Winter Fellowship Report

On

Web Based CAD Model Converter and Renderer

Submitted by

Aditya Mavle

Under the guidance of

Prof. Sidhartha Ghosh
Civil Engineering Department
IIT Bombay

Under the mentorship of

Danish Ansari
Software Engineer
Osdag,FOSSEE
IIT Bombay

Nagesh Karmali
Senior Manager(Research)
Department of Computer Science Engineering
IIT Bombay

August 17, 2023

Acknowledgment

I would like to express my deep and sincere gratitude to my guide Prof. Siddhartha Ghosh who has provided all the required resources for the successful completion of this internship. I thank Mr. Danish Ansari and Mr. Nagesh Karmali for offering me valuable suggestions throughout the internship. They have provided valuable mentorship and a sense of direction throughout. Without their patient instruction, insightful criticism and expert guidance, the completion of this report would not have been possible.

I am also thankful to everyone at FOSSEE who was involved in the selection process based on the screening tasks, for providing me a platform to work on and an opportunity to be a part of the team which promotes open-source software.

Contents

List of Figures

Chapter 1

Introduction

1.1 FOSSEE Internship

Osdag internship is provided under the FOSSEE project. FOSSEE project promotes the use of FOSS (Free/Libre and Open-Source Software) tools to improve the quality of education in our country. FOSSEE encourages the use of FOSS tools through various activities to ensure the availability of competent free software equivalent to commercial (paid) software. The FOSSEE project is a part of the National Mission on Education through Information and Communication Technology (ICT), Ministry of Education, Government of India. Osdag is one such open-source software that comes under the FOSSEE project. Osdag internship is provided through the FOSSEE project. Any UG/PG/Ph.D. holder can apply for this internship. And the selection will be based on a screening task.

1.2 What is Osdag

Osdag is Free/Libre and Open-Source Software being developed for the design of steel structures following IS 800:2007 and other relevant design codes. OSDAG helps users in designing steel connections, members and systems using interactive Graphical User Interface (GUI). The source code is written in Python, 3D CAD images are developed using PythonOCC. GitHub is used to ensure smooth workflow between different modules and team members. It is in a path where people from around the world would be able to contribute to its development. FOSSEE's "Share alike" policy would improve the standard of the software when the source code is further modified based on the industrial and educational needs across the country. Design and Detailing Checklist (DDCL) for different connections, members and structure designs is one of the main products of this project. It would create a repository and design guidebook for steel construction based on Indian Standard codes and best industry practices

1.3 What is Osdag on Web

Osdag on Web is the web version of Osdag currently under development.It will feature all the functionalities of Osdag and the user would be able to use it without any necessary software installation.The web source code is written in, additional to the initial stack,Django-REST framework for backend and APIs,ReactJs for the frontend and ThreeJS/React-Three-Fiber for the rendering of CAD models on the Web.

1.4 Who can use Osdag?

Osdag is primarily created for use in academia for students and teachers but industry professionals also find it useful. As Osdag is currently funded by MHRD, the Osdag team is developing software in such a way that it can be used by the students during their academics and to give them a better insight look in the subject. Osdag can be used by anyone starting from novice to professionals. Its simple user interface makes it flexible and attractive than other software. Video tutorials are available to help get started. The video tutorials of Osdag can be accessed here.

- The video tutorials of OSDAG can be easily accessed from https://osdag.fossee.in/resources/videos or YouTube.
- The sample design problems for different modules can be viewed from https://osdag.fossee.in/resources/sample-design
- One can view the user tools used for the development of OSDAG from https://osdag.fossee.in/resources/user-tools
- OSDAG can be downloaded from https://osdag.fossee.in/resources/downloads
- OSDAG on Web codebase(develop branch) https://github.com/osdag-admin/Osdag-web/tree/develop

Chapter 2

Building the WebGL Based CAD Model Renderer

2.1 Need of modifying Osdag's Graphics/CAD stack

- Since in the Osdag desktop app, the graphics and CAD rendering is handled by the Python-Open Cascade (OCC) library,we needed a more viable and lightweight alternative to it, so that Osdag's structural models could be rendered on Web.
- Rendering 3D models on the Web also requires WebGL compatibility ,which is a JavaScript API for rendering interactive 2D and 3D graphics.
- Hence we used ThreeJS, which is a cross-browser JavaScript library and application programming interface used to create and display animated 3D computer graphics in a web browser using WebGL.
- Using the core logic and model file generation capabilities of the Python-OCC code to generate model files, there was also a need to implement a CAD file converter that would convert the CAD file from the original OCC-generated formats of

    1. BREP (Boundary Representation)
    2. STEP (Standard for the Exchange of Product Data)
    3. IGES (Initial Graphics Exchange Specification)

  to file formats compatible with ThreeJS's model loaders like:

    1. OBJ (Wavefront OBJect)
    2. GLTF (GL Transmission Format).
- The CAD file converter was implemented using FreeCAD, which is a general-purpose parametric 3D CAD modeler.It is a freeware software which supports the functionalities of writing code macros in it,which enables us to perform various operations on CAD models and files.The CAD Converter will be further explored in Chapter 3.

2.2 Components of the WebGL Canvas and ThreeJS Scene

A Three.js scene is a container that holds all the elements required to create and display a 3D world. It serves as the stage where 3D objects, cameras, lights, and other components are placed and interact.Three.js is an object-oriented library.This means that components of a scene are represented as instances of specific classes, and these classes have methods and properties that define their behavior and characteristics.The objects of these classes have to be instantiated according to the desired characteristics of the scene components.The main components of our ThreeJS scene were:

- Scene: The heart of the 3D environment, the scene is initiated using new THREE.Scene(). This serves as a digital container for all 3D entities within a design. The scene elegantly gathers together a design's intricacies, from objects to lighting, providing a unified canvas for the 3D representation.

- Model Loader: A pivotal tool, model loaders like the OBJLoader and GLTFLoader are instrumental in the incorporation of 3D assets from external modeling software into a Three.js scene. These loaders allow the seamless import of intricate 3D elements, ranging from complex objects to intricate characters and expansive scenes. The OBJLoader and GLTFLoader facilitate the translation of these assets into a format that the Three.js engine can seamlessly render and display.

- Camera: The camera dictates the viewpoint or perspective from which the entire scene is observed. In our use case, the Perspective Camera was initially embraced. This camera mimics the human eye's perception, enhancing the realism of the visual experience. Subsequently, the Orthographic Camera was employed, strategically enabling the representation of various isometric views for objects within the scene. The camera, in essence, is the "eye" through which users perceive the 3D world.

