



# WebGL and Three.js

Web Application

**Suriya Natsupakpong, PhD**

Institute of Field Robotics (FIBO)

King Mongkut's University of Technology Thonburi (KMUTT)

# Introduction

- <https://my-room-in-3d.vercel.app/>
- <https://organic-sphere.vercel.app/>
- <https://experiment-holographic-terrain.vercel.app/>
- <https://infinite-world.vercel.app/>
- <https://prior.co.jp/discover/en>
- <https://madbox.io/>

# What will you learn?

- Three.js <https://threejs.org>
- Basics 3D Graphics
- Modeling with Blender
- Shaders
- Portal Scene
- Three.js in React Application

# What is WebGL?

- JavaScript API
- Renders triangles in a canvas at a remarkable speed
- Compatible with most modern browsers
- Uses the Graphic Processing Unit (GPU)
- 3D model consists of many triangles with position and color which WebGL can handle it.

# Three.js

- A 3D javascript library that enables developers to create 3D experience for the web. It simplifies the process of WebGL library.
- Creator: Ricardo (aka Mr.doob)
- Examples:
  - <https://bruno-simon.com>
  - <https://cornrevolution.resn.global/>
  - <https://richardmattka.com>
  - <https://lusion.co>
  - <https://www.oculus.com/medal-of-honor/>
  - <https://heraclosgame.com>

# Local Server and Build Tools

- There are many build tools available these days and you've probably heard of some of them like Webpack, Vite, Gulp, Parcel, etc.
- Vite is a build tool that will build the final website from web code like HTML/CSS/JS.
- Vite can add plugins in order to handle more features. (GLSL and React)
- Vite also do a bunch of things like optimizations, cache breaking, source mapping, running a local server, etc.
- Vite was created by Evan You, the creator of Vue.js

# Node.js

- Install Node.js from <https://nodejs.org/>
- Show node version in your computer: `nvm ls`
- Show node version online: `nvm list available`
- Install node version: `nvm install xx.xx.x`
- Uninstall node version: `nvm uninstall xx.xx.x`
- Set active node version: `nvm use xx.xx.x`

```
C:\Users\Suriya>nvm use 22.12.0  
Now using node v22.12.0 (64-bit)
```

```
C:\Users\Suriya>nvm ls
```

```
  * 22.12.0 (Currently using 64-bit executable)  
    18.20.0
```

# First Three.js Project

## Create a Node.js Project

- Open the terminal in the folder and run `npm init -y`
- It creates the `package.json` file that will contain the minimal information needed to run a Node.js project.

## Add dependencies

- Install Vite library with  
`npm install vite`  
`npm install three`

## Create index.html file

## Run npm run dev

```
VITE v6.0.6  ready in 267 ms
```

- Local: `http://localhost:5173/`
- Network: use `--host` to expose
- press `h` + enter to show help

## package.json

```
{
  "name": "exercise",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
    "dev": "vite",
    "build": "vite build"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "vite": "^6.0.6"
  }
}
```

## index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>First Three.js Project</title>
  </head>
  <body>
    <h1>Welcome to my Three.js website</h1>
    <canvas class="webgl"></canvas>
    <script type="module" src="./script.js"></script>
  </body>
</html>
```



# First Three.js Project

Add Javascript by create `script.js` file

Need 4 elements

- Scene
- Objects : primitive geometries, import models, particles. Light. etc.
- Camera : point of view used when rendering
- Renderer

`script.js`

```
console.log('Hello World! from JavaScript')
```

`script.js`

```
import * as THREE from 'three'

console.log(THREE)
```


`script.js`

```
import * as THREE from 'three'
// Canvas
const canvas = document.querySelector('canvas.webgl')
// Scene
const scene = new THREE.Scene()
// Object
const geometry = new THREE.BoxGeometry(1,1,1)
const material = new THREE.MeshBasicMaterial({color: 0xff0000})
const mesh = new THREE.Mesh(geometry, material)
scene.add(mesh)
// Sizes
const sizes = {width: 800,height: 600}
// Camera
const camera = new THREE.PerspectiveCamera(75, sizes.width/sizes.height)
camera.position.z = 3
scene.add(camera)
// Renderer
const renderer = new THREE.WebGLRenderer({canvas: canvas})
renderer.setSize(sizes.width, sizes.height)
renderer.render(scene, camera)
```

# Transform Objects

- Position


```
mesh.position.x = 0.7  
mesh.position.y = -0.6  
mesh.position.z = 1
```



```
mesh.position.set(0.7, -0.6, 1)
```

- Scale

```
mesh.scale.x = 0.5  
mesh.scale.y = 0.5  
mesh.scale.z = 1
```



```
mesh.scale.set(0.5, 0.5, 1)
```

- Rotation

```
mesh.rotation.reorder('YXZ')  
mesh.rotation.y = Math.PI * 0.25  
mesh.rotation.x = Math.PI * 0.25
```

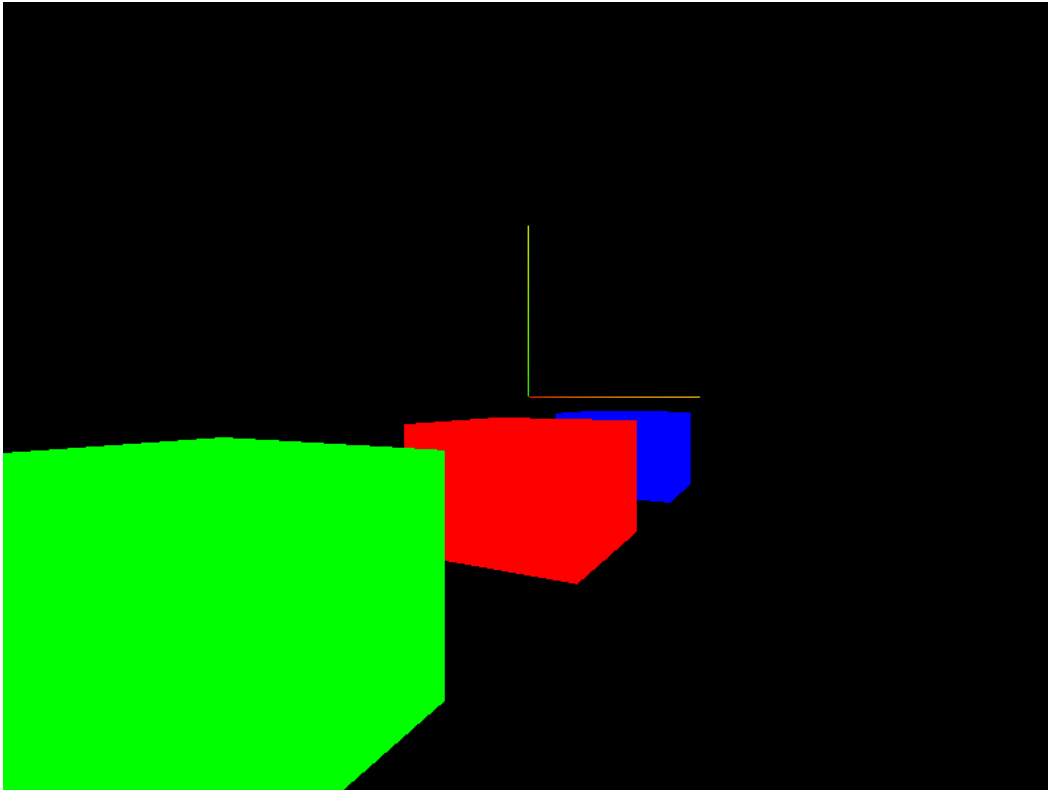
- Quaternion

```
// Axes helper  
const axesHelper = new THREE.AxesHelper()  
scene.add(axesHelper)
```

```
camera.lookAt(mesh.position)
```

# Group class

- Group inherit from Object3D



Modified in `script.js`

```
/**
 * Objects
 */
const group = new THREE.Group()
group.position.y = -0.5
group.scale.y = 0.7
group.rotation.y = 1.1
scene.add(group)

const cube1 = new THREE.Mesh(
  new THREE.BoxGeometry(1,1,1),
  new THREE.MeshBasicMaterial({color:0xff0000})
)
group.add(cube1)

const cube2 = new THREE.Mesh(
  new THREE.BoxGeometry(1,1,1),
  new THREE.MeshBasicMaterial({color:0x00ff00})
)
cube2.position.x = -2
group.add(cube2)

const cube3 = new THREE.Mesh(
  new THREE.BoxGeometry(1,1,1),
  new THREE.MeshBasicMaterial({color:0x0000ff})
)
cube3.position.x = 2
group.add(cube3)
```

# Animation

use `window.requestAnimationFrame(...)` to call this same function on the next frame

Modified in `script.js`

```
// Animations
const tick = () =>
{
  console.log('tick')
  window.requestAnimationFrame(tick)
}

tick()
```

Modified in `script.js`

```
// Time
let time = Date.now()

// Animations
const tick = () =>
{
  // Time
  const currentTime = Date.now()
  const deltaTime = currentTime - time
  time = currentTime

  console.log(deltaTime)

  // Update objects
  mesh.rotation.y += 0.001 * deltaTime

  // Render
  renderer.render(scene, camera)

  window.requestAnimationFrame(tick)
}

tick()
```

Modified in `script.js`

```
// Clock
const clock = new THREE.Clock()

// Animations
const tick = () =>
{
  // Clock
  const elapsedTime = clock.getElapsedTime()
  console.log(elapsedTime)

  // Update objects
  mesh.rotation.y = elapsedTime * Math.PI * 2
  mesh.position.y = Math.sin(elapsedTime)
  mesh.position.x = Math.cos(elapsedTime)

  // Render
  renderer.render(scene, camera)

  window.requestAnimationFrame(tick)
}

tick()
```

# GSAP Library

- More control, create tweens, create timelines

```
npm install --save gsap@3.12
```

Modified in `script.js`

```
gsap.to(mesh.position, {duration:1, delay:1, x:2})
gsap.to(mesh.position, {duration:1, delay:2, x:0})

// Animations
const tick = () =>
{
    // Render
    renderer.render(scene, camera)

    window.requestAnimationFrame(tick)
}

tick()
```

# Camera

PerspectiveCamera( fov : Number, aspect : Number, near : Number, far : Number )

- fov — Camera frustum vertical field of view.
- aspect — Camera frustum aspect ratio.
- near — Camera frustum near plane.
- far — Camera frustum far plane.

