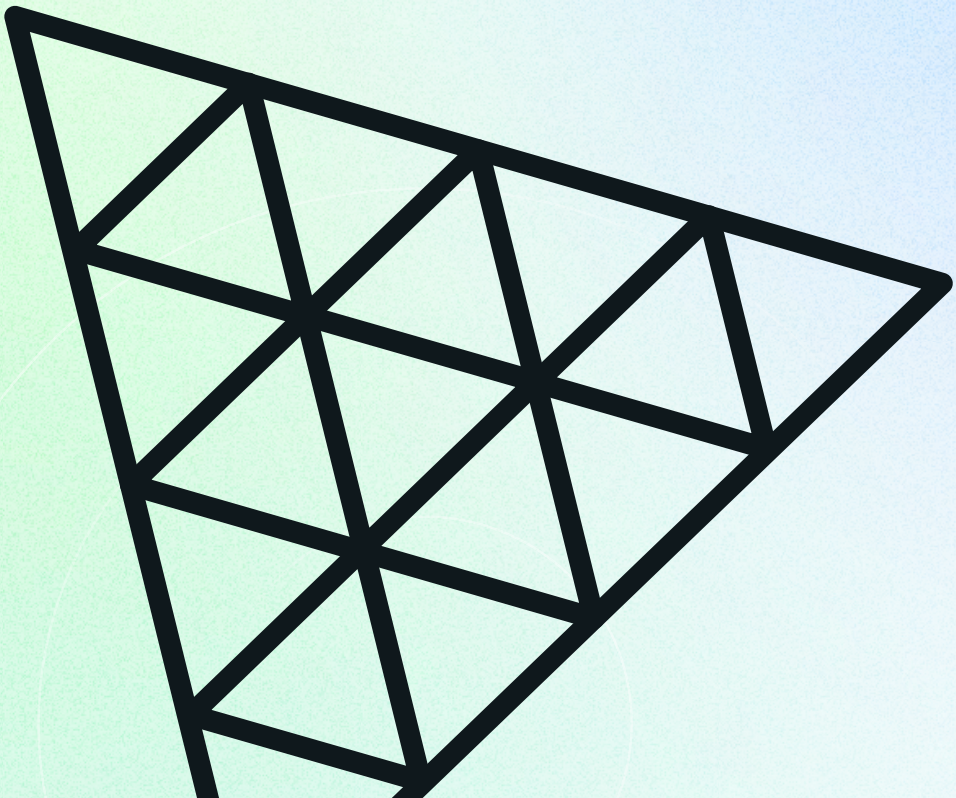
The Ultimate ThreeJS CheatSheet

Created by **JS Mastery**

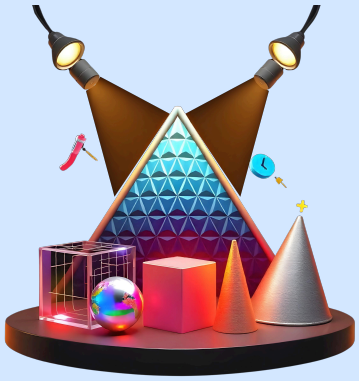
Visit ***jsmastery.pro*** for more



The Vanilla Three.js Course 🔥

If you're looking to deepen your understanding of Three.js and turn it into a career, I recently put together a course that focuses purely on Vanilla Three.js—from the basics to more advanced, interactive experiences.

It's designed to give you a solid foundation you can later apply in frameworks like React or Next.js.



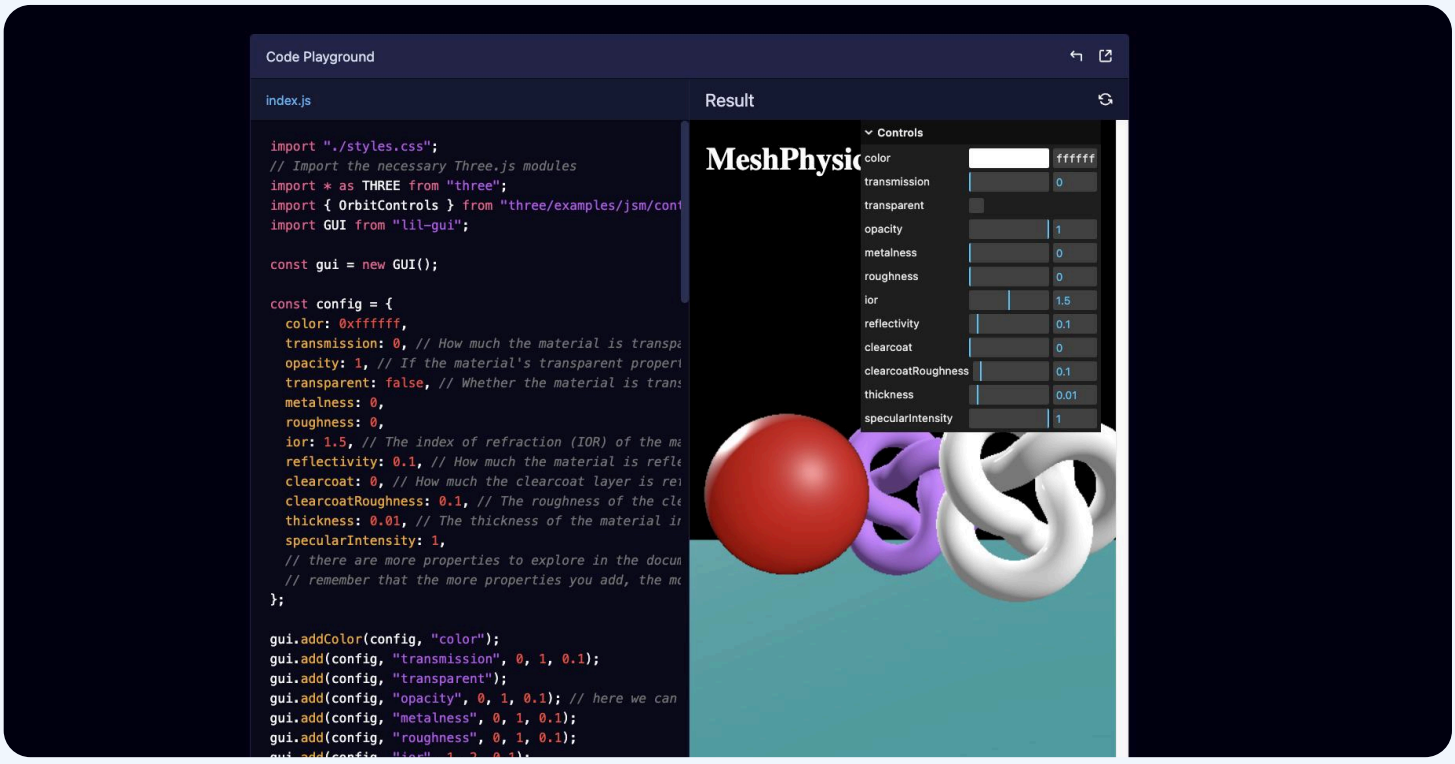
Vanilla Three.js Course

JavaScript-based WebGL engine that can run GPU-powered games & other graphics-powered apps straight from the browser.

Check out the course

The Vanilla Three.js Course 🔥

You'll get hands-on with live code playgrounds built right into the new platform, so you can experiment with lighting, animation, and different techniques in real time.



You'll see the impact of every change instantly and truly learn how Three.js works under the hood. If that sounds helpful, [check out the course now](#).

What’s in the guide?

Welcome to our Three.js cheat sheet! This guide provides a quick reference to the fundamental concepts and techniques used in Three.js, a powerful JavaScript library for creating 3D graphics and animations in the browser.

Whether you're a beginner or an experienced dev, this cheat sheet is designed to help you quickly find and use the most commonly used Three.js features.

In this cheat sheet, you'll find code snippets and examples for setting up a scene, creating and manipulating 3D objects, adding lighting and materials, using cameras, and more.

So, whether you're building a 3D game, a data visualization, or just experimenting with Three.js, this cheat sheet is a valuable resource to have. *Let’s start...*

What is Three.js

Three.js is a powerful, open-source library for creating stunning 3D visuals on the web. It provides developers with an easy-to-use set of tools and utilities for building interactive, animated 3D graphics that can be rendered in any modern web browser.

With Three.js, developers can create a wide range of 3D objects and scenes, including complex geometries, dynamic particle systems, and realistic lighting and shadow effects. Whether you're building a game, a data visualization, or an interactive product demo, Three.js offers the flexibility and power you need to bring your ideas to life.

One of the key benefits of Three.js is its broad compatibility with different web technologies and frameworks, including HTML5, CSS, and JavaScript.

What is Three.js

This means that you can easily integrate Three.js into your existing web projects or start from scratch with a new project, using familiar web development tools and techniques.

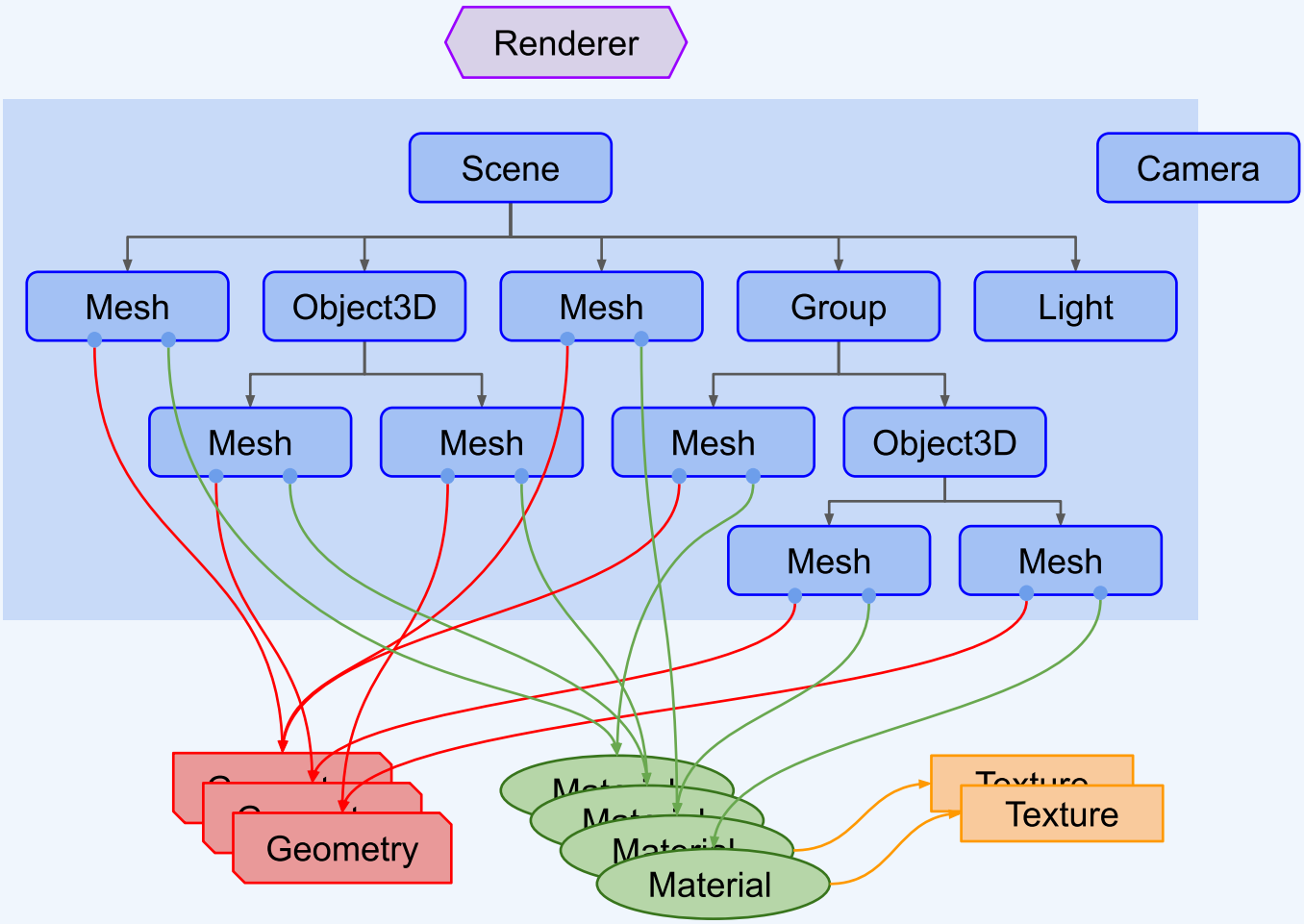
With a vibrant and supportive community of developers and designers, Three.js is constantly evolving and improving, making it an exciting and dynamic technology to explore and work with.

Whether you're a seasoned 3D programmer or just starting out, It offers a world of creative possibilities for building stunning, interactive web experiences.

Fundamentals

Before we get started 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 bunch of objects and connect them together. Here's a diagram that represents a small three.js app



Fundamentals

Things to notice about the diagram above.

There is a Renderer. This is arguably the 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 frustum of the camera as a 2D image to a canvas.

That was quite difficult, don't you think?

Let me break it down for you in simpler terms with a brief explanation or introduction of each term.

What is:

Renderer

A renderer is responsible for drawing the 3D scene onto the web page. In Three.js, the WebGLRenderer class is used to create a renderer. It uses WebGL, a graphics API based on OpenGL ES, to interact with the GPU and draw the scene onto the web page.

Geometry

Geometry defines the shape and structure of an object in Three.js.
It is made up of vertices (points in 3D space) and faces (triangles that connect the vertices). Three.js provides a number of built-in geometries, such as BoxGeometry, SphereGeometry, and PlaneGeometry, as well as the ability to create custom geometries.

What is:

Light

Lighting is used to simulate the way light interacts with objects in the scene. In Three.js, lights are used to illuminate the scene and create shadows. Three.js provides a number of built-in lights, such as AmbientLight, DirectionalLight, and PointLight, as well as the ability to create custom lights.

Camera

A camera determines the perspective and position of the viewer in the scene. The PerspectiveCamera and OrthographicCamera classes are used to create cameras. The PerspectiveCamera simulates a perspective view, while the OrthographicCamera simulates an isometric view.

What is:

Material

Material defines how an object appears in the scene, including its color, texture, and shading. The materials are applied to geometries to define their appearance.

It provides a number of built-in materials, such as MeshBasicMaterial, MeshLambertMaterial, and MeshPhongMaterial, as well as the ability to create custom materials using shaders.

Scene

A scene is the container that holds all of the objects, lights, and cameras in Three.js. In order to render anything in Three.js, you must create a scene and add objects, lights, and cameras to it.

What is:

Texture

A texture is an image applied to a material in Three.js.

Textures can be used to add detail to an object's surface, such as a wood grain pattern or a marble texture. In Three.js, textures are loaded using the TextureLoader class and applied to materials using the texture property.

Animation

Animation is the process of creating movement or change in a 3D scene over time. In Three.js, animation is achieved using the requestAnimationFrame method to update the position, rotation, or scale of objects in the scene.

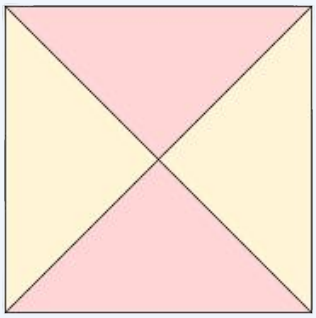
Summary

In summary, Three.js provides a number of powerful tools for creating & rendering 3D graphics on the web.

