МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ АВТОМАТИЗОВАНИХ СИСТЕМ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Комп'ютерне моделювання процесу руйнування тіл у 3D іграх**

Виконав: студент 2 курсу, групи 8.1219-пзс
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
О.Р. Гармаш
(ініціали та прізвище)
Керівник проф., д.ф.-м.н. В.Г. Вербицький
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «АйтіДіменшн»
В.С. Тряпічко
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Інженерний навчально-науковий інститут
Кафедра _____ програмного забезпечення автоматизованих систем
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри_В.Г. Вербицький
"_01_" __вересня_____2020_року

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

Гармашу Олексію Романовичу
(прізвище, ім'я, по батькові)

1.Тема роботи ___Комп'ютерне моделювання процесу руйнування тіл у 3D іграх_____

керівник роботи Вербицький Володимир Григорович, д.ф.-м.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від _"25"_травня_2020_року № 600-с_____
2. Строк подання студентом кваліфікаційної роботи___30.11.2020____
3. Вихідні дані магістерської роботи

- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми розпізнавання мов та розробка методів її вирішення;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) слайдів презентації

6. Консультанти розділів магістерської роботи

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

7. Дата видачі завдання 01.09.2020

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.09-10.09.20	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.2020	виконано
3	Аналіз існуючих методів рішення	13.09-17.09.20	виконано
4	Дослідження області руйнування у 3D іграх	18.09-24.09.20	виконано
5	Узгодження подальших дій з науковим керівником	25.09-26.09.20	виконано
6	Аналіз теоретичних відомостей	27.09-05.10.20	виконано
7	Проектування прототипу застосунку	05.10-15.10.20	виконано
8	Узгодження застосунку з науковим керівником	15.10-16.10.20	виконано
9	Реалізація функціоналу застосунку	16.10-27.10.20	виконано
10	Представлення отриманих результатів науковому керівнику і узгодження плану подальшого дослідження	27.10-28.10.20	виконано
11	Реалізація функціоналу гри	28.10-15.11.20	виконано
12	Проведення аналізу можливостей розробленого застосунку	15.11-16.11.20	виконано
13	Оформлення звіту	16.11.20-27.11.20	виконано

Студент _____ Гарнага О.Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Вербицький В.Т.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 106

Рисунків: 46

Таблиць: 5

Джерел: 35

Гармаш О.Р. Комп’ютерне моделювання процесу руйнування тіл у 3D іграх.

Кваліфікаційна робота для здобуття ступеня вищої освіти магістра за спеціальністю 121 – Інженерія програмного забезпечення, науковий керівник В.Г. Вербицький. Інженерний навчально-науковий інститут ЗНУ, 2020.

Мета кваліфікаційної роботи полягає у вивченні методів та засобів руйнування 3D об’єктів, якісному та кількісному порівнянні наявних методів між собою в контексті відеоігор, а також створенні власної системи руйнувань на основі наявних рішень та алгоритмів. Вхідними даними системи являються 3D об’єкти. В свій час система повинна при певних діях користувача над певним 3D тілом змоделювати та відобразити реалістичне руйнування даного об’єкту.

Досліджено методи, засоби та сучасні системи, що реалізують механіку руйнувань, а також можливе подальше застосування системи у відеоіграх. Спроектовано та реалізовано тривимірну гру для мобільних пристроїв під управлінням операційної системи Android за допомогою мови програмування C# та ігрового движка Unity 3D.

Ключові слова: *ІГРОВИЙ ДВИЖОК, РУЙНУВАННЯ, 3D ОБ’ЄКТ, MESH, ФРАКТУРИЗАЦІЯ, ПОЛІГОНИ, СЦЕНА.*

SUMMARY

Pages: 106

Figures: 46

Tables: 5

Sources: 35

Harmash O. R. Computer simulation of the process of destruction of bodies in 3d games.

Qualification work for higher master's degree in specialty 121 - Software Engineering, supervisor V.G. Verbitsky. Engineering Educational and Scientific Institute ZNU. Faculty of Energy, Electronics and Information Technology, 2020.

The aim of the qualification work is to study the methods and means of destruction of 3D objects, qualitative and quantitative comparison of existing methods with each other in the context of video games, as well as creating your own system of destruction based on existing solutions and algorithms. The system input is 3D objects. At one time, the system must, under certain user actions on a certain 3D body, simulate and display a realistic destruction of the object.

Methods, means and modern systems that implement the mechanics of destruction, as well as possible further application of the system in video games are studied. Designed and implemented a three-dimensional game for mobile devices running the Android operating system using the C # programming language and the Unity 3D game engine.

Keywords: GAME ENGINE, DESTRUCTION, 3D OBJECT, MESH, FRACTURING, POLYGONS, SCENE.

ЗМІСТ

ВСТУП

Актуальність теми

Відеоігри стали невід'ємною частиною людського життя, зокрема у сфері розваг. У сучасному світі вони набули такої популярності та впливу, що деякі держави, наприклад США, визнали їх видом мистецтва [9]. Відеоігри почали з'являтися ще із середини XX ст.. Першою відеогрою була шахова комп'ютерна програма, розробником якої являвся Алан Тюрінг [8]. Однак першою успішною спробою створити дійсно хороший ігровий продукт, який використовує відео для зворотного зв'язку з гравцем, належить Вільяму Гігінботаму та його грі Tennis For Two. З того часу ігрова індустрія почала свій активний розвиток: ігор розроблялося все більше, складність та важкість їх розробки збільшувалася.

