Intro to Web Graphics

From pixels to interactive 3D — your first steps into browser-based graphics.

Beginner-Friendly

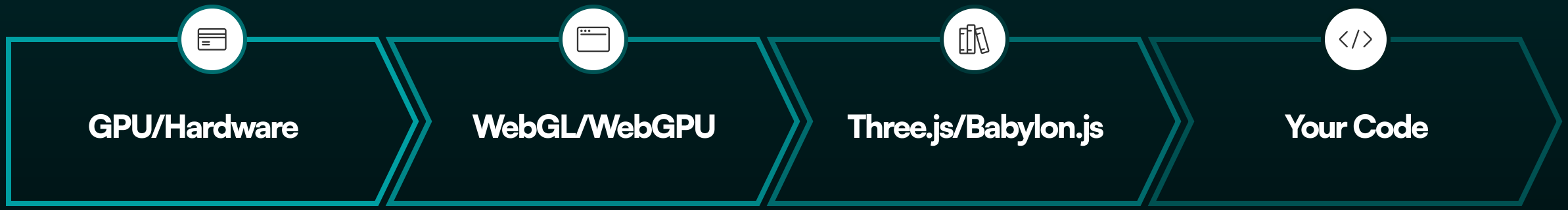
Hands-On Activity Included.

What Are Web Graphics?

Visual content rendered directly in the browser — no plugins required.

- **2D Graphics**
  Shapes, images, charts, and animations on a flat canvas.
- **3D Graphics**
  Interactive models, scenes, and immersive experiences.
- **Everywhere**
  Games, data viz, product viewers, art installations.

The Web Graphics Stack



Your graphics card does the heavy lifting.

Low-level browser API that talks to the GPU.

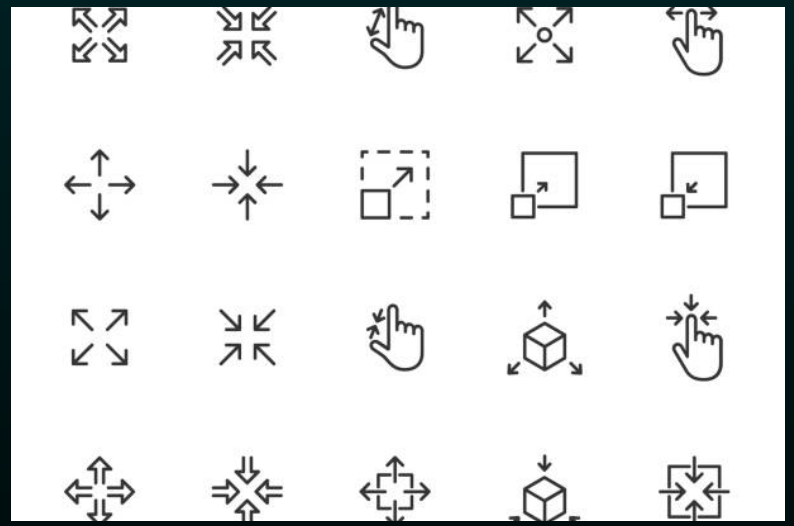
Friendly libraries that wrap WebGL complexity — "We focus here!"

JavaScript you write to create scenes and interactions.

