Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет Комп 'ютерні технології і системи
Кафедра Комп'ютерні інформаційні технології

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему: «Дослідження моделей оптичних перетворень негладких фрактальних поверхонь»
за освітньою програмою **12 Інженерія програмного забезпечення**
зі спеціальності: **121 Інженерія програмного забезпечення**

Виконав: студент групи ПЗ2222: / Олександр ЗАЙЦЕВ /

Керівник: / Віктор ШИНКАРЕНКО /

Нормоконтролер: / Світлана ВОЛКОВА /

Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

Дніпро – 2024 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty Computer technologies and systems
Department Computer information technology

Explanatory Note

to Master's Thesis

on the topic: «Study of models of optical transformations of non-smooth fractal surfaces»

according to educational curriculum **12 software engineering**
in the Speciality: **121 software engineering**

Done by the student of the group PZ2222: / Oleksandr ZAITSEV /

Scientific Supervisor: / Viktor SHYNKARENKO /

Normative controller: / Svitlana VOLKOVA /

Dnipro – 2024

Міністерство освіти і науки України
Український державний університет науки і технологій

Факультет: Комп'ютерних технологій і систем
Кафедра: Комп’ютерні інформаційні технології
Рівень вищої освіти: магістр
Освітня програма: Інженерія програмного забезпечення
Спеціальність: Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ____КІТ
_____ Вадим ГОРЯЧКІН
_____ грудня 2023 р.

ЗАВДАННЯ

На кваліфікаційну роботу ____Магістр__
студенту ЗАЙЦЕВУ ОЛЕКСАНДРУ ВАЛЕРІЙОВИЧУ

1. Тема дипломної роботи: Дослідження моделей оптичних перетворень негладких фрактальних поверхонь.
   Керівник роботи:
   затверджені наказом 1196 ст від 05.12. 2022 року

2. Строк подання студентом роботи 04.12. 2023 року

3. Вихідні дані до дипломної роботи:
   _____.

4. Зміст пояснювальної записки (перелік питань до розробки):
   4.1. Аналітична частина: _____;
   4.2. Основна частина: _____;

5. Перелік демонстраційного матеріалу:
   5.1. доповідь;
   5.2. презентація;
   5.3. демонстраційне відео.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Приміт ка
1	Вступ	03.03.23 – 14.04.23	
2	Аналіз сучасного стану дослідження проблеми за науковими літературними джерелами	15.04.23 – 22.08.23	10%
3	Аналіз сучасного стану програмно-апаратного забезпечення, яке потребує вдосконалення для вирішення проблем дослідження	23.08.23 – 18.10.23	
4	Постановка задачі, технічне завдання	19.10.23 – 20.10.23	30%
5	Розробка інструментальних засобів дослідження	08.11.23 – 13.11.23	
6	Виконання досліджень	14.11.23 – 18.11.23	60%
7	Оформлення тез доповідей	19.11.23 – 23.11.23	
8	Оформлення статті у фаховий журнал	24.11.23 – 28.11.23	
9	Оформлення пояснювальної записки	29.11.23 – 05.12.23	
10	Розробка демонстраційних матеріалів	10.01.24 – 12.01.24	100%
11	Подання кваліфікаційної роботи до кафедри	10.01.24	
12	Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії	25.01.24	

Студент _____ / Олександр ЗАЙЦЕВ /

Керівник роботи _____/ Віктор ШИНКАРЕНКО /

РЕФЕРАТ

Пояснювальна записка складається з 7 розділів:

– вступ – в даному розділі описується сутність розробки, її актуальність. Складається з 4 сторінок;

– огляд предметної галузі – у цьому розділі описується базова теорія фракталів та проводиться огляд движків для рендерінгу тривимірної графіки у WEBі, розглядаються переваги і не доліки низки найпопулярніших технологій. Складається з 25 сторінок;

– розробка інструментальних засобів для генерації та рендерінгу фракталів, збору показників прозорості зображення та автоматизації даних процесів – у цьому розділі розробляюється низка програмних комплексів, які допомагають будувати тривимірні сцени з фракталом та джерелом освітлення, збирати, зберігати та аналізувати показники прозорості, описується дизайн та архітектура проектів та їх взаємодія між собою. Складається з 39 сторінок;

– дослідження впливу фігури фракталу, ступеню його розвитку, прозорості та повороту на отриману тінь – у цьому розділі виконується автоматизація процесу збору показників прозорості, необхідних для проведення досліджень та проводиться аналіз метрик кількості пікселів відповідно до кольорів. Складається з 27 сторінок;

– загальні висновки – підсумки всієї роботи. Складається з двох сторінок;

– список літератури – включає в себе бібліографічний список використаної літератури. Складається з трьох сторінок;

– додатки – містять робочий проект без папки менеджеру пакетів «node_modules» та технічні вимоги до програми.

Кількість таблиць: 5.

Кількість рисунків: 83.

Ключові слова: фрактали, тривимірні фрактали, прозорість фрактальних фігур, показники прозорості фракталу, комп’ютерна графіка, графічні рушії, графіка у web, тривимірна графіка у браузері.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

Стек технологій (technology stack) - сукупність програмних інструментів, фреймворків, мов програмування, бібліотек та інших технічних компонентів, які використовуються для розробки та виконання програмного забезпечення або веб-додатків. Стек технологій є ключовим поняттям у сфері програмування та веб-розробки.

Веб-застосунок (веб-додаток) - це програмне забезпечення, яке взаємодіє з користувачем через веб-браузер на веб-сайті. Зазвичай веб-застосунки доступні через Інтернет і не вимагають встановлення на пристрої користувачів, оскільки вони виконуються у веб-середовищі.

DOM (Об'єктна модель документа) - це представлення об'єктів даних, що визначають структуру та вміст документа в Інтернеті;

Virtual DOM (Віртуальний DOM або VDOM) - це стратегія та набір бібліотек/алгоритмів, які дозволяють оптимізувати продуктивність на клієнтській стороні, уникаючи прямої маніпуляції з DOM. Це досягається за допомогою створення та роботи з абстракцією, що імітує DOM-дерево;

HTML (Мова гіпертекстової розмітки) - це основний компонент будівництва Інтернету, який визначає значення та структуру веб-контенту. Інші технології, такі як CSS для зовнішнього вигляду та JavaScript для функціональності, використовуються додатково;

CSS (Каскадна таблиця стилів) - це система стилів, що дозволяє форматувати вміст сторінки відповідно до визначених правил, включаючи стилі тексту, розташування блоків та анімацію;

Фреймворк - це інструмент, який надає готові компоненти або рішення, спрямовані на полегшення розробки. Він може включати бібліотеку, але його структура визначається принципом інверсії контролю (IoC), де фреймворк викликає код користувача за необхідності;

Бібліотеки JavaScript - це набори наперед написаних фрагментів коду, які можна використовувати для виконання звичайних функцій JavaScript. Цей код

можна підключити до проекту за потребою. Навідміну від фреймворку, бібліотека це не сукупність рішень для вирішення низки задач в певній області, а окреме, зазвичай невеличке рішення, для вирішення однієї конкретної задачі.

Транспілятор коду / транспайлер - це програма або інструмент, який перетворює вихідний код однієї мови програмування в еквівалентний код іншої мови програмування, зазвичай зберігаючи при цьому семантику програми. Такий процес називається транспіляцією. Однією з часто використовуваних транспіляцій є перетворення коду, написаного на високорівневій мові програмування, в код на мові, яка має більш низький рівень абстракції або призначена для виконання в іншому середовищі. Наприклад, транспілятор TypeScript може перетворювати код, написаний на TypeScript в JavaScript, або транспілятор JavaScript може перетворювати код, написаний на сучасній версії JavaScript, в код, сумісний із старішими версіями цієї мови або код для виконання в інших середовищах, таких як середовище виконання Node.js або веб-браузер.

Бандл - цей термін використовується для позначення об'єднаного файлу або групи файлів, які створюються модульним бандлером, таким як Webpack або Parcel. Основна мета створення бандлів - це зменшення кількості та розміру файлів, які потрібно завантажити веб-додатком при його запуску.

Рантайм - в контексті Node.js вказує на середовище виконання, тобто програмне забезпечення, яке виконує викладений JavaScript-код. В іншому розумінні, рантайм представляє собою виконавчу частину Node.js, яка взаємодіє з операційною системою та виконує JavaScript-застосунки.

GLSL (*OpenGL Shading Language*) – мова високого рівня для програмування шейдерів.

OpenGL (Open Graphics Library) - це специфікація та набір програмних інтерфейсів (API) для реалізації 2D та 3D графіки. Вона надає стандартні функції для взаємодії програмного забезпечення з графічним апаратним

забезпеченням комп'ютера.

WebGL — це стандарт на базі OpenGL ES 2.0, що дозволяє розробникам вебконтенту вбудовувати в браузер, які підтримують HTML5, повноцінну 3D-графіку, не вдаючись до посередництва плагінів.

ВСТУП

Фрактали є структурами, що мають самоподібність на різних масштабах. Вивчення їх оптичних властивостей допомагає краще зрозуміти природу цих структур та їх вплив на різноманітні процеси у фізиці, математиці та інших галузях.

Наукові дослідження оптичних властивостей фракталів залишаються актуальними через кілька аспектів. Дослідження фрактальних структур може призвести до розробки нових матеріалів та оптичних пристроїв з унікальними властивостями. Наприклад, фрактальні антени, оптичні фільтри та лінзи можуть мати покращені характеристики.

Дослідження фрактальних структур у контексті квантової фізики може вести до розуміння їх впливу на квантові явища та можливість застосування у квантових технологіях. Фрактальний аналіз може бути корисним у медичних дослідженнях, наприклад, в аналізі зображень біомедичних структур або в розвитку нових методів обробки медичних зображень.

Дослідження оптичних властивостей фракталів може вести до розробки нових методів вимірювань та обробки даних, що можуть мати широкі застосування в різних галузях. Використання фрактальних принципів у технологіях зображення та датчиках може призвести до покращення якості та чутливості пристроїв.

Узагальнюючи, дослідження оптичних властивостей фракталів не тільки розширює наше розуміння природи цих структур, але також відкриває нові можливості для розробки технологій і застосувань у різних галузях науки та техніки.

Представлена робота присвячена визначенню пропускної здібності просторових фракталів різної форми, в залежності від їх властивостей, таких як прозорість та ступінь розвитку. Розроблено відповдне програмне забезпечення для виконання експериментальних досліджень.

Експерименти проводились з пірамідальними фракталами різних видів та правил для формування кожної наступної ітерації. Визначається тривимірна сцена з сформованим об’єктом та площиною під ним, камера та точкове світло спрямовані в його напрямку.

При генерації фракталу надається можливість обрати фігуру, кількість ітерацій, характеристики матеріалу, такі як: прозорість, блиск, металевість, колір, поворот за віссю, позиція, характеристики джерела освітлення: колір, інтенсивність, відстань, кут, позиція. В ході експериментів зокрема було необхідно з’ясувати яким чином вид та ступінь розвитку об’ємного фракталу впливає на його загальну прозорість.

Обрано наступний сценарій експерименту: береться три види фрактальної фігури, кількість ітерацій від одного до восьми, прозорість матеріалу від 0.1 до одиниці, оберти за осями у радіанах від –π до +π та виконується перебір великої кількості комбінацій цих показників.

Експеримент виконується автоматично, показники прозорості збираються у файл на локальному обчислювальному пристрої. Для проведення експериментів була необхідність проводити велику кількість поодиночних генерацій фрактальних фігур із збором показників прозорості. Для виконання цих операцій вручну, знадобилася б значна кількість часу рутинної роботи. Використовувалася технологія selenium web driver розповсюджена у сучасних IT компаніях для автоматизації процесу тестування програмного забезпечення.

Дослідження передбачає отримання інсайтів у взаємозв'язок між параметрами генерації фракталів та їхніми оптичними властивостями. Різні конфігурації параметрів призводять до унікальних оптичних ефектів, що можуть бути використані для подальших досліджень у галузі оптики, комп'ютерної графіки та мистецтва.

За допомогою розробленого ПЗ, можуть генеруватися моделі для подальшої розробки фізичних прототипів, підібравши відповідні

характеристики матеріалу.

Актуальність роботи. В сучасному світі оптика та оптичні технології знаходять застосування в різних галузях науки та промисловості. Важливим аспектом цих досліджень є вивчення властивостей оптичних матеріалів та структур, які можуть мати значний вплив на розробку нових оптичних пристроїв та систем. Однією з цікавих областей вивчення оптичних властивостей є дослідження фрактальних поверхонь, які характеризуються нерівномірністю та структурною складністю на будь-якому масштабі.

Фрактали – це геометричні об'єкти, які мають складну структуру, яка повторюється на різних масштабах. Вони зустрічаються в природі та в технологічних застосуваннях, і їх властивості можуть бути досліджені з різних точок зору, включаючи оптичний аспект. Фрактали представляють собою математичні об'єкти зі складними формами, які можна використовувати для вивчення глибин математичних концепцій, таких як рекурсія та самоподібність.

Розроблений програмний комплекс може бути корисним для розробки нових методів генерації фракталів, розширюючи уявлення про комп'ютерну графіку. Моделювання світла та тіней для фракталів може бути використане для вивчення поведінки світла в складних структурах. Оптичні властивості фракталів можуть мати практичне застосування в розробці нових матеріалів та технологій. Використання фракталів для моделювання біологічних об'єктів може сприяти розумінню їх структури та функцій, що корисно для медичних та біологічних досліджень. Аналіз тіней допомагає в зображенні та розпізнаванні особливостей на медичних зображеннях. Створення та візуалізація складних фракталів може бути використане для творчого вираження в художній галузі, дозволяючи художникам створювати унікальні та захоплюючі візуальні твори. Використання такого програмного комплексу у навчальних цілях може допомагати студентам зрозуміти абстрактні математичні та фізичні концепції через візуалізацію.

Дослідження оптичних властивостей розвиває розуміння принципів освітлення та тіней. Застосування в AR та VR дозволить користувачам взаємодіяти з фракталами у віртуальному середовищі, розширюючи можливості дослідження та навчання.

Даний проект полягає розорбці інструментарію для моделювання фрактальних поверхонь, для подальших експериментальних дослідженнь їх оптичних характеристик, та подальший аналіз результатів.

Цей проект є актуальним, оскільки дослідження фрактальних поверхонь може мати практичний вплив на покращення вже існуючих та/або розробку нових оптичних матеріалів для промислових та наукових застосувань, таких як безпілотні апарати, літаки, лінзи, сенсори, та інші. При цьому, вивчення оптичних властивостей фрактальних структур може відкрити нові можливості для покращення оптичних систем та пристроїв.

В цілому, створення програмного комплексу для генерації та аналізу фракталів відкриває нові перспективи в ряді наукових та технологічних областей, сприяючи розвитку знань, технологій та творчості.

1 ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

1.1 Стан проблеми. Теоритичний аспект: фрактали та самоподібність, фрактальна розмірність

Щоб розглянути проблему цієї теми, ми починаємо з ширшого концепту - L-системи, які представляють собою певний вид формальної граматики. У контексті нашої роботи ми особливо цікавими вважаємо використання цієї граматики для визначення правил, що керують заміщенням об'єкту, що визначений з самого початку та від якого розпочинається конструкція. L-системи можуть бути застосовані у випадках, включаючи генерацію самоподібних фракталів.

G = (*V*, ω, *P*),

V — це множина символів, що містять елементи, які можуть бути замінені (змінні), так і елементи, які не можуть бути замінені ("константи" або "термінальні символи")

ω (старт, аксіома або ініціатор) - це рядок символів з V, що визначає початковий стан системи

P — множина правил, що визначають, яким чином змінні можуть бути замінені комбінаціями констант та інших змінних. Що породжує правило складається з двох рядків, прототип та наступник. Для будь-якого символу A, що входить до алфавіту V, що не входить до лівої частини правил P, передбачається правило виведення

A → A. Ці символи називають константами або термінальними символами.

Правила граматики L-системи використовуються ітеративно, розпочинаючи з аксіоми (початкового стану). Кожна ітерація включає застосування якнайбільше правил. Важливо відзначити, що відмінність L-системи від формальної мови, яка генерується за формальною граматикою, полягає в тому, що на кожній ітерації в L-системі може бути використано декілька правил одночасно. У формальних мовах, що генеруються граматиками, застосовується лише одне правило на ітерацію. Якщо б правила

виведення в L-системі застосовувалися по одному, ми могли б легко згенерувати мову, а не L-систему. Таким чином, L-системи є підмножиною мов.

L-система вважається контекстно-вільною, якщо кожне правило виведення посилається лише на окремі символи, а не на їхніх сусідів. Контекстно-вільні L-системи формально описуються контекстно-вільною граматикою. У випадку, коли правило залежить як від окремого символу, так і від сусідніх, система отримує назву контекстно-залежної L-системи.

Якщо для кожного символу існує рівно одне правило, то таку L-систему називають детермінованою (детермінованою контекстно-незалежною L-системою позначається як D0L система). У випадку, коли кілька правил існують, і кожне обирається з певною ймовірністю кожної ітерації, ми маємо справу зі стохастичною L-системою.

Наведемо приклади L-систем:

Дерево Піфагора:

- змінні: 0, 1
- константи: [,]
- аксіома: 0
- правила: (1 → 11), (0 → 1[0]0)

Фігура будується рекурсивним застосуванням правил виведення аксіомі. Кожен символ вхідного рядка перевіряється у списку правил, щоб визначити, на що слід замінити символ. У цьому прикладі «1» на вході перетворюється на «11» на виході, а «[» не змінюється. Застосовуючи правила виведення до аксіоми "0", отримаємо:

аксіома: 0 - нуль

1-а рекурсія: 1[0]0

Друга рекурсія: 11[1[0]0]1[0]0

3-я рекурсія: 1111[11[1[0]0]1[0]0]11[1[0]0]1[0]0

…

Ми бачимо, що рядки швидко ростуть у довжині та складності. Рядок можна намалювати у вигляді малюнка за допомогою черепашої графіки, де кожному символу відповідає графічна операція для черепахи. У цьому прикладі черепасі можуть бути такі команди:

- 0: малюємо відрізок, що закінчується листом
- 1: малюємо відрізок
- [: кладемо у стек положення та кут малювання, повертаємо вліво на 45 градусів
-]: зчитуємо зі стека положення та кут, повертаємо вправо на 45 градусів

«Покладемо в стек» і «виберемо зі стека» відноситься до LIFO-стеку. Коли інтерпретатор зустрічає «[», поточне положення черепахи і кут руху зберігаються в стеку, коли зустрічається «]», положення і кут відновлюються. Якщо кілька значень заносяться у стек, відновлюється останнє значення. У літературі L-системи, використовують такий підхід до розгалуження, називають «bracketed L-systems» (дужні L-системи) [2].

Застосовуючи ці графічні правила до отриманого раніше рядку, ми маємо (рис 1.0):

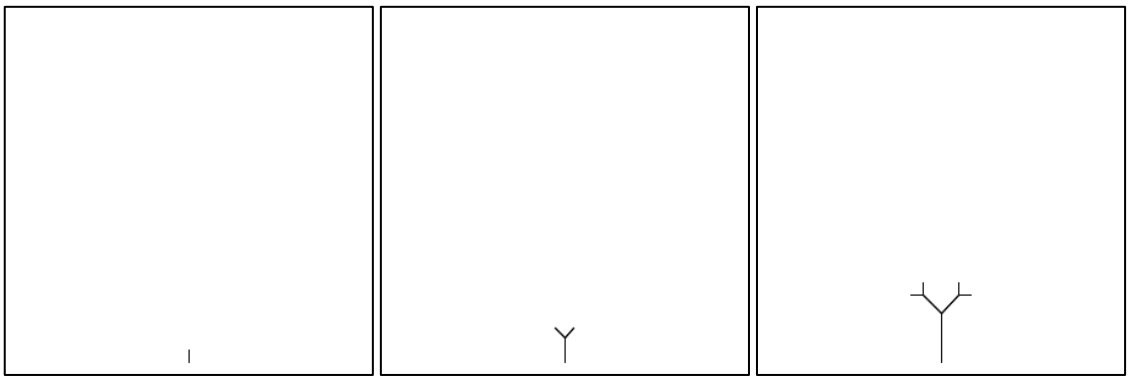


Рисунок 1.0 – Дерево Піфагора, Аксіома => Перша рекурсія => Друга рекурсія

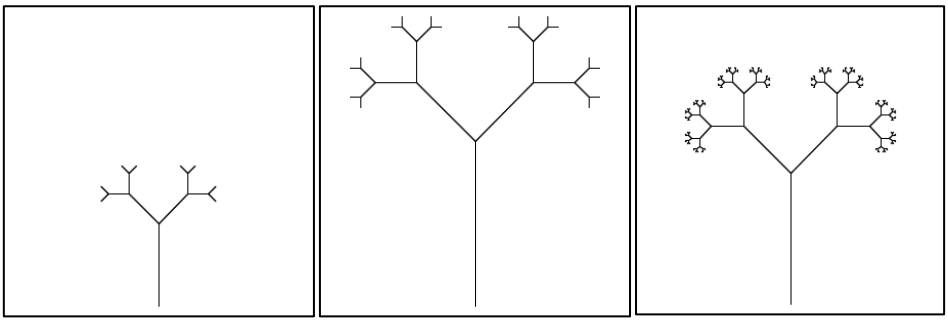


Рисунок 1.1 – Дерево Піфагора, Третя рекурсія => Четверта рекурсія => Сьома рекурсія

Трикутник Серпінського (намальований за допомогою L-системи):

змінні: F G

константи: + −

старт: F−G−G

правило: (F → F+G+F+G), (G → GG)

кут: 120°

F і G означають «малюємо відрізок», + означає «повернути праворуч на кут», а означає «повернути вліво на кут».

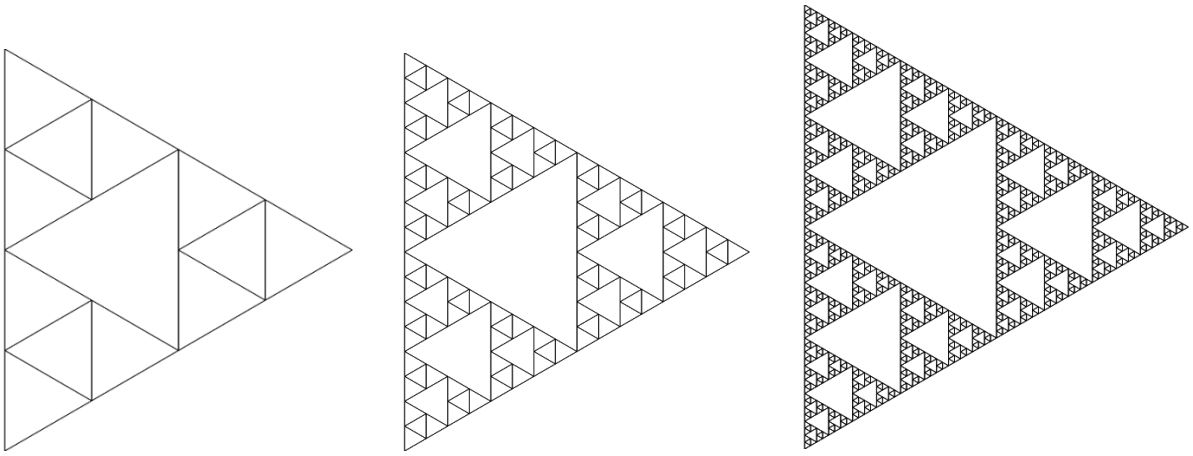


Рисунок 1.2 – Трикутник Серпінського, n = 2 => n = 4 => n = 6

Визачимо поняття фракталу. У найзагальнішому вигляді фрактал демонструє межу. Фрактали моделюють складні фізичні процеси та динамічні системи. Основний принцип фракталів полягає в тому, що простий процес, який проходить через нескінченну кількість ітерацій, стає дуже складним процесом. Фрактали намагаються змоделювати складний процес, шукаючи простий процес під ним.

Більшість фракталів працюють за принципом зворотного зв'язку. Над фрагментом даних виконується проста операція, а потім повертається знову. Цей процес повторюється нескінченно багато разів. Межею виробленого процесу є фрактал.

Майже всі фрактали принаймні частково самоподібні. Це означає, що частина фрактала ідентична всьому фракталу, за винятком менших розмірів.

Фрактали можуть виглядати дуже складно. Однак зазвичай це дуже прості процеси, які дають складні результати. І ця властивість переходить до теорії хаосу. Якщо щось має складні результати, це не обов'язково означає, що воно мало складний вхід. Можливо, закрався хаос (у чомусь такому простому, як помилка округлення для розрахунку), що дало складні результати.

Фрактальні розміри використовуються для вимірювання складності об'єктів. Тепер у нас є способи вимірювання речей, які традиційно були безглуздими або неможливими для вимірювання.

Нарешті, фрактальні дослідження є досить новою сферою інтересів. Завдяки комп'ютерам ми тепер можемо створювати та декодувати фрактали за допомогою графічних зображень. Однією з гарячих областей досліджень сьогодні є фрактальне стиснення зображень. Багато веб-сайтів присвячені обговоренню цього. Головним недоліком фрактального стиснення зображень і фракталів загалом є обчислювальна потужність, необхідна для їх кодування, а часом і декодування. У міру того, як персональні комп'ютери стають

швидшими, ми можемо побачити масові програми, які фрактально стискають зображення.

Об’єкт є самоподібним лише в тому випадку, якщо ви можете розбити його на довільну кількість маленьких частин, і кожна з цих частин є копією всієї структури. Нижче наведено кілька прикладів самоподібності. Червоний контур вказує на деякі самоподібності об’єкта.

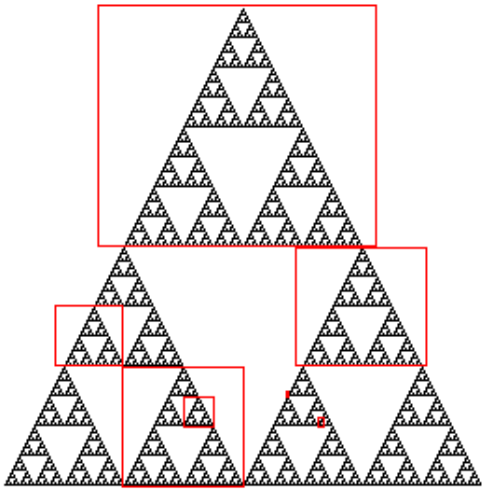


Рисунок 1.3 – Самоподібність «Трикутнику Серпінського»

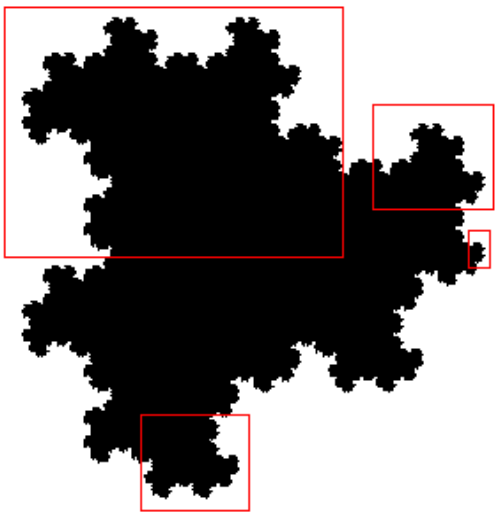


Рисунок 1.4 – Самоподібність «Кривої дракона»

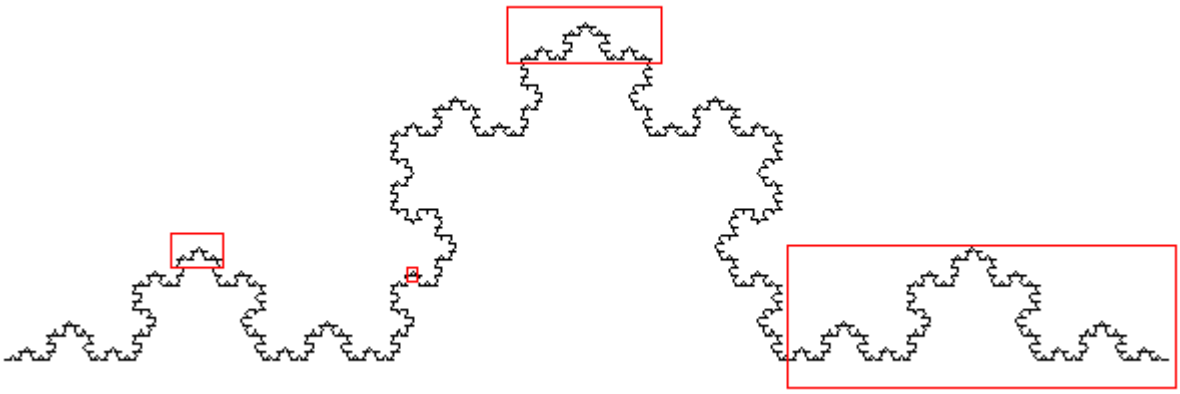


Рисунок 1.5 – Самоподібність «Кривої коха»

Фрактальна розмірність:

Всім відома розмірність прямої, квадрата і куба: один, два і три відповідно. Крім того, ми можемо виміряти відстань, площу та об’єм цих об’єктів. Однак який внутрішній розмір нирки чи мозку і як ми вимірюємо площу їх поверхні? А як щодо шматочка брокколі або цвітної капусти? Ось тут нам може допомогти фрактальний вимір. Фрактальний вимір дозволяє нам виміряти складність об’єкта.

Класичним прикладом цього є спроба виміряти берегову лінію Великої Британії. Насправді точно виміряти довжину берегової лінії неможливо. Приплив постійно приходить або відходить, що означає, що сама берегова лінія постійно змінюється. Тому будь-які звичайні вимірювання не мають сенсу.

Фрактальний вимір дозволяє нам вимірювати ступінь складності, оцінюючи, як швидко наші вимірювання збільшуються або зменшуються, коли наш масштаб стає більшим або меншим.

Розглянемо фрактальну розмірність – розмірність самоподібності. Існує багато різних видів фрактальної розмірності. Інші типи включають

топологічну розмірність, розмірність Хаусдорфа, розмірність Мінковського та евклідовий розмір. Важливо відзначити, що не всі типи вимірювання розмірів дадуть однакову відповідь на одну проблему. Однак наші вимірювання розмірів дадуть ту саму відповідь.

Перш ніж пояснювати алгоритми вимірювання розмірів, треба пояснити, як працює степеневий закон. По суті, дані поводяться за степеневим законом, якщо вони відповідають такому рівнянню: y = c * x ^ d, де c – константа. Один зі способів визначити, чи дані відповідають степеневому закону, – побудувати графік залежності log(y) від log(x). Якщо графік є прямою лінією, то це степенева залежність із нахилом d. Виявляється, методи вимірювання фрактальної розмірності, які ми збираємося обговорити, значною мірою залежать від степеневого закону.

Розмірність самоподібності. Щоб виміряти самоподібний розмір, зображення має бути самоподібним. Степеневий закон виконується і в цьому випадку є: a = 1 / (s ^ D), де a – кількість штук, s – коефіцієнт зменшення, а D – міра самоподібного розміру.

Наприклад, якщо лінію розбити на три частини, кожна з них матиме одну третину довжини оригіналу. Тому a = 3, s = (1 / 3) і D = 1.

В іншому прикладі, якщо квадрат розбити на чотири частини, кожна сторона матиме половину початкової довжини сторони. Тому a = 4, s = (1 / 2) і D = 2.

Для Трикутнику Серпінського, оригінальний трикутник розрізаний навпіл, і є три частини. Тому a = 3, s = (1 / 2) і D = 1,5850.

1.2 Вибір графічного рушію та аналоги у галузі веб-розробки

Поставлена задача вимагає рендерігу більш складних фракталів, ніж

наведені у попередньому пункті цієї роботи. Принаймні, це мають бути об’ємні фігури та можливість працювати з освітленням.

Для максимальної продуктівності, обрано більш сучасний та зручний стек технологій, а саме – web. Наразі, у вебі розрізнюють наступні рушії для рендеріну графіки:

Three.js — одна з найпопулярніших бібліотек JavaScript для створення та анімації тривимірної комп’ютерної графіки у веб-браузері за допомогою WebGL. Це також чудовий інструмент для створення 3D-ігор для веб-браузерів.

Оскільки Three.js базується на JavaScript, відносно легко додати будь-яку інтерактивність між 3D-об’єктами та інтерфейсами користувача, такими як клавіатура та миша. Це робить бібліотеку ідеальною для створення 3D-ігор у Web.

Переваги:

- Легко освоїти – найважливіша перевага Three.js — окрім його здатності дуже добре виконувати складну візуалізацію — полягає в тому, що з ним дуже легко почати роботу.
- Безліч прикладів – через його популярність є незліченна кількість прикладів, які допоможуть вам почати. Велика спільнота – Three.js має 87,1 тис. зірок і 33,4 тис. розгалужень на GitHub. Вона має багато користувачів і значну спільноту розробників, які працюють із різними сторонніми інструментами та розширеннями для бібліотеки та створюють їх.
- Гарна документація – надійна документація зазвичай є чудовим показником потужної бібліотеки, і Three.js має чудові документи
- Чудова продуктивність – Three.js має перевагу в продуктивності порівняно з іншими бібліотеками, які я використовував
- Візуалізація PBR – Three.js має вбудовану функцію фізичної візуалізації (Physical Based Rendering), яка робить рендеринг графіки точнішим.

Недоліки:

- Відсутність конвеєра візуалізації – це робить багато сучасних методів візуалізації неможливими/неможливими для реалізації за допомогою Three.js.
- Three.js має базові функції для створення ігор, це не ігровий рушій, як PlayCanvas і Unity, який надає функції, окрім інтерактивності та візуалізації. Однак API Three.js можна використовувати для створення ігрового движка; прикладом такого є двигун Rogue.
- Відсутність підтримки – немає вбудованої підтримки для просторового індексування, створення точного відкидання променів або відсікання зрізу, а виявлення зіткнень не ефективне в складних сценаріях [7].

Головні переваги Three.js включають велику спільноту талановитих користувачів і велику кількість прикладів і ресурсів.

Створено просту обертову геометрію, щоб продемонструвати, що може Three.js. Скріншот отриманного анімованого зображення наведений нижче (рис 1.6).

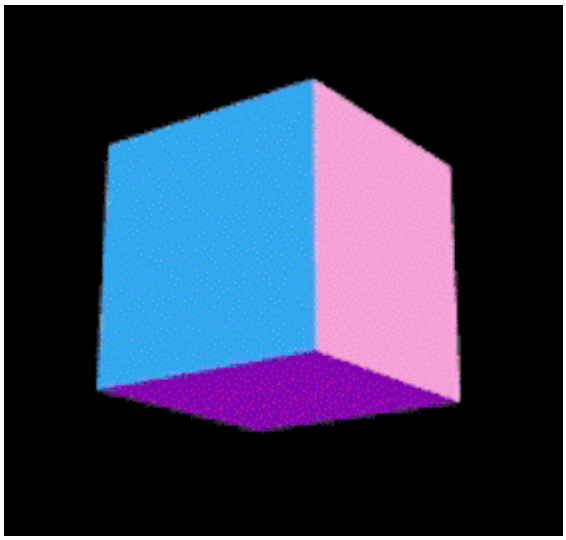


Рисунок 1.6 – Куб, намальований за допомогою ThreeJS

Pixi.js - рушій для створення насиченої та інтерактивної 2D-графіки з

підтримкою кросплатформних програм. Цей механізм створення HTML5 дає змогу створювати анімації та ігри без попереднього знання WebGL API.

Переваги:

- Швидка продуктивність – як і Three.js, Pixi.js дуже швидкий;
- Велика спільнота: 38,2 тис. зірок і 4,7 тис. розгалужень на GitHub, Pixi.js може похвалитися великою спільнотою користувачів/розробників;
- Підтримка багатьох платформ: також як Three.js, Pixi.js підтримує кілька платформ, як-от Web і власні системи, такі як Android, iOS, Windows і mac;
- Простий API – простий для розуміння;
- Підтримка резервних варіантів WebGL і Canvas: Pixi.js використовує рендерер WebGL, але також підтримує резервний варіант Canvas.

Недоліки:

- Не повне рішення: Pixi.js з гордістю підтримує лише рендерер;
- Не підлаштований під тривимірну графіку.

Піхі є сильним вибором у більшості сценаріїв, особливо якщо необхідно створити орієнтовану на продуктивність двовимірну інтерактивну графіку з урахуванням сумісності пристроїв. Підтримка Піхі резервного використання Canvas у випадках збою WebGL є особливо привабливою функцією [8].

Скріншот з результатом наведено нижче (рис 1.7).

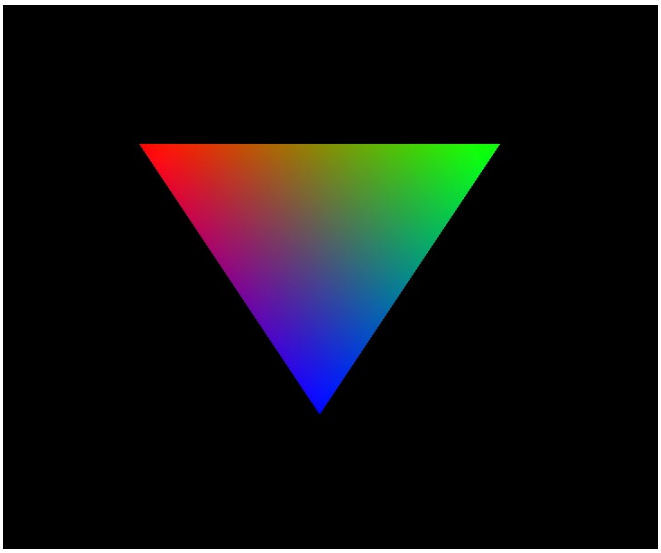


Рисунок 1.7 – Зображення, отримане за допомогою PixiJS

Phaser - це кросплатформенний ігровий рушій, який дозволяє створювати ігри на основі JavaScript і HTML5 і компілювати їх для багатьох платформ. Наприклад, якщо стоїть задача скомпілювати свою гру для iOS, Android та інших рідних програм за допомогою інструментів сторонніх розробників.

Переваги:

- Структурно надійний – відомо, що Phaser має добре продуману структуру

Підтримка TypeScript: бібліотека підтримує використання TypeScript для розробки;

- Гариній механізм розробки ігор;
- Велика спільнота – хоча й не така велика, як перші дві бібліотеки, Phaser має значну спільноту з 33,2 тис. зірок і 6,9 тис. розгалужень на GitHub;
- Велика кількість плагінів – Phaser підтримує величезний список плагінів. Це включає плагіни phaser-matter-collision, navmesh, phaser-input і slick-ui, щоб назвати декілька;
- Підтримка WebGL і Canvas – Phaser підтримує WebGL і ввімкнув Canvas як запасний варіант.

Недоліки:

- Розмір збірки: розмір збірки Phaser для настільного ПК досить великий;
- Погана підтримка розробки для мобільних пристроїв: для створення нативних програм для мобільних пристроїв потрібно використовувати Cordova або будь-яку іншу структуру третьої сторони;
- Керування станом: може бути дещо складно розпочати роботу з менеджером стану Phaser [9].

Phaser добре підходить для розробки кросплатформних ігрових програм.

Його підтримка широкого спектру плагінів і велика спільнота розробників, які створюють ігри за допомогою Phaser, спрощують початок роботи з фреймворком.

Скріншот з результатом наведено нижче (рис. 1.8).

Babylon.js - це потужний, простий, відкритий рушій та механізм візуалізації, укладений у дружню структуру JavaScript.

Переваги:

- Playground: Babylon надає інструмент Playground для тестування перед початком повної розробки — і має чудову документацію для завантаження;
- Сильна підтримка спільноти: у Babylon є форум із великою спільнотою активних розробників і користувачів, які дуже допомагають. Фреймворк має 18,9 тисяч зірок і 3 тисячі розгалужень на GitHub;
- Оновлена кодова база: фреймворк має часто оновлену кодову базу та активну розробку сторонніх інструментів. Офіційний репозиторій нещодавно оновлено;
- Відтворення PBR: підтримка відтворення PBR чудова;
- Вотум довіри: Babylon використовується та підтримується такими великими брендами, як Adobe, Microsoft тощо;

Технічне обслуговування: помилки часто усуваються швидко.

Рисунок 1.8 – Зображення, отримане за допомогою Phaser

Недоліки:

- Відсутність зрілості: Babylon вперше вийшов у 2013 році, що робить його досить молодим порівняно з багатьма його конкурентами;
- Документація: двигуну не вистачає документації API;
- Не підходить для невеликих проектів через більш різку криву в плані налаштування проекту та швидкості розробки на перших етапах [10].

Скріншот з результатом наведено нижче (рис. 1.9).

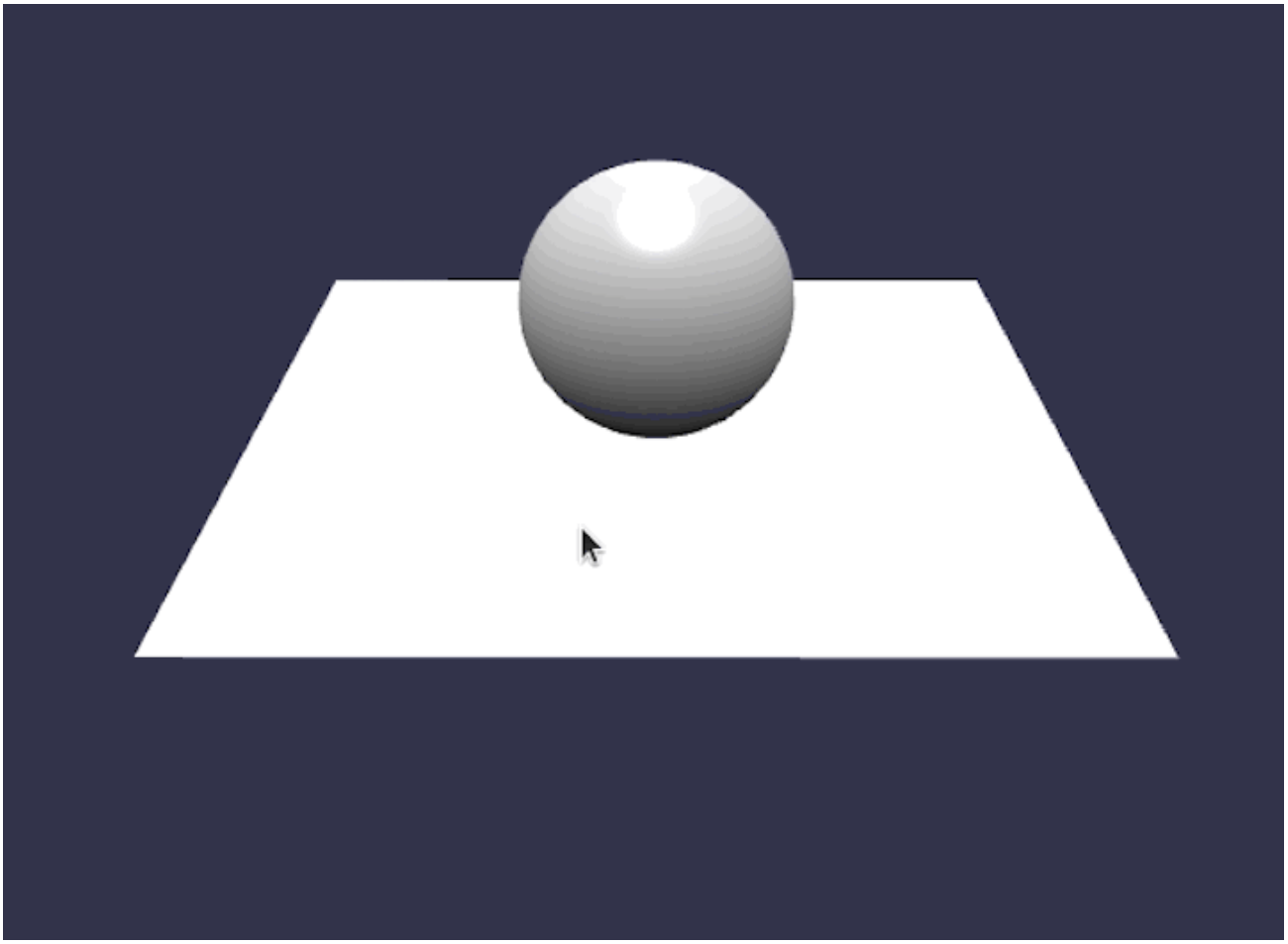


Рисунок 1.9 – Зображення, отримане за допомогою Babylon.js

Matter.js - це двовимірний механізм JavaScript для фізики твердих тіл для Web. Незважаючи на те, що це фізичний механізм JavaScript, ви можете поєднувати його з різними пакетами та плагінами для створення цікавих веб-

ігор.

Переваги:

- Захоплюючі функції: Matter.js пропонує багато функцій, як-от тверді, складені та композитні тіла, стабільне укладання та відпочинок, збереження руху та багато іншого.

Недоліки:

- Немає CCD: відсутність у Matter.js безперервного виявлення зіткнень (continuous collision detection) викликає проблему, коли об’єкти, що швидко рухаються, проходять крізь інші об’єкти.

Matter.js є одною з найкращих бібліотек для створення простих рухомих анімаційних об’єктів. Окрім іншого, це бібліотека фізики, яка більше зосереджується на 2D-об’єктах. Однак, його можна комбінувати його зі сторонніми рішеннями для створення динамічних ігор [11].

PlayCanvas - використовує HTML5 і WebGL для запуску ігор та іншого інтерактивного 3D-контенту в будь-якому мобільному або настільному браузері. Незважаючи на те, що PlayCanvas безкоштовний і має відкритий вихідний код, він більше зосереджується на ігровому движку, ніж на движку візуалізації. Тому він більше підходить для створення

3D-ігри, які використовують WebGL і HTML5 Canvas.

Переваги:

- Ігровий рушій: на відміну від решти, PlayCanvas — це ігровий рушій із функціями, які ви інакше не знайдете в бібліотеці чи фреймворку з відкритим кодом: PlayCanvas — це інструмент із відкритим кодом для потужної розробки ігор;
- Оптимізовано для мобільних пристроїв: платформа розробки ігор орієнтована на мобільні пристрої;
- Нульовий час компіляції: нульовий час компіляції двигуна природно пришвидшує процес;
- Конвеєр активів: PlayCanvas використовує найкращі практики,

щоб дозволити вам вирішити, як і в якій формі доставлятися ваш вміст;

- Інтегрований механізм фізики: Ви можете легко інтегрувати фізику у свою гру за допомогою потужного движка bullet physics ammo.js;
- Гаряче перезавантаження: вам не потрібно перезавантажувати браузер щоразу, коли ви вносите зміни;
- Механізми візуалізації лише в браузерах: PlayCanvas має розширені функції WebGL, які працюють лише в браузерах.

Недоліки:

- Платні приватні проекти: безкоштовний рівень не підтримує приватні проекти, тому весь код і активи розміщені у відкритому доступі;
- Зсув зіткнення: немає зміщення зіткнення;
- Брак прикладів: посібників для PlayCanvas дуже мало.

PlayCanvas чудово підходить для створення невеликих публічних або шкільних проектів [12]. Нажаль, якщо потрібні додаткові функції та більше контролю над процесом, то слід дивитися в бік інших аналогів.

Скріншот з результатом наведено нижче (рис 1.10).

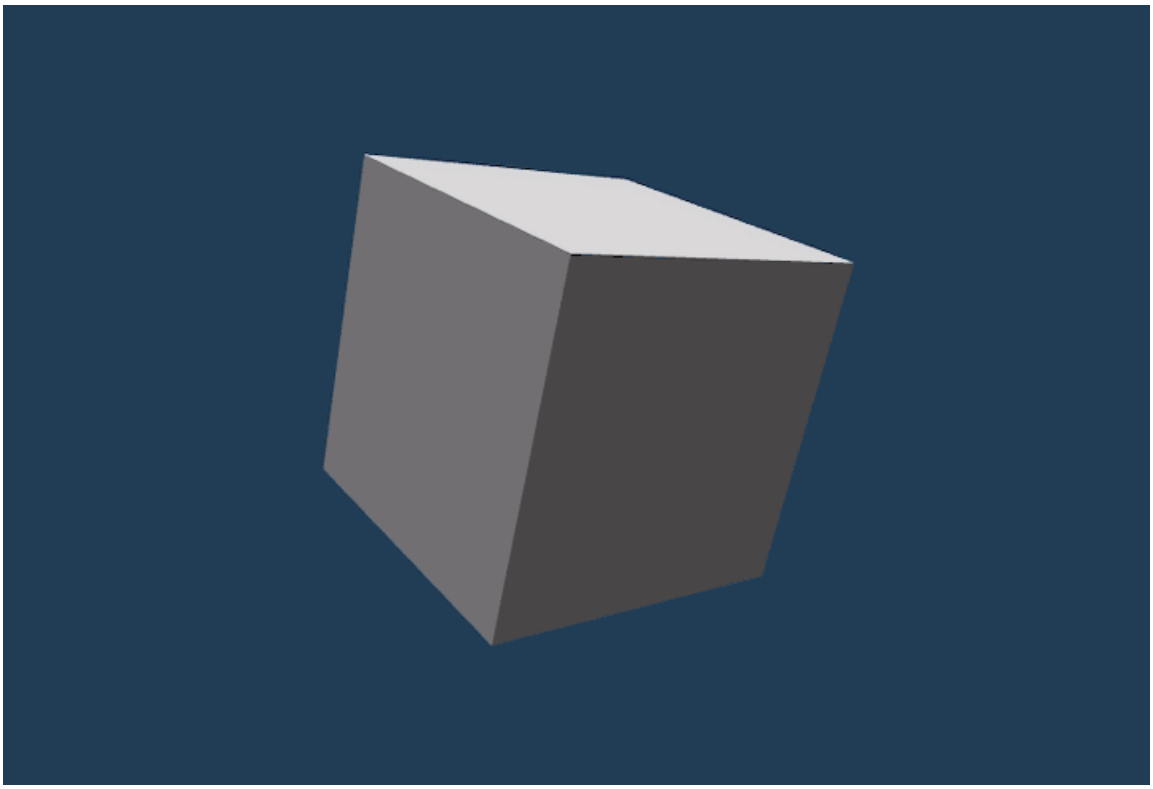


Рисунок 1.10 – Зображення, отримане за допомогою PlayCanvas

Поділившись на переваги, недоліки та випадки використання, пов’язані з кожним веб графічним рушієм, можна підсумувати важливі аспекти, необхідні для виконання поставленої задачі обрано Three.js – він досить добре підходить, якщо потрібен більше механізм візуалізації, ніж ігровий двигун. Завдяки його популярності в екосистемі розробників відносно легше знайти достатньо ресурсів, які допоможуть реалізувати поставлені вимоги.

Додатково можна відмітити, що якщо б було більше зосередження на розробці інтерактивного додатку накшталт грі, то кращим був би Babylon.js із вбудованою системою фізики. Babylon також має оновлену кодову базу та активну розробку, а функція плейграунду є чудовим інструментом для тестування.

1.3 Розбір обраного технологічного стеку та графічного рушію

Для імплементації поставленої задачі було обрано програмний застосунок

із клієнт-серверною архітектурою.

Мова програмування TypeScript – обгортка над JavaScript, що дає нам можливість використовувати типізацію, на виході компілюється у JavaScript (ECMAScript6).

Використані технології:

Система контроля версій Git. Для зручного відстежування змін та стабільних версій програми, було вирішено використовувати менеджер контроля версій. Також це даже перспективу можливості зберігати результати на віддаленому сервері для запобігання проблеми втрати даних.

За бібліотеку для створення трьохвимірної графіки обрано ThreeJS. Використана мова гіпертекстової позмітки HTML та каскадні таблиці стилів CSS. Север написано за допомогою бібліотеки Express та рантайму NodeJS. Babel для транспіляції коду. Webpack – для збірки файлів у бандл. Webpack dev server – для комфортної розробки та перезборки, автоматичного оновлення вебсторінки у браузері при внесенні змін у коді - тобто замість того щоб перезапускати сервер та клієнт після кожної зміни, всі зміни відстежуються та процес виконується автоматично – dev сервер сам оновлює збірку та сторінку у браузері, тобто сервер запускається тільки один раз, при старті.

Для роботи з тінню частково застосовано мову програмування шейдерів GLSL – синтаксис базується на мові програмування ANSI C, однак, через його специфічну спрямованість, з нього були вилучені багато можливостей, для спрощення мови та підвищення продуктивності. У мову долучені додаткові функції і типи даних, наприклад для роботи з векторами і матрицями. Перевагою GLSL є переносимість коду між платформами і операційними системами. Версії GLSL розвивались поруч з версіями OpenGL. Починаючи з OpenGL 3.3, версія GLSL збігається з версією OpenGL.

В сучасному світі багато існуючих можливостей, що були доступні на десктопних продуктах, почали імплементуватися і у Вебі. Таким прикладом можна назвати WebGL. В намірах розробників поширити стандарт WebGL не

тільки в браузерах персональних комп'ютерів, а й у мобільних інтернет-пристроях. До робочої групи WebGL входять Khronos Group, представники провідних розробників інтернет-браузерів, таких як Apple Safari, Google Chrome, Mozilla Firefox і Opera, а також фахівці AMD і Nvidia.