- Renderer: Acting as the translator between 3D models and the pixels displayed on a web page, the renderer is an indispensable component. It harnesses the power of WebGL to take the abstract 3D scene and translate it into a coherent visual. This process involves converting intricate geometry and lighting into tangible pixels, effectively materializing the 3D design for the viewer.

- Lights: Illumination is a cornerstone of any visual representation, and in the realm of 3D, lights play a pivotal role. In our context, the AmbientLight class instance was employed to emulate the global illumination effect, granting a consistent level of brightness to the entire scene. These lights contribute significantly to the visual ambiance and enhance the depth and realism of the rendered objects.

- Materials: Materials go beyond the mere geometry of 3D objects; they define how these objects appear to the viewer. Inclusive of parameters such as color, texture, metalness, transparency, and shininess, materials are the visual essence of a model. Our approach involved creating generic material parameters that could be tailored to the specifics of different steel structures. This practice ensured each type of structure within a scene could be uniquely identifiable, thus enhancing the clarity and comprehensibility of the rendered design

A WebGL canvas is an HTML5 element used to display WebGL graphics on a web page. It provides a drawing surface where WebGL, a powerful graphics technology, can render both 2D and intricate 3D graphics with hardware acceleration. This canvas acts as a viewport for visualizing interactive and dynamic content, making it a core component for creating immersive visual experiences directly within a web browser. When utilizing the Three.js library,the management and creation of the WebGL canvas are often facilitated by the Three.js renderer. This renderer streamlines the integration of WebGL's capabilities, offering us an efficient way to translate complex 3D Model scenes into captivating visuals without delving into low-level graphics programming intricacies. This amalgamation of HTML5, WebGL, and Three.js empowers web applications to deliver interactive and visually compelling renditions of 3D structures across a wide range of devices, enhancing user engagement and fostering creativity.

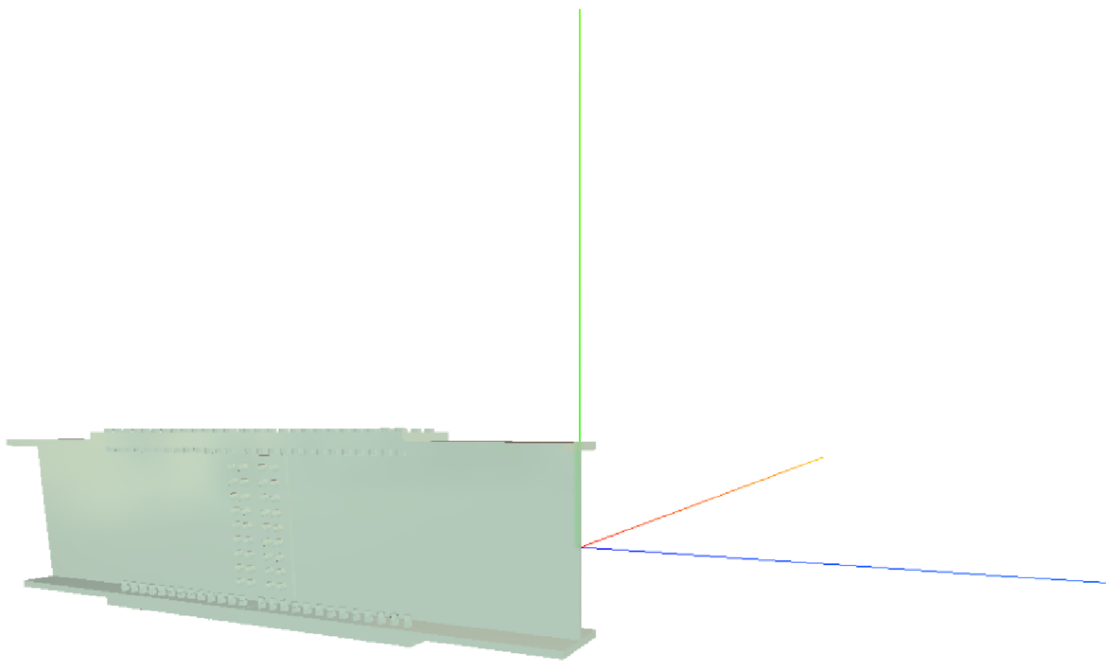


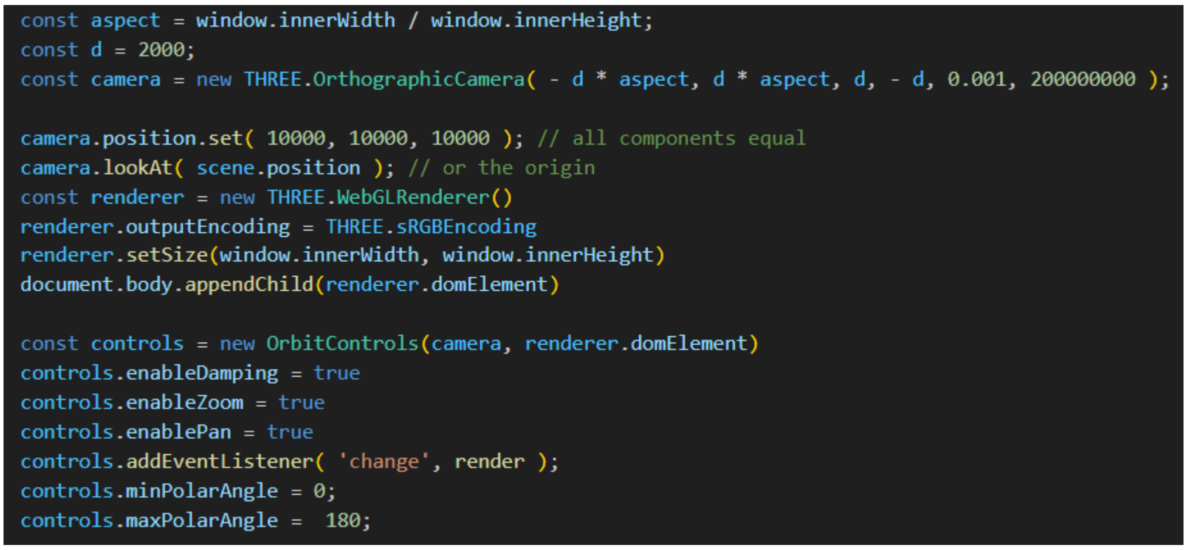
Figure 2.1: Initial Scene Render of a Beam-Beam Splice Bolted Connection