By understanding the basics of concepts such as renderer, geometry, texture, material, camera, light, scene, and animation, you can begin creating your own 3D scenes and exploring more advanced features in Three.js.

Triangles

Everything in the 3D world is made of triangles. A cube might seem like it's made up of squares, but in most 3D graphics systems (including Three.js), even squares are represented using triangles.



The complexity of the shapes comes from the sheer number of triangles involved. It's common for objects to be made up of millions of triangles, and in high-end visual effects or AAA movies, intricate scenes can involve billions.

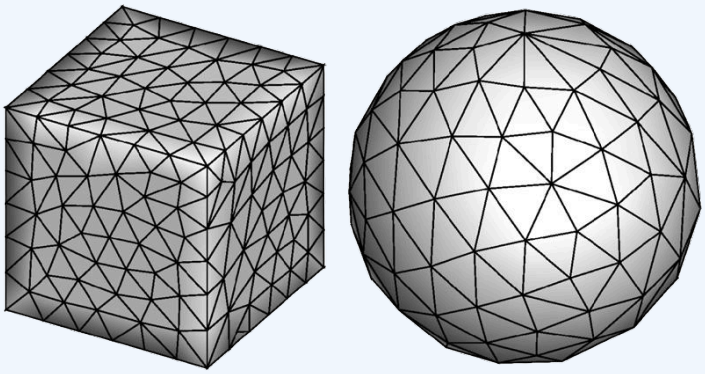
In some cases, the triangles are even smaller than a single pixel.

Triangles

But why triangles? Why not squares or circles?

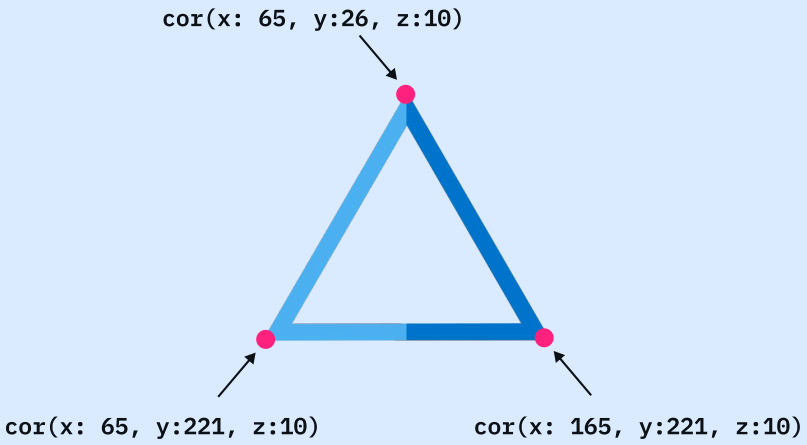
The reason is simple: triangles are the most efficient and unambiguous way to represent a surface in 3D space. They can be seamlessly connected edge to edge without gaps or overlaps, making them ideal for creating complex models.

While simple in form, triangles are versatile enough to represent any shape.



In 3D graphics, a triangle is defined by three points, each with an x, y, and z coordinate.

Triangles



The order of these points determines the orientation of the triangle, indicating which side is the outer, or front-facing, side of the surface. This orientation is often referred to as the surface's "normal."

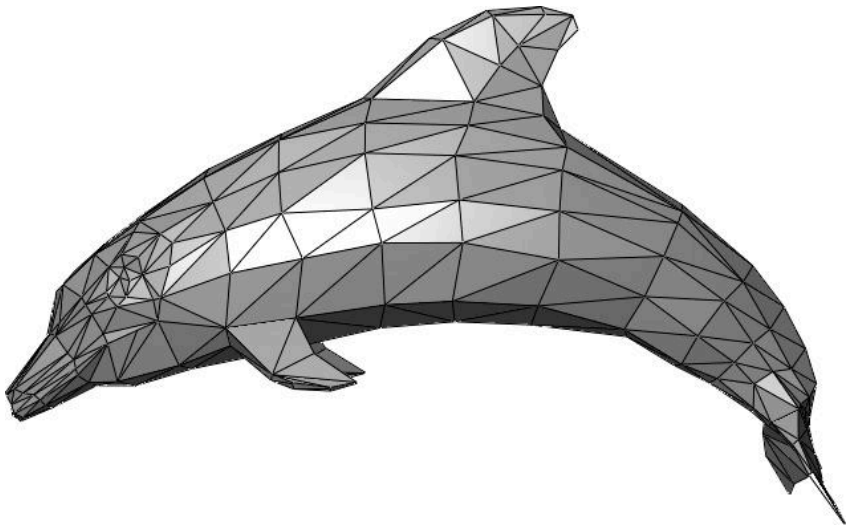
Lighting and shading effects in 3D models also rely on triangles. By calculating how light interacts with the surface of each triangle, the computer can create realistic lighting, shadows, and textures.

Triangles

This process is called "rasterization," where the 3D scene is converted into a 2D image that displays the final result on your screen.

Furthermore, modern graphics hardware, like GPUs, is optimized to handle triangles efficiently.

GPUs are designed to process large numbers of triangles in parallel, making them ideal for rendering real-time 3D environments, such as those found in video games or simulations.



Triangles

Triangles are also central to techniques like texture mapping, where a 2D image is wrapped around a 3D model, and bump mapping, which simulates small surface details.

In animation, the position and orientation of triangles can change frame by frame, creating movement. A process called "rigging" attaches a skeleton to the model, vertices of the triangles follow the movement of the skeleton to create realistic animation

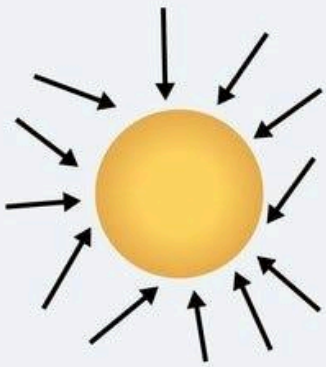
Ultimately, triangles are fundamental not just because of their geometric properties but also because of their efficiency in computer calculations. Without them, the rich, detailed 3D environments seen in today's movies, video games, and simulations would not be possible.

Types of Lights

AmbientLight

Ambient light represents overall environmental illumination that comes from all directions, simulating indirect or bounced light in a scene. It uniformly lights all objects in the scene without any specific direction.

Ambient light helps to brighten dark areas and adds global illumination to the scene. You can create an ambient light using the THREE.AmbientLight class.



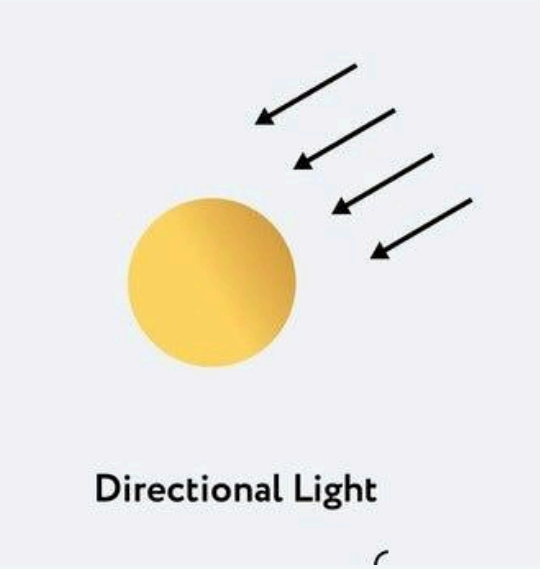
Ambient Light

Types of Lights

DirectionalLight

Directional light mimics the light emitted by a distant light source, such as the sun. It emits light rays in parallel directions, casting parallel shadows and creating realistic daylight simulations.

Directional lights have a position but no attenuation, & their intensity decreases with distance. You can create a this light using the THREE.DirectionalLight class.

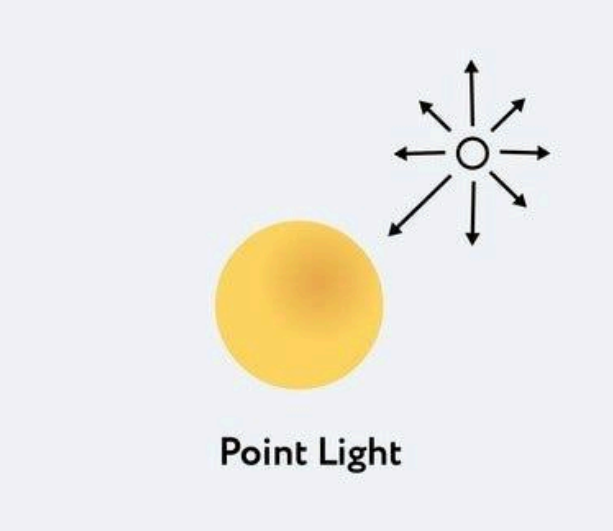


Types of Lights

PointLight

Point light represents a light source that emits light in all directions from a single point in space, creating omnidirectional illumination.

Point lights are useful for simulating light bulbs, lamps, or localized light sources. They cast shadows and attenuate with distance, becoming weaker as the distance from the light source increases. You can create a point light using the THREE.PointLight class.

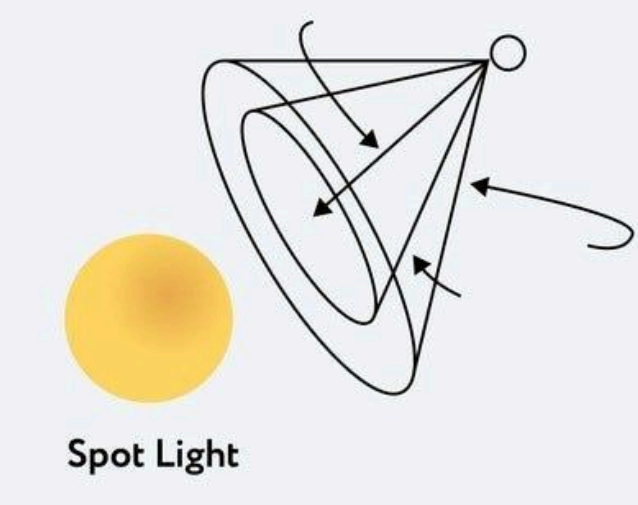


Types of Lights

SpotLight

Spot light simulates a focused beam of light emitted from a specific point in space in a cone-shaped direction. It casts shadows and can be used to create effects like flashlight beams or spotlight effects.

Spotlights have properties such as position, direction, and cone angle. You can create a spot light using the THREE.SpotLight class.

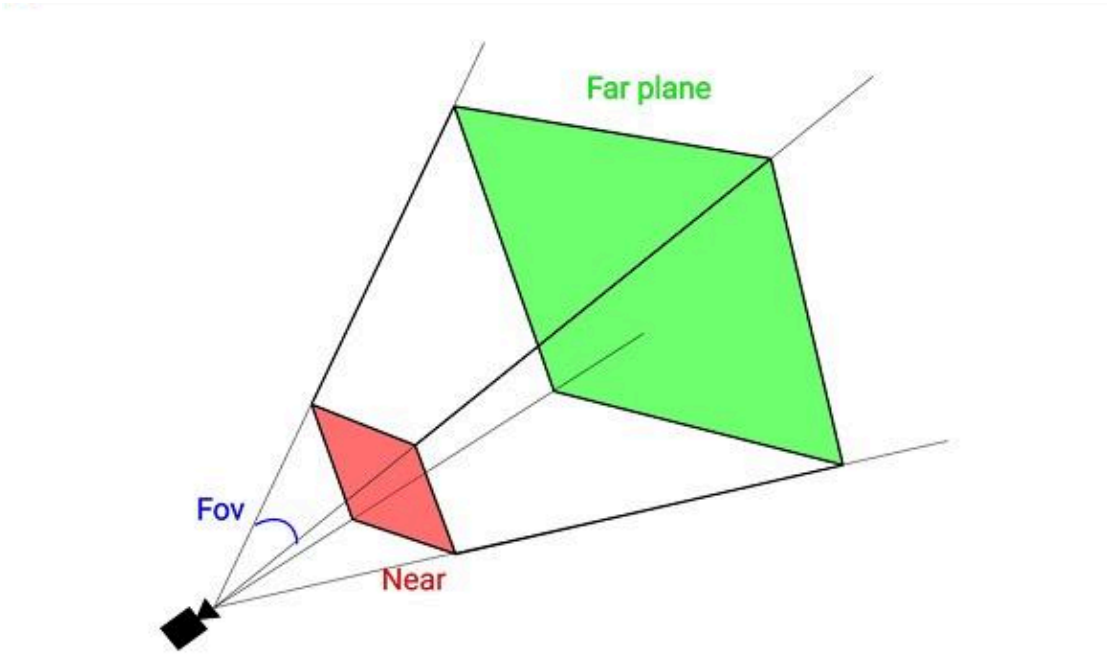


Types of Camera

PerspectiveCamera

The PerspectiveCamera is the most commonly used camera in Three.js. It mimics the way human vision works, with objects appearing smaller as they move further away from the camera.

It's ideal for creating scenes with depth & perspective, such as 3D environments or architectural renderings.

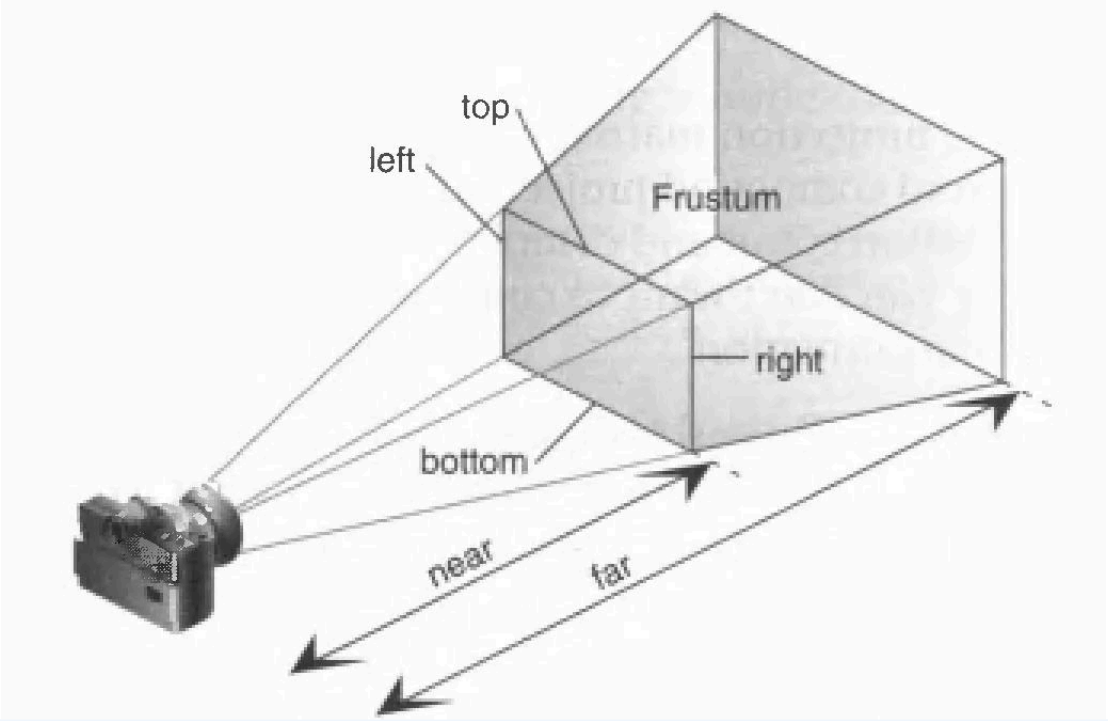


Types of Camera

OrthographicCamera

Unlike the PerspectiveCamera, the OrthographicCamera maintains the size of objects regardless of their distance from the camera.

