УДК 681.3
DOI https://doi.org/10.32782/2521-6643-2025-2-70.21

Фірсов О. Д., кандидат фізико-математичних наук, доцент, доцент кафедри комп'ютерних наук та інженерії програмного забезпечення Університету митної справи та фінансів
ORCID: 0000-0002-6528-64

ВІЗУАЛІЗАЦІЯ 3D МЕТАДОКСА ЗАСОБАМИ THREE.JS

У роботі розроблено алгоритм та відповідний додаток для візуалізації спеціалізованого 3D обєкта із застосуванням можливостей спеціалізованої кросбраузерної бібліотеки Three.js. Об'єктом дослідження виступає процес створення реалістичних моделей метадоксів із використанням теоритичних даних, які беруть за основу моделі трьох-вершинних метадоксів. У ході роботи було проаналізовано варіанти візуалізації 3D модклей метадокса з ідеєю доступністі для непрофесійного користувача, якій є спеціалістом у своєй предметній області, але не володіє на достатньому рівні технологіями програмування чи не володіє навичками експлуатації спеціалізованого програмного забезпечення для візуалізації. Результати дослідження показують, що JavaScript з розширенням у вигляді бібліотеки Three.js є потужним інструментом для 3D-візуалізації, який завдяки своїй гнучкості та можливостям підтримки браузерами дозволяє створювати ефективні та динамічнчні моделі метадоксів, що дозволяє краще зрозуміти залежності між лінгвістичними обєктами та передати розроблені конструкти зацікавленим особам.

Запропонованне рішення може бути використане для розробки навчальних інтерактивних додатків та, моделей що потребують інтеграції реальних даних довільного походження. Виявлені підходи до організації процесів обробки та візуалізації даних для метадоксів можуть знайти застосування у розробці віртуальних метадоксів та фрактальних обєктів, що іми породжуються, з урахуванням подуктивності і реалістичність представлення. Отримані результати можуть слугувати основою для побудови більш складних моделей та розширенні функціональних можливостей відповідних додатків. Аде тут виникає питання балансу між складністю рішення та доступністю для аналітиків, що працюють у конкретній предметній області.

Окремою частиною дослідження стало застосування Grok для тестування та відладки додатку. У цієй роботі штучний інтелект продемонстрував можливості аналізу виявлених розробником помилок у ручному режимі, та саме цікаве, переробка архітектури у випадку, еоли виявлені помилки залаежать від зовнішніх факторів.

Ключові слова: *3D-моделювання, метадокс, просторові дані, Three.js, візуалізація, інтеграція даних, JavaScript.*

Firsov O. D. Visualization of 3D metadoxes using Three.js

The work developed an algorithm and a corresponding application for visualization of a specialized 3D object using the capabilities of the specialized cross-browser library Three.js. The object of the study is the process of creating realistic models of metadocs using theoretical data, which are based on the model of three-vertex metadocs. In the course of the work, options for visualization of 3D modkley metadocs were analyzed with the idea of accessibility for a non-professional user who is a specialist in his subject area, but does not have a sufficient level of programming technologies or does not have the skills to operate specialized software for visualization. The results of the study show that JavaScript with an extension in the form of the Three.js library is a powerful tool for 3D visualization, which, due to its flexibility and browser support capabilities, allows you to create effective and dynamic models of metadocs, which allows you to better understand the dependencies between linguistic objects and transfer the developed constructs to interested parties.

The proposed solution can be used to develop educational interactive applications and models that require the integration of real data of arbitrary origin. The identified approaches to organizing the processes of processing and visualization of data for metadocs can be used in the development of virtual metadocs and fractal objects generated by them, taking into account the productivity and realism of the representation. The results obtained can serve as the basis for building more complex models and expanding the functionality of the corresponding applications. Here the question arises of the balance between the complexity of the solution and its accessibility for analysts working in a specific subject area.

A separate part of the research was the use of Grok for testing and debugging the application. In this work, artificial intelligence demonstrated the ability to analyze errors detected by the developer in manual mode, and most interestingly, the reworking of the architecture in the case when the detected errors are caused by external factors.

Key words: *3D modeling, metadocs, spatial data, Three.js, visualization, data integration, JavaScript.*

Постановка задачі. Метадокс – досить нішева тема, пов'язана з філософією, тріалектикою та системним мисленням, тому не часто висвітлюється в літературі.

