МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБЗАСТОСУНКУ ВІЗУАЛІЗАЦІЇ ЧИСЛОВИХ ДАНИХ З ВИКОРИСТАННЯМ ТЕХНОЛОГІЇ WEBGL»**

Виконав: студент 2 курсу, групи 8.1212-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.О. Омельченко
(ініціали та прізвище)

Керівник декан математичного факультету, професор, д.т.н. Гоменюк С.І.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри комп'ютерних наук, професор, д.т.н. Шило Г.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Омельченку Владиславу Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка вебзастосунку візуалізації числових даних з використанням технології WebGL

керівник роботи Гоменюк Сергій Іванович, д.т.н., професор
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 27.11.2023 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.
3. Код.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Дослідження теоретичних відомостей та інструментів.
2. Постановка задачі та проєктування.
3. Реалізація компонентів WebGL.
4. Реалізація вебдодатку та тестування.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 03.05.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.05.2023	
2.	Збір вихідних даних.	09.06.2023	
3.	Обробка методичних та теоретичних	23.08.2023	
	джерел.		
4.	Розробка першого та другого розділу.	18.10.2023	
5.	Розробка третього та четвертого розділу.	14.11.2023	
6.	Оформлення та нормоконтроль	20.11.2023	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	15.12.2023	

Студент _____ В.О. Омельченко
(підпис) (ініціали та прізвище)

Керівник роботи _____ С.І. Гоменюк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебзастосунку візуалізації числових даних з використанням технології WebGL»: 70 с., 23 рис., 17 джерел, 7 додатків.

ВЕБДОДАТОК, КОМПОНЕНТ, МЕТОД СТРУКТУРОВАНОГО ПРОГРАМУВАННЯ ДЖЕКСОНА, ПРОЄКТУВАННЯ, РОЗРОБКА, СТРУКТУРА, ТЕХНОЛОНІЯ WEBGL.

Об’єкт дослідження – процес розробки вебзастосунку.

Мета роботи: розробка вебзастосунку візуалізації числових даних з використанням технології WebGL.

Метод дослідження – створення вебзастосунку, використовуючи мову програмування JavaScript та метод Джексона.

У роботі проведено аналіз даних та проєктування структури вебзастосунку, розроблено та протестовано вебзастосунок візуалізації числових даних з використанням технології WebGL. Даний вебдодаток розроблено на мові програмування JavaScript з фреймворком React.js та з використанням бібліотеки glMatrix для полегшення роботи з матричними та векторними операціями.

SUMMARY

Master’s qualifying paper «Development of a Web Application for the Numerical Data Visualization Using WebGL Technology»: 70 pages, 23 figures, 17 references, 7 supplements.

WEB APPLICATION, COMPONENT, JACKSON STRUCTURED PROGRAMMING METHOD, PLANNING, DEVELOPMENT, STRUCTURE, WEBGL TECHNOLOGY.

The object of the study is the process of developing of a web application.

The aim of the study is development of a web application for the numerical data visualization using WebGL technology.

The methods of research are creating of web application using JavaScript programming language and Jackson method.

In this work data analysis and design of the structure of the web application were carried out, a web application for the numerical data visualization using WebGL technology was developed and tested. This web application is developed using JavaScript programming language with the React.js framework and using the glMatrix library to facilitate work with matrix and vector operations.

ЗМІСТ

ВСТУП

В сучасному інформаційному суспільстві обробка та візуалізація великих обсягів числової інформації вирізняються особливою актуальністю та важливістю. Щоденно збільшуючи свої масштаби, дані стають необхідним елементом прийняття стратегічних рішень у різних галузях людської діяльності. Однак, щоб ефективно розуміти та аналізувати ці дані, необхідно мати інструменти їхньої візуалізації, які надають можливість отримати з ними інсайти та визначити закономірності.

Дана робота присвячена розробці вебзастосунку для візуалізації числових даних, який базується на технології WebGL. Ця технологія дозволяє створювати динамічні та інтерактивні візуалізації безпосередньо в браузері, що робить їх доступними для широкого кола користувачів.

В рамках роботи проведеться аналіз даних та проєктування вебзастосунку. Також розглянуться технічні аспекти використання WebGL. У роботі буде приділена увага можливості комфортного користування вебзастосунку з метою забезпечення високоякісного користувацького досвіду.

Мета дослідження полягає у створенні ефективного та функціонального вебзастосунку для візуалізації числових даних з використанням технології WebGL, який задовольнятиме високим стандартам якості та забезпечить користувачам нові можливості у вивченні та аналізі інформації.

1 ДОСЛІДЖЕННЯ ТЕОРЕТИЧНИХ ВІДОМОСТЕЙ ТА ІНСТРУМЕНТІВ

В даному розділі розглядаються інструменти, необхідні для розробки та метод проєктування вебзастосунку.

1.1 Аналіз предметної області

Візуалізація числових даних у тривимірному просторі може допомогти в розумінні складних залежностей та взаємозв'язків між різними змінними. Існує кілька методів та інструментів для візуалізації 3D-даних, які можуть бути використані в різних областях, таких як наука, інженерія, медицина, фінанси та інші.

При візуалізації числових даних в тривимірному просторі можна зіткнутися з деякими проблемами, які можуть ускладнювати аналіз і сприйняття інформації. Ось деякі з можливих проблем.

Перенавантаженість інформацією. У тривимірному просторі може бути важко уникнути перенавантаження графіку. Якщо на графіку занадто багато точок, ліній або об'єктів, це може спричинити плутанину та затемнення деяких даних.

Погане сприйняття глибини. Людський зір краще розпізнає плоскі об'єкти, ніж тривимірні. Таким чином, може виникнути проблема сприйняття глибини при спробі аналізувати дані у тривимірному просторі.

Обмеженість екрану. Графік може виглядати інакше на екрані, ніж він насправді є в тривимірному просторі. Різні вигляди і перспективи можуть викликати спотворення та призводити до неправильного сприйняття даних.

Складність взаємодії з графіком. Взаємодія з тривимірними графіками може бути складною. Вибір об'єктів, обертання графіку та інші дії можуть бути менш інтуїтивними, ніж у двовимірних графіках.

Неефективність для деяких типів даних: деякі типи даних можуть бути краще візуалізовані у двовимірному просторі. Наприклад, якщо дані мають складні взаємозв'язки або залежності, їх може бути важко виразити у тривимірному вигляді.

Щоб подолати ці проблеми, можна розглядати альтернативні методи візуалізації, такі як використання кольорів, анімацій, освітлення, тіней тощо. Також важливо розглядати потреби користувачів та забезпечувати їхню зручність та зрозумілість при взаємодії з графіками.

1.2 Технологія WebGL

WebGL (Web Graphics Library) – це технологія, яка дозволяє використовувати 3D-графіку у вебдодатках, надаючи можливість відтворювати 3D-сцени безпосередньо в браузері. Це технологія для відтворення інтерактивної 2D- та 3D-графіки в будь-якому сумісному веббраузері без використання додаткових модулів [1]. WebGL повністю інтегровано з іншими вебстандартами, дозволяючи використовувати прискорену графічним процесором фізику, обробку зображень та ефекти на полотні HTML. Елементи WebGL можна змішувати з іншими елементами HTML та поєднувати з іншими частинами сторінки чи тлом сторінки. Програми WebGL складаються з керуючого коду, написаного на JavaScript, та коду шейдера, написаного мовою OpenGL ES Shading Language (GLSL ES), мовою, схожою на C або C++. WebGL широко підтримується сучасними браузерами. Однак його доступність також залежить від інших факторів, наприклад, чи підтримує його GPU.

Низькорівневий характер WebGL API, який сам по собі мало що надає для швидкого створення бажаної 3D-графіки, спонукав до створення бібліотек вищого рівня, які абстрагують загальні операції (наприклад, завантаження графів сцени та 3D-об'єктів у певних форматах; до шейдерів або перегляду зрізів). Деякі такі бібліотеки були перенесені на JavaScript з інших мов.

Як і для будь-якого іншого графічного API, створення вмісту для сцен WebGL вимагає використання інструменту створення 3D-контенту та експорту сцени у формат, який читається програмою перегляду або допоміжною бібліотекою.

Якщо розглянути можливості для шейдерів, то шейдерні програми в WebGL мають доступ до трьох типів сховищ даних, кожен з яких має певний варіант використання: атрибути, варіації та уніформи [2].

Атрибути – це змінні, які доступні лише для вершинного шейдера (як змінні) і коду JavaScript. Атрибути зазвичай використовуються для зберігання інформації про колір, координати текстури та будь-яких інших обчислених або отриманих даних, які мають бути спільно використані кодом JavaScript і вершинним шейдером.

Варіанти – це змінні, які оголошуються вершинним шейдером і використовуються для передачі даних від вершинного шейдера до фрагментного шейдера. Це зазвичай використовується для спільного використання вектора нормалі вершини після того, як він був обчислений шейдером вершин.

Уніформи встановлюються кодом JavaScript і доступні як для вершинних, так і для фрагментних шейдерів. Вони використовуються для надання значень, які будуть однаковими для всього, що намальовано у кадрі, наприклад положення та величини освітлення, глобальне перетворення та деталі перспективи тощо.

1.3 Мова програмування JavaScript

JavaScript описують як мультипарадигменну мову програмування. Підтримує об'єктно-орієнтований, імперативний і функціональний стилі. Є реалізацією специфікації ECMAScript. JavaScript має C-подібний синтаксис.

JavaScript зазвичай використовується як вбудовувана мова для

програмного доступу до об’єктів програм. Найбільш широко застосовується в браузерах як мова сценаріїв для надання інтерактивності вебсторінок, розробці односторінкових вебдодатків, програмуванні на стороні серверу, мобільній розробці. Основні архітектурні характеристики:

- динамічна типізація;
- слабка типізація;
- автоматична пам'ять;
- прототипне програмування;
- функції управління як об'єкти першого класу.

JavaScript зазвичай покладається на середовище виконання (наприклад, веббраузер), щоб забезпечити об’єкти та методи, за допомогою яких сценарії можуть взаємодіяти з середовище. JavaScript також покладається на середовище виконання, щоб забезпечити можливість імпортувати сценарії. Модель паралелізму мови описує цикл подій як неблокуючий: введення/виведення програми виконується за допомогою подій і функцій зворотного виклику.

JavaScript використовує JSON для обміну даними. JSON – це легкий формат даних, який людям легко читати й писати, а машинам легко аналізувати й генерувати.

JavaScript має низку властивостей об'єктно-орієнтованої мови, але завдяки концепції прототипів підтримка об'єктів в ній відрізняється від традиційних мов ООП. Крім того, JavaScript має кілька властивостей, притаманних функціональним мовам, – функції як об'єкти першого класу, об'єкти як списки, каррінг, анонімні функції, замикання (closures) – що додає мові додаткову гнучкість.

JavaScript містить декілька десятків вбудованих об'єктів, які поділяються на групи: фундаментальні, помилки, числа та дати, текстові, індексовані, ключові, для роботи з структурованими даними, абстрактні, рефлекційні, групи Intl та WebAssembly [3]. Крім того, JavaScript містить набір вбудованих операцій, що керують логікою виконання програм. Синтаксис JavaScript в

основному відповідає синтаксису мови Java (тобто, зрештою, успадкований від C), але спрощений у порівнянні з ним, щоб зробити мову сценаріїв легкою для вивчення. Так, наприклад, декларація змінної не містить її типу, властивості також не мають типів, а декларація функції може знаходитися в тексті програми після неї.

1.4 Node.js, npm та React.js

Node.js – це міжплатформне серверне середовище з відкритим вихідним кодом. Node.js – це серверне середовище виконання JavaScript, яке працює на механізмі JavaScript V8 і виконує код JavaScript поза веббраузером.

Node.js має наступні властивості:

- асинхронна однониткова модель виконання запитів;
- неблокуючий ввід/вивід;
- система модулів CommonJS;
- рушій JavaScript Google V8.

Node.js дозволяє розробникам використовувати JavaScript для написання інструментів командного рядка та сценаріїв на стороні сервера [4]. Можливість запуску коду JavaScript на сервері часто використовується для створення динамічного вмісту вебсторінки перед тим, як сторінка надсилається у веббраузер користувача. Отже, Node.js представляє парадигму «JavaScript всюди», об’єднуючи розробку вебдодатків навколо однієї мови програмування, на відміну від використання різних мов для програмування на стороні сервера чи клієнта.