```
const camera = new THREE.PerspectiveCamera(  
  75,  
  sizes.width / sizes.height,  
  0.1,  
  100  
)
```

OrthographicCamera( left : Number, right : Number, top : Number, bottom : Number, near : Number, far : Number )

- left — Camera frustum left plane.
- right — Camera frustum right plane.
- top — Camera frustum top plane.
- bottom — Camera frustum bottom plane.
- near — Camera frustum near plane.
- far — Camera frustum far plane.

```
const aspectRatio = sizes.width / sizes.height  
const camera = new THREE.OrthographicCamera(  
  -1*aspectRatio,  
  1*aspectRatio,  
  1*aspectRatio,  
  -1*aspectRatio,  
  0.1,  
  100  
)
```

# Custom Control

Modified in `script.js`

```
// Cursor
const cursor = { x:0, y:0 }
window.addEventListener('mousemove', (event)=>
{
  cursor.x = event.clientX / sizes.width - 0.5
  cursor.y = event.clientY / sizes.height - 0.5
  console.log(cursor)
})
```

```
// Animations
const tick = () =>
{
  // Clock
  const elapsedTime = clock.getElapsedTime()

  camera.position.x = cursor.x * 10
  camera.position.y = cursor.y * 10
  //camera.lookAt(mesh.position)

  // Render
  renderer.render(scene, camera)

  window.requestAnimationFrame(tick)
}

tick()
```



```
// Animations
const tick = () =>
{
  // Clock
  const elapsedTime = clock.getElapsedTime()

  camera.position.x = Math.sin(cursor.x * Math.PI * 2) * 2
  camera.position.z = Math.cos(cursor.x * Math.PI * 2) * 2
  camera.position.y = cursor.y * 5
  camera.lookAt(mesh.position)

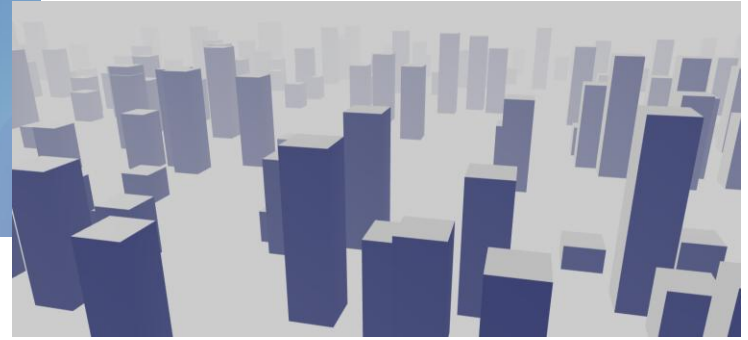
  // Render
  renderer.render(scene, camera)

  window.requestAnimationFrame(tick)
}

tick()
```

# Control Examples

- [ArcballControls](#)
- [DragControls](#)
- [FirstPersonControls](#)
- [FlyControls](#)
- [MapControls](#)
- [OrbitControls](#)
- [PointerLockControls](#)
- [TrackballControls](#)
- [TransformControls](#)



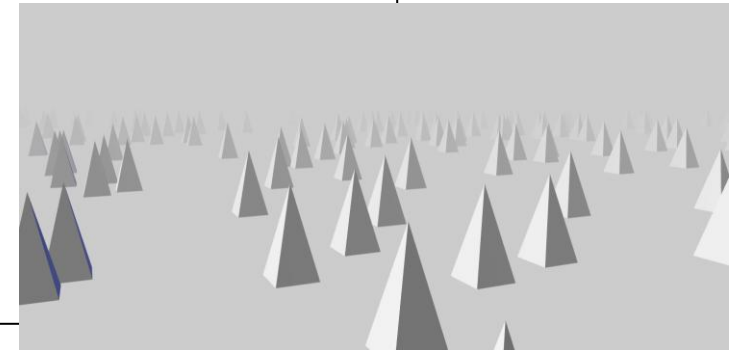
Modified in `script.js`

```
import { OrbitControls } from 'three/addons/controls/OrbitControls.js'

// Controls
const controls = new OrbitControls(camera, canvas)
controls.enableDamping = true

// Clock
const clock = new THREE.Clock()

// Animations
const tick = () =>
{
  // Clock
  const elapsedTime = clock.getElapsedTime()
  // Update controls
  controls.update()
  // Render
  renderer.render(scene, camera)
  window.requestAnimationFrame(tick)
}
tick()
```





# Fullscreen and Resizing

- Fit in the viewport
- To get the viewport width and height

```
window.innerWidth  
window.innerHeight
```

style.css

```
* {  
  margin: 0;  
  padding: 0;  
}  
  
html,  
body {  
  overflow: hidden;  
}  
  
.webgl {  
  position: fixed;  
  top: 0;  
  left: 0;  
  outline: none;  
}
```

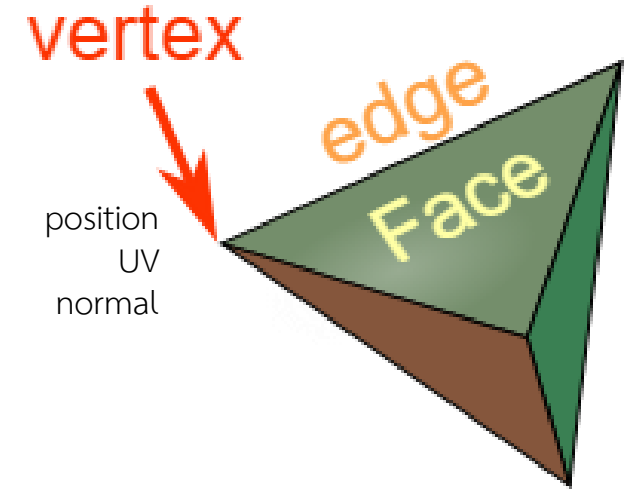
Modified in `script.js`

```
// Sizes  
const sizes = {  
  width: window.innerWidth,  
  height: window.innerHeight,  
}  
  
window.addEventListener("resize", () => {  
  // Update sizes  
  sizes.width = window.innerWidth  
  sizes.height = window.innerHeight  
  
  // Update camera  
  camera.aspect = sizes.width / sizes.height  
  camera.updateProjectionMatrix()  
  
  // Update renderer  
  renderer.setSize(sizes.width, sizes.height)  
  renderer.setPixelRatio(Math.min(window.devicePixelRatio, 2))  
})  
  
window.addEventListener('dblclick', ()=>{  
  if(!document.fullscreenElement)  
  {  
    canvas.requestFullscreen()  
  }  
  else  
  {  
    document.exitFullscreen()  
  }  
})
```

# Geometry

- [BoxGeometry](#)
- [BufferGeometry](#)
- [PlaneGeometry](#)
- [CircleGeometry](#)
- [ConeGeometry](#)
- [CylinderGeometry](#)
- [RingGeometry](#)
- [TorusGeometry](#)
- [TorusKnotGeometry](#)
- [DodecahedronGeometry](#)
- [OctahedronGeometry](#)
- [TetrahedronGeometry](#)
- [IcosahedronGeometry](#)
- [SphereGeometry](#)
- [ShapeGeometry](#)
- [TubeGeometry](#)
- [ExtrudeGeometry](#)
- [LatheGeometry](#)
- [TextGeometry](#)
- [Face3](#)

<https://www.mathsisfun.com/geometry/vertices-faces-edges.html>



Modified in `script.js`

```
const geometry = new THREE.BufferGeometry()
const positionsArray = new Float32Array([0, 0, 0, 0, 1, 0, 1, 0, 0])
const positionsAttribute = new THREE.BufferAttribute(positionsArray, 3)
geometry.setAttribute('position', positionsAttribute)
```

Modified in `script.js`

```
const geometry = new THREE.BufferGeometry()
const count = 500
const positionsArray = new Float32Array(count*3*3)
for(let i=0;i<count*3*3;i++)
{
    positionsArray[i] = Math.random()
}
const positionsAttribute = new THREE.BufferAttribute(positionsArray, 3)
geometry.setAttribute('position', positionsAttribute)
```

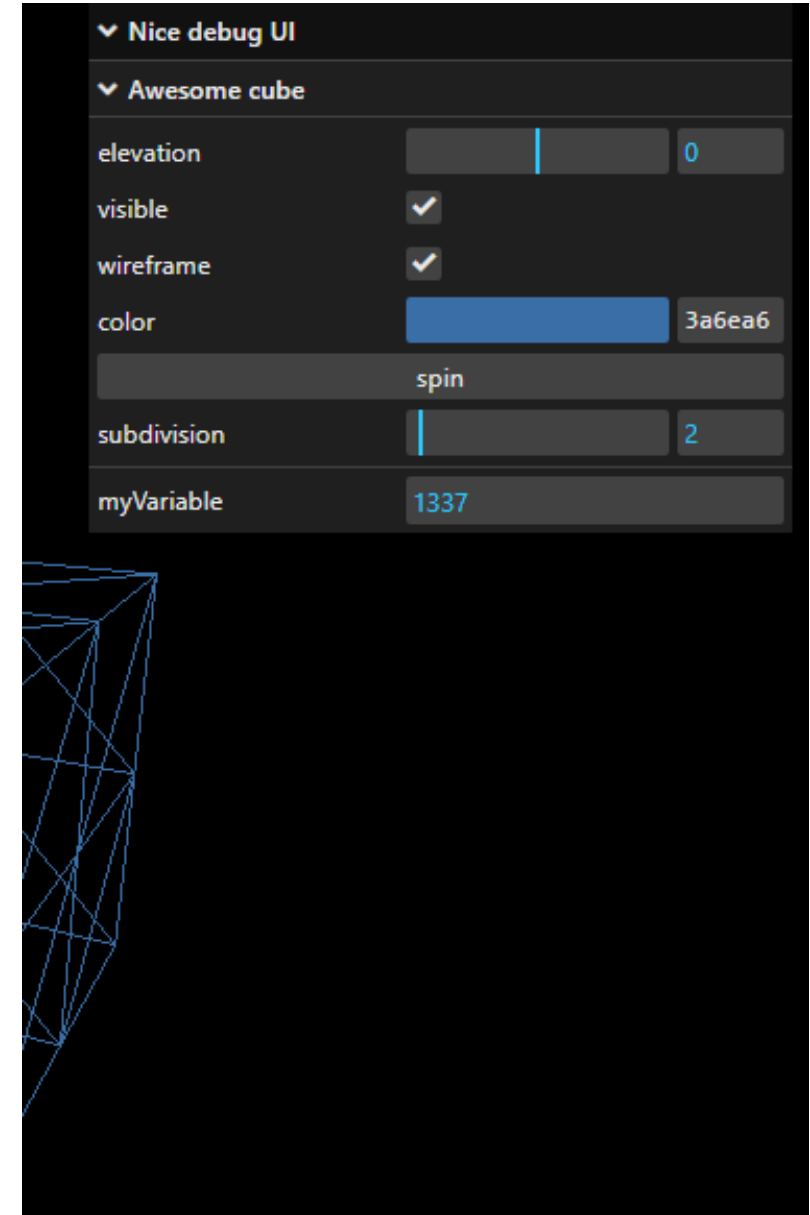
# Debug UI

- [dat.GUI](#)
- [lil-gui](#)
- [control-panel](#)
- [ControlKit](#)
- [Uil](#)
- [Tweakpane](#)
- [Guify](#)
- [Oui](#)