Класичний метадокс – це термін, запропонований дослідницькою групою RC39 як еволюція концепції парадоксу. Якщо парадокс – це бінарне протиріччя (дві протилежності, наприклад, «так» або «ні»),

то метадокс – це більш складна структура, зазвичай трикутна, де протиріччя утворюють взаємопов'язану систему. Вона виходить за рамки дуалізму і вводить «третій елемент», створюючи основу для тріалетики (логіка з основою «три» замість «двох»).

Метадокс дозволяє працювати зі складними об'єктами, де простого «так чи ні» недостатньо, а джерелом розвитку стають протиріччя.

У роботах С. Переслегіна і В. Громова метадокс часто представляється як «базова вага» – трикутник, де кожна вершина є бінарним протиріччям двом іншим, а разом вони утворюють стійку, але динамічну систему. Василь Громов розробив поняття «рівновага» як центральний метадокс.

Як випливає з опису, графічне представлення метадоксу являє собою трикутник з трьома вершинами. Завдання візуалізації такої структури досить проста.

Наступною стійкою структурою після трикутника метадокса є піраміда, яка також забезпечує рівновагу. В геометрії тетраедр – це мінімальна тривимірна структура, яка має максимальну стійкість, так як всі його вершини пов'язані між собою. У контексті метадоксу це означає перехід від плоскої тріалектики до об'ємної системи з новим рівнем складності. Подальше ускладнення структури призводить до появи структури у вигляді біпіраміди в тривимірному просторі або гіперпіраміди в чотиривимірному просторі.

Аналіз останніх досліджень та публікацій. Останні роки ознаменувалися якісним зростанням використання 3D-технологій у різних галузях науки й техніки, зокрема в хімії, комп’ютерному моделюванні, геодезії, економіці та суміжних сферах. Така динаміка зумовлена стрімким розвитком комп’ютерних обчислень, збільшенням потужностей обробки інформації та доступністю спеціалізованого програмного забезпечення для тривимірної візуалізації й аналізу [1].

Одним із ключових напрямів сучасних досліджень є інтеграція структурованих і неструктурованих даних у 3D-середовища, що дозволяє створювати реалістичні цифрові моделі об’єктів і процесів. Ці моделі відіграють критичну роль у прогнозуванні, візуалізації та симуляції – як у фундаментальних дослідженнях, так і в прикладному проєктуванні.

У хімії тривимірне моделювання дає змогу аналізувати молекулярну структуру, передбачати реакції та вивчати властивості речовин на мікроскопічному рівні. Використання програмного забезпечення, такого як Avogadro, Gaussian чи Chem3D, є стандартом у сучасній хімічній інформатиці [2].

У геодезії та картографії 3D-технології застосовуються для створення цифрових моделей рельєфу та урбаністичних об’єктів. Технології лазерного сканування, фотограмметрії та обробки супутникових знімків забезпечують високоточні геопросторові моделі [3]. Це відкриває нові можливості для планування територій, оцінки ризиків і моніторингу навколишнього середовища [4].

У галузі економіки та містобудування тривимірні моделі використовуються для просторового аналізу ринку, симуляції транспортних потоків, оптимізації логістичних мереж і моделювання сценаріїв розвитку територій. Інструменти, такі як ArcGIS Urban, CityEngine чи SketchUp, дають змогу створювати інтерактивні 3D-моделі з урахуванням соціально-економічних параметрів [5].

Таким чином, 3D-технології стають універсальним інструментом для міждисциплінарного дослідження, який значно підвищує точність аналізу, якість візуалізації та ефективність прийняття рішень.

Мета статті. Існує безліч інструментів для візуалізації таких структур, як 3D-об'єкти. Але для обивателя цікавіше отримати рішення, за допомогою якого можна легко уявити об'єкт дослідження, передати зображення по мережі і використовувати найбільш поширену технологію. Відповідно, актуально отримати рендерений об'єкт з предметної області за допомогою JavaScript. Друге питання – як розробити додаток. А саме, використання Grok як рекомендаційної системи для того, щоб отримати необхідне рішення.

Виклад основного матеріалу. Високорівневі вимоги до продукту сформулюємо наступним чином.

