**Documentation for the Web application for 3d mesh renderer**

**Languages used**

Javascript/ THREE.js v0.165

HTML 5

**Web application architecture**

**A diagram of a computer network

Description automatically generated**

**How to setup**

**For macOS only, for windows please search for parallel methods/CHATGPT☺**

Download the code from repo< <https://github.com/Wyl-ASG/webrpd_code.git> (to be updated)>

Please install vs code

Go to the directory of the code in the terminal, type

npm install

To run the localhost on your computer, type in the terminal in the same directory

npx vite --port 8089

To build it into bundle.js to run on the webpage, type

npm run build

Then just put the bundle.js into wherever you running the code from ie. Github pages. DO NOT CHANGE THE OTHER STUFF UNLESS YOU WANT TO CHANGE THE CSS/LOOK OF THE HTML PAGE

Script.js is for the loading bar  
  
\*\*\*\*EDIT IT ONLY IF YOU UNDERSTAND IT\*\*\*\*

**Web application deployment to github**

Current repo link < <https://github.com/Wyl-ASG/finale> >

Testing is < <https://wyl-asg.github.io/finale/?id=OAJBLhkNOQoOdFADDmMBGQ> >

797 is the test case id, do not change anything else in the linkA screenshot of a phone

Description automatically generated

When trying to host the website, ensure everything except ds\_store is inside the repo/hosting website, things that are missing just copy from the code repo

Change the visibility to public

A screenshot of a black screen

Description automatically generated

To deploy github pages, go to pages and press <Save>

Build has to be in MAIN REPO.A screenshot of a computer

Description automatically generated

**Code Documentation**

**Importing Required Libraries and Modules**

import \* as THREE from "../node\_modules/three/build/three.module.js";

import { OrbitControls } from "../node\_modules/three/examples/jsm/controls/OrbitControls.js";

import { TrackballControls } from "../node\_modules/three/examples/jsm/controls/TrackballControls.js";

import { STLMeshLoader } from './STLMeshLoader.js';

import { STLLoader } from '../node\_modules/three/examples/jsm/loaders/STLLoader.js';

import { OFFLoader } from './OFFLoader.js';

import { ApiClient } from './ApiClient.js';

import { addVisibilityAndTransparencyControls } from './control.js';

import { GUI } from '../node\_modules/dat.gui';

import { addResetButton } from './resetButton.js';

**Description**

This section imports the necessary modules and libraries, including THREE.js for 3D rendering, OrbitControls and TrackballControls for camera manipulation, custom STLMeshLoader and OFFLoader for loading 3D models, and ApiClient for API interactions.

Both OFFLoader and STLMeshLoader contains the processes to load the occlusion and undercutheatmap

**Initial Setup**

console.log(THREE.REVISION);

let all\_mesh\_mat = {};

window.finished = false;

const url = new URL(window.location.href);

const paramValue = url.searchParams.get('id');

const close = url.searchParams.get('close');

**Description**

* Logs the version of THREE.js.
* Initializes variables for storing mesh materials and a global finished flag.
* Retrieves URL parameters for id and close.

**Creating the Scene**

const scene = new THREE.Scene();

let camera;

const aspect = window.innerWidth / window.innerHeight;

camera = new THREE.OrthographicCamera(window.innerWidth / -2, window.innerWidth / 2, window.innerHeight / 2, window.innerHeight / -2, -500, 1000);

let mouseX = window.innerWidth / 2;

let mouseY = window.innerHeight / 2;

let undercut\_type = {};

let controls;

let orb\_controls;

let objToRender = 'dino';

let mesh\_geo;

const material = new THREE.MeshPhongMaterial({ vertexColors: true });

const materialsurface = new THREE.MeshStandardMaterial({

color: 0xA0A0A0,

metalness: 1.0,

roughness: 0.2

});

**Description**

* Sets up a Three.js scene and an orthographic camera.
* Initializes variables for mouse position tracking, controls, rendering options, and materials for meshes.

