Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Системотехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти *другий (магістерський)*

Проєктування системи адаптивної генерації елементів ігрового простору
(тема)

Виконав:
Студент 2 курсу, групи *ІТПм-21-2*
Ковальов К.В.
(прізвище, ініціали)
Спеціальність *122 Комп'ютерні науки*
(код і повна назва спеціальності)
Тип програми *освітньо-професійна*
(освітньо-професійна або освітньо-наукова)
Освітня програма *Інформаційні технології проектування*
(повна назва освітньої програми)
Керівник *доц. Губаренко Є.В.*
(посада, прізвище,ініціали)

Допускається до захисту
Зав. кафедри системотехніки _____ *проф. Гребеннік І.В.*
(підпис) (прізвище, ініціали)

2022 р.

Харківський національний університет радіоелектроніки

Факультет_____*Комп'ютерних наук*_____

Кафедра_____*Системотехніки*_____

Рівень вищої освіти_____*другий (магістерський)*_____

Спеціальність _____*122 Комп'ютерні науки*_____
(код і повна назва)

Освітня програма _____*Інформаційні технології проектування*_____
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _____*Ковальову Костянтину Віталійовичу*_____
(прізвище, ім'я, по батькові)

1. Тема роботи _*Проєктування системи адаптивної генерації елементів*_____
*ігрового простору*_____
_____
затверджена наказом по університету від *21.11.2022р. № 1504Ст*
2. Термін подання студентом роботи до екзаменаційної комісії 20.12.2022 р.
3. Вихідні дані до роботи ___*Науково-технічні публікації, дані Інтернет-джерел та відомих проектів щодо розробки ігрових додатків, GLSL documentation, Shadertoy documentation*_____
_____
_____
_____
_____
_____
_____
4. Перелік питань, що потрібно опрацювати в роботі *Вступ. Аналіз предметної області та постановка задачі. Інструменти дослідження. Розробка програмного забезпечення. Висновки*_____
_____

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) *Візуалізація адаптивної генерації ігрового простору, Інтерфейс програмного забезпечення, Використані алгоритми шуму*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	*Отримання завдання на виконання роботи*	*24.11.2022*	
2	*Огляд літератури та аналіз предметної області*	*26-28.11.2022*	
3	*Огляд та вибір методу розв'язання задачі*	*29.11.2022*	
4	*Вибір мови та середовища розробки ПЗ*	*30.11.2022*	
5	*Реалізація алгоритмів складання списку заходів*	*01.12.2022*	
6	*Проведення експерименту*	*2-3.12.2022*	
7	*Обробка результатів*	*3.12.2022*	
8	*Оформлення пояснювальної записки*	*4-9.12.2022*	
9	*Оформлення додатків*	*10-14.12.2022*	
10	*Представлення на рецензування*	*14.12.2022*	

Дата видачі завдання 24.11.2022 р.

Студент _____ *Ковальов К.В.*

(підпис)

Керівник роботи _____ *доц. Губаренко Є.В.*

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи магістра містить: 46 с., 17 рис., 15 джерел інформації.

ВІДЕОГРА, АДАПТИВНА ГЕНЕРАЦІЯ, ПРОЦЕДУРНА ГЕНЕРАЦІЯ ЛАНДШАФТУ.

Об’єктом досліджень є процес імплементації адаптивної генерації в ігровий простір.

Предметом досліджень є комп'ютерні ігри з процедурною генерацією

Мета дослідження – дослідження існуючих способів адаптивної генерації, виявлення найефективнішого алгоритму.

Методи дослідження – аналіз теоретичного матеріалу, аналіз технічної літератури та практична реалізація за допомогою методів об’єктно орієнтованої розробки.

В ході роботи було розглянуте поняття комп'ютерної гри з процедурної генерацією, також виявлено три основних методів процедурної генерації та їх застосування. Було проведено аналіз існуючих безкоштовних ігрових двигунів. Був розроблений програмний додаток.

ABSTRACT

Explanatory note to the certification work of the master's degree contains: 46 p., 17 fig., 15 sources of information.

VIDEO GAME, ADAPTIVE GENERATION, PROCEDURAL LANDSCAPE GENERATION.

The object of research is the process of implementing adaptive generation in the game space.

The subject of research is computer games with procedural generation

The purpose of the research is to study the existing methods of adaptive generation, to identify the most effective algorithm.

Research methods – analysis of theoretical material, analysis of technical literature and practical implementation using methods of object-oriented development.

In the course of the work, the concept of a computer game with procedural generation was considered, three main methods of procedural generation and their application were also revealed. An analysis of existing free game engines was conducted. A software application was developed.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ ТА ТЕРМІНІВ

VR – Virtual Reality

SDF – Signed distance functions

FOV – Field of Fiew

OpenGL – Open Graphics Library

GLSL – OpenGL Shader Language

ВСТУП

Індустрія відеоігор почала розвиватися в середині 1970-х років і на сьогоднішній день має право називатися однією з найприбутковіших індустрій, що має річний прибуток більше ніж 100 мільярдів доларів. Стрімкий ріст популярності відеоігор пояснюється в основному легкодоступністю до них. Щоб почати грати потрібно всього лиш мати майже будь-який електронний девайс: комп'ютер, смартфон або ігрову приставку, та копію гри, отримати яку можна як у фізичному магазині, так і за допомогою Інтернету. Також, вибір гри не є важкою задачею для споживача – в магазинах є багато фільтрів для потрібних жанрів, а на сторінках самих ігор (або на фізичних коробках) є багато інформації про них та зображення геймплею.

На сьогоднішній день індустрія ігор продвинулася настільки вперед, що може створювати ігри з графікою, що не відрізняється від реального світу, що використовується при розробці комплексних симуляторів та програм для навчання. Наприклад, вже зараз пілотів навчають в гіпер-реалістиних іграх, замість кабін реальних літальних апаратів.

В Україні ігрова індустрія розвинута дуже слабо. Навіть при доволі високому попиті на відеоігри, існують дуже небагато українських компаній, що можуть конкурувати з іноземними.

Тому розвинення технологій в цьому напрямі можна вважати перспективним.

В дипломному проекті досліджувалися методи адаптивної генерації в ігровому просторі, алгоритми генерації шуму та способи відображення ландшафту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття інформаційної системи, її ознаки та класифікація

Інформаційна система (ІС) визначається з двох точок зору: одна стосується її функції; інша стосується її структури. З функціональної точки зору; інформаційна система – це технологічно реалізоване середовище для запису, зберігання та розповсюдження мовних виразів, а також для підтримки створення висновків. Зі структурної точки зору; інформаційна система складається з набору людей, процесів, даних, моделей, технологій і частково формалізованої мови, утворюючи цілісну структуру, яка служить певній організаційній меті або функції. Функціональне визначення має свої переваги в тому, що зосереджується на тому, що фактичні користувачі – з концептуальної точки зору – роблять з інформаційною системою під час її використання. Вони спілкуються з експертами, щоб вирішити певну проблему. Структурне визначення чітко пояснює, що ІС є соціально-технічними системами, тобто системами, що складаються з людей, правил поведінки та концептуальних і технічних артефактів.

Технічно інформаційну систему можна визначити як набір взаємопов’язаних компонентів, які збирають (або отримують), обробляють, зберігають і поширюють інформацію для підтримки прийняття рішень і контролю в організації. Окрім підтримки прийняття рішень, координації та контролю, інформаційні системи також можуть допомогти менеджерам і працівникам аналізувати проблеми, візуалізувати складні теми та створювати нові продукти. Три дії в інформаційній системі виробляють інформацію, необхідну організаціям для прийняття рішень, контролю операцій, аналізу проблем і створення нових продуктів або послуг. Це вхід, обробка та вихід. Введення фіксує або збирає необроблені дані всередині організації або з її зовнішнього середовища. Обробка перетворює ці вихідні дані в більш значущу форму. Вихід передає оброблену

інформацію людям, які її використовуватимуть, або діяльності, для якої вона використовуватиметься. Інформаційні системи також вимагають зворотного зв'язку, тобто вихідних даних, які повертаються до відповідних членів організації, щоб допомогти їм оцінити або виправити етап введення.

Комп’ютерна інформаційна система – це інформаційна система, яка використовує комп’ютерну технологію для виконання деяких або всіх запланованих завдань. Така система може містити лише персональний комп’ютер і програмне забезпечення. Або він може включати кілька тисяч комп'ютерів різного розміру з сотнями принтерів, плотерів та інших пристроїв, а також мережі зв'язку (дротові та бездротові) і бази даних. У більшості випадків інформаційна система також включає людей. Основні компоненти інформаційних систем перераховані нижче. Зауважте, що не кожна система містить усі ці компоненти.

Компоненти інформаційних систем:

- ресурси людей (кінцеві користувачі та спеціалісти з ІБ, системний аналітик, програмісти, адміністратори даних тощо);
- апаратне забезпечення (фізичне комп’ютерне обладнання та пов’язані пристрої, машини та засоби масової інформації);
- програмне забезпечення (програми та процедури);
- дані (бази даних і знань);
- мережі (комунікаційні засоби та мережева підтримка).

Людські ресурси:

- кінцеві користувачі (їх також називають користувачами або клієнтами) – це люди, які використовують інформаційну систему або інформацію, яку вона створює. Вони можуть бути бухгалтерами, продавцями, інженерами, клерками, клієнтами або менеджерами. Більшість із нас є кінцевими користувачами інформаційних систем;
- фахівці з ІБ – люди, які фактично розробляють і керують інформаційними системами. До них входять системні аналітики, програмісти, тестувальники, оператори комп’ютерів та інший управлінський, технічний і канцелярський персонал ІБ. Коротко кажучи, системні аналітики проектують

інформаційні системи на основі інформаційних вимог кінцевих користувачів, програмісти готують комп’ютерні програми на основі специфікацій системних аналітиків, а комп’ютерні оператори керують великими комп’ютерними системами.

Апаратні ресурси:

- машини – комп'ютери та інше обладнання разом з усіма носіями даних, об'єкти, на яких дані записуються та зберігаються;
- комп'ютерні системи (складаються з різноманітних взаємопов'язаних периферійних пристроїв). Прикладами є мікрокомп’ютерні системи, комп’ютерні системи середнього класу та великі комп’ютерні системи.

Ресурси програмного забезпечення включають усі набори інструкцій з обробки інформації. Це загальне поняття програмного забезпечення включає не тільки програми, які направляють і контролюють комп'ютери, але також і набори обробки інформації (процедури).

Програмні ресурси включають:

- системне програмне забезпечення, наприклад операційна система;
- прикладне програмне забезпечення – програми, які керують обробкою для певного використання комп’ютерів кінцевими користувачами;
- процедури, які є інструкціями з експлуатації для людей, що будуть використовувати інформаційну систему. Прикладами є інструкції щодо заповнення паперової форми або використання певного програмного пакета.

Ресурси даних включають дані які є сировиною інформаційних систем і базу даних. Дані можуть приймати різні форми, включаючи традиційні буквено-цифрові дані, що складаються з чисел, літер та інших символів, які описують бізнес-операції та інші події та сутності. Текстові дані, що складаються з речень і абзаців, які використовуються в письмовому спілкуванні; дані зображення, такі як графічні форми та фігури; аудіодані, людський голос та інші звуки також є важливими формами даних.

Ресурси даних повинні відповідати таким критеріям:

- повнота – всі дані про суб'єкта фактично присутні в базі даних;

- відсутність надмірності – кожна окрема частина даних існує лише один раз у базі даних;

- відповідна структура – дані зберігаються таким чином, щоб мінімізувати вартість очікуваної обробки та зберігання.

Мережеві ресурси.

Телекомунікаційні мережі, такі як Інтернет, інтранет та екстранет, стали необхідними для успішної роботи всіх типів організацій і їх комп'ютерних інформаційних систем. Телекомунікаційні мережі складаються з комп’ютерів, комунікаційних процесорів та інших пристроїв, з’єднаних між собою засобами зв’язку та керованих комунікаційним програмним забезпеченням. Концепція мережевих ресурсів підкреслює, що комунікаційні мережі є основним ресурсним компонентом усіх інформаційних систем.