Node.js має керовану подіями архітектуру з можливістю асинхронного введення-виведення. Ці варіанти дизайну спрямовані на оптимізацію пропускної здатності та масштабованості вебдодатків із багатьма операціями вводу/виводу, а також для вебдодатків у реальному часі.

За своєю суттю Node.js схожий на фреймворки Perl AnyEvent, Ruby Event

Machine і Python Twisted, але цикл обробки подій у Node.js прихований від розробника і нагадує обробку подій у веб застосунку, що працює в браузері.

Node Package Manager (відомий як npm) – це менеджер пакетів для мови програмування JavaScript Це менеджер пакетів за замовчуванням для Node.js і включений як рекомендована функція до інсталятора Node.js.

Менеджер складається з клієнта командного рядка, який взаємодіє з віддаленим реєстром. Це дозволяє користувачам користуватися модулями JavaScript та розповсюджувати їх. Пакунки в реєстрі знаходяться у форматі CommonJS і включають в себе файли метаданих у форматі JSON. В головному реєстрі npm доступно понад 477000 пакунків. Реєстр не має процедури перевірки, а це означає, що знайдені там пакунки можуть бути низькоякісними або небезпечними. Натомість npm спирається на звіти користувачів, щоб видаляти пакунки, якщо вони порушують політику безпеки (є незахищеними, зловмисними або низькоякісними). npm показує статистику, включаючи кількість завантажень та кількість пакунків, щоб допомогти розробникам оцінювати якість пакетів.

Менеджер пакетів може управляти пакунками, які є локальними залежностями певного проєкту, а також глобально інстальованими інструментами JavaScript. При використанні npm як менеджера залежності для локального проєкту, можна встановити одною командою всі залежності проєкту. У файлі залежності кожна залежність може визначати діапазон дійсних версій, використовуючи схему семантичної версії, що дозволяє розробникам автоматично оновлювати свої пакети, одночасно уникаючи небажаних змін.

React, також відомий як React.js, – це відкрита бібліотека JavaScript для розробки користувацьких інтерфейсів. Він широко використовується для створення вебдодатків та вебсайтів з інтерактивними і адаптивними користувацькими інтерфейсами.

Однією з ключових особливостей React є його компонентна архітектура.

Кожен інтерфейс створюється як комбінація компонентів, які можна перевикористовувати та вкладати один в одного.

React використовує JSX (JavaScript XML). Це розширення синтаксису JavaScript, яке дозволяє використовувати HTML-подібний синтаксис для опису структури інтерфейсу. Як правило, компоненти написані з використанням JSX, але є можливість використання звичайного JavaScript.

React використовує віртуальний DOM. React створює кеш-структуру в пам'яті, що дозволяє обчислювати різницю між попереднім та поточним станом інтерфейсу для оптимального оновлення DOM браузера. Таким чином, можна працювати зі сторінкою, вважаючи, що вона оновлюється вся, але бібліотека самостійно вирішує, які компоненти сторінки необхідно оновити.

1.5 Бібліотека glMatrix

Javascript перетворився на мову, здатну обробляти 3D-графіку в реальному часі через WebGL і обчислювально інтенсивні завдання, такі як фізичне моделювання. Ці типи програм вимагають високопродуктивної векторної та матричної математики, чого Javascript не надає за замовчуванням.

Бібліотека glMatrix призначений для надзвичайно швидкого виконання векторних і матричних операцій [5]. Ручне налаштування кожної функції для досягнення максимальної продуктивності та заохочення ефективних шаблонів використання через конвенції API, glMatrix допоможе отримати максимальну віддачу від механізму JavaScript браузера.

glMatrix моделюється відповідно до потреб WebGL, яка, у свою чергу, використовує матричні угоди, встановлені OpenGL. Зокрема, матриця 4 на 4 – це масив із 16 суміжних плаваючих елементів із 13-м, 14-м і 15-м елементами, що представляють компоненти трансляції x, y, z.

1.6 Метод структути програмування Джексона

Метод структуи програмування Джексона (також відомий як метод Джексона та JSP) полягає в аналізі структур даних файлів, які програма повинна читати як вхідні дані та створювати як вихідні дані, а потім створювати проєкт програми на основі цих структур даних, щоб структура керування програмою обробляла ці структури даних природним чином та інтуїтивно зрозумілий спосіб.

Основні принципи даної методології:

- модульність;
- структурні блоки;
- ієрархія;
- введення та виведення як інтерфейс;
- обробка винятків.

При розробці програм на мові третього покоління традиційним способом підходу до проблеми є використання підходу зверху вниз [6]. Тут проблема записана дуже просто. Потім він розбивається на декілька менших модулів. Кожен із цих менших модулів потім розбивається на дедалі менші модулі, поки кожен модуль не виконає одну й лише одну роботу. Потім вони передаються програмістам для кодування.

Метод Джексона складається з п’яти етапів:

- зображення структури вхідних та вихідних даних;
- ідентифікація відповідностей мід структурами даних;
- спроєктування структури програми;
- перелічення виконуваних операцій;
- створення структурованого викладу.

Низхідні діаграми в програмуванні часто малюються за допомогою діаграм структурного програмування Джексона (JSP). Їх ділять на керовані частини, щоб було краще зрозуміти проблему та ефективніше спланувати рішення.

Метою є створення програм, які легко змінювати протягом усього терміну служби. Основне розуміння Джексона полягало в тому, що зміни вимог зазвичай є незначними змінами існуючих структур. Для програми, створеної з використанням методу, вхідні дані, виходи та внутрішні структури програми збігаються, тому невеликі зміни вхідних і вихідних даних мають перетворитися на невеликі зміни в програмі.

Метод Джексона структурує програми з точки зору чотирьох типів компонентів [7]. Він складається з таких типів:

– елементарні;
– послідовності;
– повторення;
– вибір.

Метод починається з опису вхідних даних програми в термінах чотирьох основних типів компонентів [8]. Далі він таким же чином описує результати програми. Кожен вхід і вихід моделюється як окрема діаграма структури даних. Щоб змусити JSP працювати з інтенсивними обчислювальними програмами, такими як цифрова обробка сигналів, також необхідно намалювати структурні діаграми алгоритмів, які зосереджені на внутрішніх структурах даних, а не на вхідних і вихідних структурах.

Потім вхідні та вихідні структури уніфікуються або об’єднуються в остаточну структуру програми, відому як діаграма структури програми. Цей крок може передбачати додавання невеликої кількості структури керування високого рівня для поєднання входів і виходів. Деякі програми обробляють усі вхідні дані перед тим, як виконувати будь-які виведення, тоді як інші читають один запис, записують один запис і повторюють. Такі підходи мають бути зафіксовані в діаграмі структури.

Діаграма структури програми, який є мовно нейтральним, потім реалізується на мові програмування. Метод структура програмуванни спрямований на програмування на рівні керуючих структур, тому реалізовані проєкти використовують лише примітивні операції, послідовності, ітерації та

вибірки. JSP не використовується для структурування програм на рівні класів і об’єктів, хоча він може корисно структурувати потік керування в методах класу.

1.7 Висновки до першого розділу

Візуалізація тривимірних числових даних відкриває можливості для аналізу складних залежностей та взаємозв'язків. Проте, при використанні візуалізації 3D-даних виникають деякі труднощі, такі як перенавантаженість графіків, складність сприйняття глибини та обмеженість екрану. Для подолання цих проблем можна використовувати альтернативні методи візуалізації та розглядати потреби користувачів.

Технологія WebGL виявляється корисною для візуалізації 3D-графіки у вебзастосунках, забезпечуючи високий рівень взаємодії та можливість відтворення складних сцен безпосередньо в браузері. Використання JavaScript, зокрема за допомогою бібліотеки React.js, дозволяє реалізовувати інтерактивні та адаптивні користувацькі інтерфейси.

Застосування бібліотеки glMatrix дозволяє ефективно виконувати векторно-матричні операції в контексті веброзробки.

Метод структурного програмування Джексона визначається своєю модульністю, структурними блоками та ієрархією. Він надає зручний підхід до розробки програм, забезпечуючи чітку структуру та інтуїтивність.

Узагальнюючи, використання візуалізації 3D-даних у поєднанні з технологією WebGL, JavaScript та методологією структурного програмування Джексона може стати потужним інструментом для розробки вебдодатків з високою взаємодією та обробки складних числових даних.

2 ПОСТАНОВКА ЗАДАЧІ ТА ПРОЄКТУВАННЯ

Даний розділ містить опис задачі та проєктування вебзастосунку за методологією Джексона.

2.1 Опис завдання

В даній роботі розглядається створення вебзастосунку для візуалізації числових даних з використанням технології WebGL.

Вимоги, які повинні вирішуватись в вебзастосунку:

– забезпечення зручним для користуванням графічним інтерфейсом;
– забезпечення контролю над переглядом результату.

Вхідні дані мають представлятися у вигляді файлу, тому для імпортування даних потрібно реалізувати метод вибору файлу.

Для створення тривимірного об’єкту вхідними даними вебзастосунку є файл формату JSON, який містить інформацію про 3D-об’єкт. Даний файл містить:

– вершини;
– індекси вершин;
– нормалі.

Вершина – це структура даних, яка описує певні атрибути, як-от положення точки у 2D чи 3D-просторі або кількох точок на поверхні В 3D-вимірі вершина представлена у вигляді тривимірної координати (x, y, z), яка вказує на одну точку на поверхні моделі.

Індекси – це числові мітки для вершин даної 3D-сцени [9]. Індекси в 3D-відтворення виконують важливу роль у комп’ютерній графіці. Вони використовуються для оптимізації процесу візуалізації шляхом усунення надлишкових даних вершин. Індекси допомагають зменшити використання

пам’яті та покращити продуктивність рендерингу.

Нормаль – це напрямок, до якого звернена грань або вершина в просторі [10]. Цей напрямок визначає, як світло взаємодіє з об’єктом – зокрема, він впливає на те, як світло відбивається або заломлюється від поверхні.

Значення нормалі використовуються програмним забезпеченням візуалізації для обчислення поведінки світла в сцені, і вони можуть значно вплинути на остаточний вигляд 3D-моделі.

Вихідними даними є HTML-сторінка з відображенням 3D-об’єкту, згенерованим обраним JSON-файлу. HTML-сторінка містить полотно, на якому відображається об’єкт, інструменти його керування та можливість завантажити вхідні дані.

Результатом даного застосунку є згенерований у спеціальному середовищі 3D-об’єкт.

2.2 Перший етап методу Джексона

Перший етап методу Джексона полягає в зображенні структури вхідних та вихідних даних. Це найважливіший етап методу Джексона, так як структура даних утворюють основу одержуваних структур програм.

При створенні коректної структури даних для початку ідентифікується можливі компоненти даних, згодом створюється графічне уявлення даних. Останнім завданням першого етапу є виконання контрольного переліку структур даних.

Рисунок 2.1 містить набір компонентів для вхідних структури даних.

Дана структура вхідних даних буде використовуватися для створення візуальної схеми структруи та в якості прикладу при подальшому розгляді методу Джексона.

Рисунок 2.2 представляє схему структури вхідних даних.