It's useful for creating 2D-like scenes or when precise measurements are required, such as in CAD applications or 2D games.



Camera

Control over View Frustum

Both PerspectiveCamera and OrthographicCamera take parameters defining the view frustum, which determines what is visible in the scene.

These parameters include the field of view (fov), aspect ratio (aspect), near clipping plane (near), and far clipping plane (far).

Position and Orientation

You can set the position and orientation of the camera using its position and lookAt methods.

The lookAt method makes the camera point towards a specific point in the scene.

Camera

Camera Controls

Three.js provides camera control libraries like **OrbitControls** or **TrackballControls**, which allow users to interactively control the camera's position and orientation using mouse or touch inputs.

These controls make it easier to navigate and explore 3D scenes.

```
const controls = new THREE.OrbitControls(camera, renderer.domElement);
controls.enableDamping = true; // Smoothly interpolate camera movement
```

By understanding these aspects of cameras in Three.js, you can effectively set up and control the viewpoint of your 3D scenes to achieve the desired perspective and interactivity.

Geometries

In Three.js, geometries serve as the building blocks for creating 3D objects. Geometries define the shape and structure of objects in a 3D scene by specifying their vertices, faces, and other attributes.

Here's a breakdown of geometries in Three.js:

Basic Geometries

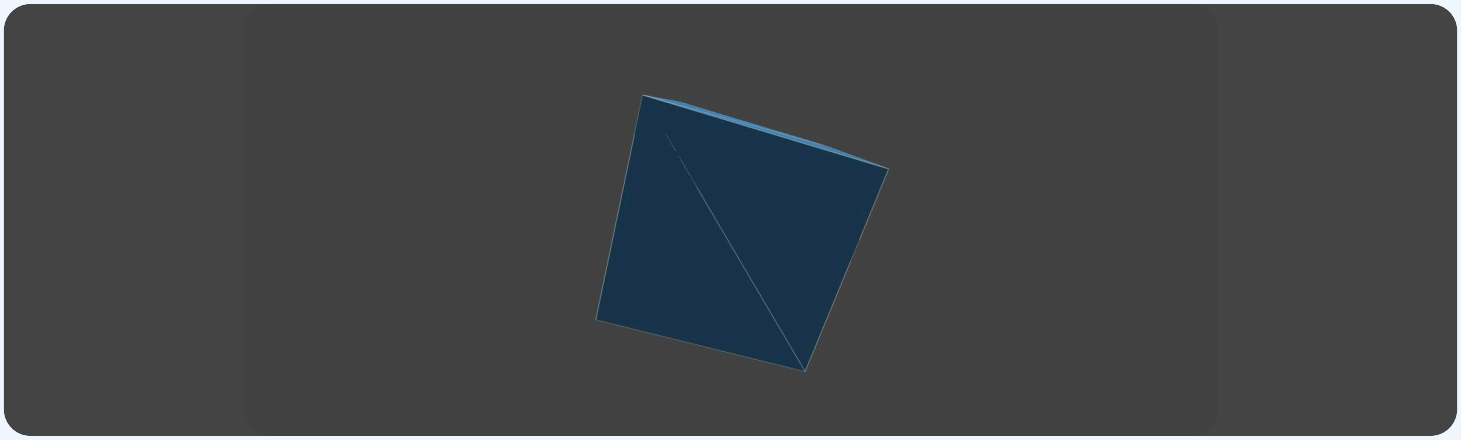
Three.js provides a set of built-in basic geometries that you can use to create common shapes such as cubes, spheres, cylinders, planes, and toruses.

These geometries are accessible through constructors like THREE.BoxGeometry, THREE.SphereGeometry, THREE.CylinderGeometry, THREE.PlaneGeometry, and THREE.TorusGeometry.

Geometries

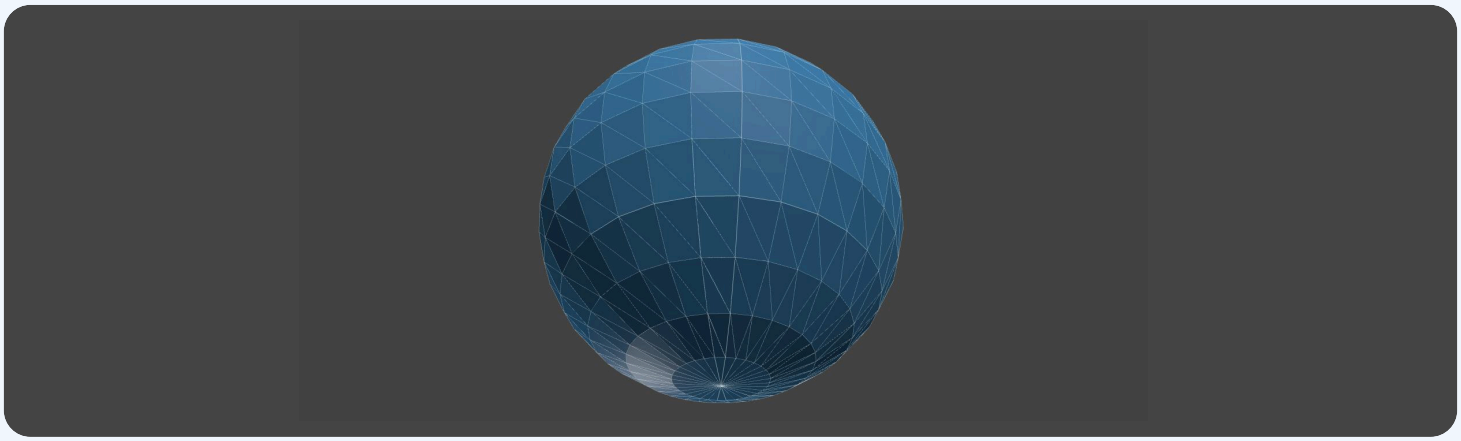
BoxGeometry

Creates a cube-shaped geometry with specified width, height, and depth.



SphereGeometry

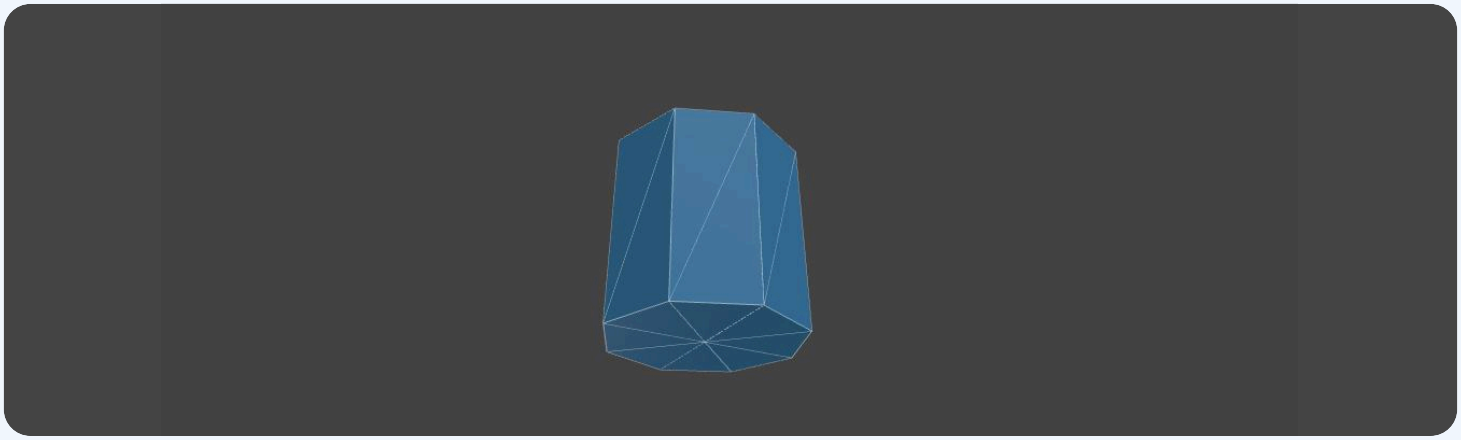
Generates a spherical geometry with a specified radius and number of segments for smoothness.



Geometries

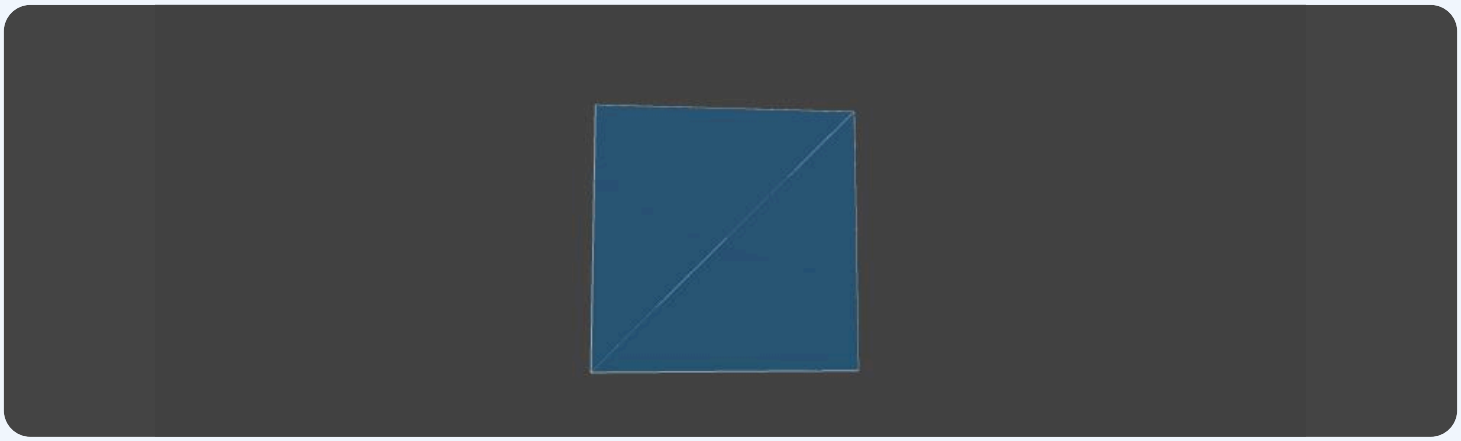
CylinderGeometry

Constructs a cylindrical geometry with specified radii, height, and segments for detail.



PlaneGeometry

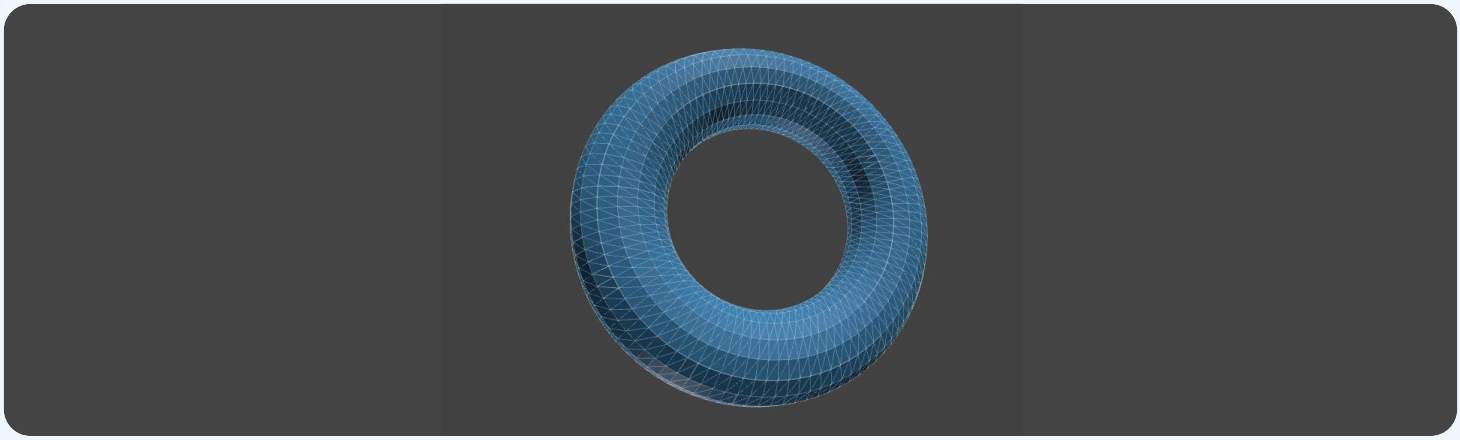
Represents a flat, rectangular plane with specified width and height.



Geometries

TorusGeometry

Creates a torus-shaped geometry (doughnut) with specified radius, tube radius, and segments for detail.



Custom Geometries

In addition to basic geometries, you can create custom geometries by defining the vertices and faces manually. This allows you to create more complex shapes and structures that are not provided by the built-in geometries.

Materials

In Three.js, a material defines the visual appearance of 3D objects in a scene. It determines how light interacts with the surface of an object, including aspects such as color, reflectivity, and transparency.

Materials play a crucial role in rendering realistic and visually appealing graphics by simulating various surface properties and lighting effects.

Essentially, a material defines how an object looks when rendered in a 3D scene.

Materials

MeshBasicMaterial

This material provides simple shading unaffected by lights, suitable for objects with uniform colors or basic textures. It's commonly used for wireframe objects or those requiring flat colors.

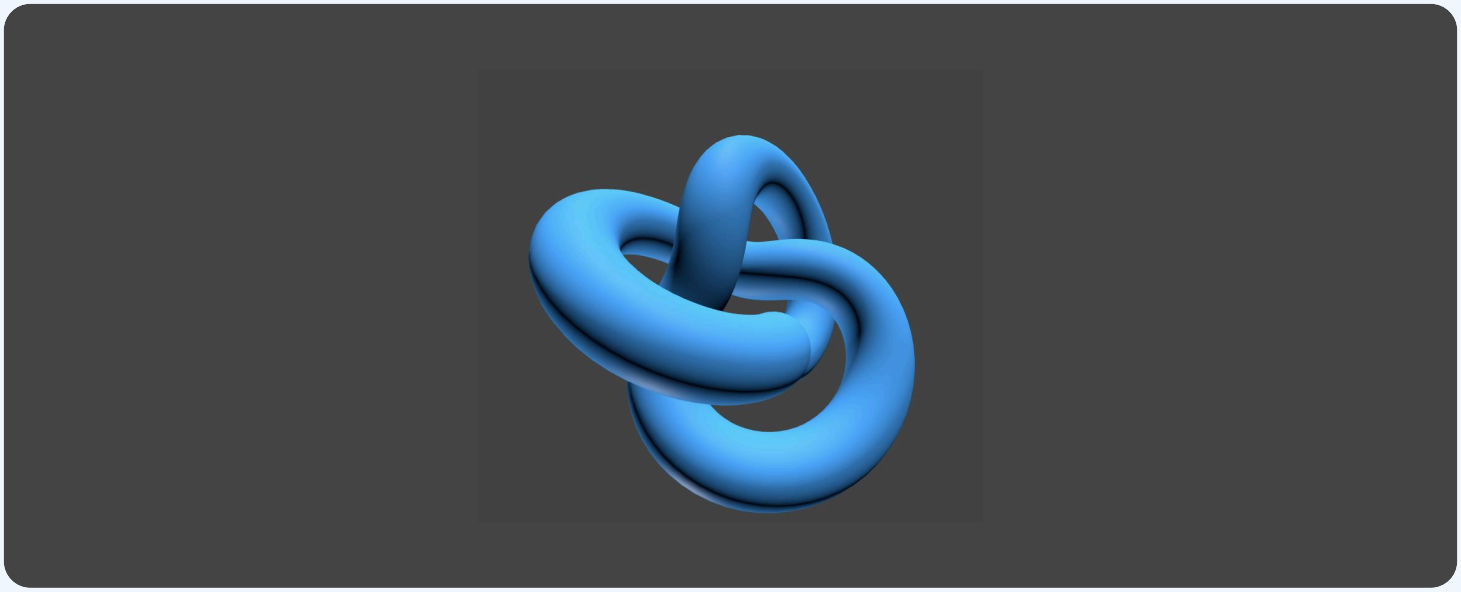
You might use MeshBasicMaterial for creating wireframe objects or objects with flat colors.