Choosing Your Tool

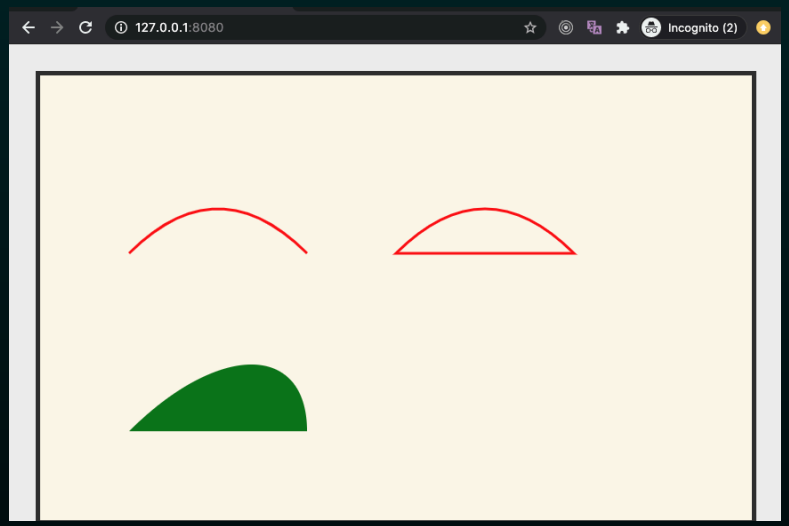
Comparison of graphics technologies

SVG



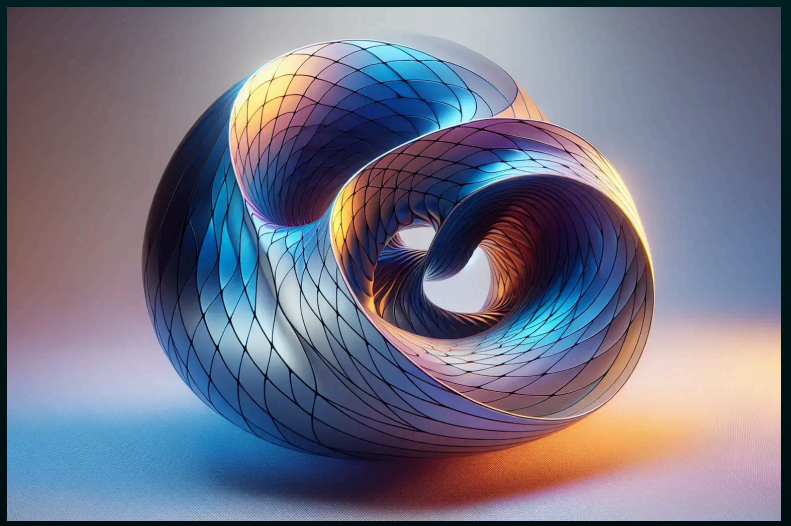
Best for icons, logos, charts, scalable 2D graphics — handles hundreds of elements.

Canvas 2D



Best for games, image editing, animations, data viz — handles thousands of pixels.

WebGL



Best for 3D scenes, VR/AR, heavy visual effects — handles millions of triangles.

What is WebGL?

Web Graphics Library - a JavaScript API for rendering 2D and 3D graphics in the browser.

Built on OpenGL ES

The same technology that powers mobile games.

Runs on the GPU

Leverages blazing-fast parallel processing.

No plugins needed

Works seamlessly in every modern browser.

Low-level API

Powerful but verbose, hence the need for libraries like Three.js.

```
// A small taste of raw WebGL "boilerplate"
const vertexShaderSource = `
  attribute vec4 position;
  void main() {
    gl_Position = position;
  }
`;

const buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);

const positionLocation = gl.getAttribLocation(program, "position");
gl.enableVertexAttribArray(positionLocation);
gl.vertexAttribPointer(positionLocation, 2, gl.FLOAT, false, 0, 0);

gl.drawArrays(gl.TRIANGLES, 0, 3);
// ...and 50 more lines just to draw a triangle!
  return go(f, seed, [])
}
```

Meet Three.js

The most popular JavaScript library for 3D web graphics.

GitHub Stars

100K+

stars

First Release

2010

year

Sites Using It

1M+

sites

Abstracts away hundreds of lines of WebGL boilerplate.

Huge ecosystem of plugins/loaders/examples.

Great documentation and beginner-friendly.

Used by Google, Apple, Nike, NASA, and many more.

The Big Three Core Objects

Every Three.js app needs these three core objects

1. **Scene**

   The container that holds all your 3D objects, lights, and cameras — think of it as your virtual stage.