WebGL створений на основі OpenGL ES 2.0 з підтримкою API для 3D-графіки. Він використовує елемент canvas з HTML5, а також взаємодіє з DOM. Автоматичне управління пам'яттю відбувається завдяки мові JavaScript. Шейдери у WebGL запрограмовані безпосередньо на мові GLSL.

Використання WebGL API напряму може вимагати написання багато шаблонного коду, якщо не використовувати деякі корисні бібліотеки, наприклад створені для легкої роботи з шейдерами, чи для завантаження графічних сцен та 3D об'єктів у популярних на сьогодення форматах. JavaScript бібліотеки вбудовані (або портовані у WebGL) забезпечують додатковими функціональними можливостями.

Неповний перелік бібліотек, які надають багато високотехнічних можливостей: three.js, O3D, OSG.JS, CopperLicht і GLGE. Також розвиваються ігрові рушії на WebGL, включаючи Unreal Engine і Stage3D/Flash-based Away3D – бібліотека високого рівня, також має порт на WebGL реалізований на TypeScript. Існують і простіші бібліотеки, котрі надають тільки векторні та матричні математичні можливості для шейдерів – sylvester.js. Іноді вона використовується в поєднанні з розширенням WebGL – glUtils.js.

Є також деякі 2D бібліотеки, побудовані на основі WebGL, такі як Cocos2d-x або Pixi.js, які були реалізовані для підвищення продуктивності (так само, як Starling Framework відносно Stage3D у світі Flash). Коли WebGL не доступний, вирішення задач 2D бібліотек перекладається на HTML5 canvas.

Видалення помилок рендерингу через надання майже повного доступу до GPU обмежує продуктивність реалізацій JavaScript. Деякі з них були

переадресовані на asm.js. (Точно так само, як впровадження Stage3D відкрило проблеми продуктивності в межах ActionScript, які були розглянуті в рамках проектів, таких як CrossBridge).

Створення контенту для WebGL сцен часто означає, використання стандартних 3D інструментів для створення та експорту сцен, відтворення їх у відповідних форматах для зовнішніх програм. Наприклад у авторському програмному забезпеченні для 3D, такому як Blender або Autodesk Maya. Але існує також деяке специфічне WebGL забезпечення, наприклад, CopperCube і онлайн редактор Clara.io на основі WebGL. Онлайн платформи, такі як Sketchfab і Clara.io дозволяють користувачам безпосередньо завантажувати свої 3D моделі і зображати їх за допомогою вбудованого WebGL переглядача.

Крім того, Mozilla Firefox реалізувала інструменти з вбудованим WebGL, починаючи з версії 27, котрі дозволяють редагування vertex і фрагменти шейдерів. З'явився також ряд інших інструментів задля налагодження й профілювання проектів.

X3D також виконали проект під назвою X3DOM для створення X3D і VRML контенту, що працює на WebGL. 3D модель розташовують між XML тегами <X3D> у HTML5, а інтерактивний скрипт використовує JavaScript і DOM для відображення. BS Content Studio разом з InstantReality X3D експортером може експортувати X3D у HTML і опрацювати його на WebGL.

В свою чергу, WebGL рекомендують використовувати не напряму для написання 3D графіки, а скоріше як платформу для написання бібліотек, які в свою чергу будуть використовуватися для написання 3D графіки.

Метою цієї дипломної роботи не було створення однієї з таких бібліотек, тому було використано одну з найпоширеніших з існуючих – ThreeJS.

Three.js – це бібліотека JavaScript з кросбраузерністю та інтерфейсом

прикладного програмування (API), що використовується для створення та відображення анімованої 3D-комп'ютерної графіки у веббраузері. Three.js скрипти можуть використовуватися спільно з елементом HTML5 Canvas, SVG або WebGL. Вихідний код бібліотеки Three.js розміщений у сховищі на GitHub.

Three.js дозволяє створювати пришвидшену на GPU, 3D-анімацію, використовуючи мову JavaScript як частину вебсайту, не покладаючись на власні плагіни браузера. Це можливо завдяки появі WebGL.

Створення складних тривимірних комп'ютерних анімації може бути дещо простішим завдяки бібліотекам високого рівня, таких як Three.js або GLGE, SceneJS, PhiloGL, а також ряд інших. Адже ці бібліотеки відображаються в браузері без зусиль, необхідних для традиційного автономного додатку чи плагіна.

Three.js включає такі функції:

- Рендерери: Canvas, SVG або WebGL
- Вміст: додавання і видалення об'єктів в режимі реального часу; туман
- Камери: перспективна або ортографічна
- Анімація: каркаси, пряма кінематика, інверсна кінематика, морфінг, ключові кадри
- Джерела світла: зовнішнє, спрямование, точкове; тіні: кинуті і отримані
- Шейдери: повний доступ до всіх OpenGL-шейдерів(GLSL)
- Об'єкти: мережі, частинки, спрайт, лінії, скелетна анімація і інше
- Геометрія: площина, куб, сфера, тор, 3D текст і інше;
- Data loaders: binary, image, JSON and scene

- Завантажники даних: двійковий, зображення, JSON і сцена
- Експорт та імпорт: утиліти, що створюють Three.js-сумісні JSON файли з форматів: Blender, openCTM, FBX, Autodesk 3ds MAX та Obj
- Підтримка: документація по API бібліотеки знаходиться в процесі постійного розширення і доповнення, є публічний форум і велике співтовариство

Приклади: на офіційному сайті можна знайти більше 150 прикладів роботи зі шрифтами, моделями, текстурами, звуком і іншими елементами сцен.

Висновки до першого розділу

Вибір графічного рушію для веб-застосунку є важливим етапом при розробці, оскільки це впливає на продуктивність, швидкість завантаження, можливості взаємодії з користувачем і загальний досвід використання застосунку.

Деякі ключові аспекти проблематики вибору графічного рушію для веб-застосунків:

- Продуктивність (швидкість рендерингу): різні движки можуть мати різні швидкості обробки та рендерингу компонентів. Швидкий рендеринг є важливим для забезпечення плавного і швидкого відгуку веб-застосунку;
- Підтримка браузерів (сумісність з браузерами): деякі рушії можуть краще підтримуватися в певних браузерах, що може впливати на кросплатформенність застосунку;
- Розмір бібліотеки (розмір файлів): розмір використовуваних бібліотек та додаткових компонентів може впливати на завантаження сторінки. Важливо обрати ефективний рушій, щоб уникнути надмірно великого розміру файлів та погіршення продуктивності;

- Розвиток та підтримка (активність розвитку): важливо обирати рушій, який активно розвивається і підтримується спільнотою або вендором;
- Спільнота та екосистема (заєднання користувачів): існування активної спільноти може допомогти знаходити рішення для проблем в процесі розробки, вивчати нові функції та отримувати підтримку в разі необхідності;
- Інтеграція з іншими технологіями (сумісність з іншими бібліотеками та фреймворками): якщо планується використовувати інші бібліотеки чи фреймворки, важливо, щоб обраний рушій мав можливість інтеграції з ними;
- Можливості та функціональність (підтримка сучасних стандартів): деякі графічні рушії можуть краще підтримувати сучасні технології, такі як Progressive Web Apps (PWA) або Web Components;
- Безпека (захист від вразливостей): важливо, щоб обраний рендеринг-рушій був безпечним і захищав ваш застосунок від вразливостей.

При виборі необхідно ретельно розглядати ці аспекти, а також провести тестування продуктивності та сумісності.

Треба додати, що не існує ідеального рішення – для будь-якої задачі можуть бути різні шляхи та кожне з них буде мати свої переваги та недоліки, тому головне зважати на всі можливі нюанси перед вибором, бо для переписування проекту з однієї технології на іншу, в разі певних невдач з початковим вибором, можуть знадобитися значні часові ресурси.

Врешті решт, необхідно чітко розуміти, що навіть якщо обране рішення буде містити не весь спектр можливостей, змінювати курс в бік розробки власного рушію може мати непередбачувані наслідки – це досить непроста задача, яка потребує багато часу та відповідних навичок.

У даному випадку, було обрано використовувати існуюче рішення для рендерінгу тривімірної графіки – ThreeJS.

2 РОЗРОБКА ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ДЛЯ ГЕНЕРАЦІЇ ТА РЕНДЕРІНГУ ФРАКТАЛІВ, ЗБОРУ ПОКАЗНИКІВ ПРОЗОРОСТІ ТА АВТОМАТИЗАЦІЯ ДАНИХ ПРОЦЕСІВ.

2.1 Розробка програмного комплексу рендерінгу тривимірної сцени з моделлю фракталу

Для того, щоб було можливим виконання коду в рантаймі браузера (в даному випадку, рушій Google Chrome – V8) – необхідно, щоб на виході це був JavaScript. Як було зазначено вище, розробка буде відбуватися на TypeScript. Механізм транспіляції відбувається наступним чином:

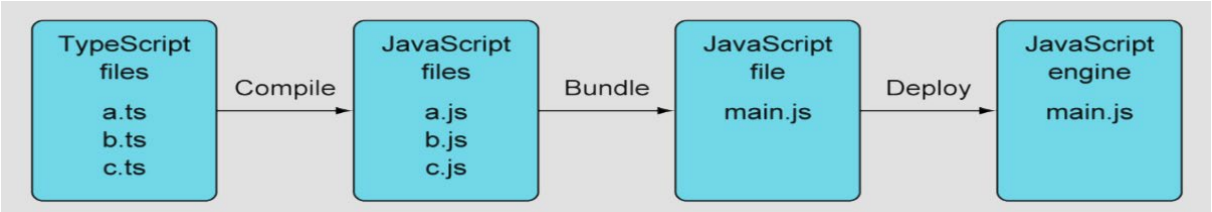


Рисунок 2.1 – Механізм транспіляції коду JavaScript у TypeScript

Тобто, кожен тайпскрипт файл компілюється у відповідний джаваскрипт файл, що далі безпосередньо збирається в один бандл, який у свою чергу завантажується у браузер та виконується браузерним движком.

Подібний механізм загальний, та актуальний як для клієнтського коду, так і для серверного.

Нижче розглянуто схему архітектури NodeJS (рис 2.2).

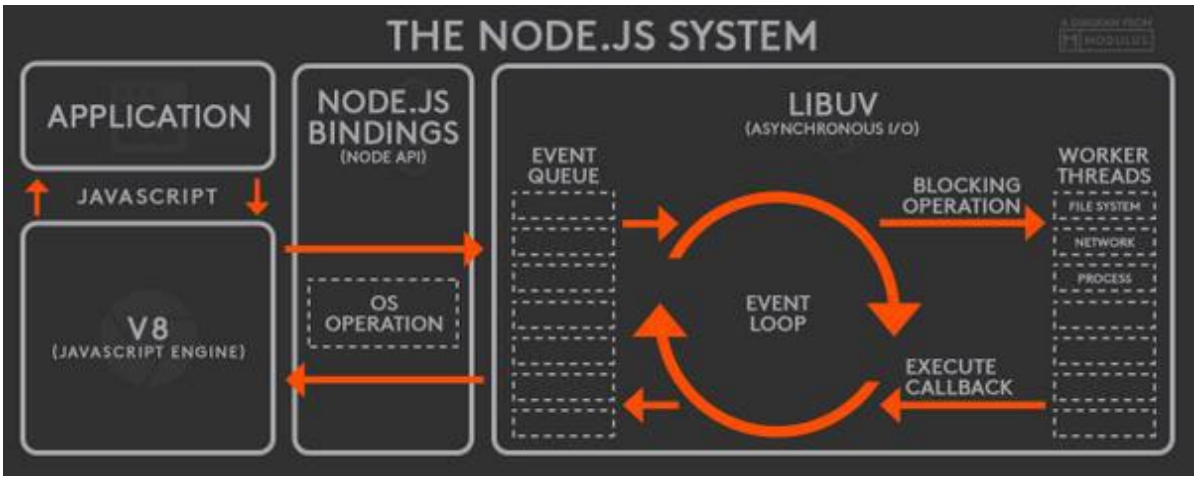


Рисунок 2.2 – Схема архітектури NodeJS

Частини архітектури Node.js:

- Запити. Вхідні запити можуть бути блокуючими (складні) або неблокуючими (прості), залежно від завдань, які користувач хоче виконати у веб-програмі;
- Сервер Node.js — це серверна платформа, яка приймає запити від користувачів, обробляє ці запити та повертає відповіді відповідним користувачам;
- Черга подій на сервері Node.js зберігає вхідні запити клієнтів і передає ці запити один за одним у цикл подій;
- Пул потоків складається з усіх потоків, доступних для виконання деяких завдань, які можуть знадобитися для виконання запитів клієнта;
- Event Loop нескінченно отримує запити та обробляє їх, а потім повертає відповіді відповідним клієнтам;
- Зовнішні ресурси. Для блокування запитів клієнтів потрібні зовнішні ресурси. Ці ресурси можуть бути для обчислень, зберігання даних тощо.

Архітектура Node.js має кілька переваг, які дають серверній платформі явну перевагу в порівнянні з іншими серверними мовами:

- Обробка кількох одночасних запитів клієнтів є швидкою та легкою;
- Завдяки використанню черги подій і пулу потоків сервер Node.js забезпечує ефективну обробку великої кількості вхідних запитів;
- Немає необхідності створювати кілька потоків;
- Event Loop обробляє всі запити один за одним, тому немає необхідності створювати кілька потоків. Натомість одного потоку достатньо для обробки блокуючого вхідного запиту;
- Вимагає менше ресурсів і пам'яті. Сервер Node.js у більшості випадків потребує менше ресурсів і пам’яті через те, як він обробляє вхідні запити. Оскільки запити обробляються по одному, загальний процес стає менш обтяжливим для пам’яті;

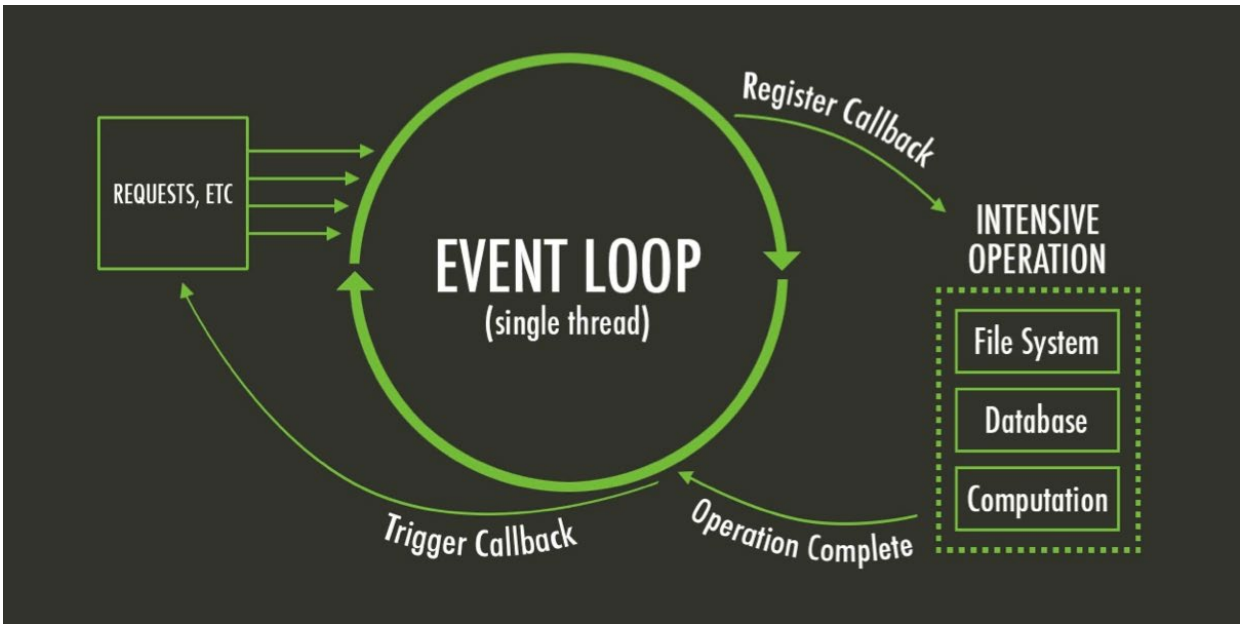


Рисунок 2.3 – Схема мехнізму Event Loop, як частини архітектури NodeJS

Загальна схема основного програмного комплексу виглядає наступним чином (рис. 2.4).

Рисунок 2.4 – Високорівнева архітектура програмного комплексу з генерації сцени з тривимірними фракталами

Для того щоб трохи ознайомитися з можливостями движку, розропачто розробку починаючи з базової тривімірної сцени. Намальовано паралелепіпед світло-блакитного кольору. Додано «навклишнє» світло для можливості побачити об’єкт на екрані.

Використано базовий матеріал для об’єкту (MeshBasicMaterial). Фігура повернена за осями X та Y.

```
import { BoxGeometry, MeshBasicMaterial, Mesh, DoubleSide, Color } from 'three';

const parallelepipedGeometry = new BoxGeometry(5, 0.5, 5);

const parallelepipedMaterial = new MeshBasicMaterial({…});

const parallelepiped = new Mesh(parallelepipedGeometry, parallelepipedMaterial);
```

Результат можна побачити нижче (рис. 2.5).

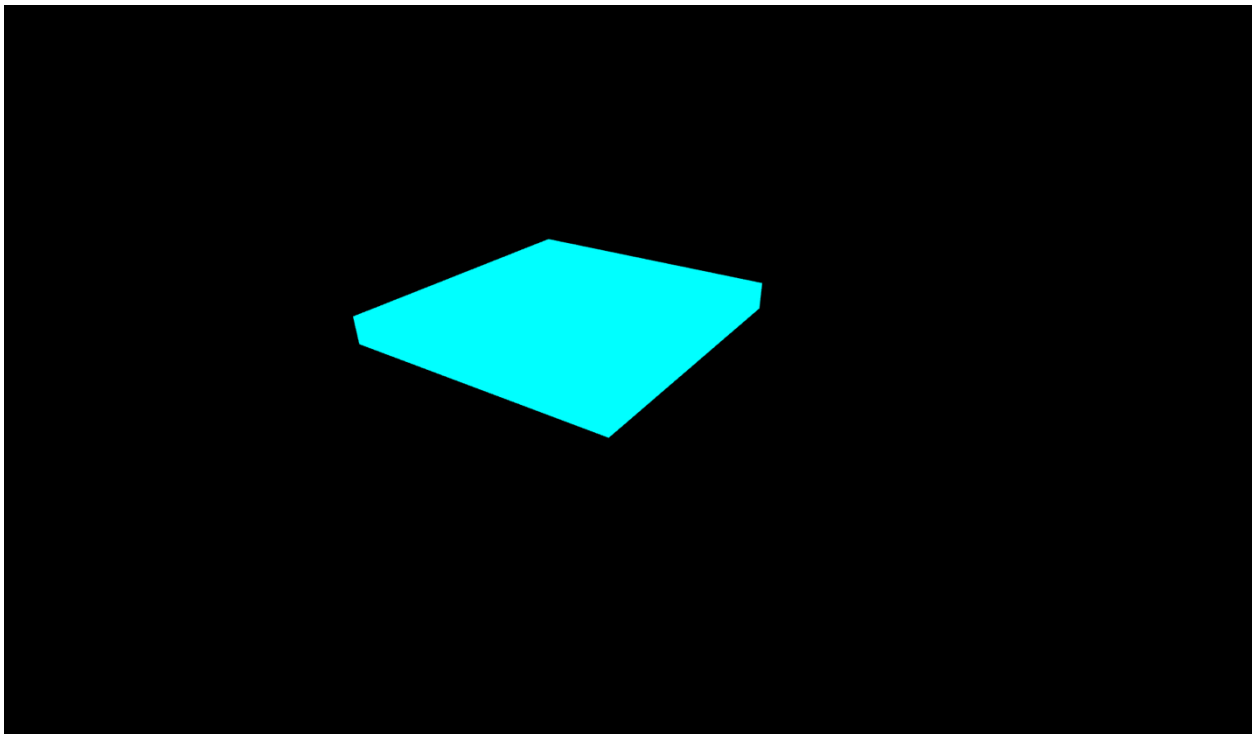


Рисунок 2.5 – Паралелепіпед, відмальований засобами ThreeJS, матеріал MeshBasicMaterial

У ході розробки було виявлено, що навіть якщо прибрати освітлення,

об’єкт все ще так само відображається на екрані.

Замінено матеріал об’єкту з MeshBasicMaterial на MeshPhysicalMaterial:

При видаленні джерела освітлення, об’єкт зникає. Тобто, він в свою чергу, у порівнянні з попереднім матеріалом, реагує на присутність/відсутність освітлення.

Навколинє освітлення хоч і дає змогу побачити об’єкт, але для проведення експериментів, необхідно мати можливість більшого контролю над кутом/інтенсивністю/напрямком світла, тому для подальшої розробки було обрано ***точкове світло (променеве)***.

Для дослідження того, як буде себе вести світло та тінь від фракталу, додано площину (рис. 2.6).

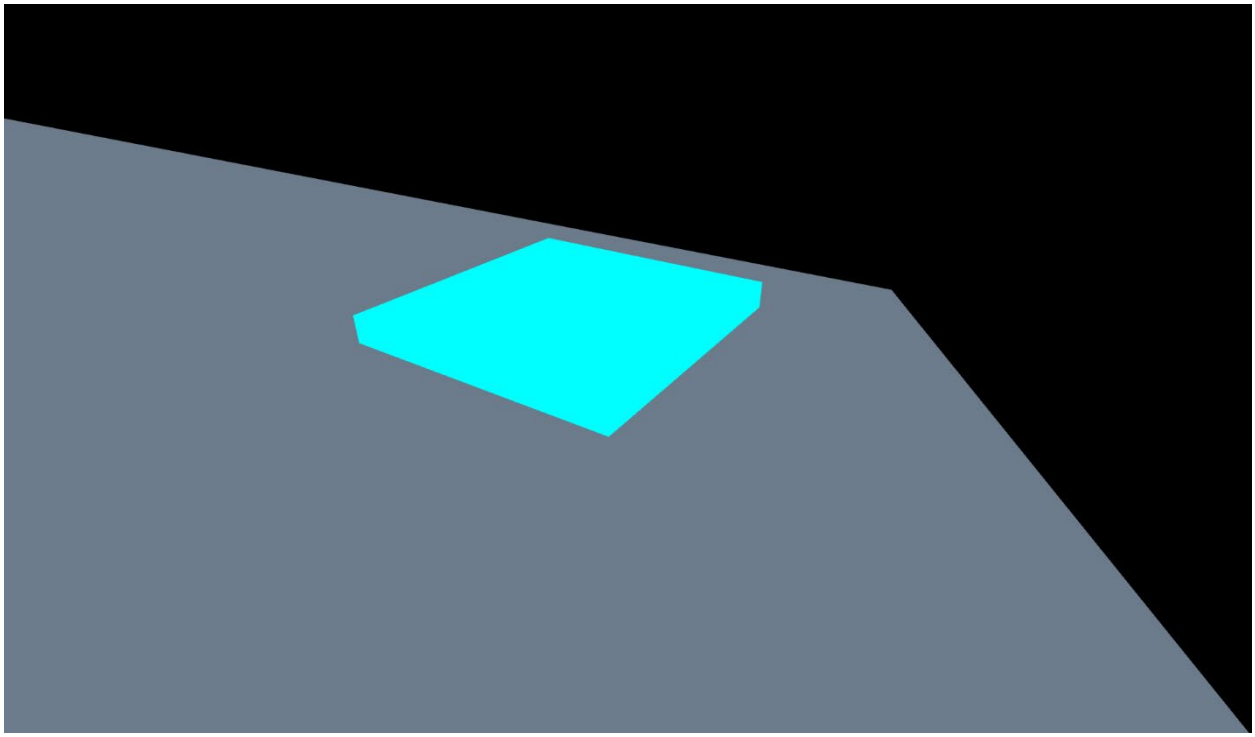


Рисунок 2.6 – Паралелепіпед, відмальований засобами ThreeJS із площиною під ним, матеріал MeshPhysicalMaterial

Додане точкове світло. Результат можна побачити нижче (рис. 2.7):

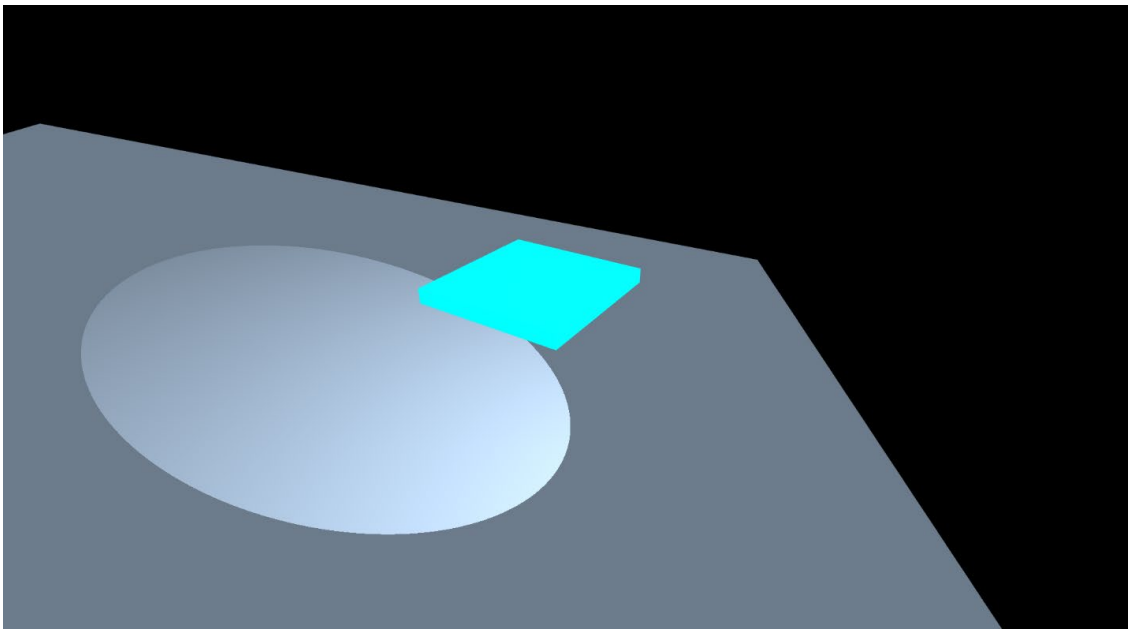


Рисунок 2.7 – Паралелепіпед, освітлений точковим світлом

Як можна побачити вище, сама фігура виглядає однаково у межах де на неї попадає світло та поза межами.

Далі об’єкту додана можливість відкидати та відображати тінь та отримано результат (рис. 2.8).

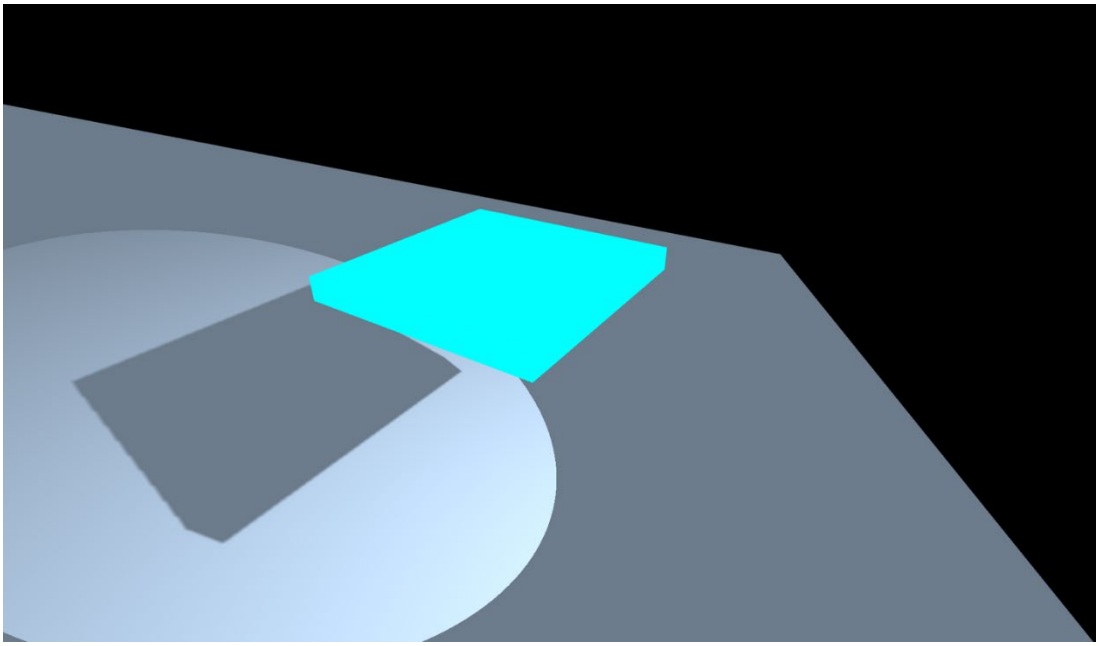


Рисунок 2.8 – Паралелепіпед, який відбиває тінь на площину

Форма тіні має коректний контур витримані лінії, кути нахилу та масштабування розміру у відповідності до відстанню між паралелепіпедом та площиною.

Далі отримано напівпрозорий матеріал та використно його для нашого об’єкту (рис. 2.9). Використаний матеріал має прозорість.

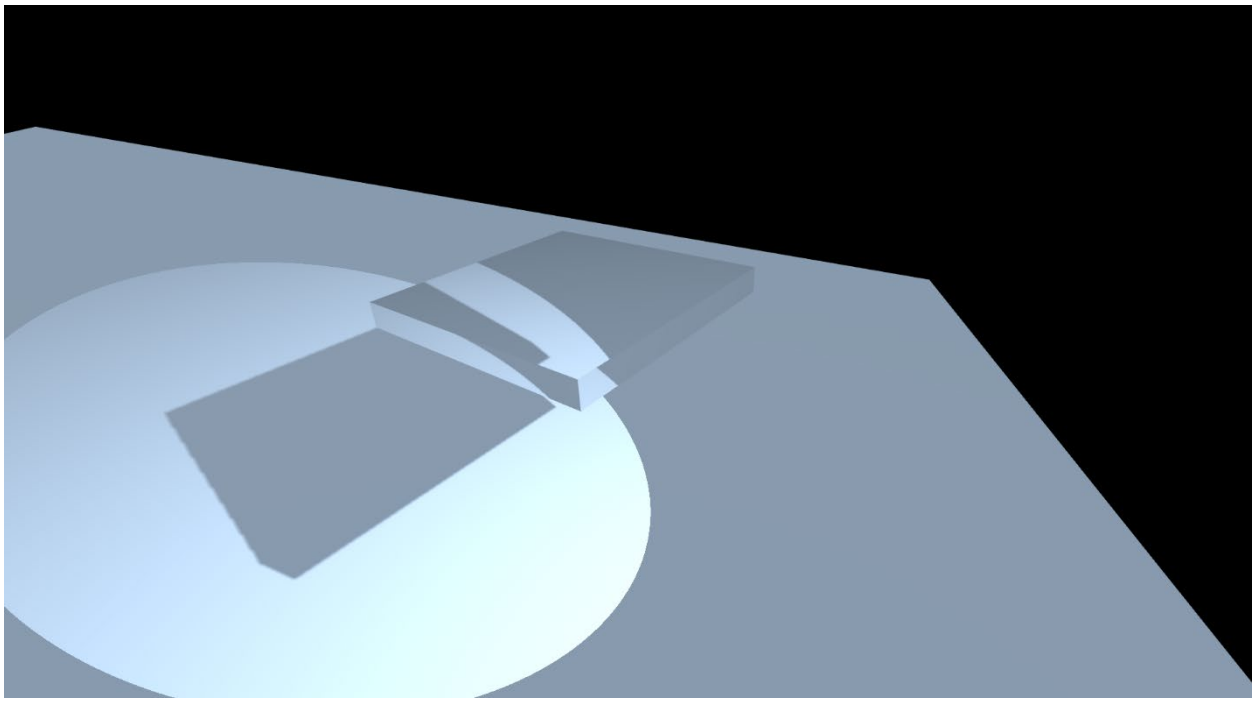


Рисунок 2.9 – Прозорий паралелепід

Як можна побачити, об’єкт скляний, має деяку товщину. Можна спостерігати відблиск світла (рис. 2.11).

Після побудови базової сцени з паралелепіпедом і точковим освітленням, було зроблено висновки щобо того який матеріал більш за все підходить для подальшої розробки. Досягнуті гарні результати у базовому випадку, проте необхідно ускладнити освновну фігуру.

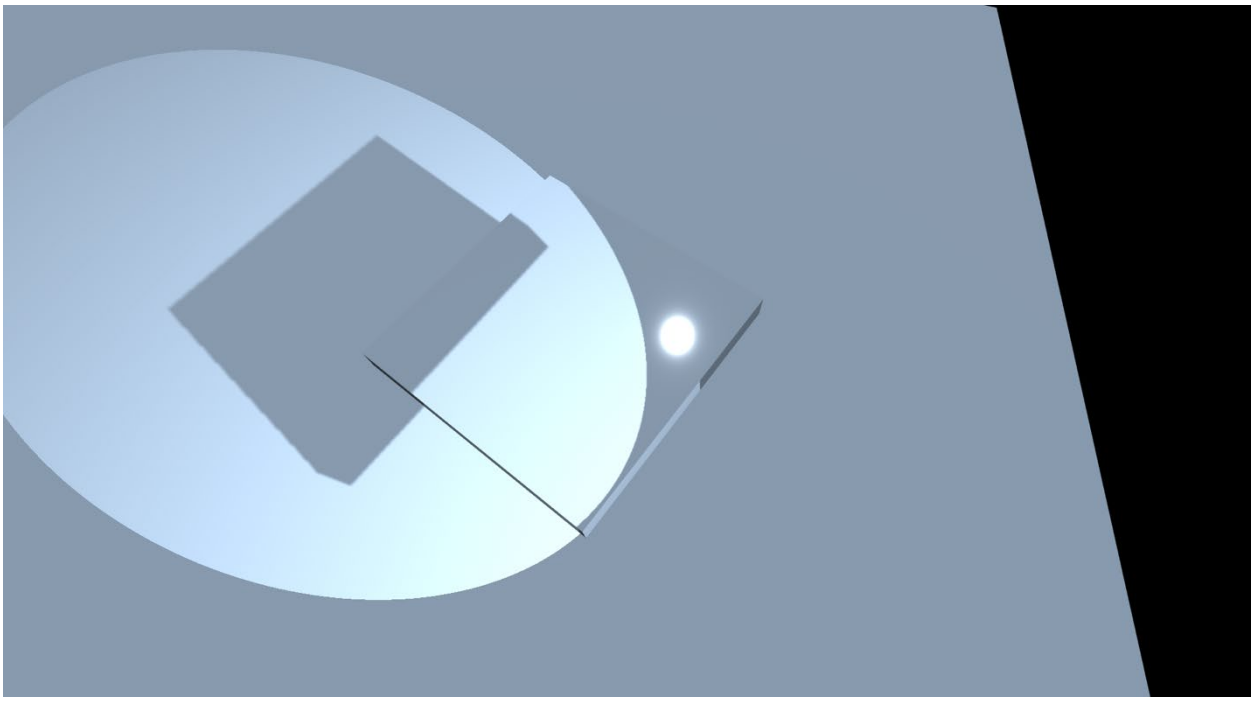


Рисунок 2.10 – Прозорий паралелепід із відблиском світла

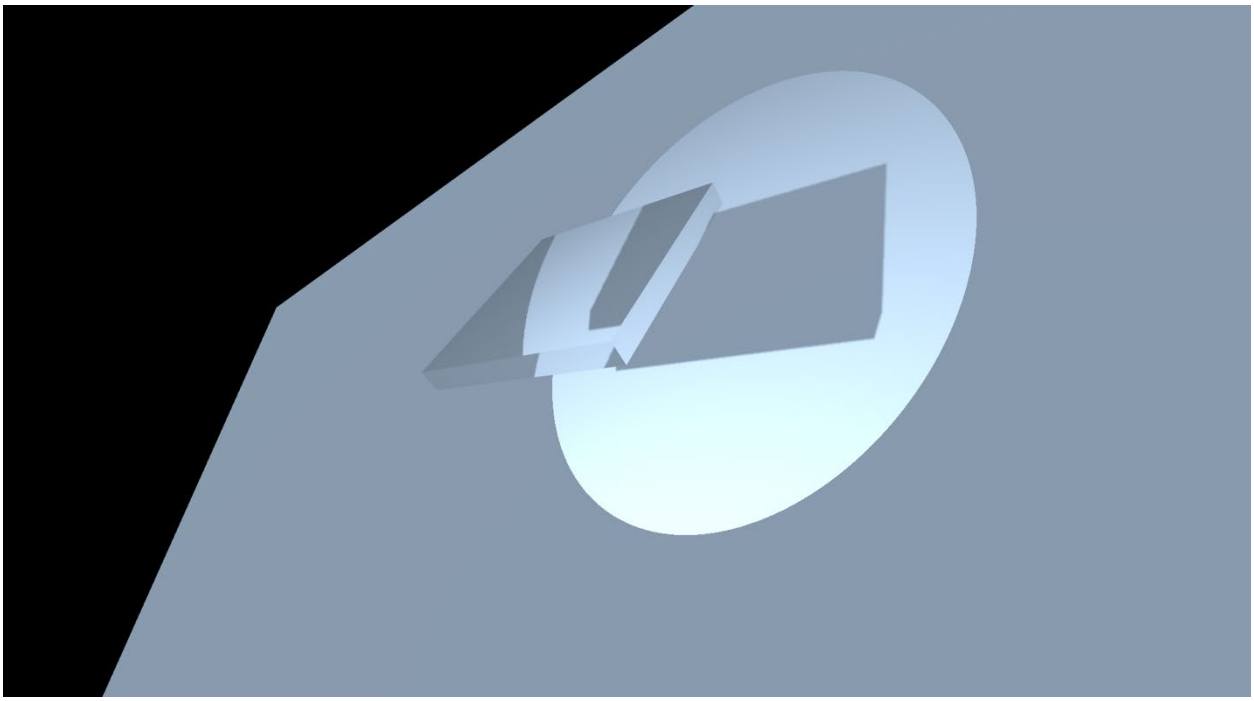


Рисунок 2.11 – Прозорий паралелепід, з іншого ракурсу

Далі побудовано 3Д фрактал – Тетраідер Серпинського. Опишемо клас «точки» та «піраміди».

```
class Point {
  constructor(x: number, y: number, z: number) {
    this.x = x;
    this.y = y;
    this.z = z;
  }
}
class Pyramid {
  constructor(a: Point, b: Point, c: Point, d: Point) {
    this.A = a;
    this.B = b;
    this.C = c;
    this.D = d;
  }
}
```

Описано алгоритм підстановки та ускладнення ступеню розвитку фракталу на подальших ітераціях.

Нижче наведено як відбувається алгоритм генерації пірамід з виконанням пошуку середніх точок, як вершин для них:

```
const getHalfPoint = (A: Point, B: Point): Point => {
  return new Point((A.x + B.x) / 2, (A.y + B.y) / 2, (A.z + B.z) / 2);
};
const createNewFigures = (figure: Pyramid): Pyramid[] => {
  const AB = getHalfPoint(figure.A, figure.B);
  const AC = getHalfPoint(figure.A, figure.C);
  const AD = getHalfPoint(figure.A, figure.D);
  const BC = getHalfPoint(figure.B, figure.C);
  const BD = getHalfPoint(figure.B, figure.D);
```

```
  const CD = getHalfPoint(figure.C, figure.D);
  const A_AB_AC_AD = new Pyramid(figure.A, AB, AC, AD);
  const B_AB_BC_BD = new Pyramid(figure.B, AB, BC, BD);
  const C_AC_BC_CD = new Pyramid(figure.C, AC, BC, CD);
  const D_AD_BD_CD = new Pyramid(figure.D, AD, BD, CD);
  return [A_AB_AC_AD, B_AB_BC_BD, C_AC_BC_CD, D_AD_BD_CD];
};
export const generatePyramids = (i, figures): Pyramid[] => {
  const figureList = [...figures];

  while (i > 0) {
    const tempList = [];

    while (figureList.length) {
      const currentFigure = figureList.pop();
      const newFigures = createNewFigures(currentFigure);
      tempList.push(...newFigures);
    }
    figureList.push(...tempList);
    i -= 1;
  }
  return figureList;
};
```

Правило заміни: кожна піраміда заміняєтсья на чотири піраміди. Результат, отриманий після трьох ітерацій (рис. 2.12).

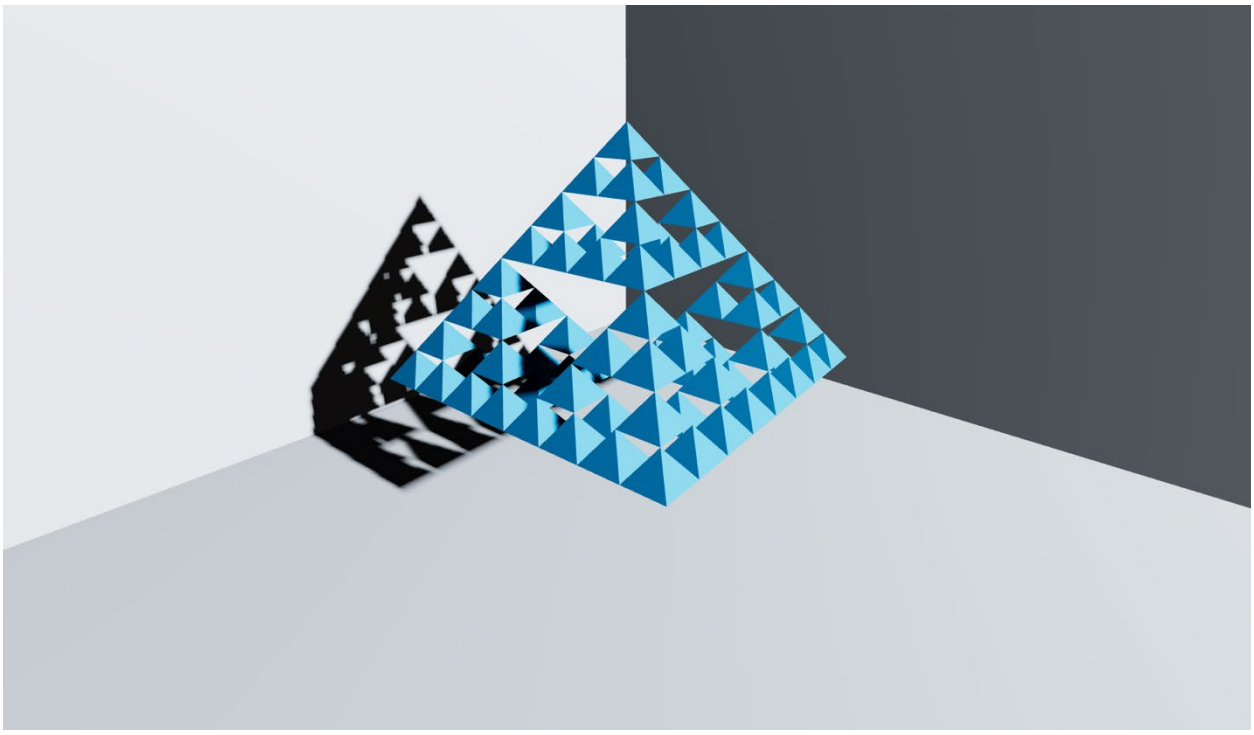


Рисунок 2.12 – Тетраідер Серпинського, n = 3

Матеріал замінено на скляний, результат наведено нижче (рис 2.13).

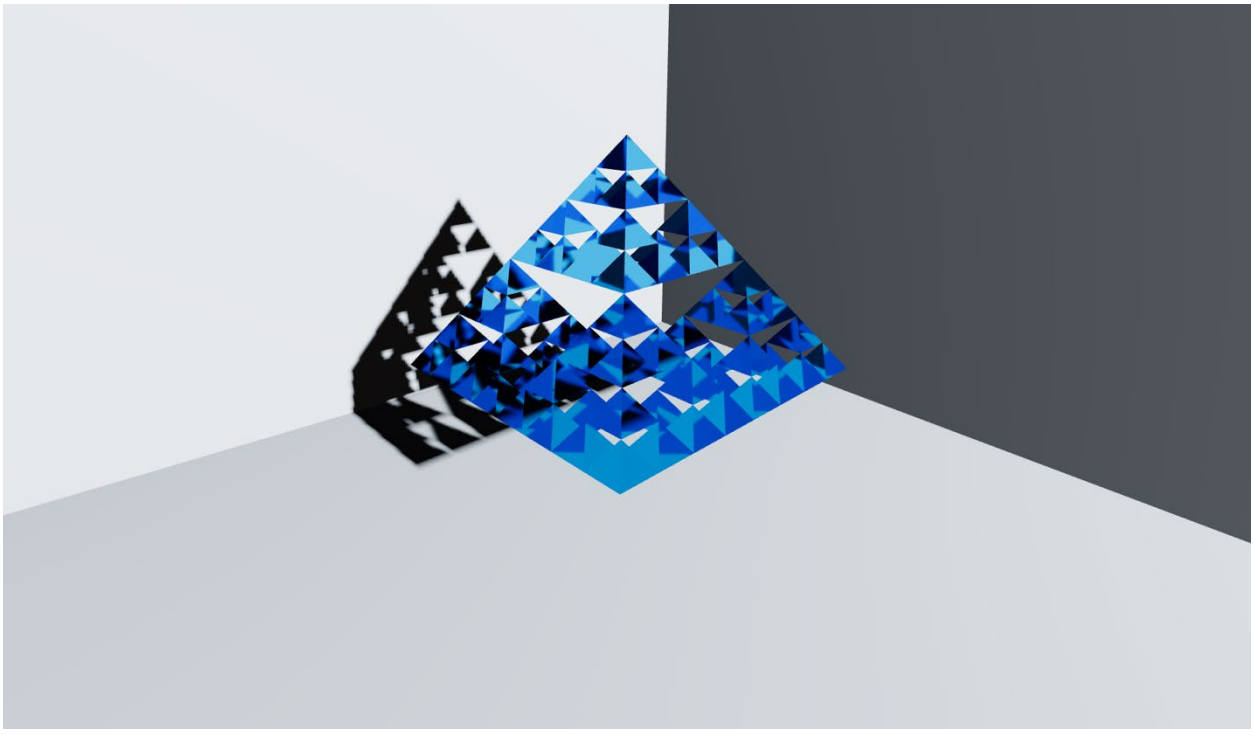


Рисунок 2.13 – Скляний Тетраідер Серпинського, n = 3

Кількість ітерації збільшена з трьох до п’яти (рис. 2.14).

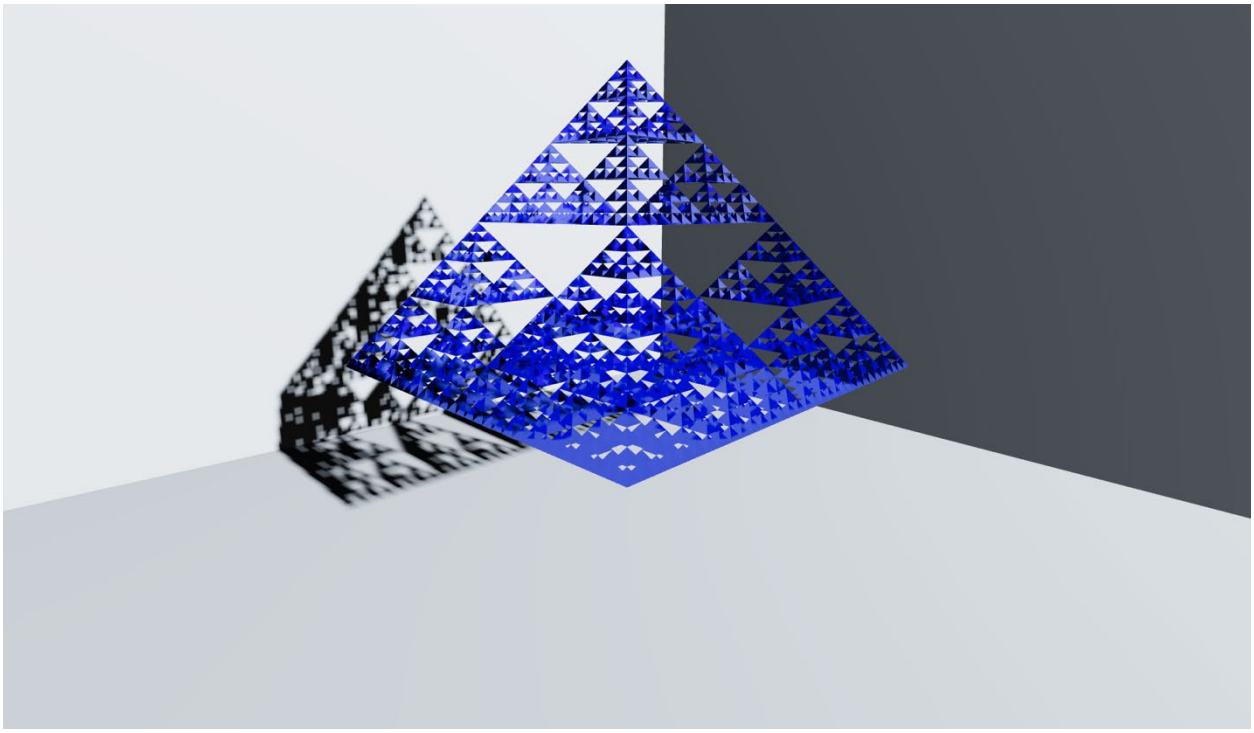


Рисунок 2.14 – Скляний Тетраідер Серпинського, n = 5

Для урізноманітення варіантів фрактальних фігур та варіацій подальших досліджень, додано інший алгоритм - для отримання дворівневої піраміди. Загальна структура алгоритму залишається такою ж, але додано крок, де будуть створюватися додаткові піраміди в середині існуючих. Нижче наведений механізм отримання мінімізованих пірамід, що будуть знаходитися у середині звичайних пірамід.

```
const getMinimizedPyramids = (figure: Pyramid) => {
    const AB = getHalfPoint(figure.A, figure.B);
    const AC = getHalfPoint(figure.A, figure.C);
    const BC = getHalfPoint(figure.B, figure.C);
```

```
const coefficientA = 0.5;
const ABC_center = new Point(
    (AB.x + figure.C.x * coefficientA) / (1 + coefficientA),
    (AB.y + figure.C.y * coefficientA) / (1 + coefficientA),
    (AB.z + figure.C.z * coefficientA) / (1 + coefficientA)
);
const ABD_center = new Point(
    (AB.x + figure.D.x * coefficientA) / (1 + coefficientA),
    (AB.y + figure.D.y * coefficientA) / (1 + coefficientA),
    (AB.z + figure.D.z * coefficientA) / (1 + coefficientA)
);
const ACD_center = new Point(
    (AC.x + figure.D.x * coefficientA) / (1 + coefficientA),
    (AC.y + figure.D.y * coefficientA) / (1 + coefficientA),
    (AC.z + figure.D.z * coefficientA) / (1 + coefficientA)
);
const BCD_center = new Point(
    (BC.x + figure.D.x * coefficientA) / (1 + coefficientA),
    (BC.y + figure.D.y * coefficientA) / (1 + coefficientA),
    (BC.z + figure.D.z * coefficientA) / (1 + coefficientA)
);
const coefficientB = 0.3;
const Ai = new Point(
    ((figure.A.x + BCD_center.x * coefficientB) / (1 + coefficientB)),
    ((figure.A.y + BCD_center.y * coefficientB) / (1 + coefficientB)),
    ((figure.A.z + BCD_center.z * coefficientB) / (1 + coefficientB))
);
const Bi = new Point(
    ((figure.B.x + ACD_center.x * coefficientB) / (1 + coefficientB)),
```

```
        ((figure.B.y + ACD_center.y * coefficientB) / (1 + coefficientB)),
        ((figure.B.z + ACD_center.z * coefficientB) / (1 + coefficientB))
      );
      const Ci = new Point(
        ((figure.C.x + ABD_center.x * coefficientB) / (1 + coefficientB)),
        ((figure.C.y + ABD_center.y * coefficientB) / (1 + coefficientB)),
        ((figure.C.z + ABD_center.z * coefficientB) / (1 + coefficientB))
      );
      const Di = new Point(
        ((figure.D.x + ABC_center.x * coefficientB) / (1 + coefficientB)),
        ((figure.D.y + ABC_center.y * coefficientB) / (1 + coefficientB)),
        ((figure.D.z + ABC_center.z * coefficientB) / (1 + coefficientB))
      );

      return new Pyramid(Ai, Bi, Ci, Di);
    }
```

Далі наведено модифікований підхід до отримання нових фігур на наступних ітераціях.

```
    const createNewFigures = (figure: Pyramid): [Pyramid[], Pyramid[]] => {
      const AB = getHalfPoint(figure.A, figure.B);
      const AC = getHalfPoint(figure.A, figure.C);
      const AD = getHalfPoint(figure.A, figure.D);
      const BC = getHalfPoint(figure.B, figure.C);
      const BD = getHalfPoint(figure.B, figure.D);
      const CD = getHalfPoint(figure.C, figure.D);
      const A_AB_AC_AD = new Pyramid(figure.A, AB, AC, AD);
      const B_AB_BC_BD = new Pyramid(figure.B, AB, BC, BD);
      const C_AC_BC_CD = new Pyramid(figure.C, AC, BC, CD);
```

```
        const D_AD_BD_CD = new Pyramid(figure.D, AD, BD, CD);

        const i_A_AB_AC_AD = getMinimizedPyramids(A_AB_AC_AD);
        const i_B_AB_BC_BD = getMinimizedPyramids(B_AB_BC_BD);
        const i_C_AC_BC_CD = getMinimizedPyramids(C_AC_BC_CD);
        const i_D_AD_BD_CD = getMinimizedPyramids(D_AD_BD_CD);

        return [[A_AB_AC_AD, B_AB_BC_BD, C_AC_BC_CD, D_AD_BD_CD],
[i_A_AB_AC_AD, i_B_AB_BC_BD, i_C_AC_BC_CD, i_D_AD_BD_CD]];
    };
```

Результат після трьох ітерацій наведено нижче (рис. 2.15).