Номер посилання	Тип компоненту	Номер посилання старшого	Ім'я компонента	Номер посилання складових компонентів			
				Повтор.	Послід.	Вибір	Елемент.
1	Послід.	—	Вхідні дані	—	1.4	—	1.1, 1.2, 1.3
1.1	Елемент.	1	Змінна vertices	—	—	—	—
1.2	Елемент.	1	Змінна indices	—	—	—	—
1.3	Елемент.	1	Змінна normals	—	—	—	—
1.4	Послід.	1	Реєстрування об'єкту	—	1.4.1	—	—
1.4.1	Послід.	1.4	Додавання екземпляру об'єкту	—	1.4.1.1	—	—
1.4.1.1	Послід.	1.4.1	Відтворення об'єкту	1.4.1.1.1	—	—	—
1.4.1.1.1	Повтор.	1.4.1.1	Тіло циклу	—	—	1.4.1.1.1.1	—
1.4.1.1.1.1	Вибір.	1.4.1.1.1	Керування об'єктом	—	—	—	1.4.1.1.1.1.1, 1.4.1.1.1.1.2, 1.4.1.1.1.1.3
1.4.1.1.1.1.1	Елемент.	1.4.1.1.1.1	Змінна x	—	—	—	—
1.4.1.1.1.1.2	Елемент.	1.4.1.1.1.1	Змінна y	—	—	—	—
1.4.1.1.1.1.3	Елемент.	1.4.1.1.1.1	Змінна z	—	—	—	—

Рисунок 2.1 – Вхідні структури даних

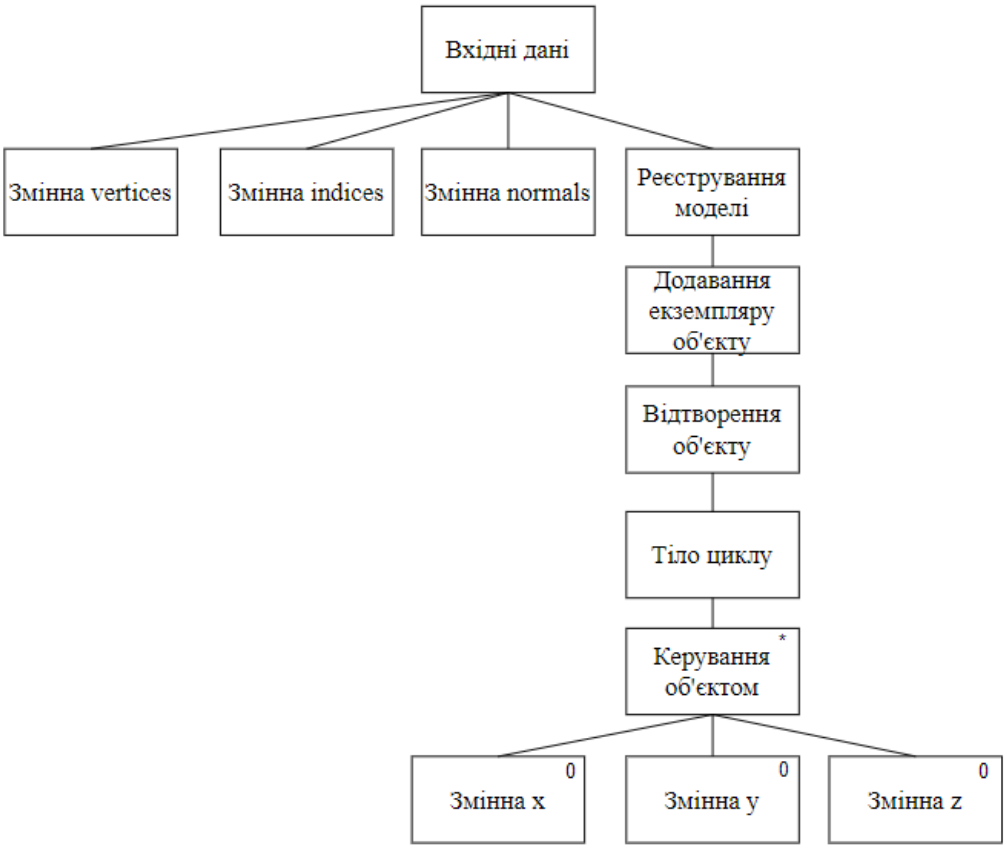

Рисунок 2.2 – Схема структури вхідних даних

Рисунок 2.3 містить набір компонентів для вихідних структури даних. Дана структура вихідних даних буде використовуватися в якості прикладу при подальшому розгляді методу Джексона.

Номер посилання	Тип компоненту	Номер посилання старшого	Ім'я компонента	Номер посилання складових компонентів			
				Повтор.	Послід.	Вибір	Елемент.
1	Послід.	—	Вихідні дані	—	1.1	—	—
1.1	Послід.	1	Тіло циклу	1.1.1	—	—	—
1.1.1	Повтор.	1.1	Результат	—	—	—	—

Рисунок 2.3 – Вихідні структури даних

Рисунок 2.4 представляє схему структури вихідних даних.

Рисунок 2.4 – Схема структури вихідних даних

В даній структурі даних усі входи та виходи ідентифіковані, кожний компонент даних ідентифіковано іменем, усі компоненти прикладного рівня та компонент найвищого рівня представляє весь вхід та вихід. Схема зображена зверху вниз та зліва направо з точки зору появлення компонентів даних та усі компоненти послідовності, повторення, вибору та елементарні компоненти є коректними.

2.3 Другий етап методу Джексона

Другий етап полягає в ідентифікації відповідності обробки між структурами вхідних і вихідних даних, створеними першому етапі.

Рисунок 2.5 представляє результат ідентифікації відповідностей розглянутих структур вхідних та вихідних даних.

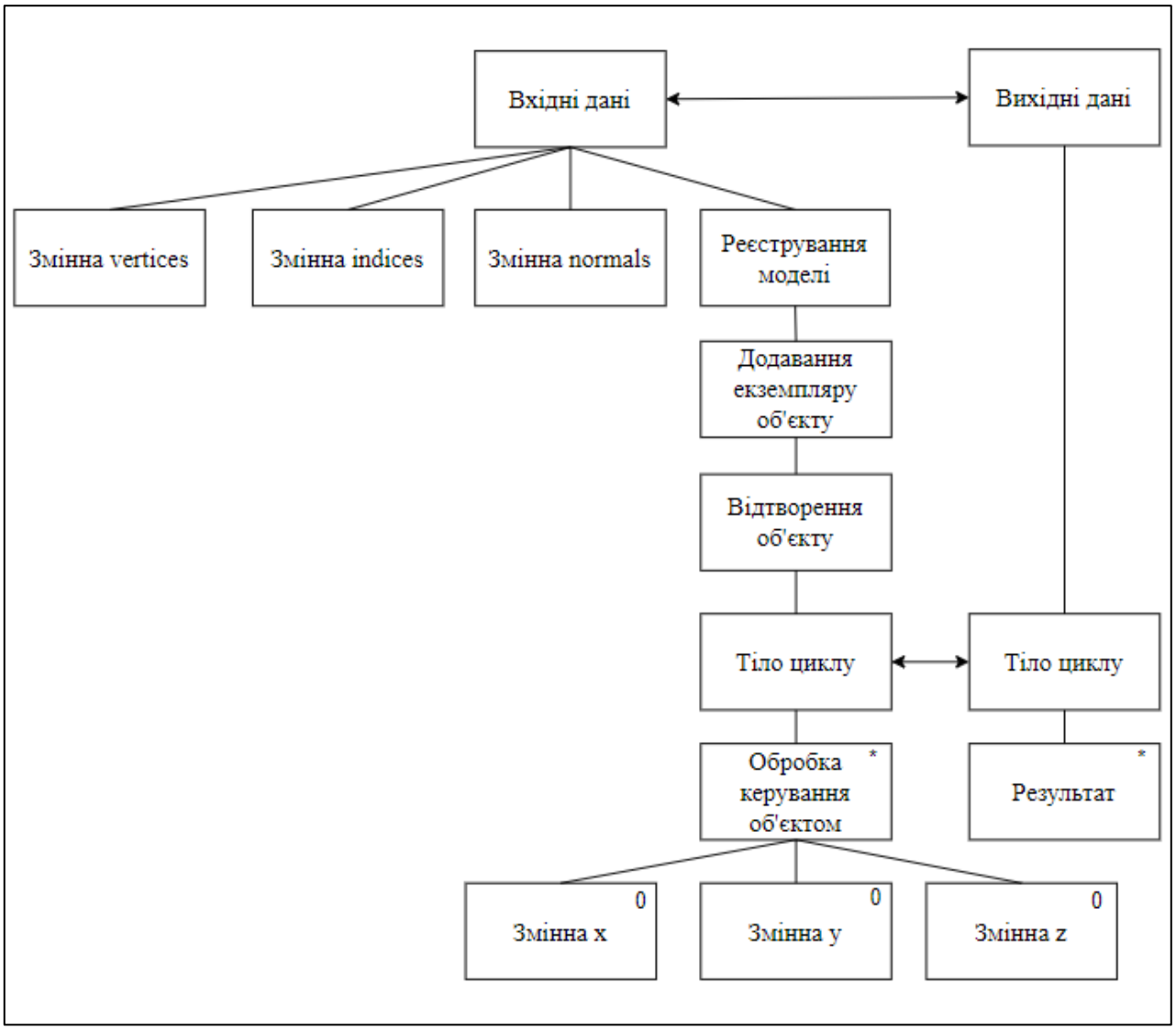

Рисунок 2.5 – Схема ідентифікацій відношень структур

В даному випадку маємо відповідності між вхідними та вихідними даними та тілами циклу.

2.4 Третій етап методу Джексона

Третій етап полягає в створенні структури програми та складається з 3 підетапів створення структур програм.

В перщому підетапі відбувається злиття відповідних компонентів даних, щоб сформувати компоненти програми.

В результаті виходить спрощена структура програми. Кожен з сформованих компонентів програми забезпечується ім'ям виду «Обробка X для створення Y».

Рисунок 2.6 ілюструє результат даного підетапу для розглянутого прикладу.

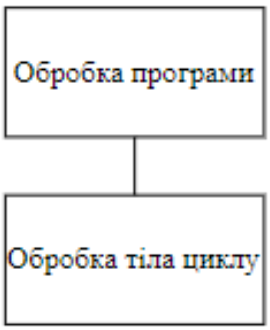

Рисунок 2.6 – Спрощенна схема структури

В другому підетаі додаємо не маючих відповідностей компоненти структури вхідних даних в сформовану структуру програми на ті ж відносні ієрархічні місця і присвоюємо їм відповідні імена по аналогії з підетапу 1.

Рисунок 2.7 містить результат виконання даного підетапи.

На даному малюнку подвійною лінією праворуч виділені додані на даному підетапи компоненти програми. Повторювані компоненти вхідної структури стали повторюваними компонентами програми.

В третьому підетапі додаюємо не маючих відповідностей компоненти структури вихідних даних в сформовану структуру програми на ті ж відносні ієрархічні місця і присвоюємо їм відповідних імен (див. рис. 2.8). Таким чином, в результаті третього етапу сформована керуюча структура програми.

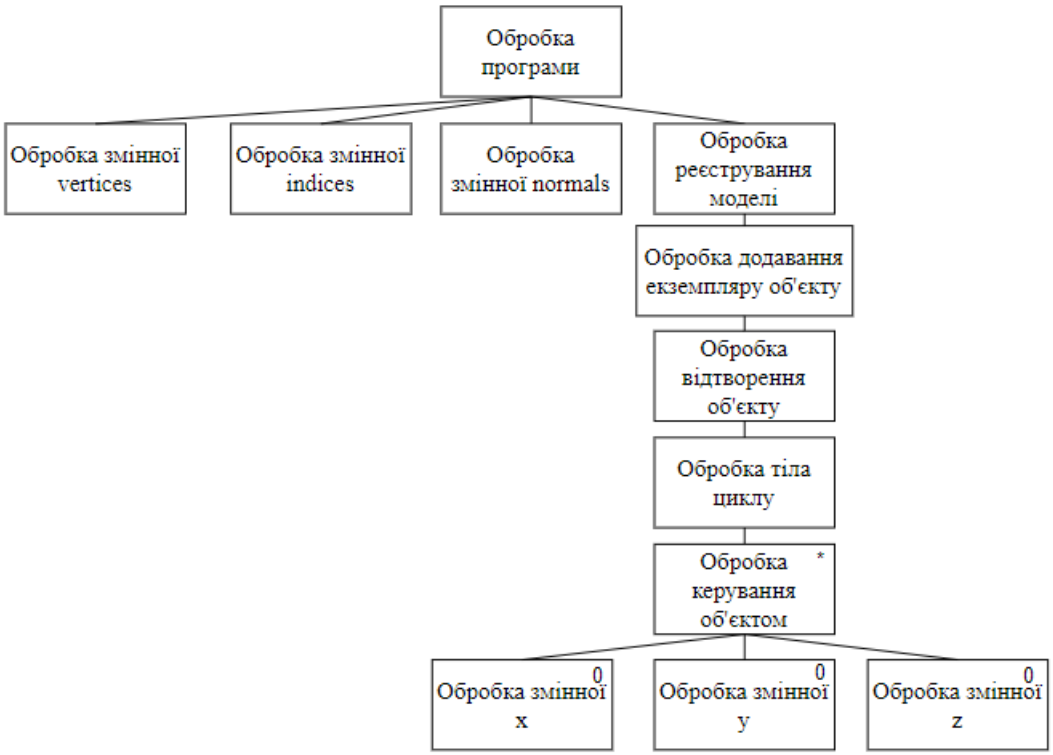

Рисунок 2.7 – Додавання не маючих співвідношень компонентів структури вхідних даних