2. **Camera**

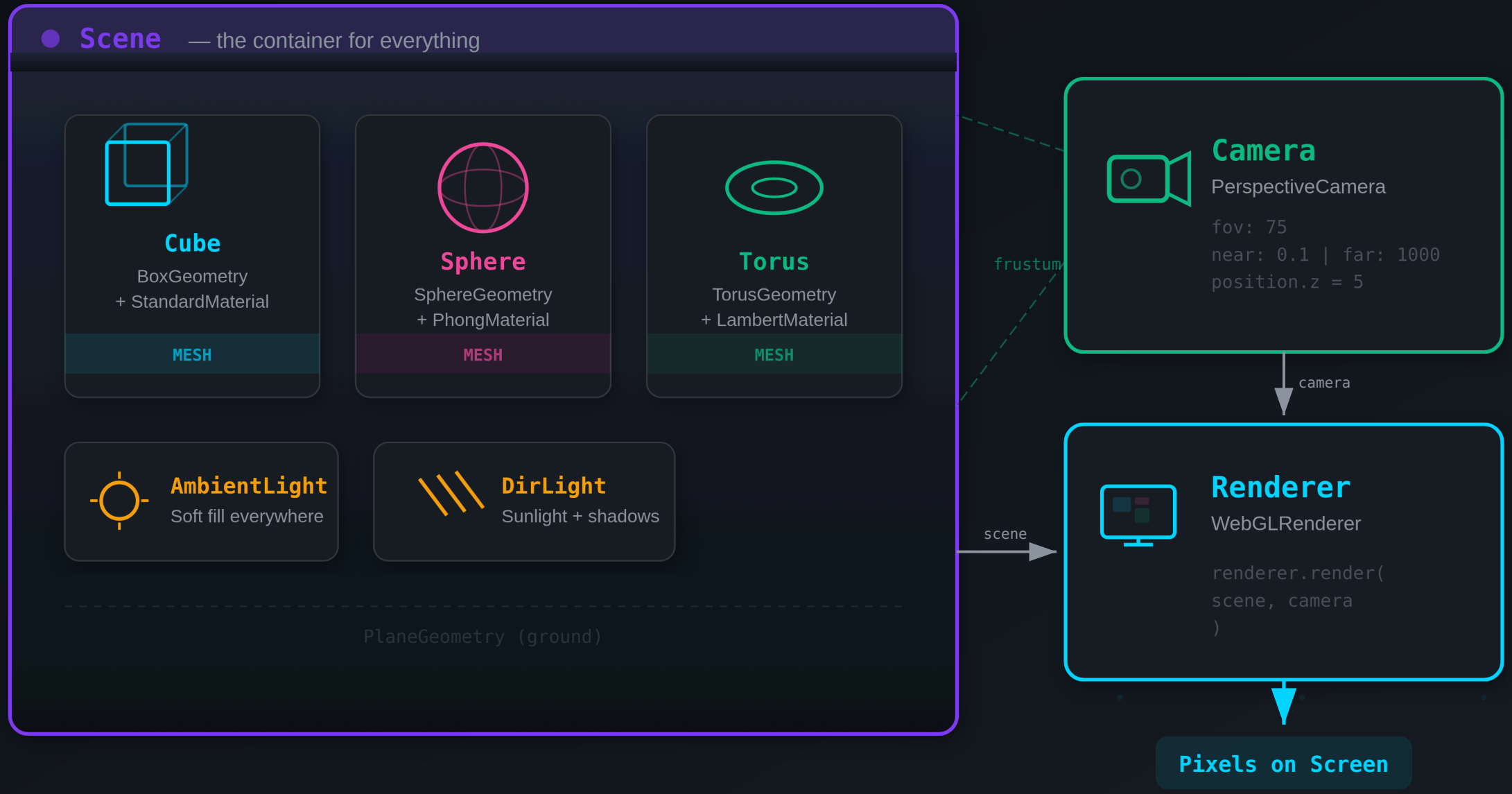
   Defines the viewpoint — PerspectiveCamera mimics how human eyes see, things farther away look smaller.