```
npm install lil-gui
```

Tweak properties of the objects with

Range	for numbers with minimum and maximum value
Color	for colors with various formats
Text	for simple texts
Checkbox	for booleans (true or false)
Select	for a choice from a list of values
Button	to trigger functions



Modified in `script.js`

```
import GUI from "lil-gui";

// Debug
const gui = new GUI({
  width: 300,
  title: 'Nice debug UI',
  closeFolders: false
});

window.addEventListener('keydown', (event)=>
{
  if(event.key == 'h')
    gui.show(gui._hidden)
})
const debugObject = {}
```

```
debugObject.color = "#3a6ea6"

const cubeTweaks = gui.addFolder('Awesome cube')
cubeTweaks.close()

cubeTweaks
  .add(mesh.position, "y").min(-3).max(3).step(0.01).name("elevation")

cubeTweaks.add(mesh, "visible")

cubeTweaks.add(material, "wireframe")

cubeTweaks.addColor(debugObject, "color")
  .onChange(() => {
    material.color.set(debugObject.color)
  })

debugObject.spin = () => {
  gsap.to(mesh.rotation, { y: mesh.rotation.y + Math.PI * 2 })
}
cubeTweaks.add(debugObject, 'spin')

debugObject.subdivision = 2
cubeTweaks
  .add(debugObject, 'subdivision').min(1).max(20).step(1)
  .onChange(()=> {
    mesh.geometry.dispose()
    mesh.geometry = new THREE.BoxGeometry(
      1,1,1,
      debugObject.subdivision, debugObject.subdivision, debugObject.subdivision
    )
  })

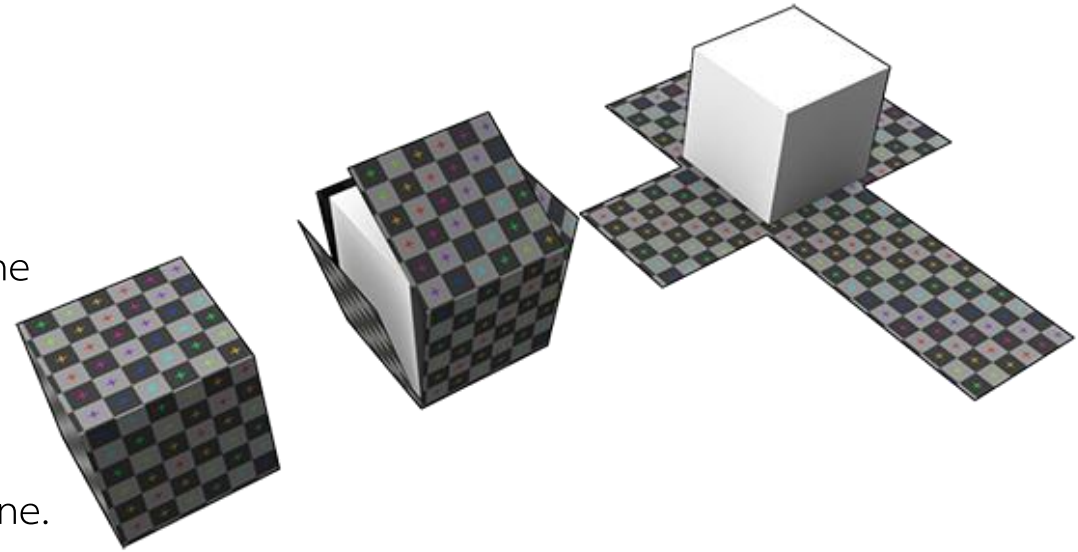
const myObject = {
  myVariable: 1337,
}
gui.add(myObject, "myVariable")
```

# Physically-Based Rendering

- <https://marmoset.co/posts/basic-theory-of-physically-based-rendering/>
- <https://marmoset.co/posts/physically-based-rendering-and-you-can-too/>

# Texture

- Texture is being stretched or squeezed in different ways to cover the geometry. It is called UV unwrapping.
- It like unwrapping an origami or a candy wrap to make it flat.
- Each vertex will have a 2D coordinate on a flat (usually square) plane.
- You need to load image and convert to texture



```
// Texture
const image = new Image()
const texture = new THREE.Texture(image)
texture.colorSpace = THREE.SRGBColorSpace

image.onload = () =>
{
    texture.needsUpdate = true
}

image.src = '/textures/door/color.jpg'
```



```
// Texture
const textureLoader = new THREE.TextureLoader()
const texture = textureLoader.load('/textures/door/color.jpg')
texture.colorSpace = THREE.SRGBColorSpace
```

```
const material = new THREE.MeshBasicMaterial({ map: texture })
```

Modified in `script.js`

```
// Texture
const loadingManager = new THREE.LoadingManager()
loadingManager.onStart = () =>
{
  console.log('loading started')
}
loadingManager.onLoad = () =>
{
  console.log('loading finished')
}
loadingManager.onProgress = () =>
{
  console.log('loading progressing')
}
loadingManager.onError = () =>
{
  console.log('loading error')
}

const textureLoader = new THREE.TextureLoader(loadingManager)
const colorTexture = textureLoader.load('/textures/door/color.jpg')
colorTexture.colorSpace = THREE.SRGBColorSpace
const alphaTexture = textureLoader.load('/textures/door/alpha.jpg')
const heightTexture = textureLoader.load('/textures/door/height.jpg')
const normalTexture = textureLoader.load('/textures/door/normal.jpg')
const ambientOcclusionTexture =
textureLoader.load('/textures/door/ambientOcclusion.jpg')
const metalnessTexture = textureLoader.load('/textures/door/metalness.jpg')
const roughnessTexture = textureLoader.load('/textures/door/roughness.jpg')
```

## Texture Transformation

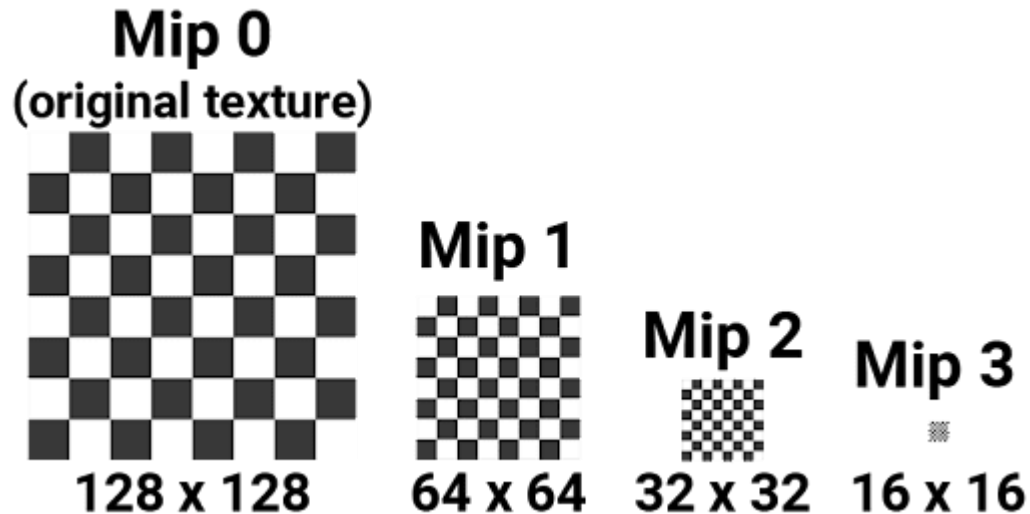
Modified in `script.js`

```
colorTexture.repeat.x = 2
colorTexture.repeat.y = 3
colorTexture.wrapS = THREE.MirroredRepeatWrapping
colorTexture.wrapT = THREE.RepeatWrapping

colorTexture.offset.x = 0.5
colorTexture.offset.y = 0.5
colorTexture.rotation = Math.PI / 4
colorTexture.center.x = 0.5
colorTexture.center.y = 0.5
```

# Texture Mipmap and Minification/Magification Filter

- Mipmapping is a technic that consists of creating half a smaller version of a texture again and again until to a 1x1 texture



- The minification filter happens when the pixels of texture are smaller than the pixels of the render.
  - THREE.NearestFilter
  - THREE.LinearFilter
  - THREE.NearestMipmapNearestFilter
  - THREE.NearestMipmapLinearFilter
  - THREE.LinearMipmapNearestFilter
  - THREE.LinearMipmapLinearFilter
- The magnification filter happens when the pixels of the texture are bigger than the render's pixels.
  - THREE.NearestFilter
  - THREE.LinearFilter

```
colorTexture.generateMipmaps = false  
colorTexture.minFilter = THREE.NearestFilter  
colorTexture.magFilter = THREE.NearestFilter
```



# Preparing Textures

- Weight
  - .jpg : lossy compression but usually lighter
  - .png : lossless compression but usually heavier
  - Use compression website : TinyPNG
- Size (resolution)
  - GPU has storage limitations; all pixels of the texture send and store to GPU
  - Try to reduce the size of images as much as possible
  - The texture width and height must be a power of 2 : 512x512 1024x1024 512x2048
- Data
  - Texture support transparency which only support in .png or use 2 .jpg
  - A normal texture should have the exact values as texture, which better use in .png

## Texture on the Web

- Poliigon.com
- 3dtextures.me
- Arroyo-texture.ch

## Software

- Photoshop
- Substance Designer

# Material

- Material are used to put a color on each visible pixel of the geometries.
- Algorithms that decide on the color of each pixel are written in programs called shaders.
- Three.js has many built-in materials with pre-made shaders.