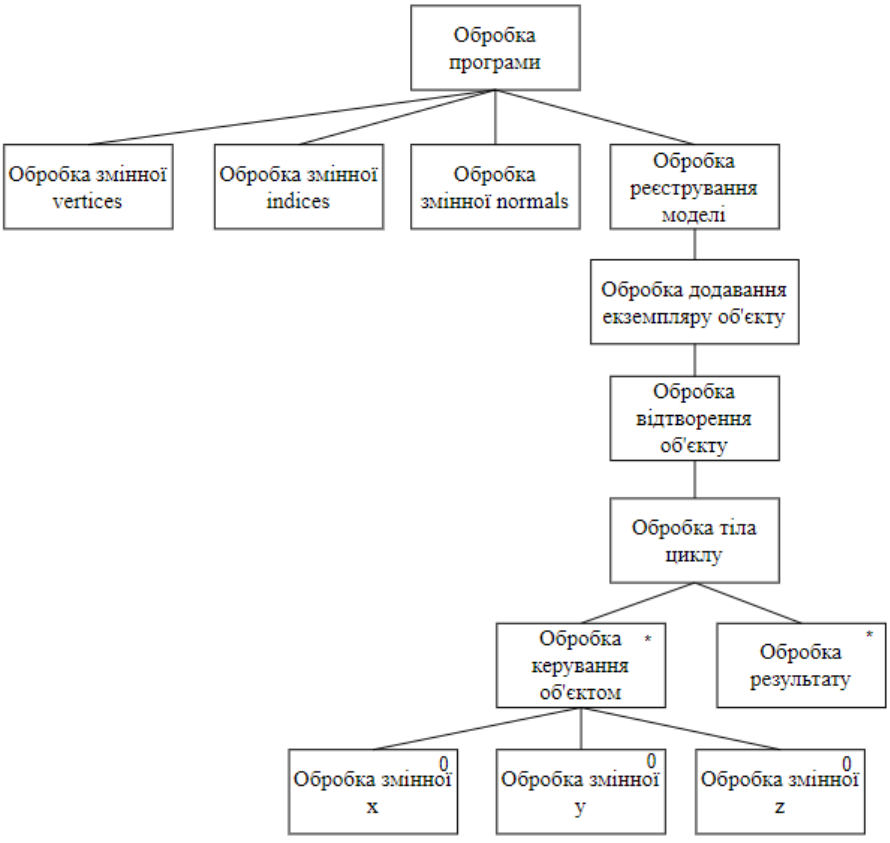

Рисунок 2.8 – Додавання не маючих співвідношень компонентів структури вихідних даних

2.5 Четвертий етап методу Джексона

В четвертому етапі перераховуються виконувані операцій. Наведемо перелік операцій для програми, що управляє структуру на метамові структурованого викладу.

Операції завершення. В даному випадку тільки одна операція завершення.

Операції відкриття і закриття. У даній програмі використовується два файли. Тому операції даної групи усього 4.

Операції виведення результатів. Для визначення операцій виведення досліджується структура вихідних даних.

Обчислення. Обчислення потрібні для генерації вихідних даних. Тому аналізується кожна з операцій виведення і визначається, які обчислення або обробка потрібні для генерації вихідних даних.

Операції введення вхідних даних. Усього в програмі дві змінні які треба ввести.

Управління внутрішніми змінними. Внутрішні змінні виявляються зазвичай в обчисленнях. При запуску програми деякі змінні мають бути порожніми В даній структурі такими змінними є vertices, indices, normals, x, y та z.

Повний набір виконуваних операцій в розглянутому прикладі містить такі операції:

а) стоп;

б) відкрити вхідні дані;

в) відкрити вихідні дані;

г) закрити вхідні дані;

д) закрити вихідні дані;

е) вивести результат операцій;

ж) x := x + 1;

з) y := y + 1;

и) z := z + 1;

к) x := x – 1;

л) y := y – 1;

м) z ;= z – 1;

н) введення vertices;

о) введення indices;

п) введення normals;

р) vertices := 0;

с) indices := 0;

т) normals := 0;

у) x := 0;

ф) y := 0;

х) z := 0.

Отже, в розглянутому прикладі усього 21 операцій. Розміщуємо виконані операції в структуру програми. Результат розміщення надано в рисунку 2.9.

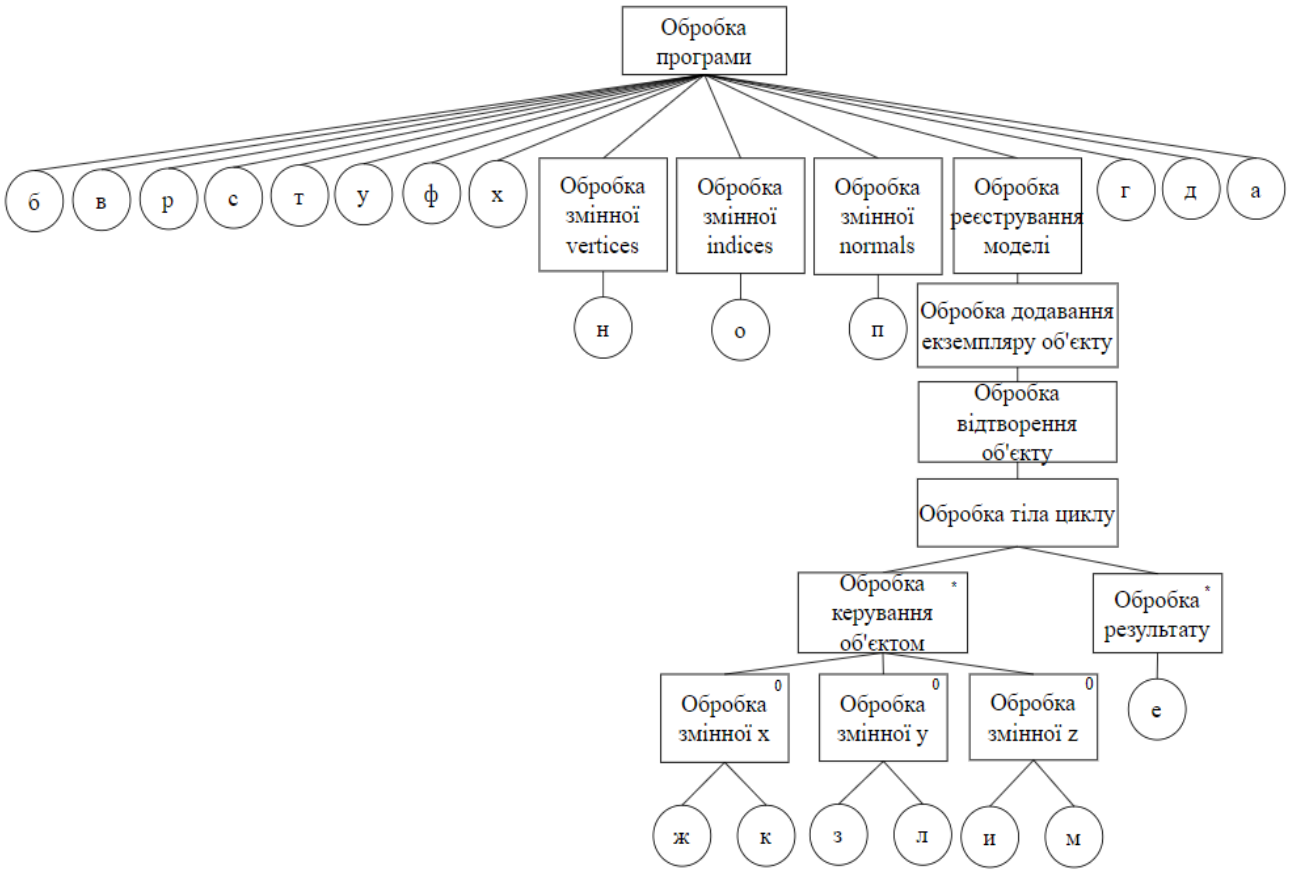

Рисунок 2.9 – Розміщення виконуваних операцій

Таким чином, у керуючій структурі програми розміщені всі виконувані операції. На наступному етапі створюється програма на метамову структурованого викладу.

2.6 П’ятий етап методу Джексона

В останньому етапі пишеться структурований виклад програми. Кожна з основних конструкцій, що використовуються у методі Джексона (послідовність, вибір, повторення), може бути записана на метамову структурованого викладу, що є однією з різновидів словесного опису алгоритму.

Структурований опис програми наведено в додатку А. Структурований виклад легко перетворюється в код програми, написаної на будь-якій мові програмування.

2.7 Висновки до другого розділу

На основі дій та інформації, описаних в цьому розділі, більш детально розглянуто завдання кваліфікаційної роботи, описано вхідні, вихідні дані та результат даного завдання.

Методом структури програмування Джексона сформовано структурований виклад програми, на основі якого можна створювати виконавчу частину вебзастосунку.

На першому етапі методу побудовано таблиці, а згодом й схеми відображення набору вхідних та вихідних даних. В структурі даних усі входи та виходи ідентифіковані, кожний компонент даних ідентифіковано іменем.

На другому етапі ідентифіковано відповідності між наборами компонентів вхідних та вихідних даних, встановлено 2 відповідності.

На третьому етапі сформовано керуючу структуру даних. На основі результатів другого етапу побудовано спрощену структуру, згодом додано не маючих співвідношень компоненти вхідних даних, потім під’єднано компонент вихідних даних.

На четвертому етапі проаналізовано та визначено виконувані операції та додано їх до керуючої структури.

На п’ятому етапі за отриманою схемою сформульовано структурований виклад вебзастосунку.

3 РЕАЛІЗАЦІЯ КОМПОНЕНТІВ WEBGL

В даному розділі описано розробка компонентів візуалізації числових даних з використанням технологій WebGL.

3.1 Представлення полотна

Даний компонент (див. рис. 3.1) представляє елемент canvas для відтворення графіки WebGL.

```
import React from 'react';
import init from './Init';
import './webgl.css';
export default class WebGL extends React.Component {
   componentDidMount(){
      init('webgl');
   }
   render(){
      return <canvas id="webgl" width="800" height="500" className='canva'></canvas>;
   }
}
```

Рисунок 3.1 – Код компоненту полотна

Метод життєвого циклу використовується для запуску ініціалізації програми WebGL під час монтування компонента. Функція init викликається з ідентифікатором елемента canvas. Цей метод зазвичай використовується для дій, які необхідно виконати після додавання компонента до DOM.

Метод відтворення повертає об’єкт JSX, що визначає структуру компонента. У цьому випадку він візуалізує елемент canvas.

Функція ініціалізації викликається в методі життевого циклу, передаючи ідентифікатор 'webgl'. Логіка ініціалізації WebGL розташована в окремому файлі компоненту ініціалізації.

3.2 Компонент ініціалізації

Даний компонент (див. Додаток Б) відповідає за ініціалізацію додатку WebGL, налаштування необхідних компонентів і обробку взаємодії користувача [11].

Спочатку даний компонент намагається отримати елемент canvas і елемент введення файлу за допомогою наданого ідентифікатора. Якщо полотно або вхідний файл не знайдено, функція вказує, що ініціалізація не може бути продовжена.

Прослуховувач подій прослуховує зміни у вхідному файлі. Коли вибрано файл, він створює об’єкт для читання вмісту файлу. Зворотний виклик onload FileReader запускається після завантаження вмісту файлу. У цьому зворотному виклику аналізується вміст JSON файлу. Проаналізовані дані JSON використовуються для вилучення вершин, індексів і нормалей.

Об’єкт ModelType створюється з використанням витягнутих даних, а джерело світла створюється. Створюється екземпляр ModelRenderer, і новий тип моделі реєструється з ідентифікатором ('objectID').

Створюються екземпляри Camera та ModelInstance. Екземпляри додаються до modelRender.