2.3 Final Features of CAD Model Renderer and Scene Results

2.3.1 Default Isometric View and Centre Alignment

- The initial rendition employed an instance of THREE.PerspectiveCamera, which simulated real-world perspective by making objects farther away appear smaller than those closer to the camera. This effect introduced a sense of depth within the 3D scene, contributing to its realism.
- However, this approach led to an inconsistency in the apparent lengths and dimensions of structural edges, particularly when the model was panned or viewed from different angles. This inconsistency compromised the accurate representation of the model's spatial relationships.
- Additionally, the initial rendition positioned the model in a manner that wasn't centered around its global coordinates. This contributed to challenges in maintaining a coherent spatial reference point and could potentially lead to confusion when interpreting the model.
- To address these limitations, the decision was made to adopt the isometric view as the default for rendering CAD models. The isometric view, achieved using a THREE.OrthographicCamera instance, ensures equal foreshortening along all three axes. This results in a clear, undistorted representation where objects retain their proportional relationships, aiding in accurate measurement and interpretation.
- Implementing the isometric view involved configuring a THREE.OrthographicCamera instance to replace the previously used THREE.PerspectiveCamera. This switch enhanced the consistency of edge lengths and dimensions throughout the model, promoting a more accurate depiction of the design.
- In addition to the camera adjustment, the model itself was aligned precisely with the axes. This alignment was achieved by utilizing a THREE.Group() instance, which allowed for the precise manipulation of the model's positioning and orientation. This step ensured that the model's inherent geometry aligned seamlessly with the coordinate system, facilitating accurate visualization and analysis.

- The final implementation code is referenced in the appendix [1]



```
const aspect = window.innerWidth / window.innerHeight;
const d = 2000;
const camera = new THREE.OrthographicCamera( - d * aspect, d * aspect, d, - d, 0.001, 200000000 );

camera.position.set( 10000, 10000, 10000 ); // all components equal
camera.lookAt( scene.position ); // or the origin
const renderer = new THREE.WebGLRenderer()
renderer.outputEncoding = THREE.sRGBEncoding
renderer.setSize(window.innerWidth, window.innerHeight)
document.body.appendChild(renderer.domElement)

const controls = new OrbitControls(camera, renderer.domElement)
controls.enableDamping = true
controls.enableZoom = true
controls.enablePan = true
controls.addEventListener( 'change', render );
controls.minPolarAngle = 0;
controls.maxPolarAngle =  180;
```

Figure 2.2: Snippet of setting up View and Camera conigurations.

2.3.2 Model Pan,Zoom and Rotation Controls

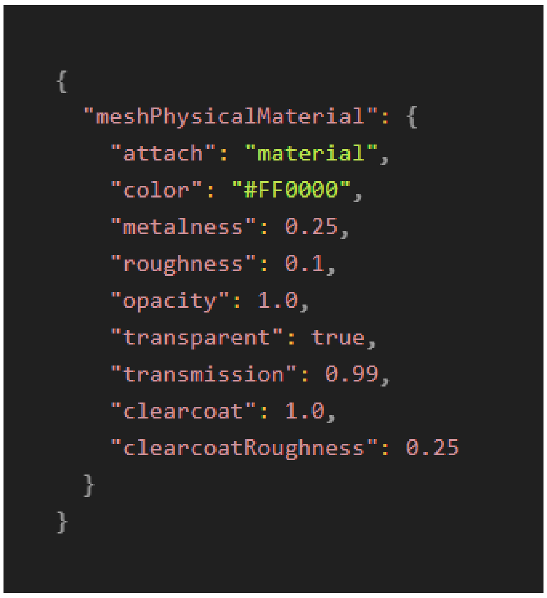
- The CAD renderer incorporates comprehensive zoom, pan, and orbit controls that allow interactive manipulation of the models within the scene, enhancing the user's ability to explore and analyze the design from various perspectives.
- Among these controls, the orbit control functionality enables the user to smoothly rotate the view around a central point, providing a dynamic and immersive experience.
- Presently, the implementation of the orbit control feature involves utilizing an instance of THREE.OrbitControls(), a specialized control provided by Three.js. This control empowers users to intuitively navigate the model by dragging the cursor or touch gestures, making the visualisation process interactive and engaging.
- By integrating these controls, the CAD renderer offers a user-friendly interface that goes beyond static visuals, enabling users to actively engage with and understand the intricacies of the 3D models. This interactive capability proves invaluable for design analysis and evaluation.

2.3.3 Model Material and Texture Properties

- For the model material representation after experimentation with different materials in ThreeJS,I proceeded finally with THREE.MeshPhysicalMaterial.
- MeshPhysicalMaterial is a material type in Three.js that simulates real-world physically-based rendering (PBR) properties, making it suitable for rendering realistic surfaces like metals and plastics. It takes into account parameters such as color, metalness, roughness, transparency, and clearcoat, allowing for accurate representation of complex materials with advanced shading effects.

2.3.4 Multiple Models single scene supported

- The CAD renderer showcases versatility by supporting the simultaneous rendering of multiple models within a shared scene. This capability is crucial for scenarios involving complex assemblies or designs that consist of multiple interconnected components.



```
{
  "meshPhysicalMaterial": {
    "attach": "material",
    "color": "#FF0000",
    "metalness": 0.25,
    "roughness": 0.1,
    "opacity": 1.0,
    "transparent": true,
    "transmission": 0.99,
    "clearcoat": 1.0,
    "clearcoatRoughness": 0.25
  }
}
```

Figure 2.3: Model Material and Texture Metadata

- Each model is meticulously positioned and visualized around its own global coordinates. This fidelity to the original geometry ensures that the visual representation remains consistent with the way these models were created using the Python-OCC backend.
- This capability not only serves the purpose of accurate visualization but also has practical applications. For instance, it allows for the representation of intricate steel structures where individual components are distinguished by distinct colors. Each color assigned to a substructure can correspond to a specific section of the larger steel assembly.
- This approach enhances the clarity of representation and facilitates efficient communication of design concepts. By enabling the clear identification of different parts within a complex whole, this feature aids in comprehension, assessment, and collaborative discussions, particularly for large and intricate structures. The rendered models collectively provide a comprehensive visual narrative of the assembly's composition and arrangement.