```
// Objects
// MeshBasicMaterial
const material = new THREE.MeshBasicMaterial()

const sphere = new THREE.Mesh(
  new THREE.SphereGeometry(0.5, 64, 64),
  material
)
sphere.position.x = -1.5

const plane = new THREE.Mesh(
  new THREE.PlaneGeometry(1, 1, 100, 100),
  material
)

const torus = new THREE.Mesh(
  new THREE.TorusGeometry(0.3, 0.2, 64, 128),
  material
)
torus.position.x = 1.5

scene.add(sphere, plane, torus)
```

```
// Animate
const clock = new THREE.Clock()

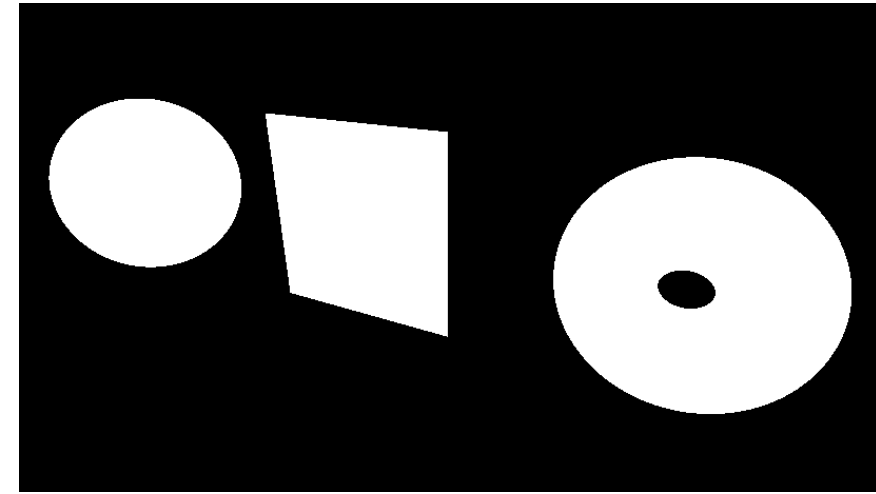
const tick = () =>
{
  const elapsedTime = clock.getElapsedTime()

  // Update objects
  sphere.rotation.y = 0.1 * elapsedTime
  plane.rotation.y = 0.1 * elapsedTime
  torus.rotation.y = 0.1 * elapsedTime

  sphere.rotation.x = -0.15 * elapsedTime
  plane.rotation.x = -0.15 * elapsedTime
  torus.rotation.x = -0.15 * elapsedTime

  // ...
}

tick()
```



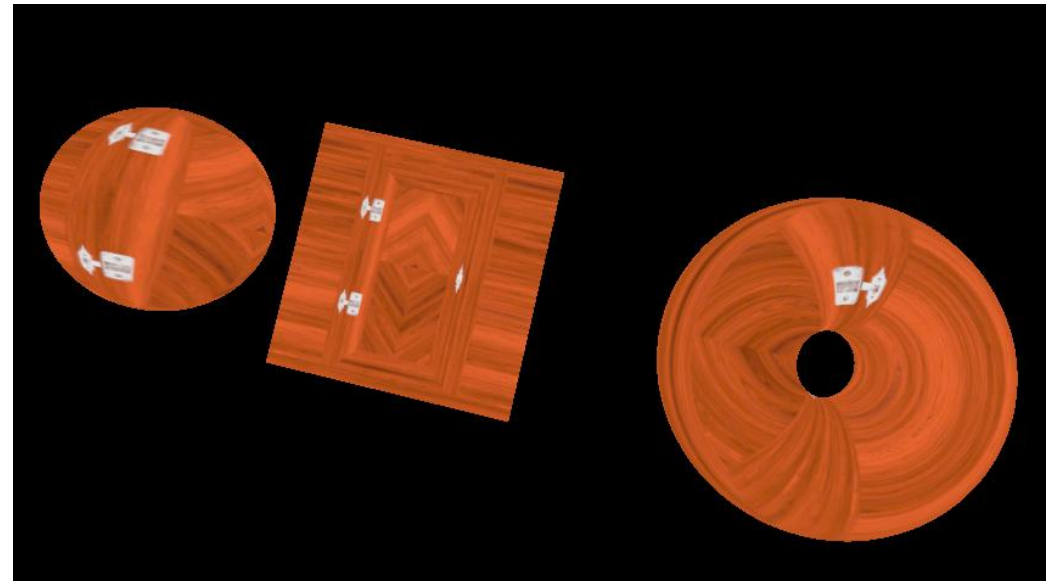
```
// Textures
const textureLoader = new THREE.TextureLoader()

const doorColorTexture = textureLoader.load('./textures/door/color.jpg')
const doorAlphaTexture = textureLoader.load('./textures/door/alpha.jpg')
const doorAmbientOcclusionTexture = textureLoader.load('./textures/door/ambientOcclusion.jpg')
const doorHeightTexture = textureLoader.load('./textures/door/height.jpg')
const doorNormalTexture = textureLoader.load('./textures/door/normal.jpg')
const doorMetalnessTexture = textureLoader.load('./textures/door/metalness.jpg')
const doorRoughnessTexture = textureLoader.load('./textures/door/roughness.jpg')
const matcapTexture = textureLoader.load('./textures/matcaps/3.png')
const gradientTexture = textureLoader.load('./textures/gradients/5.jpg')

doorColorTexture.colorSpace = THREE.SRGBColorSpace
matcapTexture.colorSpace = THREE.SRGBColorSpace
```

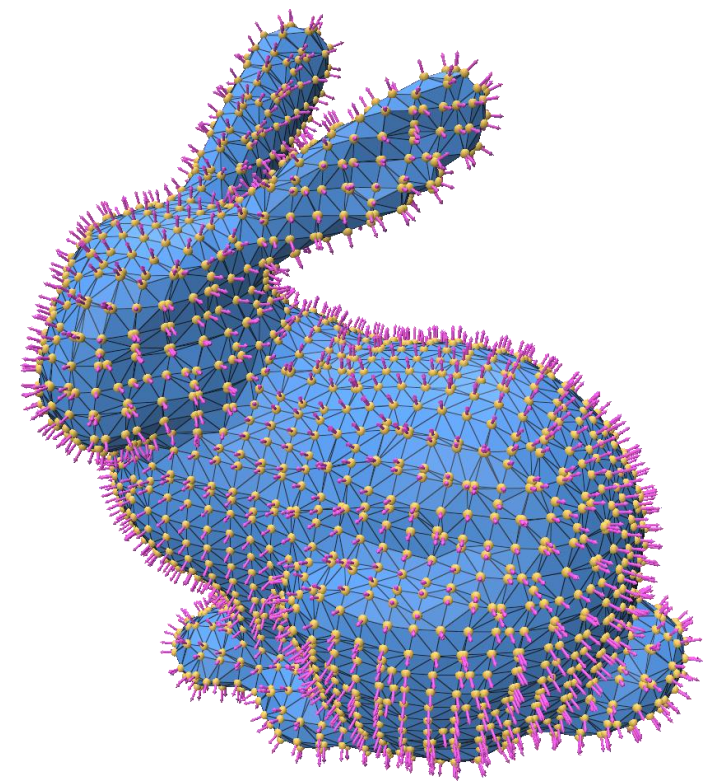
```
// Objects
// MeshBasicMaterial
const material = new THREE.MeshBasicMaterial()
material.map = doorColorTexture
```

```
material.color = new THREE.Color(0xff0000)
material.wireframe = true
material.transparent = true
material.opacity = 0.5
material.alphaMap = doorAlphaTexture
material.side = THREE.DoubleSide
```



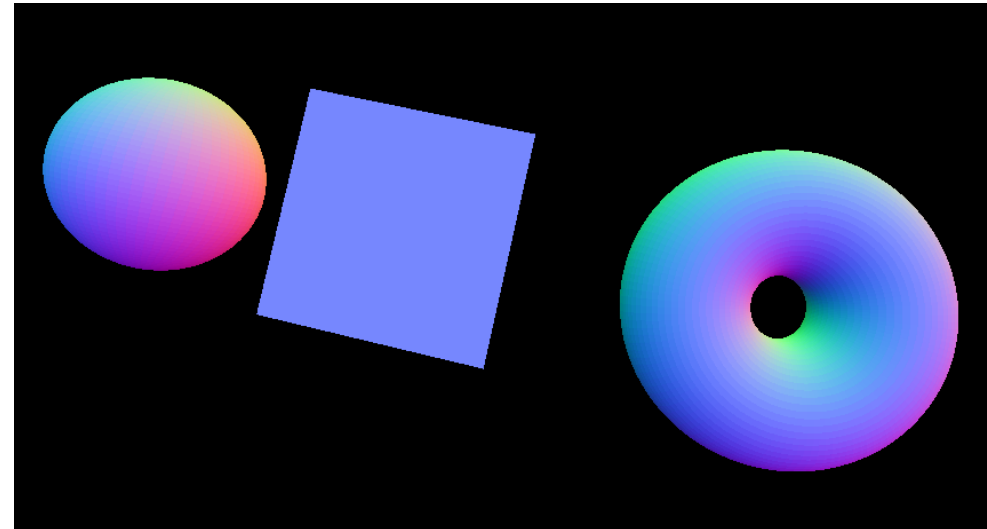
# Mesh Normal Material

- Normals are information encoded in each vertex that contains the direction of the outside of the face.
- Use normals to calculate how to illuminate the face or how the environment should reflect or refract on the geometries' surface.
- flatShading will flatten the faces, meaning that the normals won't be interpolated between the vertices.



[https://fwilliams.info/point-cloud-utils/sections/mesh\\_normal\\_estimation/](https://fwilliams.info/point-cloud-utils/sections/mesh_normal_estimation/)

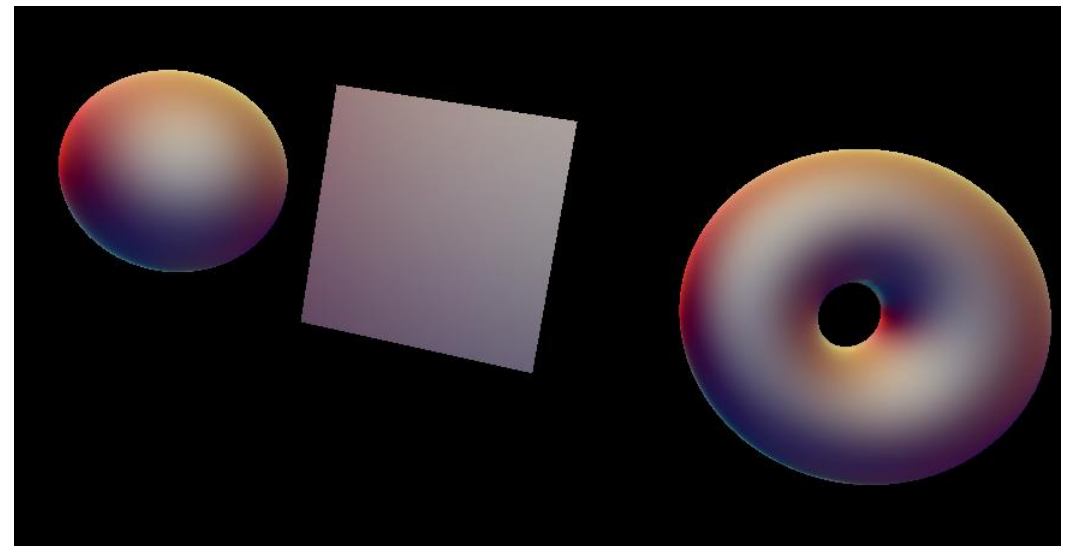
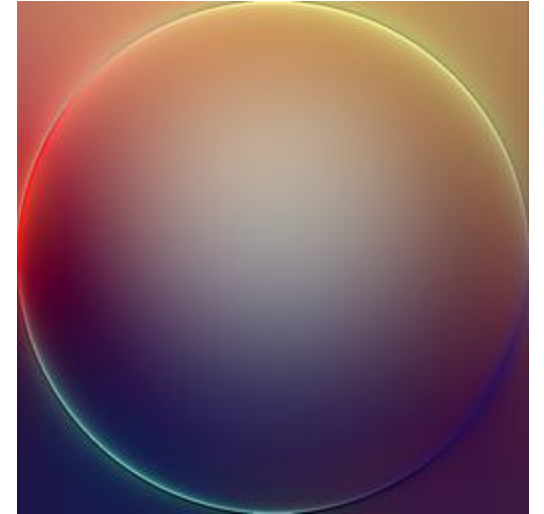
```
// MeshNormalMaterial  
const material = new THREE.MeshNormalMaterial  
material.flatShading = true
```



# Mesh Matcap Material

- MeshMatcapMaterial is defined by a MatCap (or Lit Sphere) texture, which encodes the material color and shading.
- MeshMatcapMaterial does not respond to lights since the matcap image file encodes baked lighting.
- Load matcaps textures at <https://github.com/nidorx/matcaps> please check license before use it.
- Create your matcaps at <https://www.kchapelier.com/matcap-studio/>