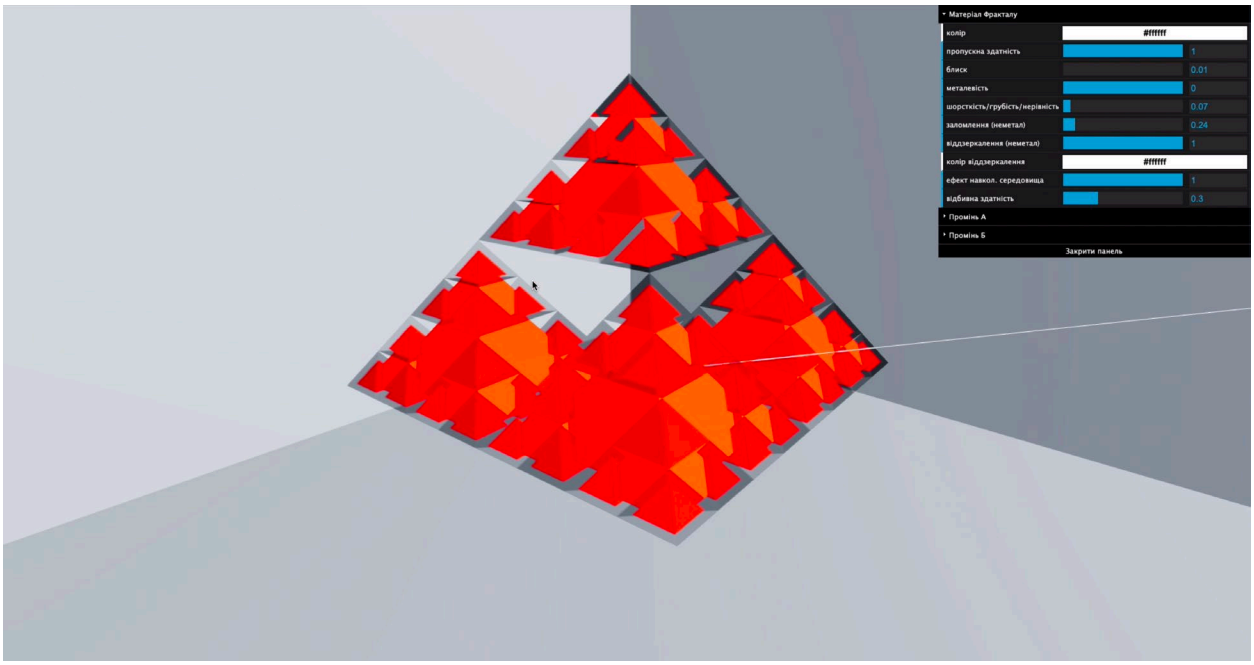


Рисунок 2.15 – Тетраідер Серпинського із пірамідами всередині, n = 3

Додатково розроблено ще один алгоритм, що також є варіацією самого першого, але тепер замість створення піраміди в піраміді, виконується поворот піраміди таким чином, щоб вони «виростали» назовні. Нижче наведено

механізм отримання нових фігур у наступних ітераціях.

```
const createNewFigures = (figure: Pyramid): Pyramid[] => {
    const AB = getHalfPoint(figure.A, figure.B);
    const AC = getHalfPoint(figure.A, figure.C);
    const BC = getHalfPoint(figure.B, figure.C);

    const coefficientA = 0.5;
    const ABC_center = new Point(
        (AB.x + figure.C.x * coefficientA) / (1 + coefficientA),
        (AB.y + figure.C.y * coefficientA) / (1 + coefficientA),
        (AB.z + figure.C.z * coefficientA) / (1 + coefficientA)
    );
    const ABD_center = new Point(
        (AB.x + figure.D.x * coefficientA) / (1 + coefficientA),
        (AB.y + figure.D.y * coefficientA) / (1 + coefficientA),
        (AB.z + figure.D.z * coefficientA) / (1 + coefficientA)
    );
    const ACD_center = new Point(
        (AC.x + figure.D.x * coefficientA) / (1 + coefficientA),
        (AC.y + figure.D.y * coefficientA) / (1 + coefficientA),
        (AC.z + figure.D.z * coefficientA) / (1 + coefficientA)
    );
    const BCD_center = new Point(
        (BC.x + figure.D.x * coefficientA) / (1 + coefficientA),
        (BC.y + figure.D.y * coefficientA) / (1 + coefficientA),
        (BC.z + figure.D.z * coefficientA) / (1 + coefficientA)
    );
    const coefficientB = 2;
    const Ai = new Point(
```

```
        (coefficientB * BCD_center.x - figure.A.x),
        (coefficientB * BCD_center.y - figure.A.y),
        (coefficientB * BCD_center.z - figure.A.z),
    );
    const Bi = new Point(
        (coefficientB * ACD_center.x - figure.B.x),
        (coefficientB * ACD_center.y - figure.B.y),
        (coefficientB * ACD_center.z - figure.B.z),
    );
    const Ci = new Point(
        (coefficientB * ABD_center.x - figure.C.x),
        (coefficientB * ABD_center.y - figure.C.y),
        (coefficientB * ABD_center.z - figure.C.z),
    );
    const Di = new Point(
        (coefficientB * ABC_center.x - figure.D.x),
        (coefficientB * ABC_center.y - figure.D.y),
        (coefficientB * ABC_center.z - figure.D.z),
    );
    const A_B_C_Di = new Pyramid(figure.A, figure.B, figure.C, Di);
    const A_B_Ci_D = new Pyramid(figure.A, figure.B, Ci, figure.D);
    const A_Bi_C_D = new Pyramid(figure.A, Bi, figure.C, figure.D);
    const Ai_B_C_D = new Pyramid(Ai, figure.B, figure.C, figure.D);

    return [A_B_C_Di, A_B_Ci_D, A_Bi_C_D, Ai_B_C_D]
};
```

Результат після проходження трьох ітерацій наведено нижче:

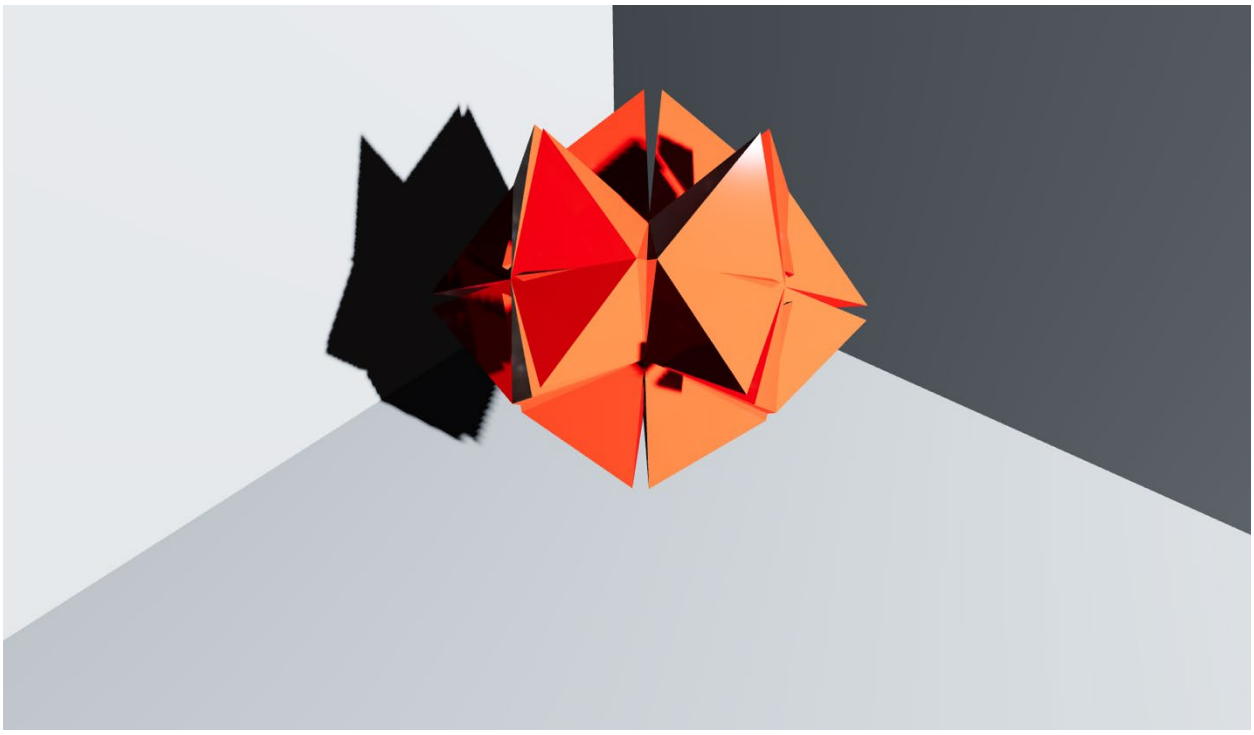


Рисунок 2.16 – Варіація Тетраідера Серпинського із пірамідами спрямованими назовні, n = 3

Задано точки, з яких будуються початкові трикутники:

const testFigure = new Pyramid(new Point(0, 0, 0), new Point(0, 0, 1), new Point(0, 1, 0), new Point(1, 0, 0));

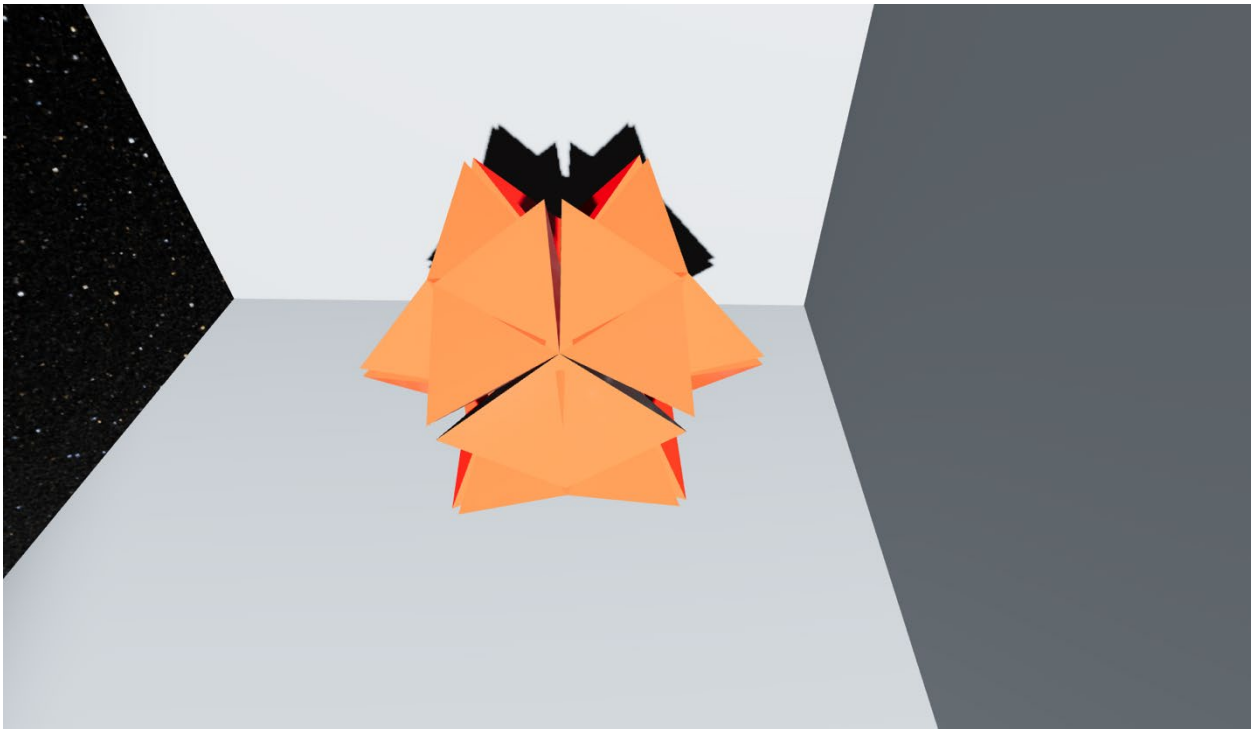


Рисунок 2.17 – Варіація Тетраідера Серпинського із пірамідами спрямованими назовні, інший ракурс, n = 3

Задано точки для побудови правильних пірамід:

```
const testFigure = new Pyramid(
    new Point(1, -1/Math.sqrt(3), -1/Math.sqrt(6)),
    new Point(-1, -1/Math.sqrt(3), -1/Math.sqrt(6)),
    new Point(0, 2/Math.sqrt(3), -1/Math.sqrt(6)),
    new Point(0, 0, 3/Math.sqrt(6))
);
```

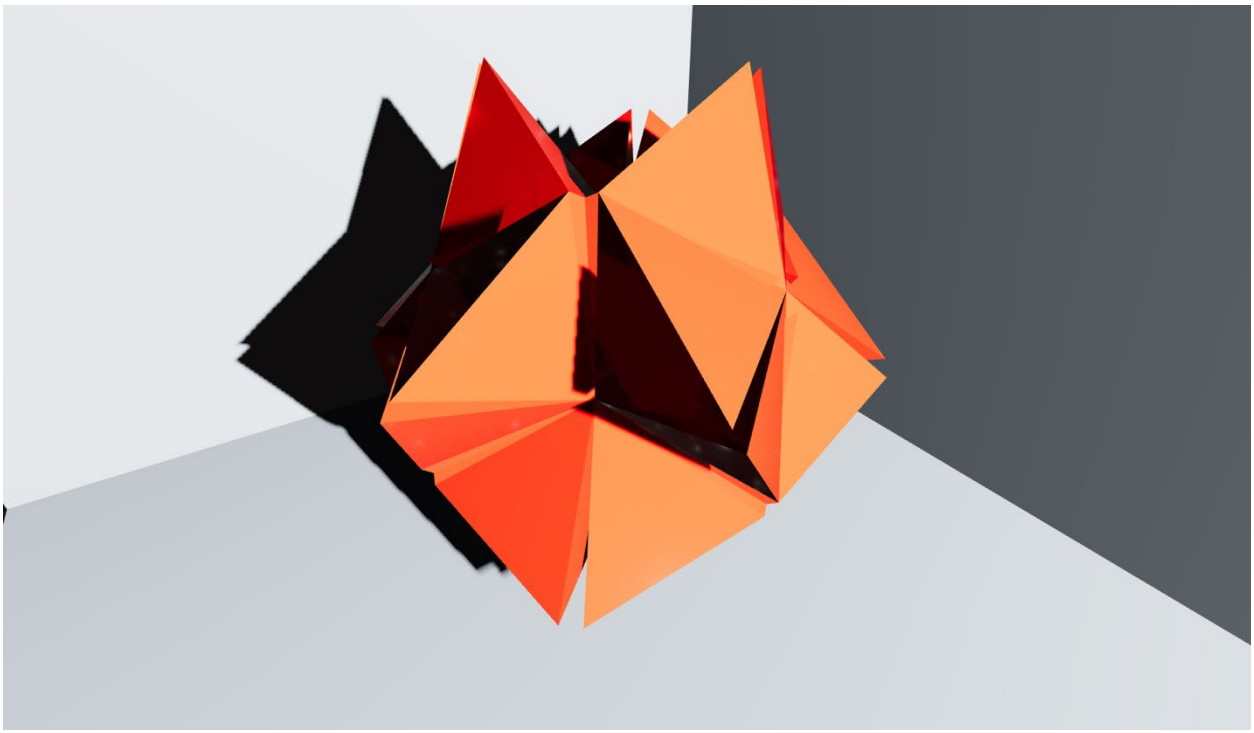


Рисунок 2.18 – Варіація Тетраідера Серпинського із пірамідами спрямованими назовні, з використанням правильних пірамід як початкових фігур, n = 3

Далі конфігурація матеріалу змінена з доданням максимальної прозорості. Результат можна побачити далі (рис. 2.19).

Можна зробити висновок, що тінь відпрацьовує некоректно. Майже

повністю прозорий матеріал не повинен відбивати повноцінну насичену тінь.

Спочатку експерементальним шляхом, а потім і знайшовши детальну інформацію було з’ясовано, що в обраному двіжку (як і в аналогах) тіні не реагують на прозорість матеріалу.

(https://github.com/mrdoob/three.js/issues/10600).

Було вирішено додати цю функціональність у рушій власноруч шляхом програмування шейдерів (автори рушію надають таку можливіть, хоч і не без обмежень).

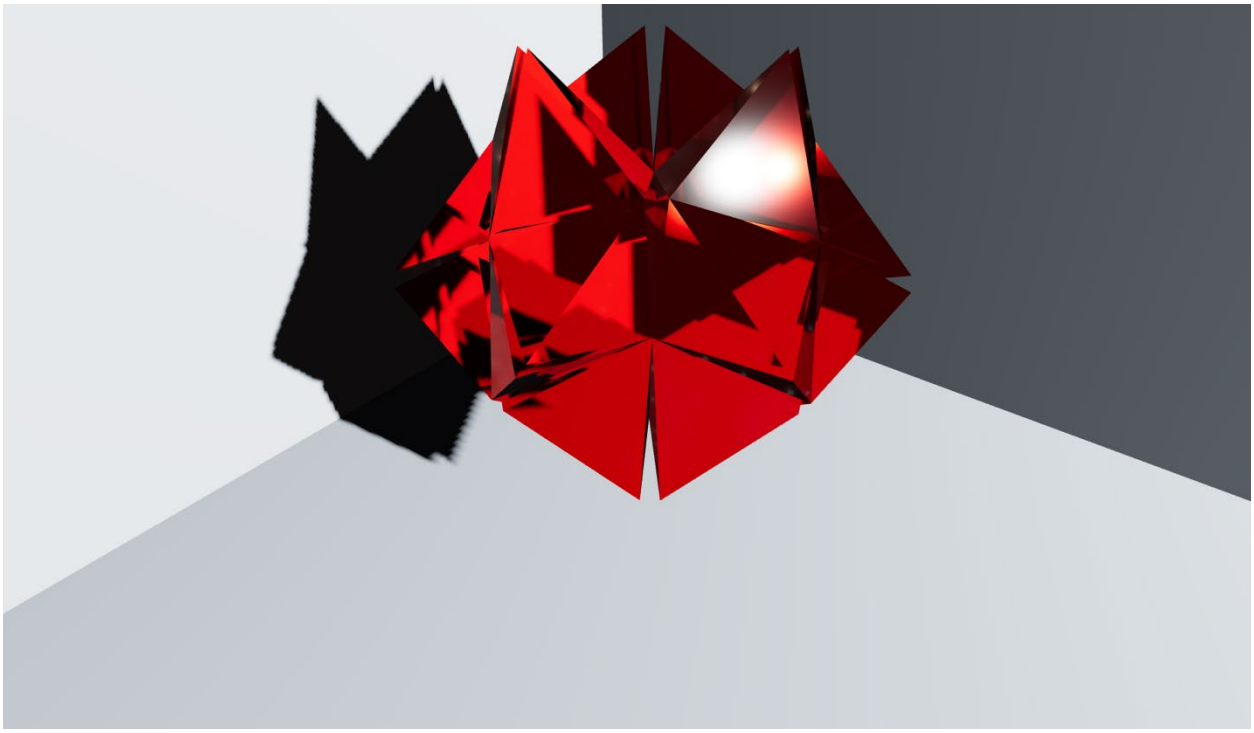


Рисунок 2.19 – Варіація Тетраідера Серпинського із пірамідами спрямованими назовні, з використанням правильних пірамід як початкових фігур, матеріал прозорий, n = 3

Фрагмент коду, метою якого є додання реації тіні на напівпрозорий об’єкт наведений нижче:

```
export function DitheredTransparencyShaderMixin( shader ) {
    const defineKeyword = 'ENABLE_DITHER_TRANSPARENCY';
    const newShader = cloneShader(shader, { ditherTex: { value: null } }, { [
```

```
defineKeyword ]: 1 });

        newShader.fragmentShader = `
            float bayerDither2x2( vec2 v ) {
              return mod( 3.0 * v.y + 2.0 * v.x, 4.0 );
            }

            float bayerDither4x4( vec2 v ) {
              vec2 P1 = mod( v, 2.0 );
              vec2 P2 = mod( floor( 0.5  * v ), 2.0 );
              return 4.0 * bayerDither2x2( P1 ) + bayerDither2x2( P2 );
            }
            ${newShader.fragmentShader}
        `.replace(
          /main\(\) {/,
          v => `
            ${v}
            #if ${defineKeyword}
            if( ( bayerDither4x4( floor( mod( gl_FragCoord.xy, 8.0 ) ) ) ) / 16.0 >=
opacity ) discard;
            #endif
          `
        );
        return newShader;
    }
```

Прозорість виставлена на 50%. Результат наведено нижче (рис. 2.20). Тінь досить помітно посвітлішала. Якщо ж прозорість збільшити до 100%, то тінь повністю зникає (рис 2.21).

Можна заявити про досить успішне застосування алгоритму дізерінгу. Хоч цей підхід має деякі погрішності з точки зору реалістичності відкидання тіні, проте з цим результатом вже можна працювати та робити відповідні експерименти.

Рисунок 2.20 – Тінь, отримана з напівпрозорої фігури, n = 3

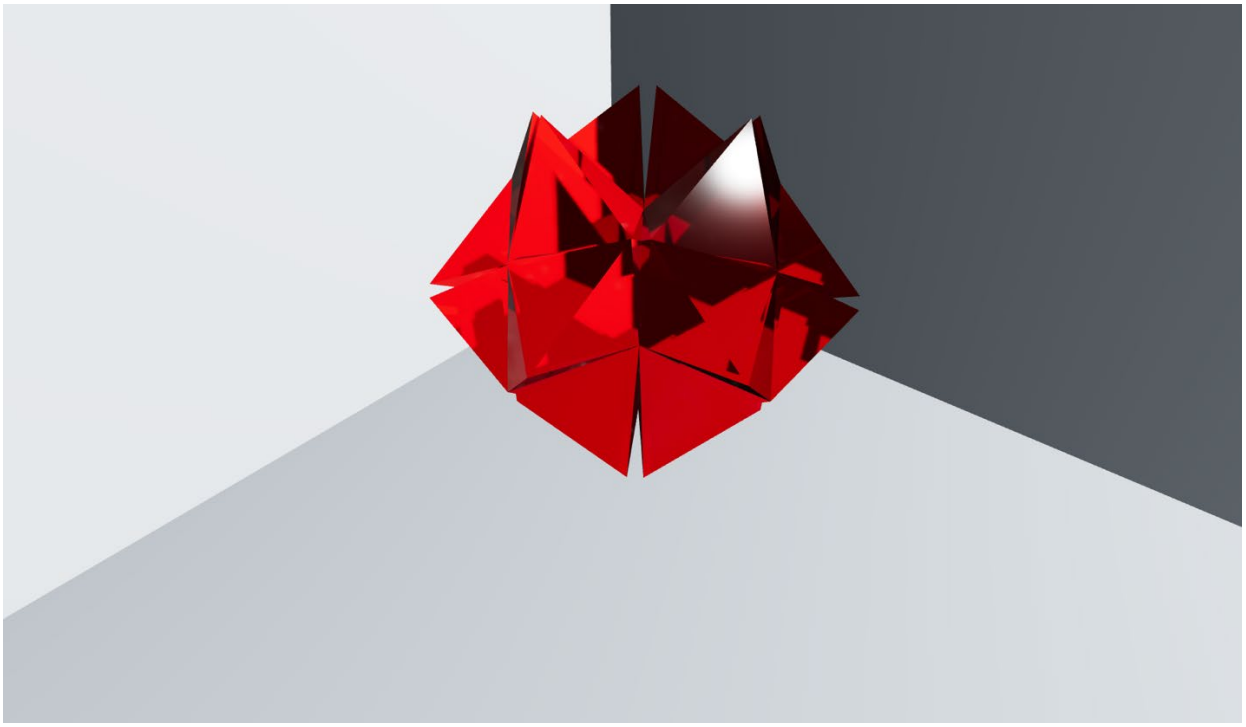


Рисунок 2.21 – Тінь, отримана з повністю прозорої фігури, n = 3

Експеремент повторено на частково освіленому фракталі (рис. 2.21).

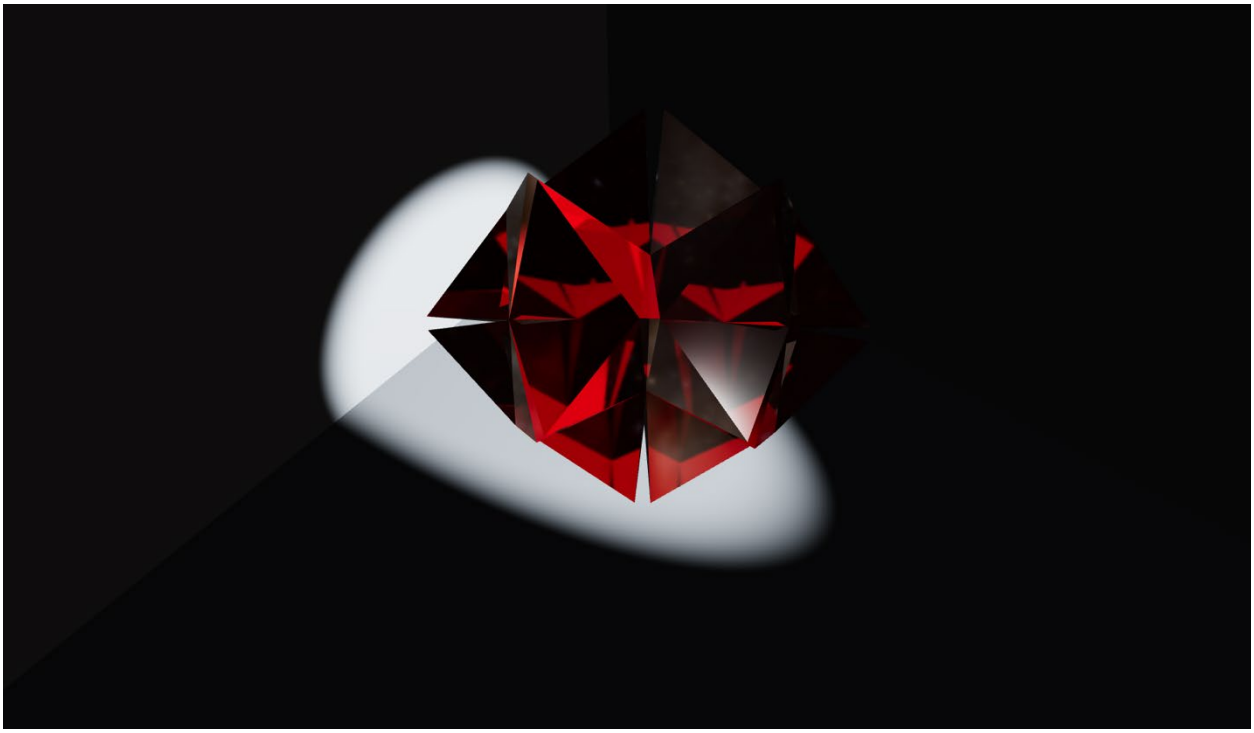


Рисунок 2.22 – Тривимірний фрактал під промінем світла із звуженим кутом

Як результат – можна побачити, що тінь зникла. Далі наведено діаграму класів, що утворюють архітектуру клієнської частини основного програмного комплексу (рис. 2.23).

Для додання можливості сконфігорувати модель фракталу, в програмний комплекс додано підготовчи кроки.

Наразі замість миттєвої генерації сцени з моделлю фракталу, обратного за здалегідь прописаних конфігураціях, користувач матиме можливість обрати: тип фракталу та кількість ітерацій. Наразі, максимальне значення кількості ітеарцій – вісім, бо чим більше ітерацій – тим більше навантаження на обчислювальний пристрій, і в даному випадку, для достатньої кількості кадрів в секунду та комфортного розглядання графічної сцени, це значення є максимально сприйнятним.

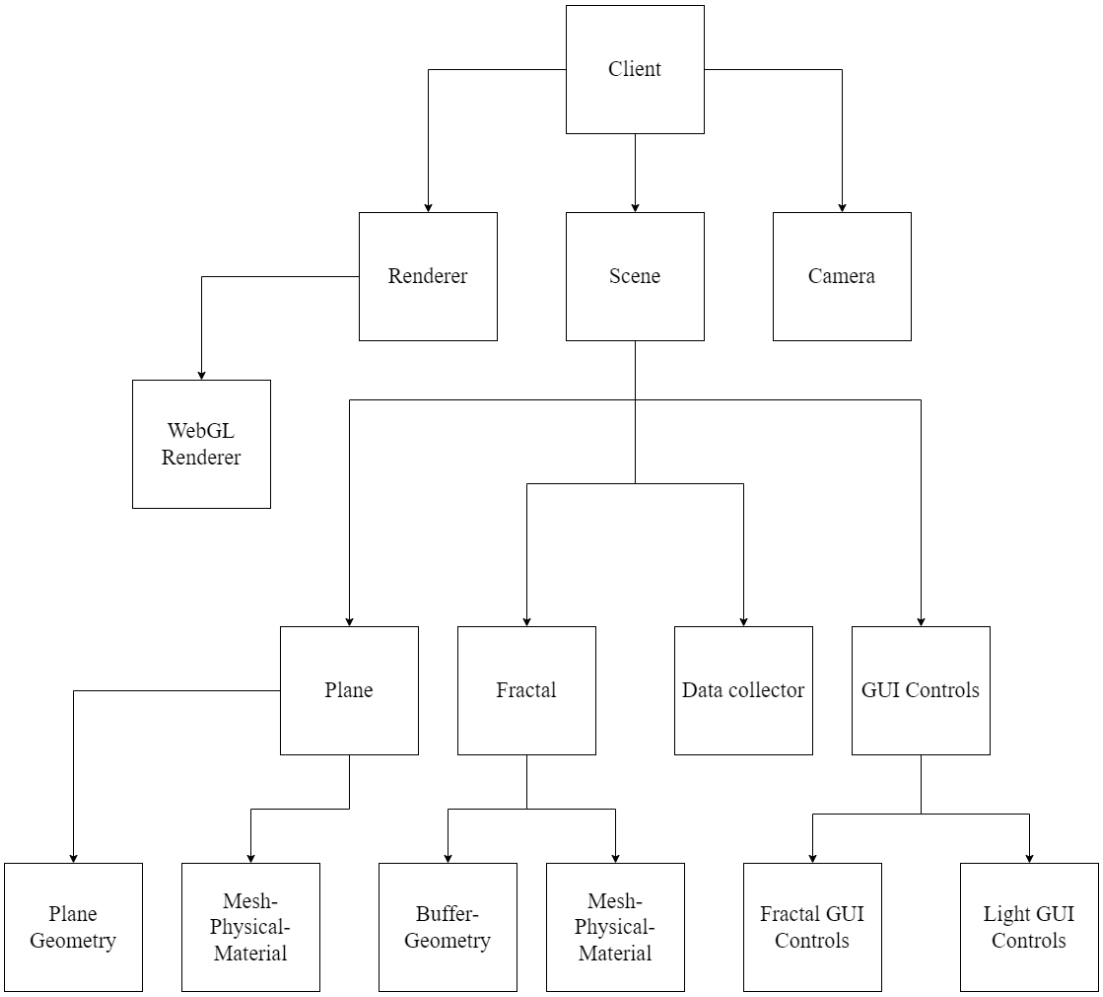


Рисунок 2.23 – Діаграма класів клієнської частини клієнт-серверного додатку

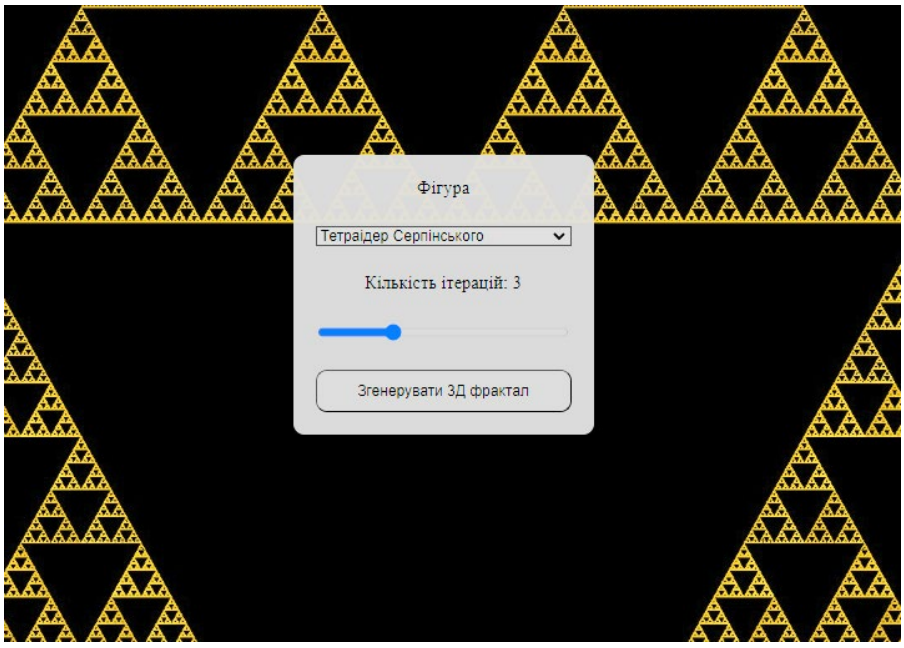


Рисунок 2.24 – Інтерфейс форми для початкової конфігурації фракталу

2.2 Розробка програмного комплексу збору показників прозорості фраталу на згенерованій тривимірній сцені та автоматизація процесу збору застосованих конфігурацій

Наразі існує можливість сконфігурувати сцену з моделлю фракталу, генерувати цю сцену та накопичувати результати у пам’яті браузера: зображення моделі фракталу та конфігурацію, відповідно.

Для того, щоб зберігати вищевказані дані та подальше їх обробляти, створено ще один додаток. Додаток «збірник метрик» так само побудований за клієнт-серверною архітектурою.

Як можна побачити далі (рис. 2.25), у новому програмному комплексі, сервер взаємодіє ордазу з клієнтом із освновного програмного комплексу, наведеного вище та своїм власним клієнтом.

Основний комплекс відправляє дані по сгенерованій сцені, та які конфігурації були обрані. Клієнт програми-збірщику метрик безпосередньо дає можливість обрати зображення, виділити область для розрахунку характеристик кольорів зображення та відправити розраховані показники прозорості на сервер.

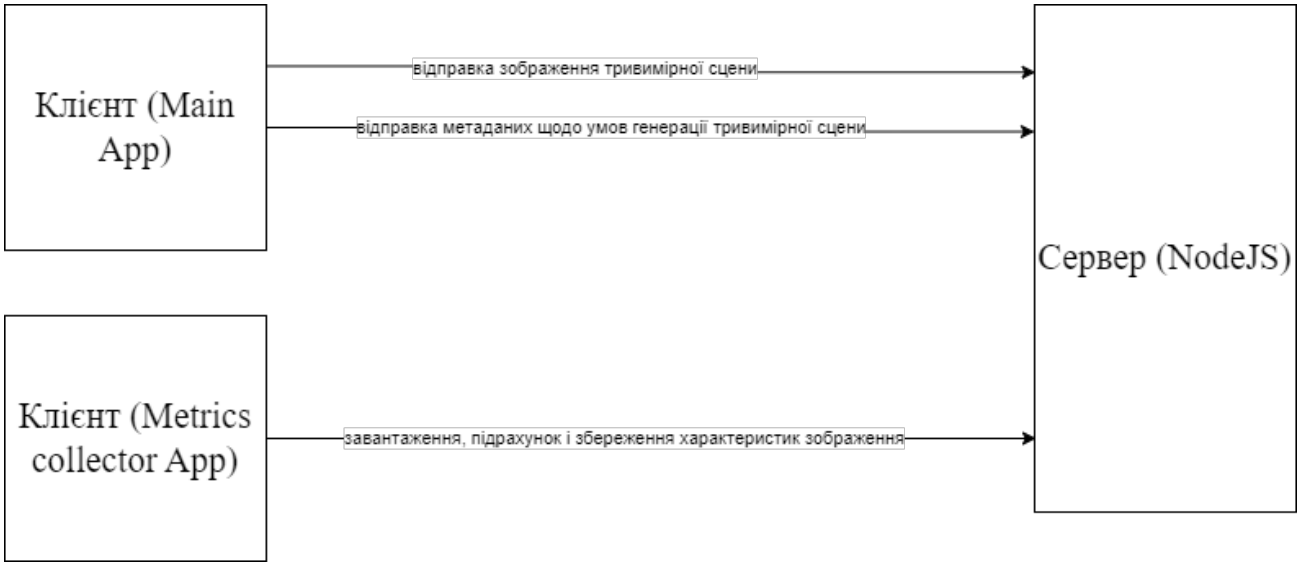


Рисунок 2.25 – Діаграма взаємодії компонентів основного програмного комплексу та додатку для збору метрик зображення

Однією з функцій сервера є можливість приймати зображення, відправлене з клієнта основної програми (генерування моделі фракталу) та відповідних конфігурацій, що для цього використовувалися. Сервер написаний мові JavaScript, для рантайму використаний NodeJS.

Сервер очікує запитів на завантаження зображення та мета-даних та коли отримує такий запит, то створює папку, використовуючи поточну дату та час та зберігає в цю папку зображення + конфігурації, при який воно було згенеровано.

Нижче наведен формат спілкування клієнту основного комплексу та серверу аналітичного комплексу (рис 2.26).

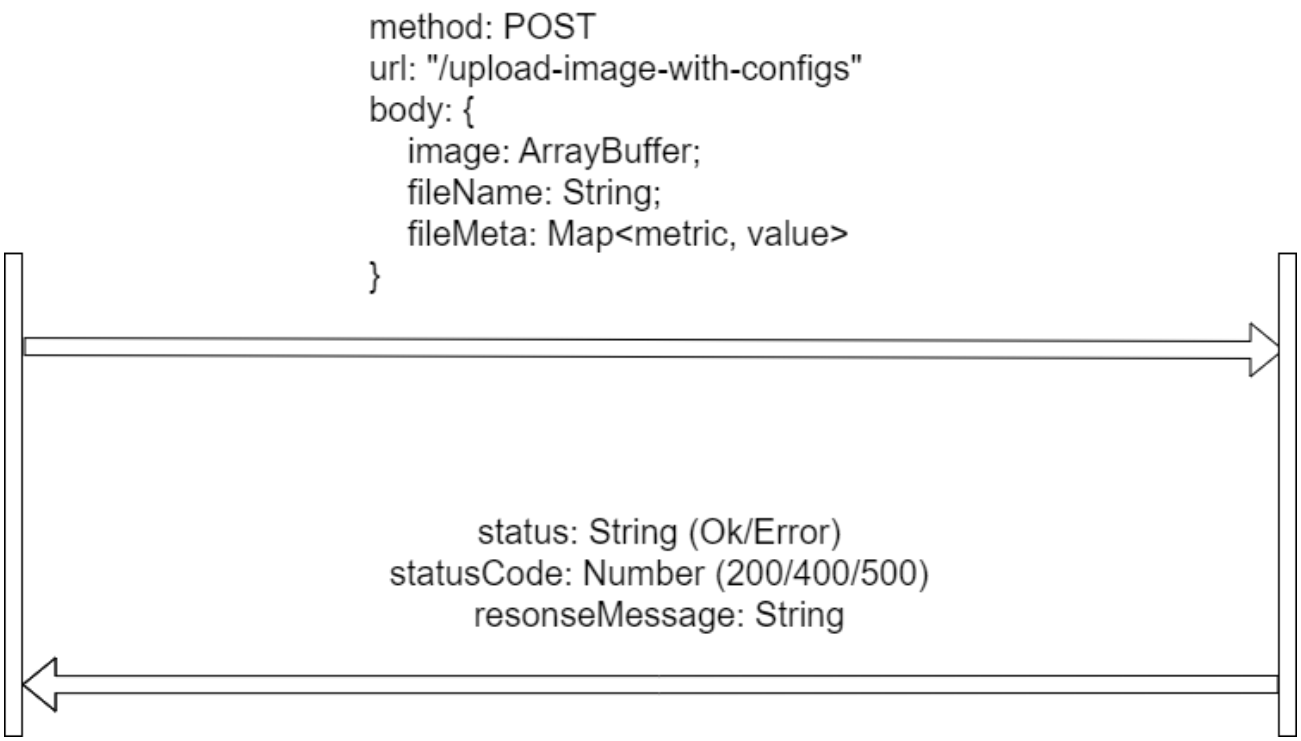


Рисунок 2.26 – Схема спілкування клієнту основного комплексу та серверу аналітичного комплексу

Далі зображена схема комунікації всередині самого аналітичного комплексу між клієнтом та сервером (рис. 2.27).

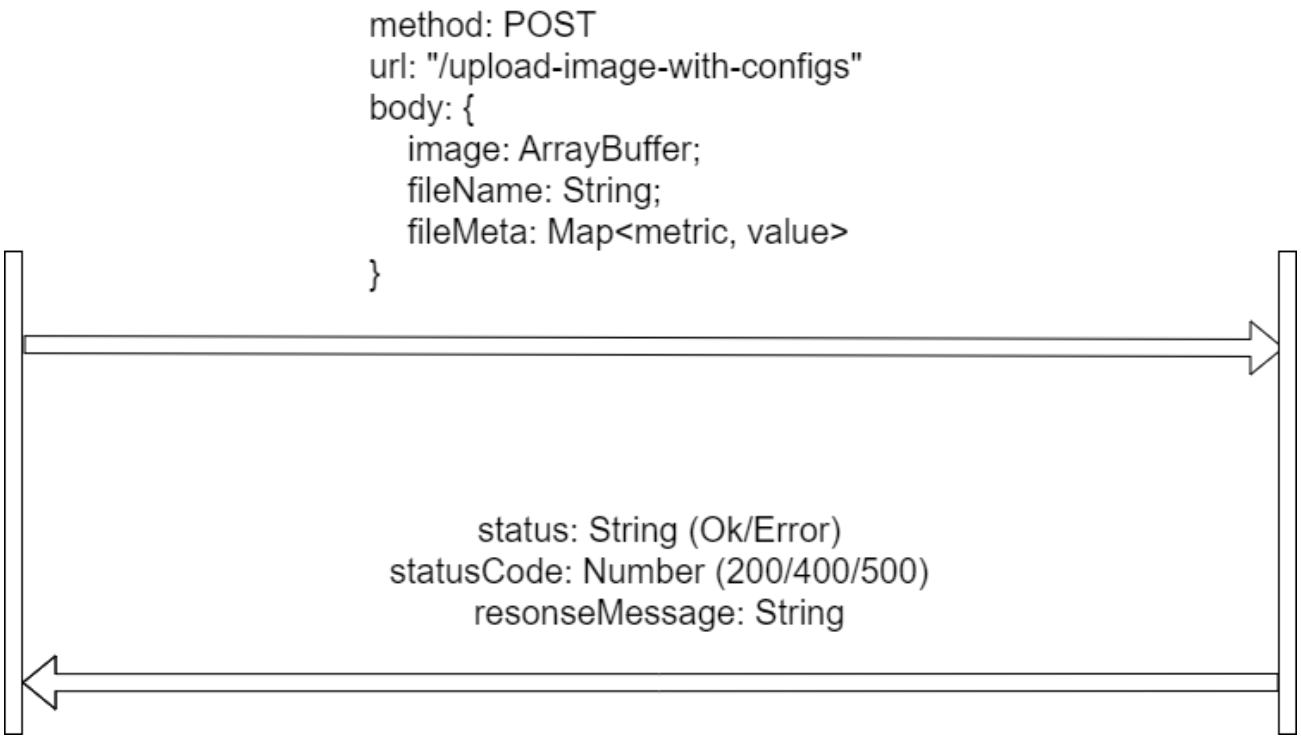


Рисунок 2.27 – Схема комунікації всередині самого аналітичного комплексу між клієнтом та сервером

Сервер програмного комплексу з аналізу зображення запускається на локальному комп’ютері за портом 3000 (рис. 2.28).

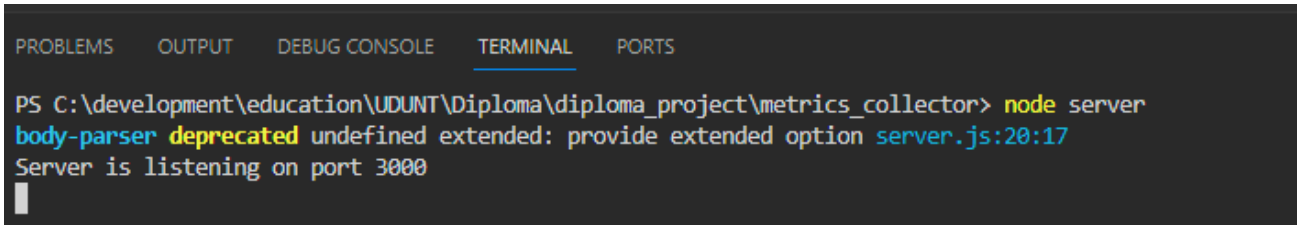


Рисунок 2.28 – Старт локального серверу програми-аналізатора зображень

Інтерфейс клієнта доступний, якщо відвідати локальний хост з цим портом у браузері (рис 2.29).

Рисунок 2.29 – Інтерфейс програми-аналізатора зображень

Для того, щоб виконати повний цикл роботи із зображенням вручну, потрібно виконати наступні дії:

1) запустити сервер основної програми;

2) запустити сервер програми аналізатора;

3) зайти на клієнта основної програми та отримати згенеровану сцену з тривимірним фракталом;

4) перейти на клієнта програми-аналізатора зображень;

5) завантажити попередньо згенероване зображення та виділити область для аналізу;

6) натиснути «порахувати метрики» щоб переглянути їх та «зберегти дані» для відправлення даних на сервер.

Детально процес використання програмного комплексу буде наведено у розділі з дослідженнями, проте важливо було показати перелік кроків, які є

загальними, бо кожен з них, у свою чергу, вимагає від користувача ще додаткову послідовність дій для отримання результату, наприклад, для того, щоб згенерувати фрактал треба вибрати фігуру, кількість ітерацій, характеристики матеріалу, поворот фігури за осями та інше – це цілком можливо при аналізі даних поштучних генерацій, але поставлена задача також вимагає показати картину у більш загальному вигляді, що вимагає генерацію сцени з фракталом в різних варіаціях фігури, кількості ітерацій, повороту фігури, характеристик матеріалу, освітлення, тощо – це значна кількість повторень, що займе дуже значний час. Причому, в разі необхідності перепроходження експерименту, витрачення часу лінійно збільшиться.

Для того, щоб прискорити подальший аналіз, було прийнято рішення про написання програми для автоматизації певного переліку дій, які будуть симулювати дії реального користувача, але виконуватимуться програмним скриптом.

У нагоді стала технолгія, яка розповсюджена для автоматизації процесу тестування програмного забезпечення. Це окрема галузь сучасного світу розробки програмного забезпечення, з різноманіттям існуючих рішень, що звісно мають свої переваги і недоліки, проте було обрано найпопулярніше рішення у вибраному стеку технологій, а саме – Cypress.io.

Cypress — це інтерфейсний інструмент тестування наступного покоління, розроблений для сучасних веб-застосунків. Він вирішує ключові проблеми, з якими стикаються розробники та інженери з контролю якості під час тестування сучасних програм. Cypress найчастіше порівнюють із Selenium (що є найпопулярнішим рішенням серед аналогів), однак Cypress принципово й архітектурно відрізняється. Cypress не обмежений такими ж обмеженнями, як Selenium.

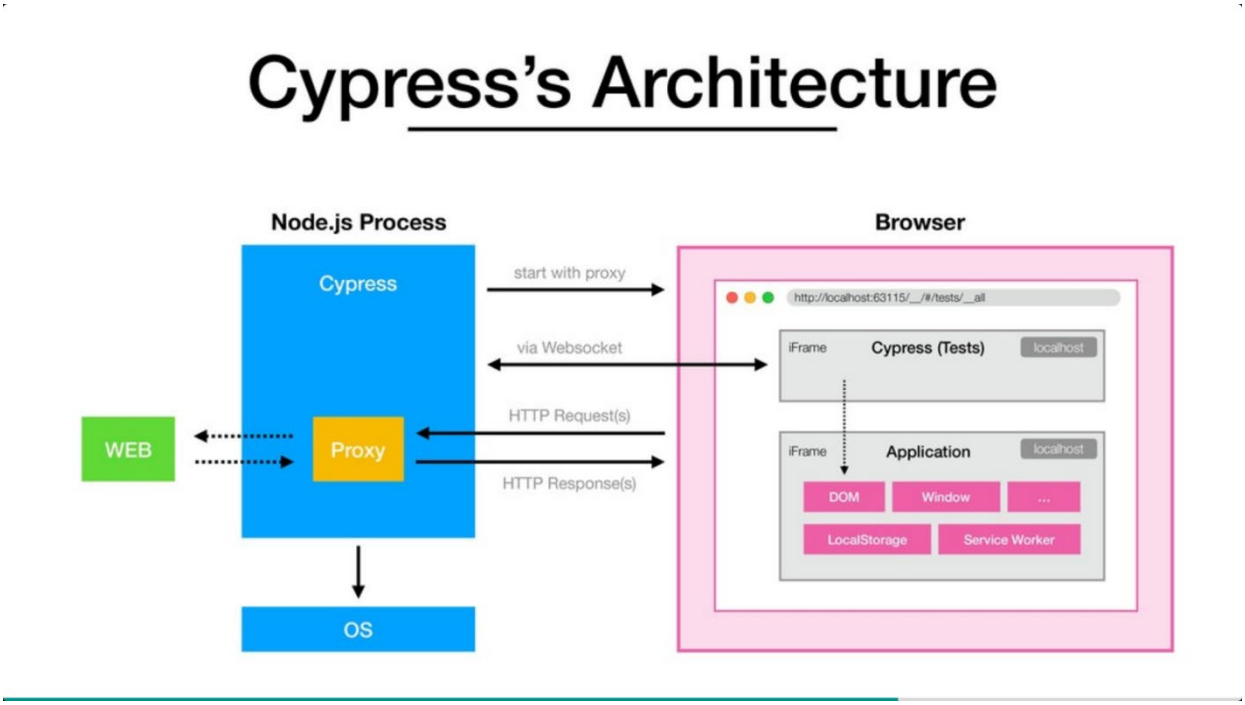


Рисунок 2.30 – Архітектура Cypress.io

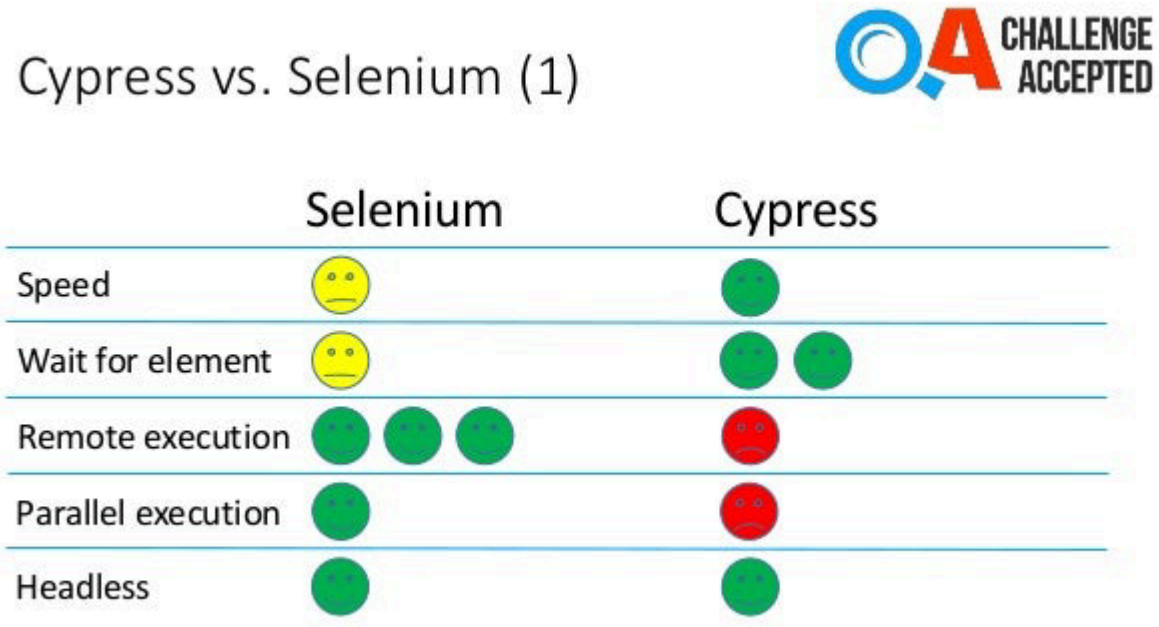


Рисунок 2.31 – Порівняльна таблиця Cypress та Selenium

Cypress дозволяє писати всі види тестів:

1) End-to-end тести;

2) Тести компонентів;

3) Інтерграційні тести;

4) Юніт тести.

