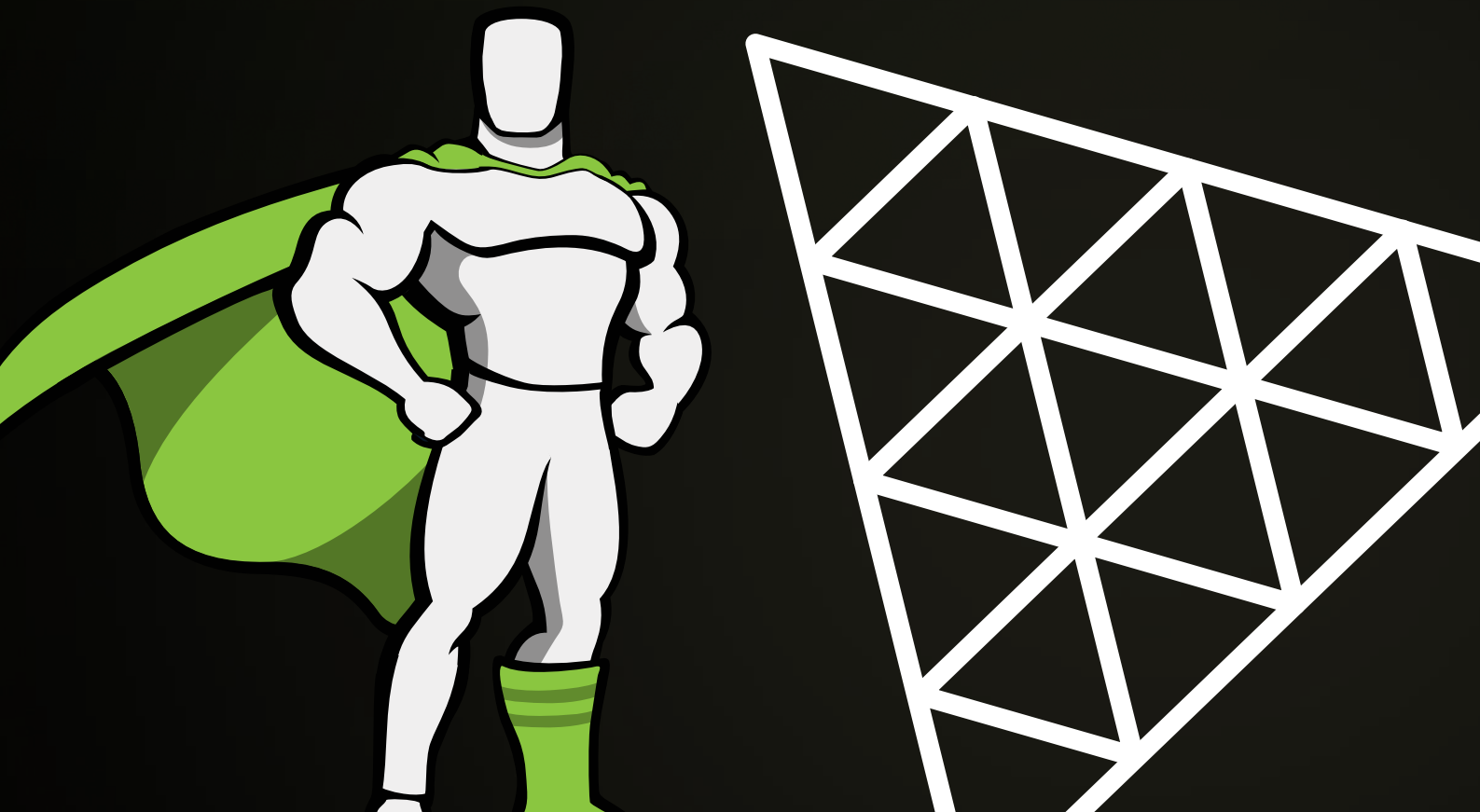
{JS}
MASTERY

The Ultimate Three JS and GSAP Guide

Created by **JS Mastery**
Visit ***jsmastery.pro*** for more



What’s in the guide?

Welcome to The Ultimate Three.js and GSAP Guide! 👋

Whether you're a beginner dipping your toes into the world of web animation or a seasoned developer looking to elevate your skills, you've arrived at the perfect destination.

This comprehensive guide is your roadmap to mastering the dynamic duo of Three.js and GSAP.

From foundational principles to advanced techniques, this guide will take you on a journey through: **step-by-step roadmap, insider tips, examples, & project ideas**.

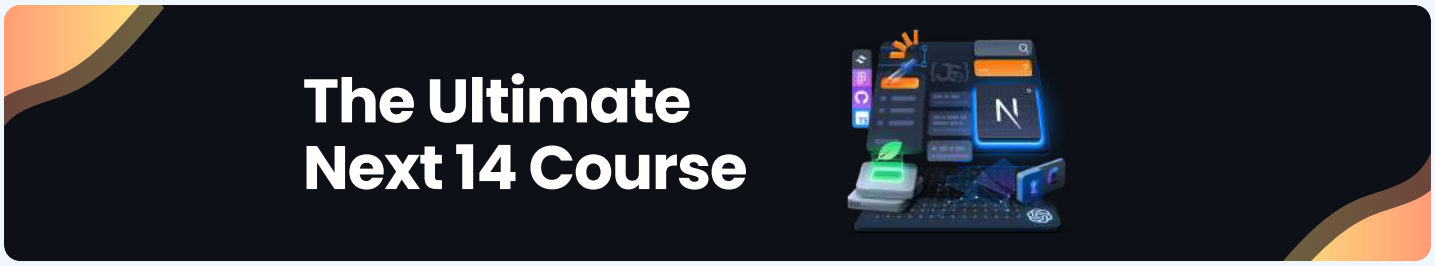
So, dive in and let's embark on an exhilarating journey to unleash the full potential of Three.js and GSAP.

Happy animating and coding! 🎨✨

...before you go

While The Three.js & GSAP Guide equips you with invaluable skills in web animation & 3D development, imagine taking your skills to the next level by seamlessly integrating these techniques with Next.js.

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.



It teaches everything from the ground up, providing a hands-on experience that goes beyond just Three.js and GSAP.

Check it out and take your skills to the next level 🚀

{JS}
MASTERY

GSAP

What is GSAP?

GSAP is a framework-agnostic JavaScript animation library that turns devs into animation superheroes.

Build high-performance animations that work in every major browser. Animate CSS, SVG, canvas, React, Vue, WebGL, colors, strings, motion paths, generic objects...anything JavaScript can touch!

GSAP is unmatched in **delivering advanced sequencing**, **reliability**, and **precise control** for animations on over 12 million websites.

It effortlessly handles browser inconsistencies, ensuring your animations work seamlessly.

GSAP is a fast property manipulator, updating values over time with precision, and it's up to 20 times faster than jQuery!

GSAP Setup

To use GSAP, you have multiple options for integrating it into your project.

1. NPM

One common approach is to install **GSAP via npm**, which allows you to manage dependencies efficiently within your project's ecosystem.

This method is particularly useful for larger projects or those using modern build tools like Webpack or Parcel.

2. CDN

Alternatively, you can opt for the quick and easy method of including GSAP directly via a **CDN** (Content Delivery Network) link in your HTML file.

This approach is convenient for smaller projects or when you want to quickly prototype an idea.

GSAP Setup

3. React

If you're working with React, you have the option to use the **@gsap/react library**, which provides seamless integration of GSAP with React components.

This allows you to harness the power of GSAP within your React applications while leveraging the component-based architecture of React.

Overall, the flexibility of GSAP's integration options ensures that you can easily incorporate it into your preferred development workflow, whether you're building a traditional website, a single-page application, or a complex web application with React.

GSAP Setup

3. React

If you're working with React, you have the option to use the **@gsap/react library**, which provides seamless integration of GSAP with React components.

This allows you to harness the power of GSAP within your React applications while leveraging the component-based architecture of React.

Overall, the flexibility of GSAP's integration options ensures that you can easily incorporate it into your preferred development workflow, whether you're building a traditional website, a single-page application, or a complex web application with React.