```
// MeshMatcapMaterial
const material = new THREE.MeshMatcapMaterial()
material.matcap = matcapTexture
```



# Mesh Depth Material

- A material for drawing geometry by depth. Depth is based off of the camera near and far plane. White is nearest, black is farthest.

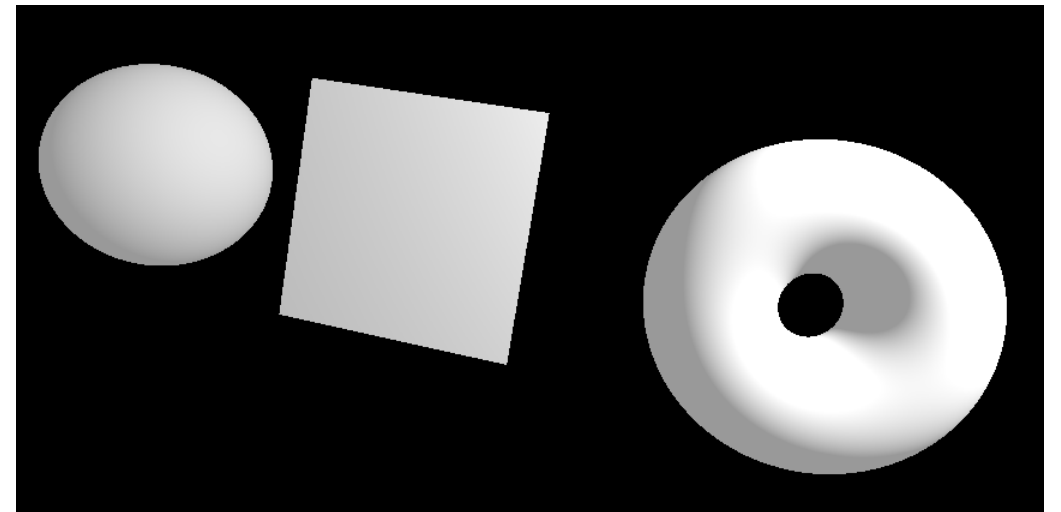
```
// MeshDepthMaterial  
const material = new THREE.MeshDepthMaterial()
```



# Mesh Lambert Material

- A material for non-shiny surfaces, without specular highlights.
- The material uses a non-physically based Lambertian model for calculating reflectance.

```
// MeshLambertMaterial  
const material = new THREE.MeshLambertMaterial()  
  
// Light  
const ambientLight = new THREE.AmbientLight(0xffffff, 1)  
scene.add(ambientLight)  
  
const pointLight = new THREE.PointLight(0xffffff, 30)  
pointLight.position.x = 3  
pointLight.position.y = 2  
scene.add(pointLight)
```



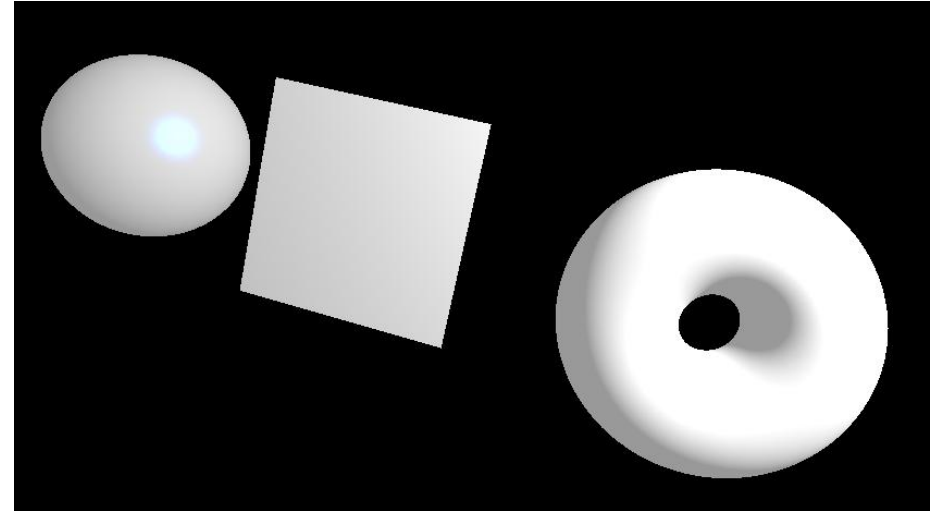
# Mesh Phong Material

- A material for shiny surfaces with specular highlights.
- The material uses a non-physically based Blinn-Phong model for calculating reflectance.

```
// MeshPhongMaterial
const material = new THREE.MeshPhongMaterial()
material.shininess = 100
material.specular = new THREE.Color(0x1188ff)
```

```
// Light
const ambientLight = new THREE.AmbientLight(0xffffff, 1)
scene.add(ambientLight)

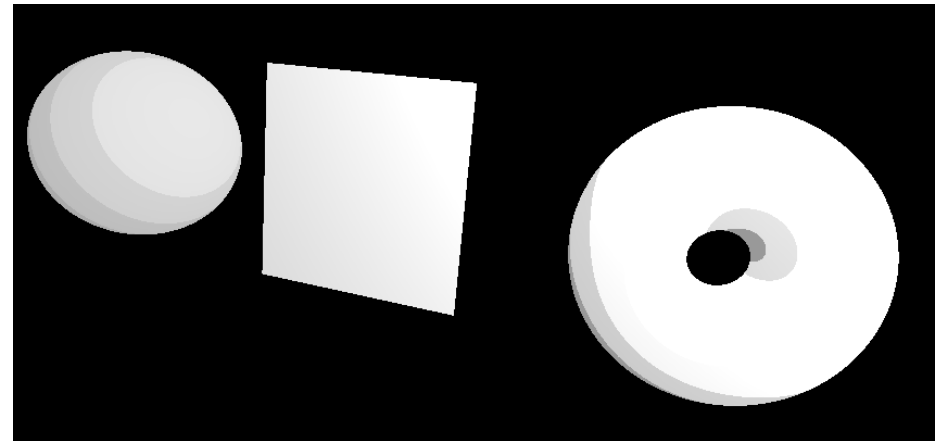
const pointLight = new THREE.PointLight(0xffffff, 30)
pointLight.position.x = 3
pointLight.position.y = 2
scene.add(pointLight)
```



# Mesh Toon Material

- A material implementing toon shading.

```
// MeshToonMaterial
const material = new THREE.MeshToonMaterial()
gradientTexture.minFilter = THREE.NearestFilter
gradientTexture.magFilter = THREE.NearestFilter
gradientTexture.generateMipmaps = false
material.gradientMap = gradientTexture
```





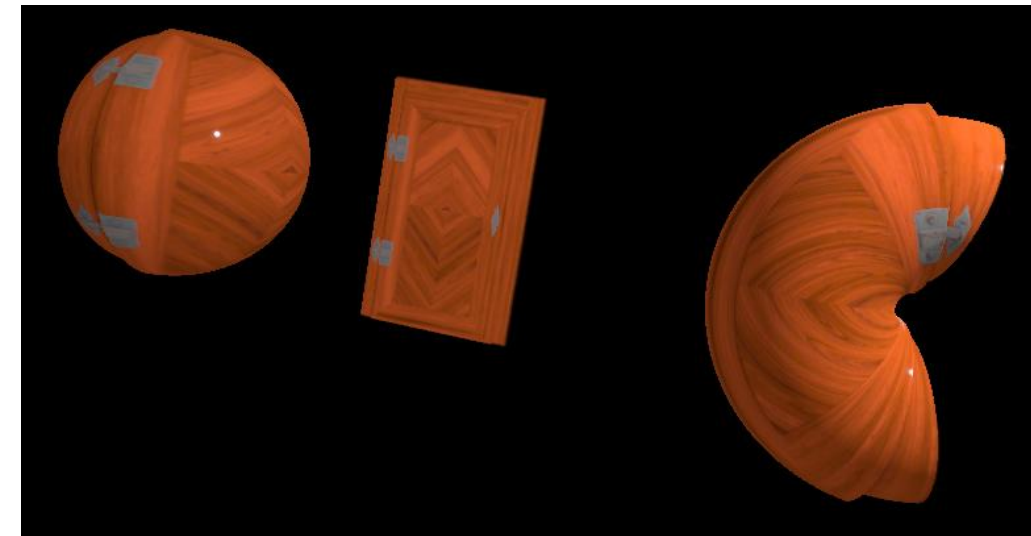
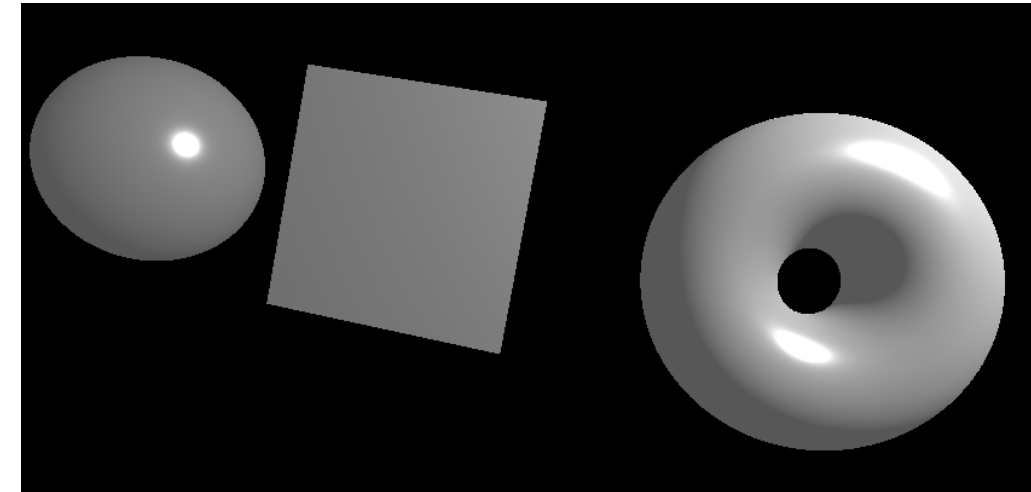
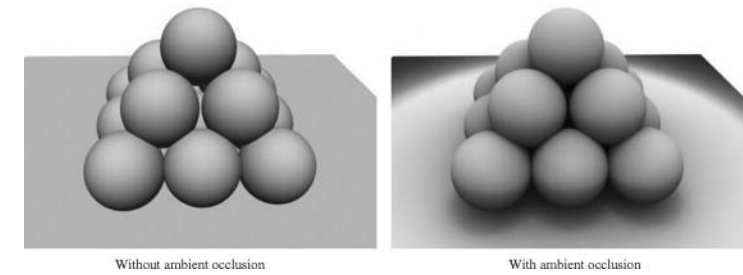
# Mesh Standard Material

- A standard physically based material, using Metallic-Roughness workflow.
- Physically based rendering (PBR) has recently become the standard in many 3D applications, such as Unity, Unreal and 3D Studio Max.