Cypress може тестувати все, що працює в браузері. Цей фреймворк має код з відкритим доступом.

При стандартному сценарії, програмний комплекс покривається тестами і чим вищий рівень абстракції та погляд на систему, тим менший відсоток таких тестів передбачається у процесі автоматизації (рис. 2.31). Причин цьому багато, але наразі достатньо зазначити, що окрім тестів, цікавий момент саме у можливостях цієї технологій.

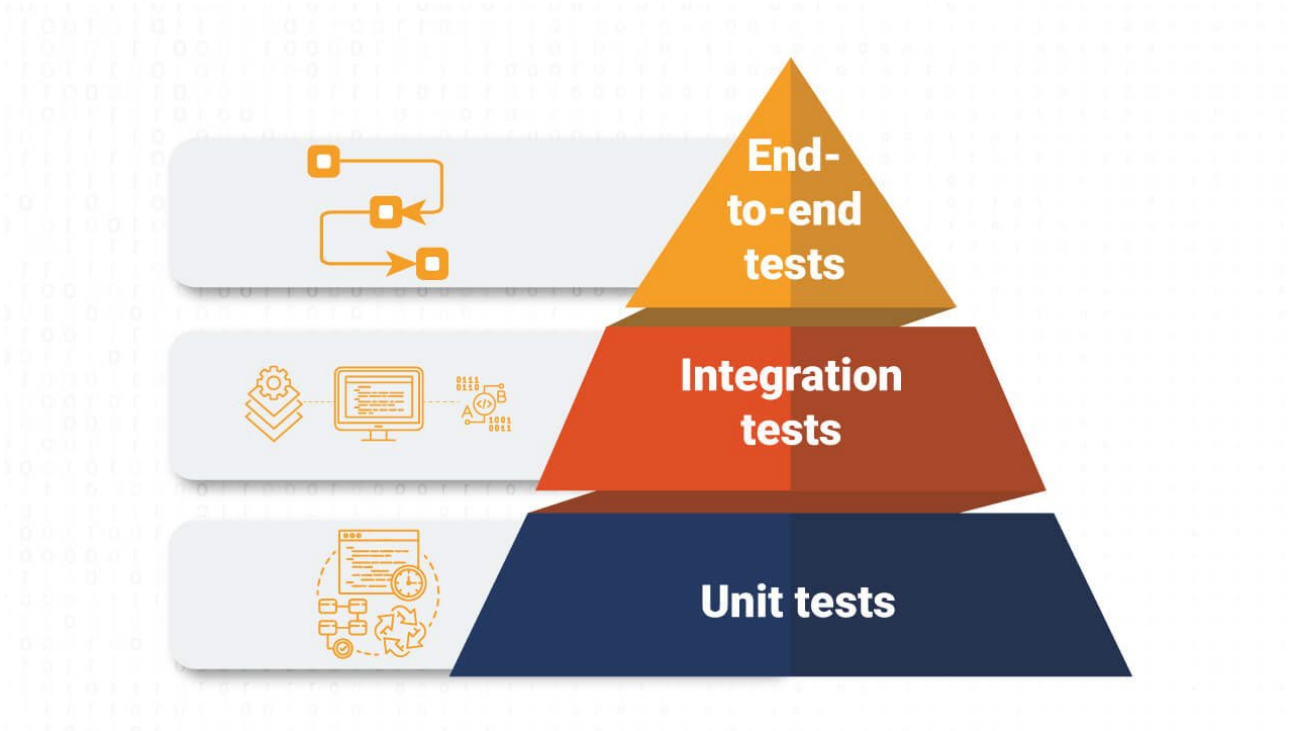


Рисунок 2.32 – Піраміда тестування із різними видами тестів

Слід ще раз зазначити, що задача, для якої застосовано Cypress полягає не в перевірці коректності функціонування ПЗ, а у виконанні певної

послідовності дій, що у реальному випадку зазвичай є «передумови» до виконання тесту. Далі буде розлянуто самі тестові сценарії.

Допустимо, що є три типи фрактальних фігур, кожна утримує власні правила підстановки при підвищенні ступеня розвитку.

Допустимо, що є кілька кутів повороту фігури, в даному випадку поворот буде симетрично за всіма осями (x, y, z), одиниця виміру – радіани.

Допустимо, що є показник як ступінь розвитку (кількість виконуваних ітерацій) – в даному випадку допустима кількість ітерацій від однієї до восьми.

Допустимо, що є показник прозорості матеріалу фракталу – в даному випадку це число у проміжку від 0,01 до 0,99. Отже, береться комбінація перелічених вище показників і з використанням відповідних значень цих показників, виконються наступні дії (попередньо запущені програми-сервери – програма-генератор фракталів та програма-збіршик метрик зображень):

1) відкрити сторінку основної програми (http://localhost:8080);

2) вибрати тип фрактальної фігури з випадаючого зписку;

3) вказати кількість ітерацій;

4) натиснути на кнопку «згенерувати фраткал»;

5) змінити прозорість матеріалу фракталу;

6) змінити кут повороту фігури за осями;

7) перейти на сторінку програми-збірщику метрик зображень (http://localhost:3000);

8) натиснути на кнопку «завантажити картинку» та вибрати її;

9) виділити необідну ділянку на зображенні (чи визначити її в коді за змовчанням);

10) натиснути на кнопки «порахувати метрики» та «зберегти дані».

При спробі автоматизувати дані кроки ближе до кінця процесу виникла технічна складність. А саме – завантаження картинки у браузер на кроці «вісім». У чому полягає суть проблеми – з міркувань безпеки, існують певні обмеження у доступі з програмного коду, який з браузеру намагається отримати доступ до файлової системи (рис. 2.33).

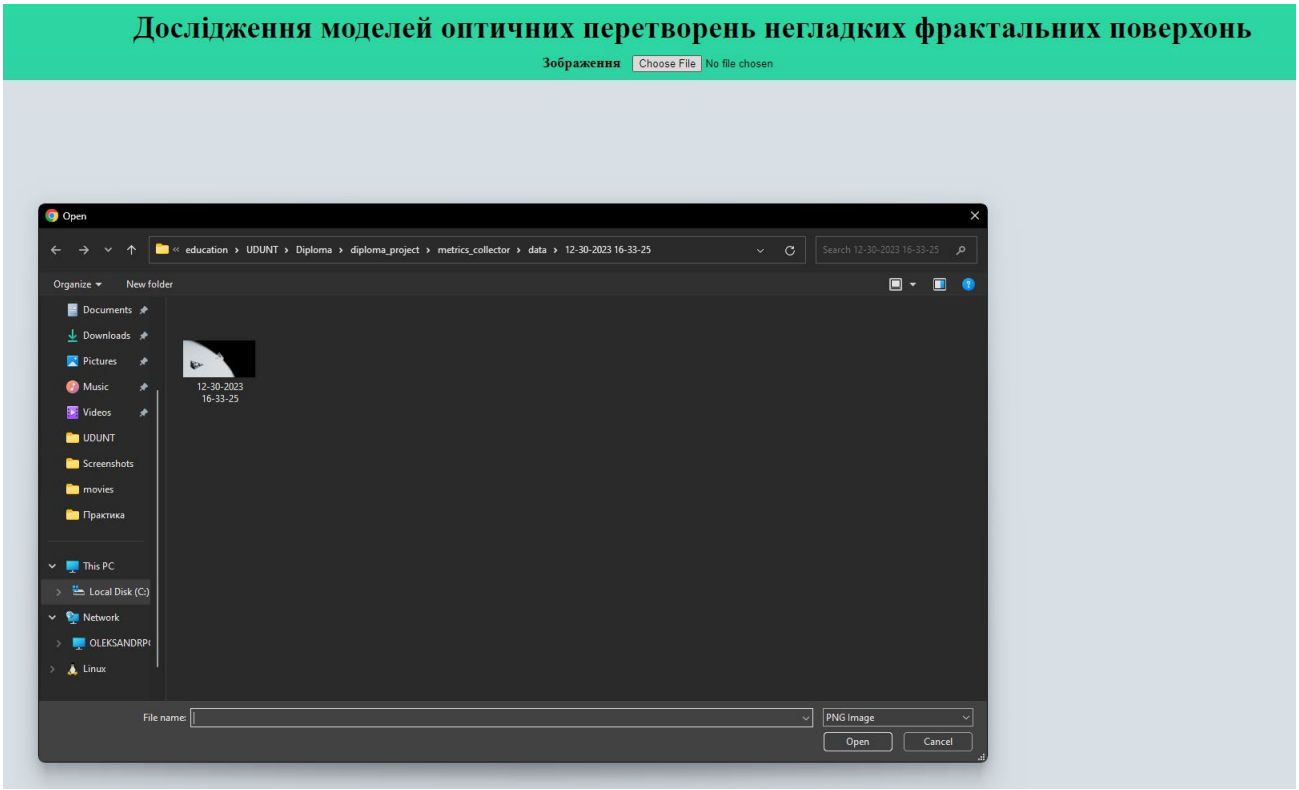


Рисунок 2.33 – Завантаження зображення у браузер

Якщо вручну це зробити не несе складностей, у автоматичному режимі це досить важко. Не можна сказати, що в автоматичному режимі заборонені всі операції з файлами, наприклад, можна зберегти файл з певним ім’ям та розширенням, проте створити директорію, або прочитати файли та їх зміст з коду, що виконується у браузері, без додаткових маніпуляцій, не є

можливим. В Cypress існує рішення для цієї проблеми – це фікстури (fixtures). Завдяки фікстурам, в тому числі, можна отримувати доступ до файлової системи завдяки попередній конфігурації, проте це рішення, нажаль не підходить до даного випадку, тому що у випадку з фікстурою, треба заздалегідь мати всю інформацію про файл – його розташування, ім’я, тощо, але в нашому випадку:

1) програмний комплекс, який генерує фрактал, зберігає дані (разом з зображенням), програмний комплекс, що збирає показники прозорості та програма автоматизації це 3 різні програми;

2) назви директорій та файлів генеруються автоматично в залежності від дати та часу, тобто у контексті Cypress тестів, назви файлів не відомі заздалегідь – відповідно, немає можливості скористатися фікстурою.

Слід відзначити, що при деяких архітектурних змінах, можна було б вирішити цю задачу без обхідних шляхів, проте автоматизація, яка додавалась в програмний комплекс, необхідна не для тестування, а саме для економії часу на виконання рутинних операцій для експерементів в перспективі.

Отже є не доцільним вносити серйозні зміни в програмні комплекси з точки зору часових затрат. Було обрано інший шлях. Замість того, щоб використовувти клієнтський інтерфейс програми-збірщику метрик, було обрано комунікувати з основної програми-генератора фракталів одразу з серверної програмою-збірщиком метрик, минуючи клієнта, з додаванням відповідної логіки з приводу використання зображення, виділення ділянки зображення та безпосередньо відправки цих даних. Вирішено, що для поставленої задачі з автоматизації, достаньо попередньо заданої ділянки з місцезнаходженням фракталу та тіні на площині під ним.

Після успішної спроби виконати послідовність дій для можливих комбінацій генерації фракталів, стало очевидним, що отримані дані є некоректними, оскрільки не залежно від прозорості, кількості ітерацій тощо – середій колір, мінімальний колір не відрізніються від аналогічних показників між експерементами. Проблема полягала в тому, що відбувався попіксельний перебір всієї вибраної ділянки (рис. 2.34).

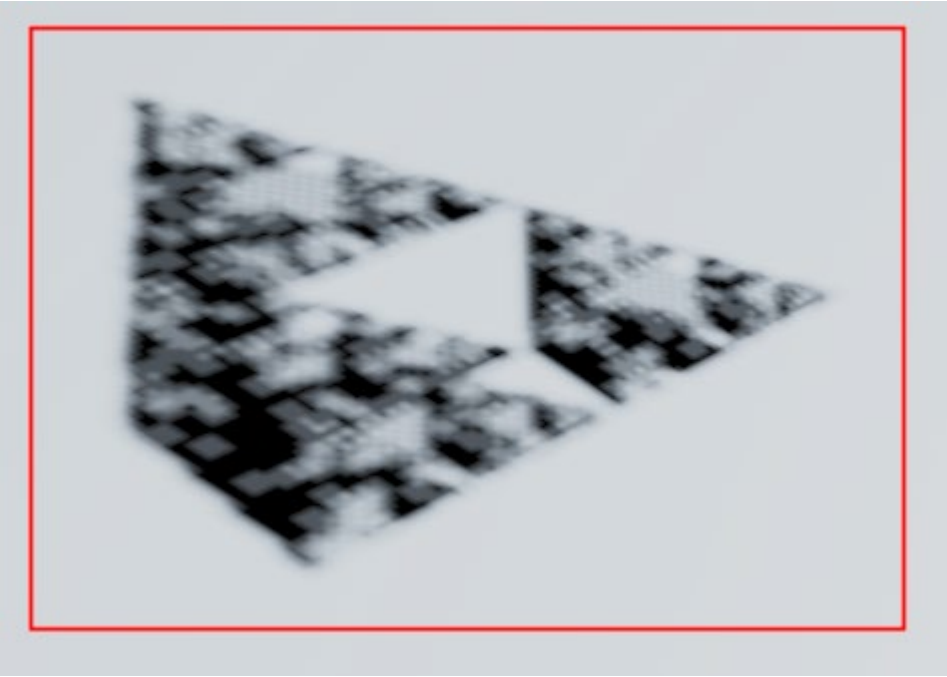


Рисунок 2.34 – Тінь від напівпрозорого фракталу Тетраідера Серпинського

Суть проблеми полягала в тому, що не достатньо просто окреслити чотирикутником ділянку для прорахування кольорових характеристик, бо фон має досить велику частку у співвідношенні з тінню, що відповідно впливає на загальні показники. Тому, було обрано в кожному ряді пікселів визначати де початок, а де кінець фігури, щоб враховувати фон тільки в межах тіні (всередині неї). Проте, цей підхід теж накладав певне обмеження – наприклад, у випадку з майже прозорим фракталом не буде можливості відокремити «потрібний» фон від «зайвого», тобто той фон, який входить у контур тіні.

Тому, додані попередні кроки – спочатку кожного експеримента вибираєтсья нульова прозорість матеріалу фракталу, дані контуру цього зображення рохраховуються та кешуються, щоб далі на них можна було спиратися як на «еталон» з точки зору розміщення фігури фракталу. Цей підхід виявився успішним рішенням поставленої задачі, що дало змогу отримувати коректні метрики зображень тіні фракталу.

Висновки до другого розділу

Треба зазначити, що обраний технологічний стек та їх інтеграція у програмний продукт на практиці підтвердили правильність висновків щодо доцільності обраних рішень у якості бази для створеного програмного комплексу. Подібні рішення дозволяють підтримувати досить високі темпи розробки, а легка інтеграція з іншими рішеннями, дозволяє коструювати досить складні системи, які складаються з багатьох компонентів, кожен з яких відіграє свою важливу роль.

Проте, також слід зазначити, що WebGL має деякі недоліки у порівнянні з OpenGL та Vulkan. При перегляді готових проектів з використанням перелічених API, виглядає перевага швидкодії останніх у порівнянні з WebGL. Це має досить значний вплив на розрахункову спроможність обчислювальних машин, які не є максимально потужними, бо зібрані не з сучасних комплектуючих. Наприклад, коли треба виконати велику кількість ітерацій для відображення фракталу з високим ступенем розвитку, то в нашому випадку, є можливим обрати максимум 8 ітерацій. Також, скоріше за все, можна було б помітити значні відмінності у візуальному вигляді моделей та освітленні.

У порівнянні веб рушіїв з десктопними, в останніх обчислювальні можливості системи використовуються у більш високій мірі, тому що самі технології, на яких вони розроблені, мають більшу сумісність на системному рівні. Якщо взяти, наприклад, Unreal Engine 5, який має сумісність з OpenGL та Vulkan API, то він максимально утилізує ресурси обчислювальної машини, а завдяки ручному керуванню пам’яттю, це приносить додаткові можливості для оптимізацій в плані використання пам’яті та швидкодії в цілому. Проте, слід зазначити, що за перевагами завжди йдуть і недоліки, в цьому випадку це більш повільні темпи розробки.

В будь-якому разі, проведена робота дає гарну можливість візуалізувати фрактали у тривимірному просторі та дослідити їх оптичні можилвості на достатньому рівні, а додаткові програми, такі як програма-змірщик метрик та програма-автоматизатор виконання експериментів, в свою чергу додають продуктивності у проведенні досліджень та дозволяють використати вигранний час на низку суміжних операцій, що слугують більшому розкриттю теми: наприклад, є можливість перепроходити експерментальні сценарії значну кількість разів, використовуючи різні конфігурації системи і краще бачити залежність одиного показника від іншого.

3 ДОСЛІДЖЕННЯ ВПЛИВУ ФІГУРИ ФРАКТАЛУ, СТУПЕНЮ ЙОГО РОЗВИТКУ, ПРОЗОРОСТІ ТА ПОВОРОТУ НА ОТРИМАНУ ТІНЬ

3.1 Аналіз показників прозорості фракталу

Для проведення досліджень існує необхідність генерувати сцену з моделлю фракталу з різним набором параметрів. Для того, щоб проаналізувати оптичні властивості фракталу, було вирішено сфокусуватися на тіні, отриманій на площі під фракталом, після спрямування джерела освітлення в бік фігури.

Відповідно до логіки вище можна завантажувати зображення у браузер, робити виділення деякої ділянки та виконати збір метрик кольорів заданої ділянки з конвертацією з RGB у Gray Scale.

Для конвертації використовуються такі формули, як:

Lightness method

$$\mathbf{grayscale} = \frac{\mathbf{min(R,G,B)} + \mathbf{max(R,G,B)}}{\mathbf{2}}$$

Цей метод, полягає в тому, щоб взяти середнє значення компонентів з найвищим і найнижчим значенням. Ми можемо легко побачити, що цей метод має дуже серйозну слабкість, оскільки не використовується один компонент RGB. Це, безумовно, проблема, тому що кількість світла, яку сприймає наше око, залежить від усіх трьох основних кольорів.

Average method

$$\mathbf{grayscale} = \frac{\mathbf{R+G+B}}{\mathbf{3}}$$

Як можна побачити, цей метод полягає у простому середньому значенні. Хоч ми зараз і беремо до уваги всі компоненти, цей метод також проблематичний, оскільки він призначає однакову вагу кожному компоненту. На основі досліджень людського зору ми знаємо, що наші очі реагують на кожен колір по-різному. Зокрема, наші очі більш чутливі до зеленого, потім до

червоного і, нарешті, до синього. Отже, ваги у наведеному вище рівнянні мають змінитися.

Luminosity Method

Найкращим методом є метод світності, який успішно вирішує проблеми попередніх методів.

На основі вищезазначених спостережень ми повинні взяти середнє зважене значення компонентів. Внесок синього кольору в кінцеве значення повинен зменшитися, а внесок зеленого – збільшитися. Після деяких експериментів і більш глибокого аналізу дослідники дійшли висновку в наступному рівнянні:

$$\mathbf{grayscale} = \mathbf{0.3} * \mathbf{R} + \mathbf{0.59} * \mathbf{G} + \mathbf{0.11} * \mathbf{B}$$

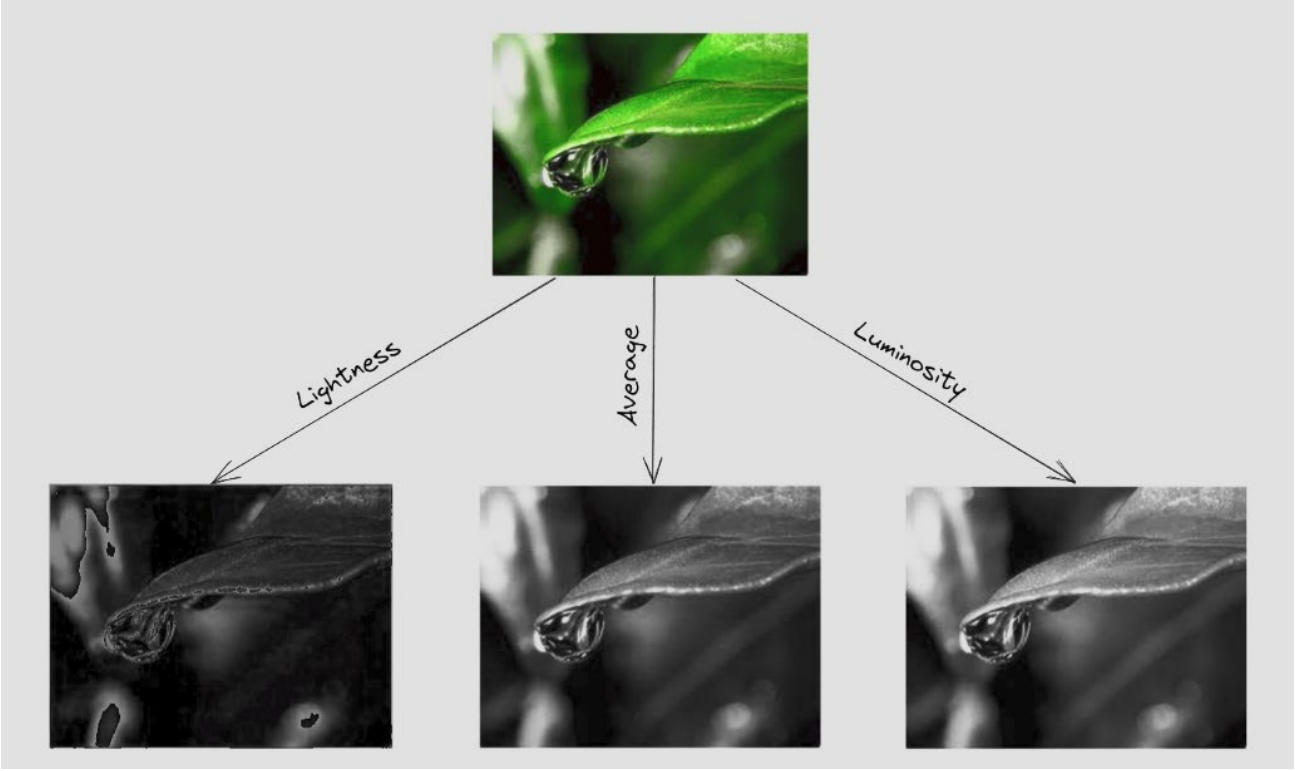


Рисунок 3.1 – Порівняння результатів роботи різних алгоритмів конвертації GRB-зображення у GrayScale

Для розрахування метрик та збереження даних використані дані функції на

стороні сервера.

Сценарій використання програмного комплексу наведений нижче (рис. 3.2). Запуск серверу основного додатку: **npm run dev**

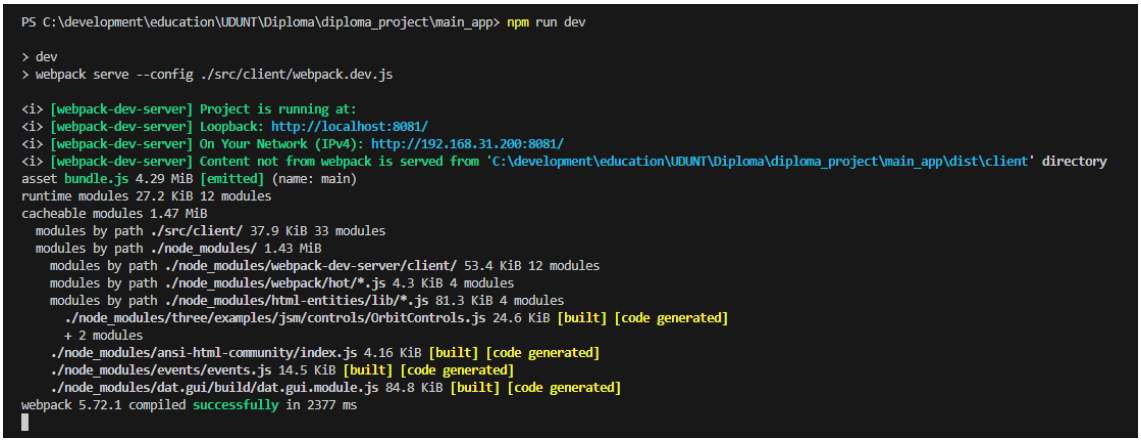


Рисунок 3.2 – Запуск серверу основного програмного комплексу для генерації фракталів

Далі, відбувється перехід на посилання, де хоститься клієнтська частина (наразі це локальний ПК, знаходиться за посиланням http://localhost:8081), обирається конфігурація та генерується модель фракталу (рис. 3.3).

Процес генерації досить швидкий і подальші конфігурації можливі вже після отримання початкової сцени з моделлю фракталу. Це дає можливість в режимі реального часу відстежувати візуальні зміни та мати можливість більш прицільно обирати ті чи інші показники матеріалу фігури, розташування, освітлення, масштабу та інше.

Загалом, можливості стосовно налаштування інтерфейсу досить зручне як в плані його програмування, так і в плані користування, є багато опцій стосовно інтеграції нових показників, або налаштування щодо користування.

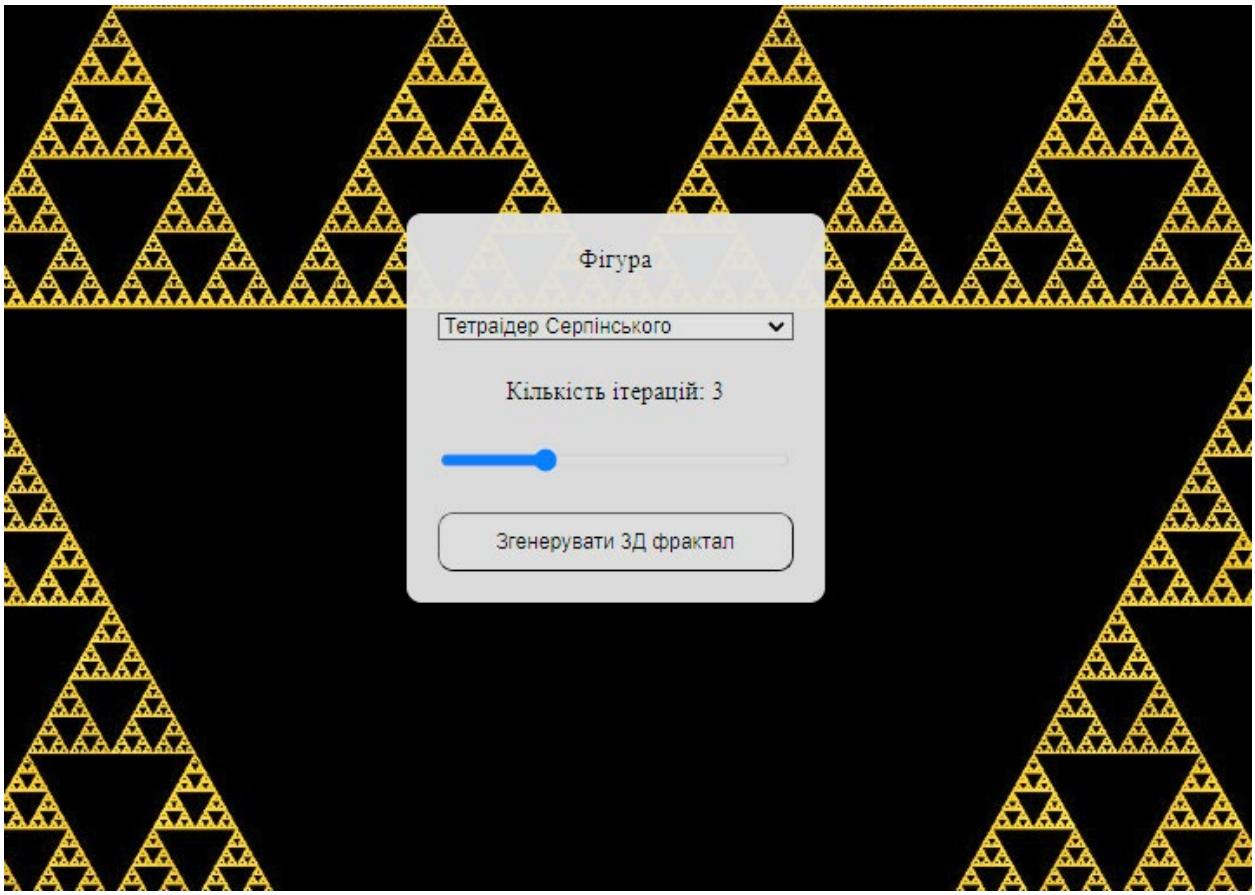


Рисунок 3.3 – Стартовий інтерфейс клієнської частини основного програмного комплексу генерації фракталів з вибором початкових конфігурацій

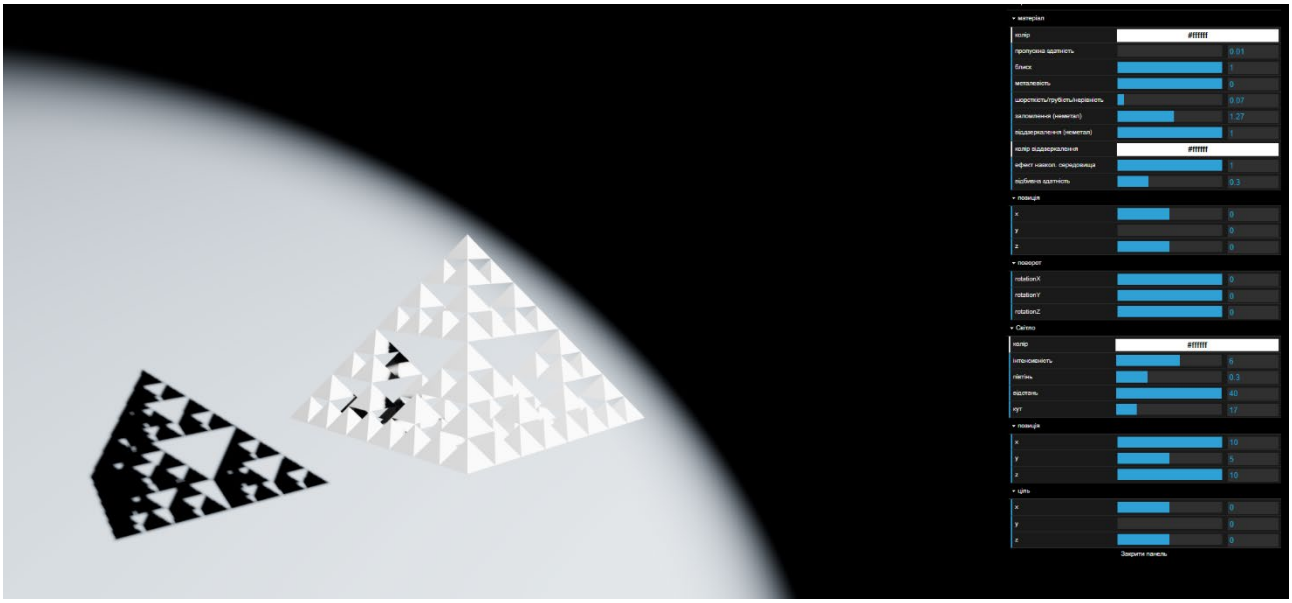


Рисунок 3.4 – Інтерфейс отриманої згенерованої тривимірної сцени з фракталом

Зверху виставляються необхідні налаштування та зберігається зображення разом з обраними налаштуваннями (нижня ліва кнопка):

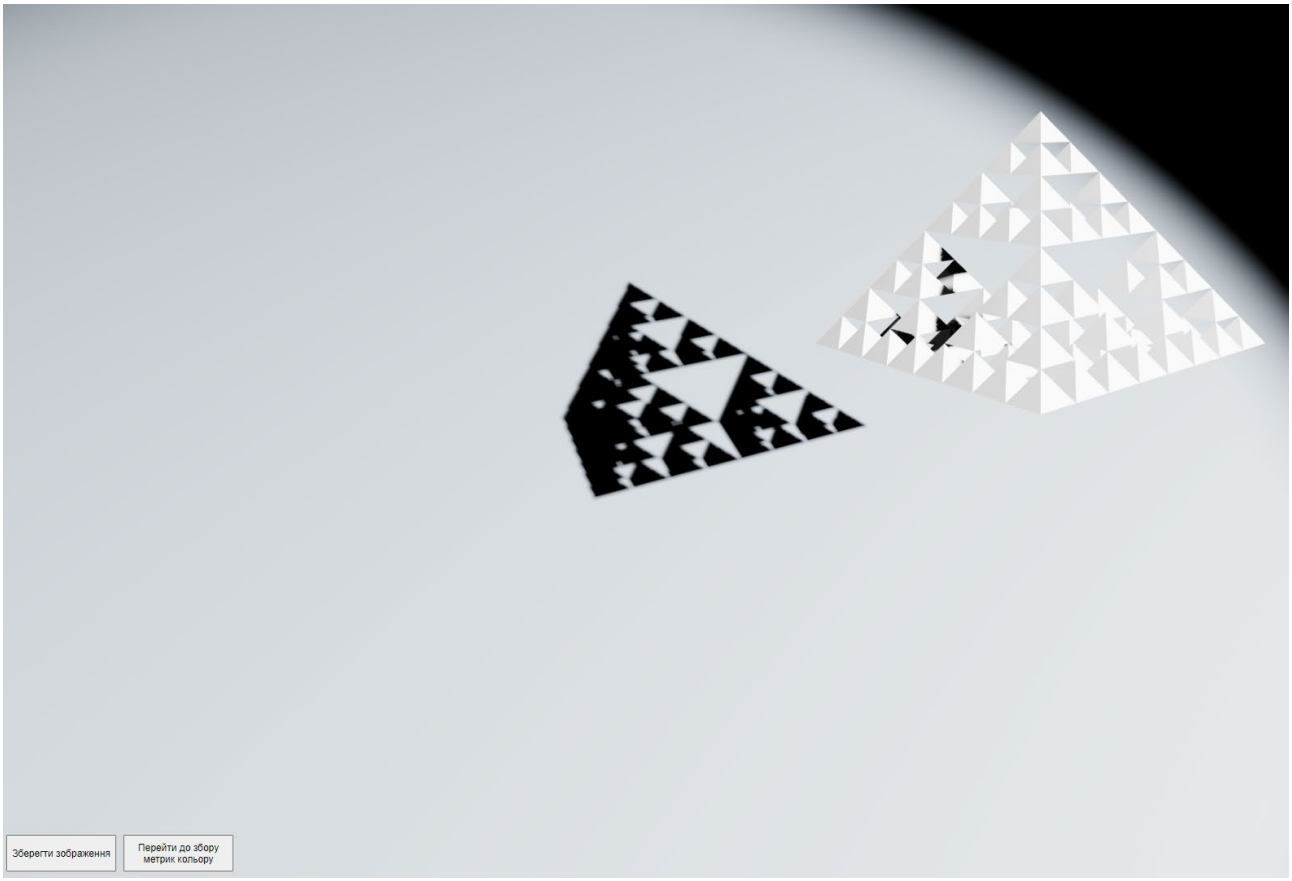


Рисунок 3.4 – Процес збереження згенерованого зображення

Після натискання кнопок «Зберегти зображення» та «Перейти до збору метрик кольору», відкривається нова вкладка браузера, же можна побачити інтерфейс програми-аналізатора зображень. Тут необхідно вибрати файл для аналізу (рис. 3.5).

Далі обирається область, для якої треба буде виконати розрахунок метрик кольорів (рис. 3.6).

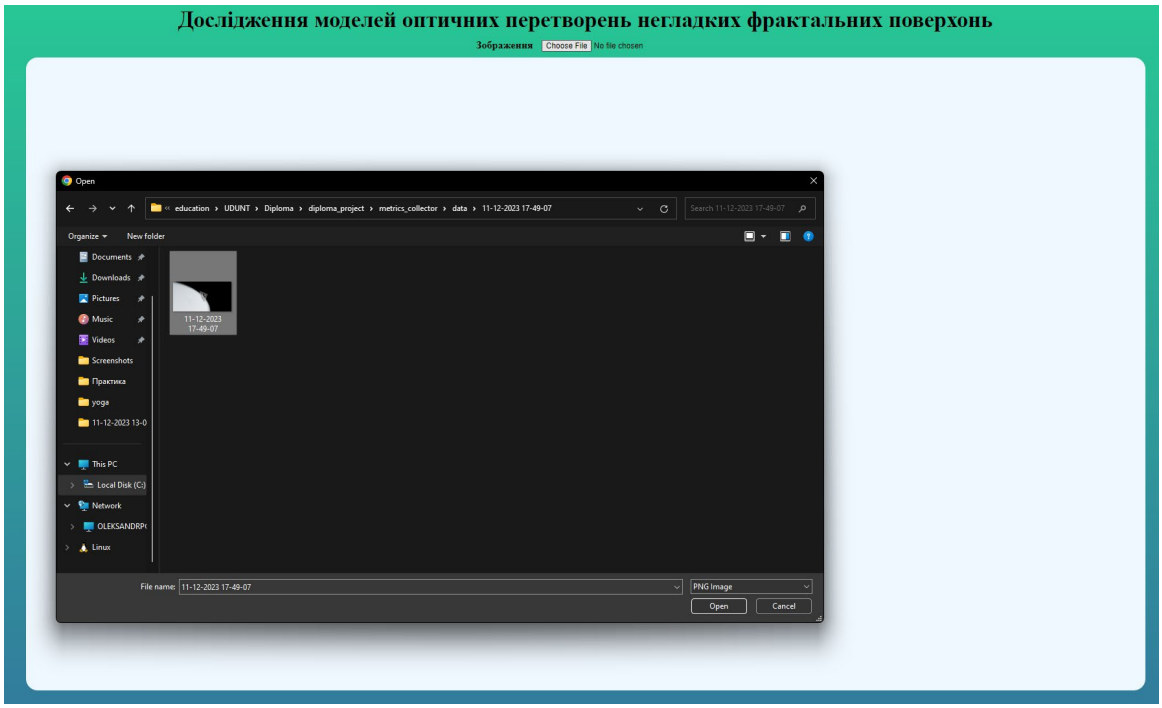


Рисунок 3.5 – Процес відкриття згенерованого зображення у програмі-аналізаторі

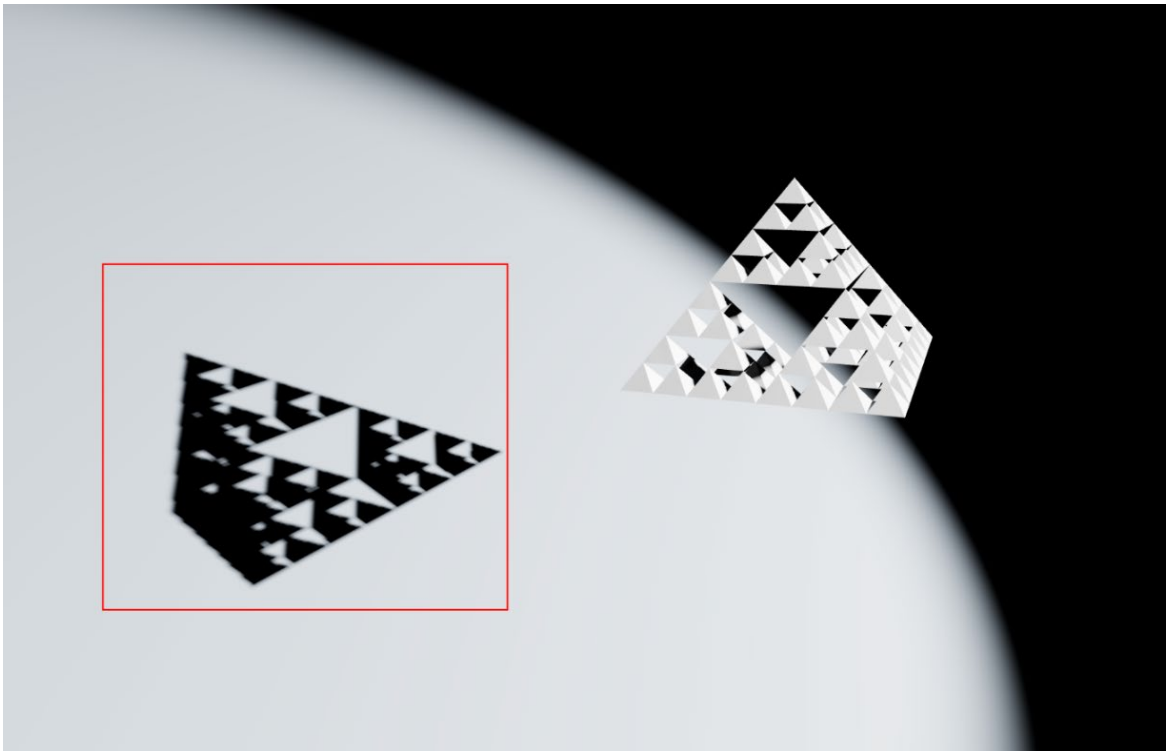


Рисунок 3.6 – Процес виділення області для виконання розрахунків характеристик кольору

У верхньому правому куті необхідно натиснути кнопки «Порахувати метрики» та «Зберегти дані». Для того, щоб переглянути збережені файли треба зайти у додаток для розрахування метрик, в папку data:

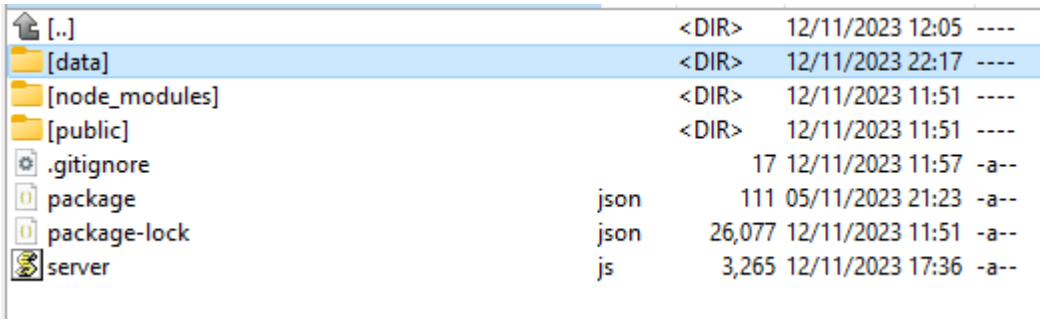


Рисунок 3.7 – Файлова система із згенерованими метримками

Директорія з поточною датою та часом містить необхідні файли (рис. 3.8

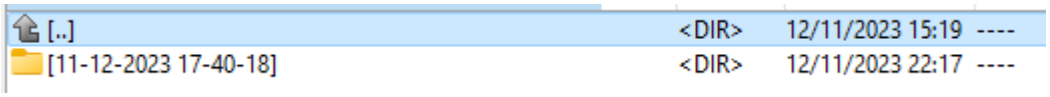


Рисунок 3.8 – Формат створення директорій

Вміст директорії «data» має такий вигляд (рис 3.9).

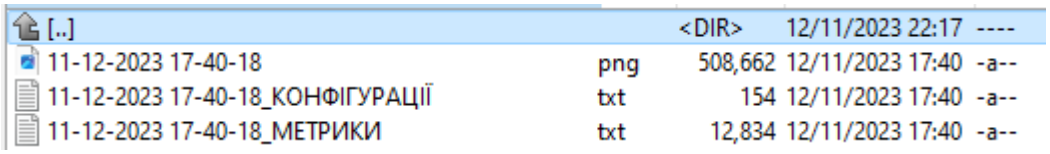


Рисунок 3.9 – Вміст директорії та формат збережених файлів

Відповідно, зображення з розширенням .png, txt-файл з конфігараціями при яких згенеровано модель фракталу та txt-файл з метриками кольорів вибраної ділянки зображення.

Приклад файлу з конфігураціями можна переглянути нижче (рис 3.10).

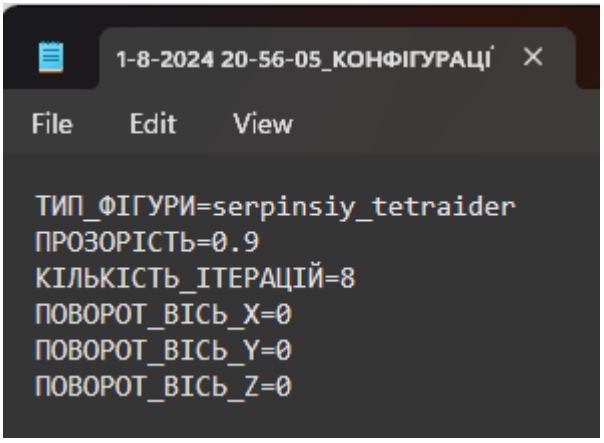


Рисунок 3.10 – Вміст файлу із збереженими конфігураціями

Далі можна переглянути приклад файлу з метриками (рис 3.11).

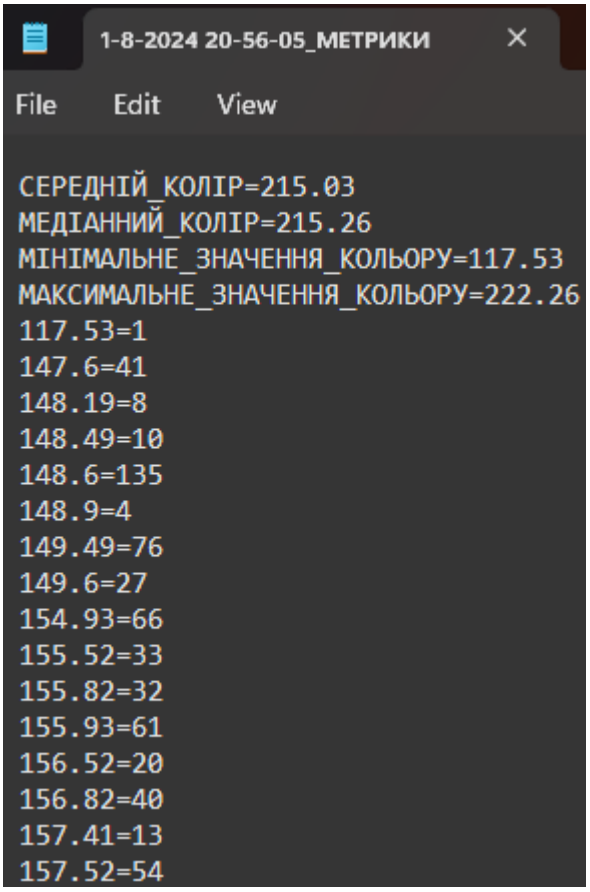


Рисунок 3.11 – Вміст файлу із розрахованими показниками кольорів

Наразі, в наявності є необхідні числові характеристики для візуалізації

отриманих даних у вигляді діаграм.

Треба виконати наступне дослідження: побудувати сімейство графіків, яке буде визначатися поворотом фігури. Сам графік повинен допомогти візуалізувати залежність кількості пікселів від GrayScale кольору.

Побудова графіків досягнута засобами програмного комплексу Excel:

Фігура – Тетраідер Серпинського. Перша група генерацій буде відбуватися за умов:

- прозорість - 0.01
- кількість ітерацій - 3
- поворот за осями X, Y та Z – не застосовується

Рисунок 3.12 – Кількість пікселів відповідного кольору в залежності від кута повороту фракталу. Фігура - Тетраідер Серпинського, без повороту

Застосовані конфігурації:

- прозорість - 0.5

- кількість ітерацій - 3
- поворот за осями X, Y та Z - 1 rad

Наразі більш цікавлять більш високі значення, тому на графіку відфільтровано значення менше ніж 200. Тобто, під фільтр попали більш насичені, темні, кольори. Зроблено ацент саме на ділянках, де колір має кількісну перевагу та дає більш характерний для експерименту результат. Результати можна побачити далі (рис. 3.13 – 3.15).

Рисунок 3.13 – Кількість пікселів відповідного кольору в залежності від кута повороту фракталу. Фігура - Тетраідер Серпинського, поворот 1 rad

Застосовані конфігурації:

- прозорість - 0.1
- кількість ітерацій - 3
- поворот за осями X, Y та Z – 1.6 rad

Як можна побачити, з поворотом впаво, скопичення зсуваються вправо та зменшуються за віссю X. Фігура приймає більш горизонтальне положення і це відповідним чином впливає на тінь.

Можна зробити висновок, що при зміні кута повороту отримано досить характерні тенденції щодо змін у графіку, тобто у разі потреби, навіть можна приблизно прогнозувати, як зміниться графік, базуючись на попередніх даних.

Рисунок 3.14 – Кількість пікселів відповідного кольору в залежності від кута повороту фракталу. Фігура - Тетраідер Серпинського, поворот 1.6 rad

В наступному експерименті буде розглянуто фрактал з більш високим ступенем розвитку, із застосуванням таких самих кутів повороту фігури (рис. 3.15). На цей раз цікавить трохи більший діапазон графіку по осі X, бо навідміну від попередніх результатів, тут відмінності проявляються починаючи з менших значень, що досить цікаво.

Друга група генерацій буде відбуватися за умов:

- прозорість - 0.01
- кількість ітерацій – 6

Слід відначии менший розброс, з хвилястими смугами на нижніх значеннях кольорів та досить кучне розташування на верхніх значеннях, де ми бачимо світлу тінь.

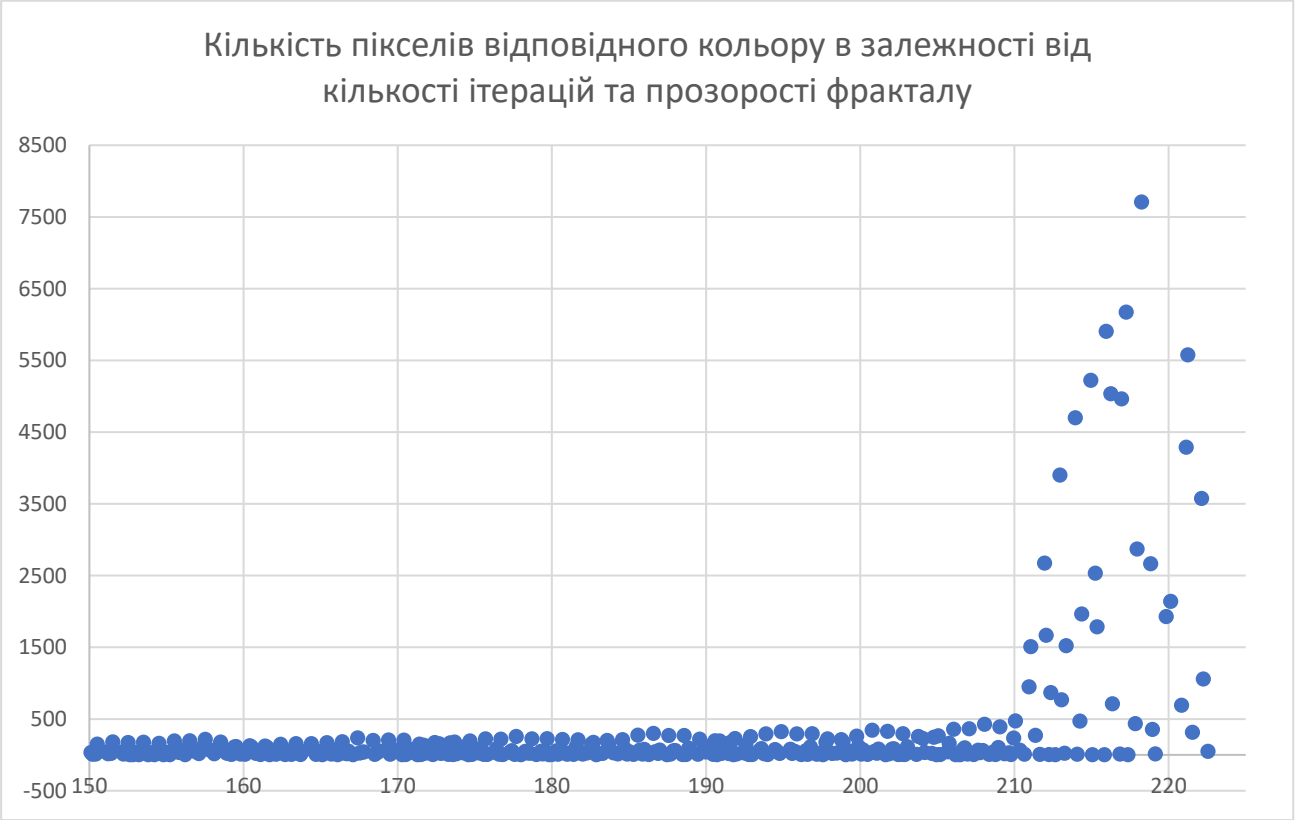


Рисунок 3.15 – Кількість пікселів відповідного кольору в залежності від кута повороту фракталу. Фігура - Тетраідер Серпинського, без повороту

Застосовані конфігурації:

- прозорість - 0.01
- кількість ітерацій - 6

Як можна побачити, поведінка на нижніх значенях, так і на високих значеннях, має схожий характер зростання у випадку без застосування повороту фігури та з поворотом за осями на 1 rad. Притому, при повороті

на 1.6 rad вже видно різке падіння показників, але при цьому більшу їх скопиченість.