CDN Setup

To get started with GSAP, we first need to add the GSAP library to our HTML file. You can do this by grabbing the **CDN link** to the GSAP library.

Here's how it will look, but with the full link.

```
<script
  src="https://cdnjs.cloudflare.com/ajax/libs/gsap/3.12....
  integrity="sha512-EZI2cBcGPnmR89wTgVnN3602Yyi7muWo8say...
  crossorigin="anonymous"
  referrerpolicy="no-referrer"
></script>
```

Make sure to place this script tag before the script tag containing your local JS file. This ensures that your local file has access to the GSAP library and its functionality.

Also, if you're using any GSAP plugins, ensure that their script tags are placed after the GSAP script tag.

GSAP Basics

GSAP is incredibly flexible; you can use it anywhere you like, and it has zero dependencies.

If you've used any version of GSAP in the past couple of years or are familiar with tools like **TweenLite, TweenMax, TimelineLite, and TimelineMax**, you'll notice that in the new version, GSAP 3.0, they have all been replaced by the **GSAP object**.

Think of the GSAP object as the main hub for everything in GSAP.

It's like a toolbox filled with all the tools you need to create and control Tweens and Timelines, which are the main things you'll be working with in GSAP.

To really get the hang of GSAP, it's important to understand Tweens and Timelines:

GSAP Basics

Understanding Tween

Think of a Tween as the magic that makes things move smoothly—it's like a super-efficient property setter. You give it targets (the objects you want to animate), set a duration, and specify which properties you want to change.

Then, as the Tween progresses, it calculates and applies the property values at each step, creating seamless animation.

Here are some common methods for creating a Tween:

- **`gsap.to()`**
- **`gsap.from()`**
- **`gsap.fromTo()`**

GSAP Basics

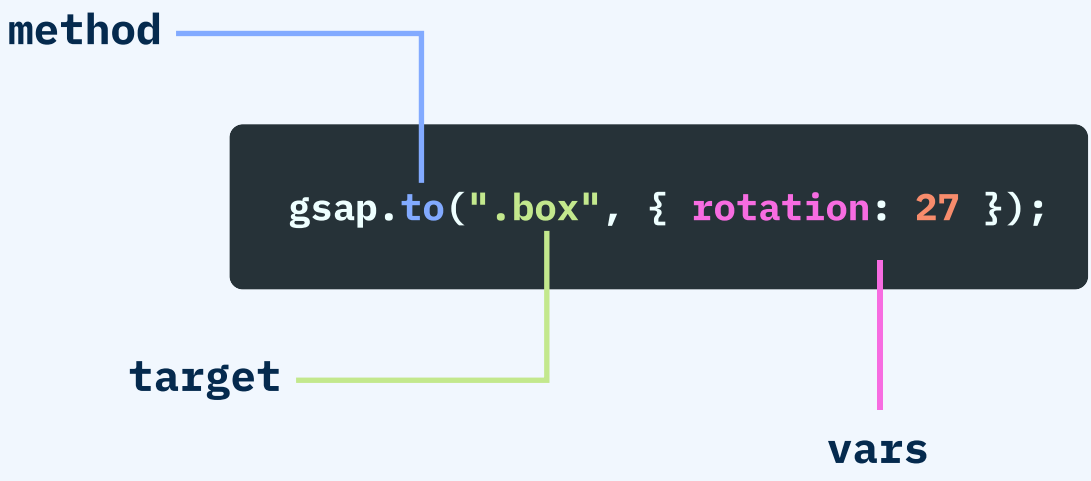
For simple animations (no fancy sequencing), the methods above are all you need! For example:

```
gsap.to(".box", { rotation: 27, x: 100, duration: 1 });
```

rotate and move elements with a class of "box" ("x" is a shortcut for a translateX() transform) over the duration of 1 second.

Basic sequencing can be achieved by utilizing the **delay** special property.

Let's take a closer look at the syntax.



GSAP Basics

We've got a method, a target and a vars object which all contain information about the animation

The method(s)

There are four types of tweens:

`gsap.to()`

his is the most common type of tween. A `to()` tween starts at the element's current state and **animates "to" the values defined in the tween.**

`gsap.from()`

This is similar to a backwards `to()` tween. It animates "from" the values defined in the tween and ends at the element's current state.

GSAP Basics

`gsap.fromTo()`

With this method, you define both the starting and ending values for the tween.

`gsap.set()`

This method immediately sets properties without any animation. It's essentially a zero-duration `to()` tween.

These methods provide flexibility in how you create tweens and allow you to achieve various effects in your animations.

GSAP Basics

The target(s)

In GSAP, you need to specify what you want to animate, known as the target or targets. GSAP internally utilizes **`document.querySelectorAll()`**, allowing you to use selector text like ".class" or "#id" for HTML or SVG targets. Alternatively, you can pass in a variable or an Array.

Here are some examples:

```
  // Using a class or ID
★ gsap.to(".box", { x: 500 });

  // Using a complex CSS selector
★ gsap.to("section > .box", { x: 900 });

  // Using a variable
  let box = document.querySelector(".box");
★ gsap.to(box, { x: 200 });

  // Using an Array of elements
  let square = document.querySelector(".square");
  let circle = document.querySelector(".circle");
★ gsap.to([square, circle], { x: 200 });
```

GSAP Basics

These examples showcase how you can specify different targets for your animations in GSAP, allowing you to animate various elements with different properties and values.

The variables

The vars object holds all the details about the animation. It includes properties you wish to animate, as well as special properties that control the animation's behavior, such as **`duration`**, or **`repeat`**.

Here's how you might use it in a **`gsap.to()`** tween:

```
gsap.to(target, { // This is the vars object
  // It contains properties to animate
  x: 200,
  rotation: 360,
  // Along with special properties
  duration: 2,
});
```

GSAP Basics

What properties can we animate?

With GSAP, you have the flexibility to animate **almost anything you can imagine**. There's no predefined list of properties you can animate because GSAP is incredibly versatile.

You can animate CSS properties like width, height, color, and font-size, as well as custom object properties. GSAP even allows you to animate CSS variables and complex strings!

While you can animate virtually any property, some of the most commonly animated properties include transforms (such as **translate, rotate, scale**), opacity, and position. These properties are frequently used to create smooth and visually appealing animations across various elements on your webpage.

GSAP Basics

However, for more advanced sequencing and complex choreography, **Timelines** are the way to go. They make the process much easier & more intuitive.

Understanding Timeline

A Timeline serves as a container for Tweens, making it the ultimate tool for sequencing animations. With a Timeline, you have the power to position animations in time exactly where you want them.

You can effortlessly control the entire sequence using methods like **`pause()`, `play()`, `progress()`, `reverse()`**, and **`timeScale()`**, among others.

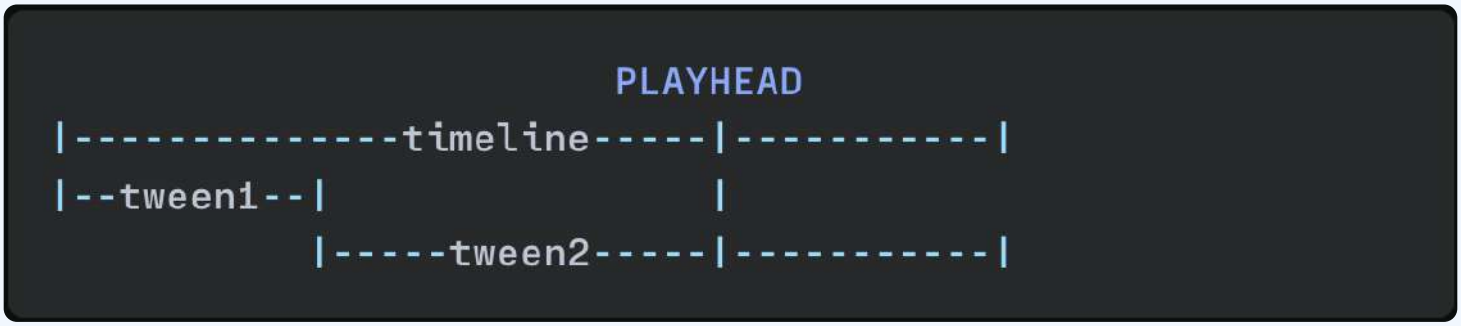
The beauty of Timelines is that you can create as many as you need, and even nest them, which is great for organizing your animation code into manageable modules.