```
// MeshStandardMaterial
const material = new THREE.MeshStandardMaterial()
material.metalness = 0.7
material.roughness = 0.2
```

```
material.map = doorColorTexture
material.aoMap = doorAmbientOcclusionTexture
material.aoMapIntensity = 1
material.displacementMap = doorHeightTexture
material.displacementScale = 0.1
material.metalnessMap = doorMetalnessTexture
material.roughnessMap = doorRoughnessTexture
material.normalMap = doorNormalTexture
material.normalScale.set(0.5, 0.5)
material.transparent = true
material.alphaMap = doorAlphaTexture
```

```
gui.add(material, 'metalness').min(0).max(1).step(0.0001)
gui.add(material, 'roughness').min(0).max(1).step(0.0001)
```





# Environment Map

- The environment map is like an image of what's surrounding the scene.
- Add reflection, refraction to objects, in addition to the current DirectionalLight and AmbientLight.



```
import { RGBELoader } from 'three/examples/jsm/loaders/RGBELoader.js'

// Environment map
const rgbeLoader = new RGBELoader()
rgbeLoader.load('./textures/environmentMap/2k.hdr', (environmentMap)=>
{
  environmentMap.mapping = THREE.EquirectangularReflectionMapping
  scene.background = environmentMap
  scene.environment = environmentMap
}))
```

# Mesh Physical Material

- An extension of the MeshStandardMaterial, providing more advanced physically-based rendering properties:
  - **Anisotropy:** Ability to represent the anisotropic property of materials as observable with brushed metals.
  - **Clearcoat:** Some materials — like car paints, carbon fiber, and wet surfaces — require a clear, reflective layer on top of another layer that may be irregular or rough. Clearcoat approximates this effect, without the need for a separate transparent surface.
  - **Iridescence:** Allows to render the effect where hue varies depending on the viewing angle and illumination angle. This can be seen on soap bubbles, oil films, or on the wings of many insects.
  - **Physically-based transparency:** One limitation of .opacity is that highly transparent materials are less reflective. Physically-based .transmission provides a more realistic option for thin, transparent surfaces like glass.
  - **Advanced reflectivity:** More flexible reflectivity for non-metallic materials.
  - **Sheen:** Can be used for representing cloth and fabric materials.

```
// Clearcoat
material.clearcoat = 1
material.clearcoatRoughness = 0

gui.add(material, 'clearcoat').min(0).max(1).step(0.0001)
gui.add(material, 'clearcoatRoughness').min(0).max(1).step(0.0001)
```

[https://threejs.org/examples/#webgl\\_materials\\_physical\\_clearcoat](https://threejs.org/examples/#webgl_materials_physical_clearcoat)

[https://threejs.org/examples/?q=sheen#webgl\\_loader\\_gltf\\_sheen](https://threejs.org/examples/?q=sheen#webgl_loader_gltf_sheen)

```
// Sheen
material.sheen = 1
material.sheenRoughness = 0.25
material.sheenColor.set(1, 1, 1)

gui.add(material, 'sheen').min(0).max(1).step(0.0001)
gui.add(material, 'sheenRoughness').min(0).max(1).step(0.0001)
gui.addColor(material, 'sheenColor')
```

```
// Iridescence
material.iridescence = 1
material.iridescenceIOR = 1
material.iridescenceThicknessRange = [ 100, 800 ]

gui.add(material, 'iridescence').min(0).max(1).step(0.0001)
gui.add(material, 'iridescenceIOR').min(0).max(2.333).step(0.0001)
gui.add(material.iridescenceThicknessRange, '0').min(1).max(1000).step(1)
gui.add(material.iridescenceThicknessRange, '1').min(1).max(1000).step(1)
```

[https://threejs.org/examples/?q=anis#webgl\\_loader\\_gltf\\_anisotropy](https://threejs.org/examples/?q=anis#webgl_loader_gltf_anisotropy)

[https://threejs.org/examples/?q=physica#webgl\\_materials\\_physical\\_transmission\\_alpha](https://threejs.org/examples/?q=physica#webgl_materials_physical_transmission_alpha)

ior stands for Index Of Refraction and depends on the type of material.

A diamond has an ior of 2.417, water has an ior of 1.333 and air has an ior of 1.000293.

[https://en.wikipedia.org/wiki/List\\_of\\_refractive\\_indices](https://en.wikipedia.org/wiki/List_of_refractive_indices)

```
// Transmission
material.transmission = 1
material.ior = 1.5
material.thickness = 0.5
gui.add(material, 'transmission').min(0).max(1).step(0.0001)
gui.add(material, 'ior').min(1).max(10).step(0.0001)
gui.add(material, 'thickness').min(0).max(1).step(0.0001)
```

# Other Materials

- PointsMaterial to handle particles, their size, their color, what's drawn in them, etc.  
<https://threejs.org/docs/index.html#api/en/materials/PointsMaterial>
- ShaderMaterial and RawShaderMaterial can both be used to create your own materials using a special language named GLSL.  
<https://threejs.org/docs/index.html#api/en/materials/ShaderMaterial>  
<https://threejs.org/docs/index.html#api/en/materials/RawShaderMaterial>

# 3D Text

- Convert a font with tools : <https://gero3.github.io/facetype.js/>
- Use fonts provided by Three.js in /node\_modules/three/examples/fonts and put in the /static/ folder

```
import { FontLoader } from 'three/examples/jsm/loaders/FontLoader.js'
import { TextGeometry } from 'three/examples/jsm/geometries/TextGeometry.js'

// Axes helper
const axesHelper = new THREE.AxesHelper()
scene.add(axesHelper)

// Textures
const textureLoader = new THREE.TextureLoader()
const matcapTexture = textureLoader.load('/textures/matcaps/8.png')
matcapTexture.colorSpace = THREE.SRGBColorSpace
```



```
// Fonts
const fontLoader = new FontLoader()
fontLoader.load(
  '/fonts/helvetiker_regular.typeface.json',
  (font) =>
  {
    const textGeometry = new TextGeometry(
      'Hello World!',
      {
        font: font,
        size: 0.5,
        depth: 0.2,
        curveSegments: 5,
        bevelEnabled: true,
        bevelThickness: 0.03,
        bevelSize: 0.02,
        bevelOffset: 0,
        bevelSegments: 4
      }
    )
    textGeometry.center()
    const material = new THREE.MeshMatcapMaterial()
    material.matcap = matcapTexture
    const text = new THREE.Mesh(textGeometry, material)
    scene.add(text)
  }
)
```

```

// Fonts
const fontLoader = new FontLoader()
fontLoader.load(
  '/fonts/helvetiker_regular.typeface.json',
  (font) =>
  {

    // ...

    const donutGeometry = new THREE.TorusGeometry(0.3, 0.2, 20, 45)
    for(let i=0; i < 500; i++)
    {
      const donut = new THREE.Mesh(donutGeometry, material)
      donut.position.x = (Math.random() - 0.5) * 10
      donut.position.y = (Math.random() - 0.5) * 10
      donut.position.z = (Math.random() - 0.5) * 10

      donut.rotation.x = Math.random() * Math.PI
      donut.rotation.y = Math.random() * Math.PI

      const scale = Math.random()
      donut.scale.set(scale, scale, scale)

      scene.add(donut)
    }
  }
)

```

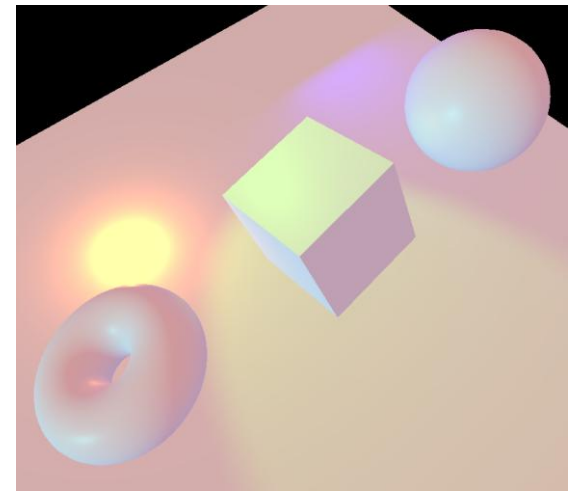


# Light

- Light can cost a lot computation.
- Try to add as few lights as possible and try to use the lights that cost less to high

AmbientLight, HemisphereLight,  
DirectionalLight, PointLight,  
SpotLight, RectAreaLight

- Bake light: bake the light into the texture and the lights can not move