Ресурси мережі включають:

- засоби зв'язку такі як вита пара, коаксіальний кабель, волоконно-оптичний кабель, мікрохвильові системи та супутникові системи зв'язку;

- підтримка мережі. Ця загальна категорія включає всіх людей, апаратне забезпечення, програмне забезпечення та ресурси даних, які безпосередньо підтримують роботу та використання мережі зв’язку. Приклади включають програмне забезпечення для керування зв’язком, таке як мережеві операційні системи та Інтернет-пакети.

В даній кваліфікаційній роботі буде розглянута проблема, яка стосується проєктування систем адаптивної генерації елементів ігрового простору.

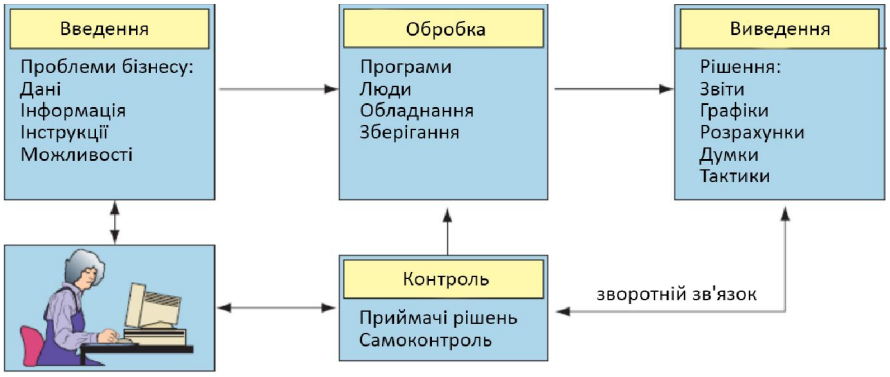

Рисунок 1.1 – Функції інформаційної системи

1.2 Поняття систем адаптивної генерації так їх ознаки

Що таке комп`ютерна гра? Комп'ютерна гра – це впорядкована послідовність системних команд, мета якої забезпечити хороше дозвілля із застосуванням комп'ютера. Вона складається з декількох компонентів: геймплей, ігровий простір, персонажі, візуалізація та звук. Якщо більшість елементів майже завжди треба створювати вручну, то ігровий простір, в який входять ландшафт та будівлі, можна створити за допомогою статичної або адаптивної генерації.

Адаптивна генерація – генерація, при якій елементи ігрового простору генеруються на льоту навколо рухомої точки огляду, гравця. Вона є протилежним підходом до статичної генерації, при якій швидкість генерації не важлива. Ігровий світ зазвичай являється нескінченним, а регіони, які створюються, недетерміновані. Він відтворюється поступово навколо глядача.

Яскравим представником з цією технологією є гра "Minecraft" від розробника Mojang. Гравець асоціюється радіусом перегляду. При розширенні від гравця радіус (відстань огляду) використовується для формування круглої чи квадратної області, усередині якої гарантовано буде створюватися та відтворюватися ландшафт.

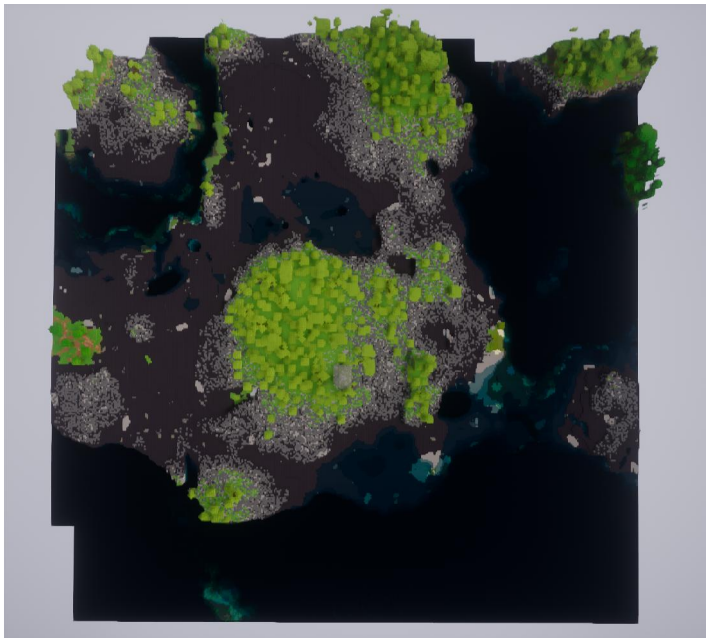

Рисунок 1.2 – Радіус адаптивної генерації світу в грі "Minecraft"

В створенні ігрового населеного світу з адаптивною генерацією є багато переваг: нескінченні розмір, варіація та деталізація; може використовувати дані гравця або системи як частини процедури генерації.

Перший пункт є типовою мотивацією процедурної генерації в іграх. Розробник може створити наповнення, щоб запропонувати гравцеві нескінченну різноманітність ігор та/або нескінченний простір для його дослідження. Завдяки реалізації алгоритму під час виконання система гравця має повну свободу генерувати світ.

Використання поведінки та ситуації гравця як вхідних даних для генерації є інтригуючим і значною мірою невивченим шляхом. Одним із прикладів може бути використання дій гравця в одній грі для створення історії світу в наступній грі.

Адаптивна генерація не позбавлена недоліків: складність обчислень впливає на гравця, чим більш детальний світ, тим довше генерація; обмежений контроль над результатом генерації; складність алгоритму зазвичай зростає, чим більше ідей та намірів в світі; алгоритми зазвичай мають бути написані тією ж мовою, що й движок гри.

1.3 Використання алгоритмів шуму для адаптивної генерації

В адаптивній генерації, будучи рішенням в реальному часі, дуже важлива швидкість і продуктивність. В ній вимагається використовувати достатньо оптимізовані генератори шуму для вирішення тих чи інших завдань та щоб система належним чином керувала всіма згенерованими даними, які потрібно відобразити.

У повсякденному житті шум вважається неприємним фактором, який заглушає інші, більш значущі елементи, на яких можна зосередитися. Однак для певних цілей це може бути корисним, а в контексті відеоігор і створення фільмів шум служить основою, на якій базується процедурна генерація ландшафту. Для

використання шуму в розробці ігор були створені різні алгоритми, деякі страждають від якості. Це здебільшого тому, що вони не фрактальні за своєю природою (тобто самоподібність не існує на різних рівнях масштабування).

Алгоритм Midpoint Displacement є фрактальним і базується на рекурсивному підході до генерації даних. Алгоритм простий у реалізації та оцінюється як найшвидший з усіх п’яти. Однак це має певну ціну, оскільки пам’ять стає обмеженням під час виконання, оскільки алгоритм вимагає, щоб усі згенеровані дані зберігалися в пам’яті, щоб він міг обчислити наступне випадкове значення. Причина полягає в тому, що кожне згенероване значення базується на кількох інших, уже згенерованих значеннях

Алгоритм Diamond-Square є вдосконаленим алгоритмом Midpoint Displacement. Що стосується продуктивності, то немає великої різниці, як процесора, так і пам'яті. Однак він виправляє деякі артефакти, внесені попереднім алгоритмом, і тому здатний давати якісніші результати. Загалом це легко реалізувати, якщо ви вже знайомі зі зміщенням середньої точки.

Порівняно з двома попередніми алгоритмами, Value Noise не є фрактальним. Його можна реалізувати для підвищення продуктивності пам’яті завдяки можливості створювати значення на льоту та не зберігати результат усередині. Це повільніше, ніж Midpoint Displacement, але загалом дуже швидке. Швидкість і якість обернено пропорційні, а це означає, що для отримання кращої якості це має відбуватися за рахунок швидкості. Внутрішньо алгоритм використовує функцію інтерполяції (лінійну, косинусну та кубічну), і залежно від того, яка з них обрана, швидкість і якість змінюватимуться. Алгоритм, як правило, простий для розуміння та реалізації, а реалістичність рельєфу можна підвищити за допомогою fBm (що також легко зробити).

Perlin Noise – це алгоритм генерації шуму, який став галузевим стандартом, оскільки він став методом вибору при роботі з даними про висоту. Зазвичай він повільніший, ніж Value Noise, хоча якщо використовується кубічна інтерполяція, він набагато швидший. Він добре працює для менших розмірів (1D, 2D, 3D), але більше, ніж це, призводить до значного сповільнення. Що стосується пам’яті,

йому потрібно зберігати таблицю градієнтів і таблицю перестановок у пам’яті, але, крім цього, він не сильно відрізняється з точки зору споживання пам’яті від Value Noise. Якість алгоритму також хороша, і розуміння ідеї, що лежить в його основі, вважається нескладним завданням. Однак реалізувати алгоритм складніше, ніж попередні три.

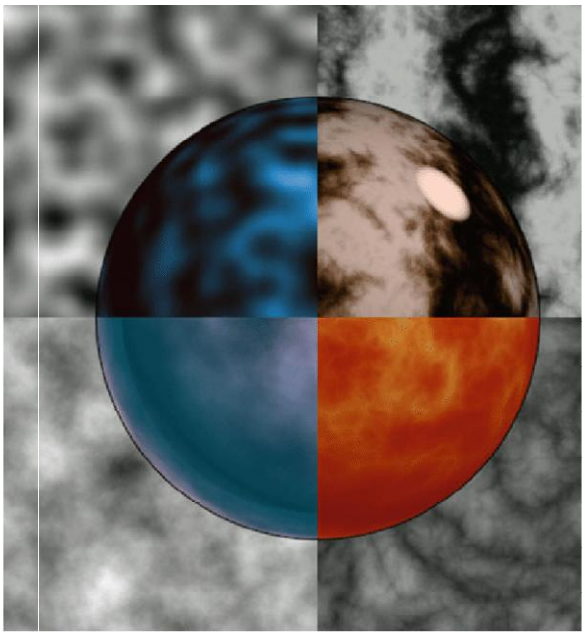

Рисунок 1.3 – Різні версії шуму, як описано Перліним

1.4 Постановка задачі дослідження

Головними елементами адаптивної генерації ландшафту завжди являлися математика і також шум, який ми створюємо за допомогою неї. За допомогою шуму ми можемо створити карту висоти і використовувати її для створення 3D об’єму та ландшафту, але є різні алгоритми адаптивної генерації їх:

- Marching Cubes – дискретизує тривимірний об’єм у сітку комірок і обчислює для кожної комірки набір примітивів на основі значень неявної функції її кутів. Менш детальний, але дуже легкий для відображення в системі користувача.
- Ray Marching – алгоритм проходиться по скалярному полю, бере вісім сусідніх точок (формуючи уявний куб), потім визначає які полігони потрібні щоб

представити частину ізоповерхні, що проходить крізь цей куб. Потім всі полігони з'єднуються в бажану поверхню. Може відображати найбільше деталей, але дуже затратний в продуктивності системи.

- Surface Nets – бере звичайний кубоподібний воксельний світ і просто ітеративно згладжує всі кути.

Об’єктом дослідження в рамках магістерської атестаційної роботи є проєктування системи адаптивної генерації елементів ігрового простору.

Предметом дослідження являються методи генерації елементів ігрового простору.

Метою даної роботи є дослідження методів генерації ландшафту в комп’ютерних іграх для:

- зменшення затрат при створенні ігрового світу;
- збільшення розміру світу до нескінченності;
- виявлення найшвидшого та найефективнішого способу генерації.

Для досягнення поставленої мети, необхідно дослідити наступні питання:

- дослідити алгоритми генерації шуму;
- виконати аналіз доцільності алгоритмів генерації шуму у процесах
- вибір оптимального алгоритму адаптивної генерації ігрового світу;
- порівняти алгоритми адаптивної генерації ігрового світу;
- обґрунтувати і вибрати оптимальний алгоритм генерації ігрового світу.