Рисунок 3.16 – Кількість пікселів відповідного кольору в залежності від кута повороту фракталу. Фігура - Тетраідер Серпинського, поворот 1 rad

Рисунок 3.17 – Кількість пікселів відповідного кольору в залежності від кута повороту фракталу. Фігура - Тетраідер Серпинського, поворот 1.6 rad

Вище було переглянута залежність отриманої тіні від фрактального об’єкту без зміни прозорості матеріалу, з поворотом навколо осів.

Далі пропонується розглянути поведінку напівпрозорої фігури та тіні від

неї, при зміненні ступня розвитку фракталу (рис 3. 18 – 3.20).

Застосовано конфігурації:

- прозорість - 0.5
- кількість ітерацій – 2

Рисунок 3.18 – Залежність кількості пікселів від GrayScale кольору та ступеню розвитку фракталу, n = 2

Застосовано конфігурації

- прозорість - 0.5
- кількість ітерацій – 4

Рисунок 3.19 – Залежність кількості пікселів від GrayScale кольору та ступеню розвитку фракталу, n = 4

Застосовано конфігурації

- прозорість - 0.5
- кількість ітерацій – 6

Можна спостерігати яскраво виражений всплеск на проміжку 180 по осі X (рис. 3.18), але у подальших генераціях картина міняється і значення більш рівномірно розташовуються по графіку.

Варто відмітити, що не дивлячись на зміни у проміжку між 210 та 220, тобто зміну світлих кольорів, вони є дуже незначні, із тенденцію до підвищення кількісної характеристики пікселів.

Рисунок 3.20 – Залежність кількості пікселів від GrayScale кольору та ступеню розвитку фракталу, n = 6

Пропонується для повної картини розглянути поведінку тіні з дуже прозорим матеріалом фракталу (90% прозорості), при цьому фрактали будуть мати велику різницю у ступені розвитку.

Застосовано конфігурації

- прозорість - 0.9
- кількість ітерацій – 1, 4, 8

Як можна побачити далі, загалом кількість пікселів значно менша, бо це висока прозорість і в цілому відтінки будуть дуже наближені до фону (в нашому випадку, площині під фрактальною фігурою), що в будь-якому випадку, досить очікувано.

Рисунок 3.21 – Залежність кількості пікселів від GrayScale кольору та ступеню розвитку фракталу, n = 1

Рисунок 3.22 – Залежність кількості пікселів від GrayScale кольору та ступеню розвитку фракталу, n = 4

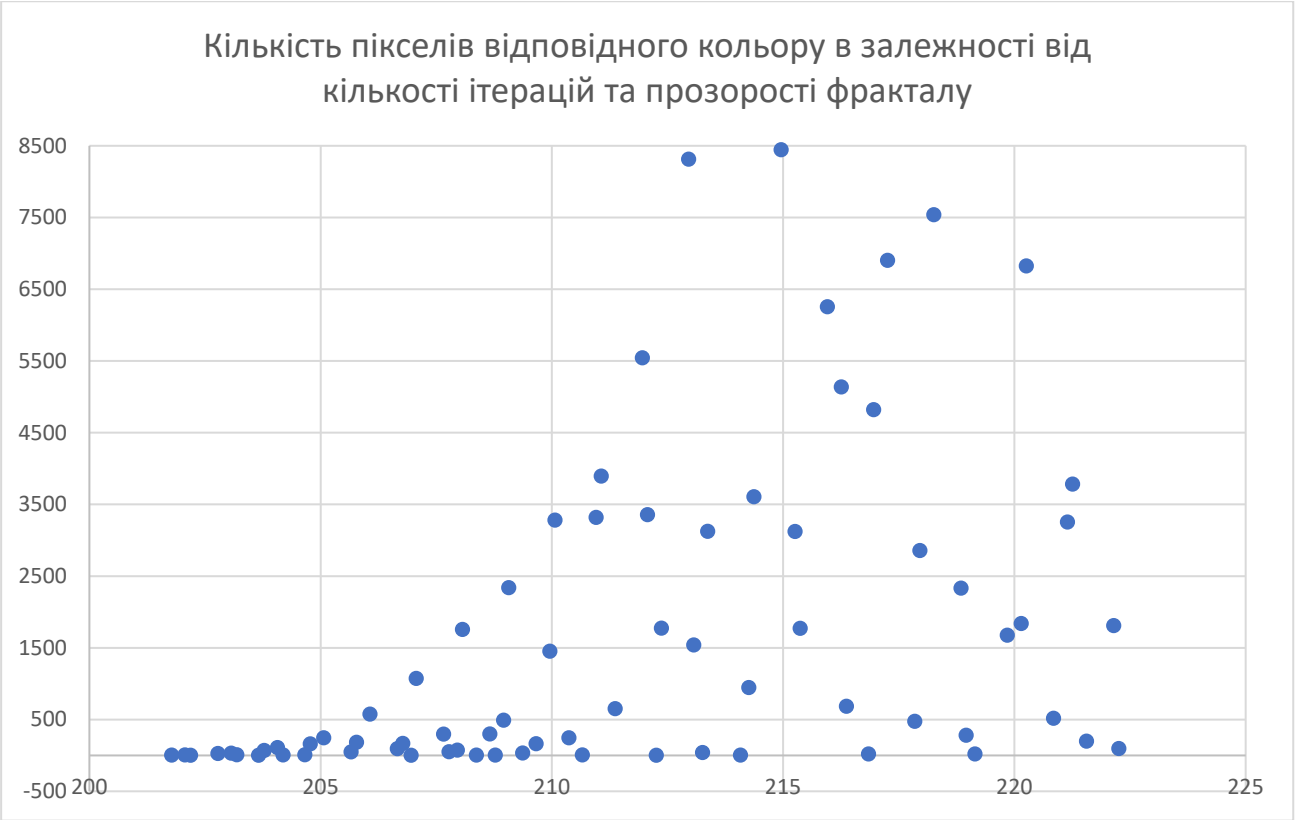


Рисунок 3.23 – Залежність кількості пікселів від GrayScale кольору та ступеню розвитку фракталу, n = 8

Результати досить цікаві, бо вони дають нам змогу побачити схожість у контурі та формі фігури, навіть на дуже різних за ступенем розвитку, фракталах. Знову прослідковується більша кількість пікселів на менших значеннях кольору на фракталі з низьким ступенем розвитку (рис. 3.21) з подальшим зростанням показнику на біль світлих кольорах, з високмими значеннями (рис. 3.22 та рис. 3.23).

Тобто, можна зробити висновок, що при накопиченні даних, можна не тільки робити прогнозування щодо можливих змін графіку при повороті фігури чи зміненні кофіцієнту прозорості матеріалу, але навіть якщо зробити ці зміни одночасно, включаючи ступінь розвитку фігури.

3.2 Аналіз загальних метрик зображень, спираючись на значну кількість проведених експериментів

Для того, щоб визначити кількість необхідних поодиночних генерацій фракталів, треба перемножити всі параметри: кількість заготовлених фракталів (типів фракталу), кути повороту фігури (5 кутів по всіх осях), прозорість матеріалу (від 0.1 до 1), кількість ітерацій (від однієї до десяти).

Рисунок 3.18 – Насиченість кольору в залежності від прозорості та типу фігури

В результаті це 1320 експериментів для збору необхідних метрик. Це значна кількість монотонної роботи, яка відповідно може зайняти дуже значну кількість часу. Притому, в разі виникнення потреби повторити експерименти з будь-яких причин – зміна умов, вимог, пошук розбіжностей тощо, ця процедура буде займати так само багато часу. Тому, необхідно було автоматизувати цей процесс.

Завдяки успішній інтеграції в програмний продукт Cepress.io, було створено автоматизаційні скрипти для симуляцію програмою дій реального користувача: відкриття браузеру, переходу на сторінку з програмою, обрання відповідних кофігурацій, збереженні мета-даних, збереження зображення, розрахунок та збереження метрик.

Можна побачити залежність показника грейскейл кольору від прозорості фігури. Одразу слід зазначити, що вбік до нуля йдуть значення з більшим віддтінком кольору та зі збільшенням числового показника віддінок кольору відповідно зменшується (рис 3.18, 3.19).

Рисунок 3.19 – Насиченість кольору в залежності від кількості ітерацій та фігури

Як видно на малюнку вище (рис. 3.18), зі збільшенням прозорості, збільшується значення кольорів (тобто, зменшується їх насиченість). У випадку з тетраідером з пірамідами назовні (custom_tetraider) ці показники

менші, ніж у Тетраідеру Серпинського та модифікованого Тетраідеру Серпинського, з пірамідами всередині.

Можна одразу побачити, що показники насиченості кольору у випадку з Тетраідер з пірамідами всередині має менші показники, аніж Тетраідер Серпінського.

Окрім іншої фігури і відповідного іншого контуру та отриманої тіні, треба зауважити про особливість ступеню розвитку. У випадку з модифікацією Тетраідеру Серпинського з пірамідами назовні, при збільшенні ступеню розвитку зменшується числовий показник кольору, тобто відтінок кольору зростає досить інтенсивно (рис 3.19). Якщо саме зростання цілком зрозуміло, то окрему увагу слід приділити динаміці змін у розмірах фракталу, бо у даному випадку фрактал збільшується з кожною ітерацією (рис 3.20 та рис 3.21).

Рисунок 3.20 – Модифікація Тетраідера Серпинського з пірамідами спрямованими назовні, n = 1

Рисунок 3.21 – Модифікація Тетраідера Серпинського з пірамідами назовні, n = 8

Відповідно, по графіку можна відстежити зростання та зниження показнику кольору. Для більш детального огляду, дані представлено поодиночно для кожної фігури. У даному маштабі чітко видно, наскільки ступінь розвитку Модифікація Тетраідера Серпинського з пірамідами назовні впливає на тінь, отриману від фракталу на площині.

У випадку з класичним Тетраідером Серпинського та Модифікацією Тетраідеру Серпинського з пірамідами всередині тенденції зміни показників мають спільний характер. Можна відмітити загальні збільшені цифри у варіанті тетраідеру з модифікацією, це обумовлено збільшенням площини фігури загалом, так як зі ступенем розвитку, отримуються нові додаткові піраміди, кожної ітерації, в свою чергу, залежно від ступеню прозорсті, це впливає на загальну картину.

Рисунок 3.22 – Насиченість кольору в залежності від кількості ітерацій, фігура - Модифікація Тетраідера Серпинського з пірамідами спрямованими назовні

Рисунок 3.23 – Насиченість кольору в залежності від кількості ітерацій, фігура - Тетраідер Серпинського

Рисунок 3.24 – Насиченість кольору в залежності від кількості ітерацій, фігура - Модифікація Тетраідера Серпинського з пірамідами всередині

Далі слід звернути увагу на тенденції у зміні середнього кольору та мінімального значення кольору в залежності від прозорості та кількості ітерацій (рис. 3.26 – 3.28).

Можна побачити, що змінення показників мінімального кольору ведуть себе досить прогнозовано, в залежності від зміни прозорості фракталу та зміни ступеня розвитку. При цьому, середнє значення у випадку з Тетраідером Серпинського мають високі показники загалом та менше повторюють тенденції мінімального кольору. У випадку з Модифікацією Тетраідера Серпинського з пірамідами всередині, у свою чергу, більше відслідковується сихнронне зниження і зростання, в той час як у випадку з Модифікацією Тетраідера Серпинського з пірамідами спрямованими назовні, синхронність змін є максимальною в даному експерименті – це обумовлено більш великими маштабами фігури при високих ступенях розвитку.

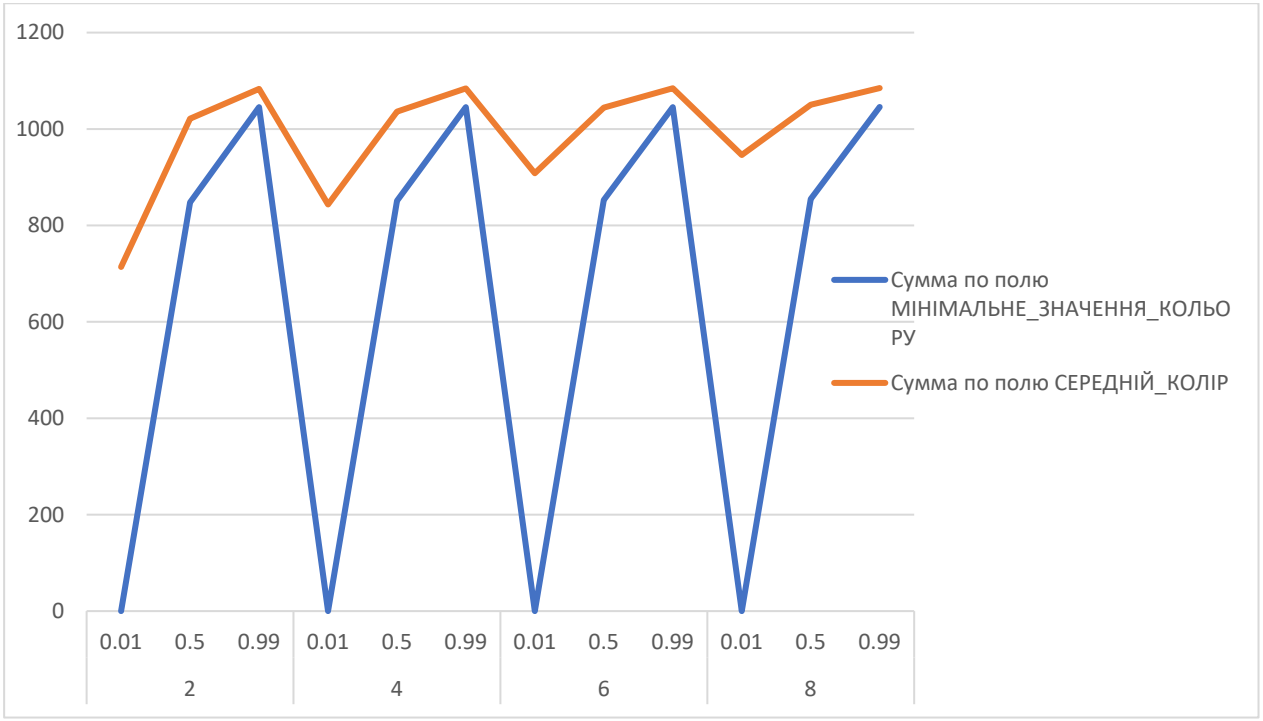


Рисунок 3.25 – Насиченість кольору в залежності від кількості ітерацій, фігура - Тетраідер Серпинського

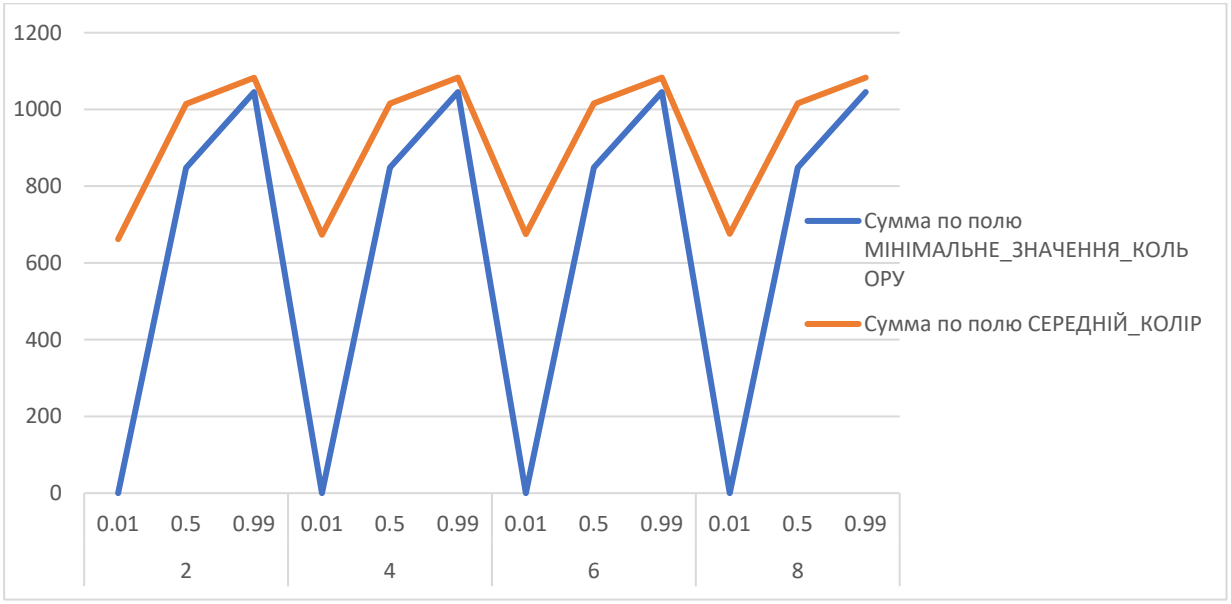


Рисунок 3.26 – Насиченість кольору в залежності від кількості ітерацій, фігура - Модифікація Тетраідера Серпинського з пірамідами всередині

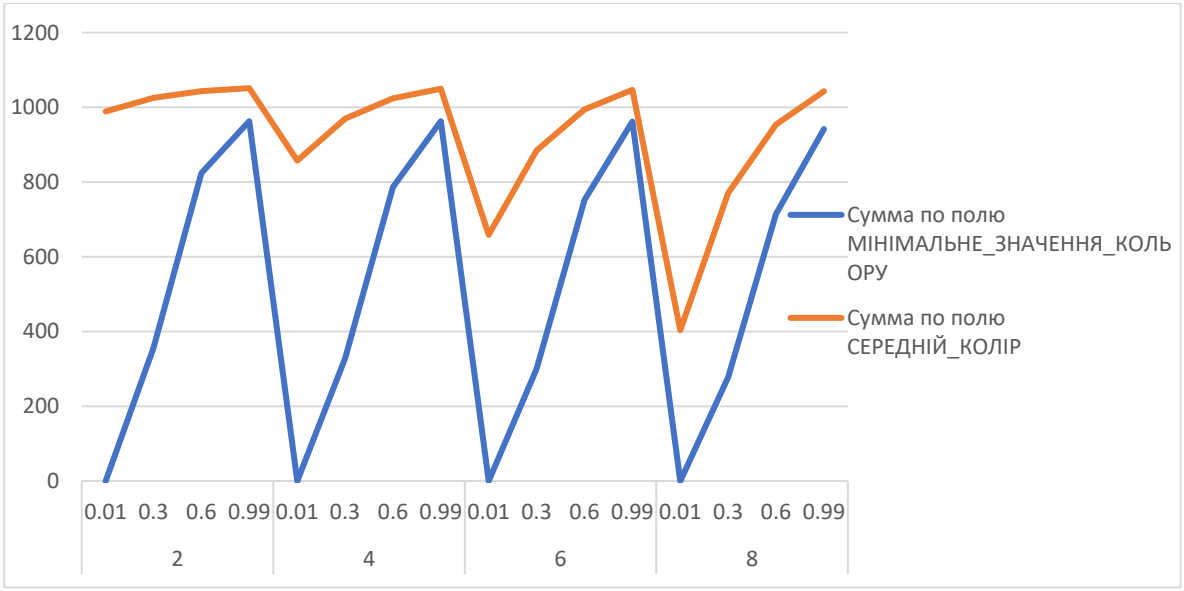


Рисунок 3.27 – Насиченість кольору в залежності від кількості ітерацій, фігура - Модифікація Тетраідера Серпинського з пірамідами спрямованими назовні

Висновки до третього розділу

В результаті проведених досліджень, було опрацьовано багато даних, які були попередньо зібрані як при поодиночних генераціях фракталів, так і при налаштованій системі автоматизації з перебором значної кількості комбінацій різних показників, як самого фракталу, так і показників його матеріалу.

Треба зазначити, що хоч загалом отримані очікувані результати, проте самі тенденції та деталі змін є досить цікавими і досить важливим є той факт, що тепер можна зробити висновок про можливість прогнозування даних речей, звісно, маючи вхідні умови для генерації.

В будь-якому разі, дані експерименти проводилися з урахуванням можливостей движку та в межах реалізованого функціоналу, тому звісно є простір для розвитку, як реалістичності зображення, так і урізноманітнення розрахунків та експериментів. Також, було б цікаво дослідити фрактали на більш високих ступенях розвитку, проте це вимагає досить потужної ЕВМ із великими процесорними та графічними можливостями.

ЗАГАЛЬНІ ВИСНОВКИ

В результаті виконаної роботи було створено програмний комплекс з можливість генерувати фрактали, додавати освітлення та виконувати налаштування трьохвимірної сцени. Також була створена окрема програма для розрахунку чисельних показників кольрів зображень.

Завдяки обраному рушію можна відрендерити тривимірні об’єкти на сцену, додати світло і побачити ці об’єкти. Також, в залежності від матеріалу буде певна реакція, або у певних випадках з використанням «нефізичних» матеріалів, ця реакція на світло буде відсутня. Це може бути пояснено як мінімум питанням швидкодії програми – в залежності від кількості додаткових операцій, створюється певне навантаження на систему, при цьому не завжди є необхідність в максимізації якості. Наприклад, в деяких ситуаціях, деякі об’єкти на екрані, відіграють другорядну роль для глядача (наприклад, фонові об’єкти). Тому, потрібен баланс якості та ціни на ресурси, що необхідні для забезпечення цієї якості.

Завдяки створеному додатку з аналізу показників прозорості фігури фракталу, ми маємо можливість порівнювати кількісні характеристики кольорів, в залежності від фігури фракталу, ступеню розвитку, прозорості матеріалу, повороту фігури тощо. Це є окремий додаток, який не залежить від створеного раніше генератора фракталів, та може використовуватися на будь-яких інших зображеннях за схожими сценаріями. Можливість завантажити картинку та вибрати область для розрахунку характеристик кольорів робить додаток досить зручним у використанні.

Додатково, треба підкреслити важливість автоматизації у сучасному світі, у тому числі при розробці програмного забезпечення. Завдяки тому, що процес генерації зображень та збір показників прозорості було автоматизовано, стало можливим проаналізувати не тільки поодиночні експерименти, але й їх

чисельну кількість, що дає змогу провести аналітику та порівняльний аналіз отриманих результатів, які залежать від обраних конфігурацій при генеруванні фракталів.

Отриманий програмний комплекс може бути використаний для подальших вдосконалень – продумана архітектура має достатню гнучкість, щоб з легкістю додавати нові види фракталів. Також, можна втрутитися у рендерінг процес рушія, задля експериментів з отримання більш реалістичної поведінки. Доступна можливість перенести логіку алгоритмів генерації фракталів в окрему програму, для використання з іншим рушієм, бо компоненти системи розроблені в ізоляції з дотриманням інкапсуляції внутрішньої реалізації та чіткими інтерфейсами для назовнішого використання.

На практиці, є можливість розвиватися в бік інтеграції з виробництвом фізичних моделей, де програмний комплекс буде давати початкові дані та прогнозовані результати, а відповідний фахівець, отримавши цікаві для себе результати в програмі, на основі застосованих конфігурацій, може приймати рішення щодо створення фізичної копії моделі.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Fractals. Physics of Solids and Liquids [Текст]: Jens Feder; Springer Science & Business Media, 2013. - 284 с.
2. Analysis on Fractals [Текст]: Jun Kigami; Cambridge University Press, 2001. - 226 с.
3. Fractals Everywhere [Текст]: Michael F. Barnsley; Academic Press, 2014. - 548 с.
4. The Fractal Geometry of Nature: Benoit B. Mandelbrot / W. H. Freeman and Company, 1982. - 468 с.
5. Fractals in Science [Текст]: Armin Bunde / Shlomo Havlin; Springer, 2013. - 300 с.
6. Комп'ютерна графіка. Навчальний посібник [Текст]: Пічугін М.Ф. / Канкін І. О. / Воротніков В. В.; К.: Центр учбової літератури, 2013. - 346 с.
7. Документація з технології ThreeJS [Електронный ресурс] // Режим доступа до сайту: https://threejs.org/docs/index.html (дата звернення 01.11.2023)
8. Документація з технології PixiJS [Електронный ресурс] // Режим доступа до сайту: https://pixijs.com/guides (дата звернення 01.11.2023)
9. Документація з технології BabylonJS [Електронный ресурс] // Режим доступа до сайту: https://doc.babylonjs.com/ (дата звернення 01.11.2023)
10. Документація з технології PlayCanvas [Електронный ресурс] // Режим доступа до сайту: https://developer.playcanvas.com/en/user-manual/ (дата звернення 01.11.2023)
11. Документація з технології MatterJS [Електронный ресурс] // Режим доступа до сайту: https://brm.io/matter-js/ (дата звернення 01.11.2023)
12. Документація з технології Phaser.io [Електронный ресурс] // Режим доступа до сайту: https://newdocs.phaser.io/docs/3.70.0 (дата звернення

01.11.2023)

13. Panagiotis Antoniadis: How to Convert an RGB Image to a Grayscale [Електронный ресурс] // Режим доступа до сайту: https://www.baeldung.com/cs/convert-rgb-to-grayscale (дата звернення 01.11.2023)
14. Основи комп'ютерної графіки: Ігор Мельник; Ліра-К, 2017. - 440 с.
15. Geometric Tools for Computer Graphics [Текст]: Philip Schneider / David H. Eberly; Elsevier, 2002. - 1056 с.
16. Основи комп'ютерної графіки. Оптика та освітлення: Олександр Бігура; Ліра-К, 2018. - 368 с.
17. Image Processing for Computer Graphics [Текст]: Jonas Gomes / Luiz Velho; Springer Science & Business Media, 2007. - 352 с.
18. Комп'ютерна графіка: Основи та стандарти: Олег Зайцев; Ліра-К, 2018. - 352 с.
19. African Fractals [Текст]: Ron Eglash; Modern computing and indigenous design, 2002. - 258 с.
20. Chaos and Fractals: New Frontiers of Science: Heinz-Otto Peitgen, et al.; Springer, 1992. - 919 с.
21. Основи графічного дизайну: Ігор Чугай; КМ Академія, 2019. - 304 с.
22.
23. Fractals Everywhere: Michael F. Barnsley; Academic Press, 1988. - 462 с.
24. Computer Graphics: Principles and Practice: John F. Hughes, et al.; Addison-Wesley, 2013. - 1264 с.
25. Real-Time Rendering: Tomas Akenine-Möller, et al.; CRC Press, 2018. - 1198 с.
26. Робота зі світлом та матеріалами: Олександр Бігура; Ліра-К, 2021. - 208 стор.
27. Computer Graphics with OpenGL: Donald Hearn, M. Pauline Baker; Pearson, 2017. - 848 с.

28. Physically Based Rendering: From Theory to Implementation: Matt Pharr, Greg Humphreys; Morgan Kaufmann, 2016. - 1264 c.
29. Introduction to Computer Graphics and the Vulkan API: Kenwright, et al.; CRC Press, 2019. - 370 c.
30. Computer Graphics: Principles and Practice in C: Foley, van Dam, et al.; Addison-Wesley, 1995. - 1178 c.
31. OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V: John Kessenich, et al.; Addison-Wesley, 2016. - 936 c.
32. Computer Graphics: From Pixels to Programmable Graphics Hardware: David S. Ebert, et al.; CRC Press, 2017. - 750 c.
33. Real-Time Collision Detection: Christer Ericson; Morgan Kaufmann, 2004. - 696 c.
34. Real-Time Rendering, Fourth Edition: Tomas Akenine-Möller, et al.; CRC Press, 2018. - 1136 c.
35. Introduction to Algorithms: Thomas H. Cormen, et al.; The MIT Press, 2009. - 1312 c.
36. Graphics Shaders: Theory and Practice: Mike Bailey, Steve Cunningham; CRC Press, 2011. - 425 c.
37. Computer Graphics: Principles, Algorithms, and Applications: J.D. Foley, et al.; Addison-Wesley, 1987. - 1248 c.
38. Graphics Programming Black Book: Michael Abrash; Coriolis Group, 1997. - 1156 c.
39. Mathematics for 3D Game Programming and Computer Graphics: Eric Lengyel; Charles River Media, 2002. - 586 c.

ДОДАТКИ

ДОДАТОК А

Тези до міжнародної науково-практичної конференції

Міністерство освіти і науки України
Міністерство інфраструктури України
Український державний університет науки та технологій
Східний науковий центр транспортної академії наук

TEMPUS: CITISET & SEREIN & CRENG

ТЕЗИ
XVII Міжнародної науково-практичної конференції
«СУЧАСНІ ІНФОРМАЦІЙНІ ТА КОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ НА ТРАНСПОРТІ, В ПРОМИСЛОВОСТІ ТА ОСВІТІ»

ABSTRACTS
of the XVII International Conference
«MODERN INFORMATION AND COMMUNICATION TECHNOLOGIES ON A TRANSPORT, IN INDUSTRY AND EDUCATION»

13.12.2023 – 14.12.2023

Дніпро
2023

Дослідження моделей оптичних перетворень негладких фрактальних поверхонь

Зайцев О. В., Шинкаренко В. І.,
Український державний університет науки і технологій, Україна

Фрактали є структурами, що мають самоподібність на різних масштабах. Вивчення їх оптичних властивостей допомагає краще зрозуміти природу цих структур та їх вплив на різноманітні процеси у фізиці, математиці та інших галузях.

Наукові дослідження оптичних властивостей фракталів залишаються актуальними через кілька аспектів. Дослідження фрактальних структур може призвести до розробки нових матеріалів та оптичних пристроїв з унікальними властивостями. Наприклад, фрактальні антени, оптичні фільтри та лінзи можуть мати покращені характеристики. Дослідження фрактальних структур у контексті квантової фізики може вести до розуміння їх впливу на квантові явища та можливість застосування у квантових технологіях. Фрактальний аналіз може бути корисним у медичних дослідженнях, наприклад, в аналізі зображень біомедичних структур або в розвитку нових методів обробки медичних зображень. Дослідження оптичних властивостей фракталів може вести до розробки нових методів вимірювань та обробки даних, що можуть мати широкі застосування в різних галузях. Використання фрактальних принципів у технологіях зображення та датчиках може призвести до покращення якості та чутливості пристроїв.

Узагальнюючи, дослідження оптичних властивостей фракталів не тільки розширює наше розуміння природи цих структур, але також відкриває нові можливості для розробки технологій і застосувань у різних галузях науки та техніки.

Представлена робота присвячена визначенню пропускної здібності просторових фракталів різної форми, в залежності від їх властивостей, таких як прозорість та дзеркальність. Розроблено відповдне програмне забезпечення для виконання експериментальних досліджень.

Експерименти проводились з пірамідальними фракталами різних видів та правил для формування кожної наступної ітерації. Визначається тривимірна сцена з сформованим об’єктом та площиною під ним, камера та точкове світло спрямовані в його напрямку.

При генерації фракталу надається можливість обрати фігуру, кількість ітерацій, характеристики матеріалу, такі як: прозорість, блиск, металевість, колір, поворот за віссю, позиція, характеристики джерела освітлення: колір, інтенсивність, відстань, кут, позиція. В ході експериментів зокрема було необхідно з’ясувати яким чином вид та ступінь розвитку об’ємного фракталу впливає на його загальну прозорість.

Обрано наступний сценарій експерименту: береться три види фрактальної фігури, кількість ітерацій від одного до восьми, прозорість матеріалу від 0.1 до одиниці, оберти за осями у радіанах від $-\pi$ до $+\pi$ та виконується перебір великої кількості комбінацій цих показників.

Експеримент виконується автоматично, метрики збираються у файл на локальному обчислювальному пристрої. Для проведення експериментів була необхідність проводити велику кількість поодиночних генерацій фрактальних фігур із збором метрик. Для виконання цих операцій вручну, знадобилася б значна кількість часу рутинної роботи. Використовувалася технологія selenium web driver розповсюджена у сучасних IT компаніях для автоматизації процесу тестування програмного забезпечення.

Дослідження передбачає отримання інсайтів у взаємозв'язок між параметрами генерації фракталів та їхніми оптичними властивостями. Різні конфігурації параметрів призводять до унікальних оптичних ефектів, що можуть бути використані для подальших досліджень у галузі оптики, комп'ютерної графіки та мистецтва.

За допомогою розробленого ПЗ, можуть генеруватися моделі для подальшої розробки фізичних прототипів, підібравши відповідні характеристики матеріалу.

ДОДАТОК Б

Проект статті

УДК 629.45.048.3:[001.891:027.7]

О. В. ЗАЙЦЕВ[1*]

[1*]Наукова бібліотека, Український державний університет науки і технологій, вул. Лазаряна, 2, Дніпро, Україна, 49010, тел. +38 (056) 371 51 05, ел. пошта lib@b.diit.edu.ua, ORCID 0000-0002-4603-4375

Дослідження моделей оптичних властивостей негладких фрактальних поверхонь

Мета. Дослідження спрямовано на розкриття теми оптичних властивостей фракталів. В сучасному світі оптика та оптичні технології знаходять застосування в різних галузях науки та промисловості. Важливим аспектом цих досліджень є вивчення властивостей оптичних матеріалів та структур, які можуть мати значний вплив на розробку нових оптичних пристроїв та систем. Однією з цікавих областей вивчення оптичних властивостей є дослідження фрактальних поверхонь, які характеризуються нерівномірністю та структурною складністю на будь-якому масштабі. **Методика.** Завдяки створеному авторами програмного комплексу було виконано значну кількість генерацій тривимірних сцен із фрактальною фігурою та джрелом освітлення, спрямованої крізь неї на площину. **Результати**. Авторами доведено, що прозорість та ступінь розвитку фракталу має безпосередній вплив на показники отриманої тіні. Завдяки розрахункам показників прозорості з отриманих зображень, є можливість використання даної інформації для втілення вдало підібраних моделей у їх фізичні копії. **Наукова новизна.** В сучасному світі оптика та оптичні технології знаходять застосування в різних галузях науки та промисловості. Важливим аспектом цих досліджень є вивчення властивостей оптичних матеріалів та структур, які можуть мати значний вплив на розробку нових оптичних пристроїв та систем. Однією з цікавих областей вивчення оптичних властивостей є дослідження фрактальних поверхонь, які характеризуються нерівномірністю та структурною складністю на будь-якому масштабі. **Практична значимість**. Дослідження фрактальних поверхонь може мати практичний вплив на покращення вже існуючих та/або розробку нових оптичних матеріалів для промислових та наукових застосувань, таких як безпілотні апарати, літаки, лінзи, сенсори, та інші. При цьому, вивчення оптичних властивостей фрактальних структур може відкрити нові можливості для покращення оптичних систем та пристроїв.

Ключові слова: фрактали; тривимірні фрактали; прозорість фрактальних фігур; показники прозорості фракталу; наукові статті; комп'ютерна графіка; графічні рушії; графіка у web; тривимірна графіка у браузері.

Вступ

Фрактали є структурами, що мають самоподібність на різних масштабах. Вивчення їх оптичних властивостей допомагає краще зрозуміти природу цих структур та їх вплив на різноманітні процеси у фізиці, математиці та інших галузях.

Наукові дослідження оптичних властивостей фракталів залишаються актуальними через кілька аспектів. Дослідження фрактальних структур може призвести до розробки нових матеріалів та оптичних пристроїв з унікальними властивостями. Наприклад, фрактальні антени, оптичні фільтри та лінзи можуть мати покращені характеристики.

Дослідження фрактальних структур у контексті квантової фізики може вести до розуміння їх впливу на квантові явища та можливість застосування у квантових технологіях. Фрактальний аналіз може бути корисним у медичних дослідженнях, наприклад, в аналізі зображень біомедичних структур або в розвитку нових методів обробки медичних зображень.

Дослідження оптичних властивостей фракталів може вести до розробки нових методів вимірювань та обробки даних, що можуть мати широкі застосування в різних галузях. Використання фрактальних принципів у технологіях зображення та датчиках може призвести до покращення якості та чутливості пристроїв

Мета

Враховуючи вищезгадане, автори мають за мету отримання нових знань про розвиток наукового напрямку вивчення оптичних властивостей тривимірних фракталів для генерації ідей щодо застосування властивостей фрактальних поверонь на практиці – при розробці непомітних безпілотних апаратів, дронів, отпичних лінз, тощо.

Методика

Завдяки створеному авторами програмного комплексу було виконано значну кількість генерацій тривимірних сцен із фрактальною фігурою та джрелом освітлення, спрямованої крізь неї на площину.

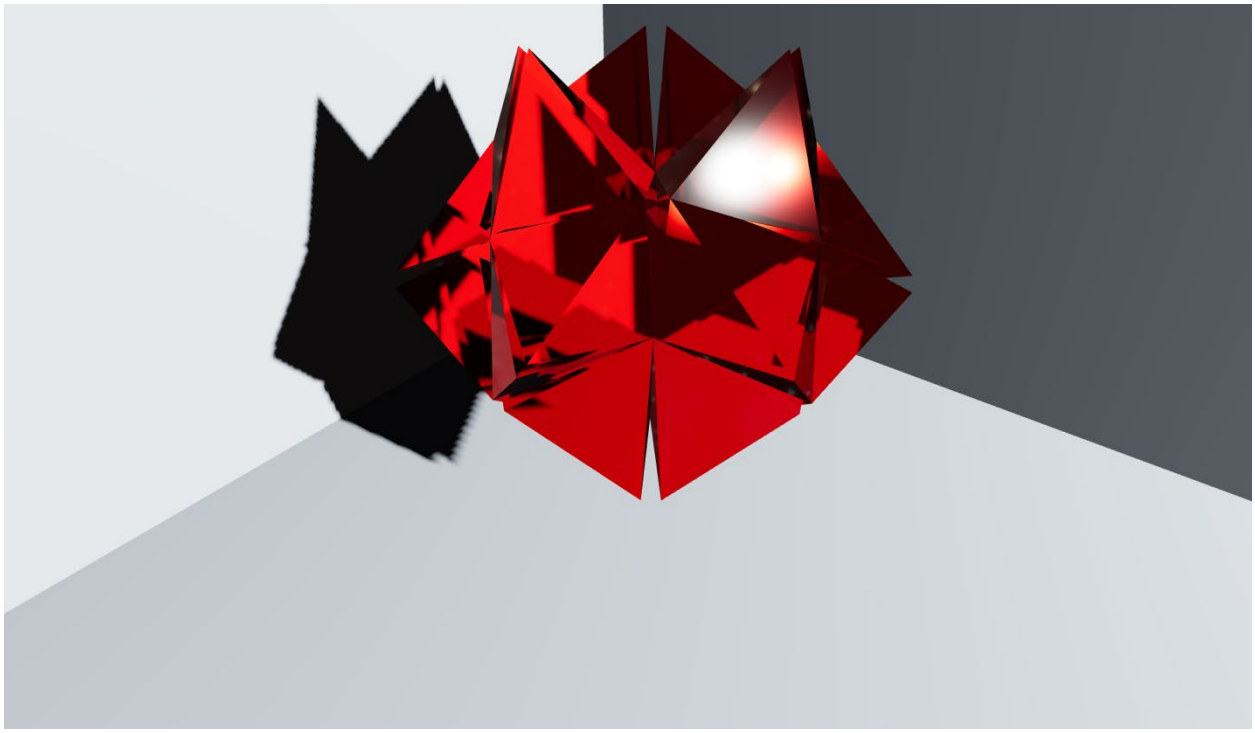


Рис. 1. Варіація Тетраідера Серпинського із пірамідами спрямованими назовні, з використанням правильних пірамід як початкових фігур, матеріал прозорий, n = 3

Fig. 1. A variation of Sierpinski's Tetrahedron with outward facing pyramids, using regular pyramids as starting figures, material transparent, n = 3

Результати

Авторами доведено, що прозорість та ступінь розвитку фракталу має безпосередній вплив на показники отриманої тіні. Завдяки розрахункам показників прозорості з отриманих зображень, є можливість використання даної інформації для втілення вдало підібраних моделей у їх фізичні копії.

Наукова новизна та практична значимість

Авторами цієї роботи було виконано дослідження дослідження оптичних властивостей тривимірних фракталів. В сучасному світі оптика та оптичні технології знаходять застосування в різних галузях науки та промисловості. Важливим аспектом цих досліджень є вивчення властивостей оптичних матеріалів та структур, які можуть мати значний вплив на розробку нових оптичних пристроїв та систем. Однією з цікавих областей вивчення оптичних властивостей є дослідження фрактальних поверхонь, які характеризуються нерівномірністю та структурною складністю на будь-якому масштабі.

Дослідження фрактальних поверхонь може мати практичний вплив на покращення вже існуючих та/або розробку нових оптичних матеріалів для промислових та наукових застосувань, таких як безпілотні апарати, літаки, лінзи, сенсори, та інші. При цьому, вивчення оптичних властивостей фрактальних структур може відкрити нові можливості для покращення оптичних систем та пристроїв.

Висновки

В результаті проведених досліджень, було опрацьовано багато даних, які були попередньо зібрані як при поодиночних генераціях фракталів, так і при налаштованій системі автоматизації з перебором значної кількості комбінацій різних показників, як самого фракталу, так і показників його матеріалу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Fractals. Physics of Solids and Liquids [Текст]: Jens Feder; Springer Science & Business Media, 2013. - 284 с.
2. Analysis on Fractals [Текст]: Jun Kigami; Cambridge University Press, 2001. - 226 с.
3. Fractals Everywhere [Текст]: Michael F. Barnsley; Academic Press, 2014. - 548 с.
4. The Fractal Geometry of Nature: Benoit B. Mandelbrot / W. H. Freeman and Company, 1982. - 468 с.
5. Fractals in Science [Текст]: Armin Bunde / Shlomo Havlin; Springer, 2013. - 300 с.
6. Комп'ютерна графіка. Навчальний посібник [Текст]: Пічугін М.Ф. / Канкін І. О. / Воротніков В. В.; К.: Центр учбової літератури, 2013. - 346 с.
7. Документація з технології ThreeJS [Електронный ресурс] // Режим доступа до сайту: https://threejs.org/docs/index.html (дата звернення 01.11.2023)
8. Документація з технології PixiJS [Електронный ресурс] // Режим доступа до сайту: https://pixijs.com/guides (дата звернення 01.11.2023)
9. Документація з технології BabylonJS [Електронный ресурс] // Режим доступа до сайту: https://doc.babylonjs.com/ (дата звернення 01.11.2023)
10. Гордукалова, Г. Ф. Индекс цитирования в науке: цели использования, основные разновидности и ограничения. Г. Ф. Гордукалова. Вестн. СПбГУКИ. 2014. № 2 (19). С. 54–57.

O. V. ZAITSEV[1*]

[1*]Scientific Library, Ukrainian State University of Science and Technology, Lazaryan St., 2, Dnipro, Ukraine, 49010, tel. +38 (056) 371 51 05, e-mail lib@b.diit.edu.ua, ORCID 0000-0002-4603-4375

Study of models of optical transformations of non-smooth fractal surfaces

Purpose. The research is aimed at revealing the topic of optical properties of fractals. In the modern world, optics and optical technologies are used in various fields of science and industry. An important aspect of these studies is the study of the properties of optical materials and structures, which can have a significant impact on the development of new optical devices and systems. One of the interesting areas of study of optical properties is the study of fractal surfaces, which are characterized by unevenness and structural complexity on any scale. **Methodology**. Thanks to the software complex created by the authors, a significant number of generations of three-dimensional scenes with a fractal figure and a source of illumination directed through it to the plane were performed. **Findings**. The authors proved that transparency and the degree of fractal development have a direct effect on the parameters of the obtained shadow. Thanks to the calculations of transparency indicators from the obtained images, it is possible to use this information to translate well-

chosen models into their physical copies. **Originality**. In the modern world, optics and optical technologies are used in various fields of science and industry. An important aspect of these studies is the study of the properties of optical materials and structures, which can have a significant impact on the development of new optical devices and systems. One of the interesting areas of study of optical properties is the study of fractal surfaces, which are characterized by unevenness and structural complexity on any scale. **Practical value**. The study of fractal surfaces can have a practical impact on the improvement of existing and/or the development of new optical materials for industrial and scientific applications, such as drones, aircraft, lenses, sensors, and others. At the same time, studying the optical properties of fractal structures can open up new opportunities for improving optical systems and devices.

Keywords: air conditioning; railway transport; science articles; scientometric studies; Web of Science; energy efficiency; climate comfort; thermal comfort

REFERENCES

1. Fractals. Physics of Solids and Liquids [Text]: Jens Feder; Springer Science & Business Media, 2013. - 284 p.
2. Analysis on Fractals [Text]: Jun Kigami; Cambridge University Press, 2001. - 226 c.
3. Fractals Everywhere [Text]: Michael F. Barnsley; Academic Press, 2014. - 548 p.
4. The Fractal Geometry of Nature: Benoit B. Mandelbrot / W. H. Freeman and Company, 1982. - 468 p.
5. Fractals in Science [Text]: Armin Bunde / Shlomo Havlin; Springer, 2013. - 300 c.
6. Computer graphics. Study guide [Text]: Pichugin M.F. / Kankin I. O. / Vorotnikov V. V.; K.: Center of Educational Literature, 2013. - 346 p.
7. Documentation on ThreeJS technology [Electronic resource] // Site access mode: https://threejs.org/docs/index.html (access date 11/01/2023)
8. PixiJS technology documentation [Electronic resource] // Site access mode: https://pixijs.com/guides (access date 11/01/2023)
9. Documentation on BabylonJS technology [Electronic resource] // Site access mode: https://doc.babylonjs.com/ (access date 11/01/2023)
10. Gordukalova, G. F. Index of citations in science: purposes of use, main varieties and limitations. G. F. Gordukalov. Vestn. SPbGUKY. 2014. No. 2 (19). P. 54–57.

ДОДАТОК В
Технічне завдання

ЗАТВЕРДЖУЮ
Проректор
Українського державного університету науки і технології
Анатолій РАДКЕВИЧ

ПРОГРАМНІ ЗАСОБИ ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОПТИЧНИХ ПЕРЕТВОРЕНЬ НЕГЛАДКИХ ФРАКТАЛЬНИХ ПОВЕРХОНЬ

Технічне завдання
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1346–01 ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Віктор ШИНКАРЕНКО
Виконавець
_____Олександр ЗАЙЦЕВ
Нормоконтролер
_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО

44165850.1346-01

ПРОГРАМНІ ЗАСОБИ ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОПТИЧНИХ ПЕРЕТВОРЕНЬ НЕГЛАДКИХ ФРАКТАЛЬНИХ ПОВЕРХОНЬ

Технічне завдання

44165850.1346-01 13 01

Листів 12

2024

ВСТУП

Дослідження джерел освітлення вже обраного для розробки движку (ThreeJS) для подальшого використання при моделюванні 3Д фракталів, їх оптичних властивостей та проведення відповідних досліджень.

Кінцевий інструментарій для «Дослідження моделей оптичних перетворень негладких фрактальних поверхонь», що розробляється, має містити 2 додатки.

Перший додаток має виконувати функції побудови 3Д фракталів за відповідними конфігураціями. Інструментарій повинен включати конфігурування матеріалу фракталів (прозорість, віддзеркалення, тощо) та джерела освітлення. Додаток має генерувати відповідну сцену з фракталом згідно з обраними конфігураціями. Додаток повинен вміти відправляти результати своєї роботи, а саме: зображення сцени з фракталом та конфігурації, при яких було отриману цю сцену.

Другий додаток має вміти приймати зображення сцени з фракталом та конфігурацій, при яких було згенеровано цю сцену. Також, додаток має надавати можливість виділення ділянки на зображенні для подальшого прорахунку метрик кольорів та збереження результатів.

Додатки реалізовані за клієнт-серверною архітектурою.

Програмний продукт є безкоштовний, що робить його більш цікавим на фоні інших аналогів, а також завдяки способу реалізації, може бути доступним з різних девайсів. Інтернет має бути присутній тільки при першому запуску додатку, тому що подальші розрахунки відбуваються безпосередньо на самому клієнтському пристрої.

1 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виробничу практику за договором №832303 на проведення виробничої практики студентів вищих навчальних закладів від 21 серпня 2023 р.

2 ПРИЗНАЧЕННЯ РОЗРОБКИ

Функціональне призначення – програмний продукт має надати можливіть генерувати тривимірну сцену з моделлю фракталу та джерелом світла, спрямованого крізь нього на площу. Має надати можливість збору інформації про конфігурації, які були застосовані для генерації фракталу та отриманих показників прозорості, отриманих при аналізі збереженного зображення.

Експлуатаційне призначення – за допомогою програмного продукту можна виконувати графічне поделювання тривимірних фракталів, застосовувати конфігурації матеріалу фракталу та джерела освітлення, та розраховуавти показники прозорості, що має практичний вплив на покращення вже існуючих та/або розробку нових оптичних матеріалів для промислових та наукових застосувань, таких як безпілотні апарати, літаки, лінзи, сенсори, та інші. При цьому, вивчення оптичних властивостей фрактальних структур може відкрити нові можливості для покращення оптичних систем та пристроїв.

3 ВИМОГИ ДО ПРОГРАМИ

3.1 Вимоги до функціональних характеристик

Програмний продукт повинний забезпечувати можливість генерації тривимірної сцени з моделлю фракталу та джерелом освітлення, за вказаними критеріями (показниками).

Алгоритм побудови графічної тривимірної сцени включає:

- генерація тривимірної сцени з фракталом та джерелом освітлення;
- можливості конфігурації матеріалу фракталу, ступеню розвитку, конфігурації освітлення та маштабу зображення;
- збереження отриманих результатів та відправлення їх у програму-аналізатор показників прозорості.

Алгоритм роботи програми-аналізатора показників прозорості включає:

- завантаження файлу із зображенням;
- виділення ділянки зображення, для якої необхідно розрахувати показники прозорості;
- збереження розрахованих показників.

Програмний продукт повинний забезпечувати можливість змінювати обрану конфігурацію в режимі реального часу.

Вимоги до вихідних даних:

Вихідними даними є:

- створена директорія, назва відповідає поточній даті та часу;
- файл із зображенням, для якого прораховані показники прозорості;
- файл із конфігураціями, що були застосовані для генерації тривимірної сцени з фракталом;

- файл з отриманими показниками прозорості, зокрема мінімальне значення кольору, максимальне значення кольору, середнє значення кольору, медіанний колір та дані за кількістю пікселів за відповідним кольором.

3.2 Вимоги до надійності

- забезпечити не більше п'яти помилок на тисячу строк програмного коду;
- забезпечити доступність серверу для користувача сім днів на тиждень, 12 годин на добу (з восьмої години ранку до восьмої години вечора);
- забезпечити перезапуск сервера при виникненні критичної помилки протягом однієї години з моменту втрати зв’язку;
- забезпечити можливість незалежного перезапуску серверу графічної програми та програми-аналізатора показників прозорості;
- забезпечити можилвість доступу до програми з будь-якої локації, де є вихід в інтернет.

3.3 Умови експлуатації

Для забезпечення надійної роботи програмного продукту користувачу необхідно дотримуватись таких умов:

- програмний засіб повинен використовуватись у приміщеннях, призначених для роботи з ЕОМ з відповідними кліматичними умовами;
- працювати з програмним засобом може людина, що має навички роботи з ПК та ознайомлена з посібником користувача.

3.4 Вимоги до складу і параметрів технічних засобів

Програмний продукт розрахований для функціонування на пристроях, що мають такі мінімальні характеристики:

- процесор: 64 розрядний процесор із тактовою частотою 3.0 ГГц або вище з набором інструкцій SSE2;
- графічний процесор: відеокарта з пам’яттю 4 гігабайти;

- місце на диску: 128 Mb;
- об’єм оперативної пам’яті: 8 Gb;
- маніпулятори: клавіатура;
- інше устаткування: монітор.

3.5 Вимоги до інформаційної і програмної сумісності

Для роботи програмного продукту, на ЕОМ повинна бути встановлена операційна система, сумісна із стандартом POSIX.

3.6 Вимоги до маркування і упаковки

Програмний продукт поширюється у вигляді програмного коду із необхідністю подальшої компіляції. Програмний продукт повинен включати інструкції для компіляції програми.

3.7 Вимоги до транспортування і зберігання

Програмний продукт поширюватиметься мережею із хмарного сховища.

ЕКОНОМІЧНІ ПОКАЗНИКИ

Розрахунки економічних показників для цієї розробки не ведуться, оскільки вона не є комерціійною.

ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

До складу документації мають входити:

- специфікація;
- керівництво користувача;
- опис програми;
- текст програми.

Вся документація програмного додатку повинна задовольняти вимоги до програмної документації.

СТАДІЇ ТА ЕТАПИ РОЗРОБКИ

Таблиці 6.1 – Стадії та етапи розробки

Стадія	Зміст	Строки виконання
Технічне завдання	Постановка задачі, збір інформації, виріб та обґрунтування критеріїв розробки. Попередній вибір методів рішення задач. Визначення вимог до технічних засобів. Узгодження і затвердження технічного завдання.	01.10.23 –05.10.23
Робочий проект	Програмування та відлагодження програми.	05.10.23 – 20.10.23
	Тестування програми	20.10.23 – 25.10.23
	Розробка, узгодження і затвердження програмної документації.	25.10.23 – 31.10.23

ПОРЯДОК ПРИЙОМУ ТА КОНТРОЛЮ

Контроль за виконанням роботи здійснює керівник практики від університету Андрющенко В.О.

Прийом здійснюється комісією у складі, визначеному університетом.