3. **Renderer**

   Takes the scene + camera and draws pixels to the screen — WebGLRenderer is the standard choice.

How Three.js Renders a Frame

Scene + Camera → Renderer → Screen



Building 3D Objects in Three.js

The Core Components: Geometry, Material, and Mesh

Geometry + Material = Mesh

Geometry

Defines the shape and wireframe of the 3D object. - BoxGeometry - SphereGeometry - CylinderGeometry - PlaneGeometry - TorusGeometry

Material

Defines the surface appearance and how light interacts with it. - MeshBasicMaterial - MeshStandardMaterial - MeshPhongMaterial - MeshLambertMaterial - MeshNormalMaterial

Mesh

The combination of Geometry and Material, creating a visible 3D object in your scene.

These three elements are fundamental to constructing any visible 3D object in Three.js.

```
// 1. Define the Shape (Geometry)
const geometry = new THREE.BoxGeometry(1, 1, 1);

// 2. Define the Surface (Material)
const material = new THREE.MeshStandardMaterial({ color: 0xff0000 });

// 3. Combine into a Mesh
const cube = new THREE.Mesh(geometry, material);

// 4. Add to the world
scene.add(cube);
```

Lighting Your Scene

Understanding Light Types in 3D Environments

- **AmbientLight**
  Soft, even light everywhere, no shadows, good as a base fill.
- **DirectionalLight**
  Sunlight-like, parallel rays from one direction, casts shadows.
- **PointLight**
  Light bulb, emits in all directions from a single point.
- **SpotLight**
  Flashlight cone, focused beam with angle and distance falloff.

Without lights, your objects are invisible (unless using BasicMaterial).

Camera & Controls

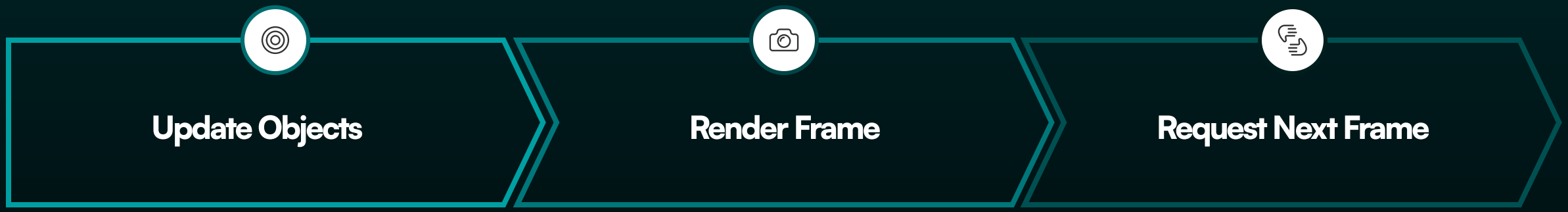
Understanding PerspectiveCamera and OrbitControls

PerspectiveCamera Mimics human vision, objects farther away appear smaller. Parameters: fov (field of view), aspect (width/height ratio), near (closest visible distance), far (farthest visible distance).

OrbitControls Lets users interactively orbit around a target point. Left drag = rotate/orbit Right drag = pan Scroll = zoom in/out Touch = works on mobile too.

The Render Loop

Animation runs at approximately 60 frames per second using requestAnimationFrame.



Move, rotate, or change properties.

The camera captures the scene as pixels.

