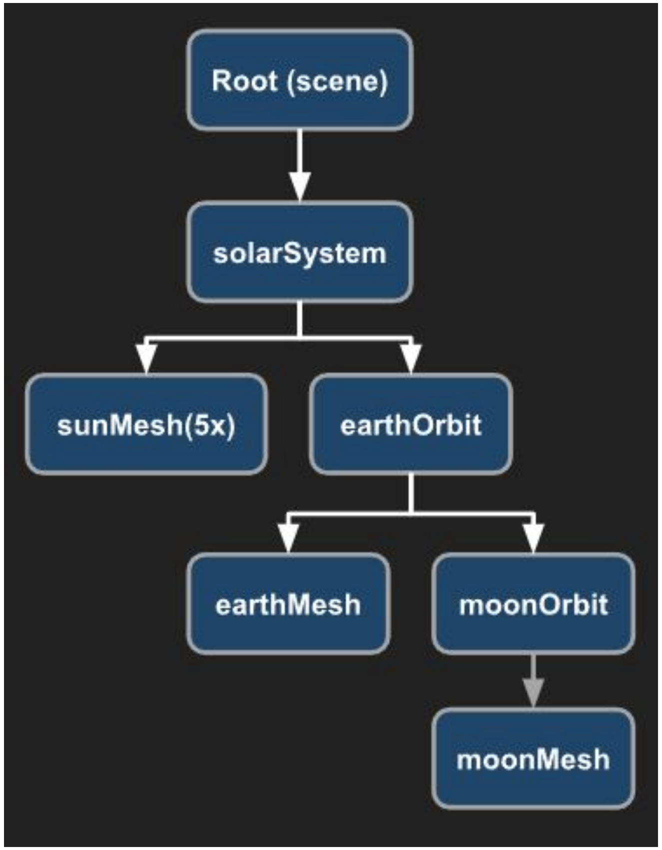
Scenegraph

Visual example: Solar system

Following this logic, our final scene graph looks like this

Sun, earth, and moon meshes are now independent of each other but they are related by their orbits

By doing it this way we achieve our goal



Materials

Materials

Three.js provides several types of materials

They define how objects will appear in the scene

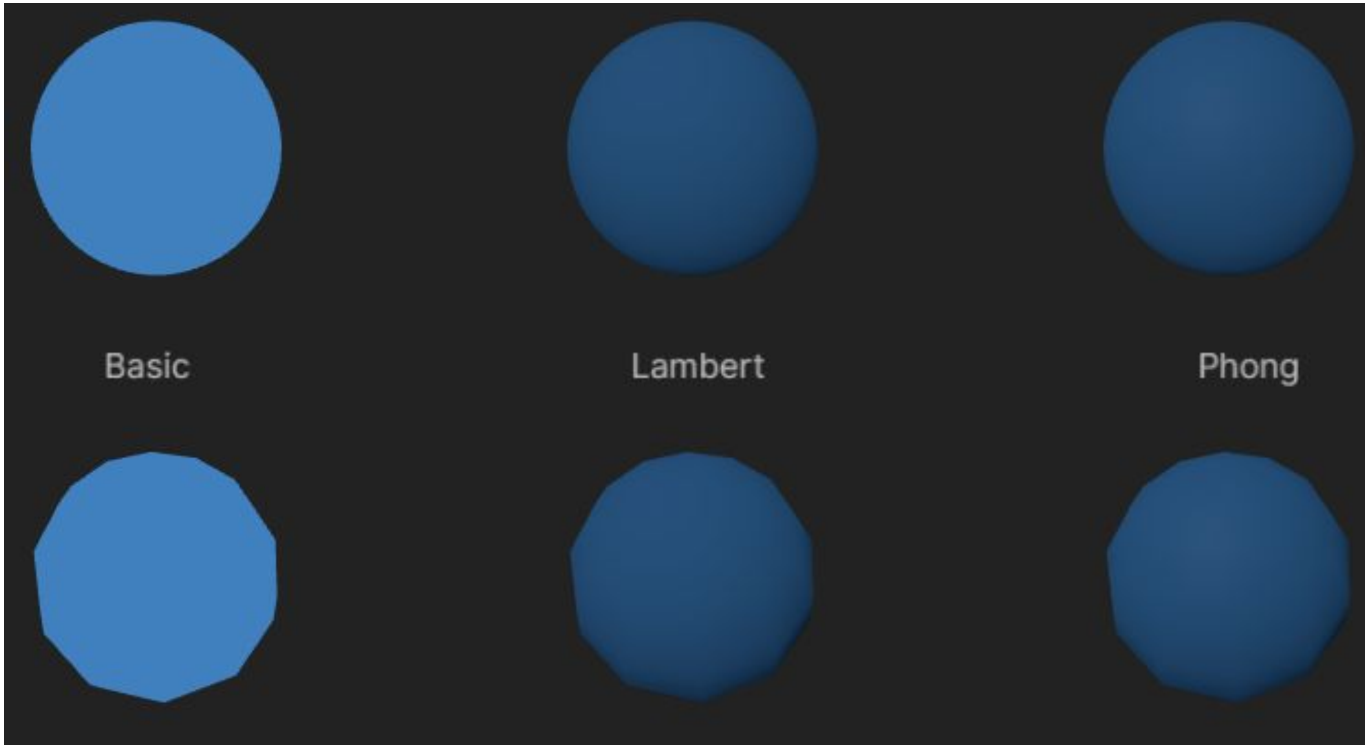
```
const material = new THREE.MeshPhongMaterial({
  color: 'blue'
});
```