Materials

MeshPhongMaterial

Implements Phong shading, simulating smooth surfaces and highlights. Ideal for surfaces interacting with lights, such as shiny or reflective objects, like metals or plastics.

You could use MeshPhongMaterial for creating objects like metals, plastics, or glossy surfaces that exhibit specular highlights and reflections.

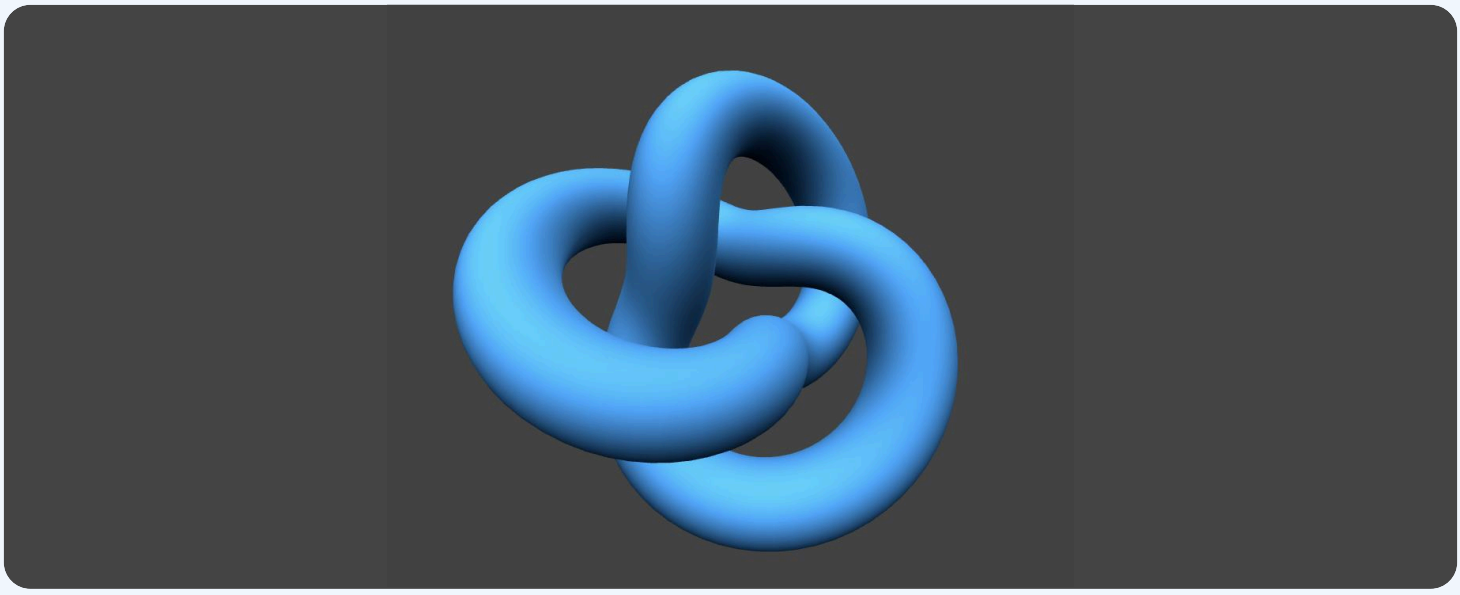


Materials

MeshStandardMaterial

Employs physically-based rendering (PBR) for realistic materials. Suitable for a wide range of materials including metals, plastics, and rough surfaces, providing a more realistic look.

You might use MeshStandardMaterial for creating objects in scenes where accurate material properties and lighting interactions are essential, such as architectural visualizations or product renderings.



React Three Fiber

React Three Fiber is a library that simplifies building 3D applications with Three.js and React. It provides a declarative and component-based approach to building and managing 3D scenes.

By using React Three Fiber, developers can use familiar React patterns to build and maintain complex 3D applications with less code and fewer bugs.

The library also provides performance optimizations that ensure smooth performance, even with complex scenes.

React Three Fiber's declarative and component-based approach can make it easier to reason about and manage the state and lifecycle of 3D objects in the scene, resulting in a more maintainable and scalable codebase.

React JS Setup

1. Create a new React project using a tool like Create React App.
2. Install the required dependencies using npm or yarn. These include **react-three-fiber**, three, and @types/three (if you are using TypeScript).
3. In your App.js or index.js file, import the necessary components from react-three-fiber and three.
4. Set up a basic Three.js scene using the Canvas component from react-three-fiber.
5. Add objects to the scene using Three.js primitives or custom 3D models.
6. Use react-three-fiber hooks like useFrame or useLoader to interact with the Three.js scene and update it as needed.

- *Here's an example of what your App.js file might look like →*

React JS Setup

```jsx
import React from 'react';
import { Canvas } from 'react-three-fiber';
import { Box } from 'three';

function App() {
  return (
    <Canvas>
      <ambientLight />
      <pointLight position={[10, 10, 10]} />
      <Box>
        <meshStandardMaterial attach="material" color="hotpink" />
      </Box>
    </Canvas>
  );
}

export default App;
```

This sets up a basic scene with ambient and point lighting, and a hot pink box in the center of the scene.

Cheat Sheet

1. Importing Three.js and React libraries

```
import * as THREE from 'three';
import React, { useRef, useEffect } from 'react';
import { Canvas } from 'react-three-fiber';
```

2. Rendering the Three.js component using the Canvas component from react-three-fiber

```
function App() {
  return (
    <Canvas>
      <MyComponent />
    </Canvas>
  );
}
```

Cheat Sheet

3. Creating a Three.js component using React hooks

```
function MyComponent() {
  const meshRef = useRef();
  useEffect(() => {
    meshRef.current.rotation.x += 0.01;
    meshRef.current.rotation.y += 0.01;
  });
  return (
    <mesh ref={meshRef}>
      <boxBufferGeometry />
      <meshStandardMaterial />
    </mesh>
  );
}
```

Cheat Sheet

4. Adding interactivity with the mouse using the useThree and useFrame hooks from react-three-fiber

```
import { useThree, useFrame } from 'react-three-fiber';

function MyComponent() {
  const meshRef = useRef();
  const { mouse } = useThree();
  useFrame(() => {
    meshRef.current.rotation.x = mouse.y * Math.PI;
    meshRef.current.rotation.y = mouse.x * Math.PI;
  });
  return (
    <mesh ref={meshRef}>
      <boxBufferGeometry />
      <meshStandardMaterial />
    </mesh>
  );
}
```

Cheat Sheet

5. Loading a 3D model using the useLoader hook from react-three-fiber

```
import { useLoader } from 'react-three-fiber';
import { GLTFLoader } from
'three/examples/jsm/loaders/GLTFLoader';

function MyComponent() {
  const gltf = useLoader(GLTFLoader, 'model.glb');
  return <primitive object={gltf.scene} />;
}
```

I hope this cheatsheet helps you get started with using Three.js in React!

Why react-three-fiber?

An example of a challenging aspect of building a 3D application with pure Three.js is managing the state and lifecycle of 3D objects in the scene.

Three.js provides a lot of low-level control over the 3D objects in the scene, such as the ability to add, remove, and update objects individually, but this can quickly become complex and hard to manage as the scene grows in complexity.

For example, imagine a 3D product website that allows users to customize and interact with a 3D model of a product.

The model may have multiple parts, each with its own material and texture, and the user may be able to change the color or texture of each part.

Why react-three-fiber?

Managing the state of all these objects in the scene, updating them in response to user input, and ensuring that they render correctly can be challenging.

Three.js provides a lot of low-level control over the 3D objects in the scene, such as the ability to add, remove, and update objects individually, but this can quickly become complex and hard to manage as the scene grows in complexity.

React Three Fiber was built to address this and other challenges by providing a declarative and component-based approach to building 3D apps.

With React Three Fiber, developers can use familiar React patterns to manage the state and lifecycle of 3D objects in the scene, making it easier to build and maintain complex 3D applications.

Code Example

Here's an example of how managing state in Three.js can become complex and how React Three Fiber simplifies the process:

Let's say we want to create a 3D cube that changes color when clicked on. Here's how we might do it in pure Three.js:

```
// create a scene, camera, and renderer
const scene = new THREE.Scene();
const camera = new THREE.PerspectiveCamera(75, window.innerWidth /
window.innerHeight, 0.1, 1000);
const renderer = new THREE.WebGLRenderer();
renderer.setSize(window.innerWidth, window.innerHeight);
document.body.appendChild(renderer.domElement);

// create a cube with a random color
const geometry = new THREE.BoxGeometry();
const material = new THREE.MeshBasicMaterial({ color: Math.random()
* 0xffffff });
```

Code Example

```
const material = new THREE.MeshBasicMaterial({ color: Math.random()
* 0xffffff });
const cube = new THREE.Mesh(geometry, material);
scene.add(cube);

// add a click event listener to the cube to change its color
cube.addEventListener('click', () => {
  cube.material.color.setHex(Math.random() * 0xffffff);
});

// render the scene
function animate() {
  requestAnimationFrame(animate);
  renderer.render(scene, camera);
}
animate();
```

This code creates a cube with a random color and adds a click event listener to it to change its color when clicked on.

Code Example

However, as the scene grows in complexity and more objects are added, managing the state and lifecycle of these objects can become challenging.

Here's how we could accomplish the same thing with React Three Fiber:

```
import { Canvas, useThree } from '@react-three/fiber';
import { useState } from 'react';
import * as THREE from 'three';

function Cube(props) {
  const { color, ...rest } = props;
  const [cubeColor, setCubeColor] = useState(color);

  const handleClick = () => {
    setCubeColor(Math.random() * 0xffffff);
  };

  return (
    <mesh {...rest} onClick={handleClick}>
      <boxGeometry />
```

Code Example

```
  const [cubeColor, setCubeColor] = useState(color);

  const handleClick = () => {
    setCubeColor(Math.random() * 0xffffff);
  };

  return (
    <mesh {...rest} onClick={handleClick}>
      <boxGeometry />
      <meshBasicMaterial color={cubeColor} />
    </mesh>
  );
}

function Scene() {
  return (
    <Canvas>
      <Cube position={[-1, 0, 0]} color={Math.random() * 0xffffff} />
      <Cube position={[1, 0, 0]} color={Math.random() * 0xffffff} />
    </Canvas>
  );
}

function App() {
  return <Scene />;
}
```

Code Example

In this code, we define a Cube component that manages its own state using the useState hook.

The Cube component renders a Three.js mesh and material, and responds to click events to change its own color.

We then use the Cube component twice in the Scene component to create two cubes with different positions and random colors.

By using React Three Fiber's declarative and component-based approach, we can easily manage the state & lifecycle of multiple 3D objects in the scene

RTF Advantages over Pure

Declarative approach:

React Three Fiber uses a declarative approach to define 3D scenes, which can make it easier to reason about and maintain the codebase over time. This is because the code describes what should be displayed, rather than how it should be displayed.

Familiar syntax:

React Three Fiber uses a syntax that is similar to that of React, which makes it easier for developers who are familiar with React to learn and use Three.js.

Component-based architecture:

It uses React's component-based architecture, which can make it easier to organize and manage the state and lifecycle of 3D objects in a scene.

RTF Advantages over Pure

Optimized for React:

React Three Fiber is optimized for React's rendering engine and lifecycle methods, which means that it can be used in large, complex applications without any performance issues.

Debugging tools:

It provides debugging tools such as a helpful error message and a built-in inspector, which can make it easier to identify and fix issues in the 3D scene.

TypeScript support:

React Three Fiber has excellent TypeScript support, which means that you can write type-safe Three.js code in React.

RTF Advantages over Pure

Hooks-based API:

React Three Fiber's hooks-based API makes it easy to manipulate Three.js objects, animate them, and add event listeners to them.

Drei library:

Drei is a collection of useful Three.js components and helpers that are built on top of React Three Fiber. Drei components make it easy to create things like 3D text, post-processing effects, and particle systems.

Overall, React Three Fiber offers a more streamlined and efficient way to work with Three.js in React, and is a great choice for developers who want to build 3D applications in React.

RTF Cheat Sheet

`<Canvas>`

A component that creates a Three.js scene and mounts it to the DOM.

`<mesh>`

A component that wraps a 3D object and its material.

`<primitive>`

A component that creates a Three.js primitive geometry, like a box or sphere.

`<group>`

A component that allows you to group objects together for easier manipulation.

RTF Cheat Sheet

`useLoader()`

A hook that loads external assets, like textures or 3D models.

`useFrame()`

A hook that runs on every frame of the animation loop, allowing you to update Three.js objects over time.

`useGraph()`

Convenience hook which creates a memoized, named object/material collection from any Object 3D.

RTF Cheat Sheet

<OrbitControls>

A component that adds mouse and touch controls to your Three.js scene.

<perspectiveCamera>

A component that defines a perspective camera for your scene.

Here's an example of using some of these features together to create a rotating cube:

```
import React, { useRef } from 'react';
import { Canvas, useFrame } from 'react-three-fiber';
import { Box } from 'drei';

function RotatingCube() {
  const meshRef = useRef();
```

RTF Cheat Sheet

```
  useFrame(() => {
    meshRef.current.rotation.x += 0.01;
    meshRef.current.rotation.y += 0.01;
  });

  return (
    <Box ref={meshRef}>
      <meshStandardMaterial attach="material" color="hotpink" />
    </Box>
  );
}

function App() {
  return (
    <Canvas>
      <ambientLight />
      <pointLight position={[10, 10, 10]} />
      <RotatingCube />
    </Canvas>
  );
}

export default App;
```

RTF Cheat Sheet

This creates a **<Canvas>** with ambient and point lighting, and a **<RotatingCube>** component that uses **useFrame()** to rotate the cube on every frame.

The cube is created using the **<Box>** primitive, and its reference is stored in meshRef. Finally, the **<Box>** is wrapped in a **<mesh>** with a hot pink material.

React Three Drei?

React Three Drei is a collection of useful helper components and hooks for building 3D applications with React Three Fiber.

It is built on top of React Three Fiber and provides a higher-level API for common 3D tasks such as camera controls, lighting, and loading 3D models.