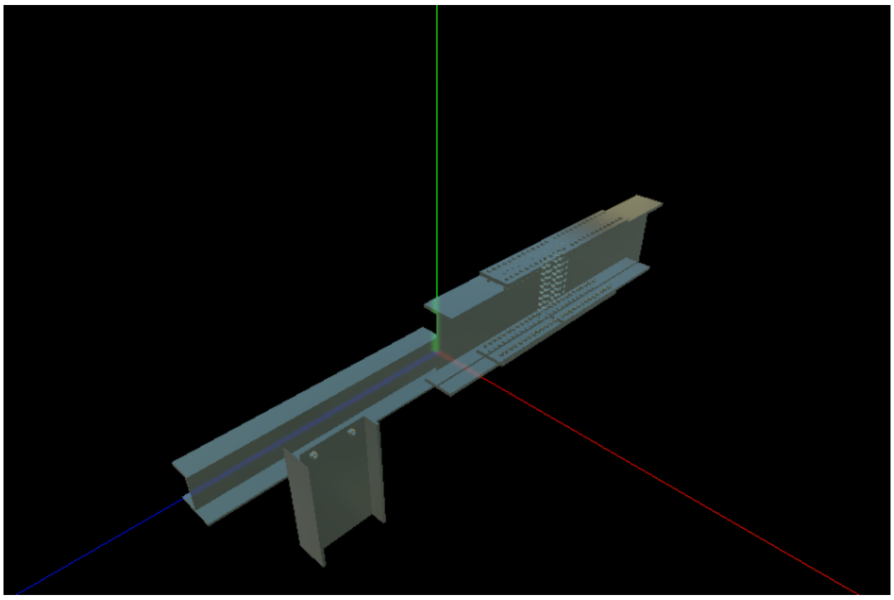


Figure 2.4: Multiple models in the same scene

2.4 Integration with Frontend

2.4.1 Framework Choice

The integration of the CAD Model Viewer with the frontend requires the ThreeJS scene of our CAD model to be embedded in a ReactJS View.

In order to facilitate this the ThreeJS canvas was integrated with the React view using a framework called React-Three-Fiber which is a React renderer for ThreeJS.
The reasons for using the above framework were

- Beyond mere compatibility, React Three Fiber introduces a declarative syntax that enables the description of complex 3D environments as React components. This high-level approach provides developers with a more intuitive and readable means of constructing and managing intricate scenes, fostering an environment where 3D graphics are authored with the same clarity as traditional user interfaces.
- An exceptional feature of React Three Fiber is its adept use of React's hooks. By employing hooks such as useState and useEffect, developers can effortlessly manage state and handle side effects within 3D components. This integration significantly simplifies the process of incorporating dynamic behaviors and interactive features into 3D scenes, enhancing their richness and interactivity.
- Furthermore, React Three Fiber's integration capitalizes on React's well-established ecosystem. This synergy empowers developers to harness React's core principles of component reusability, efficient updates, and performance optimization. By applying React's familiar development patterns to 3D graphics, the learning curve for crafting immersive environments is lowered, and best practices for web development naturally extend to the 3D realm.
- The utilization of React Three Fiber resonates particularly well with the broader React community. Developers already versed in React's principles can swiftly transfer their skills to crafting 3D experiences, minimizing the complexity of adopting a new technology stack. This translates into a more streamlined development process and ensures a more consistent and cohesive experience for users interacting with both 3D and traditional UI elements.

2.4.2 Integration Implementation

- The main operating page of Osdag on the web, which serves as the central hub for user interaction, is thoughtfully structured using a collection of div classes. Each of these div classes encapsulates specific components, such as Input Values, the CAD Model Renderer, the Output Dock, and other relevant parameters, ensuring a modular and organized layout for the user interface.
- A div class, functioning as a virtual DOM element, serves as a flexible container that can house other JSX elements or components. Notably, the div class with the class name superMainBody-mid is designed to house the content that occupies the middle section of the Model render page.
- Within this superMainBody-mid div, a crucial integration occurs: the inclusion of the Canvas component from React Three Fiber. This integration results in the creation of a WebGL canvas within the div, providing an essential canvas for rendering dynamic 3D content.
- Nestled within the Canvas component, a functional component named Model is employed. This component, derived from React Three Fiber's offerings, acts as a representation of the 3D CAD Model Scene. This structured approach not only enhances the maintainability of the codebase but also promotes code reuse through modularization.
- This Model functional component is drawn from a separate rendering script named three-render.jsx. In this script, the Model function is meticulously defined. This function, orchestrated

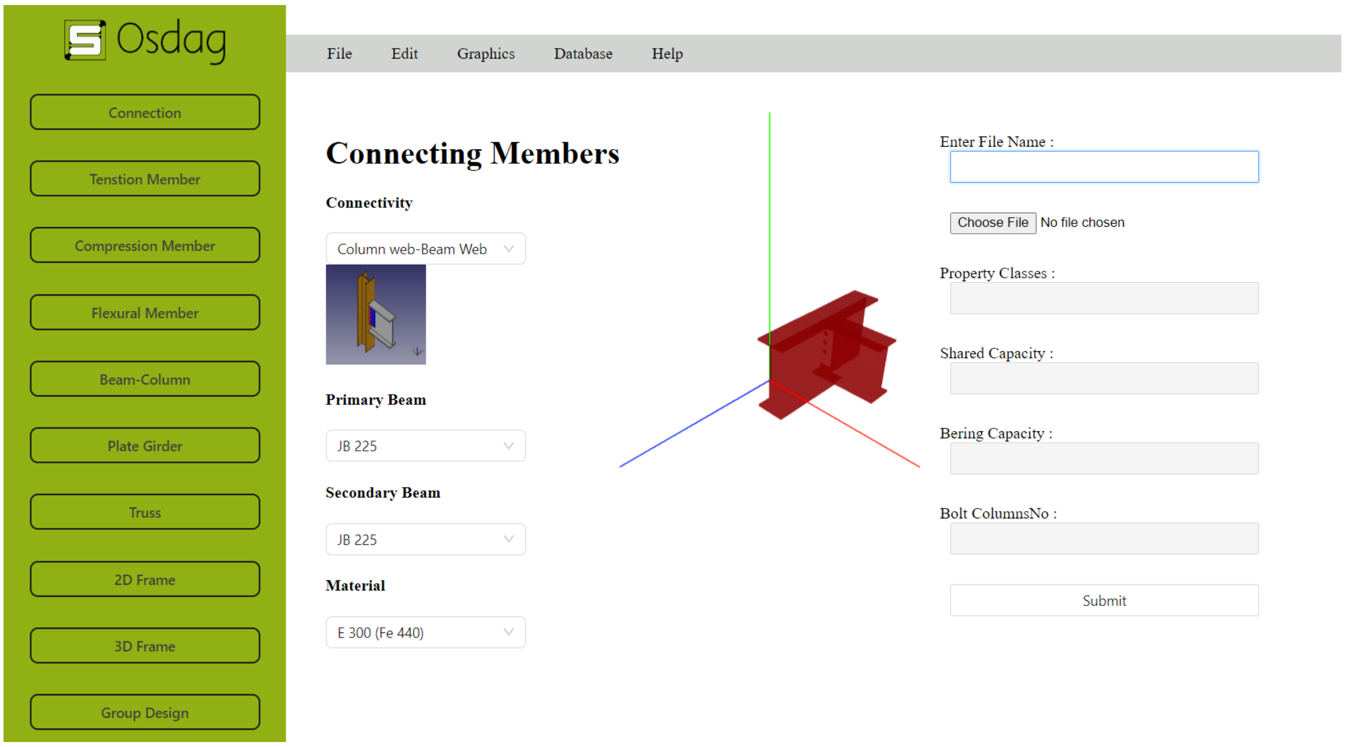
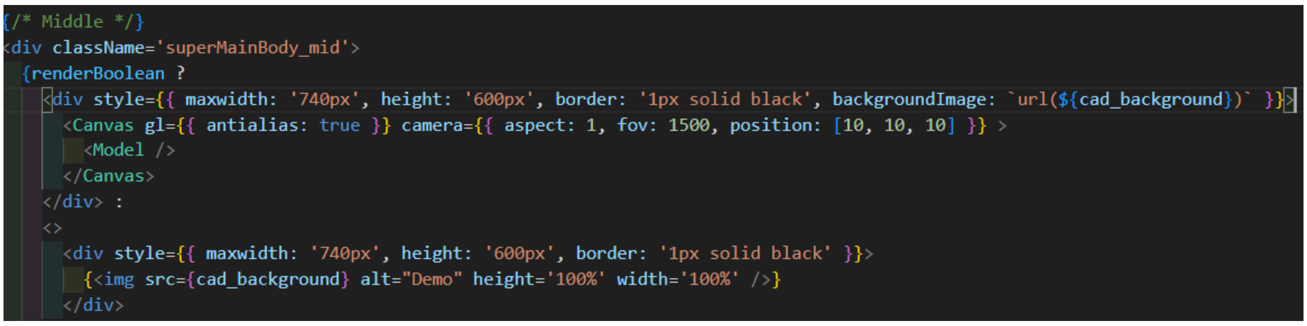