Уявімо метадоксальну біпіраміду мови за допомогою Three.js. Це бібліотека JavaScript для створення 3D-графіки в браузері, не потрібно встановлювати Python або Matplotlib – все буде працювати безпосередньо в веб-браузері (наприклад, Chrome або Edge). Код Three.js буде відображати тригональну біпіраміду з п'ятьма вершинами (Знак, Значення, Структура, Контекст, Час) і ребрами. Ви можете запустити цей код у Windows 11 без додаткових інструментів. Для його реалізації потрібен будь-який текстовий редактор (Notepad, VS Code), веб-браузер (Chrome, Firefox, Edge – будь-який сучасний), підключення до інтернету (Three.js можна завантажити через CDN).

У браузері повинна відображатися 3D-сцена з бипірамідою, що складається з ліній (ребер), які з’єднують вершини. Для кожної вершини надаються окремі текстові мітки ("Знак", "Значення", "Структура", "Контекст", "Час"") зі зміщенням від вершин.

Сцена обертається навколо осі Y зі швидкістю 0.01 радіан за кадр.

Камера налаштована для оптимального огляду бипіраміди.

Сцена адаптується до зміни розміру вікна браузера.

Можливе включення управління мишею (наприклад, OrbitControls), інакшк бипіраміда обертається автоматично.

Використовується Three.js з урахуванням можливостей підключення різних версій бібліотеки.

Шрифти завантажуються з threejs.org.

Алгоритм створення 3D-візуалізації.

Крок 1: Підключити бібліотеку Three.js через CDN.
Крок 2: Ініціалізація сцени, камери та рендерера
Створити нову 3D-сцену (THREE. Scene).
Створити перспективну камеру (THREE. PerspectiveCamera) із параметрами.
Створити WebGL-рендерер (THREE. WebGLRenderer).
Встановити розмір рендерера: ширина та висота вікна браузера.
Крок 3: Визначення вершин бипіраміди
Створити масив vertices, що містить координати вершин у 3D-просторі (x, y, z).
Крок 4: Визначення ребер бипіраміди. Створити масив edges, що містить пари індексів вершин для ребер.
Крок 5: Налаштування матеріалів
Створити матеріал для ліній (THREE. LineBasicMaterial).
Створити матеріал для вершин (THREE. MeshBasicMaterial).
Крок 6: Додавання ребер до сцени
Для кожного ребра в масиві edges:
Створити масив points, що містить дві 3D-точки (THREE. Vector3) за координатами вершин ребра (з масиву vertices).
Створити геометрію лінії (THREE. BufferGeometry) на основі масиву points.
Створити об’єкт лінії (THREE. Line) із геометрією та матеріалом ліній.
Додати лінію до сцени (scene.add(line)).
Крок 7: Додавання вершин до сцени
Створити геометрію сфери (THREE. SphereGeometry) із параметрами.
Для кожної вершини в масиві vertices:
Створити меш (THREE. Mesh) із геометрією сфери та матеріалом вершин.
Встановити позицію меша за координатами вершини (vertex[0], vertex[1], vertex[2]).
Додати меш до сцени (scene.add(sphere)).
Крок 8: Додавання текстових міток до вершин
Створити масив labels із назвами вершин.
Ініціалізувати завантажувач шрифтів (THREE. FontLoader).
Завантажити шрифт.
Після завантаження шрифту (font):
Для кожної вершини (vertex) та відповідної мітки (labels[i]):
Створити текстову геометрію (THREE. TextGeometry) із параметрами.
Створити матеріал для тексту (THREE. MeshBasicMaterial).
Створити меш (THREE. Mesh) із текстовою геометрією та матеріалом.
Встановити позицію тексту зі зміщенням.
Додати текст до сцени (scene.add(text)).
Крок 9: Налаштування камери
Встановити позицію камери.
Налаштувати напрямок камери на точку за допомогою camera.lookAt.
Крок 10: Створення анімації
Створити функцію animate:
Запланувати наступний кадр анімації через requestAnimationFrame(animate).
Обертати сцену навколо осі Y на 0.01 радіан за кадр (scene.rotation.y += 0.01).
Викликати рендеринг сцени з поточною камерою.
Викликати animate() для запуску анімації.
Крок 11: Обробка зміни розміру вікна
Додати слухач події resize на об’єкт window.
При зміні розміру вікна:
Оновити співвідношення сторін камери.
Оновити матрицю проєкції камери.
Оновити розмір рендерера.