GSAP Basics

Here's the cool part: every animation (whether it's a Tween or another Timeline) is placed onto a parent timeline (which is usually the globalTimeline by default).

This means that when you move a Timeline's playhead, it cascades down through its children, ensuring that all the animations stay perfectly synchronized.

It's important to note that a Timeline is all about grouping & coordinating animations in time, it doesn't actually set properties on targets like Tweens do.



GSAP Basics

To create a Timeline in GSAP, you simply use the method:

```
gsap.timeline()
```

With GSAP's API, you have the power to control virtually anything on-the-fly.

You can manipulate the playhead position, adjust the **startTime** of any child, play, pause, or reverse animations, alter the timeScale, and much more.

GSAP Basics

Sequencing things in a Timeline

To create a Timeline, you first initialize it like this:

```
var tl = gsap.timeline();
```

Then, you can add tweens using one of the convenience methods like `to()`, `from()`, or `fromTo()`:

```
tl.to(".box", { duration: 2, x: 100, opacity: 0.5 });
```

You can repeat this process as many times as needed. Notice that we're calling .`to()` on the timeline instance (in this case, the variable `tl`), not on the gsap object.

This creates a tween and immediately adds it to that specific Timeline.

GSAP Basics

By default, the animations will be sequenced one-after-the-other. You can even use method chaining to simplify your code:

```
// Sequenced one-after-the-other
tl.to(".box1", { duration: 2, x: 100 })
  .to(".box2", { duration: 1, y: 200 })
  .to(".box3", { duration: 3, rotation: 360 });
```

It's worth noting that while you could create individual tween instances with **`gsap.to()`** and then use **`timeline.add()`** to add each one, it's much easier to call `.to()`, `.from()`, or `.fromTo()` directly on the Timeline instance.

This approach accomplishes the same thing in fewer steps, keeping your code clean and concise.

React GSAP

Why choose GSAP with React?

While React-specific libraries provide a declarative approach to animation, GSAP offers unique advantages.

Animating imperatively with GSAP empowers you with greater **control**, **flexibility**, and **creativity**.

Whether you're animating DOM elements, SVGs, three.js, canvas, or WebGL, GSAP allows you to unleash your imagination without limits.

What sets GSAP apart is its framework-agnostic nature. This means that your animation skills seamlessly transfer to any project, be it Vanilla JS, React, Vue, Angular, or Webflow.

React GSAP

With GSAP, you won't need to switch between different libraries for different projects. It becomes your reliable toolkit, ensuring consistency and efficiency across all your goals.

React Setup

Setup for experimenting with React and GSAP.

```
npm create vite@latest my-react-app -- --template react
```

Once the project is set up we can install GSAP and the special GSAP/React package through npm,

```
# Install the GSAP library
npm install gsap

# Install the GSAP React package
npm install @gsap/react

# Start the project
npm start
```

React Setup

Then import it into the app.

```
import { useRef } from "react";

import gsap from "gsap"; // <-- import GSAP
import { useGSAP } from "@gsap/react";

gsap.registerPlugin(useGSAP);

export default function App() {
  const container = useRef();

  useGSAP(
    () => {
      // gsap code here...
      gsap.to(".box", { rotation: 180 });
    },
    { scope: container }
  );

  return (
    <div ref={container} className="app">
      <div className="box">Hello</div>
    </div>
  );
}
```

React GSAP

What is that useGSAP() Hook?

GSAP works seamlessly with any JavaScript framework, without needing any special adjustments.

However, this hook is specifically designed to smooth out some React-specific challenges, letting you focus on the fun parts.

`useGSAP()` is a convenient replacement for **`useEffect()`** or **`useLayoutEffect()`**. It automatically manages cleanup using **`gsap.context()`**. Proper cleanup is important in React, and using Context makes it easy.

Simply import the **`useGSAP()`** hook from **`@gsap/react`**, and you're ready to roll!

Any GSAP animations, ScrollTriggers, Draggables, or SplitText instances created with the **`useGSAP()`** hook

React GSAP

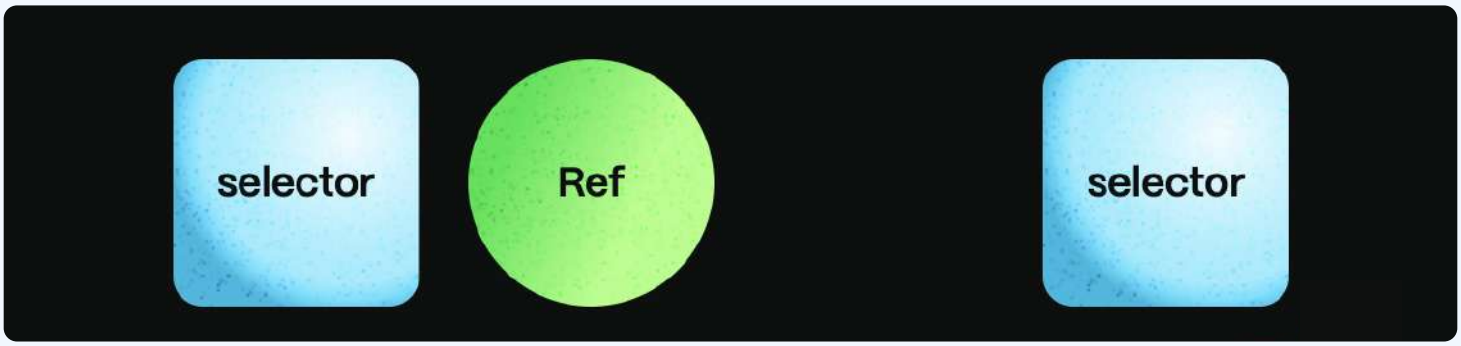
will be cleaned up automatically when the component is unmounted and the hook is removed.

```
import { useRef } from "react";
import gsap from "gsap";
import { useGSAP } from "@gsap/react";

gsap.registerPlugin(useGSAP);

const container = useRef();

useGSAP(
  () => {
    // gsap code here...
    gsap.to(".box", { x: 360 }); // <-- automatically reverted
  },
  { scope: container }
); // <-- scope is for selector text (optional)
```



React GSAP

Why is cleanup is crutial?

Cleanup is crucial in animation, particularly with frameworks like React. In React 18, there's a default strict mode setting that can cause Effects to run twice locally. This double execution can result in duplicate or conflicting animations, as well as logic issues with tweens if not properly reverted.

The **`useGSAP()`** hook adheres to React's recommended practices for animation cleanup, ensuring that your animations are handled correctly and efficiently.

You can safely use this hook in **Next.js** or similar server-side rendering environments.

GSAP Easing

The GSAP official documentation defines easing as the primary method to adjust the timing of your Tweens.

Easing determines how an object transitions between positions at different points during an animation.

It controls the rate of change of animation and sets the style of an object's movement.

