Exploring WebGL with Three.js library

Abstract:

Three.js is a popular JavaScript library that simplifies the creation and manipulation of 3D graphics using WebGL. This paper will explore the fundamentals of WebGL, the benefits of using Three.js, and the remarkable projects that have been created using this library. I will discuss the advantages of Three.js over other WebGL libraries and provide insights into the library's background, features, and documentation. Through this exploration, I'll demonstrate the power and versatility of Three.js in creating stunning and interactive 3D experiences for web applications.

All the codes can be found on my GitHub (https://github.com/denyshubh/cs544-paper) with comprehensive Readme. I have also deployed the Portal App on my Azure VM. You can access the ***Live App*** at https://cs544.mahwish.me/

1. Introduction:

The growing demand for immersive and interactive web experiences has led developers to seek out tools that can simplify the process of creating 3D graphics. **WebGL**, a JavaScript API for rendering 2D and 3D graphics within compatible web browsers, has emerged as a powerful solution. **However, the complexity of working with shaders, matrices, and other low-level aspects of WebGL can be challenging for many developers.**

Three.js is a JavaScript library built on top of WebGL to provide a more accessible and user-friendly approach to 3D graphics. Created by Ricardo Cabello (aka Mr.doob) and maintained by a large community of contributors, Three.js aims to significantly reduce the amount of code and effort required to create complex 3D scenes. The library offers an impressive range of features, extensive documentation, and a multitude of examples to help developers get started.

This paper will delve into the core concepts of WebGL and demonstrate the benefits of using Three.js for 3D web development. I will provide links various projects created using Three.js, such as interactive websites, games, and virtual reality experiences, to highlight the capabilities and potential of this versatile library.

2. WebGL Fundamentals

what is WebGL ?

WebGL (Web Graphics Library) is a JavaScript API designed for rendering interactive 2D and 3D graphics within web browsers. Some key features of WebGL include:

- Rendering Triangles at ***remarkable speed***
- Drawing results on a `<canvas>` HTML element (though not necessary)
- Utilizing GPU power to render 3D objects on web browsers

We can also use WebGL for 2D animations, but there are better libraries like ***PixiJS*** and ***GSAP*** libraries for this purpose.

The role of the GPU in rendering 3D graphics

To draw 3D models, the idea is to draw many triangles at the right position and colorize them to achieve the desired appearance.

CPU vs GPU:

- CPU - Can run instructions one by one (on a single core) but is very fast
- GPU - Can run thousands of parallel calculations but not as fast as the CPU

The GPU positions all points at once and then draws each visible pixel of those triangles. This parallel processing approach makes the rendering process extremely fast.

Shaders, matrices, and their challenges in native WebGL

The instructions to place the points and draw pixels are written in Shaders. We provide transformations to these shaders, such as:

- Points position,
- Model transformation,
- The camera coordinates, etc

These transformations help position and colorize the elements as needed.

Why do we need Three.js?

Drawing a single triangle on the canvas using native WebGL will take at least 100 lines of code! This complexity is why we need Three.js to simplify the process and make it more accessible for developers.

3. Three.js: An Overview

Three.js is a powerful JavaScript library created by Ricardo Cabello, also known as Mr.doob, to simplify the process of creating and manipulating 3D graphics using WebGL. The library was first released in April 2010 and has since been continuously developed and improved by a large community of contributors.

Key features and advantages over native WebGL

Some of the key features and advantages of Three.js over native WebGL include:

- ***Simplified API:*** Three.js provides a high-level API that abstracts the complexity of native WebGL, making it easier for developers to create 3D graphics.
- ***Rich set of features:*** Three.js comes with built-in support for a wide range of features such as geometries, materials, lighting, animation, and user interaction, reducing the need for custom code.
- ***Cross-browser compatibility:*** Three.js ensures compatibility across different web browsers, allowing developers to focus on creating their 3D scenes without worrying about browser-specific issues.
- ***Extensive documentation:*** Three.js has well-maintained and comprehensive documentation, making it easier for developers to learn and use the library effectively.

Community support, updates, and contributions

Three.js has a large and active community of developers who contribute to the library's development, report issues, and share their knowledge with others. The library is updated regularly, with new features, bug fixes, and improvements being added every month.