Доопрацюємо візуалізацію біпіраміди метадоксу на Three.js! Додамо керування мишею (обертання та масштабування) за допомогою OrbitControls.
Використання миші:
Обертається ліва кнопка.

Колісний зум.
Права кнопка – панорамування.

Розроблений код, відповідно до алгоритму виглядає наступним чином.

```
html
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Метадокс </title>
    <style>
        body { margin: 0; overflow: hidden; }
        canvas { width: 100%; height: 100%; }
    </style>
</head>
<body>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/three.js/r128/three.min.js"></
script>
    <script src="https://threejs.org/examples/js/controls/OrbitControls.js"></script>
    <script>
        // Сцена, камера, рендерер
        const scene = new THREE.Scene();
        const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.
innerHeight, 0.1, 1000);
        const renderer = new THREE.WebGLRenderer();
        renderer.setSize(window.innerWidth, window.innerHeight);
        document.body.appendChild(renderer.domElement);
        // Управління мишею
        const controls = new THREE.OrbitControls(camera, renderer.domElement);
        controls.enableDamping = true; Плавний рух
        controls.dampingFactor = 0,05;
        controls.screenSpacePanning = false;
        controls.minDistance = 1; Мінімальне збільшення
        controls.maxDistance = 10; Максимальне збільшення
        Координати вершин біпіраміди
        const вершини = [[0, 0, 0],[1,0,0],[0.5, 0.87, 0],[0.5,0.43,1],[0.5, 0.43, -1];
           const metadoxes = [...];
        Рендеринг Metadox
        metadoxes.forEach(metadox => {
            const material = new THREE.LineBasicMaterial({ color: metadox.color });
            metadox.edges.forEach(edge => {
                const points = [];
                points.push(new THREE.Vector3(vertices[edge[0]][0], vertices[edge[0]]
[1], vertices[edge[0]][2]));
                points.push(new THREE.Vector3(vertices[edge[1]][0], vertices[edge[1]]
[1], vertices[edge[1]][2]));
                const geometry = new THREE.BufferGeometry().setFromPoints(points);
                const line = new THREE.Line(geometry, material);
                scene.add(line);
            });
        });
        // Вершини (точки)
        const pointMaterial = new THREE.MeshBasicMaterial({ color: 0xffff }); // Білі
точки
        const sphereGeometry = new THREE.SphereGeometry(0.05, 32, 32);
        vertices.forEach(vertex => {
            const sphere = new THREE.Mesh(sphereGeometry, pointMaterial);
            sphere.position.set(vertex[0], vertex[1], vertex[2]);
            scene.add(sphere);
        });
```

```
        // Мітки вершин
        const labels = ['Знак', 'Значення', 'Структура', 'Контекст', 'Час'];
        const loader = new THREE.FontLoader();
        loader.load('https://threejs.org/examples/fonts/helvetiker_regular.typeface.
json', function (font) {
            vertices.forEach((vertex, i) => {
                const textGeometry = new THREE.TextGeometry(labels[i], {
                    font: font,
                    size: 0.1,
                    height: 0.01
                });
                const textMaterial = new THREE.MeshBasicMaterial({ color: 0x000000 });
// Чорні відмітини
                const text = new THREE.Mesh(textGeometry, textMaterial);
                text.position.set(vertex[0] + 0.1, vertex[1] + 0.1, vertex[2]);
                scene.add(text);
            });
        });
        // Положення камери
        camera.position.set(1, 1, 2);
        camera.lookAt(0,5, 0,43, 0);
        Анімації
        function animate() {
            requestAnimationFrame(анімувати);
            controls.update(); Оновлення системи керування мишею
            renderer.render(сцена, камера);
        }
        animate();
        Адаптація розміру вікна
        window.addEventListener('resize', () => {
            camera.aspect = window.innerWidth / window.innerHeight;
            camera.updateProjectionMatrix();
            renderer.setSize(window.innerWidth, window.innerHeight);
        });
    </script>
</body>
</html>
```

Тестування та доробка. Запропонований код, не дивлячись на логічну послідовність та обгунтованість, хоча і виконався, але біпіраміду не відобразив. Отже, виникла потреба у тестуванні додатув та доробці. У звязку із тим, що для генерації зображення виконується багато різних компонентів коду, які вирішують окремі задачі, було прийнято рішення застосувати Grok, для аналізу помилок та їх виправлення.