1.5 Висновки

Процедурна генерація пройшла довгий шлях в іграх, від 30 років тому з Elite до недавньої No Man’s Sky. Ігри частіше ніж будь-коли, починають використовувати адаптивну генерацію вмісту як фундаментальний механізм для заповнення ігрового середовища цікавими об’єктами та функціями, щоб запропонувати вміст цілої галактики з мінімальними даними. Але привабливість нескінченної галактики зникає. Якщо у вашій галактиці недостатньо новизни, навіть якщо у вас у грі мільярди світів, гравець побачить закономірності й почне

нудьгувати.

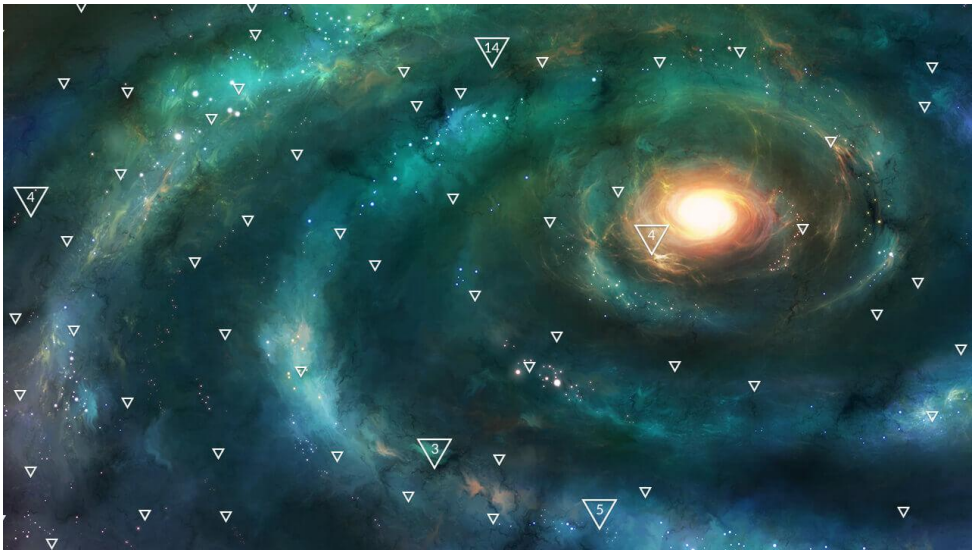

Рисунок 1.4 – Гра "No Man’s Sky" генерує 255 унікальних галактик, які складаються з 4.2 мільярдів регіонів (ліміт 32-бітних цілих чисел)

Якщо нас не приваблює обіцянка нескінченного вмісту, переваги динамічного покоління починають зменшуватися. Процедурна генерація – це машина для генерування нескінченного вмісту, але вона також може бути набагато більшою.

Прикладів багато:

- використовуйте кластер для моделювання складної та насиченої історії вашого ігрового світу;
- створіть гру, яка генерує свої рівні на суперкомп’ютері, а потім щодня розповсюджує цей новий світ своїм гравцям;
- статично генерувати набір грубих рівнів, які динамічно вдосконалюються;
- використовуйте алгоритми для статичного заповнення всіх дрібних деталей вашого всесвіту, або попередньо створіть корпус вмісту, який динамічно зшивається разом.

2 ОГЛЯД МЕТОДІВ ТА ТЕХНОЛОГІЙ, ЩО ВИКОРИСТОВУЮТЬСЯ

2.1 Огляд основних понять адаптивної генерації ландшафту

Висота ландшафту зазвичай використовується для представлення ландшафту в двовимірному контексті. Вибіркові значення генеруються в різних точках процедурної поверхні за допомогою генераторів шуму, а отримані дані потім відображаються на двовимірній або тривимірній сітці (карта висот і воксельна сітка), яка представляє область процедурної поверхні.

Карта висот – це двовимірна сітка значень, де кожне значення в сітці відбирається в певній точці 2D-простору з використанням його координат XY. Часто в наш час генерація рельєфу грунтується на генераторі фрактального шуму, наприклад, шумі Перліна в поєднанні з fbm, вперше запропонованому Бенуа Мандельбротом. Дробовий броунівський рух (Ebert, 1994) існує як рішення для більш точного представлення реалістичного, фрактального ландшафту. Однак це коштує процесору більше циклів, оскільки дані для однієї точки потрібно буде згенерувати N разів.

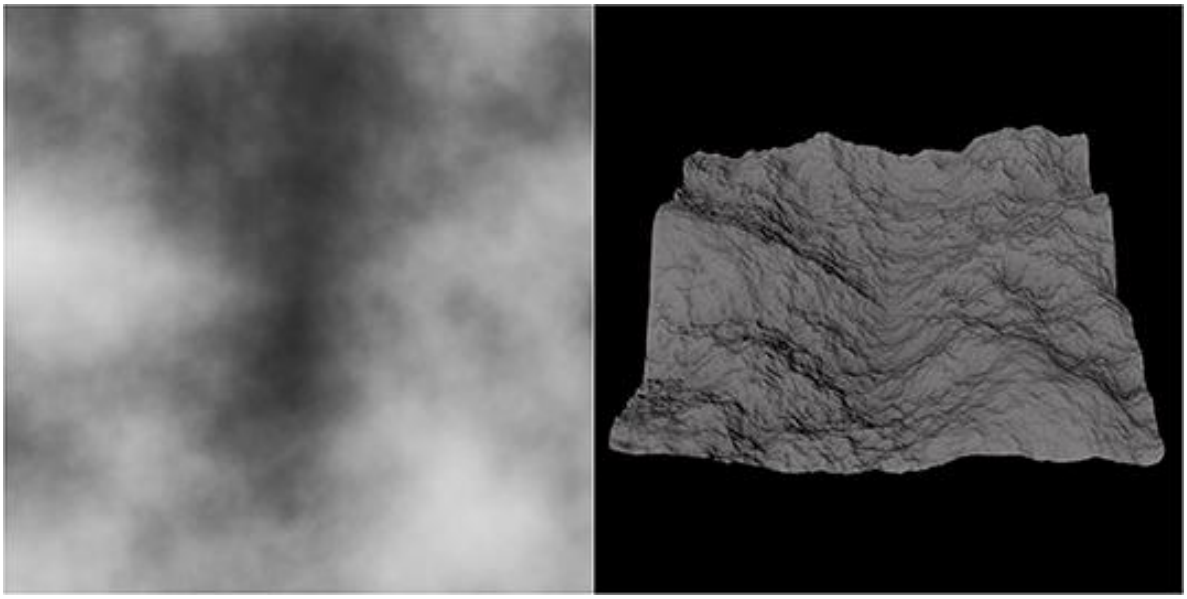

Рисунок 2.1 – Декілька накладених слоїв Perlin Noise (зліва) та віртуальна місцевість, сформована за допомогою карти висот (справа)

Карти висот зазвичай візуалізуються, спочатку заповнюючи карту вибірковими значеннями, а потім перетворюючи карту на (зазвичай квадратне) растрове зображення. У контексті процедурної генерації ландшафту висот часто служать основою для різних моделей, таких як рослинність, кількість опадів, температура тощо. Однак вони здебільшого загальновідомі тим, що використовуються як початковий етап створення рельєфу шляхом формування загальної висоти поверхні. Крім того, шарування кількох карт висот і їх перетворення в єдиний результат створить рельєф із набагато складнішими функціями, ніж у випадку використання лише карти висот.

У деяких випадках значення карти висот можуть розглядатися як порогове значення. Зазвичай із висотою вибіркове значення представлятиме межу, на якій поверхня переходить із заповненої в незаповнену або порожню. Карти висот корисні для представлення простих тривимірних структур великої кількості однорідно заповненого простору.

Варто пам’ятати, що карти висот є лише приблизними. Значення на карті відбираються через регулярні проміжки часу, і все, що знаходиться між ними, є незареєстрованою інформацією.

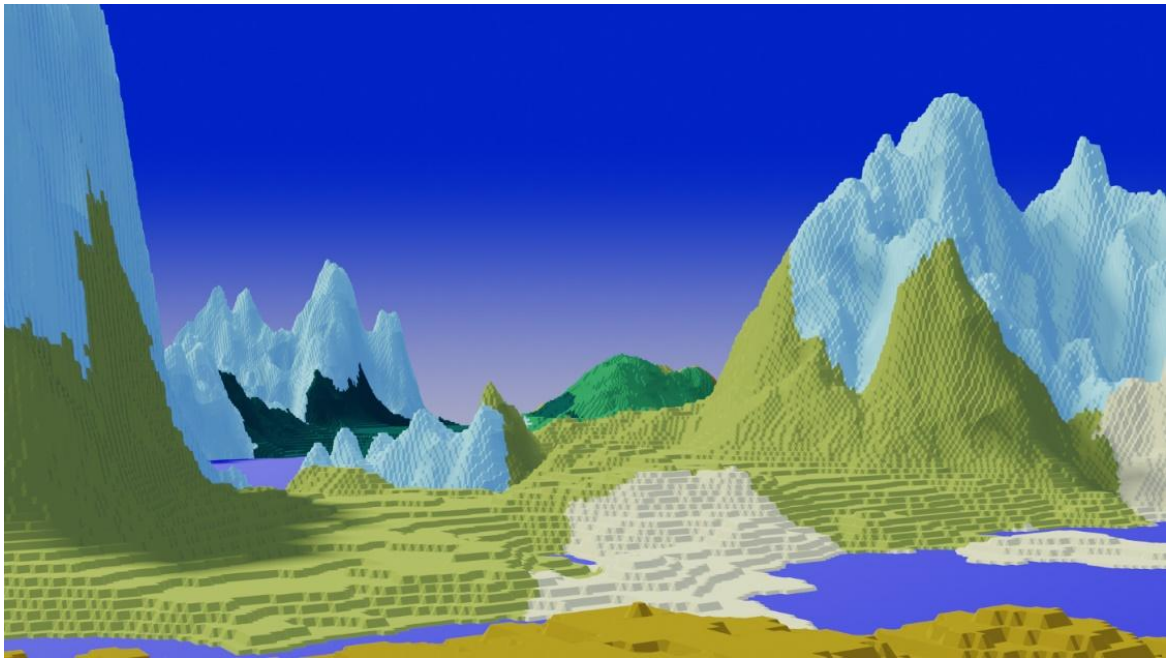

Рисунок 2.2 – Приклад процедурного створення ландшафту з моделюванням рельєфу

Хоча карти висот дуже корисні в процедурній генерації ландшафту, їх одних недостатньо для представлення місцевості зі складнішими об’єктами, такими як печери чи навіть плавучі острови. Це обмеження можна подолати за допомогою підходу до генерації на основі вокселів, на відміну від підходу 2D карти висот. Гра, відома як "Minecraft", стала дуже популярною, оскільки повністю базувалася на процедурних блоках (або вокселях).

Воксель – це точка на регулярній сітці в тривимірному просторі, яка представляє значення. Це значення можна порівняти зі зразковими значеннями, що зберігаються на карті висот, за винятком того, що вони є 2D. Основна корисність вокселів полягає в тому, що вони чудово відображають об’єми або простори, які не мають однорідного заповнення. Таким чином, багато з них об’єднуються, щоб утворити сітку вокселів, яка наближає ландшафт у 3D.

Вокселі не мають чіткої 3D-координати, пов’язаної з ними, натомість вона виводиться з відносного положення вокселя всередині об’ємного контейнера, який визначається певними розмірами.

Рисунок 2.3 – Процедурно згенерований "плаваючий острів" у грі "Minecraft"

Згенерований контент потрібно якось представити. У 2D карти висот можна застосовувати до растрових зображень для візуалізації даних. У 3D натомість

потрібно буде побудувати сітку та передати її на GPU для візуалізації. Стандартним підходом до цього є використання алгоритму вилучення ізо-поверхні. Реалізувавши такий алгоритм, можна створити кінцеву сітку, яку може відобразити GPU.