```
const ambientLight = new THREE.AmbientLight(0xffffff, 1.5)
scene.add(ambientLight)

const directionalLight = new THREE.DirectionalLight(0x00fffc, 1.2)
directionalLight.position.set(1, 0.25, 0)
scene.add(directionalLight)

const hemisphereLight = new THREE.HemisphereLight(0xff0000, 0x0000ff, 1.0)
scene.add(hemisphereLight)

const pointLight = new THREE.PointLight(0xff9000, 1.5, 10, 2)
pointLight.position.set(1, -0.5, 1)
scene.add(pointLight)

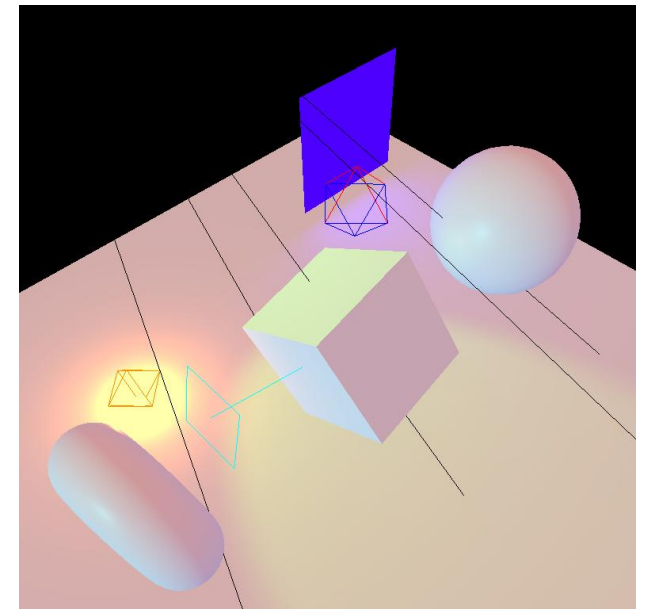
const rectAreaLight = new THREE.RectAreaLight(0x4e00ff, 2, 1, 1)
rectAreaLight.position.set(-1.5, 0, 1.5)
scene.add(rectAreaLight)

const spotLight = new THREE.SpotLight(0x78ff00, 4.5, 10, Math.PI * 0.1,
0.25, 1)
spotLight.position.set(0, 2, 3)
scene.add(spotLight)
```



# Light Helper

- LightHelper shows the position and orientation of the lights
  - HemisphereLightHelper
  - DirectionalLightHelper
  - PointLightHelper
  - RectAreaLightHelper
  - SpotLightHelper



```
import { RectAreaLightHelper } from 'three/examples/jsm/helpers/RectAreaLightHelper.js'

// ...

// Helper
const hemisphereLightHelper = new THREE.HemisphereLightHelper(hemisphereLight, 0.2)
scene.add(hemisphereLightHelper)

const directionalLightHelper = new THREE.DirectionalLightHelper(directionalLight, 0.2)
scene.add(directionalLightHelper)

const pointLightHelper = new THREE.PointLightHelper(pointLight, 0.2)
scene.add(pointLightHelper)

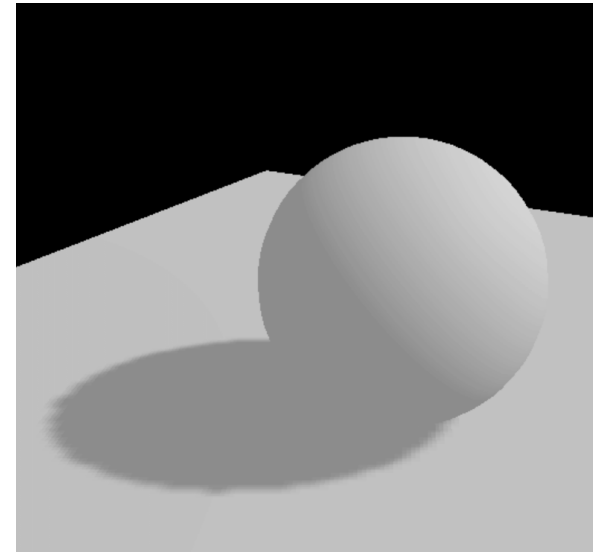
const spotLightHelper = new THREE.SpotLightHelper(spotLight, 0.2)
scene.add(spotLightHelper)

const rectAreaLightHelper = new RectAreaLightHelper(rectAreaLight)
scene.add(rectAreaLightHelper)
```



# Shadows

- Dark shadow is in the back of the objects.
- Drop shadow is the shadows on the other objects.
- How to display realistic shadows at a reasonable frame rate.
- Three.js will do a render for each light supporting shadows, a `MeshDepthMaterial` replaces all meshes materials.
- The light renders are stored as textures called shadow maps.
- Go through each object of the scene and decide if the object can cast a shadow with the `castShadow` property, and if the object can receive shadow with the `receiveShadow` property.
- Activate shadows on as few lights as possible
- Only the following types of lights support shadows: `PointLight`, `DirectionalLight`, `SpotLight`.



```
sphere.castShadow = true  
  
// ...  
  
plane.receiveShadow = true
```

```
const directionalLight = new THREE.DirectionalLight(0xffffff, 1.5)  
directionalLight.castShadow = true
```

# Shadow Map Optimization

- By default, the shadow map size is only 512x512 for performance, which can improve with power of 2.

```
directionalLight.shadow.mapSize.width = 1024  
directionalLight.shadow.mapSize.height = 1024
```

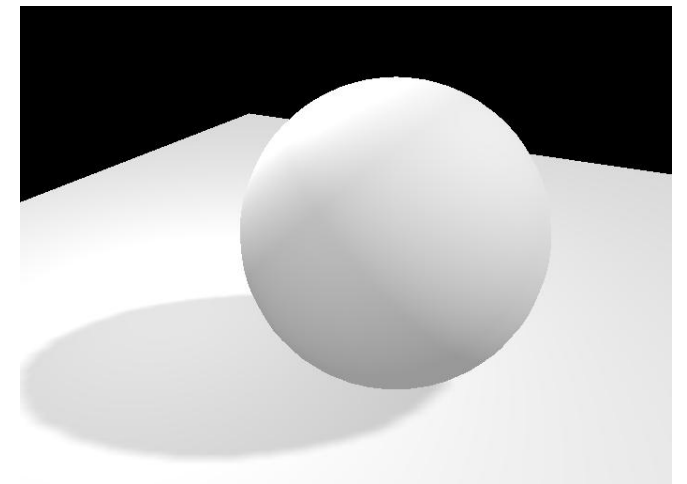
- Adjust the near, far and top, right, bottom, left to fit the scene by using CameraHelper.

```
const directionalLightCameraHelper = new THREE.CameraHelper(directionalLight.shadow.camera)  
directionalLightCameraHelper.visible = true  
scene.add(directionalLightCameraHelper)
```

```
directionalLight.shadow.camera.near = 1  
directionalLight.shadow.camera.far = 6  
directionalLight.shadow.camera.top = 2  
directionalLight.shadow.camera.right = 2  
directionalLight.shadow.camera.bottom = -2  
directionalLight.shadow.camera.left = -2
```

- Control the shadow blur with the radius property

```
directionalLight.shadow.radius = 10
```

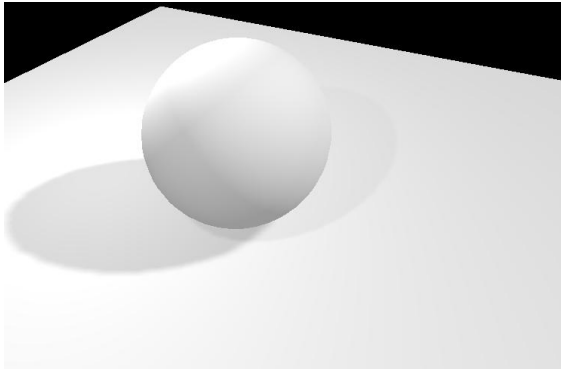


# Shadow Map Algorithm

- Different types of algorithms can be applied to shadow maps:
  - THREE.BasicShadowMap: Very performant but lousy quality
  - THREE.PCFShadowMap: Less performant but smoother edges
  - THREE.PCFSoftShadowMap: Less performant but even softer edges
  - THREE.VSMShadowMap: Less performant, more constraints, can have unexpected results
- The default is THREE.PCFShadowMap but you can use THREE.PCFSoftShadowMap for better quality.

```
// Renderer
const renderer = new THREE.WebGLRenderer({
  canvas: canvas
})
renderer.setSize(sizes.width, sizes.height)
renderer.setPixelRatio(Math.min(window.devicePixelRatio, 2))
renderer.shadowMap.enabled = true
renderer.shadowMap.type = THREE.PCFSoftShadowMap
```

# SpotLight

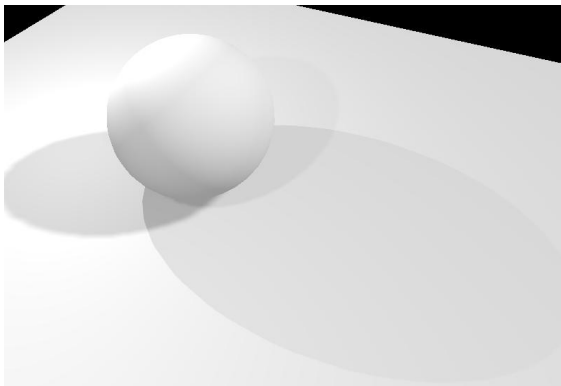


```
// Spot light
const spotLight = new THREE.SpotLight(0xffffff, 3.6, 10, Math.PI * 0.3)
spotLight.position.set(0, 2, 2)
scene.add(spotLight)
scene.add(spotLight.target)

spotLight.castShadow = true
spotLight.shadow.mapSize.width = 1024
spotLight.shadow.mapSize.height = 1024
spotLight.shadow.camera.near = 1
spotLight.shadow.camera.far = 6

const spotLightCameraHelper = new THREE.CameraHelper(spotLight.shadow.camera)
spotLightCameraHelper.visible = true
scene.add(spotLightCameraHelper)
```

# PointLight



```
// Point light
const pointLight = new THREE.PointLight(0xffffff, 2.7)
pointLight.position.set(- 1, 1, 0)
scene.add(pointLight)

pointLight.castShadow = true
pointLight.shadow.mapSize.width = 1024
pointLight.shadow.mapSize.height = 1024
pointLight.shadow.camera.near = 0.1
pointLight.shadow.camera.far = 5