БІБЛІОГРАФІЧНИЙ СПИСОК

Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

ДОДАТОК Г
Робочий проект

ЗАТВЕРДЖУЮ
Проректор
Українського державного університету науки і технології
Анатолій РАДКЕВИЧ

ПРОГРАМНІ ЗАСОБИ ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОПТИЧНИХ ПЕРЕТВОРЕНЬ НЕГЛАДКИХ ФРАКТАЛЬНИХ ПОВЕРХОНЬ

Робочий проект
ЛИСТ ЗАТВЕРДЖЕННЯ
44165850.1346–01–ЛЗ

Завідувач кафедри КІТ
_____Вадим ГОРЯЧКІН
Керівник розробки
_____Віктор ШИНКАРЕНКО
Виконавець
_____Олександр ЗАЙЦЕВ
Нормоконтролер
_____Світлана ВОЛКОВА

2024

ЗАТВЕРДЖЕНО

44165850.1346-01

ПРОГРАМНІ ЗАСОБИ ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОПТИЧНИХ ПЕРЕТВОРЕНЬ НЕГЛАДКИХ ФРАКТАЛЬНИХ ПОВЕРХОНЬ

Специфікація

44165850.1346-01

Листів 2

2024

Позначення	Найменування	Примітка
	Документація	
44165850.1346-01-ЛЗ	Лист затвердження	
44165850.1346-01 12 01-ЛЗ	Лист затвердження	
44165850.1346-01 12 01	Текст програми	
44165850.1346-01 13 01-ЛЗ	Лист затвердження	
44165850.1346-01 13 01	Опис програми	
44165850.1346-01 13 01-ЛЗ	Лист затвердження	
44165850.1346-01 13 01	Керівництво користувача	

ЗАТВЕРДЖЕНО

44165850.1346-01 І3 01

ПРОГРАМНІ ЗАСОБИ ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОПТИЧНИХ ПЕРЕТВОРЕНЬ НЕГЛАДКИХ ФРАКТАЛЬНИХ ПОВЕРХОНЬ

Керівництво користувача

44165850.1346-01 І3 01

Листів

2024

1 ПРИЗНАЧЕННЯ ТА УМОВИ ЗАСТОСУВАННЯ

1.1 Область застосування

Програмний засіб «3DFractalGenerator» (Графічний прогрмний засіб з генерації тривимірної сцени з моделлю фракталу) призначений для генерації тривимірних фракталів з можливістю вказання параметрів, таких як: фігура фракталу, кількість ітерацій, прозорість матеріалу фігури, розташування, колір, поворотот фігури, джерело освітлення, яскравість, напрямок.

Програмний засіб «ImageTransparencyAnalyzer» (засіб для збору та розрахунку показників прозорості зображень) призначений для виконання розрахунків показників прозорості попередньо збереженного зображення та конфігурацій, які було використано під час генерації фракталу.

Користування ПЗ можливе без проходження процедури ліцензування, програмний засіб безкоштовний.

1.2 Рівень підготовки користувача

Користуватися програмним засобом повинен фахівець, що має навички користування ПК та ознайомлений з даним керівництвом.

2 ПІДГОТОВКА ДО РОБОТИ

Для початку роботи з ПЗ необхідно виконати наступні кроки:

- увімкнути ПК;
- підключити ПК до інтернету;
- встановити ПЗ згідно переліку у Описі програми;
- відкрити браузер

3 ОПИСТ ОПЕРАЦІЙ

Для роботи з ПЗ треба:

- ввести адресу ресурсу з ПЗ – «3DFractalGenerator» (рис. 2. 1);

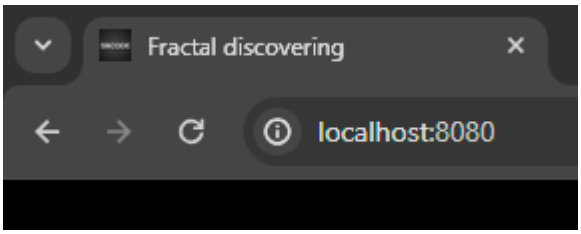


Рисунок 2.1 – Адреса ресурсу з програмним засобом «3DFractalGenerator»

- вибрати тип фігури, кількість ітерацій (рис. 2.2) та натиснути кнопку «Згенерувати 3Д фрактал»;

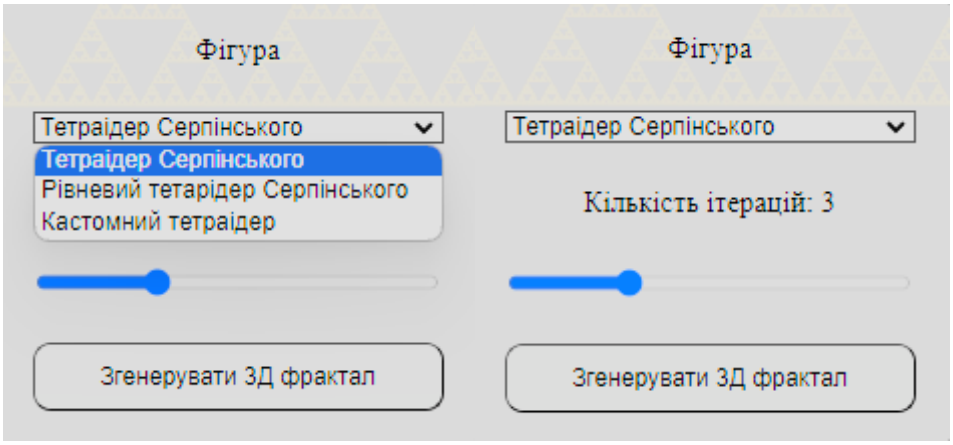


Рисунок 2.2 – Вибір фігури та кількості ітерацій

- на отриманій тривимірній сцені користувач повинен вибрати необхідні показники матеріалу фракталу та світла, коррдинати об’єктів, позицію та напрямок камери, маштаб сцени (рис. 2.3 – результат доступний зліва від меню;

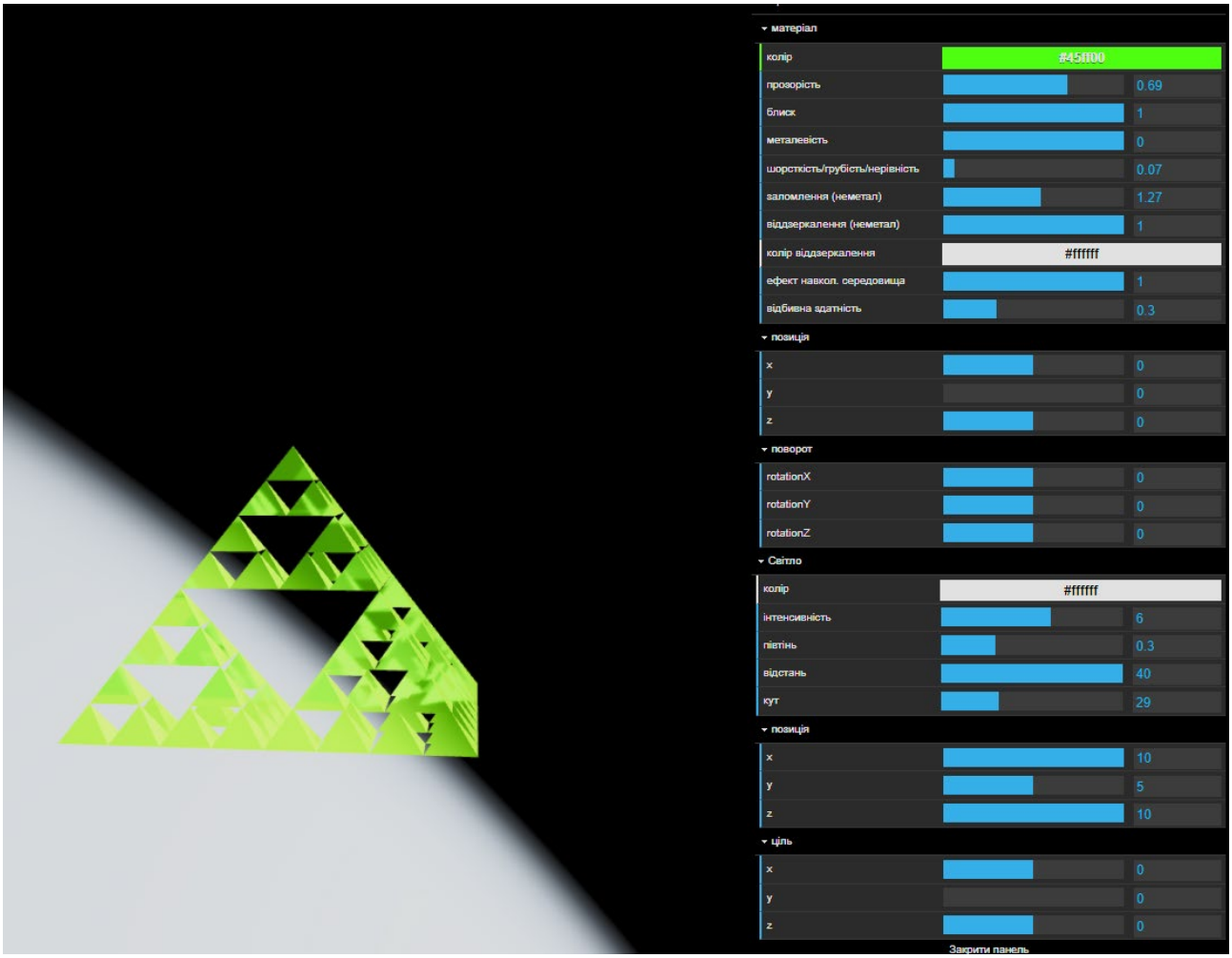


Рисунок 2.3 – Вибір показників матеріалу фракталу, конфігурацій освітлення, камери та маштабу

- натиснути на кнопку «Збереги зображення» та дочекатись повідомлення про успішне збережння, далі натиснути на кнопку «Переийти до аналізу показників прозорості» (рис. 2.4 та 2.5);

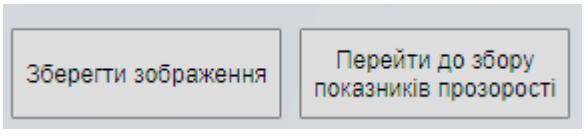


Рисунок 2. 4 – Початок процесу збереження отриманого зображення

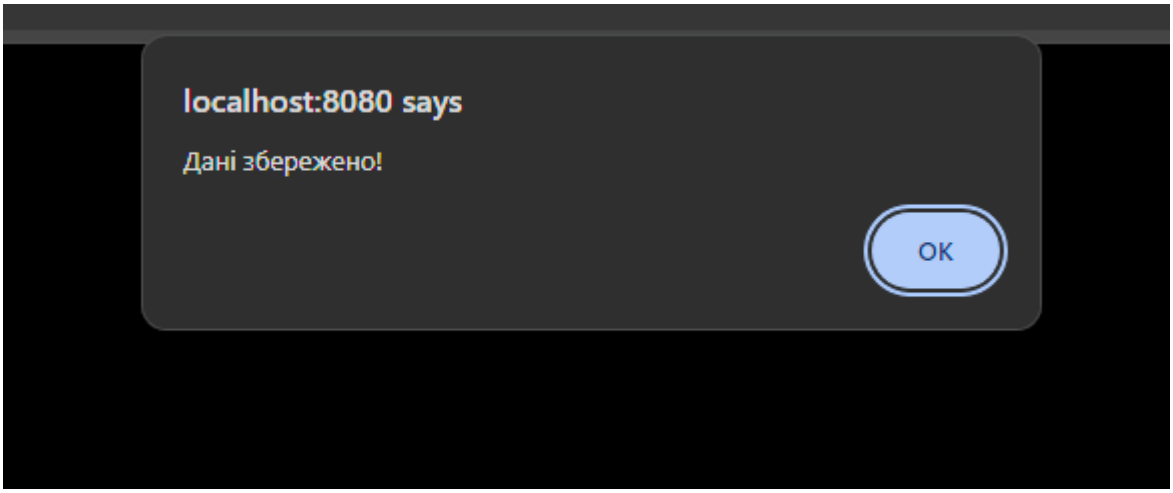


Рисунок 2.5 – Ідентифікатор успішного завершення процесу збереження зображення

- у відкритій кладці з додатком «ImageTransparencyAnalyzer» натиснути на «Choose File» та вибрати попердньо збережний файл – він буде мати наступний формат: «день-місяць-рік година-хвилина-секунда» (ім’я файлу має дату збережння файлу, відповідно);

- натиснути на ліву кнопку миші у верній ділянці, до початку фігури та протягнути до протилежного боку, де закінчується фігура (рис. 2.6);

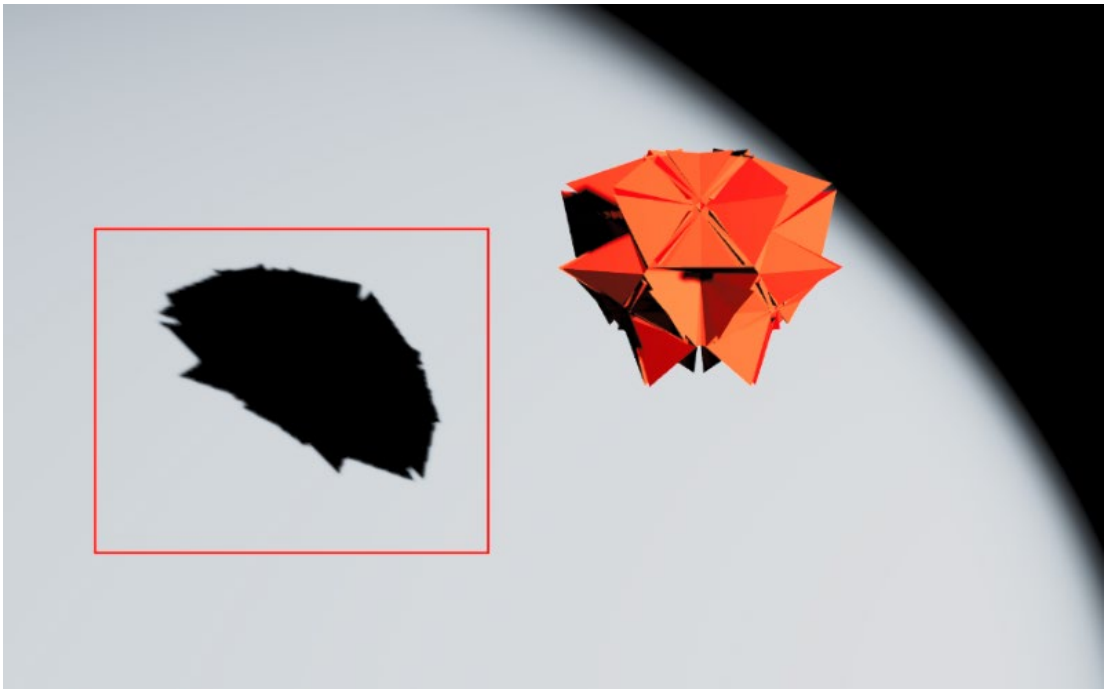


Рисунок 2.6 – Виділення області для розрахунку показника прозорості

- натиснути на кнопки «Порахувати метрики та збереги дані» (рис 2.7) та дочекатися повідомлення про успішність операції збереження (рис. 2.8);

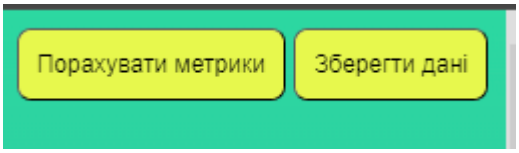


Рисунок 2.7 – Кнопки для старту початку розрахунку та збереження даних

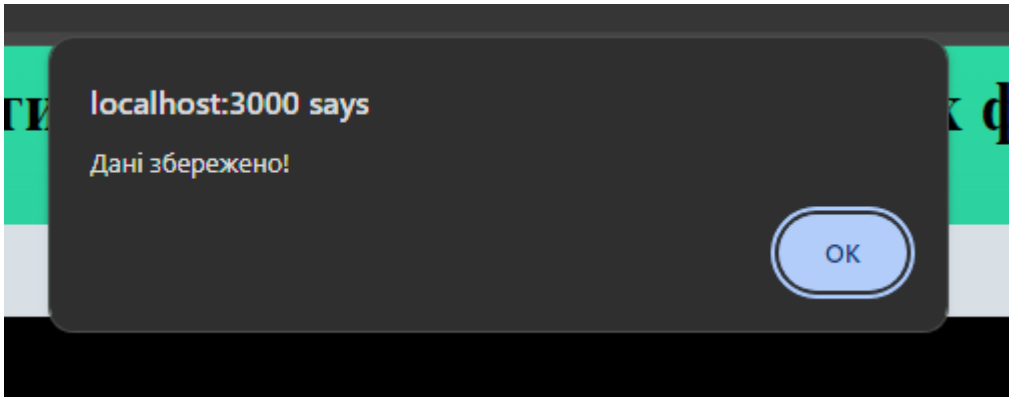
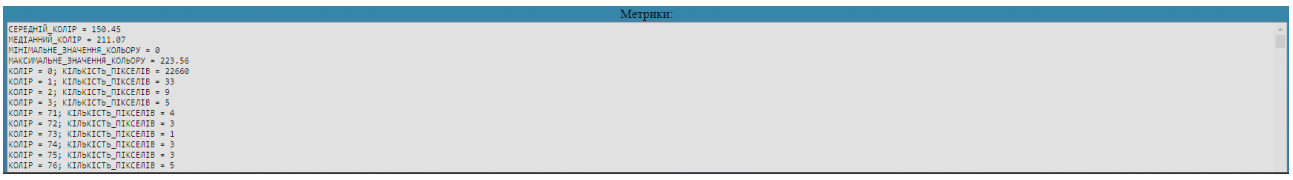


Рисунок 2.8 – Повідомлення про успішну операцію розрахунку показнику прозорості та збереження даних

- при необхідності дані також доступні для перегляду безпосередньо на екрані, під зображенням (рис. 2.9);



- далі користувач повинен відкрити директорію з поточною датою та часом, для перегляду збережених даних (рис 2.10).

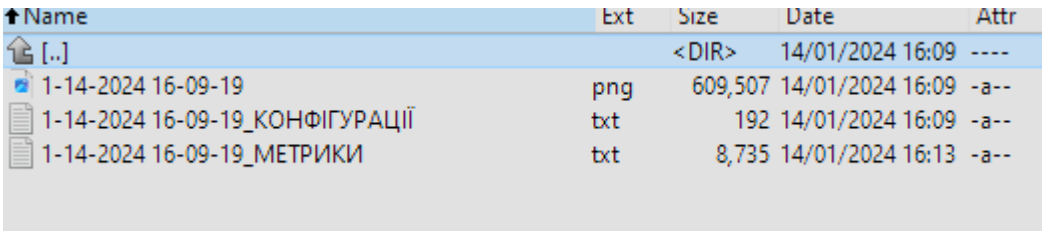


Рисунок 2.10 – Демострація збережених даних

Для виконання подальших розрахункиві користувач може застосувати засоби Excel для імпорту даних зі збережених файлів, або іншим зручним шляхом.

4 АВАРІЙНІ СИТУАЦІЇ

Якщо програма запускається на пристрої, який не відповідає необхідним технічним характеристикам, буде виведено відповідне повідомлення і запущено процедуру завершення роботи програми.

У випадку, якщо користувач закриє поточний екран моделювання до моменту збереження результатів, він буде вимушений заново повторювати відповідний перелік дій для повторення експерименту.

Якщо під час використання програма виникне критична помилка, користувачу буде рекомендовано перезапустити програму для відновлення коректного функціонування.

5 РЕКОМЕНДАЦІЇ ЩОДО ЗАСТОСУВАННЯ

Задля зниження ризику втрат даних, рекомендується робити збереження даних на кожні кілька етапів генерації фракталу.

Після закінчення експерименту, перед початком іншого, рекомендується оновити сторінку браузера задля очищення пам’яті від використоної інформації та прискорення роботи програми.

Рекомендується відкривати програмний комплекс «ImageTransparencyAnalyzer» через посилення з комплексу «3DFractalGenerator» задля більш швидкого завантаження, з використанням існуючої сессії та збережених даних.

6 БІБЛІОГРАФІЧНИЙ СПИСОК

Івченко, Ю.М. Основи стандартизації програмних систем [Текст]: методичні вказівки до дипломного проектування та лабораторних робіт / уклад.: Ю. М. Івченко, В. І. Шинкаренко, В. Г. Івченко; Дніпропетр. нац. ун-т залізн. трансп. ім. акад. В. Лазаряна. – Д.: Вид-во Дніпропетр. нац. ун-ту залізн. трансп. ім. акад. В. Лазаряна, 2009. - 38 с.

ЗАТВЕРДЖЕНО

44165850.1346-01 13 01

ПРОГРАМНІ ЗАСОБИ ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОПТИЧНИХ ПЕРЕТВОРЕНЬ НЕГЛАДКИХ ФРАКТАЛЬНИХ ПОВЕРХОНЬ

Опис програми

44165850.1346-01 13 01

Листів 15

2024

1 ЗАГАЛЬНІ ВІДОМОСТІ

Програмний продукт «ПРОГРАМНІ ЗАСОБИ ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОПТИЧНИХ ПЕРЕТВОРЕНЬ НЕГЛАДКИХ ФРАКТАЛЬНИХ ПОВЕРХОНЬ» призначений для генерування тривимірної сцени з фракталом, джерелом освітлення та площиною під ним, з можливістю використовувати різні конфігурації при генерації, зберігати отримані результати разом з використаними конфігураціями та можливістю ведення розрахунків отриманих показників прозоровсті.

Програмне забезпечення необхідне для функціонування – runtime environment NodeJS 18.17.1 або вище.

Мови програмування – TypeScript, JavaScript, GLSL, HTML, CSS.

2 ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Програмний продукт призначений для генерації тривимірних фракталів з можливістю вказання параметрів, таких як: фігура фракталу, кількість ітерацій, прозорість матеріалу фігури, розташування, колір, поворот фігури, джерело освітлення, яскравість, напрямок.

Функціональні обмеження: в системі існує виключний перелік фігур, тому не має можливості генерації фракталів довільної форми, кожна фігура повинна бути імплементована окремо.

3 ОПИС ЛОГІЧНОЇ СТРУКТУРИ

3.1 Архітектура програмного комплексу

Рисунок 3.1 – Високорівнева архітектура програмного комплексу з генерації сцени з тривимірними фракталами

3.2 Структура програми

Програмний комплекс складається з наступних програмних засобів:

1. Програмний засіб генерації тривимірної сцени із моделлю фракталу та джереом освітлення, спрямованого на нього;
2. Програмний засіб для збору та розрахунків показників прозорості, отриманих від графічного програмного засобу;
3. Програма-автоматизатор виконання генерацій фракталу, збору та розрахунків показників прозорості фракталів.

Загальний список програмних модулів для кожного з програмних засобів,

наведений нижче.

Програмний засіб генерації тривимірної сцени із моделлю фракталу та джереом освітлення, спрямованого на нього, включає наступні модулі:

1. Конфігураційна частина серверної першої програми;
2. Сервіс збору початкових фалів для відображення у браузері;
3. Конфігураційна частина клієнтської програми;
4. Модуль для відображення даних у браузері;
5. Алгоритми побудування точок для подальшого побудування тривимірних фракталних фігур;
6. Модуль роботи з камерою;
7. Код графічного інтерфейсу з елементами для керування об’єктами на тривимірній сцені;
8. Модуль налаштування світла
9. Побудування фрактальних фігур на основі отриманих точок
10. Автоматизація збору метрик згенерованних зображень

Програмний засіб для збору та розрахунків показників прозорості, отриманих від графічного програмного засобу, включає наступні модулі:

1. Модуль графічного інтерфесу для завантаження зображень та виконання розрахунків;
2. Сервіс для роботи з показниками прозорості – їх розрахунок та зберігання на локальному ПК.

Програма-автоматизатор виконання генерацій фракталу, збору та розрахунків показників прозорості фракталів, включає наступні модулі:

1. Модуль зі скриптами з переліком дій для симуляції поведінки користувача у браузері. Перелік дій включає можливі комбінації, які можуть буди застосовані при генерації тривимрної сцени з моделлю фракталу.

4 ВИКОРИСТАНІ ТЕХНІЧНІ ЗАСОБИ

Розроблюваний програмний продукт повинен використовуватись на ЕОМ, що мають:

- 8 гігабайт оперативної пам’яті;
- процесор: 64 розрядний процесор із тактовою частотою 3.0 ГГц або вище з набором інструкцій SSE2;
- графічний процесор: відеокарта з пам’яттю 4 гігабайти;
- 100 гігабайт простору на жорсткому диску;
- клавіатуру;
- мишу;
- монітор;
- доступ до інтернету.

5 ВИКЛИК ЗАВАНТАЖЕННЯ

Для того, щоб користуватись програмним продуктом необхідно:

- завантажити програмні засоби з гітхаб;
- в директорії з проектом виконати команду npm install, що зробить довстановлення необхідних пакетів програми;
- для запуску програми з генерації тривимірної сцени із моделлю фракталу необхідно виконати команду npm run dev.
- для запуску програми збору та розрахунків показників прозорості, отриманих від графічного програмного засобу, необхідно виконати команду node server;
- для запуску автоматизованих скрипитів збору показників прозорості, треба виконати команду npx cypress run automation.

6 ВХІДНІ ДАНІ

До вхідних даних відносяться:

- фрактальна фігура;
- кількість ітерацій, для отримання відповідного ступеню розвитку фракталу;
- колір фракталу;
- поворот фігури;
- прозорість матеріалу
- колір світла;
- яскравість світла;
- координати розташування фрактали, джерела світла та напрямку променів.

Дані вводяться за допомогою миші та клавіатури.

7 ВИХІДНІ ДАНІ

В ході роботи програми з генерації тривимірних фракталів відповідно генерується сцена з моделлю фракталу та зберігаються застосовані конфігурації.

В ході роботи програми збору та розрахунків показників прозорості, отриманих від графічного програмного засобу зберігаються відповідні розрахунки та прозорості.

В ході роботи роботи програми автоматизованих скрипитів збору показників прозорості формуються зображення, які програма виконує на кожному кроці для можливості відстеження поведінки та пошуку проблем у випадках, коли програма неочікувано завершує своє виконання.

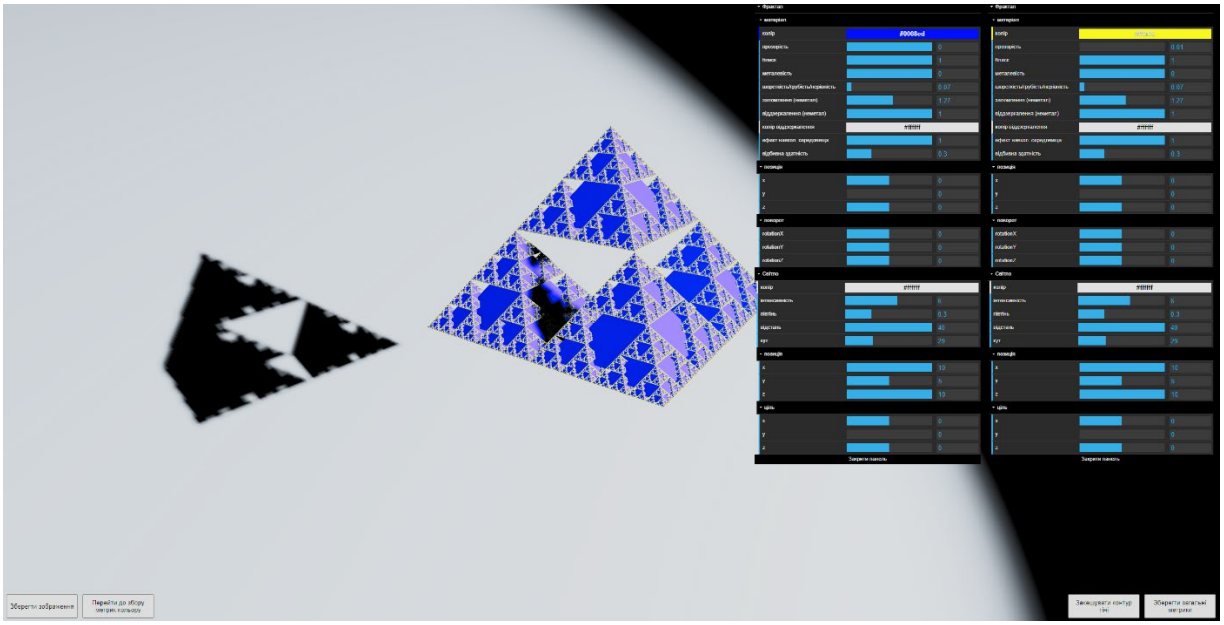


Рисунок 7.1 – Інтерфейс програми з гереації тривимірних фракталів

8 ОПИС ІНТЕРФЕЙСУ КОРИСТУВАЧА

Після запуску програми FractalGenerator (який за замовчуванням знаходиться в папці «C:\Program Files\FractalGenerator») на екрані з’являється браузер з посиланням програми із формою (рис. 8.1), для вказання початкових даних для генерації тривимірної сцени.

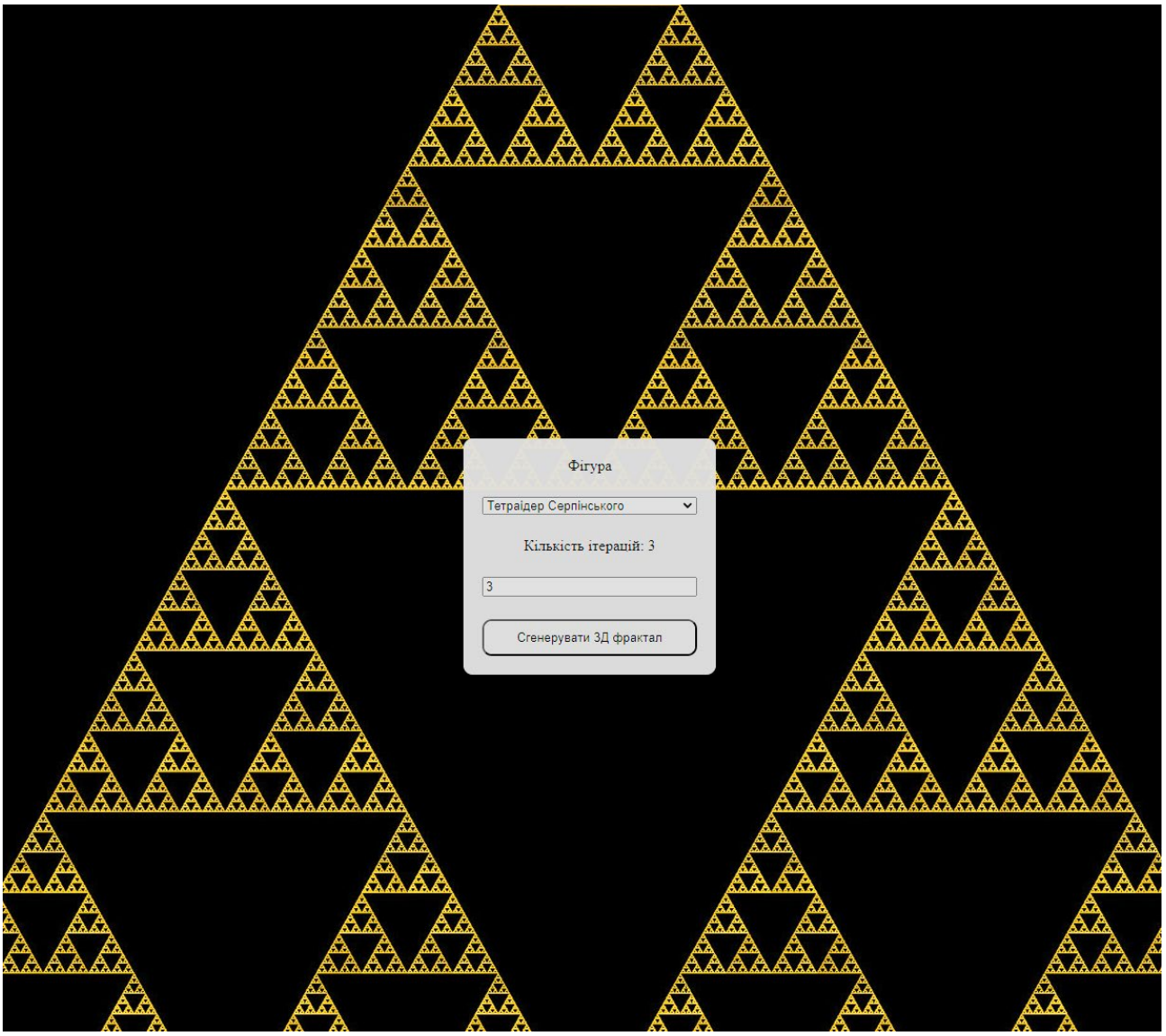


Рисунок 8.1 – Початкова форма для генерації тривимірної сцени з моделлю фракталу

Для завершення програми необхідно закрити вкладку браузеру.

Далі користувач заповнює залишкові показники, безпосередньо на сгенерованій сцені, користуючись графічними елементами (рис. 8.2).

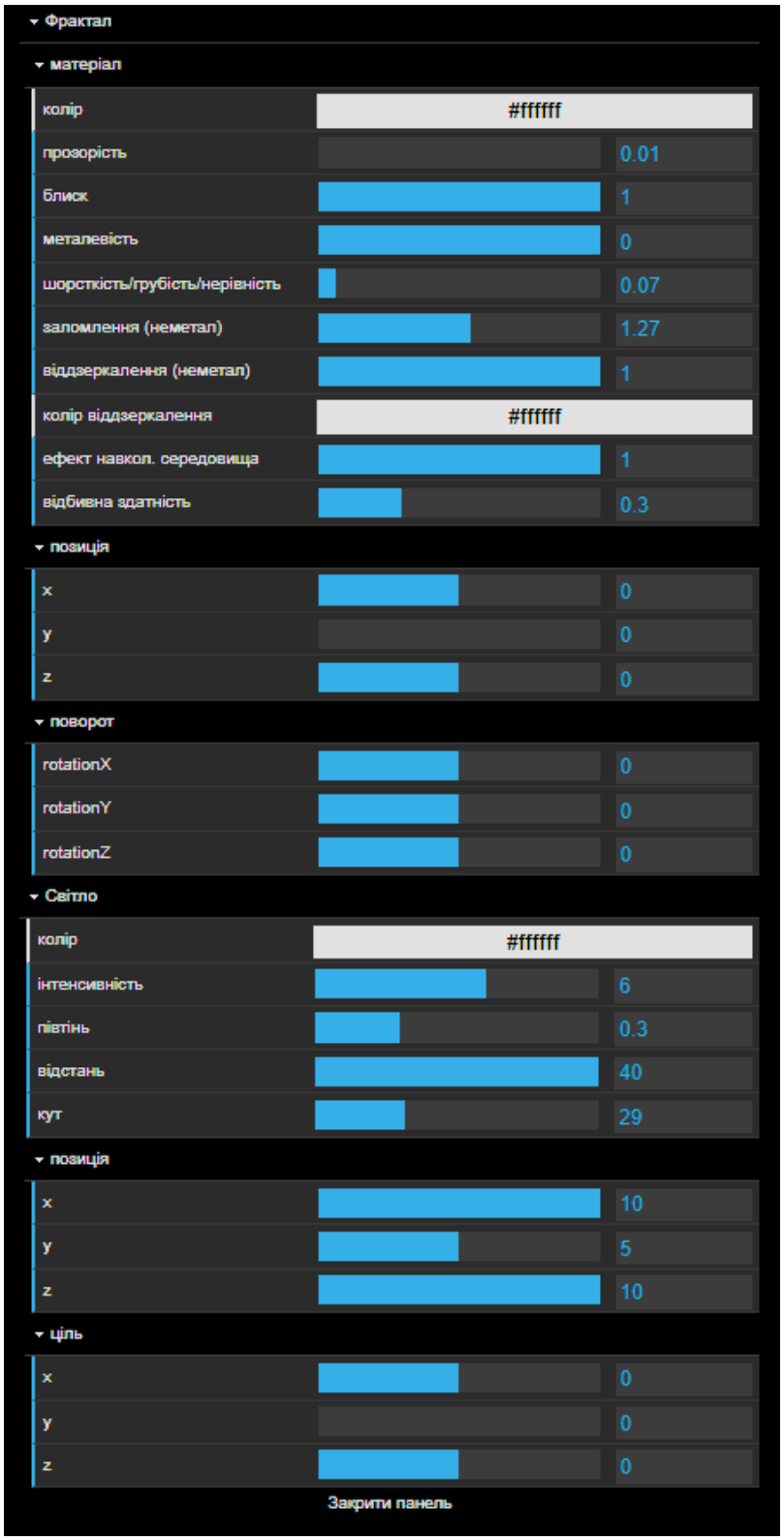


Рисунок 8.2 – Графічні елементи керуванням показниками фракталу та джерела освітлення

Для збереження треба натиснути на «Зберегти зображення» та «Перейти до аналізу показників прозорості» (рис. 8.3).

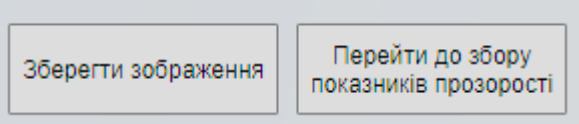


Рисунок 8.3 – Кнопки для збереження зображення, отриманого на тривимірній сцені та відкриттям додатку для розрахунків показників прозорості

Після запуску програми збору та розрахунків показників прозорості, отриманих від графічного програмного засобу, відкривається відповідне посилання у браузері (рис. 8.4).

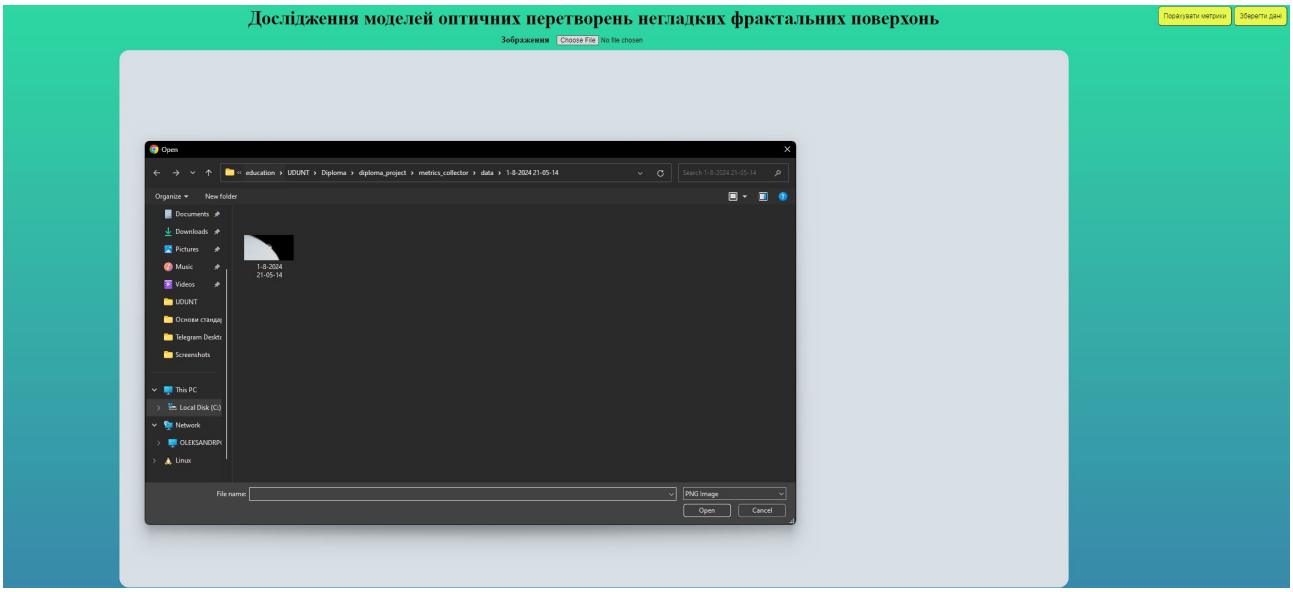


Рисунок 8.4 – Програма збору та розрахунків показників прозорості, отриманих від графічного програмного засобу

Для виконання розрахунків та збереження даних треба натиснути відповідні кнопки (рис. 8.5).

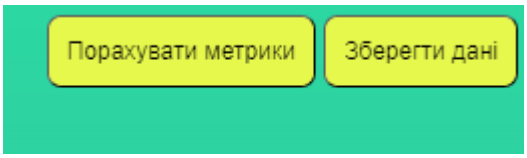


Рисунок 8.3 – Кнопки для виконання розрахунків прозорості фігури та збереження даних

Для закриття програми необхідно закрити вкладку у браузері.

9 ПОРЯДОК РОБОТИ З ПРОГРАМОЮ

В табл. 1 приведений порядок роботи з програмою.

Табл. 9.1 Порядок роботи з програмою

Актор	Етап	Час початку	Час закінчення
Користувач	Потреба в отриманні початкової сцени з фракталом	8:00	8:10
Користувач	Запуск програми	8:11	8:15
Користувач	Очікування завантаження	8:16	8:19
Користувач	Обрання елементу фракталу	8:20	8:24
Користувач	Встановлення ступеню розвитку фраталу	8:25	8:28
Користувач	Встановлення матеріалу прозорості	8:29	8:31
Користувач	Введення значення кута повороту	8:32	8:34
Користувач	Збереження отриманих результатів	8:35	8:36
Користувач	Завершення роботи (вихід)	8:37	8:38

10 ПОВІДОМЛЕННЯ

В табл. 1 приведені повідомлення користувачу в залежності від ситуацій, що виникли, а також рекомендовані дії.

Табл. 10.1 Повідомлення користувачу

Текст повідомлення	Опис ситуації	Рекомендовані дії
Помилка великої кількості ітерацій	Введене високе значення ступеню розвитку фракталу. Рекомендоване максимальне значення – 8 ітерацій.	Перезапустити програму або повторити спробу

ЗАТВЕРДЖЕНО

1116130.01346-01 12 01

ПРОГРАМНІ ЗАСОБИ ДОСЛІДЖЕННЯ МОДЕЛЕЙ ОПТИЧНИХ ПЕРЕТВОРЕНЬ НЕГЛАДКИХ ФРАКТАЛЬНИХ ПОВЕРХОНЬ

Текст програми

1116130.01346-01 12 01

Листів 31

2024

Перелік програмних модулів:

1 ТЕКСТ ОСНОВНОЇ ПРОГРАМИ – ГРАФІЧНОГО ДОДАТКУ

1.1 Конфігураційна частина серверної програми

Модуль: src/server/tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2019",
    "module": "commonjs",
    "outDir": "../../dist/server",
    "esModuleInterop": true
  },
  "include": ["**/*.ts"]
}
```

Модуль: src/client/package.json

```
{
 "version": "1.0.0",
 "description": "",
 "scripts": {
  "build": "webpack --config ./src/client/webpack.prod.js",
  "dev": "webpack serve --config ./src/client/webpack.dev.js",
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "node ./dist/server/server.js",
  "cypress:open": "cypress open"
 },
 "devDependencies": {
  "@types/dat.gui": "^0.7.7",
  "@types/express": "^4.17.13",
  "@types/node": "^17.0.23",
  "@types/three": "^0.150.1",
  "cypress": "^13.6.0",
  "dat.gui": "^0.7.9",
  "file-loader": "^6.2.0",
  "three": "^0.151.3",
  "ts-loader": "^9.2.8",
  "typescript": "^5.0.4",
  "url-loader": "^4.1.1",
  "webpack": "^5.70.0",
  "webpack-cli": "^4.9.2",
  "webpack-dev-server": "^4.7.4",
  "webpack-merge": "^5.8.0"
 },
 "dependencies": {
```

```
    "express": "^4.17.3"
  }
}
```

1.2 Серверна частина програми

Модуль: dist/server.js

```
 "use strict";
var __importDefault = (this && this.__importDefault) || function
(mod) {
    return (mod && mod.__esModule) ? mod : { "default": mod };
};
Object.defineProperty(exports, "__esModule", { value: true });
const express_1 = __importDefault(require("express"));
const path_1 = __importDefault(require("path"));
const http_1 = __importDefault(require("http"));
const port = 3000;
class App {
    constructor(port) {
        // to use this server.ts
        // # npm run build        (this creates the production version of
bundle.js and places it in ./dist/client/)
        // # tsc -p ./src/server  (this compiles ./src/server/server.ts into
./dist/server/server.js)
        // # npm start            (this starts nodejs with express and serves
the ./dist/client folder)
        //
        // visit http://127.0.0.1:3000
        this.server = new http_1.default.Server(app);
    }
    Start() {
        this.server.listen(this.port, () => {
            console.log(`Server listening on port ${this.port}.`);
        });
    }
}
new App(port).Start();
```

Модуль: src/server/server.ts

```
 import express from 'express'
import path from 'path'
import http from 'http'

const port: number = 3000

// # npm run build        (this creates the production version of
bundle.js and places it in ./dist/client/)
// # tsc -p ./src/server  (this compiles ./src/server/server.ts into
./dist/server/server.js)
// # npm start            (this starts nodejs with express and serves the
./dist/client folder)
// visit http://127.0.0.1:3000

class App {
    private server: http.Server
    private port: number

    constructor(port: number) {
        this.port = port
        const app = express()
        app.use(express.static(path.join(__dirname, '../client')));

        this.server = new http.Server(app)
    }

    public start() {
        this.server.listen(this.port, () => {
            console.log(`Server listening on port ${this.port}.`)
        })
    }
}

new App(port).start()
```

1.3 Конфігураційна частина клієнтської програми

Модуль: src/client/tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES6",
    "moduleResolution": "node",
    "strict": false
  },
  "include": ["**/*.ts"]
}
```

Модуль: src/client/webpack.common.js

```
 const path = require('path');

module.exports = {
    entry: './src/client/client.ts',
    module: {
        rules: [
            {
                test: /\.tsx?$/,
                use: 'ts-loader',
                exclude: /node_modules/,
            },
            {
                test: /\.(png|jpg|gif|env|glb|gltf|stl|hdr)$/i,
                use: [
                    {
                        loader: 'url-loader',
                        options: {
                            limit: 8192,
                        },
                    },
                ],
            },
        ],
    },
    resolve: {
        alias: {
            three: path.resolve('./node_modules/three')
        },
        extensions: ['.tsx', '.ts', '.js'],
    },
    output: {
        filename: 'bundle.js',
        path: path.resolve(__dirname, '../../dist/client'),
    }
};
```

src/client/webpack.dev.js

```
const { merge } = require('webpack-merge')
const common = require('./webpack.common.js')
const path = require('path');
```

```
module.exports = merge(common, {
   mode: 'development',
   devtool: 'eval-source-map',
   devServer: {
      static: {
         directory: path.join(__dirname, '../../dist/client'),
      },
      hot: true,
   },
})
```

Модуль: src/client/webpack.prod.js

```
const { merge } = require('webpack-merge');
const common = require('./webpack.common.js');

module.exports = merge(common, {
   mode: 'production',
   performance: {
      hints: false
   }
});
```

Модуль: src/client/utils/configuration.ts

```
 import { Point } from "../algorithm/baseGeometryUtils";

export type FigureType = 'serpinsiy_tetraider' |
'serpinsiy_tiered_tetraider' | 'custom_tetraider';

export const initConfiguration = () => {
   if (!window['configs']) {
      window['configs'] = {
         iterations: 1,
         figureType: 'serpinsiy_tetraider',
      }
   }
};

export const getIterationsCount = () => window['configs'].iterations
?? 1;

export const getFigureType = () => window['configs'].figureType ??
'serpinsiy_tetraider';

export const setIterationsCount = (count: number) => {
   window['configs'].iterations = count;
};

export const setFigureType = (type: FigureType) => {
   window['configs'].figureType = type;
};

export const setPoints = (a: Point, b: Point, c: Point, d: Point) => {
   window['points'] = { a, b, c, d };
};

export const isSerpinskiyTetraider = () => {
   return window['configs'].figureType === 'serpinsiy_tetraider';
};

export const isTieredSerpinskiyTetraider = () => {
   return window['configs'].figureType ===
'serpinsiy_tiered_tetraider';
};

export const isCustomTetraider = () => {
   return window['configs'].figureType === 'custom_tetraider';
};
```

1.4 Клієнська частина

Модуль: index.html:

```
 <!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="utf-8" />
      <meta name="viewport" content="width=device-width, initial-
scale=1" />
      <title>Fractal discovering</title>
      <style>
         body {
            overflow: hidden;
            margin: 0px;
            background: url("./sierpinski-triangle.jpg") center/cover
no-repeat fixed;
         }

         #preparation-container {
            display: grid;
            align-items: center;
            justify-content: center;
            margin-top: 20%;
         }

         #preparation-form {
            display: grid;
            align-items: center;
            justify-content: center;
            gap: 1.5rem;
            background: rgba(255, 255, 255, 0.9);
            padding: 20px;
            border-radius: 10px;
            text-align: center;
         }

         #generate-fractal-button {
            padding: 10px;
            border-radius: 10px;
         }

         #progress-container {
            width: 100%;
            text-align: center;
            padding-top: 3rem;
         }

         #progress-bar {
            height: 50px;
            width: 100%;
         }
      </style>
   </head>

   <body>
      <div id="preparation-container">
         <form id="preparation-form">
            <label for="figure-type">Фігура</label>
            <select name="figure-type" id="figure-type">
               <option value="serpinsiy_tetraider">Тетраідер
Серпінського</option>
               <option value="serpinsiy_tiered_tetraider">Рівневий
тетарідер Серпінського</option>
               <option value="custom_tetraider">Кастомний
тетраідер</option>
            </select>
            <label for="iterations">Кількість ітерацій: <span
id="range-value">1</span></label>
            <!-- <input type="range" id="range" min="1" max="8"
```

```
step="1" value="1"> -->
            <input id="range">
            <button type="button" id="generate-fractal-
button">Згенерувати 3Д фрактал</button>
          </form>
          <div id="progress-container" hidden>
            <progress id="progress-bar" max="100"
value="0"></progress>
          </div>
        </div>
        <div id="scene-container" hidden></div>
        <script type="module" src="bundle.js"></script>
    </body>
</html>
```

Модуль: src/client/client.ts

```
 import { getScene } from './scene';
import { getCamera } from './camera';
import { getRenderer } from './renderer';
import { initializeOrbitControls } from
'./guicontrols/initializeOrbitControls';
import { initActionButtons, initConfiguration, setIterationsCount,
setFigureType } from './utils';

const GENERATION_TIMEOUT = 1;

const initializeScene = () => {
  const scene = getScene();
  const camera = getCamera();
  const renderer = getRenderer();

  initializeOrbitControls(camera, renderer);

  document.getElementById('scene-
container').appendChild(renderer.domElement);

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

  renderer.setAnimationLoop(()=>{
    renderer.render(scene, camera);
  });

  initActionButtons(renderer.domElement);
};

const simulateProgress = () => {
  const progressBar: any = document.getElementById('progress-bar');
  const interval = 10;
  const max = 100;
  let value = 0;
  const increment = max / (GENERATION_TIMEOUT / interval);

  document.getElementById('progress-
container').removeAttribute('hidden');

  const progressInterval = setInterval(() => {
     value += increment;
     progressBar.value = value;

     if (value >= max) {
        progressBar.value = max;
        clearInterval(progressInterval);
     }
  }, interval);
};

const startSimulationProgressFlow = () => {
  document.getElementById('generate-fractal-
button').setAttribute('disabled', "true");
  simulateProgress();

  setTimeout(() => {
    const preparationContainer =
document.getElementById('preparation-container');
    const sceneContainer = document.getElementById('scene-
container');

    preparationContainer.style.display = 'none';
    document.body.style.background = 'none';
    sceneContainer.removeAttribute('hidden');

    initializeScene();
  }, GENERATION_TIMEOUT);
};

initConfiguration();

const rangeInput: any = document.getElementById('range');
rangeInput.addEventListener('input', () => {
  document.getElementById('range-value').textContent =
rangeInput.value;
  setIterationsCount(rangeInput.value);
});

const figureTypeSelect: any = document.getElementById('figure-
type');
figureTypeSelect.addEventListener('change', () => {
  setFigureType(figureTypeSelect.value);
});

document.getElementById('generate-fractal-
button').addEventListener('click', startSimulationProgressFlow);
```

1.5 Алгоритми побудування точок для подальшого побудування тривимірних фракталних фігур

Модуль: src/client/algorithom/baseGeometryUtils.ts

```
 export class Point {
    x: number;
    y: number;
    z: number;

    constructor(x: number, y: number, z: number) {
        this.x = x;
        this.y = y;
        this.z = z;
    }
}

export class Pyramid {
    A: Point;
    B: Point;
    C: Point;
    D: Point;

    constructor(a: Point, b: Point, c: Point, d: Point) {
        this.A = a;
        this.B = b;
        this.C = c;
        this.D = d;
    }
}
```

```
export const getHalfPoint = (A: Point, B: Point): Point => {
   return new Point((A.x + B.x) / 2, (A.y + B.y) / 2, (A.z + B.z) / 2);
};

export const buildInitialPyramid = (points: number[][]) => {
   const [[ax, ay, az], [bx, by, bz], [cx, cy, cz], [dx, dy, dz]] = points;

   return new Pyramid(new Point(ax, ay, az), new Point(bx, by, bz),
new Point(cx, cy, cz), new Point(dx, dy, dz));
}
```