Існує кілька алгоритмів вилучення ізо-поверхні для 3D-даних, таких як Dual Contouring of Hermite Data, Marching Cubes, Ray Marching, Surface Nets та інші. Marching Cubes – це найвідоміший алгоритм, який містить багато інформації.

«Marching Cubes та інші алгоритми використовують воксельне представлення об’єму, розглядаючи кожну точку даних як вершину якогось геометричного примітиву, такого як куб або тетраедр. Ці примітиви, або комірки, поділяють об’єм і забезпечують корисну абстракцію для обчислення ізо-поверхень». – Філіп М. Саттон.

2.2 Адаптивна генерація в Marching Cubes

Marching Cubes – це алгоритм, який працює з невеликою сіткою сусідніх вокселів, яка представляє 8 кутів абстрактної кубічної комірки. Алгоритм використовує значення щільності, що зберігаються в кожному куті, щоб визначити за допомогою порогового значення, які з кутів знаходяться всередині поверхні, а які зовні. Якщо будь-де в клітинці відбувається перехід між суцільним і порожнім, алгоритм визначає, уздовж яких країв слід розмістити вершини. Це залежить від конфігурації комірки, а позиція залежить від реалізації алгоритму. Розміщення вершин визначається шляхом інтерполяції значень кутів.

Існує два основних компоненти цього алгоритму. По-перше, це вирішити, як визначити секцію або ділянки поверхні, які подрібнюють окремий куб. Якщо ми класифікуємо кожен кут як нижче або вище рівнозначності, існує 256 можливих конфігурацій класифікації кутів. Два з них тривіальні; де всі точки знаходяться всередині або поза кубом, не робить внеску в ізоповерхню. Для всіх інших конфігурацій нам потрібно визначити, де вздовж кожного краю куба перетинається ізоповерхня, і використовувати ці точки перетину ребер, щоб

створити одну або більше трикутних ділянок для ізоповерхні.

Якщо врахувати симетрії, насправді існує лише 14 унікальних конфігурацій серед решти 254 можливостей. Якщо є лише один кут, менший за рівнозначне, це утворює єдиний трикутник, який перетинає ребра, які стикаються в цьому куті, при цьому нормаль звернена від кута.

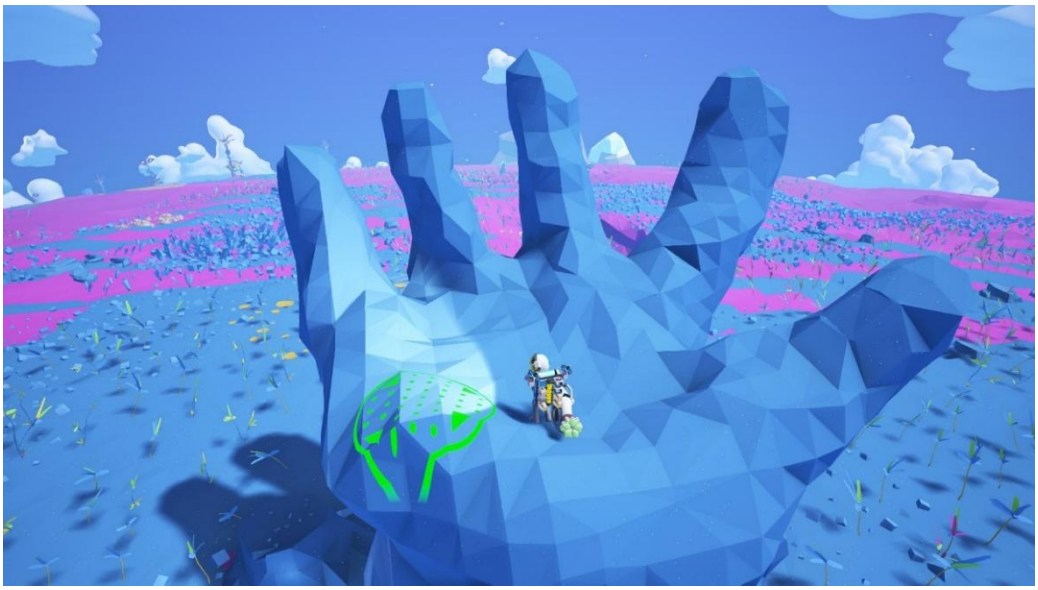

Рисунок 2.4 – Інді-гра "Astroneer" розроблена System Era Softworks використовує Marching Cubes для адаптивної генерації світу та відображення створених структур

2.3 Адаптивна генерація фігур та ландшафту за допомогою Ray Marching

Схожим до Marching Cubes алгоритмом являється Ray Marching, але працює він зовсім інше. Він виконується шляхом встановлення положення камери, а потім пропускання променів від положення камери через кожен піксель на екрані, який має бути кольоровим. Фактично Ray Marching реалізується як поступове фіксоване значення вздовж лінії в напрямку променя, доки він не перетне об’єкт, який реймаршується, наприклад. сфера або місцевість. Колір вихідного пікселя визначається з того, що промінь перетинає, наприклад колір кулі, неба або місцевості.

Замість постійного або лінійного збільшення кроку можна використовувати

оцінку відстані. Функція оцінки відстані – це функція, яка для будь-якої точки може дати нам нижню межу відстані, яку має пройти промінь, щоб перетнути те, що промінь трасується, наприклад сфера або рельєф заснований на шумі.

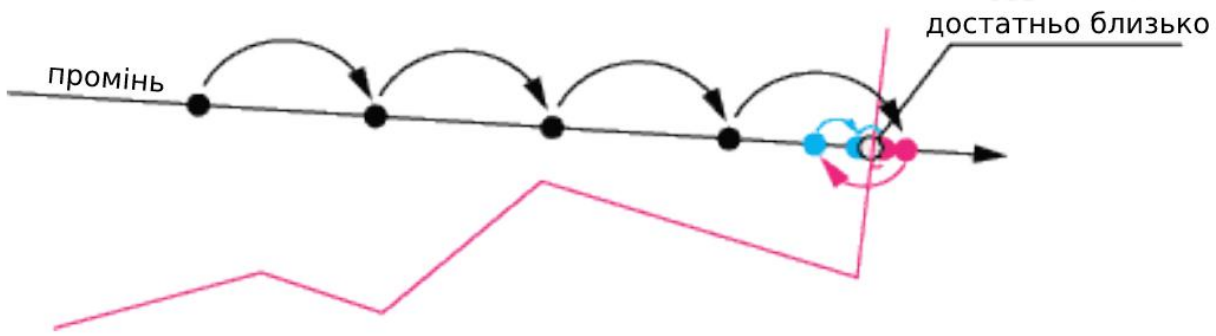

Рисунок 2.5 – Візуалізація алгоритму Ray Marching

Рельєф має максимальну висоту, яка може бути використана для оптимізації маршу променів. Однією з оптимізацій, яка використовується, є видалення всіх променів, коли камера перевищує максимальну висоту. Цей сценарій відбувається, якщо дивитися на горизонт, а верхня частина екрана показує небо.

Інша оптимізація полягає в тому, що коли камера знаходиться вище максимальної висоти, промінь може перейти з позиції камери прямо на максимально дозволену висоту місцевості (оскільки не може бути місцевості вище цієї висоти) і почати марш звідти. Ці оптимізації видаляють приблизно половину променів у звичайних сценаріях.

Ландшафт генерується за допомогою середньозваженого октав симплексного шуму. Симплексний шум спочатку був розроблений Кеном Перліном як вдосконалення його відомого Perlin Noise. Більша частина генерації шуму виконується у фрагментному шейдері, лише матриця перестановок створюється в програмі та подається у фрагментний шейдер як текстура, щоб полегшити обчислювальне навантаження. Причина цього в тому, що шум розраховується для кожного кроку променя.

Тіні обчислюються шляхом відкидання світського променя від точки перетину основного променя до джерела світла. Це наївна реалізація, яка коштує

дорого, але добре працює в цьому обмеженому проекті. Якщо він перетинає щось на шляху до джерела світла, початок координат отримує більш темний відтінок того самого кольору. Затінення рельєфу є звичайним розсіяним затіненням, де нормаль обчислюється шляхом добутку числового тангенса та бінормалі.

Джерело світла візуалізується світловою кулею, розташованою на деякій відстані над землею. Припустимо, що вектор між положенням джерела світла та положенням камери називається v. Для кожного пікселя в сцену надсилається промінь u з такою ж довжиною, що й v. Якщо відстань між кінцевою точкою u та джерелом світла мала, ми малюємо джерело світла, розфарбовуючи ці пікселі.

Одним з найкращим прикладом використання алгоритму Ray Marching є гра "Marble Marcher" від розробника CodeParade, вона використовує математичні формули для генерації тривимірних фракталів в реальному часі.

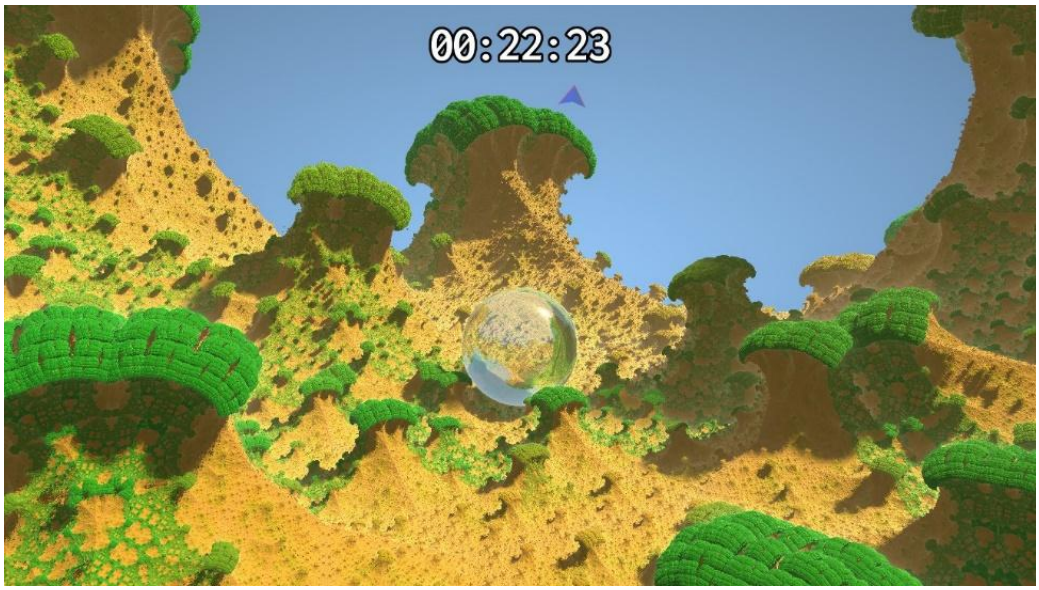

Рисунок 2.6 – Гра "Marble Marcher" використовує Ray Marching для створення математичних 3D фракталів з великою деталізацією

2.4 Генерація ландшафту за допомогою Surface Nets алгоритму

Surface Nets – це алгоритм для вилучення ізоповерхневої сітки з поля відстані зі знаком, взятого на регулярній сітці. Це майже те саме, що й Dual Contouring, але замість використання даних Ерміта (похідних) для оцінки точок

поверхні Surface Nets виконає простішу форму інтерполяції (усереднення) між точками, де ізоповерхня перетинає краї воксельного куба.