React Three Drei offers a variety of pre-built components such as OrbitControls, Sky, Html, Model, shaderMaterial, Reflector and Bloom that can be easily used in a React Three Fiber scene.

These components can help streamline the process of building 3D applications by abstracting away low-level Three.js code and providing a simpler and more intuitive interface.

React Three Drei?

React Three Drei also includes a number of hooks, such as useTexture, useGLTF, and useAnimations, that make it easier to work with assets in a 3D scene.

Overall, React Three Drei can help developers save time and effort when building 3D applications by providing pre-built components and hooks that abstract away low-level Three.js code and simplify common 3D tasks.

Here are some code examples of React Three Drei components and hooks →

RTD Code Example

1. OrbitControls component:

```
import React, { useRef } from 'react'
import { Canvas } from 'react-three-fiber'
import { OrbitControls } from  '@react-three/drei'

function App() {
  const cameraRef = useRef()

  return (
    <Canvas>
      <OrbitControls ref={cameraRef} />
      <mesh>
        <boxBufferGeometry />
        <meshStandardMaterial />
      </mesh>
    </Canvas>
  )
}

export default App
```

RTD Code Example

This code creates a simple 3D scene with a box mesh and an OrbitControls component for controlling the camera position and rotation.

The OrbitControls component is imported from drei (short for "three"), which is a part of React Three Drei.

RTD Code Example

2. useTexture hook:

```jsx
import React from 'react'
import { useTexture } from '@react-three/drei'

function App() {
  const texture = useTexture('/path/to/texture.jpg')

  return (
    <mesh>
      <boxBufferGeometry />
      <meshStandardMaterial map={texture} />
    </mesh>
  )
}

export default App
```

This code uses the useTexture hook from React Three Drei to load a texture image and apply it to a mesh material.

RTD Code Example

The useTexture hook returns a Texture object, which can be used as the map property of a mesh material.

3. Html component:

```
import React from 'react'
import { Html } from '@react-three/drei'

function App() {
  return (
    <Html>
      <div style={{ color: 'white' }}>Hello, world!</div>
    </Html>
  )
}

export default App
```

RTD Code Example

This code uses the Html component from React Three Drei to render a HTML element in the 3D scene.

The Html component creates a separate HTML layer that is rendered on top of the 3D scene, allowing developers to easily create interactive and dynamic user interfaces in their 3D applications.

These are just a few examples of the many components and hooks provided by **React Three Drei**.

By using these pre-built components and hooks, developers can simplify common 3D tasks and create more complex 3D applications with less code.

Advantages

Simplified API:

React Three Drei provides a higher-level API for common 3D tasks, which can make it easier and faster to build 3D scenes compared to writing raw Three.js code.

Component-based architecture:

Drei uses React's component-based architecture, which makes it easier to organize and manage the state and lifecycle of 3D objects in a scene.

Performance optimizations:

React Three Drei includes performance optimizations such as automatic batching of meshes and pre-loading of assets, which can help improve the overall performance of a 3D application.

Advantages

Developer-friendly:

React Three Drei can make it easier for developers to work with Three.js by providing a more familiar and developer-friendly syntax, especially for those who are already familiar with React.

Code reusability:

React Three Drei over pure Three.js is that it can help reduce code complexity & increase code reusability.

Overall, React Three Drei can make it easier and faster to build 3D applications with Three.js by providing a simpler and more intuitive interface, performance optimizations, and a component-based architecture.

RTD Cheat Sheet

`<Canvas>`

The main component that renders a Three.js scene in a React app.

`<OrbitControls>`

A pre-built camera controller that allows users to pan, zoom, and orbit around the 3D scene

`<Html>`

A component that allows you to render HTML elements in a Three.js scene.

`<Text>`

A component that allows you to render 3D text in a Three.js scene.

`<Line>`

A component that creates a 3D line mesh.

RTD Cheat Sheet

<Box>

A component that creates a 3D box mesh.

<Sphere>

A component that creates a 3D sphere mesh.

<Plane>

A component that creates a 3D plane mesh.

useTexture

A hook that loads a texture and returns a Three.js texture object.

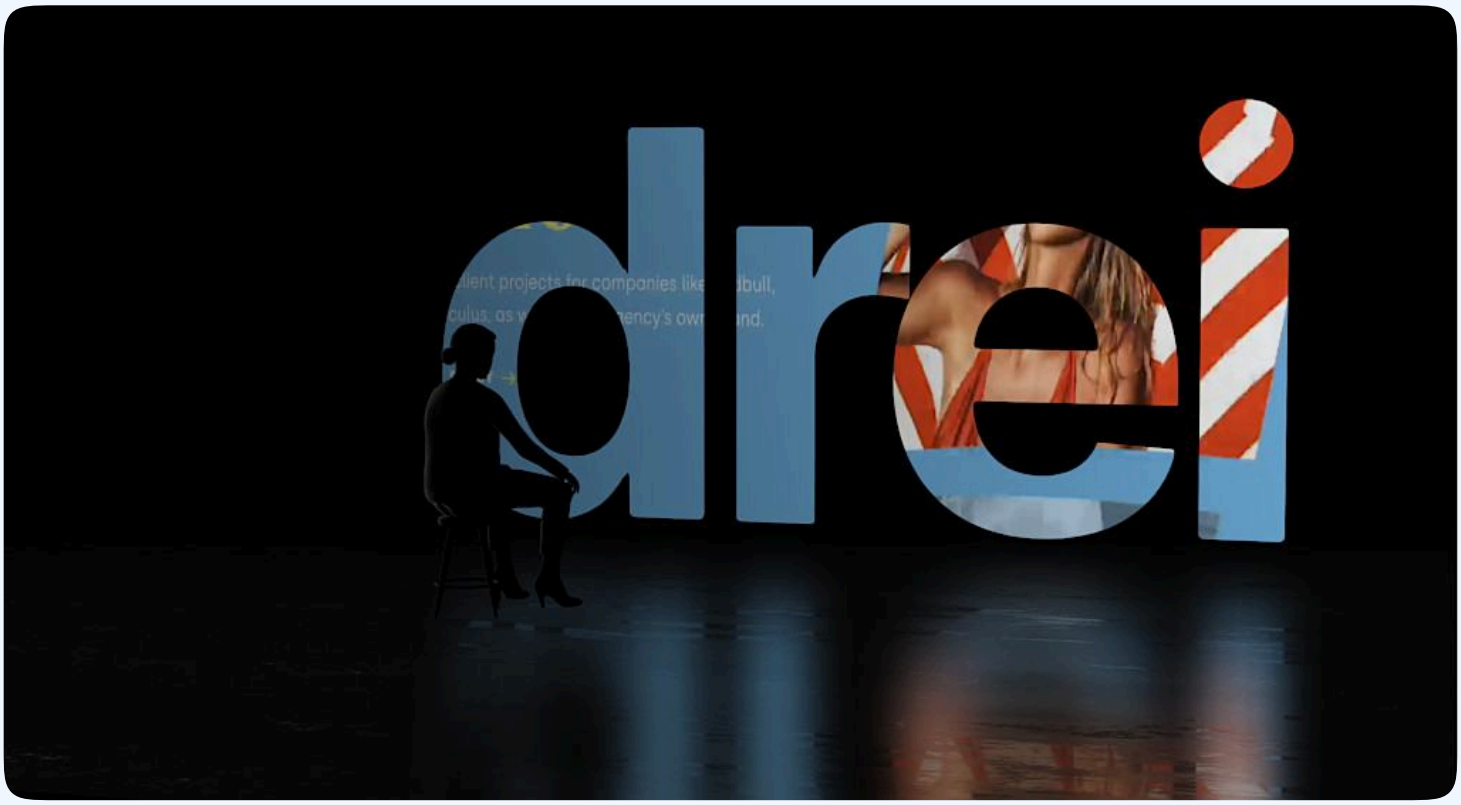
useGLTF

A hook that loads a GLTF model and returns a Three.js object.

RTD Cheat Sheet

These are just a few examples of the many components and hooks available in React Three Drei.

The library provides a wide range of pre-built components and hooks that can simplify common 3D tasks and save developers time and effort.



Beginner Project Ideas

3D Cube:

Create a 3D cube using Three.js and experiment with different materials, textures, and lighting effects to make it visually appealing.

Solar System Model:

Build a model of the solar system using Three.js and explore the different planets, moons, and other celestial bodies in a 3D environment.

3D Text:

Create 3D text using Three.js and experiment with different fonts, sizes, and colors to make it visually interesting.

Beginner Project Ideas

Interactive Gallery:

Build an interactive gallery using Three.js that allows users to navigate through different 3D objects or images.

Particle Effects:

Create particle effects using Three.js and experiment with different settings to make visually appealing effects, such as explosions, fire, or rain.

3D Terrain:

Create a 3D terrain using Three.js and experiment with different textures, heights, and shapes to create a dynamic landscape.

Beginner Project Ideas

3D Maze:

Build a 3D maze using Three.js and add interactive elements such as obstacles and rewards to make it more challenging and engaging.

3D Card Flip:

Create a simple card-flipping animation using Three.js to showcase your understanding of basic 3D transformations and animations.

3D Interactive Dice:

Build a 3D dice that users can roll and interact with using Three.js, using basic geometry and materials to create a realistic effect.

Topics you should explore

Once you have a good understanding of the fundamentals of Three.js, there are several other topics that you can explore to further develop your skills and create more complex 3D applications. Here are some examples:

Animation

Learn how to use Three.js to create animations that change the position, rotation, and scale of objects over time.

Physics

Learn how to use a physics engine such as Cannon.js or Ammo.js with Three.js to create realistic simulations of objects that interact with each other based on real-world physics principles.

Topics you should explore

Shaders

Learn how to use custom shaders to create complex visual effects and apply advanced lighting and shading techniques to your 3D objects.

Textures

Learn how to use textures to add more detail and visual interest to your 3D objects, including applying images, videos, and other media to surfaces.

Optimization

Learn how to optimize your Three.js applications for performance by reducing the number of objects, minimizing the size of textures and meshes, and using techniques such as culling and LOD (Level of Detail) to improve rendering speed.

Topics you should explore

Interactivity

Learn how to use Three.js to create interactive 3D applications that allow users to interact with objects using mouse or touch events, or even using VR/AR devices.

These are just a few examples of the topics that you can explore after learning the basics of Three.js.

The key is to keep practicing and experimenting with different techniques to improve your skills and create more advanced 3D applications.

Cool Project Ideas

Interactive 3D Gallery

Build a virtual art gallery where users can walk around and explore various artworks in a 3D environment. Add interactivity such as information about each artwork upon clicking.

3D Data Visualization

Create immersive visualizations of complex data sets using Three.js. This could include anything from financial data to scientific data, presented in a visually appealing and interactive 3D format.

Virtual Reality Tours

Develop virtual tours of real-world locations or fictional environments using Three.js and WebVR. Users can explore these environments using VR headsets or simply their web browsers.

Cool Project Ideas

3D Games

Build simple to complex 3D games such as puzzles, platformers, or first-person shooters using Three.js. You can incorporate physics engines like Ammo.js for realistic interactions.

Particle Effects

Experiment with particle systems in Three.js to create mesmerizing effects like fire, smoke, water, or dynamic particle-based animations.

Educational Simulations

Develop interactive educational simulations for subjects like physics, chemistry, or biology. For eg, you could create a simulation demonstrating gravitational forces or molecular structures.

Cool Project Ideas

Music Visualizations

Generate dynamic visualizations that respond to music or sound input. This could involve creating abstract visualizations that pulse and change based on the rhythm and intensity of the audio.

Interactive Storytelling

Combine 3D graphics with storytelling elements to create immersive interactive narratives. Users can navigate through the story world, interact with characters, & make choices that affect the outcome.

Educational Simulations

Develop interactive educational simulations for subjects like physics, chemistry, or biology. For eg, you could create a simulation demonstrating gravitational forces or molecular structures.

Cool Project Ideas

Space Exploration Simulation

Develop a 3D simulation of outer space where users can explore planets, moons, and other celestial bodies. Incorporate real astronomical data for accuracy and educational value.

Weather Visualization

Create a dynamic 3D visualization of weather patterns and phenomena such as clouds, rainfall, and storms. Users can interact with the simulation to understand weather dynamics.

3D Fractal Explorer

Implement a fractal explorer that allows users to navigate and interact with 3D fractal structures in real-time. Fractals are visually stunning and provide endless exploration possibilities.

Cool Project Ideas

Medical Visualization

Develop interactive 3D visualizations of human anatomy or medical procedures. This could be used for educational purposes or to simulate surgical scenarios for training purposes.

Time Travel Exploration

Create a time-traveling experience where users can explore historical or futuristic environments. They can witness historical events or futuristic cities in 3D.

City Building Simulation

Design a city-building simulation where users can construct and manage their own virtual cities. Incorporate elements like zoning, infrastructure development, and economic simulation.

Cool Project Ideas

Robotic Arm Simulator

Build a simulator for controlling a robotic arm in 3D space. Users can manipulate the arm's movements and interact with objects in the environment, simulating real-world robotics applications.

Fantasy World Builder

Let your imagination run wild by creating a fantasy world with mythical creatures, magical landscapes, and epic battles. Users can explore this world and uncover its secrets.

Language Learning Game

Develop a game that helps users learn new languages through immersive 3D environments. Players can navigate through different scenarios and practice language skills in context.

Cool Project Ideas

Robotic Arm Simulator

Develop interactive infographics that use GSAP to animate data visualizations, charts, and diagrams. Add user interactions such as hover effects or click-triggered animations to enhance engagement.

Animated Product Showcase

Design an animated product showcase or portfolio website using GSAP to highlight key features and details of products or projects. Incorporate smooth transitions & interactive elements to engage visitors.

Scroll-Based Website Animations

Create a website with scroll-based animations powered by GSAP. Implement effects like parallax scrolling, fade-ins, & transitions triggered as the user scrolls down the page for a dynamic experience.

Cool Project Ideas

Character Animation

Explore character animation using GSAP to bring characters to life in web-based applications or games. Create animations for character movements, expressions, and interactions with the environment.

Interactive Map with Animations

Build an interactive map with GSAP-powered animations to visualize data, highlight locations, and provide dynamic navigation features. Add animations for map markers, tooltips, and route paths.

Animated Logo Design

Design animated logos or brand identities using GSAP to create memorable and visually engaging brand experiences. Experiment with different animation techniques such as morphing, scaling, and rotation.