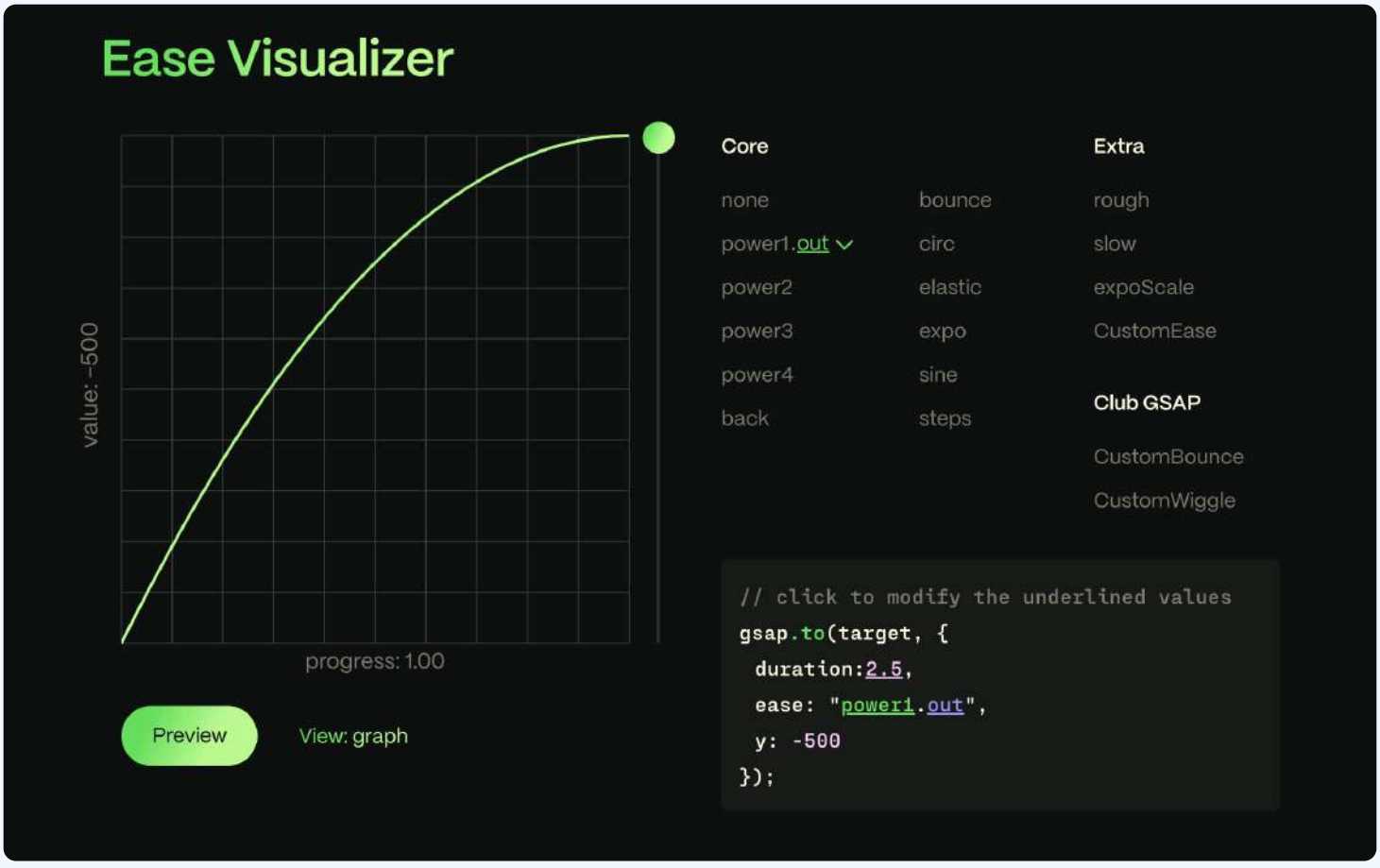
GSAP offers various types of eases and options to provide you with greater control over your animation behavior.

Additionally, GSAP provides an **Ease Visualizer tool** to assist in selecting preferred ease settings.

There are three main types of eases, each operating differently:

GSAP Easing

- **`in()`**: Animation begins slowly and accelerates towards the end.
- **`out()`**: Animation starts quickly and decelerates towards the end.
- **`inOut()`**: Animation starts slowly, accelerates in the middle, and then decelerates towards the end.



GSAP Plugins

A plugin extends the capabilities of GSAP's core functionality. By using plugins, the GSAP core remains lightweight while providing additional features that can be added as needed.

This modular approach allows you to customize your GSAP setup by including only the features that are required for your specific project, keeping your codebase efficient and focused.

GSAP Plugins

To install or load a plugin, you have several options. Plugins are JavaScript files, similar to the core GSAP library. You can install them using **script tags, npm, yarn**, or even a **tgz file**.

Here's how you can register a plugin

```
//list as many as you'd like
gsap.registerPlugin(MotionPathPlugin, ScrollTrigger);
```

Remember, you need to load the plugin file before registering it. Registering a plugin with GSAP ensures seamless integration and avoids issues with build tools and bundlers.

You only need to register a plugin once before using it. It's okay to register the same plugin multiple times, but it doesn't offer any additional benefits.

GSAP Plugins

ScrollTrigger Plugin Example

ScrollTrigger empowers users to effortlessly create nice scroll-based animations with minimal code.

It offers unparalleled flexibility, allowing you to easily implement scroll-triggered effects such as scrubbing, pinning, snapping, or triggering any scroll-related action, regardless of whether it involves animation or not.

Overall, ScrollTrigger provides a flexible and intuitive way to create engaging scroll-based animations and interactions, making it a valuable tool for enhancing user experiences on the web.

GSAP Plugins

Here's a simple example using ScrollTrigger to animate a box:

```
gsap.to(".box", {
  scrollTrigger: ".box",
  x: 500,
});
```

In this example, when the element with the class "box" enters the viewport, GSAP will animate it by moving it 500 pixels to the right.

ScrollTriggers can perform an actions on an animation (play, pause, resume, restart, reverse, complete, reset) when entering/leaving the defined area or link it directly to the scrollbar so that it acts like a scrubber (`scrub: true`).

GSAP Examples

ScrollTrigger Showcase

https://codepen.io/collection/DkvGzg

GSAP Animation Websites

https://www.awwwards.com/websites/?aw…

Spilt Text Examples

https://codepen.io/collection/XMoeqD

Filer

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

GSAP Examples

Popular Demos

https://gsap.com/demos/

MorphSVG Showcase

https://codepen.io/collection/naMaNQ

DrawSVG Showcase

https://codepen.io/collection/DYmKKD

GSAP UI Design Examples

https://codemyui.com/tag/gsap/

Talented coders that feature a ton of GSAP

Cassie Evans

https://codepen.io/cassie-codes

Blake Bowen

https://codepen.io/osublake

Craig Roblewsky

https://codepen.io/PointC/

Darin Senneff

https://codepen.io/dsenneff

Talented coders that feature a ton of GSAP

Chris Gannon

https://codepen.io/chrisgannon

Carl Schooff

https://codepen.io/snorkltv

Pete Barr

https://codepen.io/petebarr

Steve Gardner

https://codepen.io/ste-vg

Talented coders that feature a ton of GSAP

Ryan Mulligan

https://codepen.io/hexagoncircle

Cameron Knight

https://codepen.io/cameronknight

Tom Miller

https://codepen.io/creativeocean

Tips

The most important tip

Don't go overboard with animations. Not everything needs to move, and too many animations can make things confusing for users. Focus on animating things that make the user experience better.

Properties like transformations (such as translate, rotate, scale) and opacity typically offer smooth performance across browsers.

On the other hand, properties like filter or boxShadow can be computationally expensive and may cause performance issues, especially on low-end devices.

Ensuring that your animations perform well on low-end devices is crucial for delivering a consistent user experience across different devices and platforms.

Tips

Easing functions

Experiment with different easing functions to add smoothness and character to your animations. GSAP has options like "**easeIn**" (starts slow and speeds up), "**easeOut**" (starts fast and slows down), and others you can experiment with.

Explore plugins

GSAP has extra tools you can add to make your animations even cooler. For example, SplitText lets you animate text in unique ways, and ScrollTrigger lets you create animations that happen when you scroll down a webpage. These plugins can add extra flair to your animations without much extra work.

Tips

Preload assets

Preload any assets, such as images or videos, that will be used in your animations to ensure smoother playback and prevent delays.

Practice with examples

One of the best ways to learn is by trying things out yourself. Look for examples of GSAP animations online and try to recreate them in your own projects.

Experimenting with different techniques will help you get better at creating your own unique animations.