Також ініціалізуються повзунки для керування обертанням моделі та до них додаються прослуховувачі подій. Ці слухачі оновлюють обертання екземпляра моделі.

В функції відтворювання очищається полотно, оновлюється обертання моделі. Функція відтворює сцену за допомогою і запитує наступний кадр анімації.

3.3 Компонент з методами технології WebGL

В компоненті (див. Додаток В) міститься службовий клас, який абстрагує типові операції WebGL, полегшуючи взаємодію з контекстом WebGL [12]. Даний клас містить методи:

- ініціалізації класу контекстом відтворення WebGL;
- очищення буферів кольору та глибини;
- встановлення розмірів вікна перегляду;
- створення, зв’язування, розв’язування та додавання даних до буферів масиву та буферів масиву елементів;
- створення вершинних і фрагментних шейдерів, їх компіляції, створення шейдерних програм, приєднання шейдерів до програм, зв’язування програм та їх використання;
- отримання розташування атрибутів, увімкнення масивів атрибутів вершин, вказівки атрибутів на буфери вершин, малювання об’єкту на основі трикутників, завантаження матриць, векторів, чисел і логічних значень у шейдерні програми.

Екземпляр компоненту забезпечує єдиний шаблон, що дозволяє спільно використовувати один екземпляр класу у програмі.

3.4 Компонент шейдерів

Компонент шейдерів містить фрагментний шейдер та шейдер вершин, які використовуються для освітлення та трансформацій [13; 14]. Надає програму шейдера для візуалізації. Також компонент надає константи для спрощеного використання в файлах шейдерів.

Фрагментний шейдер (див. рис. 3.2) реалізує базове дифузне освітлення, поєднуючи базовий колір з інтенсивністю освітлення на основі скалярного добутку нормалі поверхні та світлового вектора. Результат впливає на остаточний колір кожного пікселя.

```
import Const from "./constants";
export default `
   precision mediump float;
   varying vec3 surfaceNormal;
   varying vec3 lightVector;
   uniform float ${Const.LIGHT_AMBIENT};
   uniform vec3 ${Const.LIGHT_COLOR};
   float _nDot() {
      vec3 unitNormal = normalize(surfaceNormal);
      vec3 unitLightVector = normalize(lightVector);
      return dot(unitNormal, unitLightVector);
   }
   vec4 diffuseLighting(){
      float brightness = max(_nDot(), ${Const.LIGHT_AMBIENT});
      return vec4(${Const.LIGHT_COLOR}.rgb * brightness, 1.0);
   }
   void main(void) {
      gl_FragColor = vec4(0.38, 0.85, 0.98, 1.0) * diffuseLighting();
   }
`;
```

Рисунок 3.2 – Код фрагментного шейдеру

Шейдер вершин (див. Додаток Г.1) відповідає за перетворення вхідних вершин із простору об’єктів у світовий простір, обчислення нормалі до поверхні та підготовку даних для інтерполяції у фрагментному шейдері.

Головний клас (див. Додаток Г.2) інкапсулює всі налаштування шейдерів, від компіляції та зв’язування до керування атрибутами та однорідними розташуваннями. Він надає методи для ввімкнення певних атрибутів і встановлює програму шейдера як активну програму для відтворення. Конструктор створює екземпляри вершинних і фрагментних шейдерів, приєднує їх до програми та зв’язує програму. Потім ця програма зберігається для подальшого використання. Клас містить метод перевірки статусу компіляції шейдера та реєстрації будь-яких помилок у разі невдачі компіляції. Метод 'use' активує програму шейдера, роблячи її поточною програмою для

відтворення. В даному класі також реалізовані методи вмикання атрибутів позиції і нормалі. Ці методи зазвичай викликаються перед рендерингом, щоб визначити, які атрибути використовуються в процесі рендерингу.

Також компонент містить константи для атрибутів і уніфікованих імен змінних (див. рис. 3.3), які використовуються в шейдерах.

```
export default {
   POSITION: 'position',
   NORMAL: 'normal',
   TRANSFORMATION_MATRIX: 'transformationMatrix',
   VIEW_MATRIX: 'viewMatrix',
   PROJECTION_MATRIX: 'projectionMatrix',
   LIGHT_POSITION: 'lightPosition',
   LIGHT_COLOR: 'lightColor',
   LIGHT_AMBIENT: 'lightAmbient',
}
```

Рисунок 3.3 – Константи атрибутів та уніфіковані імена змінних

3.5 Компоненти моделі

Задача компонентів моделі полягає в спрощуванні керування та відтворення 3D-моделей у програмі WebGL. Компонент складається з класів екземпляру з методами для оновлення її матриці обертання та типу методами генерації та зв’язування буферів вершин, нормалей та індексів.

3.5.1 Компонент екземпляру

Клас екземпляру керує положенням, обертанням і масштабуванням екземплярів [15]. Конструктор ініціалізує екземпляр із наданими параметрами позиції, повороту та масштабу (див. рис. 3.4).

```
constructor(x,y,z,rx,ry,rz,scale){
	this.x = x;
	this.y = y;
	this.z = z;
	this.rx = rx;
	this.ry = ry;
	this.rz = rz;
	this.scale = scale;
}
```

Рисунок 3.4 – Конструктор класу екземпляру

Клас автоматично генерує матрицю перетворення, викликаючи метод, що генерує матрицю перетворення на основі поточної позиції, повороту та масштабу. Матриця необхідна для візуалізації моделі в правильному місці та орієнтації.

Клас містить метод, який дозволяє оновлювати кути повороту. Після оновлення він запускає регенерацію матриці перетворення. Методи класу наведено на рисунку 3.5.

```
updateRotation = (rx,ry,rz) => {
	this.rx = rx;
	this.ry = ry;
	this.rz = rz;
	this.updateTransformationMatrix();
   }
updateTransformationMatrix = () => {
	this.transformationMatrix =
createTransformationMatrix(this.x,this.y,this.z,this.rx,this.ry,this.rz,this.scale);
}
getTransformationMatrix = () => this.transformationMatrix;
```

Рисунок 3.5 – Методи класу екземпляру

В файлі класу імпортовано компонент з матричними операціями.

3.5.2 Компонент типу

Клас типу (див. рис. 3.6) представляє тип 3D-моделі з методами для створення необхідних буферів і керування ними. Метод використання особливо важливий для підготовки моделі до візуалізації шляхом зв’язування буферів і ввімкнення атрибутів у шейдері. Цей клас інкапсулює дані та операції, пов’язані з певним типом 3D-моделі у програмі WebGL.

Конструктор ініціалізує 3D-об’єкт вершинами, індексами вершин та нормалями та викликає буфери відповідно до створених змінних.

```
constructor(vertices, indices, normals) {
    this.vertices = vertices;
    this.indices = indices;
    this.normals = normals;
    this.genVertices();
    this.genIndices();
    this.genNormals();
}
```

Рисунок 3.6 – Констуктор класу типу

Код буферів змінних наведено на рисунку 3.7.

```
genNormals = () => {
    this.normalBuffer = GLT.createBuffer();
    GLT.bindArrayBuffer(this.normalBuffer);
    GLT.addArrayBufferData(this.normals);
    GLT.unbindArrayBuffer();}
genVertices = () => {
    this.vertexBuffer = GLT.createBuffer();
    GLT.bindArrayBuffer(this.vertexBuffer);
    GLT.addArrayBufferData(this.vertices);
    GLT.unbindArrayBuffer();}
```

```
genIndices = () => {
    this.indexBuffer = GLT.createBuffer();
    GLT.bindElementArrayBuffer(this.indexBuffer);
    GLT.addElementArrayBufferData(this.indices);
    GLT.unbindElementArrayBuffer();}
```

Рисунок 3.7 – Методи генерації буферів

Метод 'use' (див. рис. 3.8) використовується для зв’язування необхідних буферів і ввімкнення відповідних атрибутів у наданому шейдері.

```
use = (shader) => {
    GLT.bindArrayBuffer(this.vertexBuffer);
    shader.enablePosition();
    GLT.bindArrayBuffer(this.normalBuffer);
    shader.enableNormals();
    GLT.bindElementArrayBuffer(this.indexBuffer);
}
```

Рисунок 3.8 – Метод 'use'

Це має вирішальне значення для візуалізації 3D-моделі, пов’язаної з цим об’єктом.

3.6 Компонент відтворювання

Компонент відтворювання (див. Додаток Д) організовує візуалізацію 3D-моделей у програмі WebGL. Він керує шейдерами, обробляє реєстрацію типів моделей та їхніх екземплярів і виконує процес відтворювання шляхом повторення зареєстрованих моделей та екземплярів.

Конструктор ініціалізує об’єкт новим екземпляром класу шейдеру і порожнім об’єктом для зберігання інформації про модель.

В класі реалізовано метод реєстрування нового типу моделі, пов’язаний його з ідентифікатором. Він ініціалізує запис в об’єкті models, створюючи масив для екземплярів.

Він пов’язує примірник із певним типом моделі за допомогою наданого ідентифікатора. Цей метод використовується для заповнення екземплярів, пов’язаних із зареєстрованою моделлю.

В класі реалізовано метод для додавання екземпляру до засобу відтворення. Він пов’язує примірник із певним типом моделі за допомогою наданого ідентифікатора. Цей метод використовується для заповнення екземплярів, пов’язаних із зареєстрованою моделлю.

Метод попередньої візуалізації налаштовує середовище візуалізації, налаштовуючи вікно перегляду та вмикаючи тестування глибини.

Метод відтворювання відповідає за візуалізацію усіх зареєстрованих моделей та їх екземплярів. Він налаштовує середовище візуалізації, використовує шейдер, вмикає світло та застосовує камеру. Потім він повторює зареєстровані моделі та їхні екземпляри, використовуючи шейдер для ввімкнення матриць трансформації та відтворення кожного екземпляра.

3.7 Компоненти взаємодії

Компоненти для керування сценою спільно обробляють рух камери, події миші та службові функції для перетворення матриці у програмі. Вони відіграють вирішальну роль у взаємодії з користувачем і відображенні.

3.7.1 Компоненти налаштування

Компонент утиліти (див. рис. 3.9) містить службові функції для звичайних математичних операцій, що використовуються в комп’ютерній

графіці, що полегшує виконання трансформацій і перетворень у програмі. В даному компоненті є допоміжна функція, яка перетворює градуси в радіани. Вона зазвичай використовується в комп'ютерній графіці для перетворення кутів повороту з градусів у радіани, оскільки багато тригонометричних функцій у JavaScript використовують радіани. Компонент ініціалізує матрицю ідентичності, а потім послідовно застосовує перетворення переміщення, обертання та масштабування. Повертає отриману матрицю перетворення.

```
import {mat4, vec3} from 'gl-matrix';
export const toRadians = (deg) => deg * (Math.PI/180);
export const createTransformationMatrix = (x,y,z,rx,ry,rz,scale) => {
    const matrix = [];
    mat4.identity(matrix);
    mat4.translate(matrix,matrix,vec3.fromValues(x,y,z));
    mat4.rotateX(matrix,matrix,toRadians(rx));
    mat4.rotateY(matrix,matrix,toRadians(ry));
    mat4.rotateZ(matrix,matrix,toRadians(rz));
    mat4.scale(matrix,matrix,vec3.fromValues(scale,scale,scale));
    return matrix;
}
```

Рисунок 3.9 – Компонент налаштування

Клас для прослуховування миші (див. Додаток Е.1) забезпечує механізм обробки подій миші, таких як прокручування коліщатка та перетягування. Він відстежує рухи миші та сповіщає зареєстрованих слухачів про такі події, дозволяючи іншим частинам програми реагувати на дії користувача. Конструктор ініціалізує масиви для зберігання слухачів подій колеса та перетягування.

Метод ініціалізації налаштовує прослуховувачі подій колеса та руху миші на полотні. Він відстежує початкове положення миші і те, чи перетягує миша в даний момент. Подія колеса запускається при використанні колеса, а події руху

обробляються при натисканні та відпусканні кнопок миші. Клас обчислює зміну положення миші під час перетягування та сповіщає слухачів подій перетягування.

Клас камери (див. Додаток Е.2) надає спосіб представлення та керування переглядом об’єкту. Він обчислює матриці перегляду та проекції на основі своїх властивостей, дозволяючи інтеграцію з іншими компонентами, такими як шейдери, для візуалізації сцен із заданої точки зору камери [16; 17].

