Understanding the Issue

The primary issue with the code snippets you provided is that they're not directly compatible with Angular's component-based architecture. Angular components have specific lifecycle hooks and rendering mechanisms that need to be followed.

Creating a Working Example

Here's a more Angular-friendly approach, leveraging the AfterViewInit lifecycle hook to ensure the 3D scene is initialized and rendered after the component's view has been initialized:

HTML Template (your-component.html):

HTML

```
<canvas #canvas></canvas>
```

TypeScript Component (your-component.ts):

TypeScript

```
import { Component, AfterViewInit, ElementRef, ViewChild } from
'@angular/core';
import * as THREE from 'three';

@Component({
  selector: 'app-your-component',[1]
  templateUrl: './your-component.html',
  styleUrls: ['./your-component.css']
})
export class YourComponent[2] implements AfterViewInit {
  @ViewChild('canvas')
  private canvasRef!: ElementRef;

  private scene!: THREE.Scene;
  private camera!: THREE.PerspectiveCamera;
  private renderer!: THREE.WebGLRenderer;[3]

  ngAfterViewInit() {
    const canvas = this.canvasRef.nativeElement;
```

```
    this.scene = new THREE.Scene();
    this.camera = new THREE.PerspectiveCamera(75, canvas.clientWidth /
canvas.clientHeight, 0.1, 1000);
    this.renderer = new THREE.WebGLRenderer({ canvas });

    // Add objects to the scene, e.g., a cube
    const geometry = new THREE.BoxGeometry(1, 1, 1);
    const material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
    const cube = new THREE.Mesh(geometry, material);
    this.scene.add(cube);

    this.animate();
  }

  animate() {
    requestAnimationFrame(() => this.animate());

    // Update objects, e.g., rotate the cube
    cube.rotation.x += 0.01;
    cube.rotation.y += 0.01;

    this.renderer.render(this.scene, this.camera);
  }
}
```

Explanation:

1. **ViewChild Decorator:** We use the @ViewChild decorator to access the canvas element in the ngAfterViewInit hook.
2. **AfterViewInit Hook:** This hook is called after the component's view is initialized, so we can safely access the canvas element and create the 3D scene.
3. **Creating the Scene, Camera, and Renderer:** We create these objects as class properties to ensure they're accessible within the animate function.
4. **Adding Objects to the Scene:** We create a simple cube as an example. You can add more complex objects as needed.
5. **Animation Loop:** The animate function is called recursively using requestAnimationFrame to create a smooth animation. Inside the loop, we update the cube's rotation and render the scene.

Key Points:

- **Component Lifecycle Hooks:** Use AfterViewInit to ensure the 3D scene is initialized after the view is ready.
- **Canvas Element Access:** Use @ViewChild to access the canvas element and set the renderer's canvas property.
- **Animation Loop:** Use requestAnimationFrame for smooth animation.
- **Object Updates:** Update object properties within the animate function to create dynamic

effects.

- **Scene Rendering:** Use renderer.render(scene, camera) to render the scene.

By following these steps, you can effectively integrate three.js into your Angular application and create interactive 3D experiences.

Sources

1. https://github.com/nichelia/nichelia.com
2. https://stackoverflow.com/questions/72238590/path-variable-to-use-in-data-in-angular
3. https://github.com/SoWieMarkus/TrackmaniaCOTDWebsite
4. https://github.com/SCIInstitute/dSpaceX
5. https://github.com/Rogerpeke97/portfolio-nextjs
6. https://blog.expo.io/introducing-expo-ar-mobile-augmented-reality-with-javascript-powered-by-arkit-b0d5a02ff23
7. https://areknawo.com/building-3d-2048-game-with-vue-and-three-js-setup/