Figure 2.5: ThreeJS canvas integrated with frontend

based on the principles discussed in sections 2.2 and 2.3, constructs a comprehensive 3D scene comprising various essential components.

- One of the notable features of this setup is the portability of the Model function. By being exportable, it can be utilized to render the model in any WebGL canvas. This flexibility ensures that the CAD Viewer is versatile, capable of rendering a diverse range of 3D models.[]



```
{/* Middle */}
<div className='superMainBody_mid'>
  {renderBoolean ?
    <div style={{ maxwidth: '740px', height: '600px', border: '1px solid black', backgroundImage: `url(${cad_background})` }}>
      <Canvas gl={{ antialias: true }} camera={{ aspect: 1, fov: 1500, position: [10, 10, 10] }} >
        <Model />
      </Canvas>
    </div> :
    <>
      <div style={{ maxwidth: '740px', height: '600px', border: '1px solid black' }}>
        {<img src={cad_background} alt="Demo" height='100%' width='100%' />}
      </div>
```

Figure 2.6: Code snippet of Canvas integration

2.5 Future Updates to Renderer

1. **Fixing Model Orientation: Aligning Axes with the Model's Global Coordinates**
   One of the upcoming enhancements involves refining the orientation of the CAD model within the Three.js scene. This advancement seeks to seamlessly align the axes of the 3D environment with the global coordinates of the imported CAD model. This alignment is pivotal for ensuring a consistent and intuitive user experience. By automatically adjusting the camera and model positions, users will be able to interact with the model more effectively, eliminating any confusion caused by misaligned axes. This update will not only enhance the overall visual appeal of the rendering but also contribute to accurate measurements and annotations, as users can confidently interpret the model's representation.

2. **Developing Color Coded Formatting of Various Components and Their Respective Colors**
   In the pursuit of enhancing the clarity and interpretability of the CAD model, we are working on the implementation of a sophisticated color-coded formatting system. This innovative feature will assign distinct and consistent colors to different components of the model, such as beams, joints, and supports. By employing a standardized color palette, the rendering will reflect the structural hierarchy and relationships among various elements. This approach will empower users to easily differentiate between components, facilitating a deeper understanding of the model's intricacies. Furthermore, this color coding system will contribute to improved analysis capabilities, enabling users to swiftly identify crucial elements and their interconnections. To ensure adaptability, we are also considering options for users to customize the color scheme according to their preferences and specific requirements.

3. **Toggle Keys for Orthographic, x, y Axis Views**
   As part of our ongoing efforts to enhance user navigation and interaction, we are developing toggle keys that provide users with swift access to different viewing perspectives. This feature will enable users to seamlessly transition between various viewpoints, including orthographic and specific x or y axis views. The inclusion of predefined views will streamline the navigation process. By incorporating keyboard shortcuts to toggle between different perspectives, we aim to optimize workflow efficiency, allowing users to effortlessly switch between views without interrupting their design or analysis tasks. This enhancement seeks to make the CAD viewer more accessible and user-friendly, catering to a diverse range of users while bolstering overall productivity.

Chapter 3

Implementation of CAD File Converter

3.1 Need of CAD Format conversion

- As previously highlighted in section 2.1, a pivotal task in preparing 3D structure models for effective web rendering is the conversion of CAD file formats. Common formats such as BREP, STL, and IGES need to be seamlessly transformed into WebGL-compatible formats, including OBJ and GLTF. These conversions are indispensable for ensuring that intricate 3D models are optimally showcased on the web platform.
- A paramount requirement during this inter-format conversion is to safeguard the fidelity of the model's texture, structure, and geometrical properties. The conversion process must uphold a strict standard of being lossless, preserving the intricate details that characterize the original model.
- Beyond maintaining fidelity, efficiency in the conversion process is paramount. To ensure a smooth and seamless rendering experience, the conversion process must transpire with minimal time delay. This responsiveness contributes to the overall user experience and prevents undue waiting periods during rendering.
- The development of this CAD Model Converter is meticulously guided by a well-defined system flow. This flow allows the CAD Model Renderer to seamlessly fetch the converted model from a pre-determined, fixed path. This structured approach enhances the integration between the converter and renderer, streamlining the rendering pipeline.
- In light of these complex requirements, the CAD Model Converter has been ingeniously crafted. This converter serves as a crucial bridge between the diverse world of CAD formats and the web's WebGL-compatible formats. It ensures that the rendering process is marked by fidelity, efficiency, and a seamless flow of data, ultimately contributing to the high-quality visual experience that Osdag's users encounter.