Конструктор ініціалізує властивості камери, включаючи її початкове положення, обертання, ближню та дальню площини відсікання та поле зору. Він підписує камеру на перетягування мишею та події коліщатка, щоб користувач міг взаємодіяти.

В класі реалізовано перетягування метод викликається під час перетягування миші. Він оновлює положення камери на основі зміни координат миші. Після оновлення позиції він викликає метод для оновлення матриці перегляду.

Метод прокручування коліщатки миші. Оновлює третю координату камери на основі дельти колеса. Після її оновлення він викликає метод для оновлення матриці перегляду.

Використовується метод ввімкнення камери шляхом передачі її матриць перегляду та проекції шейдеру. В класі є метод створення матриці перегляду, що застосовує повороти та переміщення на основі повороту та положення камери. Реазілзовано метод створення матриці проекції за допомогою формули перспективної проекції.

3.7.2 Компонент керування

Для використання компонентів, наведенених в попередньому пункті, потрібно розробити HTML-компонент. Компонент повзунків (див. рис. 3.10) розроблено як елемент інтерфейсу користувача для налаштування кутів

обертання навколо різних осей, забезпечуючи візуальний та інтерактивний спосіб керування обертаннями в 3D-середовищі.

```
import React from 'react';
import './sliders.css';
export default class RotationSliders extends React.Component {
   render(){
      return<div id="rotationSliders">
      <div>
        <label htmlFor="xRotationSlider">X Rotation:</label><br/>
        <input type="range" id="xRotationSlider" min="-180" max="180" step="1"/>
      </div>
      <div>
        <label htmlFor="yRotationSlider">Y Rotation:</label><br/>
        <input type="range" id="yRotationSlider" min="-180" max="180" step="1"/>
      </div>
      <div>
        <label htmlFor="zRotationSlider">Z Rotation:</label><br/>
        <input type="range" id="zRotationSlider" min="-180" max="180" step="1"/>
      </div>
    </div>
   }
}
```

Рисунок 3.10 – Компонент керування

Для більш коректного оформлення та додання імен повзунків елементи керування було додано в елементи мітків. Мітки прив’язані до повзунків через ідентифікатори.

Повзунки мають спеціальні атрибути, щоб визначити діапазон і деталізацію обертання. Кожний повзунок має свій ідентифікатор, який використовується в компоненті ініціалізації для передачі даних про положення. Крок повзунків при перетастуванні дорівнює 1. Максимальне значення повзунка дорівнює 180, мінімальне дорівнює -180. Це дозволяє обертати

згенерований застосунком об’єкт на 360 градусів. Початкове значення повзунків встановлено за замовчуваннм.

Компонент керування реалізовано та повзунки функціонують без помилок. Даний HTML-інструмент можна додавати до головного файлу HTML-сторінки.

3.8 Висновки до третього розділу

В третьому розділі описано розробку компонентів, потрібних для візуалізації числових даних. Кожен компонент відіграє важливу роль при взаємодії з числовими даними та їх відображенням.

Компонент представлення містить простір, в якому відображається імпортований тривимірний об’єкт.

Компонент ініціалізації відповідає за ініціалізацію додатку WebGL, налаштування необхідних компонентів і обробку взаємодії користувача. Даний компонент створений на основі структурованого опису, побудований за методом Джексона.

Компонент з методами WebGL містить спрощені версії методів для їх простішого використання в іншин компонентах.

Компонент шейдерів відповідає за відображення освітлення та тіней.

Класи компонентів моделі відповідають за спрощення в керуванні тривимірних моделей та їх подальше відтворення у вебзастосунок.

Компонент відтворювання візуалізує 3D-об’єкт у вебдодатку, обробляючи реєстрацію типів моделей та екземплярів і відтворює їх шляхом повторення.

Компоненти взаємодії дозволяють користувачу керувати відображенням згенерованим на полотні об’єктом.

Усі перелічені компоненти готові до використання у нашому вебзастосунку.

4 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ ТА ТЕСТУВАННЯ

В цьому розділі описано реалізацію вебдодатку з представленням скриншотів її роботи. Також було проведено тестування реалізованого вебзастосунку.

4.1 Реалізація

Додаємо до сторінки полотно, в якому відтворюється наша модель, повзунки обертання для контролю об’єкту та можливість завантажувати файл формату JSON (див. рис. 3.1).

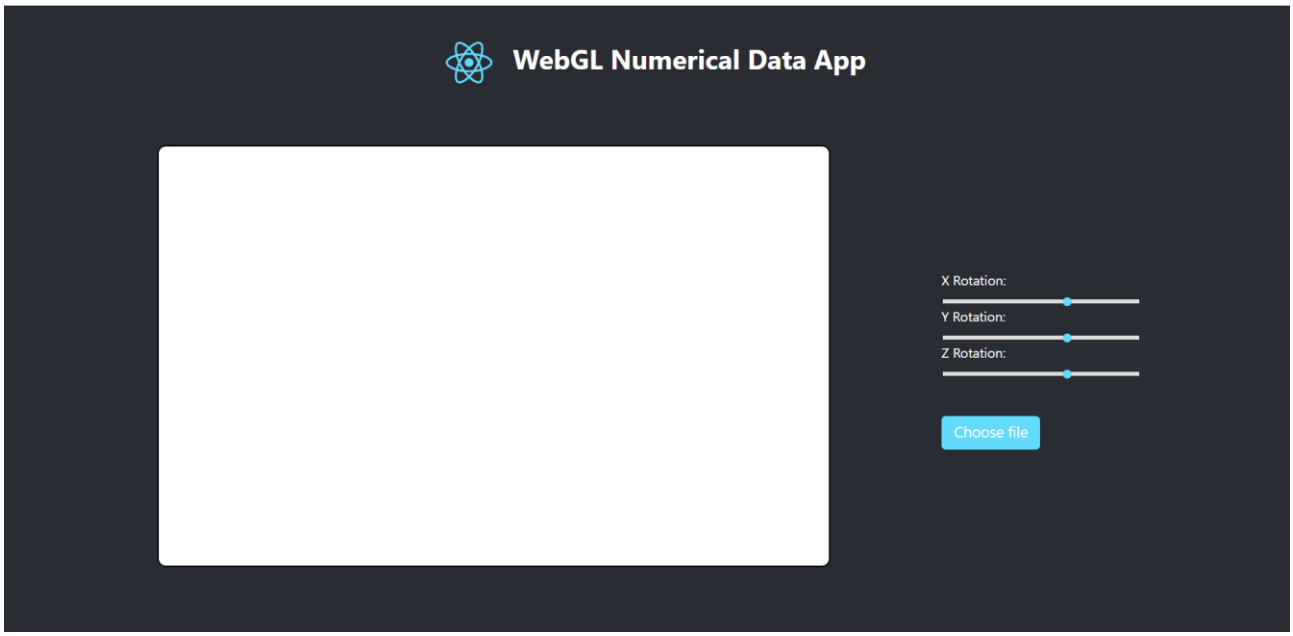

Рисунок 4.1 – Запуск вебзастосунку

4.2 Тестування додатку

Для тестування розробленого додатку було підготовлено 3 файли формату JSON, які містять дані для візуалізації об’єктів, вказаних нижче.

Першим буде протестовано файл з даними кубу. Для початку треба натиснути кнопку «Choose File», якак викликає діалогове вікно провідника. Обирається потрібний файл з числовими даними. Після передачі файлу вебзастосунок візуалізує отримані числові дані та виводить об’єкт на полотно. Результат наведено в рисунку 4.2.

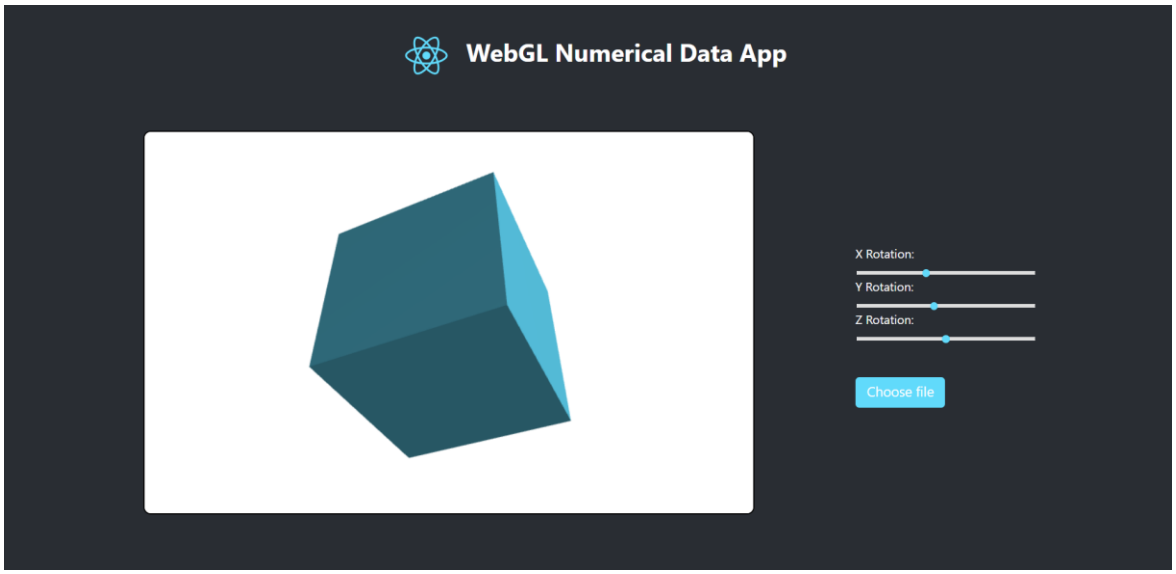

Рисунок 4.2 – Генерування кубу

Наступним об’єктом буде чотирикутна піраміда. Результат наведено на рисунку 4.3.

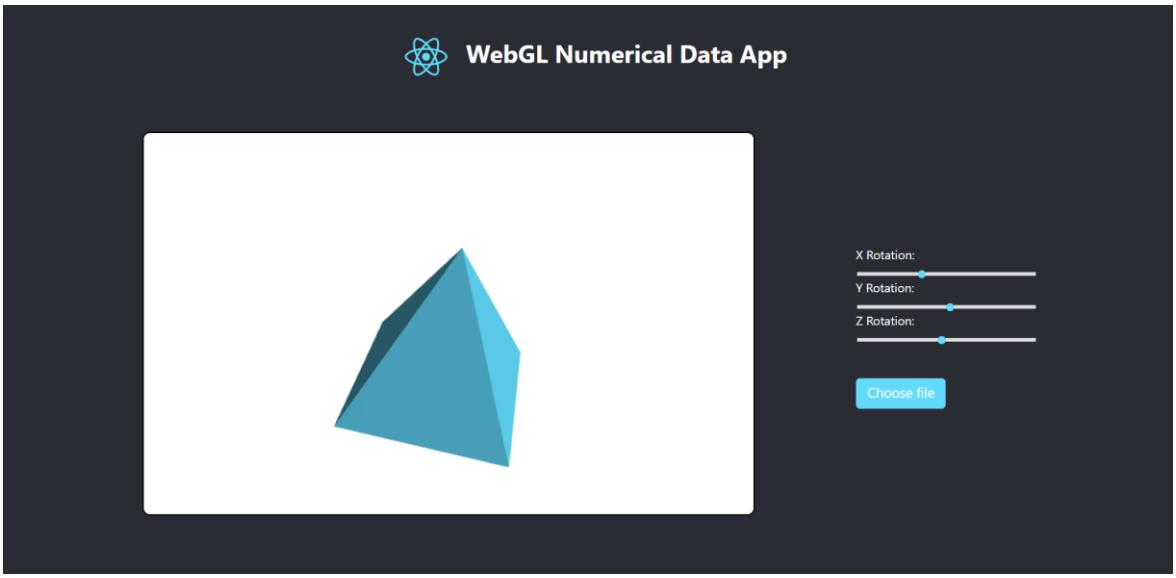

Рисунок 4.3 – Генерування чотирикутної піраміди

Останнім об’єктом для генерації буде трикутна піраміда. Як і з попередніми прикладами, даний об’єкт відтворюється вдало. Результат відтворення представлено в рисунку 4.4.