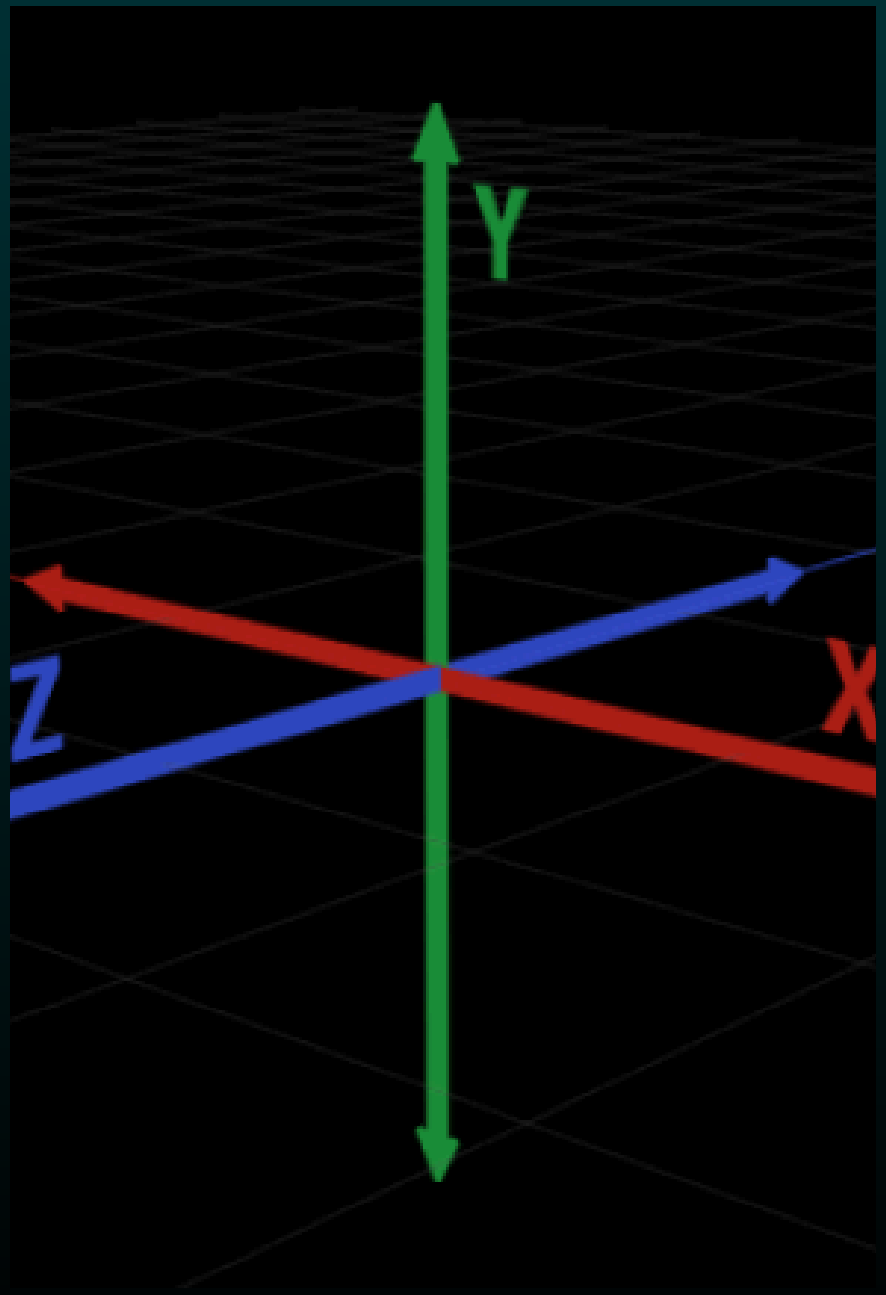
The browser calls your function again.

```
function animate() {
  // 1. Schedule the next frame
  requestAnimationFrame(animate);

  // 2. Update object properties (Animation)
  cube.rotation.x += 0.01;
  cube.rotation.y += 0.01;

  // 3. Draw the scene from the camera's perspective
  renderer.render(scene, camera);
}

animate(); // Start the loop
```