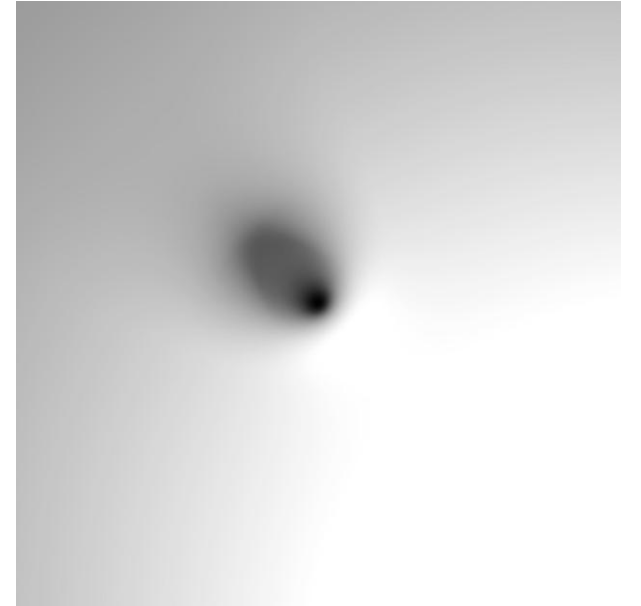
const pointLightCameraHelper = new THREE.CameraHelper(pointLight.shadow.camera)
pointLightCameraHelper.visible = true
scene.add(pointLightCameraHelper)
```

# Baking Shadows

- Baking shadows are integrated into textures that we apply on materials.
- Commenting all the shadows related lines of code, or simply deactivate them in the renderer

```
// Texture
const textureLoader = new THREE.TextureLoader()
const bakedShadow = textureLoader.load('/textures/bakedShadow.jpg')
bakedShadow.colorSpace = THREE.SRGBColorSpace
```

```
const plane = new THREE.Mesh(
  new THREE.PlaneGeometry(5, 5),
  new THREE.MeshBasicMaterial({map:bakedShadow})
)
```



# Simple Shadows

- A less realistic but more dynamic solution would be to use a more simple shadow under the sphere and slightly above the plane.

```
// Texture
const textureLoader = new THREE.TextureLoader()
const simpleShadow = textureLoader.load('/textures/simpleShadow.jpg')
simpleShadow.colorSpace = THREE.SRGBColorSpace
```

```
const plane = new THREE.Mesh(
  new THREE.PlaneGeometry(5, 5),
  material
)
```

```
const sphereShadow = new THREE.Mesh(
  new THREE.PlaneGeometry(1.5, 1.5),
  new THREE.MeshBasicMaterial({
    color: 0x000000,
    transparent: true,
    alphaMap: simpleShadow
  })
)
sphereShadow.rotation.x = - Math.PI * 0.5
sphereShadow.position.y = plane.position.y + 0.01

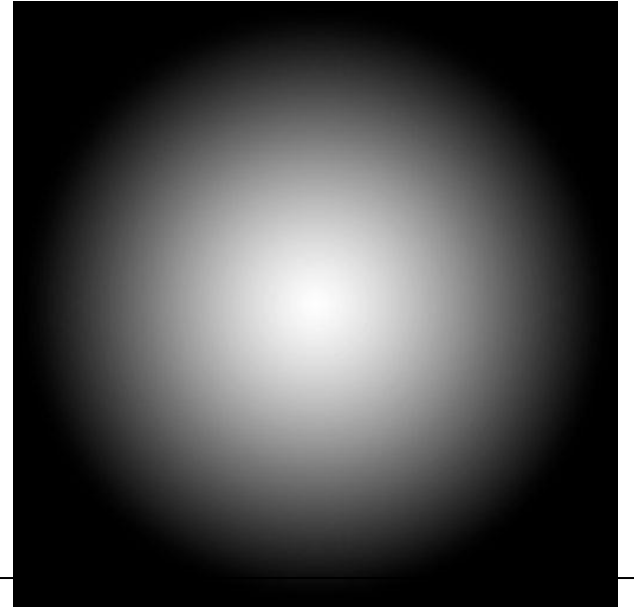
scene.add(sphere, sphereShadow, plane)
```

```
// Animate
const clock = new THREE.Clock()
const tick = () =>
{
  const elapsedTime = clock.getElapsedTime()

  // Update the sphere
  sphere.position.x = Math.cos(elapsedTime) * 1.5
  sphere.position.z = Math.sin(elapsedTime) * 1.5
  sphere.position.y = Math.abs(Math.sin(elapsedTime * 3))

  // Update the shadow
  sphereShadow.position.x = sphere.position.x
  sphereShadow.position.z = sphere.position.z
  sphereShadow.material.opacity = (1 - sphere.position.y) * 0.3

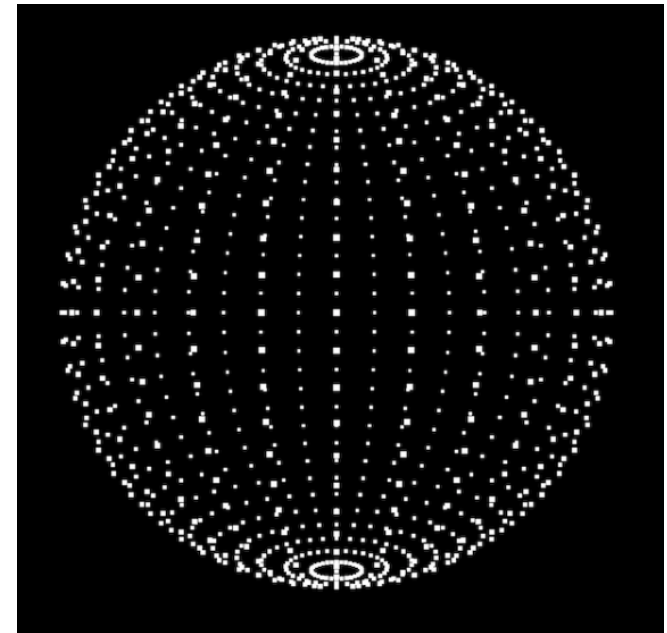
  // ...
}
tick()
```



# Particles

- Particles can be used to create stars, smoke, rain, dust, fire, etc.
- Creating particles is as simple as making a Mesh. We need a BufferGeometry, a material that can handle particles (PointsMaterial), and instead of producing a Mesh we need to create a Points.

```
/*  
 * Particles  
 */  
// geometry  
const particlesGeometry = new THREE.SphereGeometry(1, 32, 32)  
  
// material  
const particlesMaterial = new THREE.PointsMaterial()  
particlesMaterial.size = 0.02  
particlesMaterial.sizeAttenuation = true  
  
// points  
const particles = new THREE.Points(particlesGeometry, particlesMaterial)  
scene.add(particles)
```



# Custom Geometry

```
// geometry
const particlesGeometry = new THREE.BufferGeometry()
const count = 15000
const positions = new Float32Array(count * 3)
const colors = new Float32Array(count * 3)
for(let i=0; i<count*3; i++)
{
    positions[i] = (Math.random() - 0.5) * 10
    colors[i] = Math.random()
}
particlesGeometry.setAttribute('position', new THREE.BufferAttribute(positions, 3))
particlesGeometry.setAttribute('color', new THREE.BufferAttribute(colors, 3))
```

# Custom Material

```
// Textures
const textureLoader = new THREE.TextureLoader()
const particlesTexture = textureLoader.load('/textures/particles/2.png')
```

```
// material
const particlesMaterial = new THREE.PointsMaterial()
particlesMaterial.size = 0.1
particlesMaterial.sizeAttenuation = true
particlesMaterial.color = new THREE.Color('#ff88cc')
particlesMaterial.transparent = true
particlesMaterial.alphaMap = particlesTexture

//particlesMaterial.alphaTest = 0.001
//particlesMaterial.depthTest = false
particlesMaterial.depthWrite = false
particlesMaterial.blending = THREE.AdditiveBlending
particlesMaterial.vertexColors = true
```

```
// Test cube
const cube = new THREE.Mesh(
    new THREE.BoxGeometry(),
    new THREE.MeshBasicMaterial()
)
scene.add(cube)
```



# Animate Particles

```
// Animate
const clock = new THREE.Clock()

const tick = () =>
{
    const elapsedTime = clock.getElapsedTime()

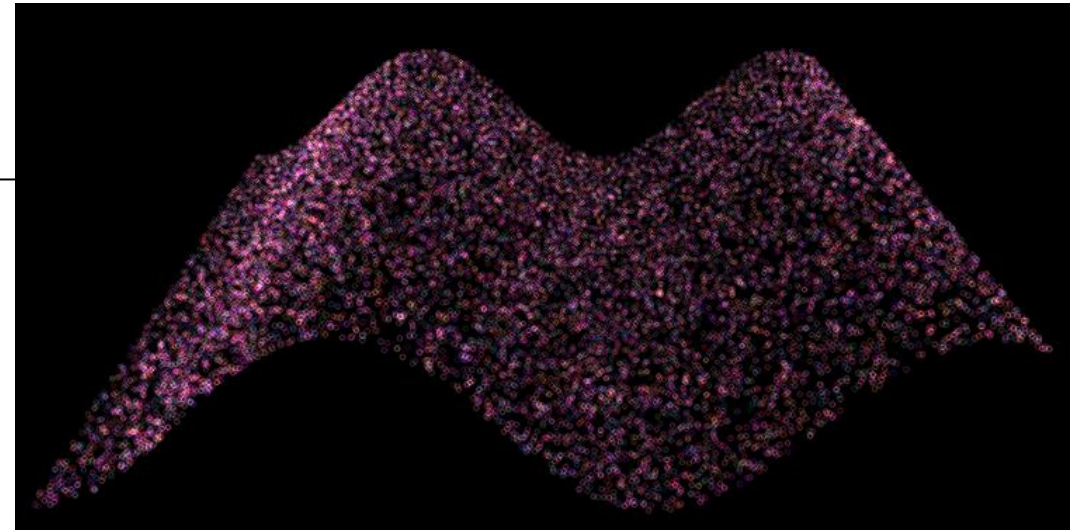
    // update particles
    particles.rotation.y = elapsedTime * 0.2
    for(let i=0; i<count; i++)
    {
        const i3 = i*3
        const x = particlesGeometry.attributes.position.array[i3]
        particlesGeometry.attributes.position.array[i3+1] = Math.sin(elapsedTime + x)
    }
    particlesGeometry.attributes.position.needsUpdate = true

    // Update controls
    controls.update()

    // Render
    renderer.render(scene, camera)

    // Call tick again on the next frame
    window.requestAnimationFrame(tick)
}

tick()
```



## Fun 06: Three.js

- สร้าง Scene 3 มิติ ตามจินตนาการ