Materials

The **MeshBasicMaterial** is not affected by lights

The **MeshLambertMaterial** computes lighting only at the vertices

The **MeshPhongMaterial** computes lighting at every pixel



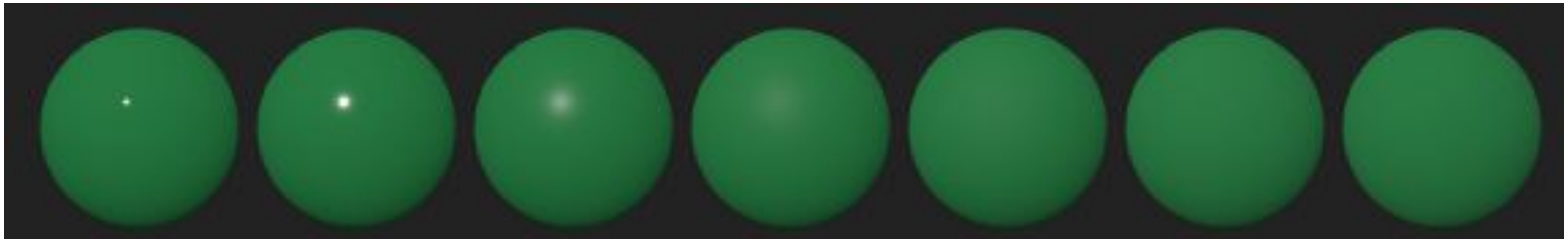
Materials

Some Materials properties:

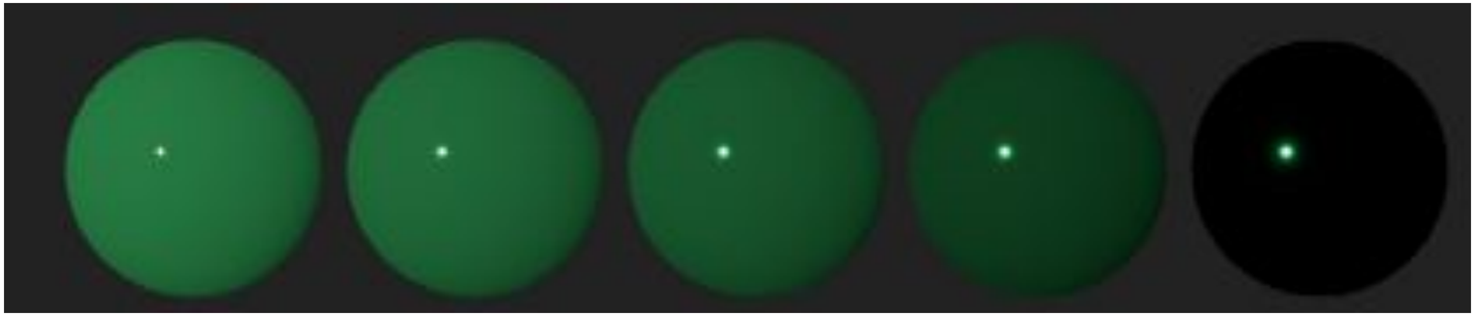
Shininess

Materials

Roughness



Metalness



Universidad de La Laguna

Textures

Textures

All we need to do is create a **TextureLoader**

Then we call its **load** method with the URL or path of an image

And set the material's **map** property to the result instead of setting its color

```
// Load texture
const loader = new THREE.TextureLoader();
const texture = loader.load('./images/flower_example.jpg');

// Create material and apply texture
const material = new THREE.MeshBasicMaterial({
  map: texture
});
```

Textures

Not all geometry types supports multiple materials

BoxGeometry can use 6 materials one for each face

ConeGeometry can use 2 materials

CylinderGeometry can use 3 materials

For other cases you will need to build or load a **custom geometry** and/or modify texture coordinates

Textures

Using the method previously shown our texture will be transparent until the image is loaded by three.js

This has the big advantage that we don't have to wait for the texture to load and our page will start rendering immediately

But not always this operating is intended

Textures

This way we wait for the texture to **load**

```
// Load texture waiting for it to load before creating the cube
const loader = new THREE.TextureLoader();
loader.load('./images/flower_example.jpg', (texture) => {
  const material = new THREE.MeshBasicMaterial({ map: texture });
  const cube = new THREE.Mesh(geometry, material);
  scene.add(cube);
});
```

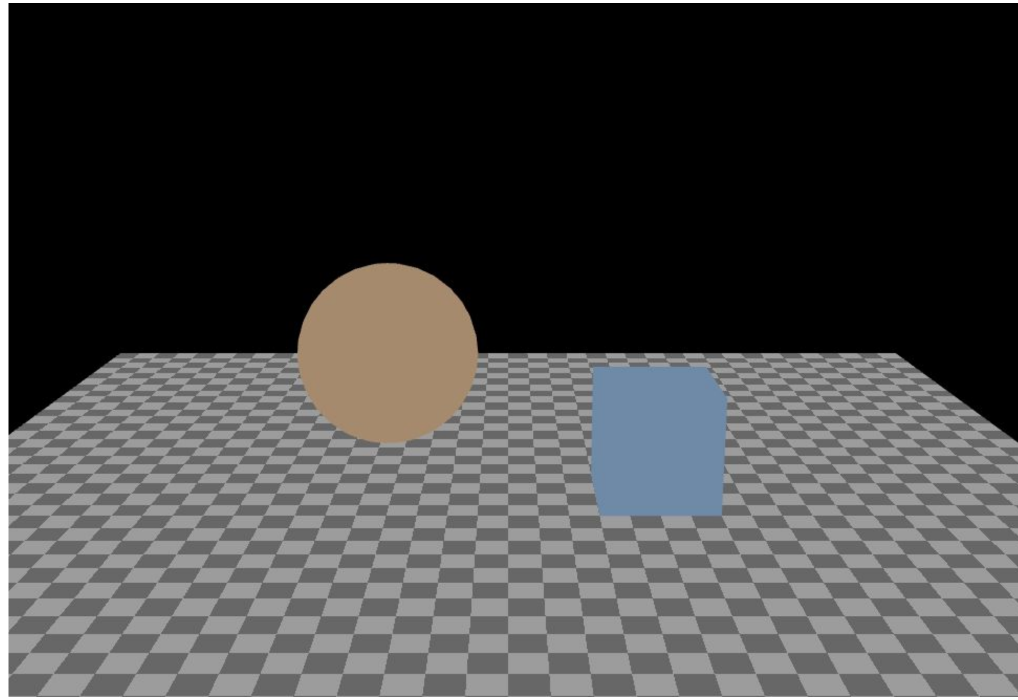
Universidad de La Laguna

Lights

Lights

Types of lights:

Ambient Light

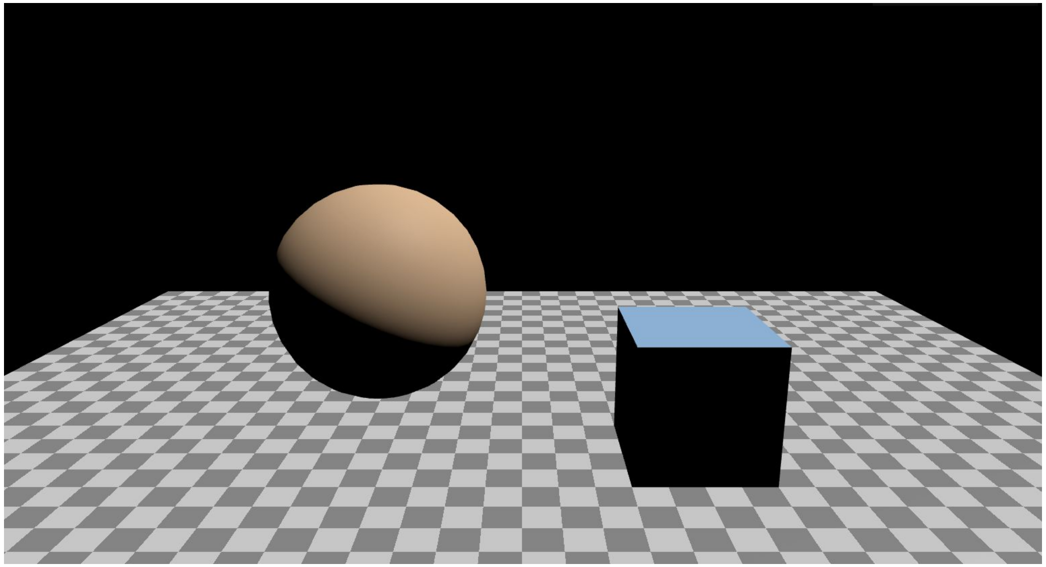


```
const color = 'white';
const intensity = 1;
const light = new THREE.AmbientLight(color, intensity);
scene.add(light);
```

Lights

Types of lights:

Directional Light

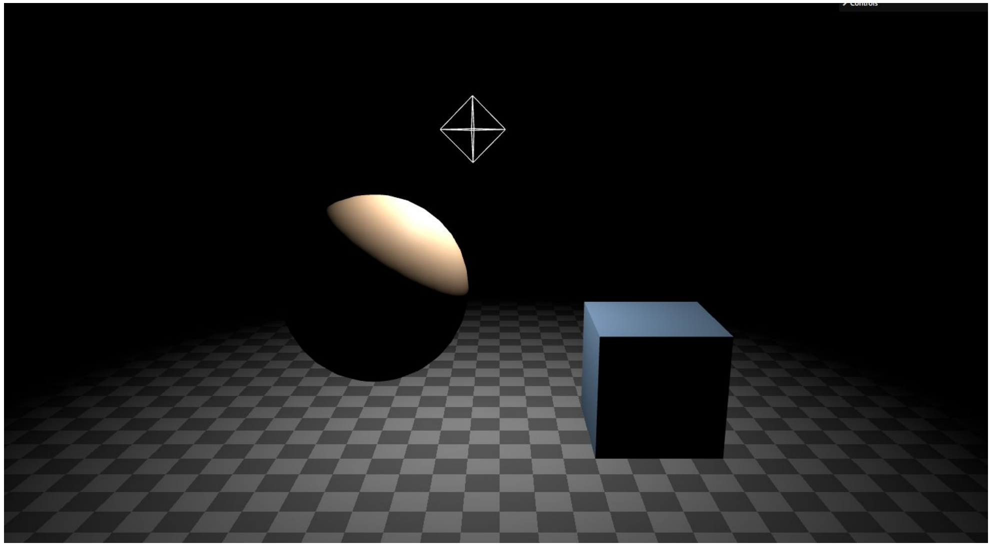


```
const color = 'white';
const intensity = 150;
const light = new THREE.DirectionalLight(color, intensity);
light.position.set(0, 10, 0);
light.target.position.set(-5, 0, 0);
scene.add(light);
scene.add(light.target);
```

Lights

Types of lights:

Point Light

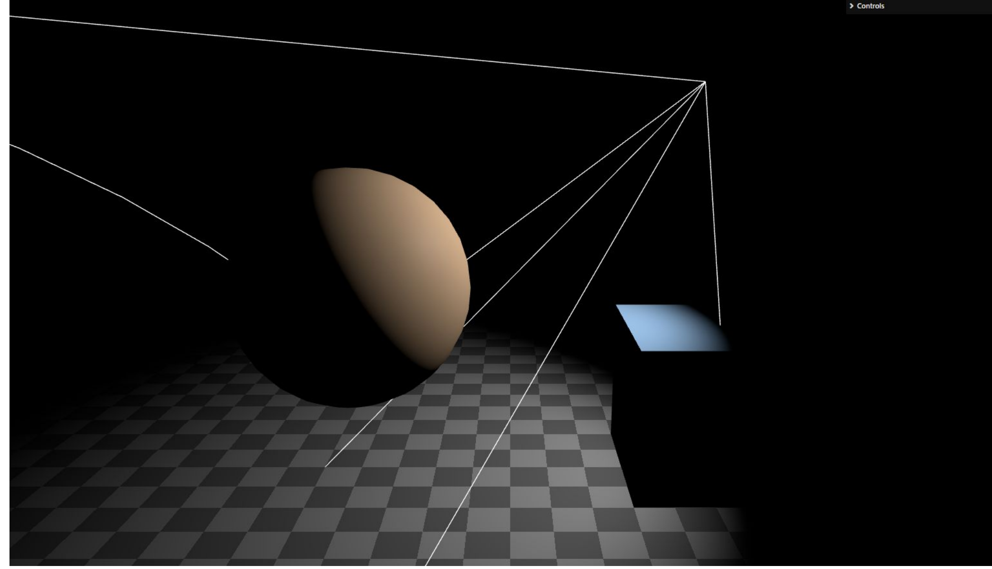


```
const color = 'white';
const intensity = 150;
const light = new THREE.PointLight(color, intensity);
light.position.set(0, 10, 0);
scene.add(light);
```

Lights

Types of lights:

Spot Light

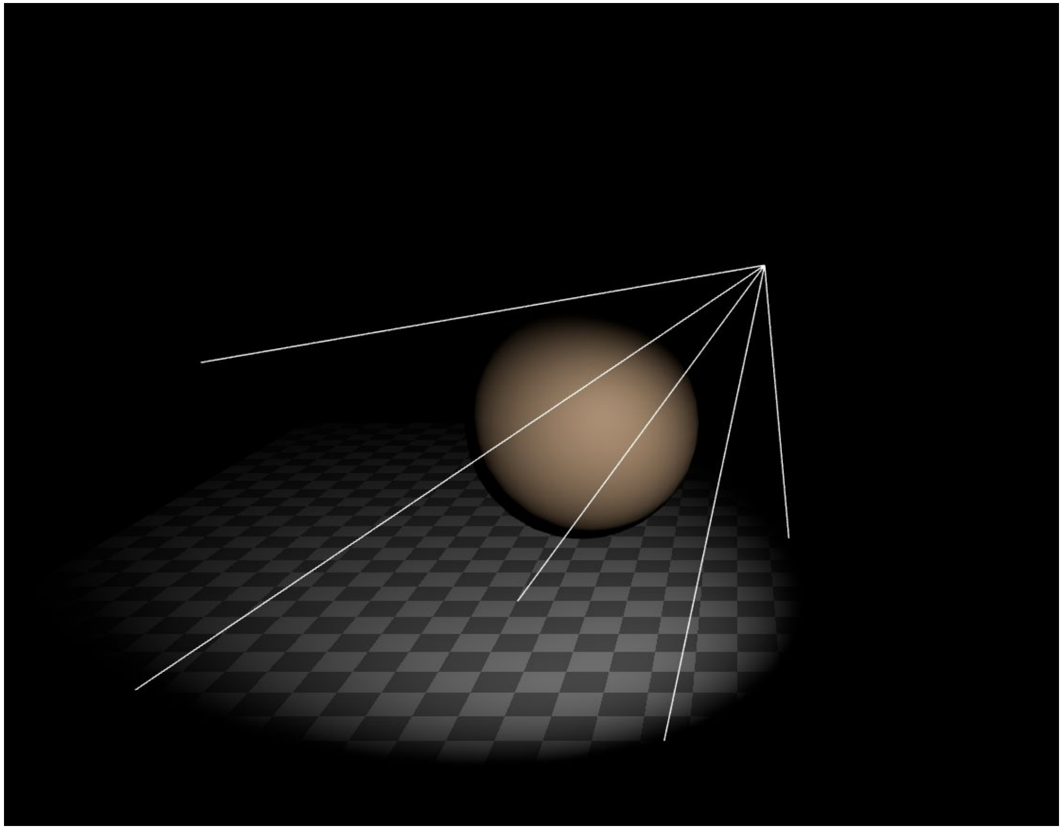


```
const color = 'white';
const intensity = 150;
const light = new THREE.SpotLight(color, intensity);
light.position.set(0, 10, 0);
light.target.position.set(-5, 0, 0);
scene.add(light);
scene.add(light.target);
```