Websites to get 3D Model

Sketchfab

https://sketchfab.com/

Poly Pizza

https://poly.pizza/

PMNDRS Market

https://market.pmnd.rs/

Filer

https://www.filer.dev/3d-models/1

Websites to get 3D Model

Clara

https://clara.io/library

Pikbest

pikbest.com/decors-models/3d-models

CG Trader

https://www.cgtrader.com/3d-models/

Grabcad

https://grabcad.com/library

Websites to get 3D Model

Autodesk Community

https://autodesk.com/community/gallery

Freepik

https://www.freepik.com/3d-models

RenderCrate

https://rendercrate.com/

Free 3D

https://free3d.com/

Websites to get 3D Model

3dsky

https://3dsky.org/3dmodels

Thingiverse

https://www.thingiverse.com/

Cults

https://cults3d.com/

Turbosquid

https://www.turbosquid.com/

Websites to get 3D Model

Design Connected

https://designconnected.com/Freebies/

Archive 3d

https://archive3d.net/

3d Export

https://3dexport.com/free-3d-models

Cadnav

https://www.cadnav.com/3d-models/

Websites to get 3D Model

All 3d Free

https://www.all3dfree.net/

3DXO

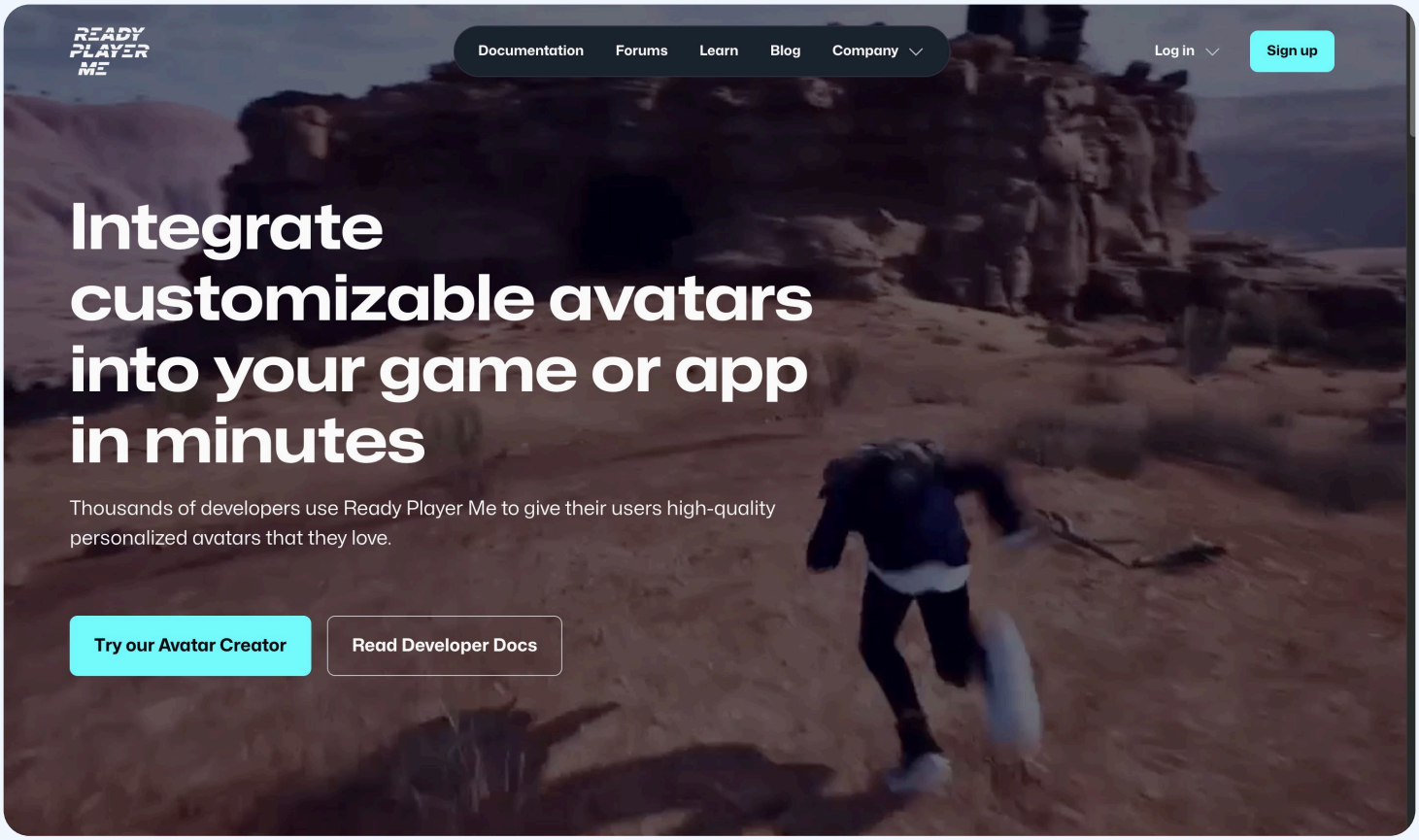
https://www.3dxo.com/textures

Sketchup Texture

https://sketchuptextureclub.com/

Ready Player Me

Ready Player Me is a platform that allows users to create highly customizable 3D avatars that can be used across various virtual worlds, games, and apps. It's like a universal digital identity that represents you in the metaverse.



These avatars can be integrated into websites, VR/AR experiences, social apps, games, and more.

Ready Player Me

The idea is simple: make it easier for anyone to have a consistent digital persona, no matter where they play, socialize, or work in virtual environments.

Why Use Ready Player Me?

1. **Cross-Platform Compatibility**

   Ready Player Me avatars work across hundreds of virtual worlds and apps, making it easier to have a consistent identity.

2. **Customizable Avatars**

   You can create a unique avatar that reflects your style, with options to customize everything from facial features to clothing.

Ready Player Me

3. **Quick and Easy Setup**

   The platform is user-friendly, allowing you to generate a 3D avatar in minutes.

4. **Free to Use**

   Ready Player Me offers a range of customization options at no cost, making it accessible to everyone.

How to use?

Step 1: Visit the Website

Go to the Ready Player Me website and create a new account

Step 2: Go to Avatar Creator

On the main page, you'll find a button that says, "Try our Avatar Creator." Click on it to get started.

Or visit this page.

How to use?

Step 3: Choose Your Avatar Type

You can create either a full-body or half-body avatar. Full-body avatars are great for VRChat, Mozilla Hubs, or any app that supports complete avatars.

Half-body avatars are designed for apps and games that only need a head and shoulders view.

How to use?

Step 4: Customize Your Avatar

- **Take a Selfie or Choose a Base Model**

  For a personalized touch, take a selfie and let the platform create a base avatar for you, or pick from a set of premade base models.

How to use?

- **Adjust Facial Features**

  Modify your avatar's face, hair, skin tone, and more. The editor is intuitive, with sliders for adjusting every detail.

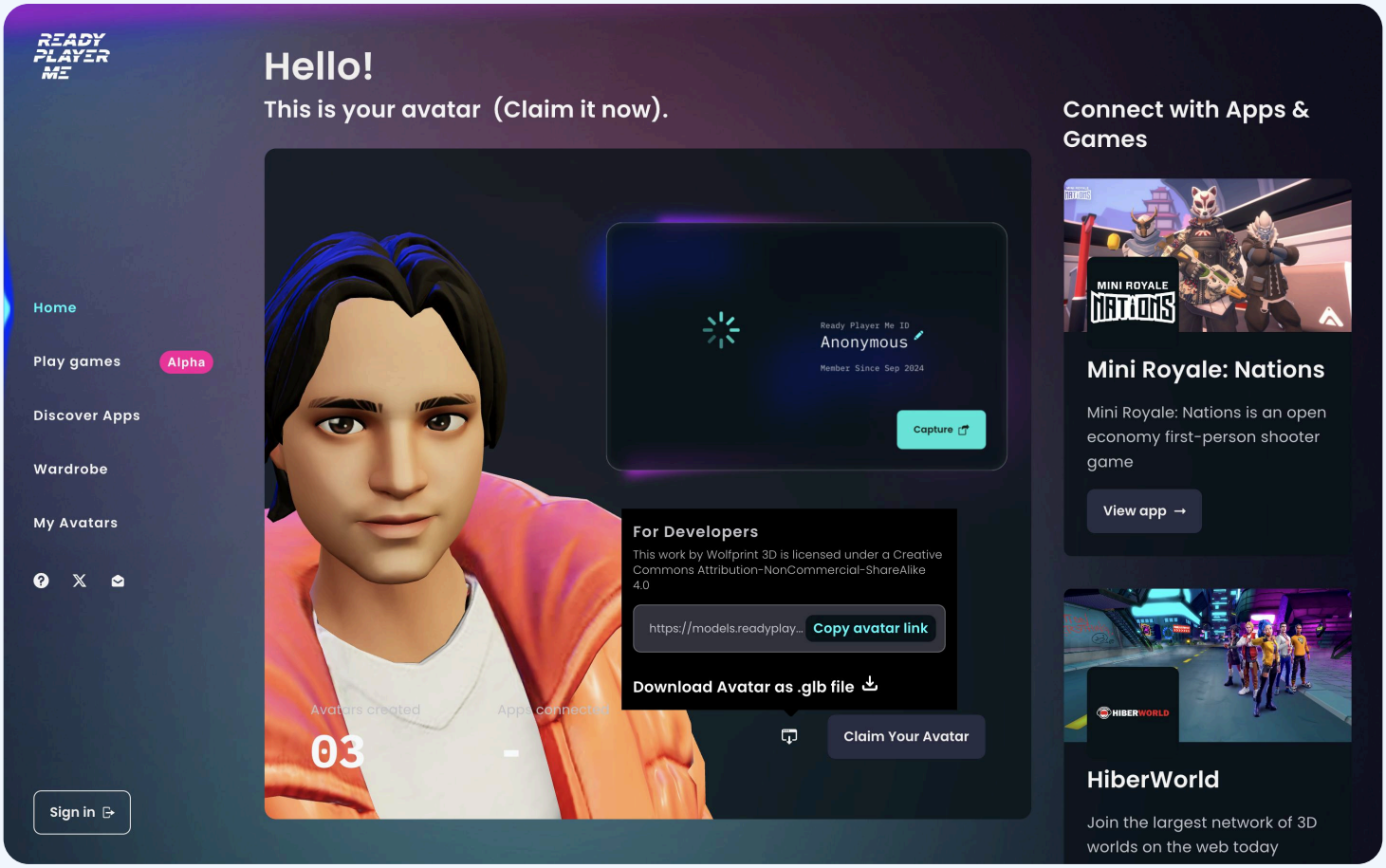
- **Choose Your Outfit**

  Select from a wide range of clothes & accessories to make your avatar look exactly how you want.

How to use?

Step 5: Export and Use Your Avatar

Once you're happy with your avatar, you can export it to use across different platforms. Ready Player Me supports numerous virtual worlds, games, and social platforms, including VRChat, Mozilla Hubs, LIV, MeetinVR, and more.



How to use?

- **Export Formats**

  You can download your avatar in GLB format

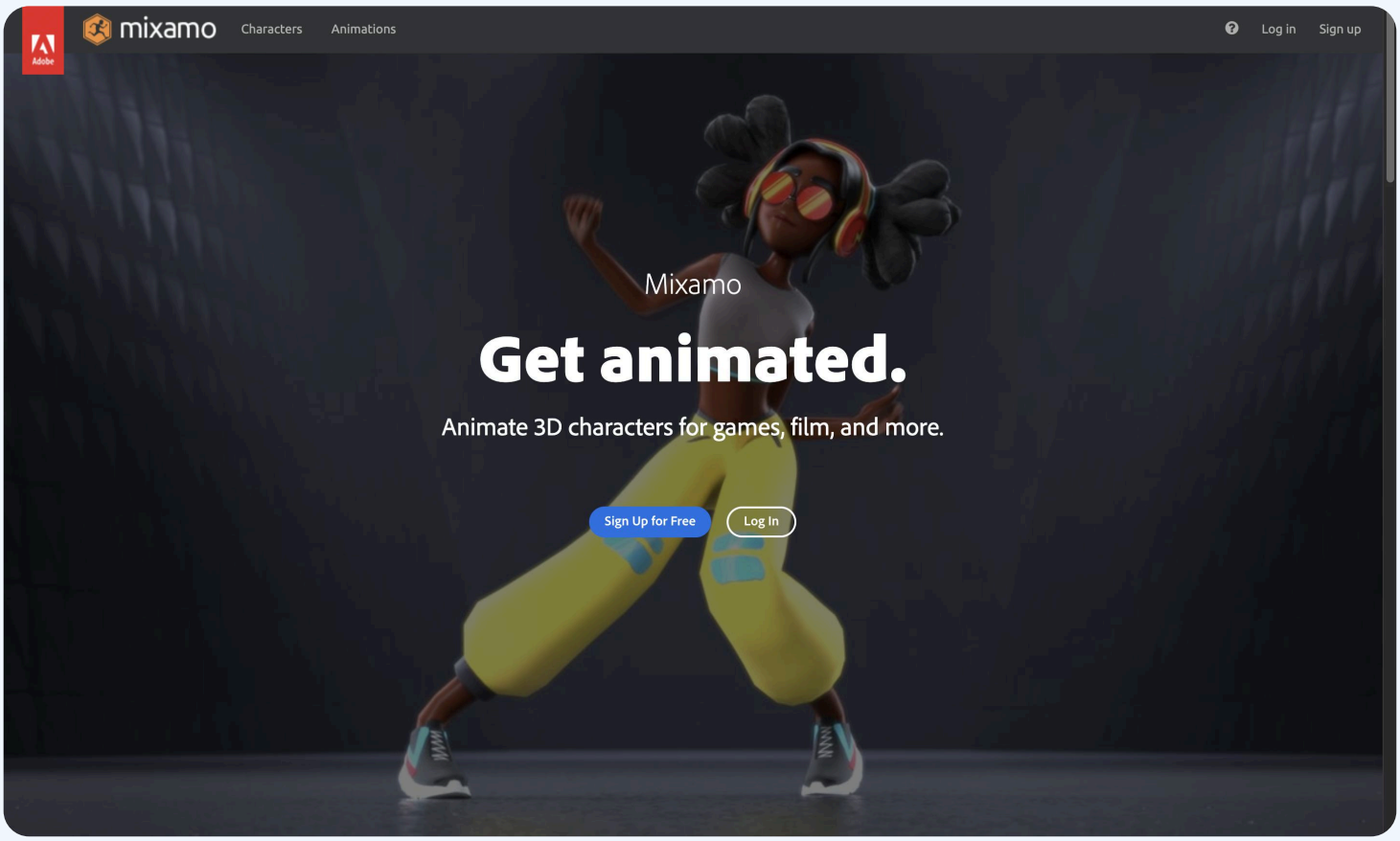
- **Direct Integrations**

  For apps that support Ready Player Me, you can directly link your avatar without needing to download or convert files.

Mixamo

Mixamo is a web-based platform by Adobe that provides an extensive library of high-quality, ready-to-use animations and 3D characters.

It's a powerful tool for animators, game developers, and designers who want to add realistic movements to their 3D models without needing to create animations from scratch.



Mixamo

With Mixamo, you can upload your custom 3D models and apply a wide range of animations, from walking and running to dancing or fighting, all in just a few clicks.

Why Use Mixamo?

1. **Ready-to-Use Animations**

   Mixamo offers a vast collection of pre-made animations that can be applied to any 3D character, saving time and effort.

2. **Ease of Use**

   No prior animation experience is needed. Mixamo's intuitive interface allows you to quickly upload, rig, and animate characters.

Mixamo

3. **Customizability**

   You can fine-tune animations to fit your character's needs by adjusting speed, arm spacing, and loop settings.

4. **Free to Use**

   Mixamo is available to anyone with an Adobe account, and there are no additional costs for downloading animations or using the service.

How to use Mixamo?

Step 1: Visit the Mixamo Website

Go to the Mixamo website. Log in with your account. If you don't have one, you can create one for free.