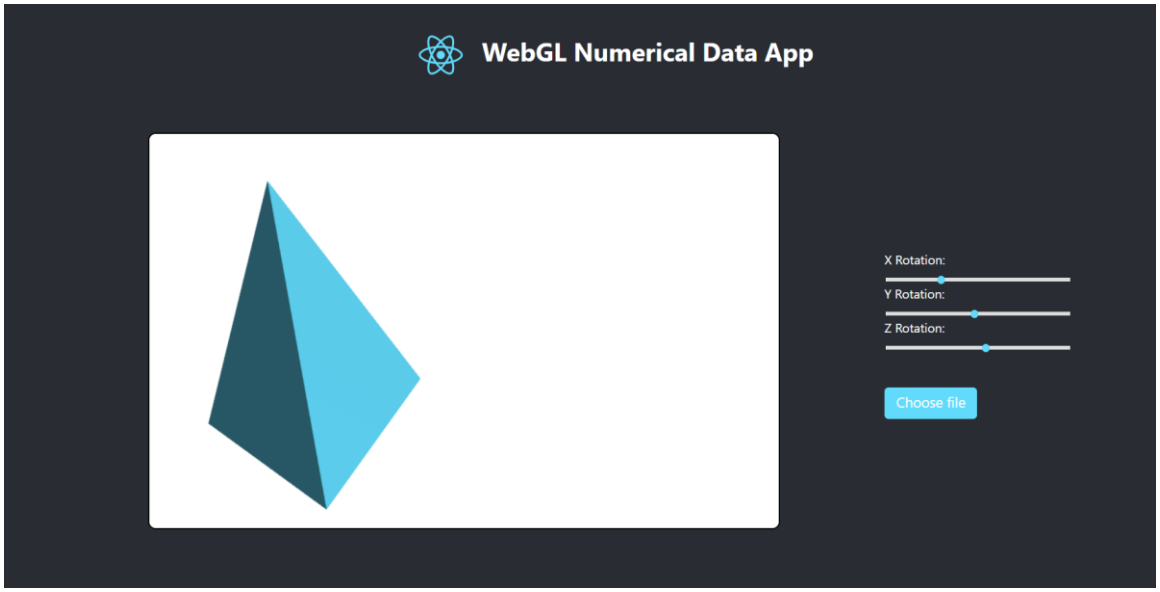

Рисунок 4.4 – Генерування трикутної піраміди

Під час перевірки генерації моделі проведено тестування переміщення та зміни кута об’єкту. Перелічені операції працювали без помилок.

4.3 Висновки до четвертого розділу

У результаті роботи був реалізований вебзастосунок для візуалізації 3D-об’єктів з використанням технології WebGL. Розроблений інтерфейс дозволяє користувачеві імпортувати дані у форматі JSON, а отриманий результат виводиться на HTML-сторінці.

Тестування додатку було проведено з використанням трьох різних файлів JSON, що містять дані для генерації кубу, чотирикутної піраміди та трикутної піраміди. Всі три об’єкти були вдало візуалізовані, що свідчить про правильну роботу програми. Також було успішно протестовано функціонал переміщення, віддалення та обертання об’єкта.

Отже, вебзастосунок відповідає вимогам, поставленим в завданні, і може ефективно використовуватися для візуалізації та аналізу 3D-моделей. Розроблене рішення відзначається зручним інтерфейсом та стабільною роботою, що робить його потенційно корисним для широкого кола користувачів.

ВИСНОВКИ

У ході аналізу роботи було проведено дослідження програмного забезпечення та засобів, необхідних для розробки вебзастосунку. Ознайомлено з особливостями технології WebGL. Досліджено метод структури програмування Джексона, необхідного для проєктування вебзастосунку.

У ході аналізу роботи було визначені вимоги, вхідні дані, вихідні дані та результат, які слугують для вирішення поставленого завдання. Спроєктовано структурований опис застосунку, виконуючи кожний етап методу Джексона.

Розроблено компоненти, необхідні для вводу імпорту даних, їх обробки та виведення кінцевого результату на екран.

Отже в ході роботи розроблено вебзастосунок візуалізації числових даних з використанням технології WebGL. Згідно результатам тестування пітверджено коректність отриманих результатів та роботи реалізованого вебдодатку. Отже застосунок відповідає вимогам.

вебзастосунок може бут використано в будь-якому браузері, який підтримує останні вебстандарти.

Дана робота покращила знання з використання технології візуалізації графічних об’єктів WebGL. Програмної платформи Node.js та роботи з фреймворком для розробки користувацьких інтерфейсів React.js. Також під час виконання роботи оновлено знання з проєктування програмного забезпечення методом Джексона.

ПЕРЕЛІК ПОСИЛАНЬ

1. Contributors to Wikimedia projects. WebGL. Wikipedia, the free encyclopedia. URL: https://en.wikipedia.org/wiki/WebGL (дата звернення: 05.09.2023).
2. MDN. Data in WebGL – Web APIs. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebGL_API/Data (дата звернення: 05.09.2023).
3. Contributors to Wikimedia projects. JavaScript. Wikipedia, the free encyclopedia. URL: https://uk.wikipedia.org/wiki/JavaScript (дата звернення: 05.09.2023).
4. DevZone. Що таке Node.js? Основи серверної розробки на JavaScript. URL: https://devzone.org.ua/post/shho-take-nodejs-osnovi-servernoyi-rozrobki-na-javascript (дата звернення: 05.09.2023).
5. Jones B., Mackenzie C. glMatrix. URL: https://glmatrix.net/ (дата звернення: 10.09.2023).
6. Theteacher.info Ltd. Jackson Structured Programming (JSP) and decomposition. URL: https://theteacher.info/index.php/algorithms-and-problem-solving-2/problem-solving-and-programming/2-2-2-all-topics/3445-jackson-structured-programming-jsp-and-decomposition (дата звернення: 11.10.2023).
7. Keywood Computer Services Limited. Jackson Structured Programming. Jackson Workbench CASE Tool for JSP and JSD Software Development. URL: http://www.jacksonworkbench.co.uk/jsp.htm (дата звернення: 01.10.2023).
8. Jackson Structured Programming (JSP). Algorythm desing methods. URL: https://daniel-gce-al.weebly.com/uploads/1/4/1/2/1412714/4.2.2_jackson_structured_programming__jsp_.pdf (дата звернення: 01.10.2023).
9. Packt. Vertices and Indices. WebGL Beginner's Guide. URL:

https://subscription.packtpub.com/book/game-development/9781849691727/2/ch02lvl1sec21/vertices-and-indices#:~:text=Indices%20are%20numeric%20labels%20for,using%20a%20WebGL%20index%20buffer (дата звернення: 06.09.2023).

10. The Tech Edvocate. A Beginner's Guide: What Are Normals in 3D Modeling? URL: https://www.thetechedvocate.org/a-beginners-guide-what-are-normals-in-3d-modeling/ (дата звернення: 05.09.2023).
11. JR Development. WebGL React component – EP1, 2018. YouTube. URL: https://www.youtube.com/watch?v=wbI55mc7ksE (дата звернення: 05.09.2023).
12. JR Development. WebGL React component – EP2 – drawing a triangle, 2018. YouTube. URL: https://www.youtube.com/watch?v=sgforQ8Gan4&ab (дата звернення: 05.09.2023).
13. WebGL Fundamentals. WebGL Shaders and GLSL. URL: https://webglfundamentals.org/webgl/lessons/webgl-shaders-and-glsl.html (дата звернення: 05.09.2023).
14. JR Development. WebGL React component – EP4 – Lighting and cubing, 2018. YouTube. URL: https://www.youtube.com/watch?v=b4jkXtrZDks (дата звернення: 05.09.2023).
15. JR Development. WebGL React component – EP3 – move, rotate, scale, 2018. YouTube. URL: https://www.youtube.com/watch?v=3ZRurTUfSt0 (дата звернення: 05.09.2023).
16. WebGL Fundamentals. WebGL 3D – Cameras. URL: https://webglfundamentals.org/webgl/lessons/webgl-3d-camera.html (дата звернення: 05.09.2023).
17. JR Development. WebGL React component – EP6 – Camera, 2018. YouTube. URL: https://www.youtube.com/watch?v=KFEP5ZrzpP4 (дата звернення: 05.09.2023).

ДОДАТОК А

Структурований опис застосунку

Обробка програми (посл)

Відкрити вхідні дані

Відкрити вихідні дані

Введення vertices

Введення indices

Введення normals

Обробка реєстрування моделі (посл)

Обробка додавання екземпляру об’єкту (посл)

Обробка відтворювання об’єкту (посл)

Обробка тіла циклу (посл)

Обробка керування об’єкту (повт) доки НЕ «Вихід»

Обробка змінної x (виб) умова «Керування об’єктом» (посл)

x := x + 1

x := x - 1

Обробка змінної x (кінець)

Обробка змінної x (виб) умова «Керування об’єктом» (посл)

y := y + 1

y := y - 1

Обробка змінної y (кінець)

Обробка змінної z (виб) умова «Керування об’єктом» (посл)

z := z + 1

z := z - 1

Обробка змінної z (кінець)

Обробка керування об’єкту (кінець)

Обробка результату (повт) доки НЕ «Вихід»

Вивести результат операції

Обробка результату (кінець)

Обробка тіла циклу (кінець)

Обробка відтворювання об’єкту (кінець)

Обробка додавання екземпляру об’єкту (кінець)

Обробка реєстрування моделі (кінець)

Закрити вхідні дані

Закрити вихідні дан

Стоп

Обробка програми (кінець)

ДОДАТОК Б

Код ініціалізації вебдодатку

```
import GLT from '../GLTools/tools';
import ModelRenderer from '../Render/render';
import ModelType from '../Models/modelType';
import ModelInstance from '../Models/modelInstance';
import Camera from '../Camera';
import MouseEvent from '../EventHandlers/mouse';
import Light from '../LightSource';
export default (id) => {
    const canvas = document.querySelector(`#${id}`);
    const fileInput = document.getElementById('fileInput');
    if (!canvas || !fileInput) {
        return;
    }
    const gl = canvas.getContext('webgl');
    if (!gl) {
        return;
    }
    GLT.init(gl);
    MouseEvent.init();
    fileInput.addEventListener('change', (event) => {
        const file = event.target.files[0];
        if (file) {
            const reader = new FileReader();
            reader.onload = (e) => {
                const jsonContent = e.target.result;
                const objectData = JSON.parse(jsonContent);
```

```
                const vertices = objectData.vertices;
                const indices = objectData.indices;
                const normals = objectData.normals;
                const modelRender = new ModelRenderer();
                const modelType = new ModelType(vertices, indices, normals);
                const light = new Light(100, 100, -100, 1.0, 1.0, 1.0, 0.4);
                modelRender.registerNewModel(modelType, 'objectID');
                const camera = new Camera();
                const instance = new ModelInstance(0, 0, 0, 0, 0, 0, 0.5);
                modelRender.addInstance(instance, 'objectID');
                const xRotationSlider =
document.getElementById('xRotationSlider');
                const yRotationSlider =
document.getElementById('yRotationSlider');
                const zRotationSlider =
document.getElementById('zRotationSlider');
                xRotationSlider.setAttribute('value', 0);
                yRotationSlider.setAttribute('value', 0);
                zRotationSlider.setAttribute('value', 0);
                xRotationSlider.addEventListener('input', () => {
                   instance.updateRotation(parseFloat(xRotationSlider.value), 0, 0);
                });
                yRotationSlider.addEventListener('input', () => {
                   instance.updateRotation(0, parseFloat(yRotationSlider.value), 0);
                });
                zRotationSlider.addEventListener('input', () => {
                   instance.updateRotation(0, 0, parseFloat(zRotationSlider.value));
                });
                const render = () => {
                   GLT.clear(200.0, 1.0, 1.0, 1.0);
```

```
                    instance.updateRotation(parseFloat(xRotationSlider.value),
parseFloat(yRotationSlider.value), parseFloat(zRotationSlider.value));
                    modelRender.render(camera, light);
                    window.requestAnimationFrame(render);
                };
                window.requestAnimationFrame(render);
            };
            reader.readAsText(file);
        }
    });
};
```