3.2 Choice of Framework:FreeCAD

- Notably, FreeCAD served as the framework underpinning the successful implementation of the CAD converter process. A pivotal aspect of FreeCAD is its extensive support for Macros—Python scripts that facilitate the automation of repetitive tasks and actions within the software. These Macros grant users the power to create personalized tools, functions, and subroutines, all orchestrated and executed by FreeCAD's powerful engine.
- The integration with FreeCAD's Macros is of paramount importance for the CAD converter's functioning. The process leverages these scripts to facilitate the seamless transformation of 3D model formats. The adaptability and automation that Macros bring are critical to the efficiency of the conversion process.

- A distinct advantage of FreeCAD lies in its support for command-line invocation. This capability proved pivotal for the integration, as the macro designed for CAD file conversion could be effortlessly executed as a subprocess through a single command-line instruction. This integration not only streamlines the workflow but also contributes to the overall efficiency and speed of the CAD conversion process.
- One of the compelling features of FreeCAD's macros is their ability to harness pre-built functions. This facet greatly simplifies the inter-format conversion process. The pre-built functions provide a solid foundation, enabling the seamless translation of intricate 3D model representations from one format to another, ensuring that the intricate details are accurately preserved throughout the transformation.

3.3 FreeCAD Macros:Implementation of CAD Converter

- The CAD converter process was implemented as a FreeCAD macro,utilising FreeCAD's pre-defined functions.
- Given the input CAD file path in BREP format and the path of the output directory to save the OBJ file in, the Macro converts the CAD file.
- Command-Line Arguments: The script receives command-line arguments, specifically the path of the BREP file (sys.argv[2]) and the desired output filename for the GLTF file (sys.argv[3]). If the number of arguments is less than three, an error message is printed, and the script exits.
- Opening the BREP File: The script uses the Part.open() function from the FreeCAD library to open the BREP file located at the specified path. The BREP file represents a 3D solid model in a geometric format suitable for FreeCAD.
- Object Retrieval: The script retrieves the FreeCAD active document (FreeCAD.activeDocument()) and iterates through the objects within the document where each objects represents a distinct steel structure Model.
- GLTF Export: After identifying the objects to be exported, the script calls the Mesh.export() function to export the object as a OBJ file with the provided output filename. The object is converted to a triangular mesh suitable for OBJ export.
- Overall, this script [3] serves as a FreeCAD macro.In this case, it automates the process of exporting 3D objects from a FreeCAD document(any CAD file format opened in FreeCAD) to a OBJ file format, making it more accessible for use in our ThreeJS scene.

3.4 Back-end Integration of the CAD Process:CAD Model API

- The API is designed to handle GET requests for the purpose of generating and retrieving CAD models. These models are constructed based on user input values provided in the input dock. The CAD Model Render process invokes this API to retrieve the essential .OBJ 3D CAD Model file, a critical element for rendering intricate 3D designs within the ThreeJS scene.

- The API's primary role revolves around processing GET requests. When triggered, it associates the provided design session id with the specific input values linked to that session. These input values are then utilized within a specialized function leveraging Python OCC. This function meticulously crafts the BREP Model file, tailored to the provided input parameters. This BREP Model is temporarily stored in a designated directory, serving as an intermediate step.

- The subsequent phase involves triggering a subprocess. This subprocess effectively summons FreeCAD through command-line instructions. Within FreeCAD, a designated macro is executed, seamlessly converting the BREP file into an OBJ file format. This OBJ file is then securely stored in a predetermined, static directory, ready for further utilization.

- With the transformed OBJ file in hand, the CAD Model Renderer's duty comes to the fore. It adeptly employs this file to create a visual representation of the model within the 3D scene orchestrated by the ThreeJS engine.

- Figure 3.1 visually summarizes this sequence, illustrating the journey of user inputs evolving into a 3D model in the ThreeJS scene. In essence, the API serves as a critical connector, transforming user inputs into a visual 3D representation. It's an intermediary between user intent and digital visualization, facilitating a seamless transition from concept to a interactive 3D representation,