З появою 3D ігор перед розробниками та геймдизайнерами з'явився величезний полігон для експериментів. Одним із найважливіших була і досі є передача якомога більш реалістичної поведінки реального світу у контексті відеогри: багатополігональні моделі середовища, реалістичне освітлення, тіні, штучний інтелект ігрових та неігрових персонажів (NPC), руйнування об'єктів, якщо вони руйнуються і в реальному світі тощо. Найцікавішим та одним із найскладніших є моделювання реалістичної поведінки руйнування об'єктів в ігровому середовищі. Фізика руйнувань або просто система руйнувань наявна в тому чи в іншому випадку в кожній комп'ютерній грі, проте більшість із них мають недоліки. До таких слід віднести, зокрема, велику примхливість до комп'ютерних ресурсів, неоднозначну візуалізацію самого процесу руйнування. Також деякі такі системи або закриті власне розробниками гри, або платні, наприклад пакети в Unity Asset Store. Тому актуальною проблемою залишається розробка легковагових систем руйнувань та подальше впровадження їх у майбутні ігрові проекти.

Мета і завдання дослідження

Мета дослідження полягає у аналізі методів візуалізації процесу руйнування 3D об’єктів, їх якісних та кількісних характеристиках, особливостей їх роботи та реалізації, а також у створенні власної системи, яка буде за своєю метою і виглядом 3D грою. У моєму випадку це представляється як ігрова сцена з різними об’єктами, які користувач (гравець) може зруйнувати. Система повинна коректно відстежувати дії користувача, та якомога швидше і краще відобразити процес та результат руйнування.

Об’єкт дослідження

Об’єктом дослідження є візуалізація 3D об’єкту, що знаходиться на ігровій сцені та з яким користувач проводить маніпуляції за допомогою систем вводу-виводу: комп’ютерна миша, клавіатура, монітор.

Предмет дослідження

Предметом дослідження є власне процес фрактуризації та механіка руйнування створеного 3D об'єкта, що знаходиться на ігровій сцені.

Методи дослідження

Для розв’язання представлених завдань та цілей використовуються такі методи дослідження:

- аналіз джерел про маніпулювання 3D об’єктами в контексті комп’ютерної гри;
- практичне тестування та вияв характеристик серед наявних систем руйнування;
- об’єднання та фільтрування отриманих результатів дослідження;
- проведення аналізу відмінностей та схожості з-поміж існуючих систем руйнування.

Наукова новизна одержаних результатів

Одержані результати, перш за все, мають показати як переваги Realtime руйнування проти заскриптованої анімації, так і недоліки у вигляді дуже великих вимог до комп'ютерного заліза, що будуть проявлятися у вигляді більш реалістичної фізики руйнування у першому випадку та низького або нестабільного FPS у другому випадку. Проаналізувавши отримані дані можливе буде створення більш універсальної системи у вигляді пакету підпрограм та викладення його в публічний доступ.

Практичне значення отриманих результатів

На основі отриманих результатів можлива розробка власної 3D гри, основною геймплейною особливістю буде саме руйнування об'єктів. Отримані результати також допомагають зрозуміти, які саме інструменти (мови програмування, середовища) роблять розробку системи руйнувань більш швидкою та продуктивною і які проблеми виникають при їх використанні. Також можна зробити висновки щодо вдалості розробленої програмної архітектури та подальшої її рефакторингу.

Загалом, отримані результати дають змогу розробити основний прототип , який реалізую більшість із поставлених задач руйнування 3D об'єктів у контексті відеогри.

Апробація результатів

Результати роботи було представлено на науково-технічних конференціях студентів, магістрантів, аспірантів, молодих вчених [34, 35].

Глосарій

Ігровий движок — центральна програмна частина будь-якої відеогри, яка відповідає за всю її технічну частину та дозволяє полегшити та уніфікувати розробку ігор на декілька платформ, т.з. багатоплатформовість. Зазвичай складається з движку рендерингу, фізичноого движку, звуків, анімацій,

системи скриптів , ігрового штучного інтелекту, мультипотоковості та системи керування пам’яттю. Також ігрові движки надають набір візуальних інструментів для розробки. Такі інструменти включають в себе інтегроване середовище розробки. Ігрові движки платформонезалежні і дозволяють запускатися на різних платформах, такі як консолі (PS4, Xbox ONE) та персональних комп’ютерах (PC).

Фізичний движок — програмний движок, що переносить закони фізики реального світу у віртуальний (наприклад, ігри). Умовно поділяються на два види: наукові та ігрові. Ігровий тип фізичного движку використовується в комп’ютерних іграх як основний компонент, ланка ігрового движку. Це накладає деякі зобов’язання на нього: він повинен працювати у режимі реального часу, щоб відтворювати всі фізичні процеси у грі с той самою точністю, що і у реальному світі. Однак разом з тим від ігрового фізичного движка не потребується висока точність розрахунків. Основна вимога до ігрового фізичного движка — візуальна реалістичність, тобто для досягнення цієї цілі не потрібно виконувати надточні розрахунки. У свій час, наукові фізичні движки використовуються у науково-дослідницькій діяльності, такі як симуляції фізичних процесів та їх розрахунки. Звичайно, в даному випадку важлива дуже висока точність розрахунків, а швидкість цих самих розрахунків відходить на другий план. Слід зазначити, що фізичні движки симулюють не всі фізичні закони, а лише деякі з них. Серед таких слід виділити наступні:

- динаміка абсолютно твердого тіла;
- динаміка деформованого тіла;
- динаміка рідин;
- динаміка газів;
- поведінка тканин;
- поведінка мотузок.