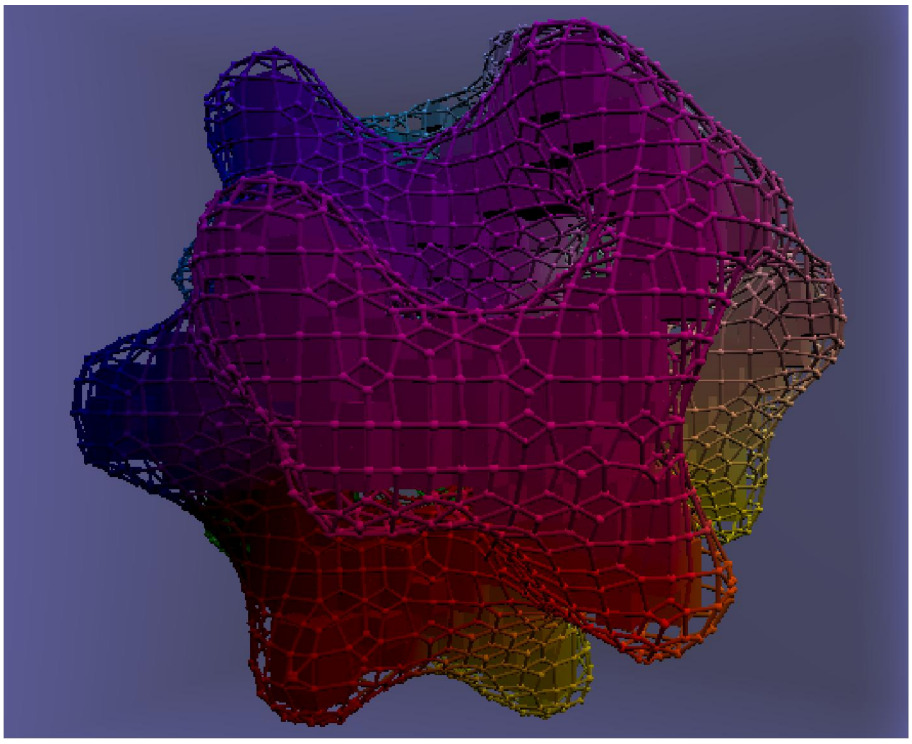

Рисунок 2.7 – Генерація "сітки" навколо воксельних блоків

Цей алгоритм використовують не тільки в розробці ігор, а також в МРТ. Наприклад, при створенні тривимірного сканування кістки людини створюється об’ємна модель, що складається з вокселів, за допомогою Surface Nets алгоритму.

Рисунок 2.8 – Сканування стегнової кістки за допомогою МРТ (зліва), згладжена модель за допомогою технології Gaussian Blur, присутні значні терасовані артефакти (центр), стегнова кістка візуалізована та затінена за допомогою карт відстаней, створених з Surface Nets, набагато менше артефактів (справа)

3 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір інструменту розробки

Visual Studio – це інтегроване середовище розробки (IDE), розроблене Microsoft для розробки GUI (графічного інтерфейсу користувача), консолі, веб-програм, веб-програм, мобільних програм, хмарних і веб-сервісів тощо. За допомогою цієї IDE ви можете створювати керований код, а також рідний код. Він використовує різні платформи програмного забезпечення Microsoft для розробки програмного забезпечення, як-от магазин Windows, Microsoft Silverlight і Windows API тощо. Це не IDE для певної мови, оскільки ви можете використовувати його для написання коду на C#, C++, VB (Visual Basic), Python, JavaScript і багато інших мов. Він забезпечує підтримку 36 різних мов програмування. Він доступний як для Windows, так і для macOS.

Оглядач рішень дозволяє переглядати файли коду, пересуватися по ньому і управляти ними. Оглядач рішень дозволяє впорядкувати код шляхом об'єднання файлів в рішення і проекти. У вікні редактора відображається вміст файлу. Тут ви можете редагувати код або розробляти призначений для користувача інтерфейс, наприклад вікно з кнопками або текстові поля. Team Explorer дозволяє відстежувати робочі елементи і використовувати код спільно з іншими користувачами за допомогою технологій управління версіями, таких як Git і система управління версіями Team Foundation (TFVC).

Існує 3 версії Microsoft Visual Studio:

1. Community: це безкоштовна версія, анонсована в 2014 році. Усі інші версії є платними. Він містить функції, подібні до версії Professional. Використовуючи цю версію, будь-який окремий розробник може розробляти власні безкоштовні або платні програми, такі як програми .Net, веб-програми та багато іншого. У корпоративній організації цей випуск має деякі обмеження.

2. Professional: це комерційна версія Visual Studio. Він доступний у Visual Studio 2010 і пізніших версіях. Він забезпечує підтримку редагування XML і

XSLT і включає такий інструмент, як Server Explorer, і інтеграцію з Microsoft SQL Server.

3. Enterprise: це інтегроване комплексне рішення для команд будь-якого розміру з вимогливими вимогами до якості та масштабу. Корпорація Майкрософт надає 90-денну безкоштовну пробну версію цього видання, і після пробного періоду користувач повинен заплатити, щоб продовжити його використання. Основна перевага цього видання полягає в тому, що воно має високу масштабованість і забезпечує високоякісне програмне забезпечення.

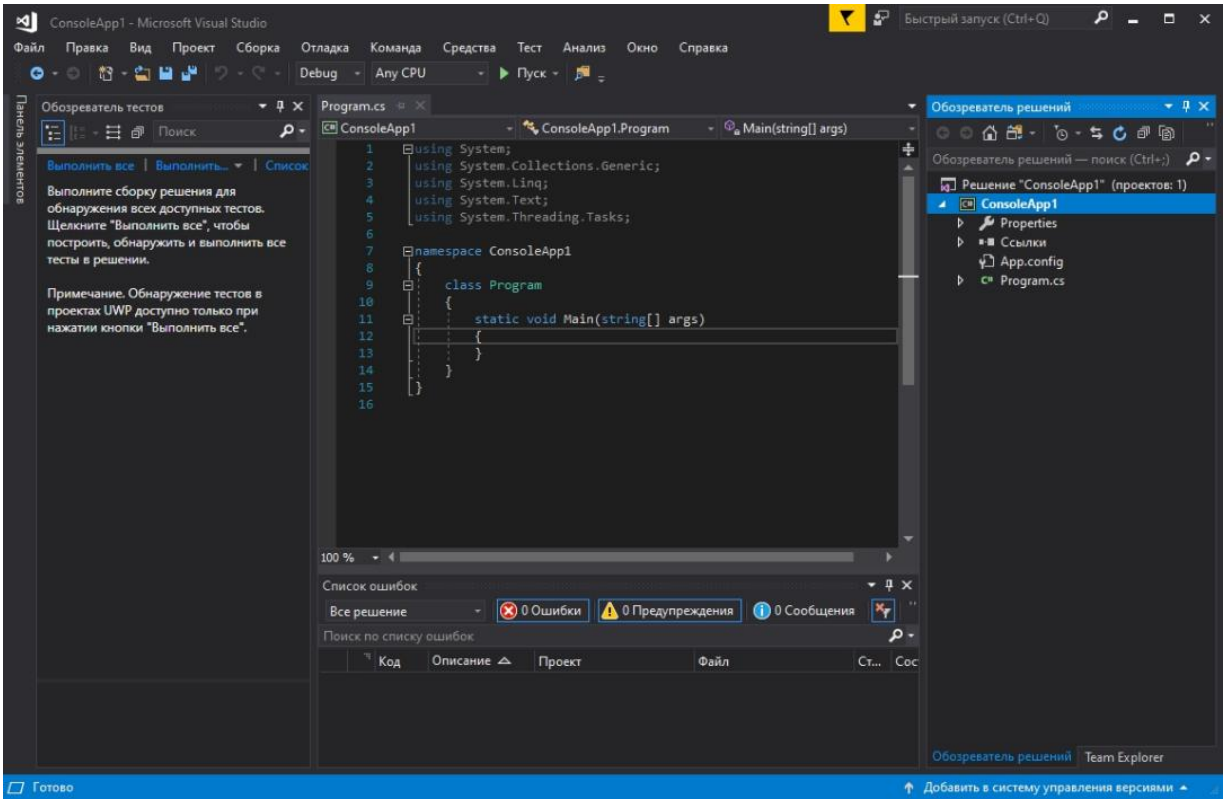

Рисунок 4.1 – Інтерфейс IDE Visual Studio

Visual Studio не має власної підтримки коду GLSL, тому община програмістів розробила свою власну інтеграцію. Вона добавляє проект VSIX, який забезпечує інтеграцію мови GLSL. Включає підсвічування синтаксису, доповнення коду (OpenGL 4.6 + ідентифікатори у файлі шейдера), позначення помилок за допомогою завивок і в списку помилок (ранньої версії). Для позначення помилок використовується окремий потік OpenGL, який компілює

шейдер на основній графічній карті. Також можна вибрати зовнішній виконуваний файл компілятора для використання з цією метою в меню параметрів.

Unity є сучасним багатоплатформовим двигуном для створення ігор і додатків, розроблений Unity Technologies. За допомогою даного двигуна можна розробляти не тільки додатки для комп'ютерів, але і для мобільних пристроїв (на базі Android та iOS), ігрових приставок і інших девайсів. Додатки, створені за допомогою Unity, підтримують DirectX і OpenGL.

Будь-який ігровий двигун надає безліч функціональних можливостей, які задіюються в різних іграх. Реалізована на конкретному двигуні гра отримує всі функціональні можливості, до яких додаються її власні ігрові ресурси і код ігрового сценарію. Додаток Unity пропонує моделювання фізичних середовищ, карти нормалей, перегородження навколишнього світу в екранному просторі (Screen Space Ambient Occlusion, SSAO), динамічні тіні тощо. Подібні набори функціональних можливостей є в багатьох ігрових двигунах, але Unity володіє двома основними перевагами над іншими передовими інструментами розробки ігор. Це вкрай продуктивний візуальний робочий процес і сильна межплатформенная підтримка.

Рисунок 4.2 – Інтерфейс двигуна Unity

Unity Personal безкоштовний і повнофункціональний, що дозволяє розробникам і студентам бути креативними та розвивати ігрову індустрію. Ця версія Unity обмежена, якщо ваша гра заробляє 100 000 доларів США або більше на рік і не містить звітів про продуктивність або оформлення інтерфейсу професійного редактора. Щоб використовувати ці функції, потрібне оновлення до пакета Unity Plus, вартість якого становить 35 доларів США на місяць за 12-місячним планом зобов'язань (49 доларів США на місяць із безстроковим контрактом). Хоча ця опція все ще зберігає максимальний дохід у 100 000 доларів США на рік, ви маєте додатковий бонус у вигляді гнучкого керування робочими місцями, пакетів проектів для зберігання активів і місячного доступу до сертифікаційного курсу Unity.

Хоча фреймворк Unity побудований на мові C++, користувачі взаємодіють із двигуном через C# або Javascript. Хоча їх досить легко вивчити та добре підтримують користувачі на форумах Unity Answers, використання Unity не наражатиме вас на код C++, що є важливим фактором, якщо ви хочете спеціалізуватися як програміст ігор у галузі загалом.

Для того щоб користуватися OpenGL шейдерами (GLSL), треба

використовувати аргумент командного рядка -force-opengl. Після створення нового проекту в Windows, Unity може перезапуститися без підтримки OpenGL; таким чином, користувачі Windows повинні завжди виходити з Unity і перезапускати його (з аргументом командного рядка -force-opengl) після створення нового проекту. Потім можна відкрити новий проект за допомогою меню Файл > Відкрити проект...