Модуль: src/client/algorithom/polygonPyramid.ts

```
 import { getIterationsCount } from '../utils';
import { Point, Pyramid, getHalfPoint, buildInitialPyramid } from
'./baseGeometryUtils';
import {
   DEFAULT_POLIGON_PYRAMID_POINTS,
   POLIGON_PYRAMID_POINTS_OPTION_2,
   POLIGON_PYRAMID_POINTS_OPTION_3
} from '../constants/pyramidPoints';

const createNewFigures = (figure: Pyramid): Pyramid[] => {
   const AB = getHalfPoint(figure.A, figure.B);
   const AC = getHalfPoint(figure.A, figure.C);
   const BC = getHalfPoint(figure.B, figure.C);

   const coefficientA = 0.5;
   const ABC_center = new Point(
      (AB.x + figure.C.x * coefficientA) / (1 + coefficientA),
      (AB.y + figure.C.y * coefficientA) / (1 + coefficientA),
      (AB.z + figure.C.z * coefficientA) / (1 + coefficientA)
   );
   const ABD_center = new Point(
      (AB.x + figure.D.x * coefficientA) / (1 + coefficientA),
      (AB.y + figure.D.y * coefficientA) / (1 + coefficientA),
      (AB.z + figure.D.z * coefficientA) / (1 + coefficientA)
   );
   const ACD_center = new Point(
      (AC.x + figure.D.x * coefficientA) / (1 + coefficientA),
      (AC.y + figure.D.y * coefficientA) / (1 + coefficientA),
      (AC.z + figure.D.z * coefficientA) / (1 + coefficientA)
   );
   const BCD_center = new Point(
      (BC.x + figure.D.x * coefficientA) / (1 + coefficientA),
      (BC.y + figure.D.y * coefficientA) / (1 + coefficientA),
      (BC.z + figure.D.z * coefficientA) / (1 + coefficientA)
   );

   const coefficientB = 2;
   const Ai = new Point(
      (coefficientB * BCD_center.x - figure.A.x),
      (coefficientB * BCD_center.y - figure.A.y),
      (coefficientB * BCD_center.z - figure.A.z),
   );
   const Bi = new Point(
      (coefficientB * ACD_center.x - figure.B.x),
      (coefficientB * ACD_center.y - figure.B.y),
      (coefficientB * ACD_center.z - figure.B.z),
   );
   const Ci = new Point(
      (coefficientB * ABD_center.x - figure.C.x),
      (coefficientB * ABD_center.y - figure.C.y),
      (coefficientB * ABD_center.z - figure.C.z),
   );
   const Di = new Point(
      (coefficientB * ABC_center.x - figure.D.x),
      (coefficientB * ABC_center.y - figure.D.y),
      (coefficientB * ABC_center.z - figure.D.z),
   );

   const A_B_C_Di = new Pyramid(figure.A, figure.B, figure.C, Di);
   const A_B_Ci_D = new Pyramid(figure.A, figure.B, Ci, figure.D);
   const A_Bi_C_D = new Pyramid(figure.A, Bi, figure.C, figure.D);
   const Ai_B_C_D = new Pyramid(Ai, figure.B, figure.C, figure.D);

   return [A_B_C_Di, A_B_Ci_D, A_Bi_C_D, Ai_B_C_D]
};

export const generatePyramids = (i, figures): Pyramid[] => {
   const figureList = [...figures];

   while (i > 0) {
      const tempList = [];

      while (figureList.length) {
         const currentFigure = figureList.pop();
         const newFigures = createNewFigures(currentFigure);

         tempList.push(...newFigures);
      }

      figureList.push(...tempList);
      i -= 1;
   }

   return figureList;
};

export const getPyramids = () => generatePyramids(
   getIterationsCount(),

[buildInitialPyramid(DEFAULT_POLIGON_PYRAMID_POINTS)]
 );
```

Модуль: src/client/algorithom/simplePyramid.ts

```
 import { getIterationsCount } from '../utils';
import { Pyramid, getHalfPoint, buildInitialPyramid } from
'./baseGeometryUtils';
import { DEFAULT_SIMPLE_PYRAMID_POINTS } from
'../constants/pyramidPoints';

const createNewFigures = (figure: Pyramid): Pyramid[] => {
  const AB = getHalfPoint(figure.A, figure.B);
  const AC = getHalfPoint(figure.A, figure.C);
  const AD = getHalfPoint(figure.A, figure.D);
  const BC = getHalfPoint(figure.B, figure.C);
  const BD = getHalfPoint(figure.B, figure.D);
  const CD = getHalfPoint(figure.C, figure.D);
  const A_AB_AC_AD = new Pyramid(figure.A, AB, AC, AD);
  const B_AB_BC_BD = new Pyramid(figure.B, AB, BC, BD);
  const C_AC_BC_CD = new Pyramid(figure.C, AC, BC, CD);
  const D_AD_BD_CD = new Pyramid(figure.D, AD, BD, CD);

  return [A_AB_AC_AD, B_AB_BC_BD, C_AC_BC_CD,
D_AD_BD_CD];
};

export const generatePyramids = (i, figures): Pyramid[] => {
  const figureList = [...figures];

  while (i > 0) {
    const tempList = [];

    while (figureList.length) {
      const currentFigure = figureList.pop();
      const newFigures = createNewFigures(currentFigure);

      tempList.push(...newFigures);
```

```
    }

    figureList.push(...tempList);
    i -= 1;
  }

  return figureList;
};

export const getPyramids = () => generatePyramids(
  getIterationsCount(),
  [buildInitialPyramid(DEFAULT_SIMPLE_PYRAMID_POINTS)]
);
```

Модуль: src/client/algorithom/tieredPyramid.ts

```
 import { getIterationsCount } from '../utils';
import { Point, Pyramid, getHalfPoint, buildInitialPyramid } from
'./baseGeometryUtils';
import { DEFAULT_TIERED_PYRAMID_POINTS } from
'../constants/pyramidPoints';

const getMinimizedPyramids = (figure: Pyramid) => {
    const AB = getHalfPoint(figure.A, figure.B);
    const AC = getHalfPoint(figure.A, figure.C);
    const BC = getHalfPoint(figure.B, figure.C);

    const coefficientA = 0.5;
    const ABC_center = new Point(
        (AB.x + figure.C.x * coefficientA) / (1 + coefficientA),
        (AB.y + figure.C.y * coefficientA) / (1 + coefficientA),
        (AB.z + figure.C.z * coefficientA) / (1 + coefficientA)
    );
    const ABD_center = new Point(
        (AB.x + figure.D.x * coefficientA) / (1 + coefficientA),
        (AB.y + figure.D.y * coefficientA) / (1 + coefficientA),
        (AB.z + figure.D.z * coefficientA) / (1 + coefficientA)
    );
    const ACD_center = new Point(
        (AC.x + figure.D.x * coefficientA) / (1 + coefficientA),
        (AC.y + figure.D.y * coefficientA) / (1 + coefficientA),
        (AC.z + figure.D.z * coefficientA) / (1 + coefficientA)
    );
    const BCD_center = new Point(
        (BC.x + figure.D.x * coefficientA) / (1 + coefficientA),
        (BC.y + figure.D.y * coefficientA) / (1 + coefficientA),
        (BC.z + figure.D.z * coefficientA) / (1 + coefficientA)
    );

    const coefficientB = 0.3;
    const Ai = new Point(
        ((figure.A.x + BCD_center.x * coefficientB) / (1 +
coefficientB)),
        ((figure.A.y + BCD_center.y * coefficientB) / (1 +
coefficientB)),
        ((figure.A.z + BCD_center.z * coefficientB) / (1 +
coefficientB))
    );
    const Bi = new Point(
        ((figure.B.x + ACD_center.x * coefficientB) / (1 +
coefficientB)),
        ((figure.B.y + ACD_center.y * coefficientB) / (1 +
coefficientB)),
        ((figure.B.z + ACD_center.z * coefficientB) / (1 +
coefficientB))
    );
    const Ci = new Point(
        ((figure.C.x + ABD_center.x * coefficientB) / (1 +
coefficientB)),
        ((figure.C.y + ABD_center.y * coefficientB) / (1 +
coefficientB)),
        ((figure.C.z + ABD_center.z * coefficientB) / (1 +
coefficientB))
    );
    const Di = new Point(
        ((figure.D.x + ABC_center.x * coefficientB) / (1 +
coefficientB)),
        ((figure.D.y + ABC_center.y * coefficientB) / (1 +
coefficientB)),
        ((figure.D.z + ABC_center.z * coefficientB) / (1 +
coefficientB))
    );

    return new Pyramid(Ai, Bi, Ci, Di);
}

const createNewFigures = (figure: Pyramid): [Pyramid[], Pyramid[]]
=> {
    const AB = getHalfPoint(figure.A, figure.B);
    const AC = getHalfPoint(figure.A, figure.C);
    const AD = getHalfPoint(figure.A, figure.D);
    const BC = getHalfPoint(figure.B, figure.C);
    const BD = getHalfPoint(figure.B, figure.D);
    const CD = getHalfPoint(figure.C, figure.D);
    const A_AB_AC_AD = new Pyramid(figure.A, AB, AC, AD);
    const B_AB_BC_BD = new Pyramid(figure.B, AB, BC, BD);
    const C_AC_BC_CD = new Pyramid(figure.C, AC, BC, CD);
    const D_AD_BD_CD = new Pyramid(figure.D, AD, BD, CD);

    const i_A_AB_AC_AD =
getMinimizedPyramids(A_AB_AC_AD);
    const i_B_AB_BC_BD =
getMinimizedPyramids(B_AB_BC_BD);
    const i_C_AC_BC_CD =
getMinimizedPyramids(C_AC_BC_CD);
    const i_D_AD_BD_CD =
getMinimizedPyramids(D_AD_BD_CD);

    return [[A_AB_AC_AD, B_AB_BC_BD, C_AC_BC_CD,
D_AD_BD_CD], [i_A_AB_AC_AD, i_B_AB_BC_BD,
i_C_AC_BC_CD, i_D_AD_BD_CD]];
};

export const generatePyramids = (i, figures): [Pyramid[], Pyramid[]]
=> {
    const figureList = [...figures];
    const innerListAll = [];

    while (i > 0) {
        const tempList = [];

        while (figureList.length) {
            const currentFigure = figureList.pop();
            const [outerList, innerList] =
createNewFigures(currentFigure);
            innerListAll.push(...innerList);

            tempList.push(...outerList);
        }

        figureList.push(...tempList);
        i -= 1;
    }

    return [figureList, innerListAll];
};

export const getPyramids = () => generatePyramids(
    getIterationsCount(),

[buildInitialPyramid(DEFAULT_TIERED_PYRAMID_POINTS)]
);
```

1.6 Налаштування камери

Модуль: src/client/camera/camera.ts

```
import { PerspectiveCamera } from 'three';
import { isCustomTetraider } from '../utils';

export const getCamera = () => {
  const camera = new PerspectiveCamera(60, innerWidth /
innerHeight, 0.1, 1000);

  if (isCustomTetraider()) {
    camera.position.set(-10, 7, 5);
  } else {
    camera.position.set(-3, 1, 4);
  }

  return camera;
};
```

Модуль: src/client/constants/pyramidPoints.ts

```
 export const DEFAULT_SIMPLE_PYRAMID_POINTS = [
    [1, 0, 0],
    [0, 0, 1],
    [-1, 0, 0],
    [0, 1, 0]
];

export const DEFAULT_TIERED_PYRAMID_POINTS = [
    [1, 0, 0],
    [0, 0, 1],
    [-1, 0, 0],
    [0, 1, 0]
];

export const DEFAULT_POLIGON_PYRAMID_POINTS = [
    [0, 0, 0],
    [1, 0, 0],
    [0, 0, 1],
    [0.5, 1, 0.5]
];

export const POLIGON_PYRAMID_POINTS_OPTION_2 = [
    [0, 0, 0],
    [0, 0, 1],
    [0, 1, 0],
    [1, 0, 0]
];

export const POLIGON_PYRAMID_POINTS_OPTION_3 = [
    [1, -1/Math.sqrt(3), -1/Math.sqrt(6)],
    [-1, -1/Math.sqrt(3), -1/Math.sqrt(6)],
    [0, 2/Math.sqrt(3), -1/Math.sqrt(6)],
    [0, 0, 3/Math.sqrt(6)]
];
```

Модуль: src/client/geometry/Angle.ts

```
 import { BufferGeometry, Float32BufferAttribute, Vector3 } from
'three'

class Angle extends BufferGeometry {
    private parameters: {
        depthSegments: number;
        depth: number;
        width: number;
        heightSegments: number;
        widthSegments: number;
        height: number
    }

    constructor( width = 1, height = 1, depth = 1, widthSegments = 1,
heightSegments = 1, depthSegments = 1 ) {
        super();

        this.parameters = {
            width: width,
            height: height,
            depth: depth,
            widthSegments: widthSegments,
            heightSegments: heightSegments,
            depthSegments: depthSegments
        };

        const scope = this;

        // segments
        widthSegments = Math.floor( widthSegments );
        heightSegments = Math.floor( heightSegments );
        depthSegments = Math.floor( depthSegments );

        // buffers
        const indices = [];
        const vertices = [];
        const normals = [];
        const uvs = [];

        // helper variables
        let numberOfVertices = 0;
        let groupStart = 0;

        // build each side of the box geometry
        buildPlane( 'z', 'y', 'x', 1, - 1, depth, height, - width,
depthSegments, heightSegments, 1 ); // nx
        buildPlane( 'x', 'z', 'y', 1, - 1, width, depth, - height,
widthSegments, depthSegments, 3 ); // ny
        buildPlane( 'x', 'y', 'z', - 1, - 1, width, height, - depth,
widthSegments, heightSegments, 5 ); // nz

        // build geometry
        this.setIndex( indices );
        this.setAttribute( 'position', new Float32BufferAttribute(
vertices, 3 ) );
        this.setAttribute( 'normal', new Float32BufferAttribute(
normals, 3 ) );
        this.setAttribute( 'uv', new Float32BufferAttribute( uvs, 2 ) );

        function buildPlane( u, v, w, udir, vdir, width, height, depth,
gridX, gridY, materialIndex ) {
            const segmentWidth = width / gridX;
            const segmentHeight = height / gridY;

            const widthHalf = width / 2;
            const heightHalf = height / 2;
            const depthHalf = depth / 2;

            const gridX1 = gridX + 1;
            const gridY1 = gridY + 1;

            let vertexCounter = 0;
            let groupCount = 0;

            const vector = new Vector3();

            // generate vertices, normals and uvs
            for ( let iy = 0; iy < gridY1; iy ++ ) {
                const y = iy * segmentHeight - heightHalf;

                for ( let ix = 0; ix < gridX1; ix ++ ) {
                    const x = ix * segmentWidth - widthHalf;

                    // set values to correct vector component
                    vector[ u ] = x * udir;
                    vector[ v ] = y * vdir;
                    vector[ w ] = depthHalf;
```

```
            // now apply vector to vertex buffer
            vertices.push( vector.x, vector.y, vector.z );

            // set values to correct vector component
            vector[ u ] = 0;
            vector[ v ] = 0;
            vector[ w ] = depth > 0 ? 1 : - 1;

            // now apply vector to normal buffer
            normals.push( vector.x, vector.y, vector.z );

            // uvs
            uvs.push( ix / gridX );
            uvs.push( 1 - ( iy / gridY ) );

            // counters
            vertexCounter += 1;

          }
        }

        for ( let iy = 0; iy < gridY; iy ++ ) {
          for ( let ix = 0; ix < gridX; ix ++ ) {
            const a = numberOfVertices + ix + gridX1 * iy;
            const b = numberOfVertices + ix + gridX1 * ( iy + 1 );
            const c = numberOfVertices + ( ix + 1 ) + gridX1 * ( iy
+ 1 );
            const d = numberOfVertices + ( ix + 1 ) + gridX1 * iy;

            // faces
            indices.push( a, b, d );
            indices.push( b, c, d );

            groupCount += 6;

          }

        }

        // add a group to the geometry. this will ensure multi material
support
        scope.addGroup( groupStart, groupCount, materialIndex );

        // calculate new start value for groups
        groupStart += groupCount;

        // update total number of vertices
        numberOfVertices += vertexCounter;

      }

    }

    copy( source ) {
      super.copy( source );
      this.parameters = Object.assign( {}, source.parameters );

      return this;

    }

    static fromJSON( data ) {
      return new Angle( data.width, data.height, data.depth,
data.widthSegments, data.heightSegments, data.depthSegments );
    }

}

export { Angle };
```

1.7 Код графічного інтерфейсу з елементами для керування об’єктами на тривимірній сцені

src/client/guicontrols/ColorGUIHelper.ts

```
export class ColorGUIHelper {
  [x: string]: any;
  constructor(object, prop) {
    this.object = object;
    this.prop = prop;
  }
  get value() {
    return `#${this.object[this.prop].getHexString()}`;
  }
  set value(hexString) {
    this.object[this.prop].set(hexString);
  }
};
```

Модуль: src/client/guicontrols/DegRadHelper.ts

```
 import { MathUtils } from 'three';

export class DegRadHelper {
  obj: any;
  prop: any;

  constructor(obj, prop) {
    this.obj = obj;
    this.prop = prop;
  }
  get value() {
    return MathUtils.radToDeg(this.obj[this.prop]);
  }
  set value(v) {
    this.obj[this.prop] = MathUtils.degToRad(v);
  }
};
```

Модуль: src/client/guicontrols/initFractalGUI.ts

```
 import { GUI } from 'dat.gui';
import { BufferGeometry, Mesh, MeshPhysicalMaterial } from
'three';
import { makeXYZGUI } from './makeXYZGUI';
import { initializeFractalMaterialGUI } from
'./initializeFractalMaterialGUI';

export function makeRotationXYZGUI(gui: GUI, fractal:
Mesh<BufferGeometry, MeshPhysicalMaterial>, folderName: string)
{
    const folder = gui.addFolder(folderName);
    const rotationParameters = {
        rotationX: 0,
        rotationY: 0,
        rotationZ: 0
    };

    window['configs'].rotateX = 0;
    window['configs'].rotateY = 0;
    window['configs'].rotateZ = 0;
```

```
    folder.add(rotationParameters, 'rotationX', -Math.PI, Math.PI,
0.01).onChange((value) => {
        fractal.rotation.x = value;
        window['configs'].rotateX = value;
    });
    folder.add(rotationParameters, 'rotationY', -Math.PI, Math.PI,
0.01).onChange((value) => {
        fractal.rotation.y = value;
        window['configs'].rotateY = value;
    });
    folder.add(rotationParameters, 'rotationZ', -Math.PI, Math.PI,
0.01).onChange((value) => {
        fractal.rotation.z = value;
        window['configs'].rotateZ = value;
    });

    folder.open();
  };

export const initFractalGUI = (gui: GUI, folderName: string, fractal:
Mesh<BufferGeometry, MeshPhysicalMaterial>, depthMaterial) =>
{
    const folder = gui.addFolder(folderName);

    folder.open();

    initializeFractalMaterialGUI(folder, 'матеріал', fractal.material,
depthMaterial);
    makeXYZGUI(folder, fractal.position, 'позиція');
    makeRotationXYZGUI(folder, fractal, 'поворот');
};
```

Модуль: src/client/guicontrols/initGUIControls.ts

```
 import { GUI } from 'dat.gui';
import { initFractalGUI } from './initFractalGUI';
import { initializeLightGUI } from './initializeLightGUI';

GUI.TEXT_CLOSED = 'Закрити панель';
GUI.TEXT_OPEN = 'Відкрити панель';

export const initGUIControls = ({ spotLightA, fractal, depthMaterial
}) => {
    const gui = new GUI();

    gui.open();

    initFractalGUI(gui, 'Фрактал', fractal, depthMaterial);
    initializeLightGUI(gui, 'Світло', spotLightA.spotLight,
spotLightA.updateLight)

    gui.width = 475;
}
```

Модуль: src/client/guicontrols/initializeFractalMaterialGUI.ts

```
 import { GUI } from 'dat.gui';
import { ColorGUIHelper } from './ColorGUIHelper';

export const initializeFractalMaterialGUI = (gui: GUI, folderName:
string, fractalMaterial, depthMaterial) => {
    const folder = gui.addFolder(folderName);

    folder.addColor(new ColorGUIHelper(fractalMaterial, 'color'),
'value').name('колір');
    folder.add(fractalMaterial, 'transmission', 0.01, 1,
0.01).name('прозорість').onChange((value) => {
        depthMaterial.uniforms.opacity.value = 1 - value;
        window['configs'].transparency = value.toFixed(2);
    });
    folder.add(fractalMaterial, 'sheen', 0.01, 1, 0.01).name('блиск');
    folder.add(fractalMaterial, 'metalness', 0.01, 1,
0.01).name('металевість');
    folder.add(fractalMaterial, 'roughness', 0.01, 1,
0.01).name('шорсткість/грубість/нерівність');
    folder.add(fractalMaterial, 'ior', 0.01, 2.333,
0.01).name('заломлення (неметал)');
    folder.add(fractalMaterial, 'specularIntensity', 0.01, 1,
0.01).name('віддзеркалення (неметал)');
    folder.addColor(new ColorGUIHelper(fractalMaterial,
'specularColor'), 'value').name('колір віддзеркалення');
    folder.add(fractalMaterial, 'envMapIntensity', 0.01, 1,
0.01).name('ефект навкол. середовища');
    folder.add(fractalMaterial, 'reflectivity', 0.01, 1,
0.01).name('відбивна здатність');
    // folder.add(fractalMaterial.thickness, 'thickness', 0, 10);
    folder.open();
};
```

Модуль: src/client/guicontrols/initializeLightGUI.ts

```
 import { GUI } from 'dat.gui';
import { ColorGUIHelper } from './ColorGUIHelper';
import { DegRadHelper } from './DegRadHelper';
import { makeXYZGUI } from './makeXYZGUI';

export const initializeLightGUI = (gui: GUI, folderName: string,
spotLight, updateLight) => {
  const folder = gui.addFolder(folderName);

  folder.addColor(new ColorGUIHelper(spotLight, 'color'),
'value').name('колір');
  folder.add(spotLight, 'intensity', 0, 10, 0.01).name('інтенсивність');
  folder.add(spotLight, 'penumbra', 0, 1, 0.01).name('півтінь');
  folder.add(spotLight, 'distance', 0,
40).onChange(updateLight).name('відстань');
  folder.add(new DegRadHelper(spotLight, 'angle'), 'value', 0,
90).name('кут').onChange(updateLight);

  folder.open();

  makeXYZGUI(folder, spotLight.position, 'позиція', updateLight);
  makeXYZGUI(folder, spotLight.target.position, 'ціль',
updateLight);
};
```

Модуль: src/client/guicontrols/initializeOrbitControls.ts

```
 import { OrbitControls } from
'three/examples/jsm/controls/OrbitControls'

export const initializeOrbitControls = (camera, renderer) => {
  const controls = new OrbitControls(camera, renderer.domElement);

  return controls;
}
```

1.8 Налаштування світла

src/client/guicontrols/makeXYZGUI.ts

```
 export function makeXYZGUI(gui, vector3, name, onChangeFn = ()
=> {}) {
  const folder = gui.addFolder(name);
  folder.add(vector3, 'x', -10, 10).onChange(onChangeFn);
  folder.add(vector3, 'y', 0, 10).onChange(onChangeFn);
  folder.add(vector3, 'z', -10, 10).onChange(onChangeFn);

  folder.open();
};
```

Модуль: src/client/light/basic.ts

```
 import { AmbientLight, DirectionalLight, ColorRepresentation }
from 'three';

export const getAmbientLight = (color?: ColorRepresentation) =>
new AmbientLight(color = 0xffffff);

export const getDirectionalLight = (color?: ColorRepresentation,
intensity?: number) =>  (
  new DirectionalLight(color = 0xffffff, intensity = 0.3)
);
```

Модуль: src/client/light/spot.ts

```
 import { SpotLight, SpotLightHelper, Color, Mesh } from 'three'

interface SpotLightParams {
  color: Color;
  intensity: number;
  distance: number;
  angle: number;
  penumbra?: number;
  position: { x: number; y: number; z: number; };
  objectToLookAt: Mesh;
}

export const getSpotLight = (
    { color, intensity, distance, angle, position, penumbra,
objectToLookAt }: SpotLightParams
) => {
  const spotLight = new SpotLight();

  spotLight.color = color;
  spotLight.intensity = intensity;
  spotLight.distance = distance;
  spotLight.angle = angle;
  spotLight.penumbra = penumbra ?? 0;
  spotLight.position.set(position.x, position.y, position.z);
  spotLight.castShadow = true;
  spotLight.shadow.needsUpdate = true;
  spotLight.shadow.mapSize.width = 1024;
  spotLight.shadow.mapSize.height = 1024;
  spotLight.shadow.camera.near = 0.5;
  spotLight.shadow.camera.far = 50;
  spotLight.shadow.camera.fov = 20;
  spotLight.shadow.camera.aspect = 1;
  spotLight.shadow.camera.position.copy(spotLight.position);
  spotLight.shadow.camera.lookAt(objectToLookAt.position);
  spotLight.shadow.bias = -0.001; // 0.0001

  const spotLightHelper = new SpotLightHelper(spotLight);

  const updateLight = () => {
    spotLight.target.updateMatrixWorld();
    spotLightHelper.update();
  }

  return {
    spotLight,
    spotLightHelper,
    updateLight
  };
};

export const getSpotLightHelper = (spotLight) => new
SpotLightHelper(spotLight);
```

1.9 Побудування фрактальних фігур на основі отриманих точок

Модуль: src/client/mesh/anglePlane.ts

```
 import { MeshPhysicalMaterial, Mesh, DoubleSide, BackSide }
from 'three'
import { Angle } from '../geometry/Angle'

interface PlaneProps {
    position: {
        x: number;
        y: number;
        z: number;
    }
    rotate?: {
        x?: number;
        y?: number;
    }
}

export const getAnglePlane = ({ position, rotate }: PlaneProps) => {
    const geometry = new Angle(5, 3, 5);
    const material = new MeshPhysicalMaterial({
        color: 0x6c7b8b,
        side: BackSide,
        metalness: 0.5,
        roughness: 1,
    });
    const mesh = new Mesh(geometry, material);

    mesh.position.set(position.x, position.y, position.z);

    if (rotate) {
        mesh.rotateX(rotate.x ?? 0);
        mesh.rotateY(rotate.y ?? 0);
    }

    material.shadowSide = DoubleSide;
    mesh.receiveShadow = true;

    geometry.computeVertexNormals();
    geometry.computeBoundingBox();
    geometry.computeBoundingSphere();

    return mesh;
}
```

Модуль: src/client/mesh/cellularNoise.ts

```
 import {
    MeshPhysicalMaterial,
    Mesh,
    DoubleSide,
    Color,
    BufferGeometry,
    Vector3,
```

```
  ShaderMaterial,
  Float32BufferAttribute,
} from 'three'

function createSeededRandom(seed) {
  const m = 4294967296;
  const a = 1664525;
  const c = 1013904223;
  let state = seed;

  return function () {
    state = (a * state + c) % m;
    return state / m;
  };
}

function generateRandomPoints(seed, gridSize, pointDensity) {
  const points = [];
  const random = createSeededRandom(seed);

  for (let z = 0; z < gridSize; z++) {
    for (let y = 0; y < gridSize; y++) {
      for (let x = 0; x < gridSize; x++) {
        for (let i = 0; i < pointDensity; i++) {
          const point = {
            x: x + random(),
            y: y + random(),
            z: z + random(),
          };
          points.push(point);
        }
      }
    }
  }

  return points;
}

function distance(a, b) {
  const dx = a.x - b.x;
  const dy = a.y - b.y;
  const dz = a.z - b.z;
  return Math.sqrt(dx * dx + dy * dy + dz * dz);
}

function worleyNoise(position, seed, gridSize, pointDensity) {
  const points = generateRandomPoints(seed, gridSize,
pointDensity);
  let minDistance = Infinity;

  for (const point of points) {
    const d = distance(position, point);
    if (d < minDistance) {
      minDistance = d;
    }
  }

  return minDistance;
}

// Generate Worley noise-based geometry
function generateWorleyNoiseGeometry(size, resolution, seed,
gridSize, pointDensity) {
  const positions = [];

  // Generate vertices based on Worley noise
  for (let z = 0; z < resolution; z++) {
    for (let y = 0; y < resolution; y++) {
      for (let x = 0; x < resolution; x++) {
        const nx = x / resolution - 0.5;
        const ny = y / resolution - 0.5;
        const nz = z / resolution - 0.5;

        const position = new Vector3(nx * size, ny * size, nz *
size);
        const noiseValue = worleyNoise(position, seed, gridSize,
pointDensity);

        if (noiseValue > 0.1) {
          positions.push(position.x, position.y, position.z);
        }
      }
    }
  }

  const geometry = new BufferGeometry();
  const positionAttribute = new Float32BufferAttribute(positions,
3);
  geometry.setAttribute("position", positionAttribute);

  return geometry;
}


interface GetCrocodileSkin {
  color: Color;
  sheen: number;
  transmission: number;
  metalness: number;
  roughness: number;
  ior: number;
  specularIntensity: number;
  specularColor: Color;
  envMapIntensity: number;
  reflectivity: number;
  thickness: number;
  customDepthMaterial: ShaderMaterial;
}

export const getCellularNoise = (params: GetCrocodileSkin) => {
  const size = 5;
  const resolution = 20;
  const seed = "your_seed_here";
  const gridSize = 5;
  const pointDensity = 1;

  const geometry = generateWorleyNoiseGeometry(size, resolution,
seed, gridSize, pointDensity);

  const material = new MeshPhysicalMaterial({
    // attenuationDistance: 0.1,
    clearcoat: 1,
    // premultipliedAlpha: true,
    sheen: 1,
    // depthWrite: false,
    color: params.color,
    transmission: params.transmission,
    metalness: params.metalness,
    roughness: params.roughness,
    ior: params.ior,
    specularIntensity: params.specularIntensity,
    specularColor: params.specularColor,
    side: DoubleSide,
    envMapIntensity: params.envMapIntensity,
    reflectivity: params.reflectivity,
  });
  material.thickness = params.thickness;
  material.needsUpdate = true;
  material.shadowSide = DoubleSide;

  const crocodileSkin = new Mesh(geometry, material);

  crocodileSkin.geometry.computeVertexNormals();
  crocodileSkin.geometry.computeBoundingBox();
  crocodileSkin.geometry.computeBoundingSphere();

  crocodileSkin.customDepthMaterial =
params.customDepthMaterial;
  crocodileSkin.receiveShadow = true;
  crocodileSkin.castShadow = true;
```

```
  crocodileSkin.position.y = 0.5;
  crocodileSkin.position.z = 0.5;
  crocodileSkin.scale.set(0.5, 0.5, 0.5);

  crocodileSkin.rotateY(2.15);

  return crocodileSkin;
};
```

Модуль: src/client/mesh/diamondSquare.ts

```
import {
  BufferAttribute,
  Color,
  DoubleSide,
  Mesh,
  MeshPhysicalMaterial,
  PlaneGeometry,
  ShaderMaterial,
  Vector3,
} from 'three'

function diamondSquare(geometry, size, maxIterations = 5) {
  const widthSegments = geometry.parameters.widthSegments;
  const heightSegments = geometry.parameters.heightSegments;

  const data = new Float32Array((widthSegments + 1) *
(heightSegments + 1));

  function set(x, y, value) {
    data[x + y * (widthSegments + 1)] = value;
  }

  function get(x, y) {
    return data[x + y * (heightSegments + 1)];
  }

  function midpointDisplacement(x, y, w, iter) {
    if (w < 2 || iter > maxIterations) return;

    const halfW = w / 2;
    const topLeft = get(x, y);
    const topRight = get(x + w, y);
    const bottomLeft = get(x, y + w);
    const bottomRight = get(x + w, y + w);

    const diamondValue = (topLeft + topRight + bottomLeft +
bottomRight) / 4;
    set(x + halfW, y + halfW, diamondValue + Math.random() * w
* 0.5);

    set(x + halfW, y, (topLeft + topRight + diamondValue) / 3 +
Math.random() * halfW * 0.5);
    set(x, y + halfW, (topLeft + bottomLeft + diamondValue) / 3 +
Math.random() * halfW * 0.5);
    set(x + w, y + halfW, (topRight + bottomRight +
diamondValue) / 3 + Math.random() * halfW * 0.5);
    set(x + halfW, y + w, (bottomLeft + bottomRight +
diamondValue) / 3 + Math.random() * halfW * 0.5);

    midpointDisplacement(x, y, halfW, iter + 1);
    midpointDisplacement(x + halfW, y, halfW, iter + 1);
    midpointDisplacement(x, y + halfW, halfW, iter + 1);
    midpointDisplacement(x + halfW, y + halfW, halfW, iter + 1);
  }

  midpointDisplacement(0, 0, widthSegments, 0);

  // Update the PlaneGeometry vertex positions
  const positionAttribute = geometry.attributes.position as
BufferAttribute;

  for (let i = 0; i < positionAttribute.count; i++) {
    const vertex = new
Vector3().fromBufferAttribute(positionAttribute, i);
    vertex.y = data[i];
    positionAttribute.setXYZ(i, vertex.x, vertex.y, vertex.z);
  }
}


interface GetCrocodileSkin {
  color: Color;
  sheen: number;
  transmission: number;
  metalness: number;
  roughness: number;
  ior: number;
  specularIntensity: number;
  specularColor: Color;
  envMapIntensity: number;
  reflectivity: number;
  thickness: number;
  customDepthMaterial: ShaderMaterial;
}

export const getDiamondSquare = (params: GetCrocodileSkin) => {
  const width = 200;
  const height = 200;
  const resolution = 256;
  const maxIterations = 7;
  const geometry = new PlaneGeometry(width, height, resolution,
resolution);
  geometry.rotateX(-Math.PI / 2);
  diamondSquare(geometry, resolution, maxIterations);

  const material = new MeshPhysicalMaterial({
    // attenuationDistance: 0.1,
    clearcoat: 1,
    // premultipliedAlpha: true,
    sheen: 1,
    // depthWrite: false,
    color: params.color,
    transmission: params.transmission,
    metalness: params.metalness,
    roughness: params.roughness,
    ior: params.ior,
    specularIntensity: params.specularIntensity,
    specularColor: params.specularColor,
    side: DoubleSide,
    envMapIntensity: params.envMapIntensity,
    reflectivity: params.reflectivity,
  });
  material.thickness = params.thickness;
  material.needsUpdate = true;
  material.shadowSide = DoubleSide;

  const crocodileSkin = new Mesh(geometry, material);

  crocodileSkin.geometry.computeVertexNormals();
  crocodileSkin.geometry.computeBoundingBox();
  crocodileSkin.geometry.computeBoundingSphere();

  crocodileSkin.customDepthMaterial =
params.customDepthMaterial;
  crocodileSkin.receiveShadow = true;
  crocodileSkin.castShadow = true;

  crocodileSkin.position.y = 0.5;
  crocodileSkin.position.z = 0.5;
  crocodileSkin.scale.set(0.5, 0.5, 0.5);

  crocodileSkin.rotateY(2.15);

  return crocodileSkin;
};
```

```
import {
  BufferAttribute,
  Color,
  DoubleSide,
  Mesh,
  MeshPhysicalMaterial,
  PlaneGeometry,
  ShaderMaterial,
  Vector3,
} from 'three'

function diamondSquareAlgorithm(geometry, iterations = 5) {
  const widthSegments = geometry.parameters.widthSegments;
  const heightSegments = geometry.parameters.heightSegments;
  const data = new Float32Array((widthSegments + 1) *
(heightSegments + 1));

  function set(x, y, value) {
    data[x + y * (widthSegments + 1)] = value;
  }

  function sample(x, y) {
    x = x % (widthSegments + 1);
    y = y % (heightSegments + 1);
    return data[x + y * (widthSegments + 1)];
  }

  function setInitialCorners() {
    const initialValue = 0;
    set(0, 0, initialValue);
    set(widthSegments, 0, initialValue);
    set(0, heightSegments, initialValue);
    set(widthSegments, heightSegments, initialValue);
  }

  setInitialCorners();

  let length = widthSegments;
  let scale = 20;
  let currentIteration = 0;

  while (currentIteration < iterations) {
    // Diamond step
    for (let x = length / 2; x <= widthSegments; x += length) {
      for (let y = length / 2; y <= heightSegments; y += length) {
        const topLeft = sample(x - length / 2, y - length / 2);
        const topRight = sample(x + length / 2, y - length / 2);
        const bottomLeft = sample(x - length / 2, y + length / 2);
        const bottomRight = sample(x + length / 2, y + length / 2);

        const average = (topLeft + topRight + bottomLeft +
bottomRight) / 4;
        const offset = Math.random() * scale * 2 - scale;
        set(x, y, average + offset);
      }
    }

    // Square step
    for (let x = 0; x <= widthSegments; x += length / 2) {
      for (let y = (x + length / 2) % length; y <= heightSegments; y
+= length) {
        const left = sample(x - length / 2, y);
        const right = sample(x + length / 2, y);
        const top = sample(x, y - length / 2);
        const bottom = sample(x, y + length / 2);

        let sum = 0;
        let count = 0;

        if (typeof left !== "undefined") {
          sum += left;
          count++;
        }

        if (typeof right !== "undefined") {
          sum += right;
          count++;
        }

        if (typeof top !== "undefined") {
          sum += top;
          count++;
        }

        if (typeof bottom !== "undefined") {
          sum += bottom;
          count++;
        }

        const average = sum / count;
        const offset = Math.random() * scale * 2 - scale;
        set(x, y, average + offset);
      }
    }

    length /= 2;
    scale /= 2;
    currentIteration++;
  }

  // Update the PlaneGeometry vertex positions
  const positionAttribute = geometry.attributes.position as
BufferAttribute;
  for (let i = 0; i < positionAttribute.count; i++) {
    const vertex = new
Vector3().fromBufferAttribute(positionAttribute, i);
    vertex.y = data[i];
    positionAttribute.setXYZ(i, vertex.x, vertex.y, vertex.z);
  }
}


interface GetCrocodileSkin {
  color: Color;
  sheen: number;
  transmission: number;
  metalness: number;
  roughness: number;
  ior: number;
  specularIntensity: number;
  specularColor: Color;
  envMapIntensity: number;
  reflectivity: number;
  thickness: number;
  customDepthMaterial: ShaderMaterial;
}

export const getDiamondSquare = (params: GetCrocodileSkin) => {
  const width = 200;
  const height = 200;
  const resolution = 128;
  const maxIterations = 8; // The specified number of iterations
  const geometry = new PlaneGeometry(width, height, resolution,
resolution);
  geometry.rotateX(-Math.PI / 2);

  // Provide the iterations as an argument
  diamondSquareAlgorithm(geometry, maxIterations);

  const material = new MeshPhysicalMaterial({
    // attenuationDistance: 0.1,
    clearcoat: 1,
    // premultipliedAlpha: true,
    sheen: 1,
    // depthWrite: false,
    color: params.color,
    transmission: params.transmission,
```

```
      metalness: params.metalness,
      roughness: params.roughness,
      ior: params.ior,
      specularIntensity: params.specularIntensity,
      specularColor: params.specularColor,
      side: DoubleSide,
      envMapIntensity: params.envMapIntensity,
      reflectivity: params.reflectivity,
    });
    material.thickness = params.thickness;
    material.needsUpdate = true;
    material.shadowSide = DoubleSide;

    const crocodileSkin = new Mesh(geometry, material);

    crocodileSkin.geometry.computeVertexNormals();
    crocodileSkin.geometry.computeBoundingBox();
    crocodileSkin.geometry.computeBoundingSphere();

    crocodileSkin.customDepthMaterial =
params.customDepthMaterial;
    crocodileSkin.receiveShadow = true;
    crocodileSkin.castShadow = true;

    crocodileSkin.position.y = 0.5;
    crocodileSkin.position.z = 0.5;
    crocodileSkin.scale.set(0.5, 0.5, 0.5);

    crocodileSkin.rotateY(2.15);

    return crocodileSkin;
};

 import {
    BufferAttribute,
    Color,
    DoubleSide,
    Mesh,
    MeshPhysicalMaterial,
    PlaneGeometry,
    ShaderMaterial,
    Vector3,
} from 'three'

function diamondSquare(geometry, size) {
    const widthSegments = geometry.parameters.widthSegments;
    const heightSegments = geometry.parameters.heightSegments;

    const data = new Float32Array((widthSegments + 1) *
(heightSegments + 1));

    function set(x, y, value) {
      data[x + y * (widthSegments + 1)] = value;
    }

    function get(x, y) {
      return data[x + y * (widthSegments + 1)];
    }

    function sample(x, y) {
      x = x % (widthSegments + 1);
      y = y % (heightSegments + 1);
      return get(x, y);
    }

    function midpointDisplacement(x, y, w) {
      if (w === 1) return;

      const halfW = w / 2;
      const topLeft = sample(x, y);
      const topRight = sample(x + w, y);
      const bottomLeft = sample(x, y + w);
      const bottomRight = sample(x + w, y + w);

      const diamondValue = (topLeft + topRight + bottomLeft +
bottomRight) / 4;
      set(x + halfW, y + halfW, diamondValue + Math.random() * w
- halfW);

      const squareTopValue = (topLeft + topRight + diamondValue) /
3;
      const squareLeftValue = (topLeft + bottomLeft +
diamondValue) / 3;
      const squareRightValue = (topRight + bottomRight +
diamondValue) / 3;
      const squareBottomValue = (bottomLeft + bottomRight +
diamondValue) / 3;

      set(x + halfW, y, squareTopValue + Math.random() * halfW -
halfW / 2);
      set(x, y + halfW, squareLeftValue + Math.random() * halfW -
halfW / 2);
      set(x + w, y + halfW, squareRightValue + Math.random() *
halfW - halfW / 2);
      set(x + halfW, y + w, squareBottomValue + Math.random() *
halfW - halfW / 2);

      midpointDisplacement(x, y, halfW);
      midpointDisplacement(x + halfW, y, halfW);
      midpointDisplacement(x, y + halfW, halfW);
      midpointDisplacement(x + halfW, y + halfW, halfW);
    }

    midpointDisplacement(0, 0, widthSegments);

    const positionAttribute = geometry.attributes.position as
BufferAttribute;
    for (let i = 0; i < positionAttribute.count; i++) {
      const vertex = new
Vector3().fromBufferAttribute(positionAttribute, i);
      vertex.y = data[i];
      positionAttribute.setXYZ(i, vertex.x, vertex.y, vertex.z);
    }
}


interface GetCrocodileSkin {
    color: Color;
    sheen: number;
    transmission: number;
    metalness: number;
    roughness: number;
    ior: number;
    specularIntensity: number;
    specularColor: Color;
    envMapIntensity: number;
    reflectivity: number;
    thickness: number;
    customDepthMaterial: ShaderMaterial;
}

export const getDiamondSquare = (params: GetCrocodileSkin) => {
    const width = 200;
    const height = 200;
    const resolution = 256;
    const geometry = new PlaneGeometry(width, height, resolution,
resolution);
    geometry.rotateX(-Math.PI / 2);

    diamondSquare(geometry, resolution);

    const material = new MeshPhysicalMaterial({
      // attenuationDistance: 0.1,
      clearcoat: 1,
      // premultipliedAlpha: true,
      sheen: 1,
      // depthWrite: false,
      color: params.color,
      transmission: params.transmission,
      metalness: params.metalness,
```

```
    roughness: params.roughness,
    ior: params.ior,
    specularIntensity: params.specularIntensity,
    specularColor: params.specularColor,
    side: DoubleSide,
    envMapIntensity: params.envMapIntensity,
    reflectivity: params.reflectivity,
  });
  material.thickness = params.thickness;
  material.needsUpdate = true;
  material.shadowSide = DoubleSide;

  const crocodileSkin = new Mesh(geometry, material);

  crocodileSkin.geometry.computeVertexNormals();
  crocodileSkin.geometry.computeBoundingBox();
  crocodileSkin.geometry.computeBoundingSphere();

  crocodileSkin.customDepthMaterial =
params.customDepthMaterial;
  crocodileSkin.receiveShadow = true;
  crocodileSkin.castShadow = true;

  crocodileSkin.position.y = 0.5;
  crocodileSkin.position.z = 0.5;
  crocodileSkin.scale.set(0.5, 0.5, 0.5);

  crocodileSkin.rotateY(2.15);

  return crocodileSkin;
};
```

Модуль: src/client/mesh/parallelepiped.ts

```
 import { BoxGeometry, MeshPhysicalMaterial, Mesh, DoubleSide,
Color } from 'three';

export const getParallelepiped = () => {
  const parallelepipedGeometry = new BoxGeometry(5, 0.5, 5);
  const parallelepipedMaterial = new MeshPhysicalMaterial({
    color: 0xffffff,
    transmission: 0.01,
    metalness: 0,
    roughness: 0.07,
    ior: 1.5,
    specularIntensity: 1,
    specularColor: new Color(0xffffff),
    side: DoubleSide,
    envMapIntensity: 1.5,
    reflectivity: 0.3,
  });

  parallelepipedMaterial.thickness = 3;

  const parallelepiped = new Mesh(parallelepipedGeometry,
parallelepipedMaterial);

  parallelepiped.receiveShadow = true;
  parallelepiped.castShadow = true;
  parallelepiped.position.set(-1, 1, 1);
  parallelepiped.rotateX(0.5);
  parallelepiped.rotateY(1);

  parallelepiped.material.shadowSide = DoubleSide;

  parallelepiped.geometry.computeVertexNormals();
  // cube.geometry.computeFaceNormals();
  parallelepiped.geometry.computeBoundingBox();
  parallelepiped.geometry.computeBoundingSphere();

  return parallelepiped;
};
```

Модуль: src/client/mesh/perlineNoise.ts

```
 import {
   MeshPhysicalMaterial,
   Mesh,
   DoubleSide,
   Color,
   Vector3,
   ShaderMaterial,
   PlaneGeometry,
   BufferAttribute,
} from 'three'


export class PerlinNoise {
   private permutation: number[];

   constructor(seed: number) {
      this.permutation = [];
      const random = new Random(seed);
      for (let i = 0; i < 256; i++) {
         this.permutation.push(Math.floor(random.nextFloat * 255));
      }
   }

   private fade(t: number): number {
      return t * t * t * (t * (t * 6 - 15) + 10);
   }

   private lerp(a: number, b: number, x: number): number {
      return a + x * (b - a);
   }

   private grad(hash: number, x: number, y: number, z: number):
number {
      const h = hash & 15;
      const u = h < 8 ? x : y;
      const v = h < 4 ? y : h === 12 || h === 14 ? x : z;

      return (((h & 1) === 0 ? u : -u) + ((h & 2) === 0 ? v : -v));
   }

   noise(x: number, y: number, z: number): number {
      const X = Math.floor(x) & 255;
      const Y = Math.floor(y) & 255;
      const Z = Math.floor(z) & 255;

      x -= Math.floor(x);
      y -= Math.floor(y);
      z -= Math.floor(z);

      const u = this.fade(x);
      const v = this.fade(y);
      const w = this.fade(z);

      const A = this.permutation[X] + Y;
      const AA = this.permutation[A] + Z;
      const AB = this.permutation[A + 1] + Z;
      const B = this.permutation[X + 1] + Y;
      const BA = this.permutation[B] + Z;
      const BB = this.permutation[B + 1] + Z;

      return this.lerp(
         this.lerp(this.grad(this.permutation[AA], x, y, z),
            this.grad(this.permutation[BA], x - 1, y, z), u),
         this.lerp(this.grad(this.permutation[AB], x, y - 1, z),
            this.grad(this.permutation[BB], x - 1, y - 1, z), u), v);
   }
}

class Random {
```

```
    private _seed: number;
    private m: number;
    private a: number;
    private c: number;

    constructor(seed: number) {
        this.m = 4294967296;
        this.a = 1664525;
        this.c = 1013904223;

        this.seed = seed % this.m;
    }

    set seed(val: number) {
        this._seed = val % this.m;
    }

    get seed() {
        return this._seed;
    }

    get nextFloat(): number {
        this._seed = (this.a * this._seed + this.c) % this.m;
        return this._seed / this.m;
    }
}

function pointToVector3(point) {
    return new Vector3(point.x, point.y, point.z);
}

interface GetCrocodileSkin {
    color: Color;
    sheen: number;
    transmission: number;
    metalness: number;
    roughness: number;
    ior: number;
    specularIntensity: number;
    specularColor: Color;
    envMapIntensity: number;
    reflectivity: number;
    thickness: number;
    customDepthMaterial: ShaderMaterial;
}

export const getPerlinNoise = (params: GetCrocodileSkin) => {
    const perlin = new PerlinNoise(84);
    const width = 200;
    const height = 200;
    const geometry = new PlaneGeometry(width, height, width - 1,
height - 1);
    geometry.rotateX(-Math.PI / 2);
    const scale = 8;
    const intensity = 1;

    const positionAttribute = geometry.getAttribute('position');

    if (geometry.attributes.position instanceof BufferAttribute) {
        const positionAttribute = geometry.attributes.position as
BufferAttribute;
        const numVertices = positionAttribute.count;
        const elementSize = positionAttribute.itemSize;

        for (let i = 0; i < numVertices; i++) {
            const vertex = new
Vector3().fromBufferAttribute(positionAttribute, i);
            const noiseValue = perlin.noise(vertex.x / scale, vertex.z /
scale, 0);
            vertex.y += intensity * noiseValue;

            positionAttribute.set([vertex.x, vertex.y, vertex.z], i *
elementSize);
        }
    }

    const material = new MeshPhysicalMaterial({
        // attenuationDistance: 0.1,
        clearcoat: 1,
        // premultipliedAlpha: true,
        sheen: 1,
        // depthWrite: false,
        color: params.color,
        transmission: params.transmission,
        metalness: params.metalness,
        roughness: params.roughness,
        ior: params.ior,
        specularIntensity: params.specularIntensity,
        specularColor: params.specularColor,
        side: DoubleSide,
        envMapIntensity: params.envMapIntensity,
        reflectivity: params.reflectivity,
    });
    material.thickness = params.thickness;
    material.needsUpdate = true;
    material.shadowSide = DoubleSide;

    const crocodileSkin = new Mesh(geometry, material);

    crocodileSkin.geometry.computeVertexNormals();
    crocodileSkin.geometry.computeBoundingBox();
    crocodileSkin.geometry.computeBoundingSphere();

    crocodileSkin.customDepthMaterial =
params.customDepthMaterial;
    crocodileSkin.receiveShadow = true;
    crocodileSkin.castShadow = true;

    crocodileSkin.position.y = 0.5;
    crocodileSkin.position.z = 0.5;
    crocodileSkin.scale.set(0.5, 0.5, 0.5);

    crocodileSkin.rotateY(2.15);

    return crocodileSkin;
};
```

Модуль: src/client/mesh/plane.ts

```
 import { PlaneGeometry, MeshPhysicalMaterial, Mesh, DoubleSide
} from 'three';

interface PlaneProps {
  position: {
    x: number;
    y: number;
    z: number;
  }
  rotate: {
    x?: number;
    y?: number;
  }
}

export const getPlane = ({ position, rotate }: PlaneProps) => {
  const geometry = new PlaneGeometry(20, 20);
  const material = new MeshPhysicalMaterial({
    color: 0x6c7b8b,
    side: DoubleSide,
    metalness: 0.5,
    roughness: 1,
  });
  const mesh = new Mesh(geometry, material);

  mesh.position.set(position.x, position.y, position.z);
  mesh.rotateX(rotate.x ?? 0);
  mesh.rotateY(rotate.y ?? 0);

  material.shadowSide = DoubleSide;
```

```
  mesh.receiveShadow = true;

  geometry.computeVertexNormals();
  geometry.computeBoundingBox();
  geometry.computeBoundingSphere();

  return mesh;
```

Модуль: src/client/mesh/polygonPyramid.ts

```
 import {
   MeshPhysicalMaterial,
   Mesh,
   DoubleSide,
   BufferGeometry,
   Float32BufferAttribute,
   Color,
   ShaderMaterial,
} from 'three'
import { mergeBufferGeometries } from
'three/examples/jsm/utils/BufferGeometryUtils';
import { getPyramids } from '../algorithm/polygonPyramid';

interface GetPolygonPyramid {
   color: Color;
   sheen: number;
   transmission: number;
   metalness: number;
   roughness: number;
   ior: number;
   specularIntensity: number;
   specularColor: Color;
   envMapIntensity: number;
   reflectivity: number;
   thickness: number;
   customDepthMaterial: ShaderMaterial;
}

export const getPolygonPyramid = (params: GetPolygonPyramid) =>
{
   const pyramidFigures = getPyramids();

   const sides = [];

   for (const pyramidFigure of pyramidFigures) {
      const aSide = [pyramidFigure.A, pyramidFigure.B,
pyramidFigure.C];
      const bSide = [pyramidFigure.A, pyramidFigure.B,
pyramidFigure.D];
      const cSide = [pyramidFigure.A, pyramidFigure.D,
pyramidFigure.C];
      const dSide = [pyramidFigure.D, pyramidFigure.B,
pyramidFigure.C];
      const sidesWithPoints = [aSide, bSide, cSide, dSide];

      for (const sideWithPoints of sidesWithPoints) {
         const sideGeometry = new BufferGeometry();
         sideGeometry.setAttribute('position', new
Float32BufferAttribute(sideWithPoints.flatMap(Object.values), 3));

         sides.push(sideGeometry);
      }
   }

   const pyramidFractalGeometry = mergeBufferGeometries(sides);

   const material = new MeshPhysicalMaterial({
      // attenuationDistance: 0.1,
      clearcoat: 1,
      // premultipliedAlpha: true,
      sheen: 1,
      // depthWrite: false,
      color: params.color,
      transmission: params.transmission,
      metalness: params.metalness,
      roughness: params.roughness,
      ior: params.ior,
      specularIntensity: params.specularIntensity,
      specularColor: params.specularColor,
      side: DoubleSide,
      envMapIntensity: params.envMapIntensity,
      reflectivity: params.reflectivity,
   });
   material.thickness = params.thickness;
   material.needsUpdate = true;
   material.shadowSide = DoubleSide;

   const pyramidFractal = new Mesh(pyramidFractalGeometry,
material);

   pyramidFractal.geometry.computeVertexNormals();
   pyramidFractal.geometry.computeBoundingBox();
   pyramidFractal.geometry.computeBoundingSphere();

   pyramidFractal.customDepthMaterial =
params.customDepthMaterial;
   pyramidFractal.receiveShadow = true;
   pyramidFractal.castShadow = true;

   pyramidFractal.position.y = 0.5;
   pyramidFractal.position.z = 0.5;
   pyramidFractal.scale.set(0.5, 0.5, 0.5);

   pyramidFractal.rotateY(2.15);

   return [pyramidFractal];
};
```