**API Client and Data Fetching**

This takes data from smartrpd api and database

Example for the json data

<

// this for the undercut upper and the main json data use to retrieve stuff

const data = {

machine\_id: '3a0df9c37b50873c63cebecd7bed73152a5ef616',

uuid: 'm+Cakg1hzVqCwVeJfNGRpSyvRXv4',

case\_int\_id: paramValue,

jaw\_type: 2,

caseIntID: paramValue

};

// this for the undercut lower

const data2 = {

machine\_id: '3a0df9c37b50873c63cebecd7bed73152a5ef616',

uuid: 'm+Cakg1hzVqCwVeJfNGRpSyvRXv4',

case\_int\_id: paramValue,

jaw\_type: 1,

caseIntID: paramValue

};

>

const apiClient = new ApiClient('https://35.198.233.36:8090/api/smartrpd');

const parentObject = new THREE.Object3D();

scene.add(parentObject);

(async () => {

const data = { /\*...\*/ };

const data2 = { /\*...\*/ };

let positionDatas = [];

let positionData;

const urldatas = ['/case/get/' + paramValue];

try {

for (const urldata of urldatas) {

positionData = await apiClient.post(urldata, [data]);

positionDatas = positionDatas.concat(positionData);

}

} catch (error) {

console.error('Error:', error);

}

const time = unixToHumanReadable(positionData.creation\_date);

const update\_time = unixToHumanReadable(positionData.last\_updated);

createTextbox("Creation Date: " + time, 'bottom-left');

createTextbox("Case ID: " + positionData.case\_id, 'bottom-right');

createTextbox("Last Updated: : " + update\_time, 'bottom-left2');

**Description**

* Initializes the ApiClient with the base URL and adds a parent object to the scene.
* Fetches case data from the API and processes the response, creating text boxes for displaying the creation date, case ID, and last updated time.

**Fetching and Processing Undercut Heatmap Data**

let undercut\_values = [];

const heatmap\_urldatas = ['/undercutheatmap/get'];

try {

const undercut\_value = await apiClient.post(heatmap\_urldatas, data);

undercut\_values = undercut\_values.concat(undercut\_value);

undercut\_type[undercut\_value.jaw\_type] = [Boolean(undercut\_value.surveying\_values), Boolean(undercut\_value.occlusion\_values)];

const undercut\_value1 = await apiClient.post(heatmap\_urldatas, data2);

undercut\_values = undercut\_values.concat(undercut\_value1);

undercut\_type[undercut\_value1.jaw\_type] = [Boolean(undercut\_value1.surveying\_values), Boolean(undercut\_value1.occlusion\_values)];

} catch (error) {

console.error('Error:', error);

}

**Description**

* Fetches undercut heatmap data for the specified case and jaw types from the API, storing the results in undercut\_values and undercut\_type.

**Fetching and Processing 3D Mesh Data**

let stl = false;

const urls = ['/parameterisation/mesh/getall', '/surface/getall'];

let responseDatas = [];

let responseData;

let loop = 0;

try {

for (const url of urls) {

loop += 1;

if (!close) {

responseData = await apiClient.post(url, [data]);

if (responseData != 'stl') {

responseDatas = responseDatas.concat(responseData);

}

if (loop == 1) {

const test = await apiClient.post('/stl/get', [data], 'test');

if (test != 'stl') {

// Create and append a button to the document

createCloseButton();

}

}

}

if (responseData == 'stl' && url == '/parameterisation/mesh/getall' && !close) {

responseData = await apiClient.post('/stl/raw/get', [data]);

stl = true;

} else if (close && url == '/parameterisation/mesh/getall') {

responseData = await apiClient.post('/stl/get', [data]);

stl = false;

createBackButton();

} else {

responseData = 'stl';

}

if (responseData != 'stl') {

responseDatas = responseDatas.concat(responseData);

}

}

} catch (error) {

console.error('Error:', error);

}

for (const offFile of responseDatas) {

let loader;

if (offFile.filename.includes('surface')) {

loader = new OFFLoader(materialsurface.clone());

} else {

loader = new OFFLoader(material.clone());

}

const offdata = atob(offFile.data);

let x;

if (offFile.filename.includes('ParameterisationMesh') || offFile.filename.includes('closed')) {

x = true;

}

if (stl) {

const stlMeshLoader = new STLMeshLoader(material);

if (offFile.type.includes('upper')) {

mesh\_geo = stlMeshLoader.load(offdata, undercut\_values[1]);

} else if (offFile.type.includes('lower')) {

mesh\_geo = stlMeshLoader.load(offdata, undercut\_values[0]);

}

} else if (offFile.type.includes('upper')) {

mesh\_geo = loader.parse(offdata, undercut\_values[1], x);

} else if (offFile.type.includes('lower')) {

mesh\_geo = loader.parse(offdata, undercut\_values[0], x);

}

const mesh = mesh\_geo[0];

mesh.name = offFile.filename;

mesh.userData = { jaw\_type: offFile.type };

if (all\_mesh\_mat != null) {

all\_mesh\_mat[offFile.filename] = mesh\_geo[1].slice();

}

if (offFile.type.includes('upper') && !stl && !close) {

changeMeshRotation(mesh, 1, 1, 180);

mesh.position.y += 5;

}

parentObject.add(mesh);

}

**Description**

* Fetches 3D mesh data from the API and processes the responses.
* Loads the data into the scene using custom loaders for OFF and STL files.
* Adjusts the mesh positions and rotations based on their types.