Grok, створений xAI, – це ШІ-помічник, який ефективно допомагає у тестуванні JavaScript-коду, оптимізуючи процес розробки та підвищуючи якість програмного забезпечення. Завдяки здатності аналізувати код, генерувати тести та виявляти помилки, Grok стає цінним інструментом для розробників.

Grok може аналізувати на всіх етапах тестування. На початковому рівні він перевіряє синтаксис, знаходить потенційні помилки, такі як невизначені змінні чи неправильні логічні конструкції, і пропонує виправлення. Наприклад, для функції, яка обробляє масиви, Grok може виявити відсутність обробки граничних випадків і порадити додати відповідні перевірки. Це економить час і знижує ймовірність багів.

Для модульного тестування Grok генерує тестові випадки, використовуючи популярні фреймворки, такі як Jest або Mocha. На основі аналізу функції він створює тести для перевірки основних сценаріїв, крайових випадків і виняткових ситуацій. Наприклад, для функції, що обробляє голосування в системі виборів, Grok може згенерувати тести для перевірки коректності підрахунку голосів або реакції на невалідні дані.

Grok також допомагає в інтеграційному тестуванні, аналізуючи взаємодію між модулями. Він може перевірити, чи правильно сервер Node.js обробляє запити, і запропонувати тести для API-ендпоінтів. Крім того, Grok підтримує автоматизацію тестів, створюючи скрипти для CI/CD-систем, таких як GitHub Actions.

Переваги використання Grok включають швидкість, точність і підтримку складних сценаріїв, як-от асинхронний код чи 3D-візуалізації з Three.js. Однак він не замінює людську інтуїцію, а доповнює її, допомагаючи зосередитися на логіці, а не рутинних перевірках.

Використання Grok для тестування JavaScript-коду – це крок до ефективної розробки, що забезпечує надійність і масштабованість проєктів, від веб-сторінок до систем голосування.

У процесі виконання коду виникла помилка:

Uncaught TypeError: THREE. OrbitControls is not a constructor

Возможные причины, предложенные Grok:

– Скрипт OrbitControls.js не загрузился (сеть, блокировка, ошибка 404).
– Версия Three.js (r134) не полностью совместима с этим файлом OrbitControls.js.
– Порядок загрузки скриптов нарушен, и OrbitControls вызывается до полной инициализации Three.js.

Оскільки r134 може бути нестабільним через CDN, давайте перепишемо код, використовуючи останню версію Three.js та модульний підхід через <script type="module">. Це забезпечить сумісність і належне завантаження OrbitControls.

Наступним кроком є виправлення неспійманої помилки ReferenceError: THREE не визначено. Grok вказує, що об'єкт THREE недоступний на момент виконання коду, хоча він підключений Three.js за допомогою <script src="https://cdn.jsdelivr.net/npm/three@0.165.0/build/three.min.js"></script>. Це може означати, що Three.js скрипт або не завантажився (проблема з мережею, блокування), або код виконується до того, як Three.js буде повністю ініціалізований.

Неявна помилка виконання у вигляді білого екрану при відкритті файлу проекту в браузері означає, що Three.js запустився, але щось пішло не так з рендерингом сцени: або скрипти завантажилися не повністю, або є баг в коді, який не відображається в консолі, або сцена порожня/не видима камері. Так як екран білий, а не чорний, швидше за все, рендерер працює, але сцена або порожня, або не видно.

Виконаємо етапи налагодження.

Додамо код для тесту:

```
const testCube = new THREE.Mesh(
    new THREE.BoxGeometry(0.5, 0.5, 0.5),
    new THREE.MeshBasicMaterial({ color: 0xff0000 })
);
testCube.position.set(0.5, 0.43, 0);
scene.add(testCube);
console.log(«Додано тестовий куб»);
```

Якщо з'являється червоний куб, то рендеринг працює, але біпіраміда не відображається.

Спрощення роботи камери та органів управління для усунення впливу OrbitControls. Куб працює, тому Three.js працює локально. Тепер давайте додамо повний код з біпірамідою і OrbitControls.