Axis Definitions

The 3D Coordinate System

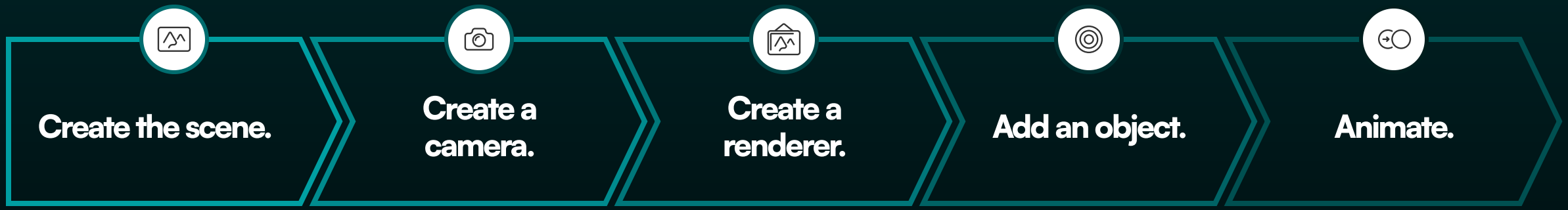
X axis: Left/Right (positive = right)

Y axis: Up/Down (positive = up)

Z axis: Forward/Back (positive = toward you)

Minimal Three.js Setup

Full annotated code walkthrough



This sequence covers the fundamental steps for setting up a basic Three.js scene and animation.

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

// 1. Scene
const scene = new THREE.Scene();

// 2. Camera (Field of view, Aspect ratio, Near, Far)
const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.innerHeight, 0.1, 1000);
camera.position.z = 5;

// 3. Renderer