Згідно з останніх опитувань найпопулярнішими фізичними движками є nVidia PhysX та Havok.

Полігональна сітка — або «меш» (від анг. *Mesh*) — це сукупність вершин, ребер та граней, котрі визначають форму об’єкта в комп’ютерній графіці та моделюванні. Гранями в меші зазвичай виступають полігони — трикутники, чотирикутники або інші прості опуклі багатокутники. Полігональні сітки представляються багатьма способами:

- список граней;
- «крилате» представлення — кожна точка ребра вказує на дві вершини, дві грані та чотири ребра, котрі її торкаються;
- полуреберні, або half-edge сітки — теж саме, що і крилате представлення, за тією відмінністю, що виуористовується інформація лише половини грані;
- таблиця кутів — таблиця вершин складена таким чином, що її обхід неявно задає полігони;
- вершинне представлення — задаються тільки вершини, що вказують на інші вершини, а інформація про грані та ребра неявно викладена в цьому представленні.

Сцена — це деякий віртуальний простір, що представляє собою сукупність об’єктів, їх матеріалів, анімації, освітлення, часток тощо. Розміри сцени визначаються кубом мінімального розміру, щоб охопити усі елементи цієї сцени.

Камера — невізуалізований об’єкт, що відображає вигляд простору в сцені з конкретної оглядової точки.

Рендеринг — процес отримання зображення за деякою моделлю за допомогою комп’ютерної програми.

Рейкастинг — метод рендерингу, при якому сцена будується за допомогою розрахунків пересікання променів з візуалізованою поверхнею.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРОБЛЕМИ РУЙНУВАННЯ ОБ'ЄКТІВ У 3D ІГРАХ

1.1 Загальні відомості про моделювання 3D об'єктів

Руйнування — один із процесів візуалізації порушення цілісності деякої структури. Однак, щоб зруйнувати об'єкт на сцені його необхідно створити. Створення об'єкту від етапу ідеї до етапу її реалізації називають моделюванням. На сьогоднішній день в комп'ютерній графіці переважають два способи представлення об'єктів на сцені — 2D та 3D графіка. Якщо у традиційній 2D графіці об'єкт «малюють» за допомогою двох координат X та Y, то у 3D графіці додається ще так звана вісь глибини — Z координата. За її допомогою можна досягти представлення інформації про об'єкт з усіх його сторін, а не тільки з однієї сторони, як у 2D графіці.

Перші ігрові проекти, звичайно, були побудовані за допомогою 2D графіки. Перші кроки до становлення 3D ігор такими, якими ми їх знаємо, відбулися у 80 рр. XX ст. Так, в листопаді 1982 року було випущено Microsoft Flight Simulator 1.0 для платформи PC і користувалася величезною популярністю у гравців. А вже у 1992 році світ побачив перший 3D шутер — Wolfenstein 3D від добре відомої всім компанії id Software. В ній вперше була застосована перспектива — по мірі віддалення або приближення головного героя до будь яких об'єктів вони також віддалялися або наближувалися. Також вперше було застосоване так зване текстурування — накладення растрових зображень поверх каркасу об'єкта. І вже настпуного року знову ж id Software випускає настпуний свій хіт — шутер від першої особи Doom. Не хважаючи на те, що вороги були спрайтами, все навколишє середовище було виконано в 3D стилі і гра користувалась не меншою популярністю аніж Wolfenstein 3D.

Однак справжній переворот 3D комп'ютерної графіки відуьвся у 1996 році з виходом Quake від тієї ж компанії id Software. В ній з'явилося багато

нових технологій, такі як Z-буфер, використаний повністю полігональний движок, текстурування також використовувалося у повній мірі.

З розвитком 3D ігор розвивалася і апаратна частина, бо справлятися з такою кількістю обчислень тільки процессору було занадто важко, а місцями і нереально. Почали з'являтися перші відеокарти 3D-прискоренням — DirectX, VooDoo Graphics та інші. Причиною такого розвитку стало істотне збільшення кількості полігонів 3D моделі. Саме полігони — це ті примітиви, з яких складається 3D об'єкт.

Існує декілька способів моделювання, однак самими популярним є полігональне моделювання. Смисл такого підходу заключається в тому, шо для побудови 3D моделі використовуються геометричні двомірні примітиви. Для ігрових движків це трикутники, для інших цілей зазвичай використовують чотирикутники або фігури з більшою кількістю кутів. Основним правилом моделювання 3D об'єкту можна вважати те, що чим більша кількість полігонів використовується у моделі, тим більш наближеним до реального об'єкту стає ця 3D модель. Однак у цього підходу є і інша сторона. Зі збільшенням кількості полігонів збільшується кількість точок по яким вони будується, тим самим більше даних оброблюється процесором або відеокартою. Тому для ігор використовуються полігони, представлені трикутниками і їх кількість не безрозмірна.

Слід зазначити, що полігональне моделювання відноситься до пустотілого моделювання, де об'єкт має тільки об'єм, але всередині пустий. Існують програмні продукти, які дозволяють моделювати твердотільні об'єкти. Однак таке моделювання занадто громіздке для ігор, але дуже корисно для будь-яких технологічних процесів.

1.2 Приклади задач моделювання 3D об'єктів

Моделювання об'єктів може застосовуватися у будь-яких сферах життєдіяльності людини. Найпопулярнішою задачею можна вважати створення

ігор, а саме ігрових сцен: всі об’єкти на ній повинні бути змодельовані, щоб надати гравцеві максимальний ігровий досвід та занурення у віртуальний світ.