Step 2: Upload Your 3D Model

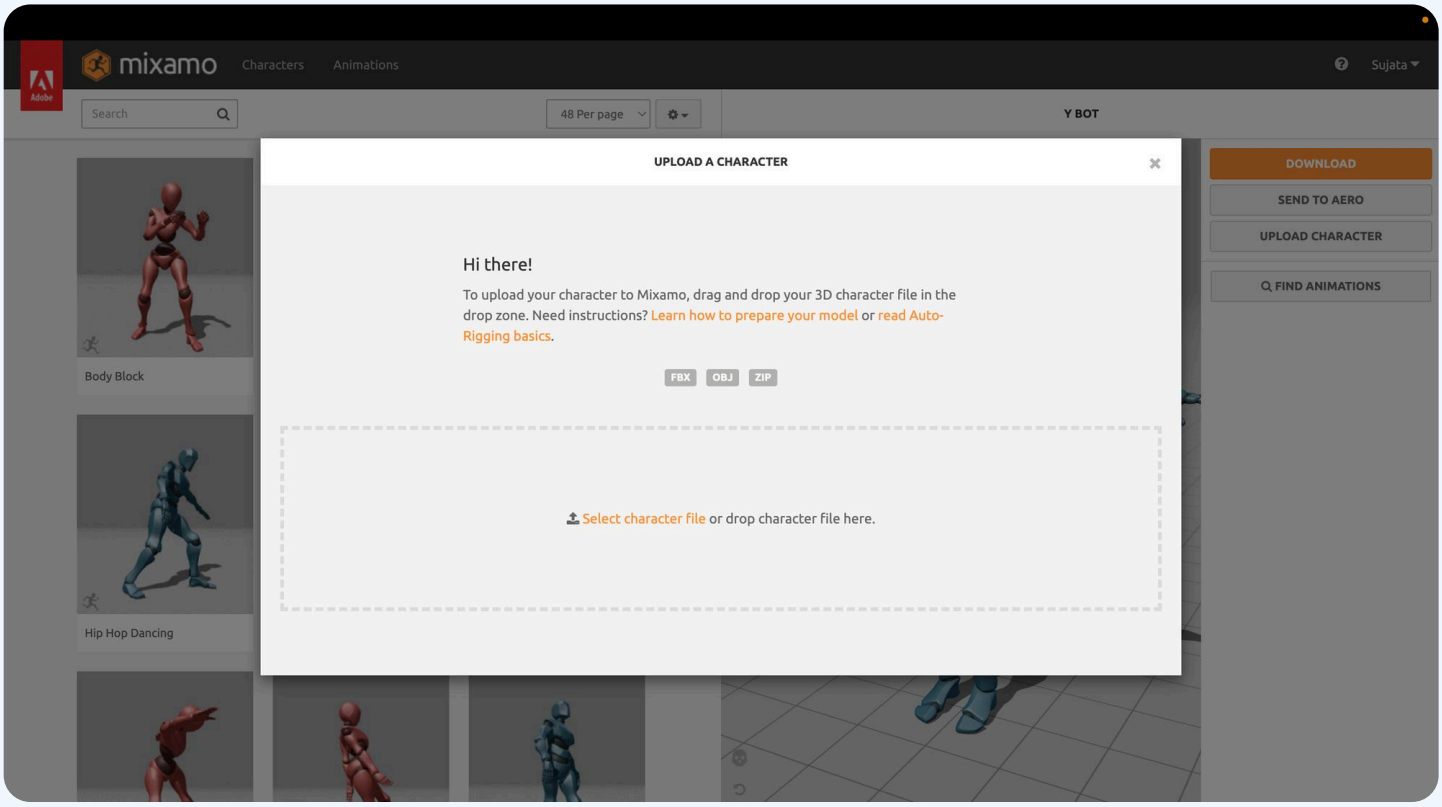
1. **Click "Upload Character"**

   On the homepage, click the "Upload Character" button to begin the process.

2. **Select Your Model**

   Choose the 3D model file you want to upload. Mixamo supports several file formats, including FBX, OBJ, ZIP (containing textures), and more.

How to use Mixamo?



Step 3: Auto-Rig Your Character

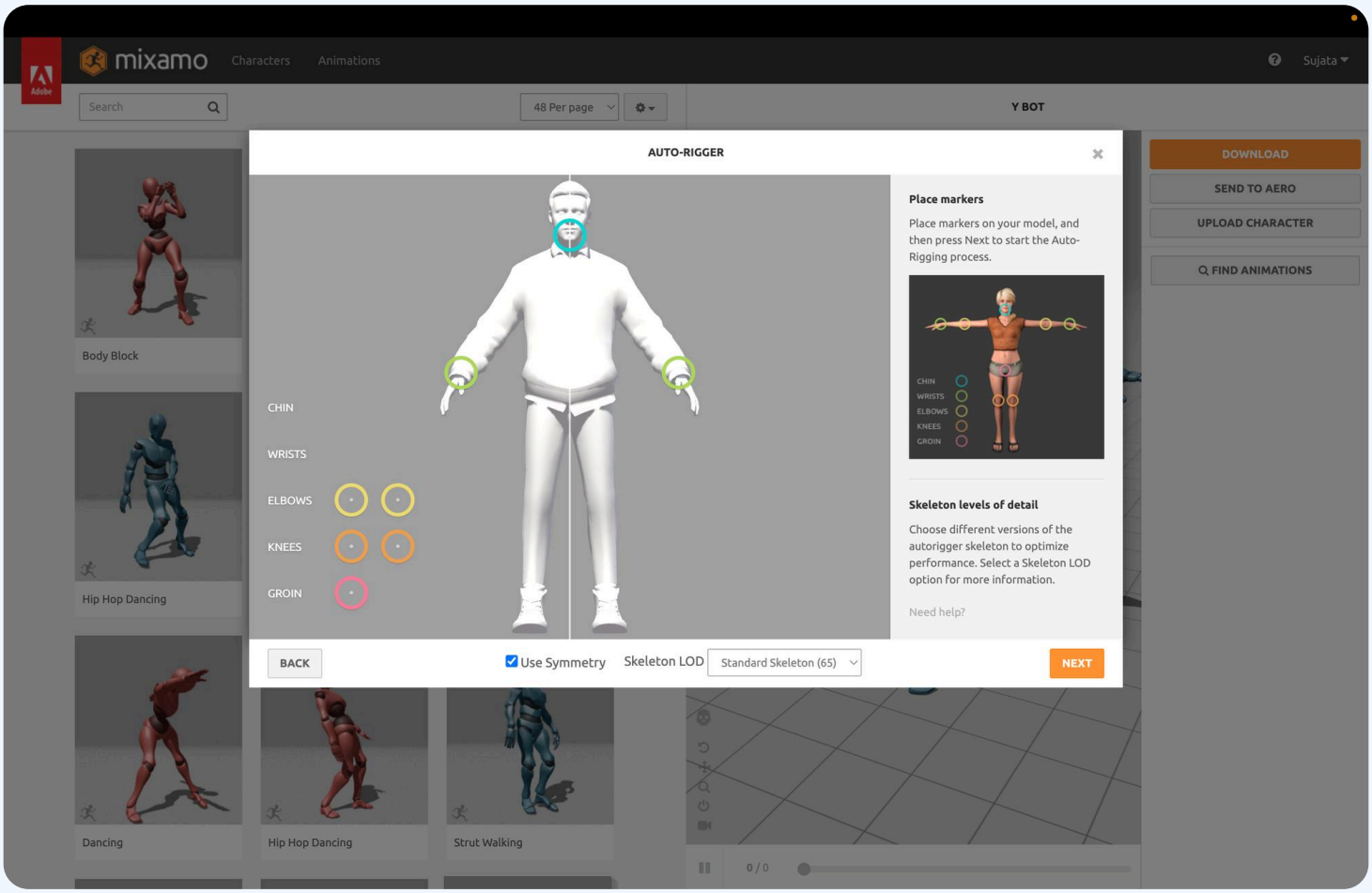
1. **Place the Markers**

   If your model is not rigged (i.e., doesn't have a skeleton), Mixamo will guide you through the auto-rigging process. Place markers on key points like the chin, wrists, elbows, knees, and groin.

Mixamo

2. Start Rigging

Click “Next,” and Mixamo will automatically rig your character, creating a skeleton that allows it to move.



Mixamo

Step 4: Choose an Animation

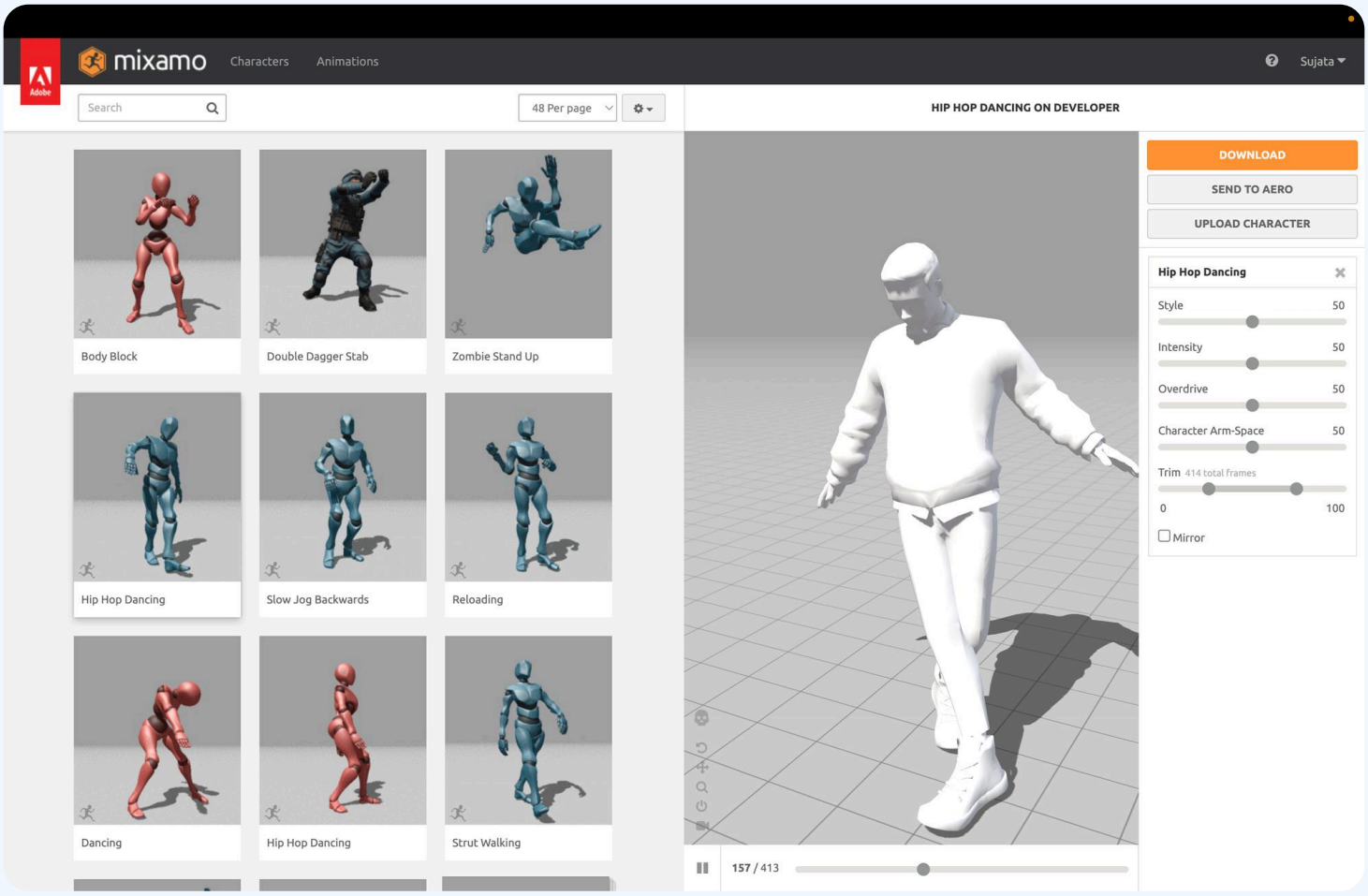
1. **Select from the Animation Library**

   Once your model is rigged, you can browse Mixamo's extensive animation library. Pick any animation to see a real-time preview of how it looks on your character.

2. **Customize the Animation**

   Adjust the animation settings such as speed, arm spacing, and more to fit your needs.

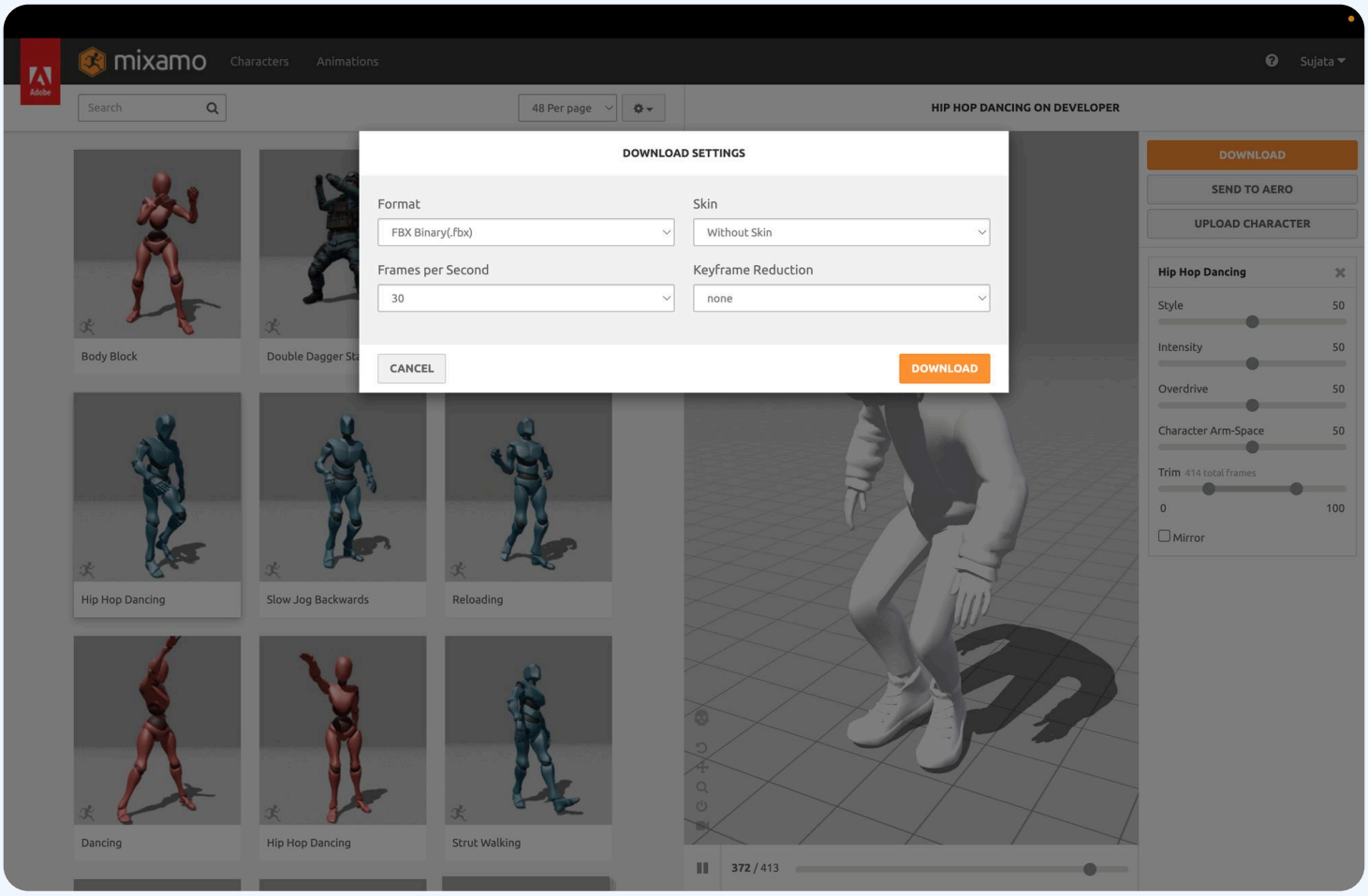
Mixamo



Step 5: Download Your Animated Character

After choosing and customizing an animation, click the “Download” button. Choose the format, frames per second (FPS), and whether you want it with or without skin (i.e., with or without the model's textures). Download the animation to your computer.