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

// 4. Object
const mesh = new THREE.Mesh(
  new THREE.BoxGeometry(),
  new THREE.MeshNormalMaterial()
);
scene.add(mesh);

// 5. Start Animation
function animate() {
  requestAnimationFrame(animate);
  mesh.rotation.y += 0.01;
  renderer.render(scene, camera);
}
animate();
```

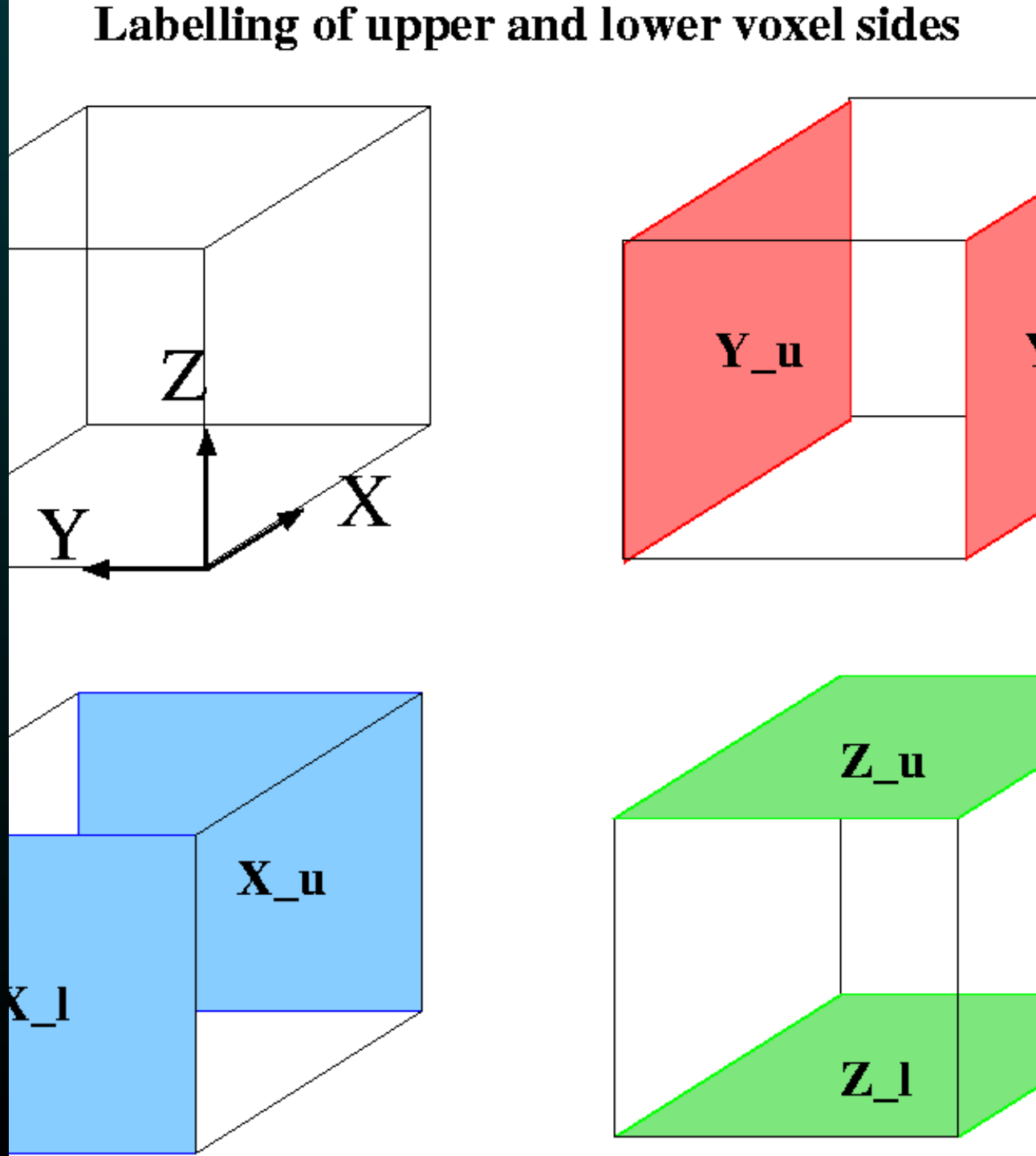
Core Transformations

Transformations

Every object can be moved, rotated, and scaled.

- **Position: Move an object in 3D space along X, Y, or Z.**
- **Rotation: Spin an object around any axis in radians.**
- **Scale: Stretch or shrink along each axis independently.**

[Code examples for each transformation]



