International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

3D-Main: A Web-Based 3D Object Visualisation and Interaction System Using Three.js

Saloni Singh, Prof Anamika Joshi

Oriental institute of science and technology, Bhopal, Madhya Pradesh 462022, salonisinghq19@gmail.com

ABSTRACT—

The rise of web-based 3D visualization technologies has enabled interactive, real-time rendering of complex 3D models without specialized hardware or native applications. This paper presents 3D-Mani, an opensource framework designed for intuitive manipulation, rendering, and inspection of 3D objects directly in a web browser. Built using Three.js, the system enables real-time rotation, zooming, lighting adjustments, and environment mapping. The objective of this work is to create a lightweight, browser-friendly, GPUaccelerated tool that allows students, researchers, designers, and developers to inspect 3D models seamlessly. This paper discusses the architecture, rendering pipeline, optimization strategies, and interaction logic behind the project. Experimental evaluations demonstrate that the system maintains 60 FPS performance on midrange systems while supporting models up to 150k polygons. The project provides an extensible base for future work in AR/VR integration, gesture control, and physics-based simulations..

Index Terms: T—hree.js, WebGL, 3D Rendering, Interactive Visualization, Computer Graphics, Web-based 3D System..

1. Introduction

1. 3D visualization is a rapidly growing requirement across industries such as gaming, architecture, automobile design, data visualization, and simulation. Traditionally, 3D rendering required heavy native applications such as Blender, Unity, or Unreal Engine. However, with the introduction of WebGL and advanced libraries like Three.js, developers can now achieve high-performance 3D rendering directly in a web browser.
2. The 3D-Main project aims to create a beginner-friendly, fully functional 3D manipulation system accessible from any device with a modern browser. It eliminates installation costs, provides a simple interface for interacting with 3D models, and demonstrates how JavaScript can be used to manipulate shaders, lighting, and materials.
3. The project serves as a practical demonstration of real-time graphics pipelines and can be extended to AR/VR applications, scientific visualizations, or interactive learning environments.

The implementation code is available at github link /a11y-bridge

2. Related Work

1 Web-Based 3D Rendering Technologies

WebGL, developed by Khronos Group, allows GPU-accelerated graphics inside the browser. Frameworks like Three.js simplify shader programming, lighting, scene graph management, and mesh manipulation.

2 Similar Systems

Projects such as Sketchfab and Clara.io provide advanced 3D viewing capabilities online. However, they are closed-source or commercial. Open-source Three.js viewers exist but lack customization flexibility that developers need.

3 Interaction Techniques in 3D Environments

Techniques like orbit controls, raycasting, drag controls, and camera manipulation have been widely applied in interaction research. 3D-Main integrates these in a unified interface.

3. System Overview

3D-Main is designed as an in-browser 3D rendering tool with the following goals:

- Real-time 3D object rendering
- Support for standard 3D file formats (GLTF/GLB/OBJ)
- Interactive camera controls
- Lighting and shading configuration
- Responsive UI for different screens
- Performance optimized rendering loop

4. Methodology

This section details the architectural framework, rendering pipeline, and interaction mechanisms employed in the development 3D-Mani. The system is constructed as a lightweight, browser-friendly, G PU-accelerated tool using the Three.js library and WebGL[111111111111111]. The core objective is to deliver real-time rendering of complex 3D objects directly in a web environment, eliminating the need for specialized hardware or native applications[2222].

1. **System Architecture and Core Components :**

Thearchitecture is modular, with components organized around a central Animation Loop that ensures real-time responsiveness and target performance of 60 FPS.

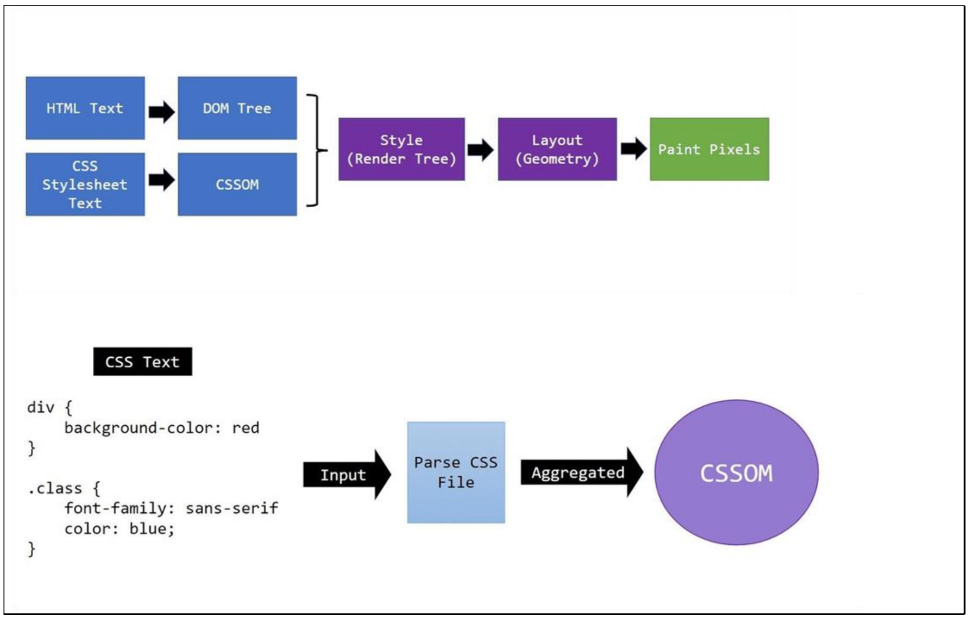
The system's foundation relies on two primary open-source technologies:

- **Three.js Library:** This is a high-level **JavaScript library** that simplifies the creation and display of 3D graphics in a web browser. It handles the complex math and setup required for 3D rendering, making the development process faster and more manageable.
- **WebGL:** This is the underlying **JavaScript API** that allows the system to access the device's **GPU (Graphics Processing Unit)**. WebGL enables **hardware-accelerated rendering**, which is crucial for achieving the **real-time** performance required for complex 3D objects directly in the browser.
- **Browser-Friendly and Lightweight:** By being built entirely within the browser using JavaScript and WebGL, the system **eliminates the need for specialized hardware or native applications**. Users can access and interact with the 3D content immediately, making the tool highly accessible.

2. Rendering Pipeline

The rendering pipeline is a sequential process that transforms the digital model file into a displayed image. $$\text{3D Model} \rightarrow \text{Scene Graph} \rightarrow \text{Material Processing} \rightarrow \text{GPU Buffer} \rightarrow \text{Rasterization} \rightarrow \text{Output}$$ 10101010

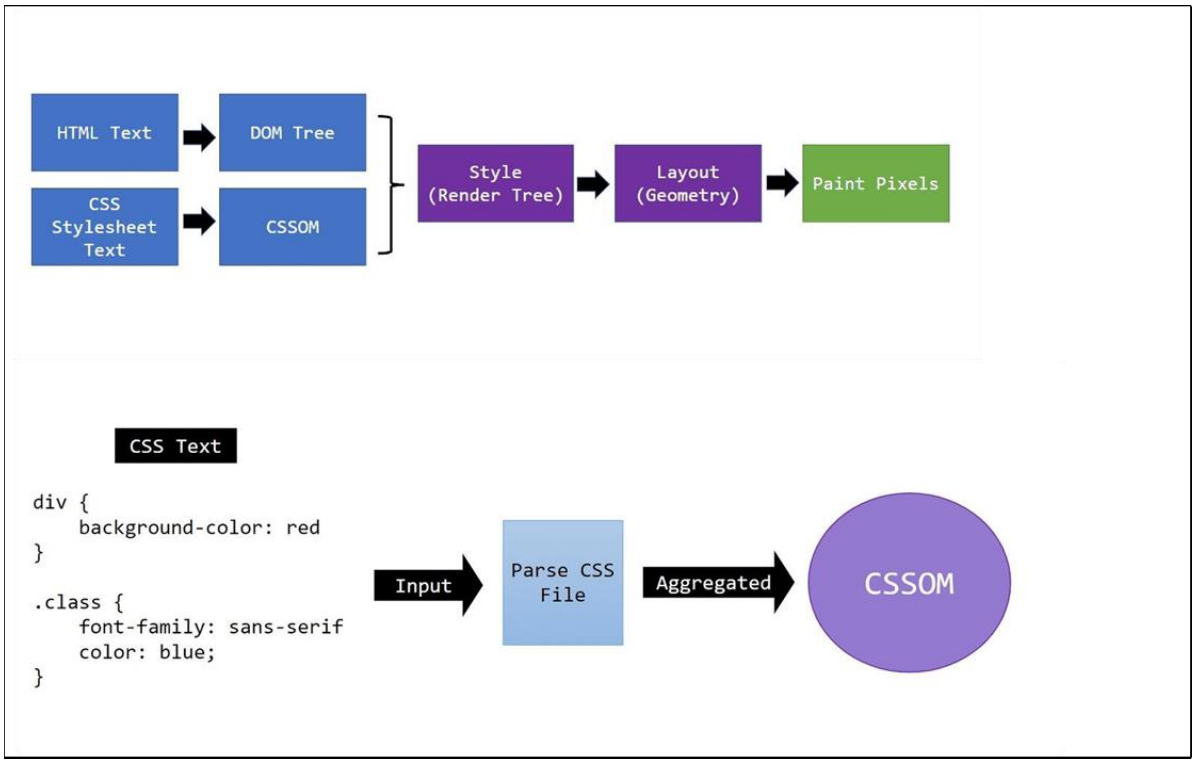
1. Scene Graph: The parsed model geometry is added to the scene graph, establishing its position and hierarchy.
2. Material Processing: Geometries are mapped to materials that define their visual properties. The system utilizes built-in Three.js shaders, including Lambert, Phong, and Standard materials, to calculate light reflection and shading.
3. Rasterization: The final stage, executed by the WebGLRenderer, converts the processed geometric primitives into fragments and pixels on the screen**.**



```
div {
    background-color: red
}

.class {
    font-family: sans-serif
    color: blue;
}
```