Піраміда працює шляхом додавання міток вершин («Знак», «Значення», «Структура», «Контекст», «Час») за допомогою FontLoader.

Мітки вершин відображаються у вигляді чорних знаків питання на чорному тлі, що вказує на проблему із завантаженням або рендерингом шрифту. Швидше за все, шрифт або helvetiker_regular.typeface.json завантажився некоректно, або Three.js версії 0.124.0 не змогла його обробити, і замість тексту обробляються "заповнювачі" (знаки питання).

Виправлений код представлений нижче.

```
html
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <title>Метадокс мови як біпіраміди</title>
    <style>
        body { margin: 0; overflow: hidden; background: #000; }
        canvas { width: 100%; height: 100%; }
    </style>
</head>
<body>
    <script src="three.js"></script>
    <script src="OrbitControls.js"></script>
    <script>
        console.log("Сценарій розпочато виконання");

        if (typeof THREE === 'undefined') {
            console.error('Three.js не завантажився');
            alert('Three.js не загрузился');
        } else if (typeof THREE.OrbitControls === 'undefined') {
```

```
            console.error('OrbitControls не завантажувався');
            alert('OrbitControls не завантажився');
        } else {
            console.log('Three.js и OrbitControls завантажено, версія:', THREE.
REVISION);

            const scene = new THREE.Scene();
            const camera = new THREE.PerspectiveCamera(75, window.innerWidth / window.
innerHeight, 0.1, 1000);
            const renderer = new THREE.WebGLRenderer();
            renderer.setSize(window.innerWidth, window.innerHeight);
            document.body.appendChild(renderer.domElement);

            const controls = new THREE.OrbitControls(camera, renderer.domElement);
            controls.enableDamping = true;
            controls.dampingFactor = 0.05;
            controls.minDistance = 1;
            controls.maxDistance = 10;

            const vertices = [
                [0, 0, 0],      // Знак (0)
                [1, 0, 0], // Значення (1)
                [0.5, 0.87, 0], // Структура (2)
                [0.5, 0.43, 1], // Контекст (3)
                [0.5, 0.43, -1] // Час (4)       ];

            const metadoxes = [
                { edges: [[0, 1], [1, 2], [2, 0]], color: 0xff00ff },
                { edges: [[0, 3], [0, 4], [3, 4]], color: 0x00ff00 },
                { edges: [[1, 3], [1, 4], [3, 4]], color: 0xffff00 },
                { edges: [[2, 3], [2, 4], [3, 4]], color: 0x00ffff },
                { edges: [[0, 1], [0, 3], [1, 3]], color: 0xff0000 },
                { edges: [[0, 1], [0, 4], [1, 4]], color: 0x0000ff },
                { edges: [[0, 2], [0, 3], [2, 3]], color: 0xff8000 },
                { edges: [[0, 2], [0, 4], [2, 4]], color: 0x800080 },
                { edges: [[1, 2], [1, 3], [2, 3]], color: 0x808000 },
                { edges: [[1, 2], [1, 4], [2, 4]], color: 0x008080 } ];

            metadoxes.forEach(metadox => {
                const material = new THREE.LineBasicMaterial({ color: metadox.color });
                metadox.edges.forEach(edge => {
                    const points = [];
                    points.push(new THREE.Vector3(vertices[edge[0]][0],
vertices[edge[0]][1], vertices[edge[0]][2]));
                    points.push(new THREE.Vector3(vertices[edge[1]][0],
vertices[edge[1]][1], vertices[edge[1]][2]));
                    const geometry = new THREE.BufferGeometry().setFromPoints(points);
                    const line = new THREE.Line(geometry, material);
                    scene.add(line);         });             });

            const pointMaterial = new THREE.MeshBasicMaterl({color:0xffff });
            const sphereGeometry = new THREE.SphereGeometry(0.05, 32, 32);
            vertices.forEach(vertex => {
                const sphere = new THREE.Mesh(sphereGeometry, pointMaterial);
                sphere.position.set(vertex[0], vertex[1], vertex[2]);
                scene.add(sphere);             });

            Мітки вершин
            const labels = ['Знак', 'Значення', 'Структура', 'Контекст', 'Час'];
            const loader = new THREE.FontLoader();
            loader.load('helvetiker_regular.typeface.json', function (font) {
                console.log("Шрифт завантажено");
```

```
                vertices.forEach((vertex, i) => {
                    const textGeometry = new THREE.TextGeometry(labels[i], {
                        font: font,   size: 0.1,   height: 0.01            });
                    const textMaterial = new THREE.MeshBasicMaterial({ color: 0xffff });
// Белый текст
                    const text = new THREE.Mesh(textGeometry, textMaterial);
                    text.position.set(vertex[0] + 0.15, vertex[1] + 0.15, vertex[2]);
                    scene.add(text);
                });
                console.log('Додано мітки вершин');
            }, undefined, function (error) {
                console.error('Помилка завантаження шрифту:', error);
            });

            camera.position.set(1, 1, 2);
            camera.lookAt(0, 0, 0);

            function animate() {
                requestAnimationFrame(animate);
                controls.update();
                renderer.render(scene, camera);
            }
            animate();

            window.addEventListener('resize', () => {
                camera.aspect = window.innerWidth / window.innerHeight;
                camera.updateProjectionMatrix();
                renderer.setSize(window.innerWidth, window.innerHeight);
            });
        }
    </script>
</body>
</html>
```