Створення шейдера GLSL не є складним: у вікні проекту натисніть "Створити" та виберіть "Шейдер". Має з’явитися редактор із стандартним шейдером у Cg. Треба видалити стандартний шейдер та використати цей код:

```
Shader "GLSL basic shader" { // назва шейдеру
   SubShader { // Unity вибирає сабшейдер який краще підходе для GPU
      Pass { // деяким шейдерам потрібно декілька проходів рендера
         GLSLPROGRAM // початок коду шейдера в Unity GLSL

         #ifdef VERTEX // вершинний шейдер

         void main()
         {
            gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
               // цей рядок перетворює попередньо визначений атрибут
               // gl_Vertex типу vec4 за допомогою попередньо визначеного
               // уніфікованого gl_ModelViewProjectionMatrix типу mat4
               // і зберігає результат у попередньо визначеній
               // вихідній змінній gl_Position типу vec4.
         }

         #endif // закінчення вершинного шейдера
```

```
                #ifdef FRAGMENT // фрагментний шейдер

                void main()
                {
                   gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
                      //  цей  фрагментний  шейдер  просто  встановлює
вихідний
                      // колір на непрозорий червоний (червоний = 1.0,
                      // зелений = 0.0, синій = 0,0, прозорість = 1.0)
}

                #endif // закінчення фрагментного шейдера
                ENDGLSL // кінець коду GLSL
            }
        }
    }
```

Shadertoy.com – це онлайн-спільнота та платформа для професіоналів комп'ютерної графіки, науковців та ентузіастів, які діляться, навчаються та експериментують із техніками візуалізації та процедурним мистецтвом за допомогою коду GLSL. WebGL дозволяє Shadertoy отримати доступ до обчислювальної потужності графічного процесора для створення процедурного зображення, анімації, моделей, освітлення, логіки на основі стану та звуку.

Шейдери Image реалізують функцію mainImage() для створення процедурних зображень шляхом обчислення кольору для кожного пікселя. Очікується, що ця функція буде викликатися один раз для кожного пікселя, і головна програма несе відповідальність за надання правильних вхідних даних для неї, отримання від неї вихідного кольору та призначення його пікселю екрана. Прототип:

```
void mainImage(out vec4 fragColor, in vec2 fragCoord);
```

де fragCoord містить координати пікселів, для яких шейдер повинен

обчислити колір. Координати вказуються в одиницях пікселів у діапазоні від 0.5 до resolution-0.5 над поверхнею візуалізації, де роздільна здатність передається шейдеру через форму iResolution

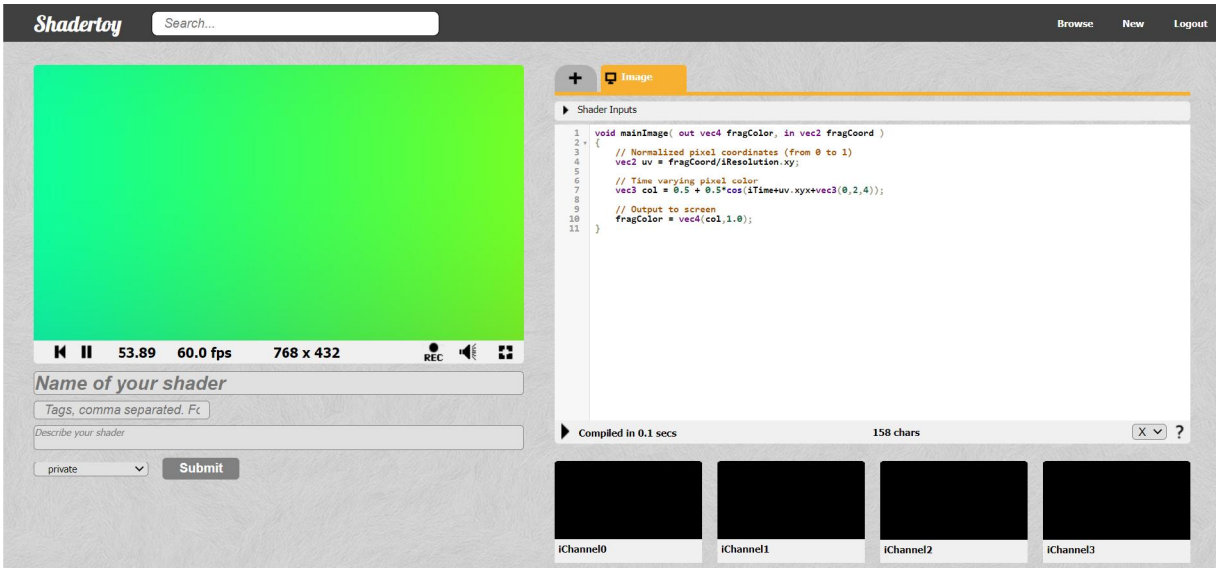

Рисунок 4.3 – Інтерфейс сайту Shadertoy.com

Отриманий колір збирається у fragColor як чотирикомпонентний вектор, останній з яких ігнорується клієнтом. Результат збирається як "вихідна" змінна для прогнозування майбутнього додавання кількох цілей візуалізації.

Шейдери Shadertoy можуть безперешкодно створювати зображення для середовищ віртуальної реальності (VR). Для цього шейдери повинні реалізувати точку входу mainVR(), яка є необов’язковою та має такий прототип:

```
void mainVR( out vec4 fragColor, in vec2 fragCoord, in vec3 fragRayOri, in vec3 fragRayDir )
```

де fragCoord і fragColor працюють так само, як у звичайних шейдерах двовимірних зображень: вони містять координати пікселів у просторі поверхні та колір вихідного пікселя.

Змінні fragRayOri та fragRayDir містять джерело та напрямок променя, який проходить через цей піксель у віртуальному світі, наданий у просторі трекера.

Очікується, що шейдер перетворить ці значення в простір камери, якщо рухому камеру потрібно переміщати у віртуальному світі. Походження променя надається як змінне та неоднакове для систем VR, де камера не моделюється як камера-обскура.

Шейдери, які реалізують точку входу mainVR, автоматично позначатимуться як готові до VR, і їх можна буде шукати через систему фільтрів за кваліфікатором VR.

Для цього створення візуалізацій алгоритмів використовувався онлайн сервіс Shadertoy. Він дуже легкий у використанні, немає потреби налаштувань, закачувань програм та компонентів, також компіляція коду відбувається в хмарі.

Також Shadertoy має шейдери звуку. Звук генерується шейдерами GLSL через точку входу mainSound(), яка має такий прототип:

```
vec2 mainSound( float time )
```

де змінна часу містить час у секундах зразка звуку, для якого необхідно обчислити амплітуду хвилі. Цей час відбиратиметься з частотою дискретизації, визначеною уніформою iSampleRate, яка зазвичай становить 44 100 або 48 000 залежно від програми хосту.

Потрібна амплітуда хвилі виводиться як пара значень для стереозвуку (лівий і правий канали), незважаючи на значення, що повертається функцією mainSound().

Шейдери, які реалізують точку входу mainSound, автоматично позначатимуться як звукові шейдери, і їх можна буде шукати через систему фільтрів у кваліфікаторі Sound Output.

3.2 Розробка візуалізацій алгоритмів

Візуалізація Ray Marching

Коли ми використовуємо Ray Marching, ми більше не маємо справу з

полігонами. Отже, багато речей, до яких ми звикли в типовому наборі інструментів для 3D - геометрія, світло, камери – відсутні. Але якщо у нас немає точок, ліній, трикутників і сіток, як нам щось відобразити? Хитрість полягає в тому, щоб використовувати знакові функції відстані (SDF). Це математичні функції, які беруть точку в просторі та повідомляють, на якій відстані ця точка знаходиться від поверхні.

Скажімо, ми знаходимося в точці *p* у тривимірному просторі. У GLSL ми можемо представити p як:

```
vec3 p = vec3(x, y, z);
```

для деяких координат x, y і z. Тепер припустимо, що у нас є сфера з центром у початку координат і одиничним радіусом. Ми хочемо відповісти на запитання: «Яка відстань від p до найближчої точки на сфері?» Виявляється, що ця відстань відповідає довжині вектора, що вказує від p до центру кулі, мінус радіус кулі.

Щоб відобразити сцену, треба використовувати техніку під назвою Ray Marching. На високому рівні ми будемо випускати купу уявних променів з віртуальної камери, яка дивиться на наш світ. Для кожного з цих променів ми збираємося «маршировати» вздовж напрямку променя, і на кожному кроці оцінюємо нашу SDF. Це скаже нам: «Як далеко я від найближчої точки на поверхні нашої сфери» з того місця, де я зараз стою?

Наша мета – робити кроки вздовж цього променя, доки ми не підійдемо настільки близько до об’єкта, що зможемо безпечно зупинитися. Але як нам рухатися вздовж променя? Одним із підходів було б робити невеликі, рівномірні кроки. У GLSL це може виглядати так:

```
const float step_size = 0.1;

for (int i = 0; i < NUMBER_OF_STEPS; ++i)
{
```

```
    vec3 current_position = ro + (i * step_size) * rd;
}
```

Промінь має дві складові: початок і напрям. Уявімо, що кожен промінь починається від камери та проходить через уявну «площину зображення», яка знаходиться десь перед нашою камерою. Весь код виконується всередині одного фрагментного шейдера, тому є певний стрибок, коли ми якимось чином повинні описати 3D-світ із 2D-площини, над якою виконується наш фрагментний шейдер. Фрагментний шейдер виконується один раз для кожного пікселя, який утворює наше остаточне відтворене зображення. У кожному місці пікселя ми можемо отримати UV-координату в діапазоні [0.0, 1.0].

Далі переналаштуємо UV-координати з діапазону [0.0, 1.0] на [-1.0, 1.0]. Робиться це здебільшого для зручності, оскільки це розміщує піксель у центрі нашого зображення на (0.0, 0.0). Тепер, у нашому уявному 3D-просторі, припустимо, що камера знаходиться на відстані 5 одиниць від початку координат, у негативному напрямку Z. Для простоти ми не будемо мати справу з такими речами, як кут зору (FOV), обертання камери тощо.

Ми вже знаємо, що кожен промінь походить від камери. Його напрямний вектор можна розглядати як трасування лінії від положення камери через точку на площині зображення. Наступний фрагмент коду GLSL демонструє цей процес у дії:

```
vec2 uv = fragCoord/iResolution.xy;
vec3 camera_position = vec3(0.0, 0.0, -5.0);
vec3 ro = camera_position;
vec3 rd = vec3(uv, 1.0);
```

Z-координата rd діє на кшталт FOV камери, наближаючи або віддаляючи площину зображення від камери. Залишимо це значення 1.0.

Отже, оскільки кожен фрагмент має унікальні UV-координати, кожне

виконання нашого фрагментного шейдера генеруватиме унікальний промінь. Через те, як шейдери виконуються на вашій графічній карті, усі ці обчислення відбуватимуться паралельно.

Щоб розрахувати деякі базові затінення, щоб ми могли підтвердити, що це справді тривимірна поверхня, нам потрібні нормальні вектори для розрахунку. Для сфери нормаль у будь-якій точці поверхні можна обчислити, просто нормалізувавши вектор c – p, де, як і раніше, c є центром сфери, а p є точкою на поверхні сфери. Однак цей метод має обмеження (він не поширюється на інші форми), коли ми деформуємо наш SDF, нам потрібен більш динамічний спосіб обчислення нормалей.

Ідея полягає в тому, що ми можемо трохи «підштовхнути» нашу точку p у позитивному та негативному напрямку вздовж кожної з осей X/Y/Z, перерахувати наш SDF і побачити, як змінюються значення. Треба обчислити градієнт поля відстані на p. У 2D є похідна, яка визначає швидкість зміни функції відносно її вхідних даних, це зазвичай візуалізовано як нахил лінії, яка лежить по дотичній до функції в певній точці. Градієнт є лише розширенням цього для функцій кількох вимірів (SDF має 3 виміри, X/Y/Z). Нормалі зазвичай мають бути одиничними векторами, тому ми також нормалізуємо їх. Цей метод дозволяє обчислювати нормалі для довільно складних об’єктів, за умови, що у нас є відповідний SDF для представлення його поверхні.