Mixamo


mixamo
Characters
Animations
Sujata
Search
48 Per page
HIP HOP DANCING ON DEVELOPER
DOWNLOAD SETTINGS
Format
FBX Binary(.fbx)
Skin
Without Skin
Frames per Second
30
Keyframe Reduction
none
CANCEL
DOWNLOAD
DOWNLOAD
SEND TO AERO
UPLOAD CHARACTER
Hip Hop Dancing
Style 50
Intensity 50
Overdrive 50
Character Arm-Space 50
Trim 414 total frames
0
100
Mirror
Body Block
Hip Hop Dancing
Slow Jog Backwards
Reloading
Dancing
Hip Hop Dancing
Strut Walking
372 / 413

Using Your Ready Player Me Avatar in Mixamo

Step-by-Step Guide to Exporting and Importing Custom Model created using **Ready Player Me** with **Mixamo** animations.

Step 1: Export Your Avatar from Ready Player Me

1. **Go to the Ready Player Me Website**

   Visit Ready Player Me and log in to your account.

2. **Access Your Avatar**

   Choose the avatar you want to export.

3. **Export Your Avatar**

   Click on the “Download” or “Export” option in GLB format, which is Ready Player Me's default format.

Using Your Ready Player Me Avatar in Mixamo

Step 2: Prepare Your Avatar for Mixamo

1. **Check the File Format**

   If your avatar is in GLB format, you'll need to convert it to FBX to be compatible with Mixamo.

2. **Convert GLB to FBX Format**

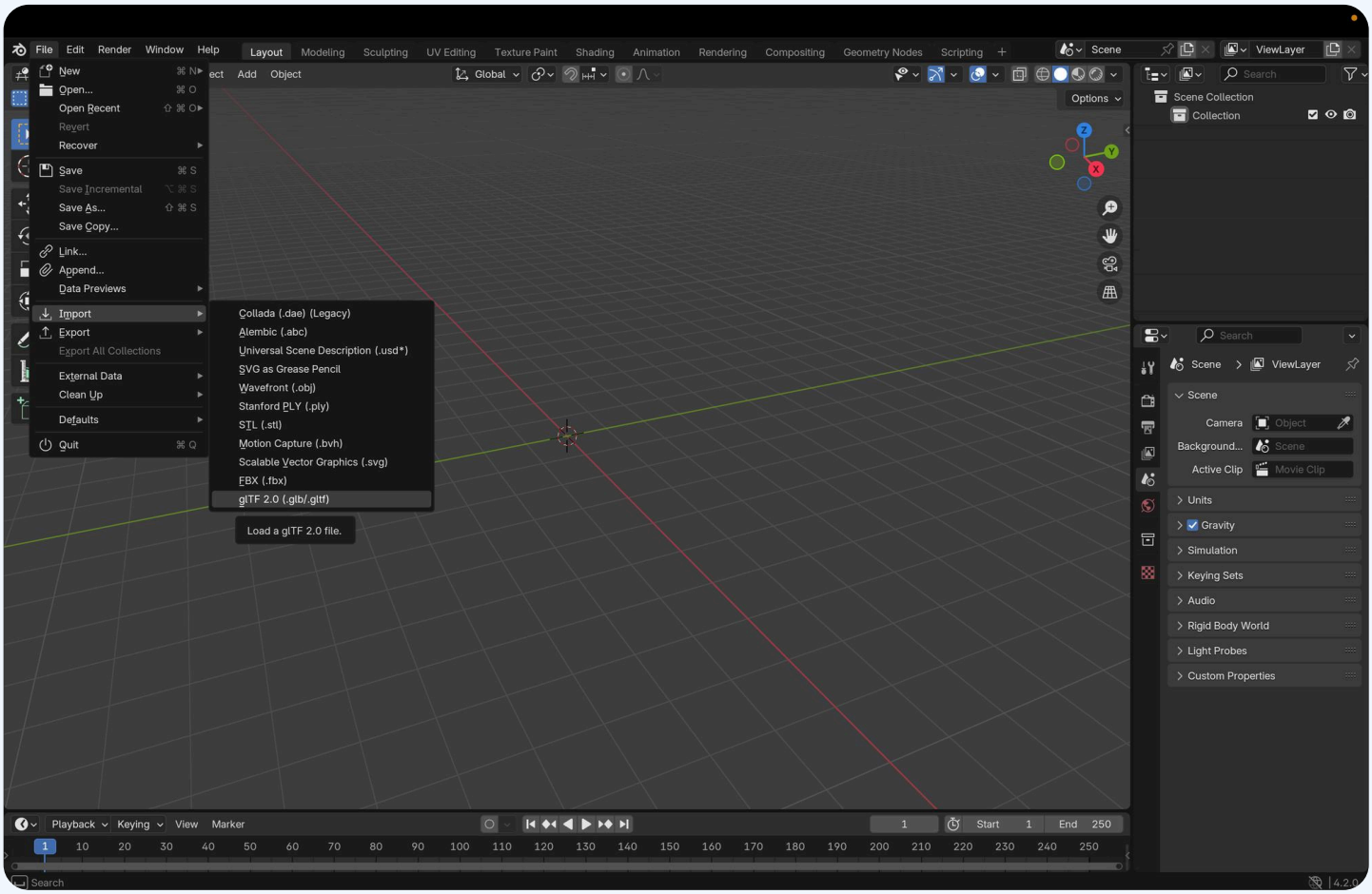
   a. Use Blender

   Download and install Blender, a free and powerful 3D modeling tool.

   Open Blender and delete the default objects (usually a cube and a camera).

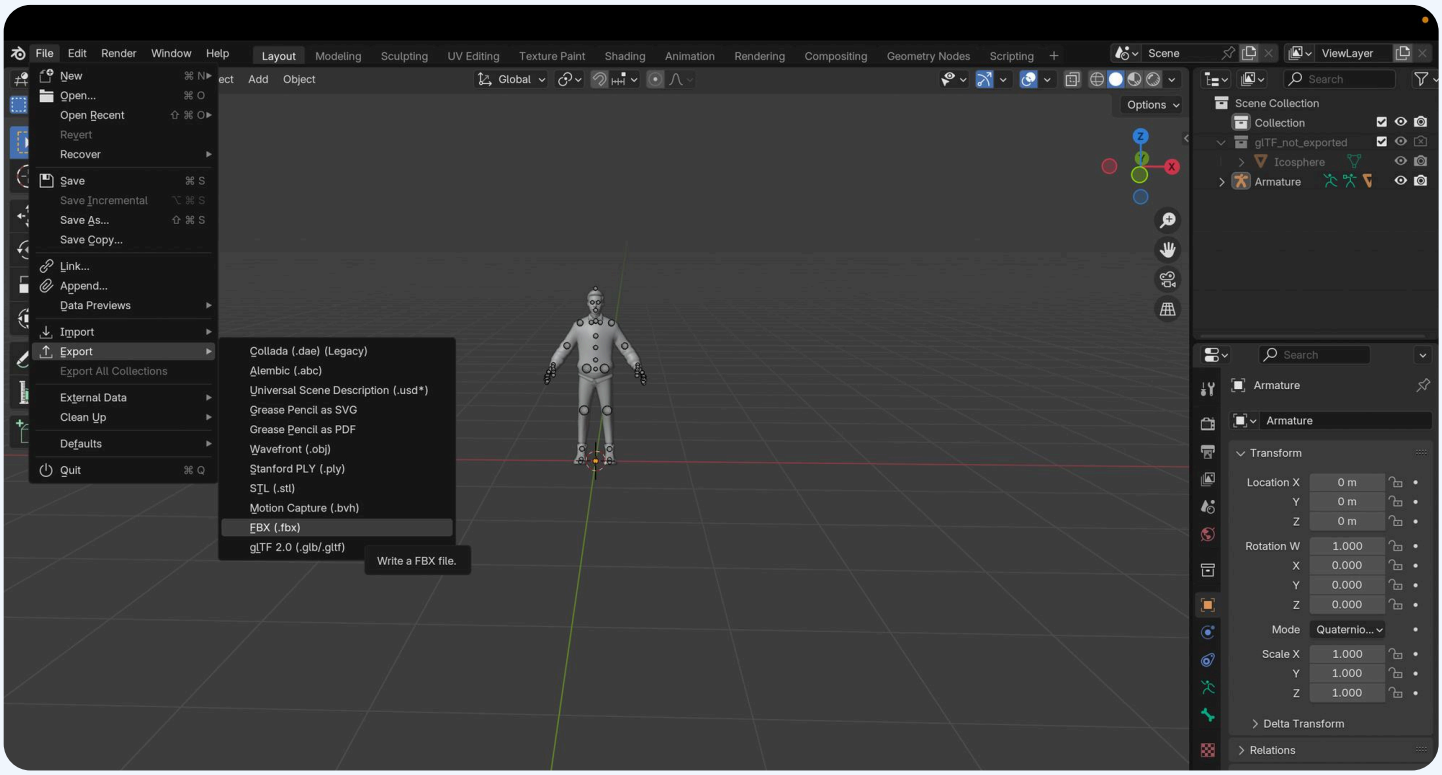
Using Your Ready Player Me Avatar in Mixamo

Go to **File > Import > glTF 2.0 (.glb/.gltf)** and select your Ready Player Me avatar (GLB file).



Using Your Ready Player Me Avatar in Mixamo

Once the model is imported, go to **File > Export > FBX (.fbx).**



In the export settings, ensure you check "Selected Objects" and set the appropriate scale. Then, click "Export FBX."

Using Your Ready Player Me Avatar in Mixamo

2. **Convert GLB to FBX Format**

   b. Alternative Online Tools

   If you prefer not to use Blender, there are online converters like AnyConv or Aspose that can convert GLB to FBX. Simply upload your GLB file, choose FBX as the output format, and download the converted file.

2. **Simplify the Model**

   Mixamo works best with models that are under 70,000 polygons. If your avatar is complex, consider simplifying it using 3D software like Blender before uploading.

Using Your Ready Player Me Avatar in Mixamo

Step 3: Upload Your Avatar to Mixamo

1. **Visit Mixamo**

   Go to the Mixamo website and log in.

2. **Click "Upload Character"**

   Upload the FBX file of your Ready Player Me avatar.

3. **Rig the Character**

   If the character is not already rigged, Mixamo will guide you through the auto-rigging process.

Using Your Ready Player Me Avatar in Mixamo

Step 4: Apply Animations in Mixamo

1. **Customize the Animation**

   Browse the animations and select one that suits your avatar. The animation will be automatically applied, and you can see a preview in real time.

2. **Customize the Animation**

   Adjust any settings to refine how the animation looks.

Using Your Ready Player Me Avatar in Mixamo

Step 5: Download the Animated Avatar

1. **Click “Download”**

   Choose your preferred settings, such as format (FBX).

2. **Integrate into Your Project**

   Use the animated avatar in your website, game, VR experience, or other digital projects.

Tips for Using Ready Player Me Avatars in Mixamo

- **Maintain a Low Poly Count**

  To avoid performance issues, keep your model's polygon count low.

- **Check Textures and Materials**

  Ensure that all textures and materials are properly mapped after importing to Mixamo.

- **Test Different Animations**

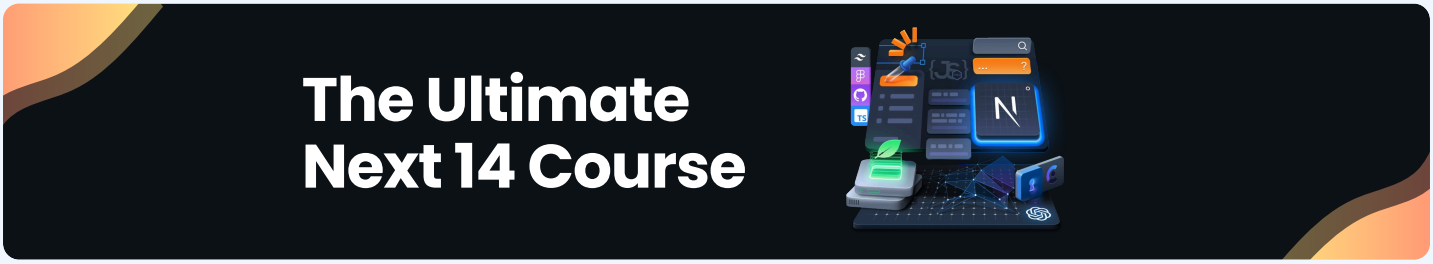
  Not all animations will look perfect with every avatar. Experiment with different ones to find the best fit.

That's all you need to know to create your own 3D avatars and animate them for use anywhere you like.

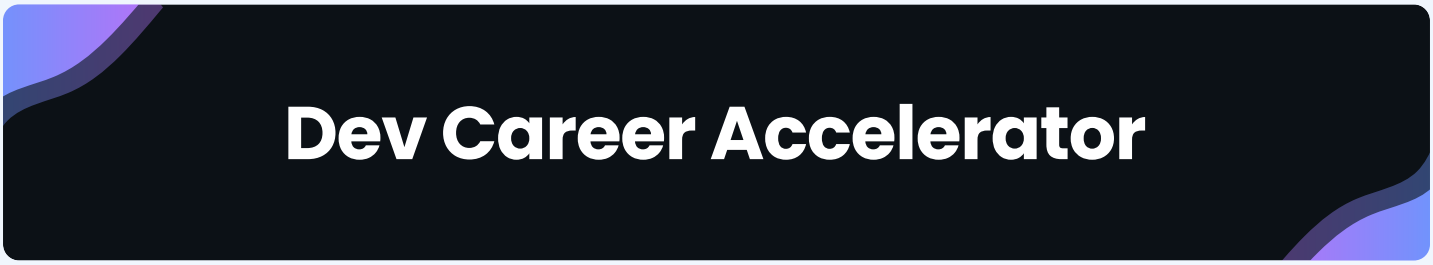
The End

Congratulations on reaching the end of our guide! But hey, learning doesn't have to stop here.

If you're eager to dive deep into something this specific and build substantial projects, our **special course on Next.js** has got you covered.



If you're craving a more personalized learning experience with the guidance of expert mentors, we have something for you — [Dev Career Accelerator](#).



If this sounds like something you need, then don't stop yourself from leveling up your skills from junior to senior.

Keep the learning momentum going. Cheers! 🚀