Lights

Types of lights:

Spot Light- helpers



```
const helper = new THREE.SpotLightHelper(light);
scene.add(helper);
```

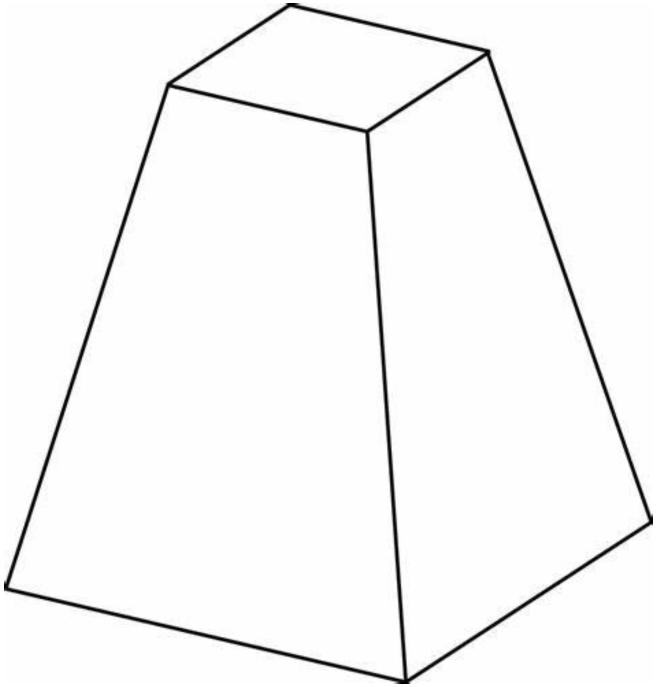
Universidad de La Laguna

Cameras

Cameras

The most common camera in three.js is **PerspectiveCamera**. It gives a 3d view where things in the distance appear smaller than things up close

The **PerspectiveCamera** defines a frustum. A frustum is a solid pyramid shape with the tip cut off