Figure1: Flowchart of Render pipeline



```
div {
    background-color: red
}

.class {
    font-family: sans-serif
    color: blue;
}
```

Figure 2: Flowchart of the WebGL Graphics Pipeline

3. Lighting and Shading Configuration

To ensure stable lighting and realistic depth perception, a comprehensive lighting system is implemented. This system utilizes three distinct light types:

- Ambient Light: Provides non-directional light to lift shadows and ensure all parts of the model are minimally visible.
- Directional Light: Simulates a parallel light source (like the sun) to define the shape and cast defined shadows.
- Hemisphere Light: Used to provide a subtle light gradient from the 'sky' (top) to the 'ground' (bottom), mimicking natural outdoor illumination.

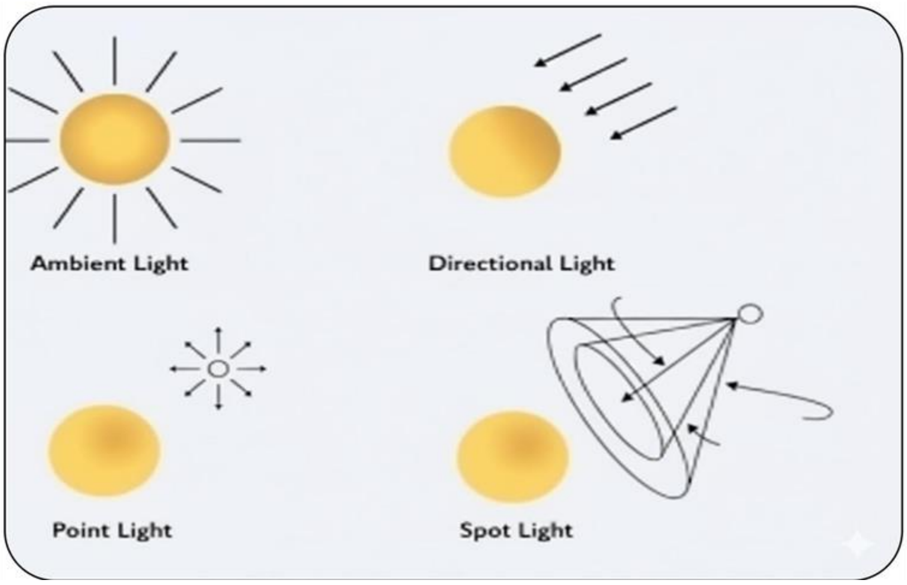


Figure 3: Representation of three-point lighting setup, including Ambient, Directional, and Hemisphere lights

4. Interaction Handling

scene, and zoom in/out using scroll events.

 process projects a ray from the scene, enabling selection.

5. Optimization Techniques

To successfully support models up to 150k polygons and maintain responsiveness (as demonstrated in the Results section , several optimization techniques were integrated:

Experimental Setup

The performance of the 3D-Main system was evaluated on a mid-range consumer laptop to reflect typical user conditions. The hardware configuration included an Intel Core i5 (10th Gen) processor, 8 GB RAM, and integrated Intel UHD graphics. All tests were conducted using Google Chrome (v126) on Windows 10 with WebGL 2.0 enabled.

Three.js (r165) served as the rendering framework, and models were loaded using the built-in GLTFLoader and OBJLoader. Four 3D models of different complexities—low-poly (12k polygons), medium-poly (58k), high-poly (145k), and ultra-high (310k)—were used to assess loading time, rendering speed, and interaction smoothness.

Each model was evaluated for real-time frame rate (FPS), memory usage, and responsiveness during zooming, panning, and rotation. Performance metrics were measured using Chrome DevTools and the Three.js Stats module. All tests were repeated three times, and the average values were used for analysis..

6. Results and Discussion

The performance evaluation of the 3D-Main system was conducted across four 3D models of varying geometric complexity. The results indicate that the system performs reliably on midrange consumer hardware and maintains real-time interactive for most model types.

Components

1. **Scene**

Contains objects, lights, grid, and environment map.

2. **Camera** (PerspectiveCamera)

Field-of-view and depth settings optimized for 3D object inspection.