Іншим не менш значним прикладом задачі є моделювання поведінки тих чи інших об’єктів у певних технологічних процесах.

3D моделювання часто застосовується при створенні нових видів продукції, коли 2D зображення недостатньо для реалізації поставлених цілей.

Також моделювання незамінне для демонстрації майбутнього виробу: неможливо розпочати виготовляти виріб без готової 3D моделі.

1.3 Основні постановки задач моделювання 3D об’єктів

На сьогоднішній день неможливо представити ігрову та кіноіндустрію без використання 3D моделей. Особливою відмінністю 3D об’єктів, що створюються на даний час, від об’єктів минулого, являється їх комплексність та підвищена складність, пов’язана із самими реальними об’єктами моделювання. Побудова таких складних моделей потребує чітких правил і підходів, які дозволили б знизити вартість їх розробки та зменшити появу складновирішальних помилок. Тому основними постановками задач у моделюванні 3D об’єктів можна вважати наступні:

- **Дослідження об’єкту моделювання** та розробка детального технічного завдання щодо створення моделі;
- **Концептуальна постановка задачі** — сформульований перелік основних питань та правил відносно властивостей та поведінки об’єкту;
- **Реалізація моделі** — власне вирішення задачі шляхом розробки моделі за допомогою існуючих рішень (програмні продукти, комп’ютерні системи тощо)

1.4 Моделювання та руйнування

Як вже сказано було раніше моделювання та руйнування тісно пов’язані один з одним. Однак у іграх довгий час ця задача була досить нетривіальною та складною у реалізації. Перші спроби руйнування у відеоіграх були впроваджені ще в 1979 році. Цю механіку у деякому її прототипному вигляді було реалізовано в грі «Asteroids» від компанії Atari [7], де ви знищували астероїд, і він розщеплювався на більш дрібні шматочки, які в свою чергу також могли руйнуватися ще на більш менші шматочки і так далі. Однак, не зважаючи на відносну простоту, у той час навіть такий спосіб реалізації руйнування не був звичайною задачею у розробці комп’ютерних ігор та у комп’ютерній графіці вцілому. По мірі прогресування комп’ютерної графіки та комп’ютерів ефекти руйнування стали більш поширеними в сучасних комп'ютерних іграх завдяки підвищеній потужності цих самих комп’ютерних систем. Такі ефекти, як вибух будівель, руйнування скла та інші види розщеплювання предметів, значно покращують занурення та відчуття від гри, особливо при поєднанні їх з деякою анімацією або системою часток, однак ці два методи частіше за все комбінують один із одним. Більшість сучасних ігор, щоб досягти ефективності в режимі реального часу, попередньо розбивають свої ігрові об’єкти, тому при попаданні в них деякого іншого об’єкту вони переставляють їх на попередньо розбиті версії за ефектами диму або вибухом, щоб гравці насправді нічого не помітили.

Деякі проблеми зі створенням системи руйнування мають багато спільного з тим, як працює комп'ютерна графіка, зокрема моделювання. Перша проблема полягає в тому, що 3D-сітки — це порожнисті оболонки замість заповненого об'єму, які створюють наступну проблему: якщо розрізати сітку, вона просто залишить шматок оболонки без наповненості у місті розрізу. Тому потрібно чимось заповнити створений проміжок. Такими примітивами у полігональних сітках , з якого і складається 3D об’єкт слугують трикутники. Ще одна проблема пов'язана із тим, як зберігаються дані на відеокарті. Сітка

може зберігатися в об'єкті вершинного буфера, VBO, і динамічно змінювати її — це дороге і трудомістке завдання, яке може уповільнити роботу навіть досить потужної системи. Тож потрібні були певні дії та алгоритми, що дозволяли виконувати такі задачі швидко та без великих втрат у продуктивності. Прикладом таких дій може служити використання діаграм Вороного. Діаграма Вороного є особливим видом розкладання метричного простору і названа на честь російсько-українського математика Георгія Вороного, але вона розглядалася ще в 1644 р. Діаграма має цілком природний малюнок, який можна знайти в різних частинах природи. Він використовується в багатьох різних сферах, таких як металургія, астрономія та інформатика.

Діаграми Вороного можна легко використовувати з іншими алгоритмами, наприклад з тріангуляцією Делоне, і легко переходити один від одного, використовуючи той самий набір вершин. Використання таких алгоритмів і підходів дозволяє розробити систему динамічного руйнування, тобто система буде видавати кожний раз різний результат, на відміну від заскриптованого руйнування.

1.5 Методи візуалізації руйнування об’єктів

Більшість систем руйнування в іграх можна поділити на дві частини, які визначають певну послідовність дій, потрібні для досягнення поставленої задачі, а саме:

- Підготовка 3D геометрії до розбиття її на фрагменти, що можна виконувати не в рантаймі, або Geometry Preparation;
- Власне симуляція руйнування вже фрагментованого 3D об’єкту у режимі реального часу, або Runtime Destruction.

Такий загальний підхід ще називають *preshattering*, або попереднє руйнування. Коли гостро стоїть задача впровадження системи руйнування у ігрові продукти, зазвичай першочерговою задачею являється якомога реалістичніший вигляд і обчислювальна дешевизна у порівнянні з повністю реаль-

ним фізичним процесом. Саме тому preshattering є дуже популярною моделлю для систем руйнування, особливо у проектах з невеликими командами розробників та студій з недостатнім бюджетом для реалізації власної системи руйнування. І справді, набагато дешевше імітувати знищення, маючи обмежену кількість частин, що розпадаються, ніж насправді визначати, як об’єкт ділиться на шматки, виходячи з його фізичних властивостей під час виконання. Так відбувається тому, що підтримка стабільної та високої частоти кадрів є надзвичайно важливою за багатьма причинами: гравець менше втомлюється під час гри; низька кількість кадрів неприємна у деяких онлайн-іграх, де це може вирішити результат турніру або змагання. З цих причин дозволяти кількості фрагментів безперервно збільшуватися досить не бажано.