Cameras

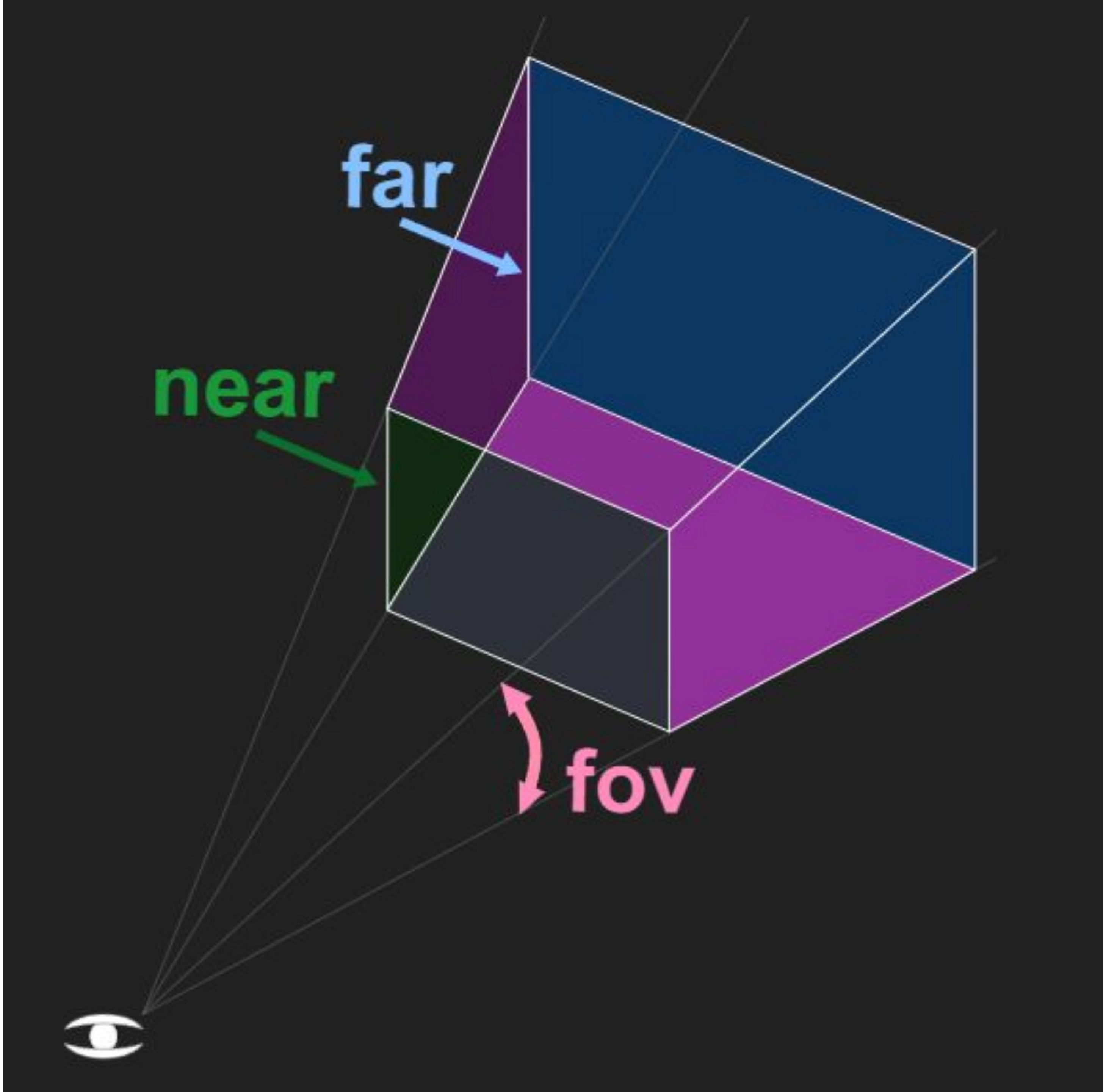
A **PerspectiveCamera** defines its frustum based on 4 properties:

fov, the field of view, defines how tall the front and back of the frustum are by computing the correct height to get the specified field of view

aspect defines how wide the front and back of the frustum are. The width of the frustum is just the height multiplied by the aspect

near defines where the front of the frustum starts

far defines where it ends


far
near
fov

Cameras

The 2nd most common camera is the **OrthographicCamera**

Rather than specify a frustum it specifies a box with the settings **left**, **right**, **top**, **bottom**, **near**, and **far**

Because it's projecting a box there is no perspective

```
const left = -1;
const right = 1;
const top = 1;
const bottom = -1;
const near = 5;
const far = 50;
const camera = new THREE.OrthographicCamera(left, right, top, bottom, near, far);
```

Shadows

Shadows

There are 3 lights which can cast shadows :
DirectionalLight, the **PointLight**, and the **SpotLight**

Activate Shadows options:

```
const renderer = new THREE.WebGLRenderer();
renderer.shadowMap.enabled = true;
```