Tips

Consider accessibility

Some people might have trouble with certain types of animations, like ones that flash or move around a lot.

Avoid animations that may cause discomfort or motion sickness, provide alternative content for screen readers, and allow users to pause or disable animations if needed.

Additionally, ensure that important information is still accessible even if someone can't see the animations.

Ensure that your animations are accessible to all users, including those with disabilities.

Tips

Test across devices

While GSAP is excellent in performance and provides a robust framework for creating animations, it's essential to remember that different devices have varying capabilities and screen sizes.

Therefore, thorough testing is necessary to guarantee a consistent and optimal UX across the board.

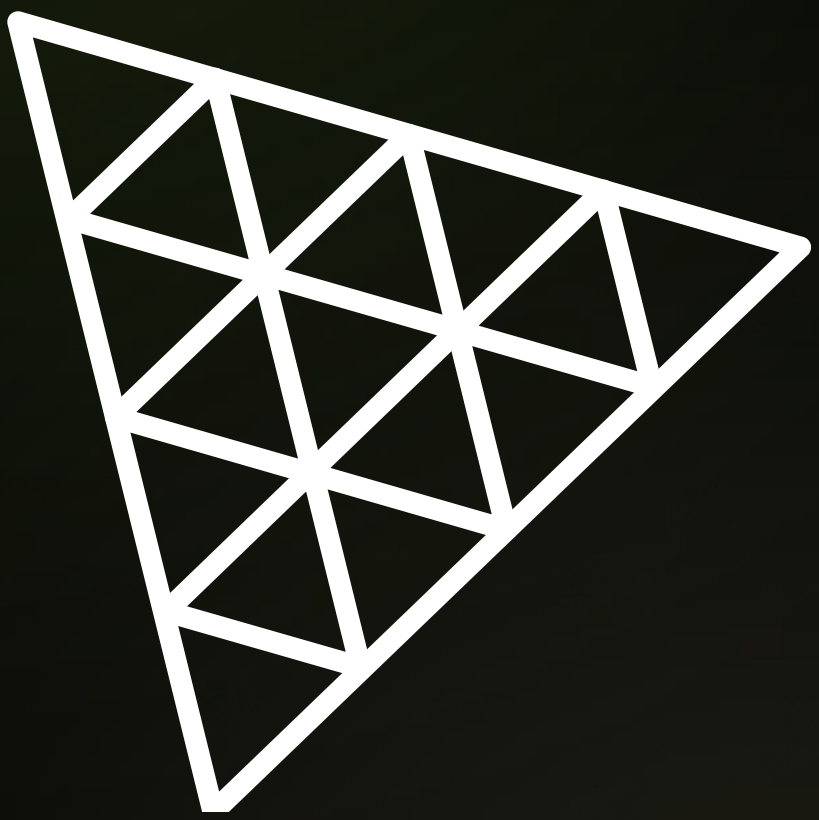
Not all devices are the same, so it's important to make sure your animations work well on all of them. Test your animations on different phones, tablets, and computers to make sure they look good and run smoothly everywhere.

Conclusion

The GSAP library is undeniably fascinating, and I'm glad I could assist you in understanding how to use it while adhering to good practices and tips. 🙌

For a better understanding and access to more features, I suggest reading the **documentation** and experimenting with different animation techniques.

{JS}
MASTERY



Three JS

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.

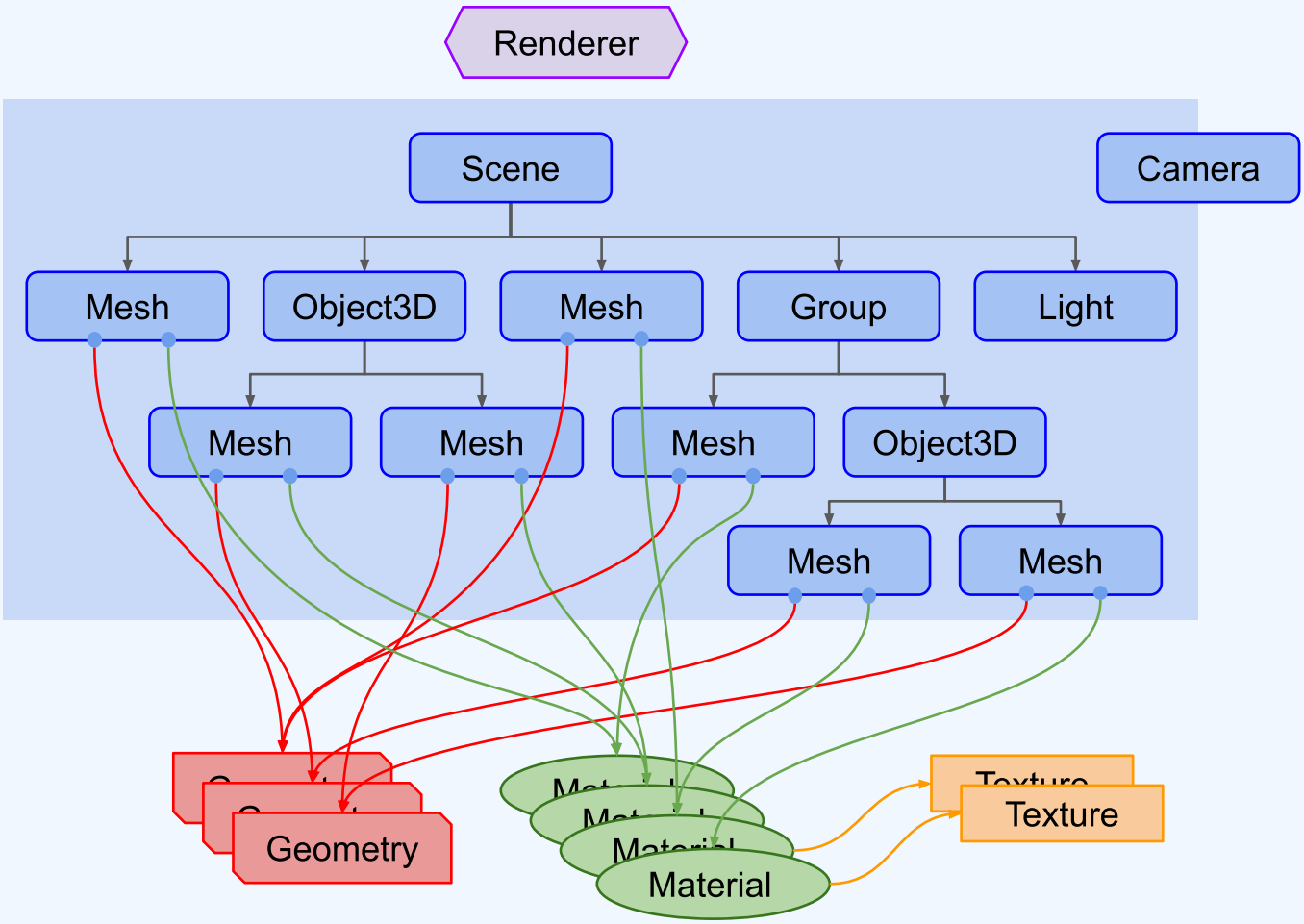
With a vibrant and supportive community of developers and designers, Three.js is constantly evolving and improving, making it an exciting and dynamic technology to explore and work with.

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.

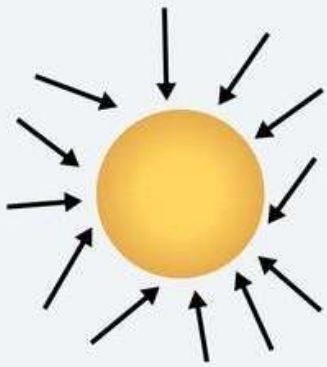
By understanding the basics of concepts such as renderer, geometry, texture, material, camera, light, scene, and animation, you can begin creating your own 3D scenes and exploring more advanced features in Three.js.

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.