У таблиці 1 наведені загальноприйняті методи та підходи, що відносяться до кожної з двох частин систем руйнування.

Таблиця *1*

Методи та підходи системи руйнувань

Підготовка геометрії (*Geometry Preparation*)	**Знищення під час виконання (*Runtime Destruction*)**
Ручне моделювання	Заскриптована анімація
Метод вирізання	Булеві операції реального часу
Булеві операції	Метод скінчених елементів
Тетраедралізація	Метод ламких зв’язків
Опукла декомпозиція	Метод складеного твердого тіла
Діаграми Вороного	
Слайсинг (Slicing)	

Не зважаючи на кількість методів та підходів як для Geometry Preparation, так і для Runtime Destruction системи руйнування, деякі з них працюють та добре імітують лиш певні властивості матеріалу, але дуже погані для імітації інших властивостей. Також слід зазначити, що деякі методи

Geometry Preparation розраховують та візуалізують фрагменти цільного об’єкту з гладкими поверхнями., які підходять для таких матеріалів, як мінерали, скло та інші, а деякі методи добре працюють для руйнування жорстких, негнучких тіл. Тому такі методи можуть не виконувати своїх функцій або просто бути невідповідними для таких матеріалів, як дерево, яке розбивається на фрагменти і являється гнучким тілом.

1.6 Підготовка геометрії (Geometry Preparation)

Першим етапом у створенні системи руйнування виступає підготовка геометрії, або Geometry Preparation. Цільний об’єкт, який треба зруйнувати, спочатку повинен розбитися на шматочки під час моделювання. Існує багато популярних методів, які використовуються для поділу 3D моделі на фрагменти. При визначенні придатності до використання того чи іншого методу треба дотримуватися певних їх якостей, а саме:

- Контроль якості — тобто дизайнер, або інша відповідальна за фрагментацію об’єкту людина, використовуючи систему руйнування, має постійний та високий рівень контролю над тим, як саме створюються фрагменти;
- Метод не повинен бути занадто дорогим в плані комп’ютерних ресурсів та простий у використанні;
- Метод повинен вирішувати проблему заповнення вільного простору фрагментованого об’єкту, оскільки робота над розділенням моделі ведеться на полігональній сітці (mesh). *Полігональна сітка* — або «меш» (від анг. *Mesh*) — це сукупність вершин, ребер та граней, котрі визначають форму об’єкта в комп’ютерній графіці та моделюванні. Гранями в меші зазвичай виступають полігони — трикутники, чотирикутники або інші прості опуклі багатокутники. Саме тому метод повинен реконструювати внутрішню частину об’єкту, коли розбиває (фрагментує) модель.

1.6.1 Ручне моделювання

Найпростішим способом зробити з цільного об’єкту фрагментований — це змоделювати таке розбиття за допомогою спеціалізованого програмного забезпечення. До таких можна віднести 3D редактори. З серед великої кількості таких редакторів можна виділити декілька найпопулярніших з них: Blender, 3ds Max, Autodesk Maya. Кожний з цих редакторів має на меті найосновнішу задачу — моделювати та анімувати 3D об’єкти. Однак, не зважаючи на одну ціль, різняться між собою.

Blender — безкоштовний, професійний пакет для створення 3D графіки, який містить засоби моделювання, анімації, рендерингу, постобробки тощо [6]. Слід зазначити, що це найпопулярніший з безкоштовних редакторів. Причиною тому слугують постійне оновлення, невеликий розмір самого пакета, додатки, які можуть публікувати будь-який користувач. Також програмна система має велику кількість додаткових функцій:

- Вбудовані механізми рендерингу, які по своїй суті являються універсальними, або використання зовнішніх рендерів;
- Професійні інструменти для створення анімацій, такі як скелетна анімація, сіткова деформація, анімація за ключем та динаміка твердих і м’яких тіл;
- Має підтримку геометричних примітивів, використовуючи їх у полігональному моделюванні, а також скульптурне моделювання.

Система Blender має два режими роботи: *об’єктний режим* та *режим редагування* [5]. Об’єктний режим дозволяє маніпулювати з конкретними об’єктами. До таких дій відносяться: масштабування, наближення, обертання об’єкту. Режим редагування призначений для маніпуляцій саме з даними об’єкту, такі як вершини, ребра, поверхні тощо.

Однією з особливостей Blender є використання гарячих клавіш, кількість яких може заплутати нових користувачів цього 3D редактор. Для зручності побудови моделей за ескізом, або просто для спостереженням за моделлю з різних ракурсів, в Blender є функція розділу робочого простору,

який, за бажанням, можна перенести на інший монітор, тим самим ще більше покращити досвід використання даного програмного продукту для роботи з 3D графікою.

Autodesk 3ds Max являється повнофункціональним професійним програмним продуктом для роботи з 3D графікою та анімацією. Сам редактор має при собі величезну кількість готових архітектурних проектів, таких яяк двері, вікна, елементи декору, рослинність тощо. За допомогою потужного вбудованого движка рендеру цей редактор дозволяє створювати сцени з високою правдоподібністю об’єктів на ній. Також він дозволяє налаштовувати освітленість тривимірного проекту, щоб досягти максимального фотореалістичного ефекту [4].