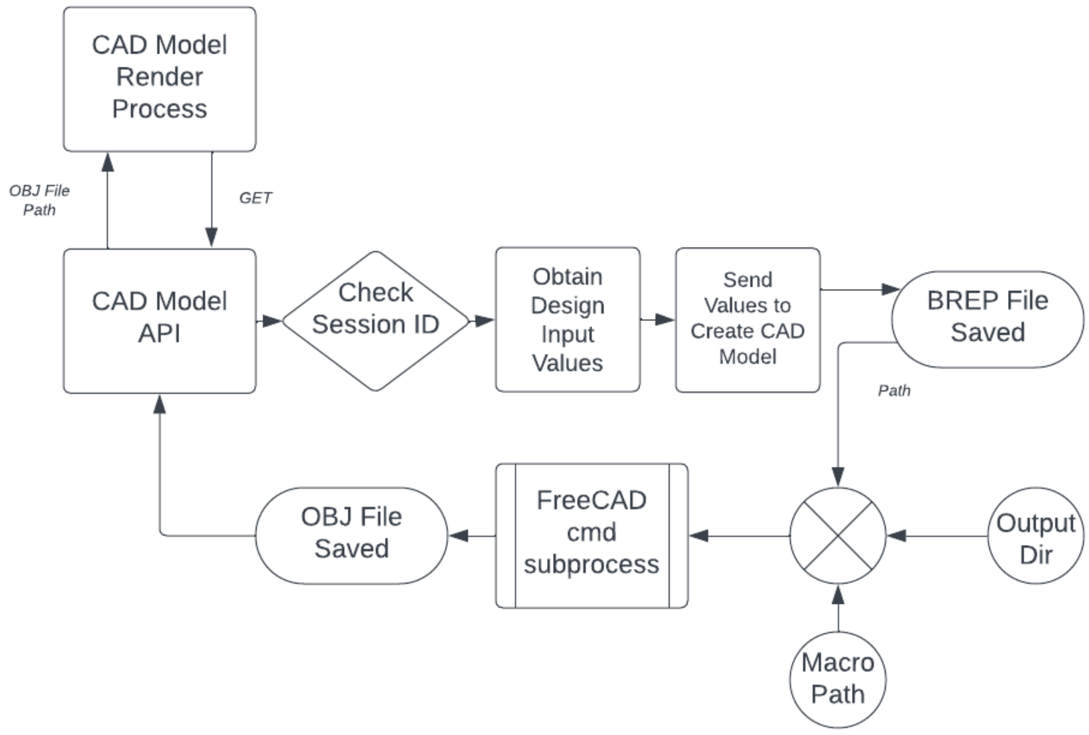


Figure 3.1: CAD Render and Converter Flow

Chapter 4

References

1. Three.js documentation
   https://threejs.org/docs/index.htmlmanual/en/introduction/Creating-a-scene

2. Three.js official forum
   https://discourse.threejs.org/

3. SBCode's Three.js tutorials and samples
   https://sbcode.net/threejs/loaders-obj/

4. Python OpenCascade documentation
   $https://liuxinwin_admin.gitee.io/pythonocc - docs/index.html$

5. React-Three-Fiber Canvas component documentation
   https://docs.pmnd.rs/react-three-fiber/api/canvas

6. Code Sandbox React-Three-Fiber examples
   https://codesandbox.io/examples/package/react-three-fiber

7. FreeCAD Macros documentation
   $https://wiki.freecad.org/Macros_recipes$

8. Stack Overflow
   https://stackoverflow.com/questions/23450588/isometric-camera-with-three-js

9. Animating scenes with WebGL and Three.js
   https://www.august.com.au/blog/animating-scenes-with-webgl-three-js/

10. MDN Canvas component documentation
    https://developer.mozilla.org/en-US/

Chapter 5

Appendix

5.1 Code Samples

1. **Three.js renderer implementation**

```
import * as THREE from './three.js-master/build/three.module.js'
import { STLLoader }  from './three.js-master/examples/jsm/loaders/STLLoad
import { OBJLoader }  from './three.js-master/examples/jsm/loaders/OBJLoad
import { OrbitControls } from './three.js-master/examples/jsm/controls/Orb
import Stats from './three.js-master/examples/jsm/libs/stats.module.js'
const scene = new THREE.Scene()
scene.add(new THREE.AxesHelper(10000))
const light = new THREE.AmbientLight()
light.position.set(10000, 10000, 10000)
scene.add(light)
const aspect = window.innerWidth / window.innerHeight;
const d = 2000;
const camera = new THREE.OrthographicCamera( - d * aspect, d * aspect, d, -

camera.position.set( 10000, 10000, 10000 ); // all components equal
camera.lookAt( scene.position ); // or the origin
const renderer = new THREE.WebGLRenderer()
renderer.outputEncoding = THREE.sRGBEncoding
renderer.setSize(window.innerWidth, window.innerHeight)
document.body.appendChild(renderer.domElement)

const controls = new OrbitControls(camera, renderer.domElement)
controls.enableDamping = true
controls.enableZoom = true
controls.enablePan = true
controls.addEventListener( 'change', render );
controls.minPolarAngle = 0;
controls.maxPolarAngle =  180;

const envTexture = new THREE.CubeTextureLoader().load([
    './three.js-master/examples/textures/cube/pisa/px.png',
    './three.js-master/examples/textures/cube/pisa/nx.png',
    './three.js-master/examples/textures/cube/pisa/py.png',
    './three.js-master/examples/textures/cube/pisa/ny.png',
    './three.js-master/examples/textures/cube/pisa/pz.png',
    './three.js-master/examples/textures/cube/pisa/nz.png'
])
```

```
envTexture.mapping = THREE.CubeReflectionMapping

const material = new THREE.MeshPhysicalMaterial({
    color: 0xb2ffc8,
    envMap: envTexture,
    metalness: 0.25,
    roughness: 0.1,
    opacity: 1.0,
    transparent: true,
    transmission: 0.99,
    clearcoat: 1.0,
    clearcoatRoughness: 0.25
})
const material2 = new THREE.MeshPhysicalMaterial({
    color: 0x0000ff,
    envMap: envTexture,
    metalness: 0.25,
    roughness: 0.1,
    opacity: 1.0,
    transparent: true,
    transmission: 0.99,
    clearcoat: 1.0,
    clearcoatRoughness: 0.25
})

const group = new THREE.Group();
var loader = new OBJLoader();
loader.load('/assets/output-obj.obj', function (obj) {
    obj.traverse(function (child) {
        if (child.isMesh) {
            var m = child;
            m.receiveShadow = true;
            m.castShadow = true;
            m.material = material;
        }
        if (child.isLight) {
            var l = child;
            l.castShadow = false;
            l.shadow.bias = -0.003;
            l.shadow.mapSize.width = 2048;
            l.shadow.mapSize.height = 2048;
        }
    });
    obj.traverse(function(object) {
        if (object.isMesh && object.name) {
            console.log(object.name);
        }
    });
    group.add(obj);
}, function (xhr) {
    console.log((xhr.loaded / xhr.total) * 100 + '% loaded');
}, function (error) {
    console.log(error);
```

```
});

var loader2 = new OBJLoader();
loader2.load('/assets/bb-splice-bolted (1).obj', function (obj) {
    obj.traverse(function (child) {
        if (child.isMesh) {
            var m = child;
            m.receiveShadow = true;
            m.castShadow = true;
            m.material = material2;
        }
        if (child.isLight) {
            var l = child;
            l.castShadow = false;
            l.shadow.bias = -0.003;
            l.shadow.mapSize.width = 2048;
            l.shadow.mapSize.height = 2048;
        }
    });
    obj.traverse(function(object) {
        if (object.isMesh && object.name) {
            console.log(object.name);
        }
    });
    group.add(obj);
}, function (xhr) {
    console.log((xhr.loaded / xhr.total) * 100 + '% loaded');
}, function (error) {
    console.log(error);
});
window.addEventListener('resize', onWindowResize, false)
function onWindowResize() {
    camera.aspect = window.innerWidth / window.innerHeight
    camera.updateProjectionMatrix()
    renderer.setSize(window.innerWidth, window.innerHeight)
    render()
}

const material3 = new THREE.MeshPhysicalMaterial({
    color: 0xff0000 ,
    envMap: envTexture,
    metalness: 0.25,
    roughness: 0.1,
    opacity: 1.0,
    transparent: true,
    transmission: 0.99,
    clearcoat: 1.0,
    clearcoatRoughness: 0.25
})

var loader3 = new OBJLoader();
loader3.load('/assets/fin-plate-og.obj', function (obj) {
    obj.traverse(function (child) {
        if (child.isMesh) {
            var m = child;
            m.receiveShadow = true;
            m.castShadow = true;
            m.material = material3;
        }
        if (child.isLight) {
            var l = child;
            l.castShadow = false;
            l.shadow.bias = -0.003;
            l.shadow.mapSize.width = 2048;
            l.shadow.mapSize.height = 2048;
        }
    });
    obj.traverse(function(object) {
        if (object.isMesh && object.name) {
            console.log(object.name);
        }
    });
    group.add(obj);
}, function (xhr) {
    console.log((xhr.loaded / xhr.total) * 100 + '% loaded');
}, function (error) {
    console.log(error);
});

scene.add(group);



new_origin = group.getWorldPosition((0,0,0));
console.log(new_origin)
const stats = Stats()
document.body.appendChild(stats.dom)

function animate() {
    requestAnimationFrame(animate)

    controls.update()

    render()

    stats.update()
}

function render() {
    renderer.render(scene, camera)
}

animate()
```