ДОДАТОК В

Код компоненту з абстрагованими операціями WebGL

```
class GLTools {
    init(gl){
        this.gl = gl;
    }
    clear = (r, g, b, a) => {
        this.gl.clearColor(r, g, b, a);
        this.gl.clear(this.gl.COLOR_BUFFER_BIT |
this.gl.DEPTH_BUFFER_BIT);
    }
    viewport = () => this.gl.viewport(0, 0, this.gl.canvas.width,
this.gl.canvas.height);
    depthTest = (use) => use ? this.gl.enable(this.gl.DEPTH_TEST) :
this.gl.disable(this.gl.DEPTH_TEST);
    createBuffer = () => this.gl.createBuffer();
    bindArrayBuffer = (buffer) => this.gl.bindBuffer(this.gl.ARRAY_BUFFER,
buffer);
    unbindArrayBuffer = () => this.gl.bindBuffer(this.gl.ARRAY_BUFFER,
null);
    addArrayBufferData = (vertices) =>
this.gl.bufferData(this.gl.ARRAY_BUFFER, new Float32Array(vertices),
this.gl.STATIC_DRAW);

    bindElementArrayBuffer = (buffer) =>
this.gl.bindBuffer(this.gl.ELEMENT_ARRAY_BUFFER, buffer);
    unbindElementArrayBuffer = () =>
this.gl.bindBuffer(this.gl.ELEMENT_ARRAY_BUFFER, null);
```

```
            addElementArrayBufferData = (indices) =>
this.gl.bufferData(this.gl.ELEMENT_ARRAY_BUFFER, new Uint16Array(indices),
this.gl.STATIC_DRAW);
            createVertexShader = () =>
this.gl.createShader(this.gl.VERTEX_SHADER);
            createFragmentShader = () =>
this.gl.createShader(this.gl.FRAGMENT_SHADER);
            addShaderSource = (shader, source) => this.gl.shaderSource(shader, source);
            compileShader = (shader) => this.gl.compileShader(shader);
            createShaderProgram = () => this.gl.createProgram();
            attachShaderToProgram = (program, shader) =>
this.gl.attachShader(program, shader);
            linkProgram = (program) => this.gl.linkProgram(program);
            useProgram = (program) => this.gl.useProgram(program);
            getAttribLocation = (program, attribute) =>
this.gl.getAttribLocation(program, attribute);
            enableVertexAttribArray = (attribute) =>
this.gl.enableVertexAttribArray(attribute);
            pointToAttribute = (data, dimensions) => this.gl.vertexAttribPointer(data,
dimensions, this.gl.FLOAT, false, 0, 0);
            drawTriangles = (noOfIndices) =>
this.gl.drawElements(this.gl.TRIANGLES, noOfIndices,
this.gl.UNSIGNED_SHORT, 0);
            uploadMatrix4fv = (location,matrix) =>
this.gl.uniformMatrix4fv(location,false,matrix);
            getUniformLocation = (program,uniform) =>
this.gl.getUniformLocation(program,uniform);
            uploadVec3f = (location, vec3) => this.gl.uniform3fv(location, vec3);
            uploadFloat = (location, value) => this.gl.uniform1f(location, value);
        }
```

```
const GLT = new GLTools();
export default GLT;
```

ДОДАТОК Г

Коди компонентів шейдерів

Г.1 Шейдер вершин

```
import Const from "./constants";
export default `
   precision mediump float;
   attribute vec3 ${Const.POSITION};
   attribute vec3 ${Const.NORMAL};
   varying vec3 surfaceNormal;
   varying vec3 lightVector;
   varying vec2 pass_textureCoords;
   uniform mat4 ${Const.TRANSFORMATION_MATRIX};
   uniform mat4 ${Const.VIEW_MATRIX};
   uniform mat4 ${Const.PROJECTION_MATRIX};
   uniform vec3 ${Const.LIGHT_POSITION};
   vec4 getWorldPosition() {
      return ${Const.PROJECTION_MATRIX} * ${Const.VIEW_MATRIX}
* ${Const.TRANSFORMATION_MATRIX} * vec4(${Const.POSITION}, 1.0);
   }
   vec3 getSurfaceNormal() {
      return (${Const.PROJECTION_MATRIX} * ${Const.VIEW_MATRIX}
*  ${Const.TRANSFORMATION_MATRIX}  *  vec4(${Const.NORMAL},
0.0)).xyz;
   }
   void main(void) {
      vec4 worldPos = getWorldPosition();
      surfaceNormal = getSurfaceNormal();
```

```
        lightVector = ${Const.LIGHT_POSITION} - worldPos.xyz;
        gl_Position = worldPos;
    }
`;
```

Г.2 Головний клас шейдерів

```
import GLT from '../../GLTools/tools';
import VertexSource from './vertex';
import FragmentSource from './fragment';
import Const from './constants';
export default class ModelShader {
    constructor(){
        const vertexShader = GLT.createVertexShader();
        GLT.addShaderSource(vertexShader, VertexSource);
        GLT.compileShader(vertexShader);
        this.compileStatus(vertexShader);
        const fragmentShader = GLT.createFragmentShader();
        GLT.addShaderSource(fragmentShader, FragmentSource);
        GLT.compileShader(fragmentShader);
        this.compileStatus(fragmentShader);
        const program = GLT.createShaderProgram();
        GLT.attachShaderToProgram(program, vertexShader);
        GLT.attachShaderToProgram(program, fragmentShader);
        GLT.linkProgram(program);
        this.positionAttribute = GLT.getAttribLocation(program,
Const.POSITION);
        this.normalAttribute = GLT.getAttribLocation(program,
Const.NORMAL);
```

```
            this.transformationMatrix = GLT.getUniformLocation(program,
Const.TRANSFORMATION_MATRIX);
            this.viewMatrix = GLT.getUniformLocation(program,
Const.VIEW_MATRIX);
            this.projectionMatrix = GLT.getUniformLocation(program,
Const.PROJECTION_MATRIX);
            this.lightPosition = GLT.getUniformLocation(program,
Const.LIGHT_POSITION);
            this.lightColor = GLT.getUniformLocation(program,
Const.LIGHT_COLOR);
            this.lightAmbient = GLT.getUniformLocation(program,
Const.LIGHT_AMBIENT);
            this.program = program;
        }
        compileStatus = (shader) => {
            if(!GLT.gl.getShaderParameter(shader, GLT.gl.COMPILE_STATUS)) {
                console.error(GLT.gl.getShaderInfoLog(shader));
            }
        }
        use = () => {
            GLT.useProgram(this.program);
        }
        enablePosition = () => {
            GLT.enableVertexAttribArray(this.positionAttribute);
            GLT.pointToAttribute(this.positionAttribute, 3);
        }
        enableNormals = () => {
            GLT.enableVertexAttribArray(this.normalAttribute);
            GLT.pointToAttribute(this.normalAttribute, 3);
        }
```

```
    enableTransformationMatrix = (matrix) => {
        GLT.uploadMatrix4fv(this.transformationMatrix, matrix);
    }
    enableLight = (light) => {
        GLT.uploadVec3f(this.lightPosition, light.getPosition());
        GLT.uploadVec3f(this.lightColor, light.getColor());
        GLT.uploadFloat(this.lightAmbient, light.getAmbient());
    }
    enableViewProjectionMatrices = (view, projection) => {
        GLT.uploadMatrix4fv(this.viewMatrix, view);
        GLT.uploadMatrix4fv(this.projectionMatrix, projection);
    }
}
```

ДОДАТОК Д

Код компоненту відтворення

```
import GLT from '../GLTools/tools';
import Shader from '../Shaders';
export default class ModelRenderer {
    constructor(){
        this.shader = new Shader();
        this.models = {};
    }
    registerNewModel = (model, id) => {
        if(!this.models[id]) {
            this.models[id] = {
                type: model,
                instances: [],
            }
        }
    }
    addInstance = (instance, id) => {
        this.models[id].instances.push(instance);
    }
    preRender = () => {
        GLT.viewport();
        GLT.depthTest(true);
    }
    render = (camera, light) => {
        this.preRender();
        this.shader.use();
        this.shader.enableLight(light);
```

```
        camera.enable(this.shader);
        Object.keys(this.models).forEach(model => {
            this.models[model].type.use(this.shader);
            this.models[model].instances.forEach(instance => {
this.shader.enableTransformationMatrix(instance.getTransformationMatrix());
                GLT.drawTriangles(this.models[model].type.indices.length);
            })
        })
    }
}
```

ДОДАТОК Е

Коди компонентів взаємодії

Е.1 Код компоненту прослуховування миші

```
import GLT from "../GLTools/tools";
class MouseListener {
    constructor(){
        this.onWheelListeners = [];
        this.onDragListeners = []
    }
    init = () => {
        let x = 0;
        let y = 0;
        let dragging = false;
        GLT.gl.canvas.onwheel = (e) => {
            this.onWheelListeners.forEach(listener => {
                listener.onWheel(e);
            })
        }
        GLT.gl.canvas.onmousedown = (e) => {
            x = e.clientX;
            y = e.clientY;
            dragging = true;
        }
        GLT.gl.canvas.onmouseup = () => {
            dragging = false;
        }
        GLT.gl.canvas.onmousemove = (e) => {
```

```
            if (dragging) {
                const dx = x - e.clientX;
                const dy = y - e.clientY;
                x = e.clientX;
                y = e.clientY;
                this.onDragListeners.forEach(listener => {
                    listener.onDrag(dx, dy);
                });
            }
        }
    }
    subscribeToDrag = (listener) => {
        this.onDragListeners.push(listener);
    }
    subscribeToWheel = (listener) => {
        this.onWheelListeners.push(listener)
    }
}
const MouseEvent = new MouseListener();
export default MouseEvent;
```

E.2 Код компоненту камери

```
import { vec3,  mat4} from "gl-matrix";
import { toRadians } from "../Utils/maths";
import GLT from "../GLTools/tools";
import MouseEvent from "../EventHandlers/mouse";
export default class Camera {
    constructor(x = 0, y = 0, z = 3, pitch = 0, yaw = 0, roll = 0, near = 0.1, far =
```

```
1000, fov = 40){
            this.x = x;
            this.y = y;
            this.z = z;
            this.pitch = pitch;
            this.roll = roll;
            this.yaw = yaw;
            this.near = near;
            this.far = far;
            this.fov = fov;
            this.generateMatrices();
            MouseEvent.subscribeToDrag(this);
            MouseEvent.subscribeToWheel(this);
        }
        onDrag = (dx, dy) => {
            this.x += dx * 0.01;
            this.y -= dy*0.01;
            this.generateMatrices();
        }

        onWheel = (e) => {
            this.z += e.deltaY * 0.01;
            this.generateMatrices();
        }
        enable = (shader) => {
            shader.enableViewProjectionMatrices(this.viewMatrix,
this.projectionMatrix);
        }
        generateMatrices = () => {
            this.viewMatrix = this.createViewMatrix();
```

```
            this.projectionMatrix = this.createProjectionMatrix();
        }
        createViewMatrix = () => {
            const matrix = [];
            mat4.identity(matrix);
            mat4.rotateX(matrix, matrix, toRadians(this.pitch));
            mat4.rotateY(matrix, matrix, toRadians(this.yaw));
            mat4.rotateZ(matrix, matrix, toRadians(this.roll));
            mat4.translate(matrix, matrix, vec3.fromValues(-this.x, -this.y, -this.z));
            return matrix;
        }
        createProjectionMatrix = () => {
            const aspectRatio = GLT.gl.canvas.width / GLT.gl.canvas.height;
            const matrix = [];
            mat4.perspective(matrix, toRadians(this.fov), aspectRatio, this.near,
this.far);
            return matrix;
        }
    }
```

ДОДАТОК Ж

Код HTML-сторінки

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import logo from './logo.svg';
import reportWebVitals from './reportWebVitals';
import WebGL from './WebGLComponent';
import RotationSliders from './AdditionalComponents/RotationSliders';
const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <div className='container'>
      <div className='headerWrap'>
        <img src={logo} className='img' alt='logo'/>
        <h1>WebGL Numerical Data App</h1>
      </div>
      <div className="wrap">
        <WebGL></WebGL>
        <div className="blockWrap">
          <RotationSliders></RotationSliders>
          <label htmlFor="fileInput" className='inputFile'>
            <input type="file" id="fileInput" accept=".json"/>
            Choose file
          </label>
        </div>
      </div>
    </div>
```

```
  </React.StrictMode>
);
reportWebVitals();
```