3ds Max дозволяє створювати різні за складністю 3D об’єкти за допомогою різних методів, які вже вбудовані в цей редактор. Вони включають в себе полігональне моделювання, який може використовуватися при створення як багатополігональних, так і малополігональних об’єктів; модлювання за допомогою сплайнів різними модифікаторами, скульптурне моделювання тощо.

До переваг цього 3D редактора слід віднести також легкість в освоєнні, незважаючи на загальну його громіздкість; величезну мережу плагінів, які розширюють потенціал 3D моделювання та можуть буті налаштовані на специфічні задачі.

Особливістю цього 3D редактору також є його система рендеру, яку користувач може змінити на іншу за бажанням: замість вбудованого движку рендеру mental ray можна використовувати більш продуктивні та швидкі, як Maxwell або V-ray.

Загалом, 3ds Max має дуже багатий набір функцій та можливостей, однак на відміну від Blender, це платний програмний продукт і новачкам у комп’ютерній графіці може не підійти.

Autodesk Maya також відноситься до сімейства програмних продуктів, за допомогою яких можна працювати з тривимірною комп’ютерною графі-

кою та анімацією. На відміну від Blender та 3ds Max, які найчастіше використовуються дизайнерами та розробниками у іграх, Мауа широко застосовується в мультиплікації та кінематографії [3].

Програма має всі необхідні інструменти для роботи з 3d графікою та дозволяє реалізувати створення найкращої 3D графіки. Цей редактор має низку цікавих функцій серед яких слід виділити наступні:

- Моделювання фізики твердих та м’яких тіл;
- Моделюванні поведінки тканин;
- Симуляція поведінки рідких об’єктів, наприклад симуляція води;
- Потужний інструмент для створення та детальної проробки волосся, хутра тощо.

Однією з найважливіших особливостей Мауа є її відкритість, тобто будь-який сторонній розробник може налаштовувати її під конкретні потреби користувачів.

Також слід зазначити, що в Мауа вбудована мультиплатформенна мова MEL — Maya Embedded Language. Ця мова легка в освоєнні та дозволяє допрацьовувати основний функціонал за потребою.

Як видно з наведених інструментів, які дозволяють моделювати 3D об’єкти, ручне моделювання має найвищий ступінь контролю за процесом фрагментації, однак потребує спеціально навчену людину, що може поводитися з одним або декількома цими редакторами.

1.6.2 Булеві операції

Незважаючи на те, що один і той же самий об’єкт можна змоделювати декількома способами, таким як полігональне моделювання, існує найбільш простий спосіб — булеві операції. Булеві операції прийшли у 3D графіку з математики, а точніше з булевої алгебри, яка була названа так на честь свого засновника Джорджа Буля [2]. До булевих операцій відносять *об’єднання*, *віднімання* та *перетин*. В цих операціях завжди використовуються два

об’єкти, які називаються операндами. Роздивимось булеві операції детальніше.

Результатом булевого **об’єднання** двох об’єктів буде поверхня, що утворена поверхнями об’єктів, котрі беруть участь в даній операції. Результат булевої операції об’єднання зображений на рисунку 1.

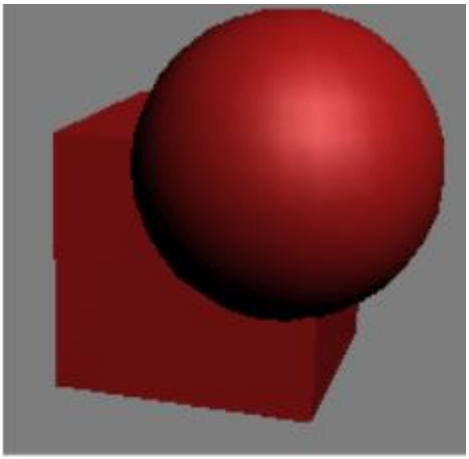

Рис. *1 Результат виконання булевої операції об’єднання*

Об’єкт, отриманий в результаті виконання булевої операції **перетину**, складається з поверхонь , які є спільними для двох об’єктів. Результат виконання булевої операції перетину наведений на рисунку 2.

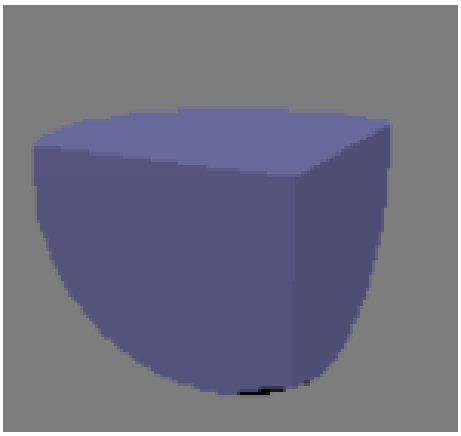

Рис. *2 Результат виконання булевої операції перетину*

Об’єкт, що виходить в результаті виконання логічної операції **віднімання**, містить в собі тільки точки поверхні першого операнда (об’єкта) за виключенням тих точок, які являються спільними для двох об’єктів. На відміну від булевої операції об’єднання та перетину, операція віднімання являється некомутативною, тобто різниця A-B не дорівнює B-A. На рисунках 3 та 4 наведені результати виконання булевою операції віднімання A-B та B-A відповідно.