**Utility Functions**

function changeMeshRotation(mesh, x, y, z) {

mesh.rotation.set(THREE.MathUtils.degToRad(x), THREE.MathUtils.degToRad(y), THREE.MathUtils.degToRad(z));

}

function createTextbox(text, position) {

const textbox = document.createElement('div');

textbox.textContent = text;

textbox.style = { /\* styling properties \*/ };

document.body.appendChild(textbox);

}

function unixToHumanReadable(unixTimestamp) {

const date = new Date(unixTimestamp \* 1000);

return `${date.getFullYear()}-${(date.getMonth() + 1).toString().padStart(2, '0')}-${date.getDate().toString().padStart(2, '0')} ${date.getHours().toString().padStart(2, '0')}:${date.getMinutes().toString().padStart(2, '0')}:${date.getSeconds().toString().padStart(2, '0')}`;

}

function createCloseButton() {

const button = document.createElement('button');

button.textContent = 'Close.off';

button.style = { /\* styling properties \*/ };

button.addEventListener('click', () => {

window.location.href = window.location.href + '&close=1';

});

document.body.appendChild(button);

}

function createBackButton() {

const button = document.createElement('button');

button.textContent = 'Back';

button.style = { /\* styling properties \*/ };

button.addEventListener('click', () => {

window.location.href = window.location.href.replace('&close=1', '');

});

document.body.appendChild(button);

}

**Description**

* Provides utility functions for changing mesh rotation, creating text boxes, converting Unix timestamps to human-readable dates, and creating navigation buttons.

**Rendering and Animation Loop**

const renderer = new THREE.WebGLRenderer({ antialias: true });

renderer.setPixelRatio(window.devicePixelRatio);

renderer.setSize(window.innerWidth, window.innerHeight);

renderer.setClearColor(0x000000, 1);

renderer.gammaOutput = true;

renderer.outputEncoding = THREE.sRGBEncoding;

document.body.appendChild(renderer.domElement);

if (objToRender == 'dino') {

camera.position.z = 500;

orb\_controls = new OrbitControls(camera, renderer.domElement);

orb\_controls.addEventListener('change', render);

orb\_controls.enablePan = false;

orb\_controls.enableDamping = true;

orb\_controls.rotateSpeed = 0.5;

orb\_controls.update();

} else if (objToRender == 'jaws') {

camera.zoom = 1.4;

camera.updateProjectionMatrix();

controls = new TrackballControls(camera, renderer.domElement);

controls.noPan = true;

controls.noZoom = true;

controls.rotateSpeed = 5;

controls.addEventListener('change', render);

controls.update();

}

window.addEventListener('resize', onWindowResize);

function onWindowResize() {

camera.left = window.innerWidth / -2;

camera.right = window.innerWidth / 2;

camera.top = window.innerHeight / 2;

camera.bottom = window.innerHeight / -2;

camera.updateProjectionMatrix();

renderer.setSize(window.innerWidth, window.innerHeight);

render();

}

function render() {

renderer.render(scene, camera);

}

function animate() {

requestAnimationFrame(animate);

if (controls) controls.update();

render();

}

animate();

addResetButton(parentObject, all\_mesh\_mat);

addVisibilityAndTransparencyControls(parentObject);

**Description**

* Sets up the Three.js renderer and adds it to the document body.
* Configures camera controls based on the object type to render.
* Adds event listeners for window resizing and rendering.
* Defines the animation loop to update controls and render the scene continuously.
* Adds functionality for resetting the scene and controlling visibility and transparency of meshes.

**Conclusion**

This documentation covers the main sections of the provided code, detailing its purpose and functionality. The code sets up a 3D scene using Three.js, fetches and processes 3D model data from an API, and renders the models with interactive controls. Utility functions aid in manipulating the meshes and handling user interface elements.