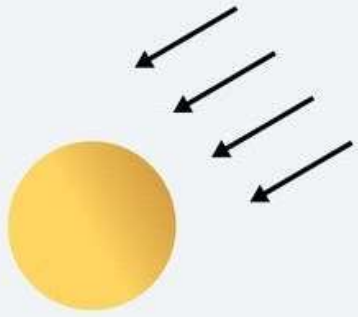
Ambient Light

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.



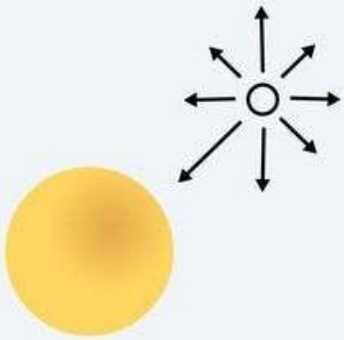
Directional Light

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.



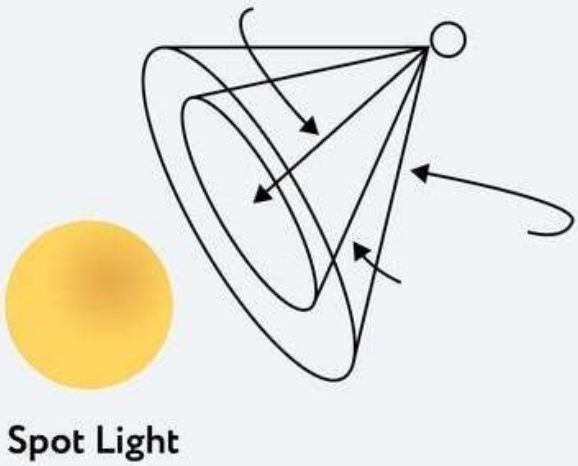
Point Light

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.



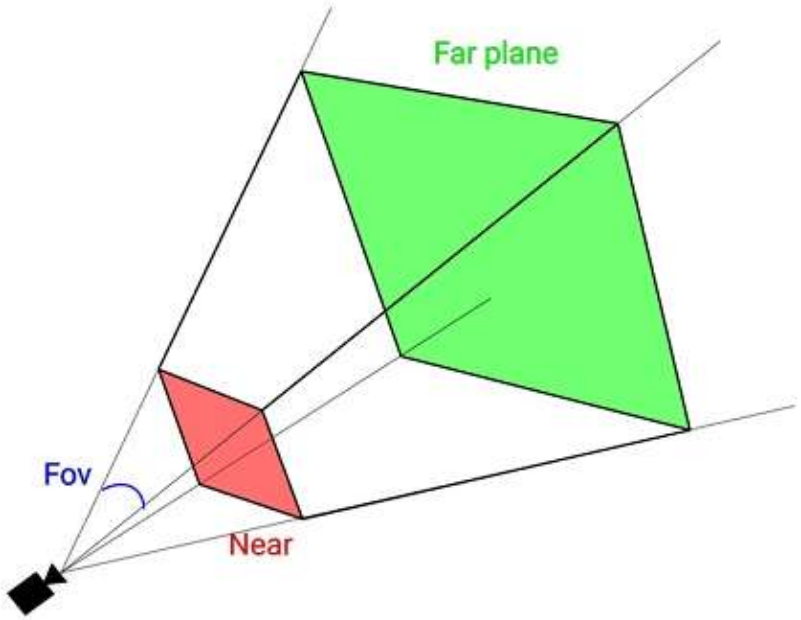
Spot Light

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.

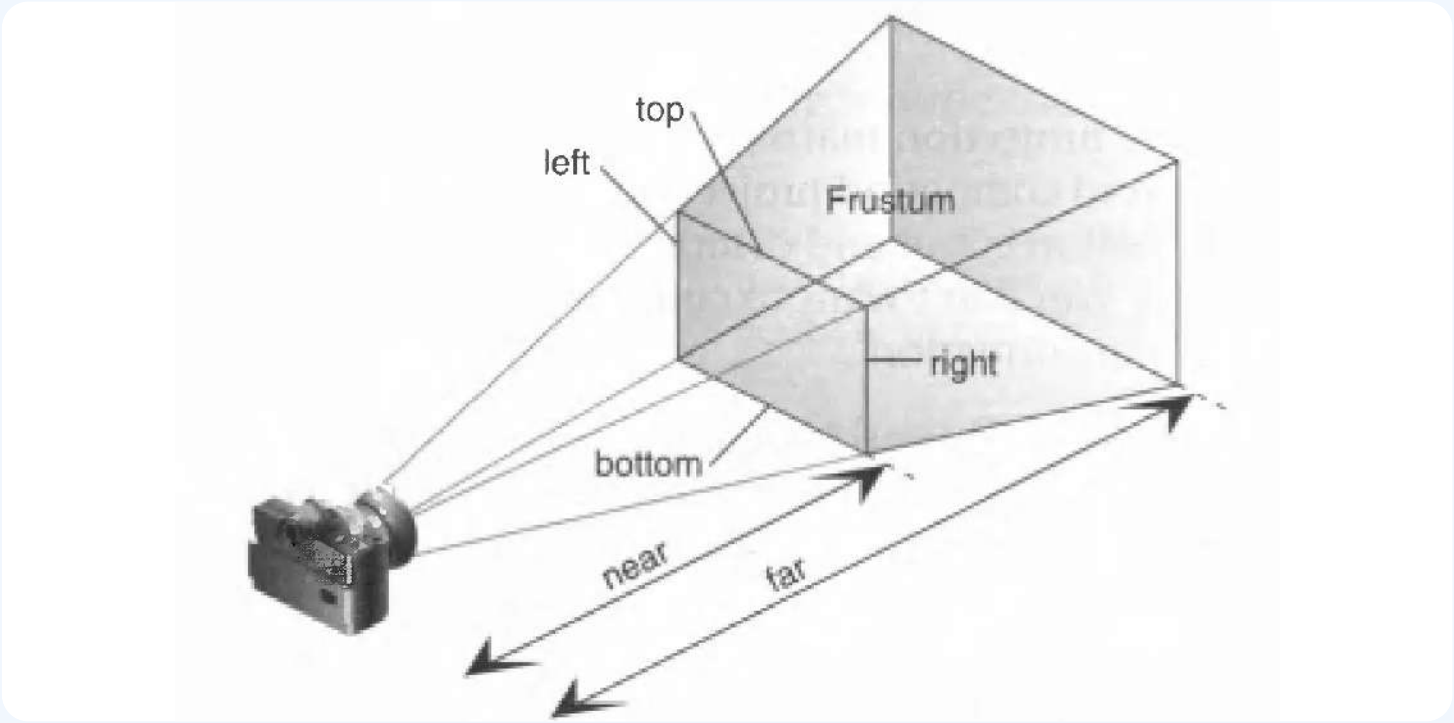


Types of Camera

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

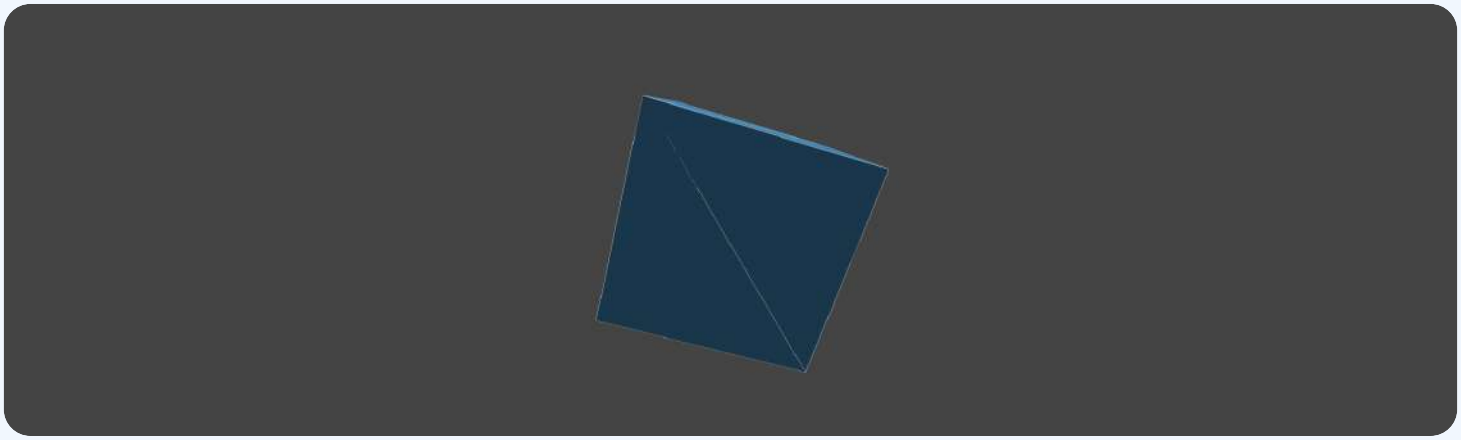
Three.js provides a set of built-in basic geometries that you can use to create common shapes such as cubes, spheres, cylinders, planes, and toruses.