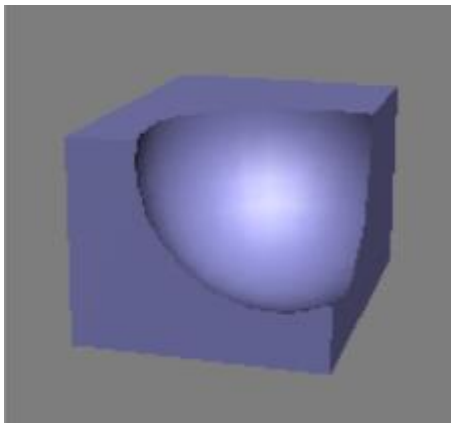

Рис. *3 Результат виконання булевої операції віднімання A-B*

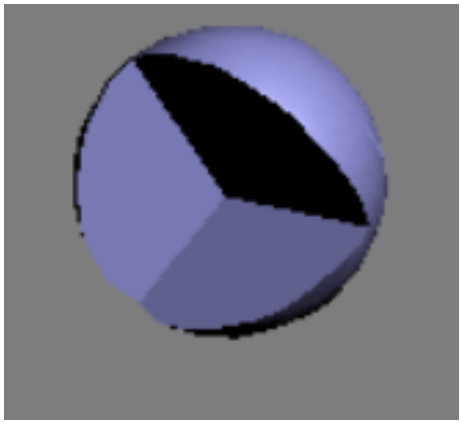

Рис. *4 Результат виконання булевої операції віднімання B-A*

Таким чином, булеві операції дозволяють розбити меш 3D моделі на фрагменти, однак недоліком цього методу є те, що метод являється не процедурним. Саме тому він повільний, однак має високу ступінь контролю над створенням фрагментів.

1.6.3 Метод вирізання

Цей метод в основному характеризується тим, що основним компонентом, що дозволяє розбити цільний 3D об’єкт на фрагменти являється так звана *карта зламів*, або *fracture map*. Така карта зазвичай представлена у вигляді звичайного файлу растрового зображення.

Основною ідеєю такої карти являється розташування білих ліній зламу на чорному тлі. Лінії зламу, або розбиття — це лінії різної форми, довжини, за проекцією яких на 3D об’єкт утворюються нові фрагменти, тим самим симулюючи руйнування. Таку карту можна створити вручну за допомогою спеціальних засобів для роботи з зображеннями, такими як Adobe Photoshop, Adobe Illustrator або інші програмні продукти, що дозволяють виконувати поставлені задачі з розробки карти розбиття.

Розбиття за допомогою карт переламів відбувається у декілька етапів. Після етапу створення карти, вона імпортується у вигляді файлу зображення у будь-який 3D редактор або движок, який дозволяє працювати з ними. Наступним кроком є проектування карти розбиття на власне 3D об’єкт. Після цього модель розбивається на фрагменти за заданими лініями на карті розбиття. На рисунку 5 наведений приклад розбиття об’єкту «стіна», що представлений 3D примітивом «куб», на фрагменти у вигляді цегли за допомогою карти розбиття з аналогічним малюнком.

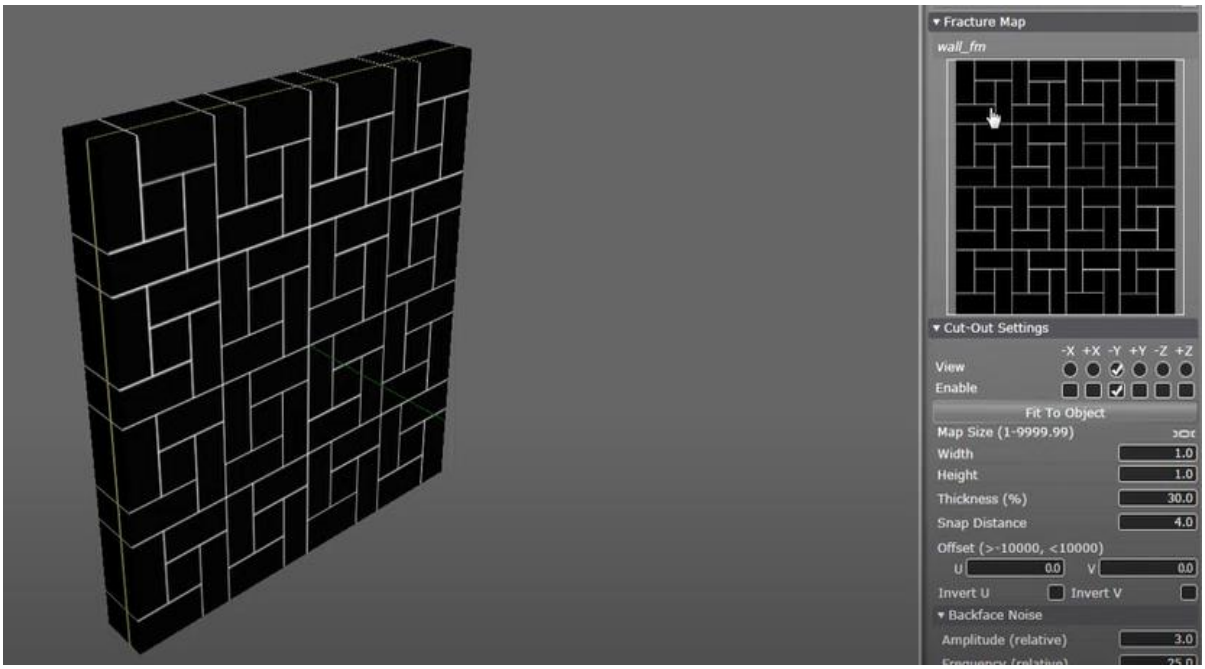

Рис. *5 Приклад розбиття на фрагменти за допомогою карти розбиття*

Цей метод також дозволяє в достатній мірі контролювати створення фрагментів на всіх стадіях. Це також є і недоліком цього підходу: малювання карти розбиття приходиться робити вручну. Але якщо не потрібна деяка специфічна форма фрагментів, а достатньо симулювати розбиття на різні фрагменти, то можна застосувати процедурну генерацію таких карт за допомогою, наприклад, діаграм Вороного. Це привносить більшу автоматизацію у процес створення та застосування карт розбиття та, відповідно, пришвидшує розробку системи руйнування.