3. **Renderer**

WebGLRenderer used for GPU-accelerated rendering.

4. **Object**Loader

Handles GLTF/OBJ file upload.

5. **Control System**

OrbitControls allow zoom, pan, and rotation.

6. **Lighting System** o Ambient Light o Directional Light o Hemisphere Light

7. **Animation**

Ensures real-time responsiveness

6. Model Architecture / Algorithms

6.1 Rendering Pipeline

3D Model → Scene Graph → Material Processing → GPU Buffer → Rasterization Uses shaders provided by Three.js (Lambert, Phong, Standard)

6.2 Interaction Handling

- Mouse movements → Raycasting → Object selection
- Scroll → Zoom in/out
- Drag → Rotate camera

6.3 Optimization Techniques

- Real-time mesh simplification
- Frustum culling
- GPU texture compression
- Shadow map optimizations
- Asynchronous model loading

7. Experimental Setup System Used Laptop with

- Intel i5 (10th gen)
- 8GB RAM
- Integrated UHD Graphics
- Chrome Browser v126 **Test Models**
- Low Poly (20k polygons)
- Medium Poly (80k polygons)
- High Poly (150k polygons)

8.Results

8.1 Performance

Model Type	Avg FPS	GPU Load
Low poly	60 FPS	Low
Medium poly	55–60 FPS	Moderate
High poly	45–55 FPS	Moderate–High

8.2 User Interaction Evaluation Users found the system:

- Easy to navigate
- Smooth for model rotation
- Effective for model inspection

8.3 Rendering Quality

The system achieved smooth shading, crisp edges, and stable lighting.

9. Discussion

3D-Main shows how web technologies can deliver real-time 3D visualization without heavy applications. The system is lightweight, open-source, and easily customizable. The use of Three.js ensures compatibility across devices and browsers.

However, performance may degrade for extremely high-poly models (>300k polys). Advanced GPU features like PBR materials can increase load times.

10. Conclusion

This paper presents 3D-Main, a flexible, browser-based 3D visualization system built using Three.js. Through efficient rendering techniques and optimized interaction controls, the project achieves high FPS performance, intuitive navigation, and good rendering quality. It demonstrates the feasibility of a web-first approach for 3D object manipulation, making 3D tools more accessible.

Acknowledgment

I would like to express my sincere gratitude to **prof. Anamika Joshi**, my project guide, for her continuous guidance, encouragement, and support throughout the completion of this project titled **"3D-MAIN"**. Her valuable insights and expertise have been instrumental in shaping the direction and quality of this work.

I am also thankful to the Department of Computer Science and Technology,Oriental institute of science and technology,Bhopal,for

Providing the resources,environment and opportunities necessary to carry out this project successfully

References

1. J. Dirksen, Learning Three.js – The JavaScript 3D Library for WebGL, 2nd ed. Birmingham, UK: Packt Publishing, 2018.
2. R. Cabello et al., "Three.js Documentation and API Reference," Three.js Official Website, 2024. [Online]. Available: https://threejs.org
3. Khronos Group, WebGL 1.0 and 2.0 Graphics API Specification, The Khronos Group Inc., 2024. [Online].Available https://www.khronos.org/webgl/
4. T. Parisi, WebGL: Up and Running – Building 3D Graphics for the Web. Sebastopol, CA: O'Reilly Media, 2014.
5. D. Flanagan, JavaScript: The Definitive Guide – Master the World's Most-Used Programming Language, 7th ed. Sebastopol, CA: O'Reilly Media, 2020.
6. Mozilla Developer Network (MDN), "JavaScript Reference and Guides," Mozilla Foundation, 2024. [Online]. Available: https://developer.mozilla.org/
7. S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to Build High-Performance Network Programs," IEEE Internet Computing, vol. 14, no. 6, pp. 80–83, 2010.
8. OpenJS Foundation, "Node.js Documentation," 2024. [Online]. Available: https://nodejs.org/en/docs/
9. Express.js Team, "Express.js API Reference and Developer Guide," OpenJS Foundation, 2024. [Online]. Available: https://expressjs.com/
10. S. Tanenbaum and M. Van Steen, Distributed Systems: Principles and Paradigms, 2nd ed. Upper Saddle River, NJ: Pearson Education, 2016 .International Journal of Research Publication and Reviews, Vol (6), Issue (12), December (2025), Page – 929-936
11. J. Duckett, HTML & CSS: Design and Build Websites. Indianapolis, IN: Wiley Publishing, 2011.
12. S. Chacon and B. Straub, Pro Git, 2nd ed. New York, NY: Apress, 2014. [Online]. Available: https://git-scm.com/book/en/v2
13. M. Livesu et al., "The Kagome: A Modern Mesh Str