2. **React-Three-Fiber Renderer Implementation**

```
import { OBJLoader } from 'three/examples/jsm/loaders/OBJLoader.js'
import { OrbitControls, useTexture} from '@react-three/drei'
import { useLoader } from '@react-three/fiber';
import React, {useMemo} from "react";
function Model() {
    const obj = useLoader(OBJLoader,"/output-obj.obj"); //issue is here that o
    //console.log('obj loader :  ', obj)
    //return <primitive object={obj} />
    const texture = useTexture("/texture.png");
    //console.log('texture  : ', texture)
    const geometry = useMemo(() => {
      let g;
      obj.traverse((c) => {
        if (c.type === "Mesh") {
          const _c = c ;
          g = _c.geometry;
        }
      });
      //console.log("Done Loading")
      return g;
    }, [obj]);

    // I've used meshPhysicalMaterial because the texture needs lights to be s
    // AxesHelper param changing the axes lengths
    // scale:model scale
    return (
      <group name='scene'>
          <axesHelper args={[200]}/>
          <mesh geometry={geometry} scale={0.008}>
            <meshPhysicalMaterial attach = "material" color={'#FF0000'} metaln
          </mesh>
          <OrbitControls />
      </group>
    );
}

export default Model;
```

3. **FreeCAD code**

```
import FreeCAD
import Part
import Mesh
import sys
import os
if len(sys.argv) < 3:
   print('Error: No output path argument provided')
   sys.exit()

# Retrieve the path argument
path = sys.argv[2]
output_filename = sys.argv[3]

#print(type(path))
print('The path of the brep file ',path)
```

```
Part.open(path)

doc = FreeCAD.activeDocument()
print(doc.Label)
doc_name = doc.Label
__objs__=[]
for objz in doc.Objects:
    print(objz.Name)
    print(doc_name)
    print("Inside For")
    #if objz.Name == doc_name:
    print("Inside if")
        # Export the object to gltf for
    __objs__.append(objz)
    break  # Stop looping once you find the object
print('This is the file to which we export',output_filename)
Mesh.export(__objs__,output_filename)
del __objs__
```

4. **CAD Model API**

```
from django.shortcuts import render, redirect
from django.utils.html import escape, urlencode
from django.http import HttpResponse, HttpRequest
from django.views import View
from osdag.models import Design
from django.utils.crypto import get_random_string
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator
from osdag_api import developed_modules, get_module_api
from osdag_api.errors import OsdagApiException
import typing
import json
import os
import subprocess
import time

# rest_framework
from rest_framework import status
from rest_framework.response import Response

# importing models
from osdag.models import Design


@method_decorator(csrf_exempt, name='dispatch')
class CADGeneration(View):
    """
        Update input values in database.
            CAD Model API (class CADGeneration(View)):
                Accepts GET requests.
                Returns BREP file as content_type text/plain.
                Request must provide session cookie id.
    """
```

```
def get(self, request: HttpRequest):
    # Get design session id.
    cookie_id = request.COOKIES.get("fin_plate_connection_session")
    print(cookie_id)
    # Error Checking: If design session id provided.
    if cookie_id == None or cookie_id == '':
        # Returns error response.
        return HttpResponse("Error: Please open module", status=400)
    # Error Checking: If design session exists.
    if not Design.objects.filter(cookie_id=cookie_id).exists():
        # Return error response.
        return HttpResponse("Error: This design session does not exist", s
    try: # Error checking while loading input data
        # Get session object from db.
        try :
            design_session = Design.objects.get(cookie_id=cookie_id)
        except :
            print('Error in obtaining the fin_plate_connection_session')

        try :
            module_api = get_module_api(
                design_session.module_id) # Get module api
        except :
            print('error in obtaining modele_api from the design_session')
        # Error Checking: If input data not entered.


        #if not design_session.current_state:
        #    # Return error response.
        #    return HttpResponse("Error: Please enter input data first", s
        # Load input data into dictionary.
        try :
            input_values = design_session.input_values
        except :
            print('error in loading the input_values from the design_sessio
    except Exception as e:
        # Return error response.
        print('first erorr')
        return HttpResponse("Error: Internal server error: " + repr(e), st
    section = "Model" # Section of model to generate (default full model)
    if request.GET.get("section") != None: # If section is specified,
        section = request.GET["section"] # Set section
        print('section : ' , section)
    try: # Error checking while Generating BREP File.
        # Generate CAD Model.
        print('creating cad model')
        path = module_api.create_cad_model(
            input_values, section, cookie_id)
        print('path : ' , path)
        designObject = Design.objects.get(cookie_id = cookie_id)
        try :
            if(not path) :
                print('path is false')
                # set the cad_design_status to False
                designObject.cad_design_status = False
                designObject.save()

                return HttpResponse('CAD model generation failed', status
            if(path) :
                # set the cad_design_status to True
                print('path is valid')
                designObject.cad_design_status = True
                designObject.save()
        except Exception as e :
            print('Exception found while saving the CAD design status : '

    except OsdagApiException as e:  # If section does no exist
        return HttpResponse(repr(e), status=400)  # Return error response.
    except Exception as e:
        # Return error response.
        return HttpResponse("Error: Internal server error: " + repr(e), st

    #try :
    #     os.chdir('/home')
    #except Exception as e :
    #     print('chdir e : ', e)

    try :
        # Pass the path variable as a command-line argument to the FreeCAD
        current_dir = os.path.dirname(os.path.abspath(__file__))
        # Get the path of the parent directory
        parent_dir = os.path.dirname(os.path.dirname(current_dir))
        macro_path = os.path.join(
            parent_dir, 'freecad_utils/open_brep_file.FCMacro')
        command = '/snap/bin/freecad.cmd'
        # path = 'file_storage/cad_models/Uv9aURCfBDmhoosxMUy2UT7P3ghXcvV
        path_to_file = os.path.join(parent_dir, path)
        output_dir = os.path.join(
            parent_dir, 'osdagclient/public/output-obj.obj')
    except Exception as e :
        print('output dir e : ', e)
    # Call the subprocess to create the empty output file
    try :
        subprocess.run(["touch", output_dir])
    except Exception as e :
        print('subprocess run e : ', e)

    command_with_arg = f'{command} {macro_path} {path_to_file} {output_dir
    # Execute the command using subprocess.Popen()
    process = subprocess.Popen(command_with_arg.split())

    time.sleep(3)
    response = HttpResponse(output_dir, status=201)
    response["content-type"] = "text/plain"
    return response
```