Модуль: src/client/mesh/simplePyramid.ts

```
 import {
  MeshPhysicalMaterial,
  Mesh,
  DoubleSide,
  BufferGeometry,
  Float32BufferAttribute,
  Color,
  ShaderMaterial,
} from 'three'
import { mergeBufferGeometries } from
'three/examples/jsm/utils/BufferGeometryUtils';
import { getPyramids } from '../algorithm/simplePyramid';

interface GetFlatPyramidParams {
  color: Color;
  sheen: number;
  transmission: number;
  metalness: number;
  roughness: number;
  ior: number;
  specularIntensity: number;
  specularColor: Color;
  envMapIntensity: number;
  reflectivity: number;
  thickness: number;
  customDepthMaterial: ShaderMaterial;
}

export const getSimplePyramid = (params: GetFlatPyramidParams)
=> {
  const pyramidFigures = getPyramids();

  const sides = [];
```

```
  for (const pyramidFigure of pyramidFigures) {
    const aSide = [pyramidFigure.A, pyramidFigure.B,
pyramidFigure.C];
    const bSide = [pyramidFigure.A, pyramidFigure.B,
pyramidFigure.D];
    const cSide = [pyramidFigure.A, pyramidFigure.D,
pyramidFigure.C];
    const dSide = [pyramidFigure.D, pyramidFigure.B,
pyramidFigure.C];
    const sidesWithPoints = [aSide, bSide, cSide, dSide];

    for (const sideWithPoints of sidesWithPoints) {
      const sideGeometry = new BufferGeometry();
      sideGeometry.setAttribute('position', new
Float32BufferAttribute(sideWithPoints.flatMap(Object.values), 3));

      sides.push(sideGeometry);
    }
  }

  const pyramidFractalGeometry = mergeBufferGeometries(sides);

  const material = new MeshPhysicalMaterial({
    // attenuationDistance: 0.1,
    clearcoat: 1,
    // premultipliedAlpha: true,
    sheen: 1,
    // depthWrite: false,
    color: params.color,
    transmission: params.transmission,
    metalness: params.metalness,
    roughness: params.roughness,
    ior: params.ior,
    specularIntensity: params.specularIntensity,
    specularColor: params.specularColor,
    side: DoubleSide,
    envMapIntensity: params.envMapIntensity,
    reflectivity: params.reflectivity,
  });
  material.thickness = params.thickness;
  material.needsUpdate = true;
  material.shadowSide = DoubleSide;

  const pyramidFractal = new Mesh(pyramidFractalGeometry,
material);

  pyramidFractal.geometry.computeVertexNormals();
  pyramidFractal.geometry.computeBoundingBox();
  pyramidFractal.geometry.computeBoundingSphere();

  pyramidFractal.customDepthMaterial =
params.customDepthMaterial;
  pyramidFractal.receiveShadow = true;
  pyramidFractal.castShadow = true;

  return [pyramidFractal];
};
```

Модуль: src/client/mesh/simplexNoise.ts

```
 import {
    Color,
    DoubleSide,
    Mesh,
    MeshPhysicalMaterial,
    PlaneGeometry,
    ShaderMaterial,
    Vector3,
} from 'three'
import { ImprovedNoise } from
'three/examples/jsm/math/ImprovedNoise.js';

function diamondSquareAlgorithm(geometry, resolution, iterations =
5) {
    let length = resolution + 1;
    let scale = 10;
    let currentIteration = 0;

    while (currentIteration < iterations) {
        const halfLength = Math.floor(length / 2);

        // Diamond step
        for (let x = halfLength; x < resolution; x += length) {
            for (let y = halfLength; y < resolution; y += length) {
                const topLeft = geometry.attributes.position.array[((x -
halfLength) + (y - halfLength) * (resolution + 1)) * 3];
                const topRight = geometry.attributes.position.array[((x +
halfLength) + (y - halfLength) * (resolution + 1)) * 3];
                const bottomLeft = geometry.attributes.position.array[((x -
halfLength) + (y + halfLength) * (resolution + 1)) * 3];
                const bottomRight = geometry.attributes.position.array[((x
+ halfLength) + (y + halfLength) * (resolution + 1)) * 3];
                const average = (topLeft + topRight + bottomLeft +
bottomRight) / 4;
                geometry.attributes.position.array[(x + y * (resolution + 1))
* 3] = average + scale * (Math.random() * 2 - 1);
            }
        }

        // Square step
        for (let x = 0; x <= resolution; x += halfLength) {
            for (let y = (x + halfLength) % length; y <= resolution; y +=
length) {
                const maxX = x + halfLength <= resolution;
                const maxY = y + halfLength <= resolution;
                const minX = x - halfLength >= 0;
                const minY = y - halfLength >= 0;

                const top = (minY ? geometry.attributes.position.array[(x +
(y - halfLength) * (resolution + 1)) * 3] : 0);
                const bottom = (maxY ?
geometry.attributes.position.array[(x + (y + halfLength) * (resolution
+ 1)) * 3] : 0);
                const left = (minX ? geometry.attributes.position.array[((x
- halfLength) + y * (resolution + 1)) * 3] : 0);
                const right = (maxX ?
geometry.attributes.position.array[((x + halfLength) + y * (resolution
+ 1)) * 3] : 0);
                const count = (minY ? 1 : 0) + (maxY ? 1 : 0) + (minX ? 1 :
0) + (maxX ? 1 : 0);
                const average = (top + bottom + left + right) / count;
                geometry.attributes.position.array[(x + y * (resolution + 1))
* 3] = average + scale * (Math.random() * 2 - 1);
            }
        }

        // Update length and scale for the next iteration
        length = halfLength;
        scale /= 2;
        currentIteration += 1;
    }
}

function applyPerlinNoise(geometry, scale = 5, frequency = 10) {
    const perlinNoiseGenerator = new ImprovedNoise();
    const positionAttribute = geometry.attributes.position;
    let noiseZ = 0;

    for (let i = 0; i < positionAttribute.count; i++) {
        const vertex = new
Vector3().fromBufferAttribute(positionAttribute, i);

        const noiseValue = perlinNoiseGenerator.noise(vertex.x * 0.1 *
frequency, vertex.y * 0.1 * frequency, noiseZ);
        vertex.z += noiseValue * scale;

        positionAttribute.setXYZ(i, vertex.x, vertex.y, vertex.z);
```

```
      // Increment noiseZ so each noise call has a different z-
coordinate, affecting the generated pattern
      noiseZ += 0.1;
    }
}

interface GetCrocodileSkin {
    color: Color;
    sheen: number;
    transmission: number;
    metalness: number;
    roughness: number;
    ior: number;
    specularIntensity: number;
    specularColor: Color;
    envMapIntensity: number;
    reflectivity: number;
    thickness: number;
    customDepthMaterial: ShaderMaterial;
}

export const getSimplexNoise = (params: GetCrocodileSkin) => {
    const width = 200;
    const height = 200;
    const resolution = 200; // Increase this value to add more triangles
    // const resolution = 1000;
    const geometry = new PlaneGeometry(width, height, resolution -
1, resolution - 1);
    geometry.rotateX(-Math.PI / 2);

    // diamondSquareAlgorithm(geometry, resolution, 5);
    applyPerlinNoise(geometry);

    const material = new MeshPhysicalMaterial({
        // attenuationDistance: 0.1,
        clearcoat: 1,
        // premultipliedAlpha: true,
        sheen: 1,
        // depthWrite: false,
        color: params.color,
        transmission: params.transmission,
        metalness: params.metalness,
        roughness: params.roughness,
        ior: params.ior,
        specularIntensity: params.specularIntensity,
        specularColor: params.specularColor,
        side: DoubleSide,
        envMapIntensity: params.envMapIntensity,
        reflectivity: params.reflectivity,
    });
    material.thickness = params.thickness;
    material.needsUpdate = true;
    material.shadowSide = DoubleSide;

    const crocodileSkin = new Mesh(geometry, material);

    crocodileSkin.geometry.computeVertexNormals();
    crocodileSkin.geometry.computeBoundingBox();
    crocodileSkin.geometry.computeBoundingSphere();

    crocodileSkin.customDepthMaterial =
params.customDepthMaterial;
    crocodileSkin.receiveShadow = true;
    crocodileSkin.castShadow = true;

    crocodileSkin.position.y = 0.5;
    crocodileSkin.position.z = 0.5;
    crocodileSkin.scale.set(0.5, 0.5, 0.5);

    crocodileSkin.rotateY(2.15);

    return crocodileSkin;
};
```

Модуль: src/client/mesh/tieredPyramid.ts

```
 import {
    MeshPhysicalMaterial,
    Mesh,
    DoubleSide,
    BufferGeometry,
    Float32BufferAttribute,
    Color,
    ShaderMaterial,
} from 'three'
import { mergeBufferGeometries } from
'three/examples/jsm/utils/BufferGeometryUtils';
import { getPyramids } from '../algorithm/tieredPyramid';

interface GetFlatPyramidParams {
    color: Color;
    sheen: number;
    transmission: number;
    metalness: number;
    roughness: number;
    ior: number;
    specularIntensity: number;
    specularColor: Color;
    envMapIntensity: number;
    reflectivity: number;
    thickness: number;
    customDepthMaterial: ShaderMaterial[];
}

export const getTieredPyramid = (params: GetFlatPyramidParams)
=> {
    const [outerPyramidFigures, innerPyramidFigures] =
getPyramids();

    // outerPyramidFigures
    const outerSides = [];
    for (const pyramidFigure of outerPyramidFigures) {
        const aSide = [pyramidFigure.A, pyramidFigure.B,
pyramidFigure.C];
        const bSide = [pyramidFigure.A, pyramidFigure.B,
pyramidFigure.D];
        const cSide = [pyramidFigure.A, pyramidFigure.D,
pyramidFigure.C];
        const dSide = [pyramidFigure.D, pyramidFigure.B,
pyramidFigure.C];
        const sidesWithPoints = [aSide, bSide, cSide, dSide];

        for (const sideWithPoints of sidesWithPoints) {
            const sideGeometry = new BufferGeometry();
            sideGeometry.setAttribute('position', new
Float32BufferAttribute(sideWithPoints.flatMap(Object.values), 3));

            outerSides.push(sideGeometry);
        }
    }

    const outerPyramidFractalGeometry =
mergeBufferGeometries(outerSides);

    const outerMaterial = new MeshPhysicalMaterial({
        // attenuationDistance: 0.1,
        clearcoat: 1,
        // premultipliedAlpha: true,
        sheen: 1,
        // depthWrite: false,
        color: params.color,
        transmission: params.transmission,
        metalness: params.metalness,
        roughness: params.roughness,
```

```
      ior: params.ior,
      specularIntensity: params.specularIntensity,
      specularColor: params.specularColor,
      side: DoubleSide,
      envMapIntensity: params.envMapIntensity,
      reflectivity: params.reflectivity,
    });
    outerMaterial.thickness = params.thickness;
    outerMaterial.needsUpdate = true;
    outerMaterial.shadowSide = DoubleSide;

    const outerPyramidFractal = new
Mesh(outerPyramidFractalGeometry, outerMaterial);

    outerPyramidFractal.geometry.computeVertexNormals();
    outerPyramidFractal.geometry.computeBoundingBox();
    outerPyramidFractal.geometry.computeBoundingSphere();

    outerPyramidFractal.customDepthMaterial =
params.customDepthMaterial[0];
    outerPyramidFractal.receiveShadow = true;
    outerPyramidFractal.castShadow = true;

    // innerPyramidFigures
    const innerSides = [];
    for (const pyramidFigure of innerPyramidFigures) {
      const aSide = [pyramidFigure.A, pyramidFigure.B,
pyramidFigure.C];
      const bSide = [pyramidFigure.A, pyramidFigure.B,
pyramidFigure.D];
      const cSide = [pyramidFigure.A, pyramidFigure.D,
pyramidFigure.C];
      const dSide = [pyramidFigure.D, pyramidFigure.B,
pyramidFigure.C];
      const sidesWithPoints = [aSide, bSide, cSide, dSide];

      for (const sideWithPoints of sidesWithPoints) {
        const sideGeometry = new BufferGeometry();
        sideGeometry.setAttribute('position', new
Float32BufferAttribute(sideWithPoints.flatMap(Object.values), 3));

        innerSides.push(sideGeometry);
      }
    }

    const innerPyramidFractalGeometry =
mergeBufferGeometries(innerSides);

    const innerMaterial = new MeshPhysicalMaterial({
      // attenuationDistance: 0.1,
      clearcoat: 1,
      // premultipliedAlpha: true,
      sheen: 1,
      // depthWrite: false,
      color: new Color(0x0008ed),
      // transmission: params.transmission,
      transmission: 0,
      metalness: params.metalness,
      roughness: params.roughness,
      ior: params.ior,
      specularIntensity: params.specularIntensity,
      specularColor: params.specularColor,
      side: DoubleSide,
      envMapIntensity: params.envMapIntensity,
      reflectivity: params.reflectivity,
    });
    innerMaterial.thickness = params.thickness;
    innerMaterial.needsUpdate = true;
    innerMaterial.shadowSide = DoubleSide;

    const innerPyramidFractal = new
Mesh(innerPyramidFractalGeometry, innerMaterial);

    innerPyramidFractal.geometry.computeVertexNormals();
    innerPyramidFractal.geometry.computeBoundingBox();
    innerPyramidFractal.geometry.computeBoundingSphere();

    innerPyramidFractal.customDepthMaterial =
params.customDepthMaterial[1];
    innerPyramidFractal.receiveShadow = true;
    innerPyramidFractal.castShadow = true;

    return [outerPyramidFractal, innerPyramidFractal];
};
```

Модуль: src/client/mesh/voronoiNoise.ts

```
 import {
    MeshPhysicalMaterial,
    Mesh,
    DoubleSide,
    Color,
    Vector3,
    ShaderMaterial,
    PlaneGeometry,
    BufferAttribute,
} from 'three'

export class VoronoiNoise {
    private seed: number;
    private pointCount: number;
    private gridSize: number;
    private random: Random;

    constructor(seed: number, gridSize: number = 10, pointCount:
number = 10) {
        this.seed = seed;
        this.pointCount = pointCount;
        this.gridSize = gridSize;
        this.random = new Random(seed);
    }

    private distance(x1: number, y1: number, x2: number, y2:
number): number {
        return Math.sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
    }

    noise(x: number, y: number): number {
        const cellX = Math.floor(x / this.gridSize);
        const cellY = Math.floor(y / this.gridSize);

        let minDist = Infinity;

        for (const sample of this.samplesInRegion(cellX, cellY)) {
            const dist = this.distance(sample.x, sample.y, x, y);
            if (dist < minDist) {
                minDist = dist;
            }
        }

        return minDist / this.gridSize;
    }

    private *samplesInRegion(x: number, y: number): Iterable<{ x:
number; y: number }> {
        for (let dx = -1; dx <= 1; ++dx) {
            for (let dy = -1; dy <= 1; ++dy) {
                const samples = this.samplesIn(x + dx, y + dy);
                for (const sample of samples) {
                    yield sample;
                }
            }
        }
    }

    private *samplesIn(x: number, y: number): Iterable<{ x: number;
y: number }> {
```

```
        this.random.seed = x * 1664525 + y;
        for (let i = 0; i < this.pointCount; ++i) {
          yield {
            x: x * this.gridSize + this.random.nextFloat * this.gridSize,
            y: y * this.gridSize + this.random.nextFloat * this.gridSize,
          };
        }
      }
    }

    class Random {
      private _seed: number;
      private a: number;
      private c: number;
      private m: number;

      constructor(seed: number) {
        this.a = 1664525;
        this.c = 1013904223;
        this.m = 4294967296;
        this._seed = seed % this.m;
      }

      get seed(): number {
        return this._seed;
      }

      set seed(val: number) {
        this._seed = val % this.m;
      }

      get nextFloat(): number {
        this._seed = (this.a * this._seed + this.c) % this.m;
        return this._seed / this.m;
      }
    }

    function pointToVector3(point) {
      return new Vector3(point.x, point.y, point.z);
    }

    interface GetCrocodileSkin {
      color: Color;
      sheen: number;
      transmission: number;
      metalness: number;
      roughness: number;
      ior: number;
      specularIntensity: number;
      specularColor: Color;
      envMapIntensity: number;
      reflectivity: number;
      thickness: number;
      customDepthMaterial: ShaderMaterial;
    }

    export const getVoronoiNoise = (params: GetCrocodileSkin) => {
      const width = 200;
      const height = 200;
      const resolution = 200; // Increase this value to add more triangles
      // const resolution = 1000;
      const geometry = new PlaneGeometry(width, height, resolution -
    1, resolution - 1);
      geometry.rotateX(-Math.PI / 2);

      const voronoi = new VoronoiNoise(84); // Voronoi Noise instance
    with a seed
      const scale = 10;
      const intensity = 10;
      const exponent = 4;

      if (geometry.attributes.position instanceof BufferAttribute) {
        const positionAttribute = geometry.attributes.position as
    BufferAttribute;
        const numVertices = positionAttribute.count;
        const elementSize = positionAttribute.itemSize;

        for (let i = 0; i < numVertices; i++) {
          const vertex = new
    Vector3().fromBufferAttribute(positionAttribute, i);
          const noiseValue = voronoi.noise(vertex.x / scale, vertex.z /
    scale);

          // Use exponentiation to create more angular geometry
          const angularNoiseValue = intensity * Math.pow(noiseValue,
    exponent);

          // Update the vertex position
          vertex.y += angularNoiseValue;

          positionAttribute.set([vertex.x, vertex.y, vertex.z], i *
    elementSize);
        }
      }

      const material = new MeshPhysicalMaterial({
        // attenuationDistance: 0.1,
        clearcoat: 1,
        // premultipliedAlpha: true,
        sheen: 1,
        // depthWrite: false,
        color: params.color,
        transmission: params.transmission,
        metalness: params.metalness,
        roughness: params.roughness,
        ior: params.ior,
        specularIntensity: params.specularIntensity,
        specularColor: params.specularColor,
        side: DoubleSide,
        envMapIntensity: params.envMapIntensity,
        reflectivity: params.reflectivity,
      });
      material.thickness = params.thickness;
      material.needsUpdate = true;
      material.shadowSide = DoubleSide;

      const crocodileSkin = new Mesh(geometry, material);

      crocodileSkin.geometry.computeVertexNormals();
      crocodileSkin.geometry.computeBoundingBox();
      crocodileSkin.geometry.computeBoundingSphere();

      crocodileSkin.customDepthMaterial =
    params.customDepthMaterial;
      crocodileSkin.receiveShadow = true;
      crocodileSkin.castShadow = true;

      crocodileSkin.position.y = 0.5;
      crocodileSkin.position.z = 0.5;
      crocodileSkin.scale.set(0.5, 0.5, 0.5);

      crocodileSkin.rotateY(2.15);

      return crocodileSkin;
    };
```

Модуль: src/client/renderer/renderer.ts

```
 import {
  WebGLRenderer,
  ACESFilmicToneMapping,
  PCFSoftShadowMap,
  sRGBEncoding,
} from 'three'

export const getRenderer = () => {
  const renderer = new WebGLRenderer({
    antialias: true,
    preserveDrawingBuffer: true,
```

```
  });

  renderer.shadowMap.enabled = true;
  renderer.shadowMap.needsUpdate = true;
  renderer.shadowMap.type = PCFSoftShadowMap;
  renderer.toneMapping = ACESFilmicToneMapping;
  renderer.toneMappingExposure = 2;
  renderer.outputEncoding = sRGBEncoding;

  renderer.setSize(innerWidth, innerHeight);

  return renderer;
};
```

Модуль: src/client/scene/pyramidFactory.ts

```
 import { Color, ShaderMaterial } from 'three'
import { getSimplePyramid, getPolygonPyramid, getTieredPyramid
} from '../../mesh';
import { getFigureType } from '../../utils';
import { FigureType } from '../../utils';

export const getPyramidFractal = (depthMaterial: ShaderMaterial[])
=> {
    switch (getFigureType() as FigureType) {
      case 'serpinsiy_tetraider': {
        return getSimplePyramid({
          color: new Color(0xffffff),
          sheen: 1,
          transmission: 0.01,
          metalness: 0,
          roughness: 0.07,
          ior: 1.5,
          specularIntensity: 1,
          specularColor: new Color(0xffffff),
          envMapIntensity: 1,
          reflectivity: 0.3,
          thickness: 3,
          customDepthMaterial: depthMaterial[0],
        });
      }
      case 'serpinsiy_tiered_tetraider': {
        return getTieredPyramid({
          color: new Color(0xfffe00),
          sheen: 1,
          transmission: 0.01,
          metalness: 0,
          roughness: 0.07,
          ior: 1.5,
          specularIntensity: 1,
          specularColor: new Color(0xffffff),
          envMapIntensity: 1,
          reflectivity: 0.3,
          thickness: 3,
          customDepthMaterial: depthMaterial,
        });
      }
      case 'custom_tetraider': {
        return getPolygonPyramid({
          color: new Color(0xffffff),
          sheen: 1,
          transmission: 0.01,
          metalness: 0,
          roughness: 0.07,
          ior: 1.5,
          specularIntensity: 1,
          specularColor: new Color(0xffffff),
          envMapIntensity: 1,
          reflectivity: 0.3,
          thickness: 3,
          customDepthMaterial: depthMaterial[0],
        });
    }
  }
}
```

src/client/scene/scene.ts

```
 import { Scene, Color } from 'three'
import { getSpotLight } from '../light/spot';
import { initGUIControls } from '../guicontrols'
import { getDepthMaterialWithShadowOpacity } from '../utils';
import { getPlane } from '../mesh';
import { getPyramidFractal } from
'./pyramidFactory/pyramidFactory';

export const getScene = () => {
  const scene = new Scene();

  const plane = getPlane({
    position: { x: 0, y: 0, z: -6 },
    rotate: { x: -1, y: 0.2 }
  });
  const depthMaterials = [getDepthMaterialWithShadowOpacity(),
getDepthMaterialWithShadowOpacity()];
  const fractals = getPyramidFractal(depthMaterials);
  const spotLightA = getSpotLight({
    color: new Color(0xffffff),
    intensity: 6,
    penumbra: 0.3,
    distance: 40,
    angle: 0.5,
    position: { x: 10, y: 5, z: 10 },
    objectToLookAt: fractals[0],
  });

  scene.background = new Color('black');

  scene.add(plane);

  scene.add(spotLightA.spotLight);
  // scene.add(spotLightA.spotLight.target);
  // scene.add(spotLightA.spotLightHelper);
  spotLightA.updateLight();


  fractals.forEach((fractal, index) => {
    scene.add(fractal);

    initGUIControls({
      fractal: fractal,
      depthMaterial: depthMaterials[index],
      spotLightA,
    });

  });

  return scene;
};
```

Модуль: src/client/shader/ScreenDoorShader.ts

```
 import { UniformsUtils } from 'three';

function cloneShader( shader, uniforms, defines ) {

		const newShader = Object.assign( {}, shader );
		newShader.uniforms = UniformsUtils.merge( [
			newShader.uniforms,
			uniforms
		] );
		newShader.defines = Object.assign( {}, defines );
```

```
		return newShader;

}

export function DitheredTransparencyShaderMixin( shader ) {
		const defineKeyword =
'ENABLE_DITHER_TRANSPARENCY';
		const newShader = cloneShader(shader, { ditherTex: {
value: null } }, { [ defineKeyword ]: 1 });

		newShader.fragmentShader = `
				float bayerDither2x2( vec2 v ) {
					return mod( 3.0 * v.y +
2.0 * v.x, 4.0 );
				}

				float bayerDither4x4( vec2 v ) {
					vec2 P1 = mod( v, 2.0 );
					vec2 P2 = mod( floor(
0.5  * v ), 2.0 );
					return 4.0 *
bayerDither2x2( P1 ) + bayerDither2x2( P2 );
				}
				${newShader.fragmentShader}
		`.replace(
			/main\(\) {/,
			v => `
				${v}
				#if ${defineKeyword}
				if( ( bayerDither4x4( floor( mod(
gl_FragCoord.xy, 8.0 ) ) ) ) / 16.0 >= opacity ) discard;
				#endif
			`
		);

		return newShader;
}
```

Модуль: src/client/utils/getDepthMaterialWithShadowOpacity.ts

```
 import { RGBADepthPacking, ShaderLib, ShaderMaterial } from
'three'
import { DitheredTransparencyShaderMixin} from
'../shader/ScreenDoorShader';

export const getDepthMaterialWithShadowOpacity = () => {
   const depthShader =
DitheredTransparencyShaderMixin(ShaderLib.depth);
   depthShader.fragmentShader = `uniform float opacity;\n${
depthShader.fragmentShader }`;
   const depthMaterial = new ShaderMaterial(depthShader);
   depthMaterial.defines.DEPTH_PACKING =
RGBADepthPacking;
   depthMaterial.uniforms.opacity.value = 0.99;
   window['configs'].transparency = 0.01;

   return depthMaterial;
}
```

Модуль: src/client/utils/initActionButtons.ts

```
 import { METRICS } from './metrics';
import { isCustomTetraider } from './configuration';

const saveImageToFile = async (canvasEl: HTMLCanvasElement)
=> {
  const imageDataURL = canvasEl.toDataURL();
  const filename = (new Date().toLocaleString('en-US', { hour12:
false }) as any).replaceAll(':', '-').replaceAll(',', '').replaceAll('/', '-');

  const { ok } = await fetch(
   'http://localhost:3000/upload-image-with-configs', {
     method: 'POST',
     headers: { 'Content-Type': 'application/json' },
     body: JSON.stringify({
       image: imageDataURL,
       filename: filename,
       configs: window['configs']
     }),
   }
  );

  if (ok) {
   window.alert('Дані збережено!')
  }
};

const initImageSaver = (canvasEl: HTMLCanvasElement) => {
  const saveImagebButton = document.createElement('button');
  saveImagebButton.style.position = 'absolute';
  saveImagebButton.style.bottom = '10px';
  saveImagebButton.style.left = '10px';
  saveImagebButton.style.width = '150px';
  saveImagebButton.style.height = '50px';
  saveImagebButton.style.userSelect = 'none';
  saveImagebButton.innerText = 'Зберегти зображення';

  saveImagebButton.addEventListener('click', () =>
saveImageToFile(canvasEl));
  document.body.appendChild(saveImagebButton);
};

const getAverageColor = (colors: Array<number>) => {
  const total = colors.reduce((result, currentColor) => result +
currentColor, 0);

  return Number(total / colors.length).toFixed(2);
};

const getMedianColor = (colors: Array<number>) => {
  const sortedColors = [...colors].sort((left, right) => right - left);
  const medianColor = sortedColors[Math.round(sortedColors.length
/ 2)];

  return medianColor;
};

const getMinColor = (colors: Array<number>) =>
Math.min(...Array.from(new Set(colors)));

const getMaxColor = (colors: Array<number>) =>
Math.max(...Array.from(new Set(colors)));

const calculateMetrics = (width: number, height: number,
imageData: ImageData) => {
  const grayScaleColors = [];
  const cachedContur = window['_cachedContur'];

  for (let row = 0; row < height; row++) {
    for (let col = 0; col < width; col++) {
      const index = (row * width + col) * 4;
      const isWithinContur = (index >= cachedContur[row].left &&
index <= cachedContur[row].right);

      if (isWithinContur) {
        const red = imageData.data[index];
        const green = imageData.data[index + 1];
        const blue = imageData.data[index + 2];

        const grayScaleColor = Number((0.299 * red + 0.587 * green +
```

```
0.114 * blue).toFixed(2));

        grayScaleColors.push(grayScaleColor);
      }
    }
  }

  return {
    [METRICS.figure]: window['configs'].figureType,
    [METRICS.iterations]: window['configs'].iterations,
    [METRICS.transparency]: window['configs'].transparency,
    [METRICS.rotationX]: window['configs'].rotateX,
    [METRICS.rotationY]: window['configs'].rotateY,
    [METRICS.rotationZ]: window['configs'].rotateZ,
    [METRICS.avgColor]: getAverageColor(grayScaleColors),
    [METRICS.medianColor]: getMedianColor(grayScaleColors),
    [METRICS.minColor]: getMinColor(grayScaleColors),
    [METRICS.maxColor]: getMaxColor(grayScaleColors),
  }
};

// just for debugging purposes
const debugImageCropping = (width: number, height: number,
imageData: ImageData) => {
  const outputCanvas = document.createElement('canvas');
  const outputContext = outputCanvas.getContext('2d');

  outputCanvas.width = width;
  outputCanvas.height = height;

  outputContext.putImageData(imageData, 0, 0);

  const summary = calculateMetrics(width, height, imageData);

  console.log('summary', summary);

  document.getElementById('scene-container').remove()
  document.body.appendChild(outputCanvas);

  debugger;
};

const saveSummary = async (width: number, height: number,
imageData: ImageData) => {
  const summary = calculateMetrics(width, height, imageData);

  const { ok } = await fetch(
    'http://localhost:3000/upload-summary', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ summary }),
    }
  );

  if (ok) {
    console.log('Дані збереженo!');
  }
}

const getImageDataOfPredefinedRect = (canvasEl:
HTMLCanvasElement) => {
  const context = canvasEl.getContext('webgl2');
  const serpinskiyTetraiderDimensions = { x: 220, y: 220, width: 560,
height: 420 };
  const customTetraiderDimensions = { x: 420, y: 300, width: 720,
height: 420 };
  const { x, y, width, height } = isCustomTetraider() ?
customTetraiderDimensions : serpinskiyTetraiderDimensions;

  const pixels = new Uint8Array(width * height * 4);

  context.readPixels(x, y, width, height, context.RGBA,
context.UNSIGNED_BYTE, pixels);

  return new ImageData(new Uint8ClampedArray(pixels), width,
height);
};

const cacheShadowImageData = (canvasEl: HTMLCanvasElement)
=> {
  const imageData = getImageDataOfPredefinedRect(canvasEl);
  const { width, height } = canvasEl;
  const rowsConturValues = [];
  const redMargin = 205;
  const greenMargin = 210;
  const blueMargin = 218;
  const isShadowStarts = (idx) => (
    imageData.data[idx] < redMargin && imageData.data[idx + 1] <
greenMargin && imageData.data[idx + 2] < blueMargin
  );

  for (let row = 0; row < width; row++) {
    for (let col = 0; col < height; col++) {
      const index = (col * width + row) * 4;

      if (isShadowStarts(index)) {
        if (!rowsConturValues[row]) {
          rowsConturValues[row] = {
            left: -1,
            right: -1,
          }
        }

        rowsConturValues[row].left = index;

        break;
      }
    }

    for (let col = height - 1; col >= 0; col--) {
      const index = (col * width + row) * 4;

      if (isShadowStarts(index)) {
        rowsConturValues[row].right = index;

        break;
      }
    }
  }

  window['_cachedContur'] = rowsConturValues;
};

const onSaveSummaryButtonClick = (canvasEl:
HTMLCanvasElement) => {
  const imageData = getImageDataOfPredefinedRect(canvasEl);

  // debugImageCropping(width, height, imageData);

  saveSummary(canvasEl.width, canvasEl.height, imageData);
};

const initSummarySaver = (canvasEl: HTMLCanvasElement) => {
  const saveSummaryButton = document.createElement('button');
  saveSummaryButton.style.position = 'absolute';
  saveSummaryButton.style.bottom = '10px';
  saveSummaryButton.style.right = '10px';
  saveSummaryButton.style.width = '150px';
  saveSummaryButton.style.height = '50px';
  saveSummaryButton.style.userSelect = 'none';
  saveSummaryButton.innerText = 'Зберегти загальні метрики';
  saveSummaryButton.classList.add('save_summary');

  saveSummaryButton.addEventListener('click', () =>
onSaveSummaryButtonClick(canvasEl));

  document.body.appendChild(saveSummaryButton);
};

const initGoToAnalyzerPageButton = () => {
```

```
  const analyzerURL = 'http://localhost:3000/';
  const goToAnalyzerPageButton =
document.createElement('button');
  goToAnalyzerPageButton.style.position = 'absolute';
  goToAnalyzerPageButton.style.bottom = '10px';
  goToAnalyzerPageButton.style.left = '170px';
  goToAnalyzerPageButton.style.width = '150px';
  goToAnalyzerPageButton.style.height = '50px';
  goToAnalyzerPageButton.style.userSelect = 'none';
  goToAnalyzerPageButton.innerText = 'Перейти до аналізу
показників прозорості';

  goToAnalyzerPageButton.addEventListener('click', () =>
window.open(analyzerURL))

  document.body.appendChild(goToAnalyzerPageButton);
};

const initSaveShadowImageToCache = (canvasEl:
HTMLCanvasElement) => {
  const cacheShadowImageDataButton =
document.createElement('button');
  cacheShadowImageDataButton.style.position = 'absolute';
  cacheShadowImageDataButton.style.bottom = '10px';
  cacheShadowImageDataButton.style.right = '170px';
  cacheShadowImageDataButton.style.width = '150px';
  cacheShadowImageDataButton.style.height = '50px';
  cacheShadowImageDataButton.style.userSelect = 'none';
  cacheShadowImageDataButton.innerText = 'Закешувати контур
тіні';

cacheShadowImageDataButton.classList.add('cache_shadow_contur'
);

  cacheShadowImageDataButton.addEventListener('click', () =>
cacheShadowImageData(canvasEl));

  document.body.appendChild(cacheShadowImageDataButton);
}

export const initActionButtons = (canvasEl: HTMLCanvasElement)
=> {
  initImageSaver(canvasEl);
  initGoToAnalyzerPageButton();
  initSaveShadowImageToCache(canvasEl);
  initSummarySaver(canvasEl);
};
```

Модуль: src/client/utils/metrics.ts

```
 export enum METRICS {
    figure ='figure',
    iterations = 'iterations',
    transparency = 'transparency',
    rotationX = 'rotationX',
    rotationY = 'rotationY',
    rotationZ = 'rotationZ',
    avgColor = 'avgColor',
    medianColor = 'medianColor',
    minColor = 'minColor',
    maxColor = 'maxColor',
}
```

1.10 Автоматизація збору метрик згенерованних зображень

Модуль: cypress/e2e/automation.ts

```
 import { METRICS } from '../../src/client/utils/metrics';

const MAIN_APP_URL = 'http://localhost:8080/';

const WAIT_TIME = 1000;

const VARIATIONS = {

  [METRICS.figure]: ['serpinsiy_tetraider',
'serpinsiy_tiered_tetraider', 'custom_tetraider'],

  rotation: [0, 0.8, 1.6, 2.4, Math.PI],

  [METRICS.iterations]: [1, 2, 3, 4, 5, 6, 7, 8],

  [METRICS.transparency]: [0.01, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
0.9, 0.99]

};


const selectors = {

  transparency: {

    primary: 'body > div.dg.ac > div:nth-child(1) > ul > li:nth-child(1)
> div > ul > li:nth-child(2) > div > ul > li:nth-child(3) > div > div >
div:nth-child(1) > input[type=text]',

    secondary: 'body > div.dg.ac > div:nth-child(2) > ul > li:nth-
child(1) > div > ul > li:nth-child(2) > div > ul > li:nth-child(3) > div
> div > div:nth-child(1) > input[type=text]'

  },

  rotation: {

    primary: {

      x: 'body > div.dg.ac > div:nth-child(1) > ul > li:nth-child(1) > div
> ul > li:nth-child(4) > div > ul > li:nth-child(2) > div > div >
div:nth-child(1) > input[type=text]',

      y: 'body > div.dg.ac > div:nth-child(1) > ul > li:nth-child(1) > div
> ul > li:nth-child(4) > div > ul > li:nth-child(3) > div > div >
div:nth-child(1) > input[type=text]',

      z: 'body > div.dg.ac > div:nth-child(1) > ul > li:nth-child(1) > div
> ul > li:nth-child(4) > div > ul > li:nth-child(4) > div > div >
div:nth-child(1) > input[type=text]'

    },

    secondary: {

      x: 'body > div.dg.ac > div:nth-child(2) > ul > li:nth-child(1) > div
> ul > li:nth-child(4) > div > ul > li:nth-child(2) > div > div >
div:nth-child(1) > input[type=text]',

      y: 'body > div.dg.ac > div:nth-child(2) > ul > li:nth-child(1) > div
> ul > li:nth-child(4) > div > ul > li:nth-child(3) > div > div >
div:nth-child(1) > input[type=text]',

      z: 'body > div.dg.ac > div:nth-child(2) > ul > li:nth-child(1) > div
> ul > li:nth-child(4) > div > ul > li:nth-child(4) > div > div >
div:nth-child(1) > input[type=text]'

    }

  }

}
```

```
const setRotation = (locator: string, rotation: number) => {
  cy.get(locator)
    .clear()
    .type(String(rotation))
    .trigger('change');
};

const setTransparency = (locator: string, transparency: number) => {
  cy.get(locator)
    .clear()
    .type(String(transparency))
    .trigger('change');
};

describe('template spec', () => {
  for (let f = 0; f < VARIATIONS[METRICS.figure].length; f++) {
    for (let rx = 0; rx < VARIATIONS['rotation'].length; rx++) {
      for (let i = 0; i < VARIATIONS[METRICS.iterations].length; i++) {
        for (let t = 0; t < VARIATIONS[METRICS.transparency].length; t++) {
          const figure = VARIATIONS[METRICS.figure][f];
          const rotation = VARIATIONS['rotation'][rx];
          const iterations = VARIATIONS[METRICS.iterations][i];
          const transparency = VARIATIONS[METRICS.transparency][t];

          it('automates metrics summary collection', () => {
            cy.viewport(2560, 1315);
            cy.visit(MAIN_APP_URL);

            cy.get('#figure-type').select(figure)
              .get('#range').type(String(iterations))
              .get('#generate-fractal-button').click();

            cy.wait(WAIT_TIME);

            setRotation(selectors.rotation.primary.x, rotation);
            setRotation(selectors.rotation.primary.y, rotation);
            setRotation(selectors.rotation.primary.z, rotation);

            const isInnerFractalAvailable = figure === 'serpinsiy_tiered_tetraider';
            if (isInnerFractalAvailable) {
              setRotation(selectors.rotation.secondary.x, rotation);
              setRotation(selectors.rotation.secondary.y, rotation);
              setRotation(selectors.rotation.secondary.z, rotation);
            }

            cy.get('.cache_shadow_contur').click();

            cy.wait(WAIT_TIME);

            setTransparency(selectors.transparency.primary, transparency);

            if (isInnerFractalAvailable) {
              setTransparency(selectors.transparency.secondary, transparency);
            }

            cy.wait(WAIT_TIME);

            cy.get('.save_summary').click();

            cy.wait(WAIT_TIME);
          });
        }
      }
    }
  }
});

// npx cypress run automation
// npx cypress run automation --headed
```

2 ТЕКСТ ПРОГРАМИ – ЗБІРЩИК ПОКАЗНИКІВ ПРОЗОРОСТІ ФРАКТАЛІВ

2.1 Клієнтська частина програми

index.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>Розробка класифікатора для розпізнавання тексту</title>
  <link rel="stylesheet" href="./styles.css">
  <script src="./app.js" defer></script>
 </head>

 <body>
  <header>
   <h1>Дослідження моделей оптичних перетворень негладких фрактальних поверхонь</h1>
  </header>
  <main>
   <section class="image_block">
    <div>
     <span class="title">Зображення</span>
     <input type="file" accept=".png" class="upload_button" onchange="onUploadInputChange(event)" />
    </div>
    <div class="original_image image_container">
     <canvas id="original_canvas" width="1920" height="1080"></canvas>
    </div>
   </section>
   <section class="metrics_block">
    <span>Метрики:</span>
    <textarea name="letter_metrics_logger" id="letter_metrics_logger" cols="100" rows="20" readonly></textarea>
   </section>
   <section class="buttons_section">
    <button onclick="calculate_metrics()" class="button calculate_metrics">Порахувати метрики</button>
    <button onclick="save_data()" class="button save_data">Зберегти дані</button>
   </section>
  </main>
 </body>
</html>
```

Модуль: styles.css

```
body {
 display: grid;
 justify-items: center;
 /* grid-template-rows: 42px 1fr 198px; */
 background: linear-gradient(#26C794, #31659E);
 padding: 0 40px 17px 40px;
 font-size: larger;
}

main {
 position: relative;
 display: grid;
 /* width: 100%; */
 /* grid-template-columns: 1fr 20px 1fr; */
 gap: 20px;
}

header {
 line-height: 0px;
}

.image_block {
 position: relative;
 display: inline-block;
 text-align: center;
 grid-template-rows: 40px 1fr;
}

.image_container {
 position: relative;
 /* height: 82vh; */
 /* max-width: 1920px; */
 /* max-height: 1080px; */
 background-color: aliceblue;
 border-radius: 20px;
 /* overflow: scroll; */
```

```
}

.title {
  font-size: large;
  font-weight: bold;
}

.button {
  border-radius: 8px;
  padding: 10px;
  user-select: none;
  height: 40px;
  background-color: #f8ff41ed;
}

.upload_button {
  padding: 10px;
}

.save_file {
  position: absolute;
  top: 10px;
  right: 10px;
}

.letter_metrics_logger {
  border-radius: 10px;
}

.metrics_block {
  display: grid;
  text-align: center;
}

.buttons_section {
  position: fixed;
  top: 10px;
  right: 10px;
}
```

Модуль: app.js

```
 const INITIAL_WIDTH = 600;
const INITIAL_HEIGHT = 800;
const MAX_ZOOM_VALUE = 8;
const MIN_ZOOM_VALUE = 1;
const ZOOM_MULTIPLICATOR = 2;
const VALUES_PER_PIXEL = 4;
const SELECTION_LINE_WIDTH = 2;
const WHITE = 255;
const BLACK = 0;
const canvas = document.getElementById('original_canvas');
const context =
document.getElementById('original_canvas').getContext('2d', {
willReadFrequently: true });
const zoomLabel = document.getElementById('zoom_label');
let currentZoomValue = MIN_ZOOM_VALUE;
let originalImage = null;
let currentTreeshold = 0;
let isDragging = false;
let initialMouseX, initialMouseY;
let selectedRectengle = {};
const grayScaleColorToCount = {};
const pixelsColors = [];
const mainAppURL = 'http://localhost:8080/';
let filename = '';

const onUploadInputChange  = ({ target }) => {
  const imageFile = target.files[0];
  const reader = new FileReader();

  filename = imageFile.name.replaceAll('.png', '');

  reader.readAsDataURL(imageFile);
  reader.onloadend = onReaderLoadEnd;
};

const onReaderLoadEnd = (event) => {
  originalImage = new Image();
  originalImage.src = event.target.result;
```

```
  originalImage.onload = onImageLoad;
};

const onImageLoad = () => {
  const scale_factor = Math.min(canvas.width / originalImage.width, canvas.height / originalImage.height);
  const width = originalImage.width * scale_factor;
  const heigth = originalImage.height * scale_factor;
  const x = (canvas.width / 2) - (width / 2);
  const y = (canvas.height / 2) - (heigth / 2);

  context.drawImage(originalImage, x, y, width, heigth);

  const imageData = context.getImageData(0, 0, canvas.width, canvas.height);
  const resultData = new ImageData(canvas.width, canvas.height);

  for (let i = 0; i < imageData.data.length; i += VALUES_PER_PIXEL) {
    const [red, green, blue, alpha] = [i, i + 1, i + 2, i + 3];

    resultData.data[red] = imageData.data[red];
    resultData.data[green] = imageData.data[green];
    resultData.data[blue] = imageData.data[blue];
    resultData.data[alpha] = imageData.data[alpha];
  }

  context.putImageData(resultData, 0, 0);
};

const drawResults = (width, height) => {
    context.strokeStyle = 'red';
    context.lineWidth = SELECTION_LINE_WIDTH;
    context.strokeRect(initialMouseX, initialMouseY, width, height);
};

canvas.addEventListener('mousedown', (e) => {
  const mouseX = e.clientX - canvas.getBoundingClientRect().left;
  const mouseY = e.clientY - canvas.getBoundingClientRect().top;
    isDragging = true;
    initialMouseX = mouseX;
    initialMouseY = mouseY;
});

canvas.addEventListener('mouseup', (e) => {
    if (isDragging) {
        const mouseX = e.clientX - canvas.getBoundingClientRect().left;
        const mouseY = e.clientY - canvas.getBoundingClientRect().top;
        const minX = Math.min(initialMouseX, mouseX);
        const minY = Math.min(initialMouseY, mouseY);
        const maxX = Math.max(initialMouseX, mouseX);
        const maxY = Math.max(initialMouseY, mouseY);
        const topLeft = { x: minX, y: minY };
        const topRight = { x: maxX, y: minY };
        const bottomLeft = { x: minX, y: maxY };
        const bottomRight = { x: maxX, y: maxY };

        selectedRectengle = { topLeft, topRight, bottomLeft, bottomRight, width: maxX - minX, height: maxY - minY }

        drawResults(selectedRectengle.width, selectedRectengle.height);
    }

    isDragging = false;
});

const getAverageColor = (colors) => {
  const total = colors.reduce((result, currentColor) => result + currentColor, 0);

  return Number(total / colors.length).toFixed(2);
};

const getMedianColor = (colors) => {
  const sortedColors = [...colors].sort((left, right) => right - left);
  const medianColor = sortedColors[Math.round(sortedColors.length / 2)];
```

```
  return medianColor;
};

const getMinColor = (colors) => Math.min(...Array.from(new
Set(colors)));

const getMaxColor = (colors) => Math.max(...Array.from(new
Set(colors)));

const calculate_metrics = () => {
  const { width, height } = selectedRectengle;
  const { x, y } = selectedRectengle.topLeft;
  const imageData = context.getImageData(x +
SELECTION_LINE_WIDTH, y + SELECTION_LINE_WIDTH,
width - SELECTION_LINE_WIDTH, height -
SELECTION_LINE_WIDTH);

  for (let i = 0; i < imageData.data.length; i += 4) {
    const red = imageData.data[i];
    const green = imageData.data[i + 1];
    const blue = imageData.data[i + 2];

    const grayScaleColor = Number((0.299 * red + 0.587 * green +
0.114 * blue).toFixed(2));

    if (grayScaleColorToCount[grayScaleColor]) {
      grayScaleColorToCount[grayScaleColor] += 1;
    } else {
      grayScaleColorToCount[grayScaleColor] = 1;
    }

    pixelsColors.push(grayScaleColor);
  }

  const loggerEl =
document.getElementById('letter_metrics_logger');

  loggerEl.value =  `СЕРЕДНІЙ_КОЛІР =
${getAverageColor(pixelsColors)} \nМЕДІАННИЙ_КОЛІР =
${getMedianColor(pixelsColors)}
\nМІНІМАЛЬНЕ_ЗНАЧЕННЯ_КОЛЬОРУ =
${getMinColor(pixelsColors)}
\nМАКСИМАЛЬНЕ_ЗНАЧЕННЯ_КОЛЬОРУ =
${getMaxColor(pixelsColors)} \n`;
  Object.keys(grayScaleColorToCount).forEach((color) => {
    loggerEl.value += `КОЛІР = ${color}; КІЛЬКІСТЬ_ПІКСЕЛІВ
= ${grayScaleColorToCount[color]} \n`;
  });
};

const save_data = async () => {
  const { ok } = await fetch(
    '/upload-analysis', {
      method: 'POST',
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        filename,
        average: getAverageColor(pixelsColors),
        median: getMedianColor(pixelsColors),
        min: getMinColor(pixelsColors),
        max: getMaxColor(pixelsColors),
        colorToCount: grayScaleColorToCount
      })
    }
  );

  if (ok) {
    window.alert('Дані збережено!')
  }
};
```

Модуль: package.json

```
{
  "dependencies": {
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "express-fileupload": "^1.4.2"
  }
```

2.2 Серверна частина програми

Модуль: server.js

```
const express = require('express');
const fs = require('fs');
```

```
const cors = require('cors');
const fileUpload = require('express-fileupload');
const path = require('path');
const app = express();
const port = 3000;
const dataFolderPath = './data';
const configsToUkr = {
  figureType: 'ТИП_ФІГУРИ',
  transparency: 'ПРОЗОРІСТЬ',
  iterations: 'КІЛЬКІСТЬ_ІТЕРАЦІЙ',
  rotateX: 'ПОВОРОТ_ВІСЬ_X',
  rotateY: 'ПОВОРОТ_ВІСЬ_Y',
  rotateZ: 'ПОВОРОТ_ВІСЬ_Z',
};

app.use(cors());
app.use(express.json({limit: '25mb'}));
app.use(express.urlencoded({limit: '25mb'}));
app.use(express.static('public'));

app.use(fileUpload({
  limits: { fileSize: 10 * 1024 * 1024 },
}));

if (!fs.existsSync(dataFolderPath)) {
  fs.mkdirSync(dataFolderPath);
}

const SUMMARY_FILE = path.join(__dirname, 'data',
'ЗАГАЛЬНІ_РЕЗУЛЬТАТИ.txt')

fs.writeFileSync(SUMMARY_FILE,
'ФІГУРА;КІЛЬКІСТЬ_ІТЕРАЦІЙ;ПРОЗОРІСТЬ;ПОВОРОТ_X;П
ОВОРОТ_Y;ПОВОРОТ_Z;СЕРЕДНІЙ_КОЛІР;МЕДІАННИЙ_К
ОЛІР;МІНІМАЛЬНЕ_ЗНАЧЕННЯ_КОЛЬОРУ;МАКСИМАЛЬН
Е_ЗНАЧЕННЯ_КОЛЬОРУ\n', 'utf8');

const getConfigsString = (configs) =>
Object.keys(configsToUkr).reduce((result, configName) => result +
`${configsToUkr[configName]}=${configs[configName]} \n`, '');

const getMetricsString = (metrics) => Object.entries(metrics)
  .sort(([colorLeft], [colorRight]) => colorLeft - colorRight)
  .reduce((result, [color, count]) => result + `${color}=${count} \n`,
'');

app.post('/upload-image-with-configs', (req, res) => {
  if (!req.body || !req.body.image) {
    return res.status(400).json({ error: 'Image data is required' });
  }

  const { filename, configs, image } = req.body;
  const imageBinary = Buffer.from(image.split(',')[1], 'base64');
  const folderPath = path.join(__dirname, 'data', filename);

  if (!fs.existsSync(folderPath)) {
    fs.mkdirSync(folderPath, { recursive: true });
  }

  fs.writeFileSync(path.join(folderPath, `${filename}.png`),
imageBinary);

  fs.writeFileSync(path.join(folderPath,
`${filename}_КОНФІГУРАЦІЇ.txt`), getConfigsString(configs));

  res.status(201).json({ message: 'Image uploaded to the server
successfully' });
});

app.post('/upload-analysis', (req, res) => {
  if (!req.body || !req.body.colorToCount || !req.body.median ||
!req.body.average) {
    return res.status(400).json({ error: 'Image data is required' });
  }

  const { filename, colorToCount, average, median, min, max } =
req.body;
  const folderPath = path.join(__dirname, 'data', filename);

  if (!fs.existsSync(folderPath)) {
    fs.mkdirSync(folderPath, { recursive: true });
  }
```

```
  let analysisString = `СЕРЕДНІЙ_КОЛІР=${average}
\nМЕДІАННИЙ_КОЛІР=${median}
\nМІНІМАЛЬНЕ_ЗНАЧЕННЯ_КОЛЬОРУ=${min}\nМАКСИМА
ЛЬНЕ_ЗНАЧЕННЯ_КОЛЬОРУ=${max}\n${getMetricsString(colo
rToCount)}`;


  fs.writeFileSync(path.join(folderPath,
`${filename}_МЕТРИКИ.txt`), analysisString);


  res.status(201).json({ message: 'Image uploaded to the server
successfully' });

});


app.post('/upload-summary', async (req, res) => {

  if (!req.body || !req.body.summary) {

    return res.status(400).json({ error: 'Summary data is required' });

  }

  const { figure, iterations, transparency, rotationX, rotationY,
rotationZ, avgColor, medianColor, minColor, maxColor } =
req.body.summary;


  try {

    await fs.appendFileSync(

      SUMMARY_FILE,


`${figure};${iterations};${transparency};${rotationX};${rotationY}
;${rotationZ};${avgColor};${medianColor};${minColor};${maxCol
or}\n`,

      'utf8'

    );

  } catch (error) {

    console.error(error);

  }


  res.status(201).json({ message: 'Summary uploaded to the server
successfully' });

});


app.listen(port, () => {

  console.log(`Server is listening on port ${port}`);
```