Коли у нас є базові налаштування маршу променів, ми можемо «підштовхнути» або порушити наші функції відстані, щоб створити більш цікаві форми. Наприклад, ми можемо додати синусоїдальні спотворення до SDF нашої сфери, змінивши функцію map_the_world таким чином:

```
float map_the_world(in vec3 p)
{
    float displacement = sin(5.0 * p.x) * sin(5.0 * p.y) * sin(5.0 * p.z) * 0.25;
    float sphere_0 = distance_from_sphere(p, vec3(0.0), 1.0);
```

```
    return sphere_0 + displacement;
}
```

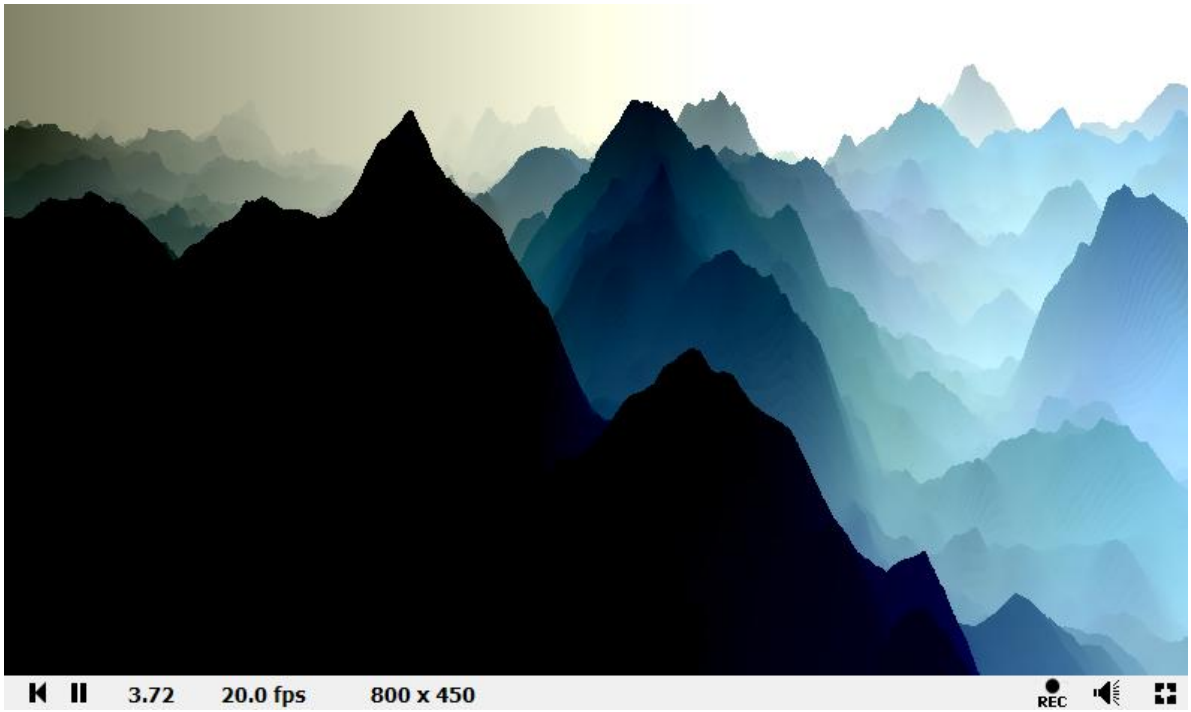

Рисунок 4.4 – Візуалізація шуму за допомогою алгоритму Ray Marching

Візуалізація Surface Nets

Перший крок – зрозуміти, як виглядає сітка кубів відносно вашого введення. Наші вхідні дані є тривимірною сіткою вокселів. Для Surface Nets вам слід уявити, що кожен воксель насправді не є кубом, а натомість однією точкою в 3D-просторі (тому самому 3D-просторі, де ми визначаємо функцію відстані зі знаком). Ця точка є кутом, спільний для 8 кубів у сітці.

Кожен воксель повинен зберігати значення функції відстані зі знаком у кутку сітки. Тепер ми знаємо одну з частин даних для зберігання в нашому типі вокселів.

Для кожного куба в сітці нам потрібно визначити, де ми знайдемо точку ізоповерхні, якщо вона взагалі є. Ми можемо подивитися на 8 кутів куба, і якщо в цих кутах є суміш позитивних і негативних відстаней, ми можемо сказати, що десь у кубі має бути точка поверхні.

Подібним чином ми можемо поглянути на 12 ребер куба, і ми знаємо, що

якщо ребро з’єднує дві точки (вокселі) з різними знаками, то згідно з теоремою проміжного значення, повинна бути якась точка на ребру, яка має значення 0, тобто це точка поверхні, і це ребро перетинає поверхню там. Ми беремо всі такі ребра в кубі та усереднюємо точки перетину їхніх поверхонь, щоб отримати приблизну точку поверхні, яка лежить усередині куба. Якщо таких ребер немає, то ми пропускаємо цей куб, оскільки він не містить точки поверхні.

Отже, як ми обчислюємо точку перетину для одного ребра? Ось тут і вступають у дію фактичні значення відстані. Технічно ми знаємо лише відстань від кута до найближчої точки на ізоповерхні (це визначення функції відстані зі знаком), але це не обов’язково точка перетину поверхні та краю (точка, позначена Ps на діаграмі нижче). .

Ми не можемо мати достатньо інформації в нашій кінцевій сітці, щоб обчислити точне перетин. Але що, якщо ми зробимо припущення, що поверхня приблизно рівна в околицях перетину? Це вірно в граничному випадку для різноманіття, тому ви отримуєте краще наближення зі збільшенням роздільної здатності сітки.

P_1 і P_2 – кути куба. Ребро між ними перетинає поверхню в точці Ps. f – функція відстані зі знаком. Перпендикулярні пунктирні лінії вказують відстані від P_1 і P_2 до поверхні.

Є ребро, що з’єднує два кути з тривимірними координатами P_1 і P_2, з відповідними значеннями відстані d_1 і d_2 зі знаком. Поверхня перетинає ребро в точці з координатами Ps.

За подібністю трикутників відношення

d_1 / d_2

має дорівнювати відношенню

$|P_1 – Ps| / |Ps – P_2|$

це єдине співвідношення, яке потрібно для обчислення точки перетину поверхні та краю. Отже, використовуємо лінійну інтерполяцію, щоб знайти точку на краю, де відстань зі знаком має дорівнювати нулю.

Обчислення нормалей насправді досить просте. Вектори, ортогональні до

набору рівнів, завжди є градієнтами функції. Для функції з 2D доменом можна уявити топографічну карту. Набори рівнів, криві постійної висоти, завжди ортогональні вектору градієнта, тобто вектору, спрямованому в найкрутішому напрямку. Отже, для функції з тривимірним доменом, як наша функція відстані зі знаком, ізоповерхня є набором рівнів, а її нормальні вектори є градієнтами функції відстані зі знаком.

Таким чином, ми будемо приблизно оцінювати градієнт у точці поверхні, дивлячись на той самий набір із 8 кутів куба. Щоб отримати координату x градієнта, візьміть суму різниць між кутами вздовж країв у напрямку x. Так само для y і z.

Останнім кроком є написання буфера індексу, який визначає всі трикутники у вашій сітці. Будьте впевнені, тому що це, мабуть, найбільш заплутана частина.

Знову ж таки, ми будемо дивитися на ребра кубів, але трохи по-іншому. Ми повторимо всі точки поверхні, які ми знайшли раніше. Опис псевдокоду виглядає так:

```
mesh_quads = []
for P1 in surface_points:
    for axis in [x_axis, y_axis, z_axis]:
        edge = get a cube edge in the direction of "axis"
        quad = get a quad, orthogonal to "axis", of surface
points
                where P1 is the maximal corner
        if edge is (-,+) or edge is (+,-):
            insert quad into mesh_quads
```

Воксельна мапа може бути набором чанків. Кожен чанк має окрему сітку. Отже, щоразу, коли воксель змінюється, сітка для цього чанка має змінюватися. Замість того, щоб намагатися з’ясувати, як застосувати дельти до сітки, досить просто регенерувати всю сітку щоразу, коли чанк змінюється.

Якщо дивитися лише на вокселі у чанкі під час створення сітки, ми отримуємо розриви у сітці. Це відбувається тому, що Surface Nets обчислює лише точки поверхні всередині кубів, які надаються. Але, дивлячись на весь набір чанків, є куби з кутами з кількох шматків.

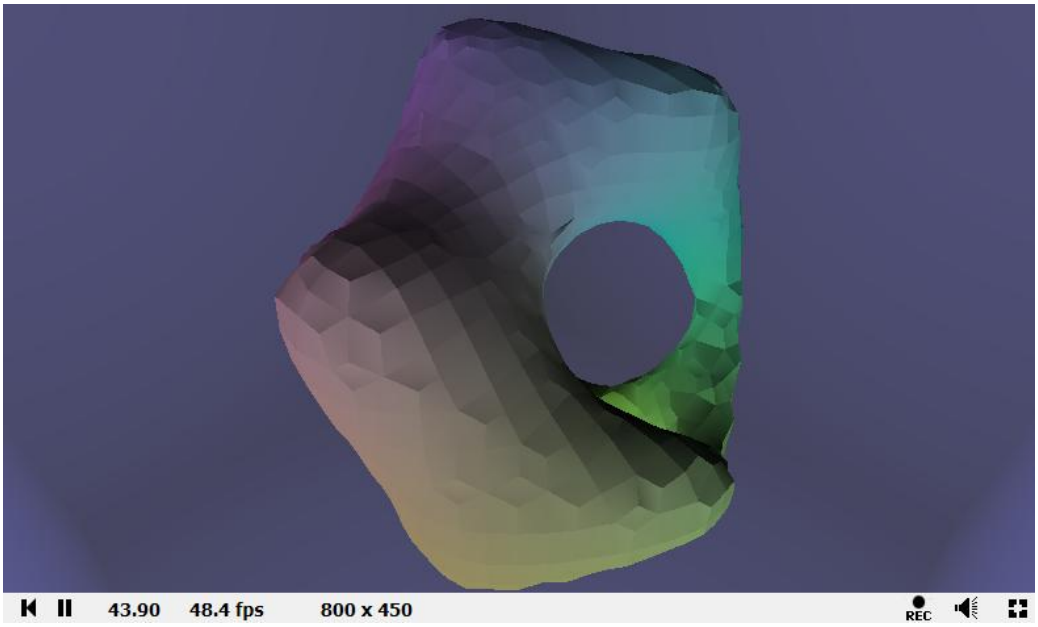

Рисунок 4.5 – Візуалізація тривимірного шуму за допомогою алгоритму Surface Nets

Рішення не дуже складне. Під час об’єднання чанку в сітку треба дивитися не лише на вокселі в цьому чанкі, а й на всі суміжні вокселі. Це гарантує, що ми не пропустимо жоден куб, а сітки обчислять ідентичні точки поверхні на межі, тому вони з’єднаються без розривів.

Але це рішення має певну вартість. Тепер, коли оновлюється воксель на межі чанка, це затроне сітки всіх суміжних чанків. Фактично, будь-які вокселі, суміжні з граничними вокселями, також впливатимуть на сітки сусідніх чанків.

Візуалізація Marching Cubes

Marching Cubes – це алгоритм візуалізації ізоповерхень в об’ємних даних. Основне поняття полягає в тому, що ми можемо визначити воксель (куб) за значеннями пікселів у восьми кутах куба. Якщо один або кілька пікселів куба мають значення, менші за вказане користувачем рівнозначення, а один або більше

мають значення, вищі за це значення, ми знаємо, що воксель повинен вносити певний компонент ізоповерхні. Визначивши, які грані куба перетинає ізоповерхня, ми можемо створити трикутні ділянки, які ділять куб між областями всередині ізоповерхні та областями поза нею. З'єднавши патчі з усіх кубів на межі ізоповерхні, ми отримаємо зображення поверхні.

Фундаментальна проблема полягає в тому, щоб сформувати фасетне наближення до ізоповерхні через скалярне поле, відібране на прямокутній тривимірній сітці. Враховуючи одну комірку сітки, визначену її вершинами, і скалярні значення в кожній вершині, необхідно створити планарні грані, які найкраще представляють ізоповерхню через цю комірку сітки. Ізоповерхня може не проходити через комірку сітки, вона може відсікати будь-яку з вершин або може проходити будь-яким із кількох більш складних способів. Кожна можливість буде характеризуватися кількістю вершин, які мають значення вище або нижче ізоповерхні. Якщо одна вершина знаходиться над ізоповерхнею, скажімо, а сусідня вершина знаходиться під ізоповерхнею, тоді ми знаємо, що ізоповерхня перетинає ребро між цими двома вершинами. Положення, в якому він перетинає край, буде лінійно інтерпольоване, співвідношення довжини між двома вершинами буде таким же, як і співвідношення значення ізоповерхні до значень у вершинах комірки сітки.