4. Getting Started with Three.js

In this section, we will explore the process of setting up a basic 3D scene using Three.js, understand the core components involved, and learn how to add materials, lighting, and textures. ***SOURCE CODE*** I have also deployed the Portal App on my Azure VM. You can access the ***Live App*** at https://cs544.mahwish.me/

The core components

- ***Scene:*** The scene acts as a container for all the objects, models, particles, lights, and other elements in the 3D scene. Create a scene using the `THREE.Scene()` class.
- ***Geometry:*** Geometries define the shape of 3D objects. Create a cube's geometry using the `THREE.BoxGeometry()` class.
- ***Material:*** Materials determine the appearance of 3D objects. Create a material with a specified color using the `THREE.MeshBasicMaterial()` class.
- ***Mesh:*** Meshes combine geometry and material to create 3D objects. Combine the geometry and the material using the `THREE.Mesh()` class and add the resulting mesh to the scene.
- ***Camera:*** Cameras define the point of view from which the scene is rendered. Create a camera using the `THREE.PerspectiveCamera()` class, specifying the field of view and aspect ratio, and add it to the scene.
- ***Renderer:*** The renderer draws the 3D scene on a canvas element in the HTML file. Create a renderer using the `THREE.WebGLRenderer()` class, specify the canvas element, and set the size of the renderer.

Adding materials, lighting, and textures

In this basic setup, I have already added a material to the cube. To enhance the scene, we can further explore adding different materials, lighting, and textures. These advanced features will allow us to create more realistic and visually appealing 3D scenes using Three.js.

After setting up the basic 3D scene we can render the scene by calling the `render()` method on the renderer, passing in the scene and the camera as parameters. Adjust the position of the camera along the z-axis to move it backward and make the cube visible. This setup provides a foundation for more advanced features such as animations, lighting, and textures.

5. Advanced Techniques and Interactivity

I learned some of these advance techniques from three.js course. These are used in ***Advance Project*** created For this Paper. These techniques will enrich our creations with realistic physics, optimized performance, and custom models, giving us more control over your 3D web experiences.

1. ***Adding Physics :*** Incorporating physics into our WebGL projects can make them more immersive and interactive. Users delight in manipulating objects and witnessing collisions, collapses, falls, and bounces that mimic real-world behavior, as demonstrated in this website https://bruno-simon.com/ To add physics to our project, we can create custom solutions using mathematics and tools like ***Raycaster***. We can also use external physics libraries if we want realistic physics with tension, friction, bouncing, constraints, pivots, and more in 3D space with three.js

2. ***Baking:*** Baking is an optimization technique that precomputes lighting and other computationally expensive effects, resulting in a texture map that can be applied to our 3D models. This process ***reduces the computational load during runtime***, allowing for smoother performance and faster rendering. Baking is particularly useful for static scenes or objects that do not require real-time lighting updates, as it allows for complex lighting effects without sacrificing performance.

3. ***Creating Custom Models:*** To personalize our 3D projects, we can create custom models using industry-standard software like Cinema 4D, Maya, 3DS Max, Blender, ZBrush, Marmoset Toolbag, and Substance Painter. These tools provide advanced features for modeling, texturing, and animating 3D assets, giving you the freedom to craft unique experiences tailored to our vision.

By incorporating these advanced techniques into our WebGL and Three.js projects, we'll be able to create sophisticated, engaging, and performant 3D web experiences that truly stand out.

Use Cases

WebGL and Three.js enable a wide range of use cases, providing 3D graphics and interactive experiences on the web. Some notable use cases include:

- ***Virtual Reality (VR) and Augmented Reality (AR):*** WebGL and Three.js facilitate the creation of immersive VR and AR experiences directly in web browsers, without the need for additional plugins or installations.

- ***Data Visualization:*** By leveraging the power of 3D graphics, complex datasets can be visualized in innovative and intuitive ways, allowing users to interact with and better understand the information presented.

- ***3D Product Showcases:*** Online retailers and businesses can showcase their products in 3D, offering customers a more engaging and realistic view of items before they make a purchase.

- ***Architectural Visualization:*** Architects and designers can use WebGL and Three.js to create interactive 3D models of buildings and spaces, allowing clients to explore designs in a virtual environment.

- ***Gaming:*** Developers can create rich, interactive 3D games that run directly in web browsers, providing users with a seamless gaming experience without the need for additional software installations.

- ***Educational Tools:*** 3D simulations and interactive learning tools can be developed using WebGL and Three.js, enhancing the learning experience for students and educators alike.

Reference

1. Bruno Simon - https://threejs-journey.com/ This is the place where I learned about WebGl and Three.js library. My projects are part of this learning
2. WebGl Docs : https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API
3. Three.js Docs - https://threejs.org/docs/ : The basic code implementation was created from this docs.