```
// Move 2 units to the right
mesh.position.x = 2;

// Rotate 45 degrees (converted to Radians)
mesh.rotation.y = Math.PI / 4;

// Double the size on the Y axis (vertical stretch)
mesh.scale.set(1, 2, 1);

// Short-hand for moving all at once
mesh.position.set(0, 5, -10);
```

Materials at a Glance

Choose the right material for the look and performance you need:

1

Basic

No lighting needed, flat color, fast — unlit.

2

Lambert

Matte surfaces, diffuse light only — low cost.

3

Phong

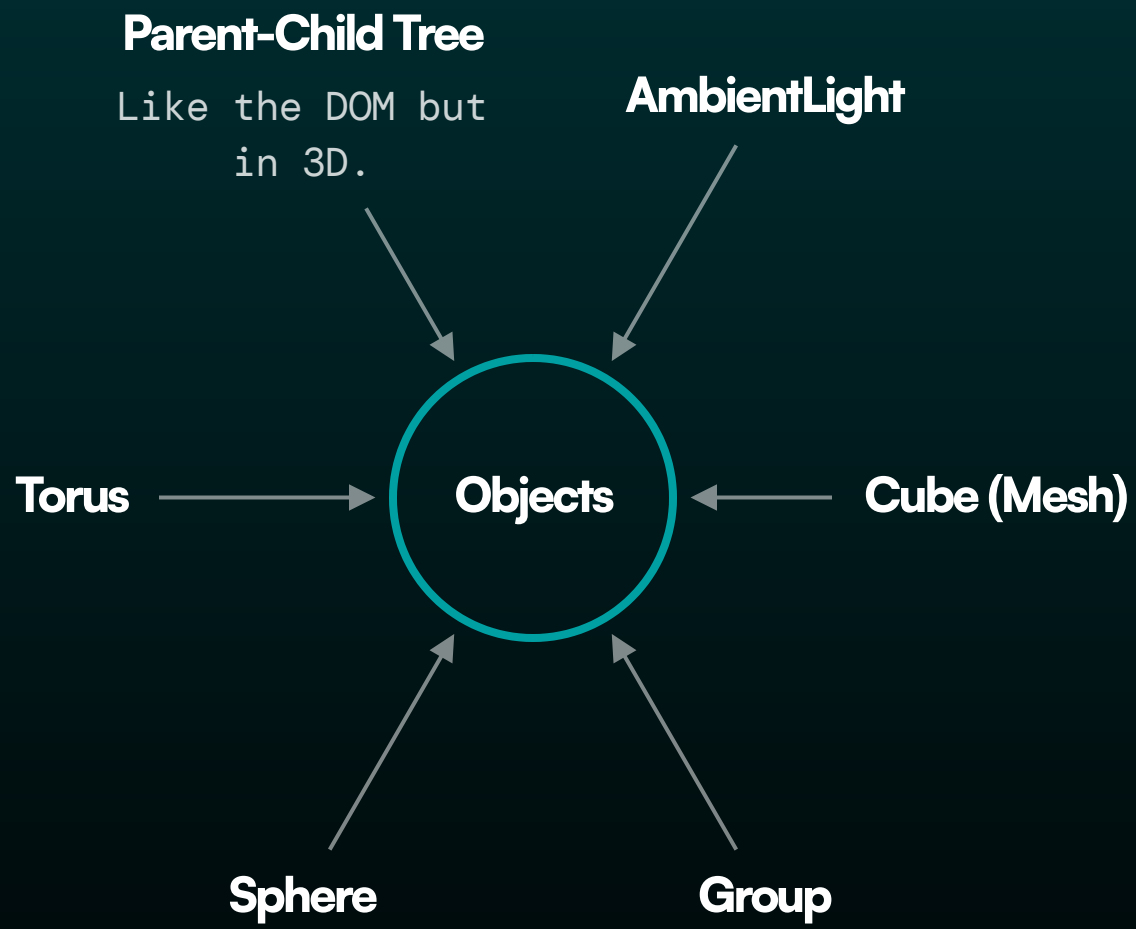
Shiny highlights, good for plastic/metal — medium.

4

Standard

Physically-based/PBR, most realistic look.

The Scene Graph



When a parent moves or rotates, all children move with it — great for complex animations.

Recap: Key Concepts

1. **Scene**
   Container for all 3D objects.
2. **Camera**
   Your viewpoint into the scene.
3. **Renderer**
   Draws the scene to the screen.
4. **Mesh**
   Geometry + material = visible object.
5. **Light**
   Illuminates objects, required for most materials.
6. **Render Loop**
   Continuously updates and redraws at ~60fps.
7. **Controls**
   Lets users orbit, pan, and zoom.
8. **Transforms**
   Position, rotation, and scale of any object.

Activity: Build an Interactive 3D Scene

Put everything you've learned into practice.

What you'll build:

A scene with multiple 3D objects (cubes, spheres, torus). Ambient + directional lighting with shadows. OrbitControls for interactive camera movement. Animated rotation on each object. A colored background and ground plane.

Skills you'll practice:

Scene setup, Mesh creation, Lighting, Camera controls, Animation

```
<script type="importmap">
  {
    "imports": {
      "three": "https://unpkg.com/three@latest/build/three.module.js",
      "three/addons/": "https://unpkg.com/three@latest/examples/jsm/"
    }
  }
</script>

<script type="module">
  import * as THREE from 'three';
  import { OrbitControls } from 'three/addons/controls/OrbitControls.js';

  // Your code starts here!
</script>
```

Resources & Next Steps

- **Learn More**

  threejs.org (official docs and examples) threejs-journey.com (comprehensive paid course) discoverthreejs.com (free book and tutorials) shadertoy.com (shader playground and inspiration)

- **Where to Go Next**

  Load 3D models (import GLTF/GLB files). Add textures (images mapped onto surfaces). Physics engines (Cannon.js or Rapier for realism). Post-processing (bloom, blur, color grading).

Now let's build something awesome!