Результат виконання коду:

Чорний фон.

Біпіраміда з 10 різнокольоровими метадоксами (лініями).

Білі точки (вершини).

Білі мітки поруч з вершинами: Знак, Значення, Структура, Контекст, Час.

Органи управління: обертання (ліва кнопка), зум (коліщатко).

Висновки. Результати роботи продемонстрували, що мова JavaScript з розширенням у вигляді бібліотеки Three.js є ефективним інструментом для реалізації 3D-моделей метадоксів та більш складних структур. Використання відкритої бібліотеки, дозволило створити високоякісні та реалістичні моделі, що можуть бути легко інтегровани у веб-додатки, та надати можливість дослідникам, що використовують метадокси сконцентруватися на своїх безпосередніх дослідженнях.

У процесі роботи розроблено алгоритм побудови графічної структури, що може змінювати масштаб та положення у просторі, який є універсальним та досить легко розширюваним.

Для розвязання технічних проблем, що повязані з тестуванням та модифікацією кода був задіян Grok. У ітераційному процесі за допомогою рекомендацій ШІ було виправлено помилки та нестиковки реалізації. Тобто підтверджено ефективність використання Grok для роботи з JavaScript.

Список використаних джерел:

1. Li Z., Zhu Q., Gold C. *Digital Terrain Modeling: Principles and Methodology*. CRC Press, 2005.
2. Chen B., Guo Y., Zhang X. Visualization of Molecular Structures with 3D Tools. *Journal of Chemical Information and Modeling*. 2021. Vol. 61, No. 3. P. 945–952.
3. Li R., Chapman M.A., Zhang J. LIDAR Technology for Urban Modeling. *Remote Sensing*. 2020. Vol. 12, No. 7. P. 1125.
4. Захарченко М. В., Печорін С. В. Геоінформаційні технології у геодезії: методи та застосування. *Вісник геодезії та картографії*. 2022. Т. 8, № 2. С. 24–29.
5. Batty M. et al. Smart Cities of the Future. *Environment and Planning B: Planning and Design*. 2012. Vol. 39, No. 4. P. 625–636.

References:

1. Li, Z., Zhu, Q., Gold, C. (2005). Digital Terrain Modeling: Principles and Methodology. CRC Press.
2. Chen, B., Guo, Y., Zhang, X. (2021). Visualization of Molecular Structures with 3D Tools. *Journal of Chemical Information and Modeling*. Vol. 61, No. 3. P. 945–952.
3. Li, R., Chapman, M. A., Zhang, J. (2020). LIDAR Technology for Urban Modeling. *Remote Sensing*. Vol. 12, No. 7. P. 1125.
4. Zaharchenko, M. V., Petcherin, S. V. (2022). Geoinformacijni tehnologiyi u geodeziyi: metodi ta zastosuvannya. *Visnik geodeziyi ta kartografiyi*. T. 8, № 2. P. 24–29.
5. Batty, M. et al. (2012). Smart Cities of the Future. *Environment and Planning B: Planning and Design*. Vol. 39, No. 4. P. 625–636.

Дата надходження статті: 21.10.2025
Дата прийняття статті: 10.11.2025
Опубліковано: 30.12.2025