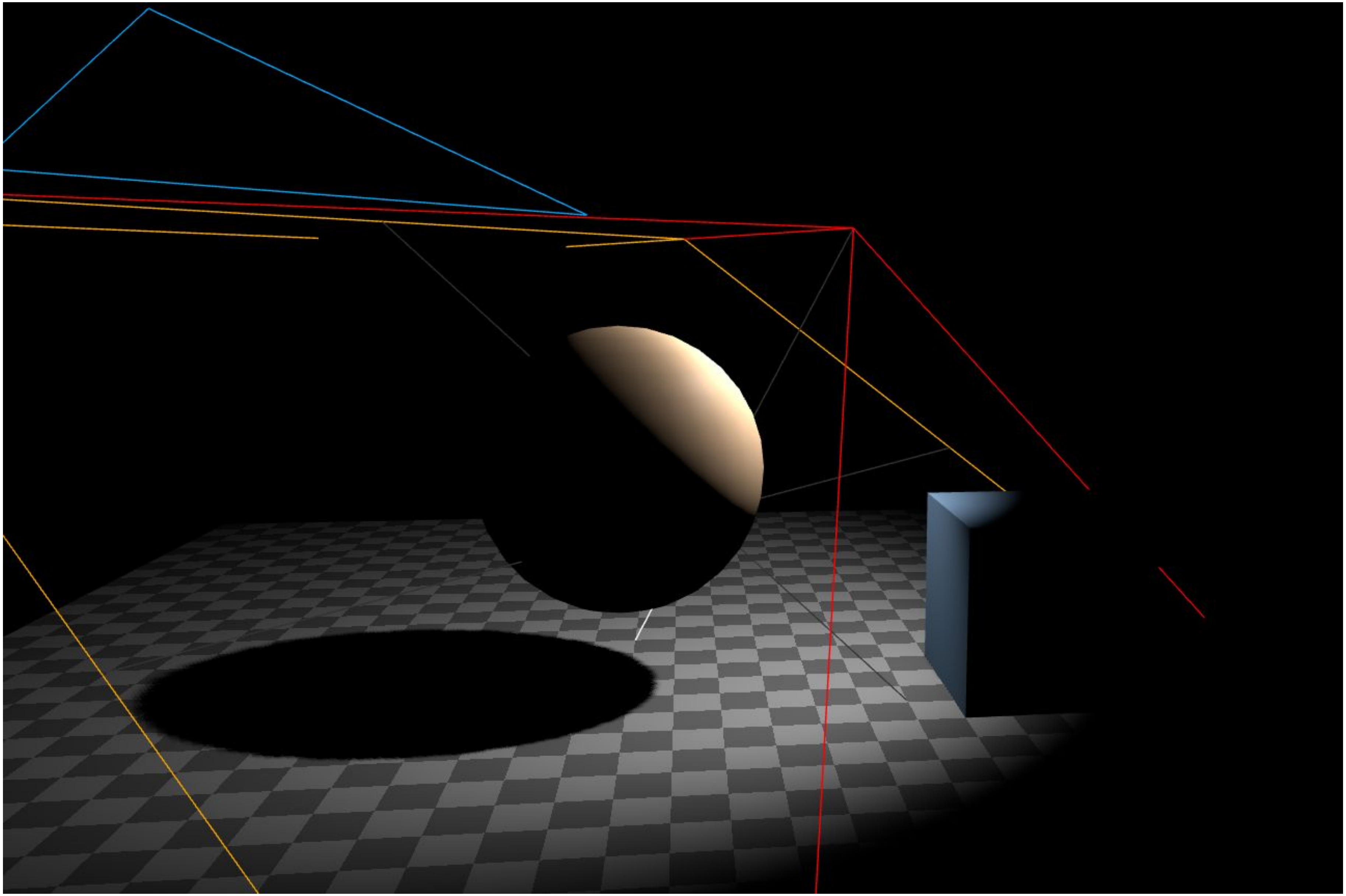
```
const light = new THREE.DirectionalLight(color, intensity);
light.castShadow = true;
```

Shadows

```
const mesh = new THREE.Mesh(planeGeo, planeMat);
mesh.receiveShadow = true;
```

```
const mesh = new THREE.Mesh(sphereGeo, sphereMat);
mesh.receiveShadow = true;
mesh.castShadow = true;
```

Shadows



Setup

Very easy installation using NPM:

```
npm install three
```

```
npm install --save-dev @types/three
```

```
import * as THREE from 'three';
```

Referencias

Getting started with three.js

Three.js Fundamentals

Intro to WebGL with Three.js

Threejs-cookbook

Universidad
de La Laguna