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

Geometries

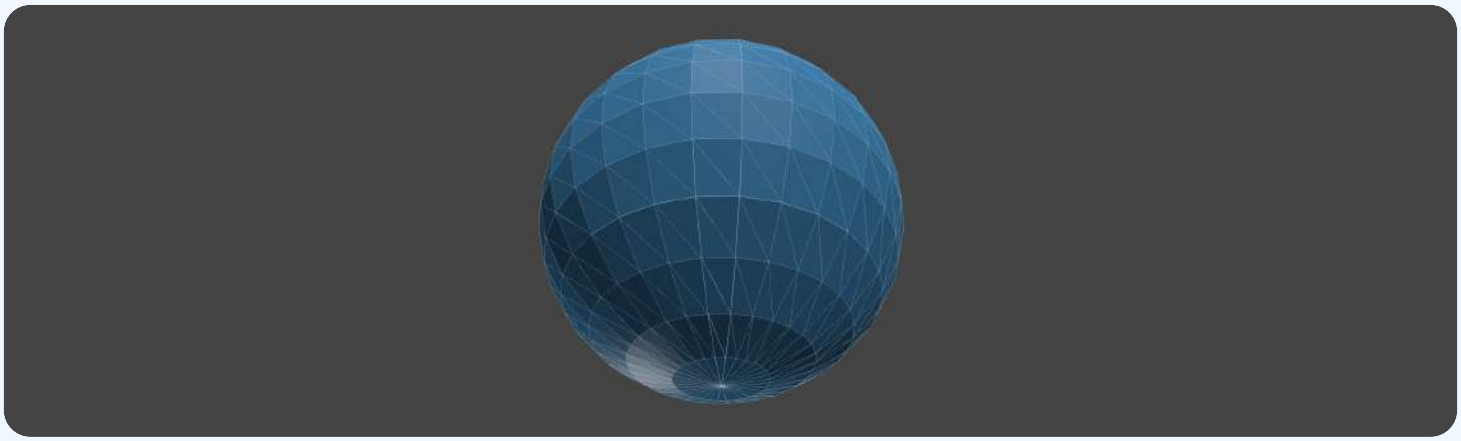
BoxGeometry

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



SphereGeometry

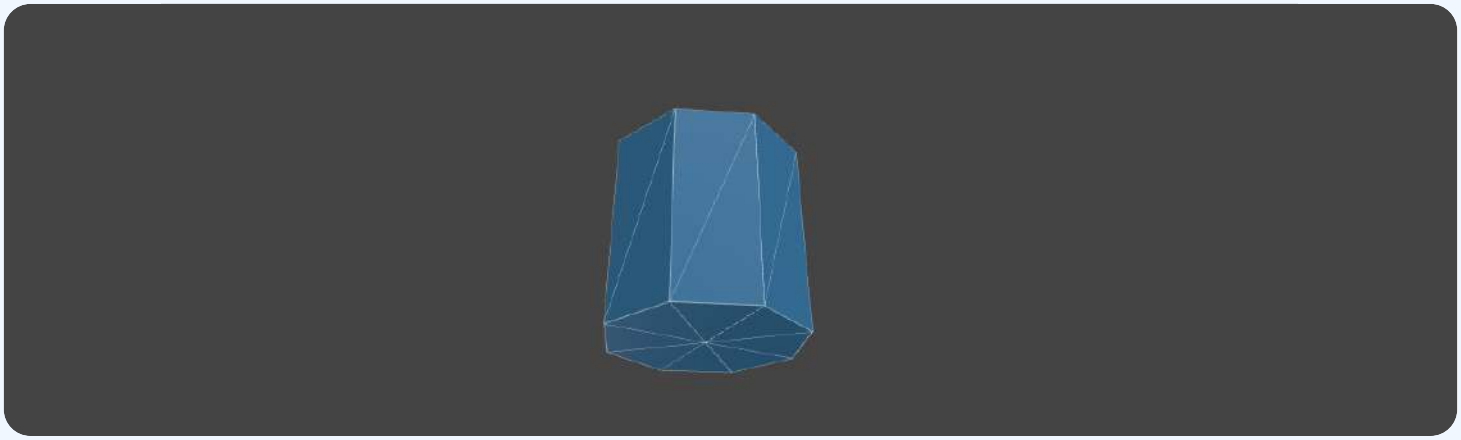
Generates a spherical geometry with a specified radius and number of segments for smoothness.