Перша частина алгоритму використовує таблицю (edgeTable), яка відображає вершини під ізоповерхнею на пересічні ребра. Формується 8-бітовий індекс, де кожен біт відповідає вершині.

```
cubeindex = 0;
if (grid.val[0] < isolevel) cubeindex |= 1;
if (grid.val[1] < isolevel) cubeindex |= 2;
if (grid.val[2] < isolevel) cubeindex |= 4;
if (grid.val[3] < isolevel) cubeindex |= 8;
if (grid.val[4] < isolevel) cubeindex |= 16;
if (grid.val[5] < isolevel) cubeindex |= 32;
```

```
if (grid.val[6] < isolevel) cubeindex |= 64;
if (grid.val[7] < isolevel) cubeindex |= 128;
```

Перегляд таблиці ребер повертає 12-бітове число, кожен біт відповідає ребру, 0, якщо ребро не порізане ізоповерхнею, 1, якщо ребро порізано ізоповерхнею. Якщо жодне з ребер не обрізано, таблиця повертає 0, це відбувається, коли cubeindex дорівнює 0 (усі вершини під ізоповерхнею) або 0xff (усі вершини над ізоповерхнею).

Використовуючи попередній приклад, де лише вершина 3 була під ізоповерхнею, кубіндекс дорівнюватиме 0000 1000 або 8. edgeTable[8] = 1000 0000 1100. Це означає, що ребра 2, 3 і 11 перетинаються ізоповерхнею.

Точки перетину тепер обчислюються шляхом лінійної інтерполяції. Якщо P_1 і P_2 є вершинами краю зрізу, а V_1 і V_2 є скалярними значеннями в кожній вершині, то точка перетину P визначається як

$$P = P_1 + (isovalue - V_1)\ (P_2 - P_1) / (V_2 - V_1)$$

Остання частина алгоритму передбачає формування правильних граней із позицій, у яких ізоповерхня перетинає краї комірки сітки. Знову використовується таблиця (triTable), яка цього разу використовує той самий індекс куба, але дозволяє шукати послідовність вершин, оскільки для представлення ізоповерхні в клітинці сітки необхідно багато трикутних граней. Потрібно щонайбільше 5 трикутних граней.

Одним дуже бажаним контролем під час полігонізації поля, де значення відомі або можуть бути інтерпольовані в будь-якому місці простору, є роздільна здатність сітки вибірки. Це дозволяє генерувати курс або точне наближення до ізоповерхні залежно від необхідної гладкості та/або потужності обробки, доступної для відображення поверхні.

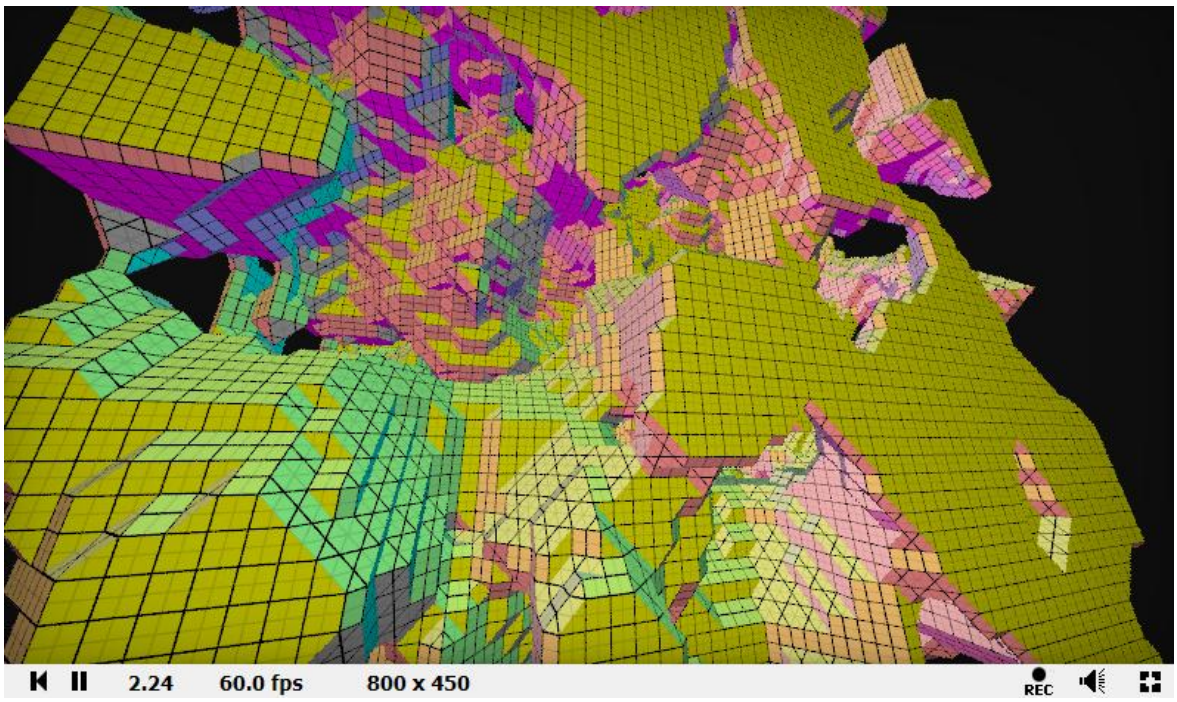

Рисунок 4.6 – Візуалізація тривимірного шуму за допомогою алгоритму Marching Cubes

3.3 Аналіз

При візуалізації шуму в тривимірному просторі за допомогою Ray Marching (рис. 4.4) ми отримуємо в розширенні 800x450 пікселів лише ~20 кадрів в секунду (fps). Код для візуалізації шуму дуже малий і деталей дуже багато, але при цьому ми маємо дуже вимогливу генерацію.

За допомогою алгоритму Surface Nets (рис. 4.5) в розширенні 800x450 пікселів ми отримуємо більш стабільні ~48 fps, код візуалізації набагато більше, але менш гнучкий, він більше підходить для згладжування заданого воксельного об’єму.

Найкращим результатом візуалізації 3D шуму є алгоритм Marching Cubes (рис. 4.6), ми маємо стабільні 60 кадрів в секунду в розширенні 800x450, алгоритм має більш гнучкий код, за допомогою якого ми можемо генерувати не лише поверхність ігрового світу, а також його об’єкти, наприклад печери. Також за допомогою цього алгоритму ми можемо позволити гравцю трансформувати ландшафт, який паралельно буде адаптивно згладжуватися схилами.

Для більш повного аналізу отриманих результатів дослідження, адаптивні алгоритми перевірені в різних розширеннях відображення (табл. 4.1). Як ми можемо побачити щвидкість алгоритму Ray Marching та Surface Nets стрімко падає при збільшенні розширення, найбільш стабільним виявився Marching Cubes. Хоча й кількість кадрів в секунду далеко не найкраща, ми використовуємо алгоритм на максимум, чого не повинно відбуватися в ігровому застосунку.

Таблиця 4.1 – Порівняння продуктивності відображення адаптивних алгоритмів в реальному часі

Розширення рендеру	Ray Marching	Marching Cubes	Surface Nets
	середня кількість кадрів в секунду		
800x450	18	60	45
960x540	12	60	30
1920x1080	3	23	8

ВИСНОВКИ

В ході виконання кваліфікаційної роботи були проаналізовані спеціалізовані літературні та інтернет-джерела з питань генерації шуму, адаптивної генерації елементів ігрового світу, вивченні схожі, вже існуючі розробки, вивчені методи роботи з відповідним ПО. Були проаналізовані популярні засоби розробки. В ході аналізу були обрані найбільш актуальні засоби розробки для початківців. Вибір пріоритетних засобів розробки проходив за двома критеріями: продуктивність та деталізація.

Найкращим алгоритмом генерації шуму являється Perlin Noise, який є стандартом між проєктами з процедурною генерацією ландшафту.

Було проаналізовано та порівняно декілька методів процедурної генерації та обраний найкращий. У розробленому демо-додатку додатку було реалізовано варіації методів процедурної генерації.

Більш деталізованим алгоритмом являється Ray Marching, за допомогою якого можна відтворювати тривимірні фрактали в реальному часі.

Найшвидшим алгоритмом виявився Marching Cubes, за допомогою якого можна згладжувати воксельний світ за допомогою схилів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 3D Procedural Generation with Adaptive Generation Strategy [Електронний ресурс] – Режим доступу: https://pats.cs.cf.ac.uk/@archive_file?p=985&n=final&f=1-c1544230.pdf&SIG=68a9f412e5121b0156ca4124fb74c51b7fbbb29f8badbf87a90b9c0450a2bb44, вільний.

2. Gameplay semantics for the adaptive generation of game worlds [Електронний ресурс] – Режим доступу: https://graphics.tudelft.nl/Publications-new/2014/Lop14/Lop14.pdf, вільний.

3. How to use BSP trees to generate game maps [Електронний ресурс] – Режим доступу: https://gamedevelopment.tutsplus.com/tutorials/how-to-use-bsp-trees-to-generate-game-maps-gamedev-12268, вільний.

4. Constrained Elastic SurfaceNets: Generating Smooth Models from Binary Segmented Data [Електронний ресурс] Режим доступу: https://www.merl.com/publications/docs/TR99-24.pdf, вільний.

5. Rendering noise-generated terrain with ray marching [Електронний ресурс] Режим доступу: https://fileadmin.cs.lth.se/cs/Education/EDAN35/projects/16NiklasJohan_Terrain.pdf, вільний.

6. Understanding Surface Nets [Електронний ресурс] Режим доступу: https://cerbion.net/blog/understanding-surface-nets/, вільний.

7. Ray Marching [Електронний ресурс] Режим доступу: https://michaelwalczyk.com/blog-ray-marching.html, вільний.

8. Polygonising a scalar field [Електронний ресурс] Режим доступу: http://paulbourke.net/geometry/polygonise/, вільний.

9. Smooth Voxel Mapping: a Technical Deep Dive on Real-time Surface Nets and Texturing [Електронний ресурс] Режим доступу: https://bonsairobo.medium.com/smooth-voxel-mapping-a-technical-deep-dive-on-real-

time-surface-nets-and-texturing-ef06d0f8ca14, вільний.

10. Coding Adventure: Marching Cubes [Електронний ресурс] Режим доступу: https://www.youtube.com/watch?v=M3iI2l0ltbE, вільний.

11. An Implementation of the Marching Cubes Algorithm [Електронний ресурс] Режим доступу: https://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html, вільний.

12. Coding Adventure: Ray Marching [Електронний ресурс] Режим доступу: https://www.youtube.com/watch?v=Cp5WWtMoeKg, вільний.

13. Ray Marching for Dummies! [Електронний ресурс] Режим доступу: https://www.youtube.com/watch?v=PGtv-dBi2wE, вільний.

14. Volume Rendering for Developers: Foundations [Електронний ресурс] Режим доступу: https://www.scratchapixel.com/lessons/3d-basic-rendering/volume-rendering-for-developers/ray-marching-algorithm, вільний.

15. Ковальов К.В. ПРОЄКТУВАННЯ СИСТЕМИ АДАПТИВНОЇ ГЕНЕРАЦІЇ ІГРОВОГО ПРОСТОРУ //3-я Міжнародна науково-практична конференція «SCIENTIFIC PROGRESS: INNOVATIONS, ACHIEVEMENTS AND PROSPECTS» Зб. матеріалів форуму. Т.7. Конференція «Технічні науки» – Харків: ХНУРЕ. 2022. – С.135-137.