Geometries

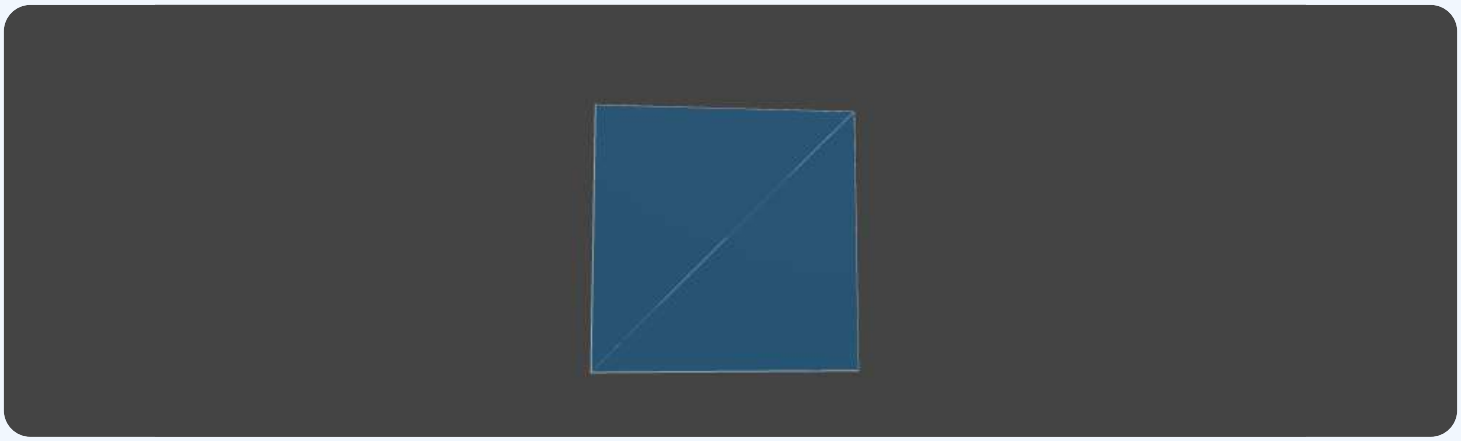
CylinderGeometry

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



PlaneGeometry

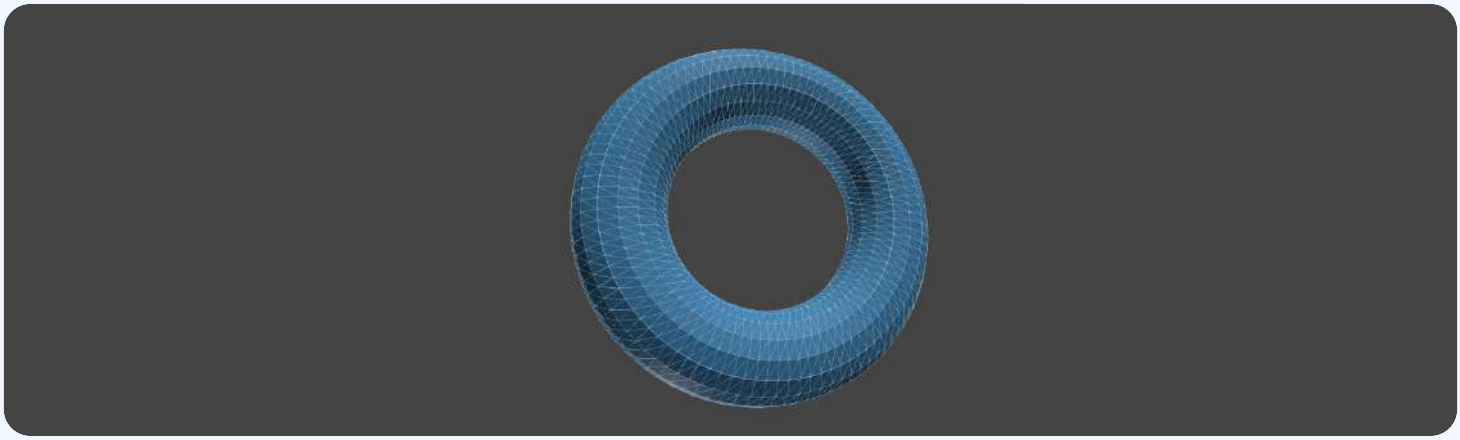
Represents a flat, rectangular plane with specified width and height.



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.

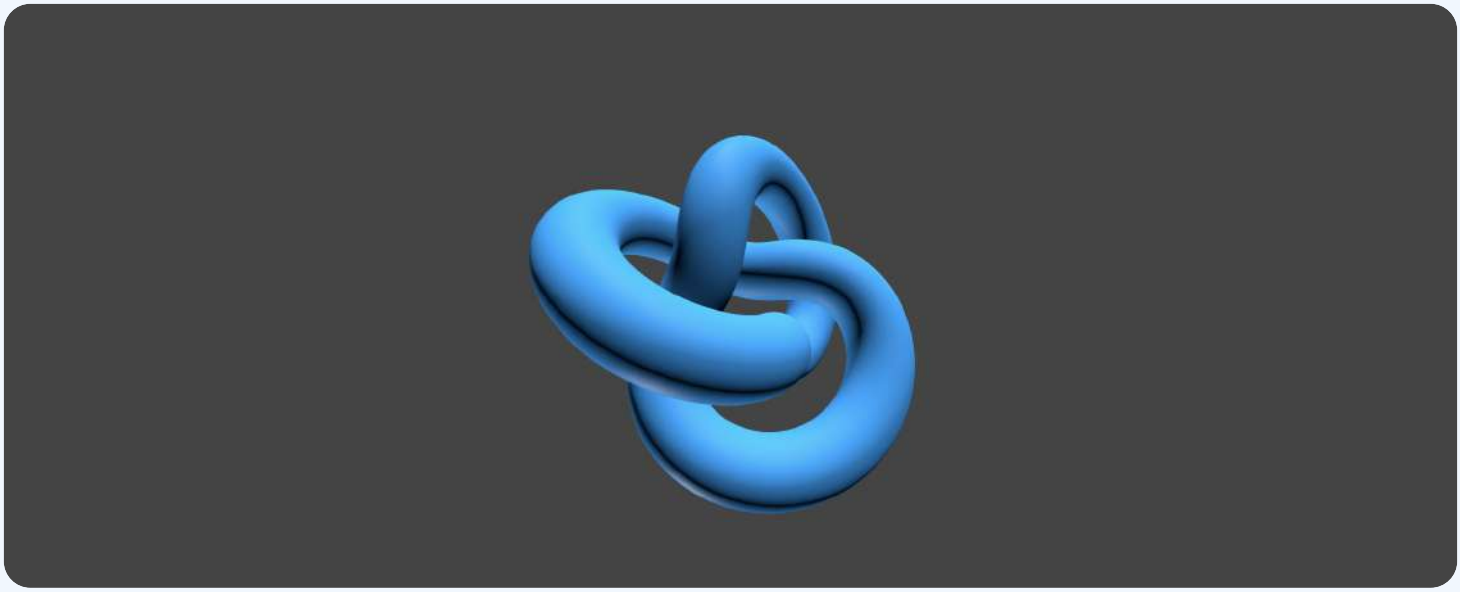
You might use MeshBasicMaterial for creating wireframe objects or objects with flat colors.

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.

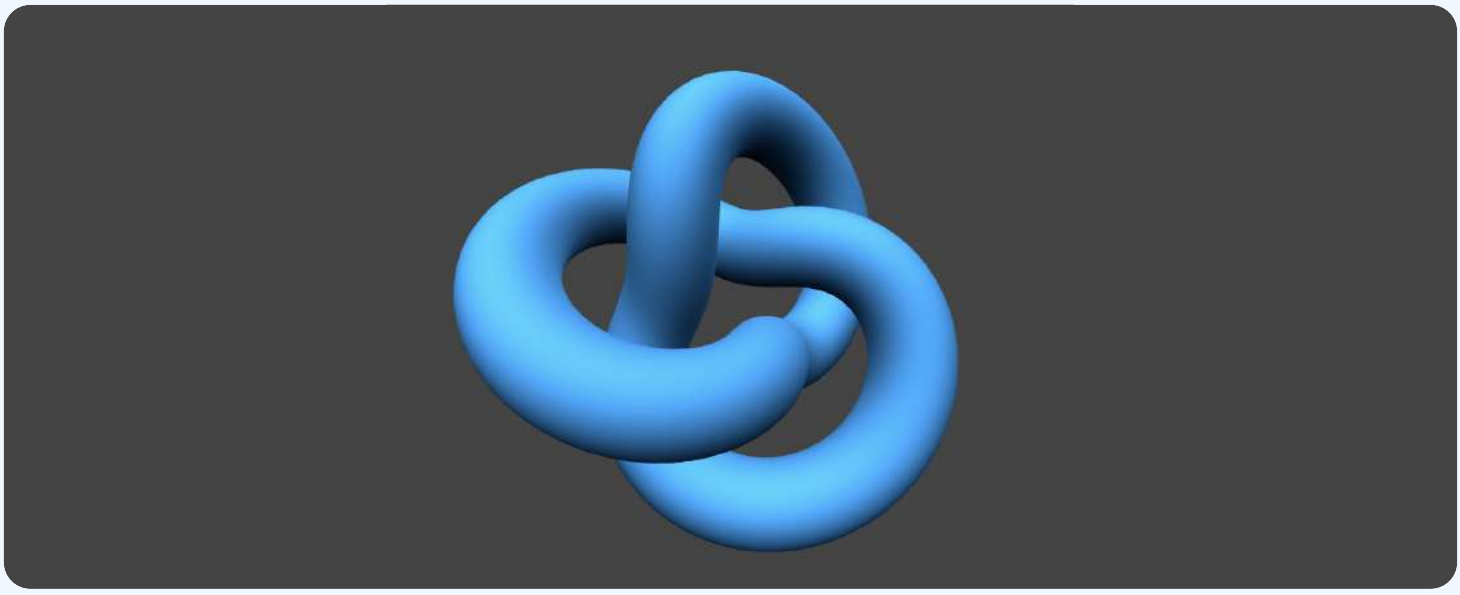


Materials

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:

```jsx
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:

```
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/

The End

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

If you're craving a more personalized learning experience with the guidance of expert mentors, we have something for you — **Our Masterclass**.

JSM Masterclass Experience

In this special program, we do not just teach concepts – offering hands-on training, workshops, one on one with senior mentors, but also help you build production-ready applications in an industry-like environment, working alongside a team and doing code reviews with mentors. It's almost a real-world experience simulation, showcasing how teams and developers collaborate.

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! 🚀