1.6.4 Тетраедралізація

Тетраедралізація — це процес генерування тетраедричних сіток, які здані безліччю точок, що знаходяться у 3D просторі, та багатогранних сіток.

Тетраедрична сітка — це сітка, яка складається з графічних примітивів, такі як тетраедри.

Тетраедр — це чотиригранний об’єкт, де кожна сторона представлена у вигляді графічного примітиву трикутника.

Тетраедричні сітки потрібні для розрахування поведінки різних динамічних систем, таких як моделювання скінченних елементів для динаміки м’якого тіла. У контексті проектування системи руйнувань, тетраедрична сітка дуже корисна для отримання реалістичних переломів.

Основною ідеєю тетраедралізації для фрагментації об’єкту являється застосування цього методу на вхідному полігональній сітці 3D об’єкту за допомогою алгоритму тріангуляції Делоне. Після застосування цього методу, раніше порожня оболонка об’єкту заповнена тетраедрами, які точно підходять під вхідні полігональні сітки, і не видає артефактів при візуалізації. На рисунку 6 наведена схема, як виглядає внутрішня частина об’єкту, яка зазвичай порожня.

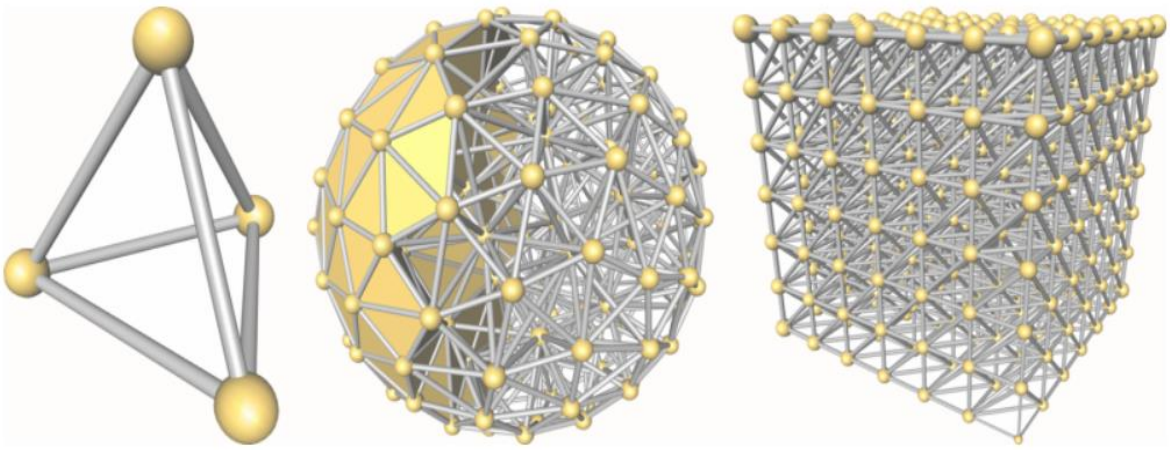

Рис. *6 Внутрішня структура об’єкту при тетраедралізації*

Метод тетраедралізації, незважаючи на всі свої переваги, не достатньо добре підходить для системи руйнувань в іграх, бо має достатньо високу розрахункову складність і тетраедралізувати достатньо велику частину об’єктів на сцені буде досить складно для комп’ютерної системи. До цього ж результат не завжди виходить таким, яким планувався спочатку.

1.6.5 Опукла декомпозиція

Більшість алгоритмів 3D графіки можуть застосовуватися тільки до моделей, які являються опуклими. Однак під час процесу моделювання, дизайнер або розробник може створити об’єкт, який логічно не поділений на частини, наприклад, руки, ноги, голова у моделі людини, або ці частини не є опуклими. Саме тому для таких об’єктів може застосовуватися метод опуклої декомпозиції.

Суть цього методу полягає в розбитті цільного 3D об’єкту на мінімальну кількість опуклих частин, застосовуючи набір подвійних графів. Результатом генерування сегментів є вихідна сітка, яка представляє собою набір опуклих поверхонь. Крім того, при правильній реалізації цього підходу, метод автоматично може виявляти анатомічну структуру вхідних 3D моделей. Тобто з високою вірогідністю метод опуклої декомпозиції може розбити ту ж модель людини на складові — руки, ноги голова, ніс, вуха тощо — доволі правильним і природнім чином.

Використання тільки цього методу для підготовки геометрії не є частою практикою. Зазвичай це метод використовують, щоб попередньо розбити об’єкт на опуклі частини, а вже потім застосовувати інші алгоритми для подальшого розбиття цих самих опуклих фрагментів. На рисунку 7 наведений приклад опуклої декомпозиції, де можна бачити чітке розділення складеної фігури на певні опуклі фрагменти, які є доволі логічними та природньо виглядаючи у контексті даної 3D моделі.

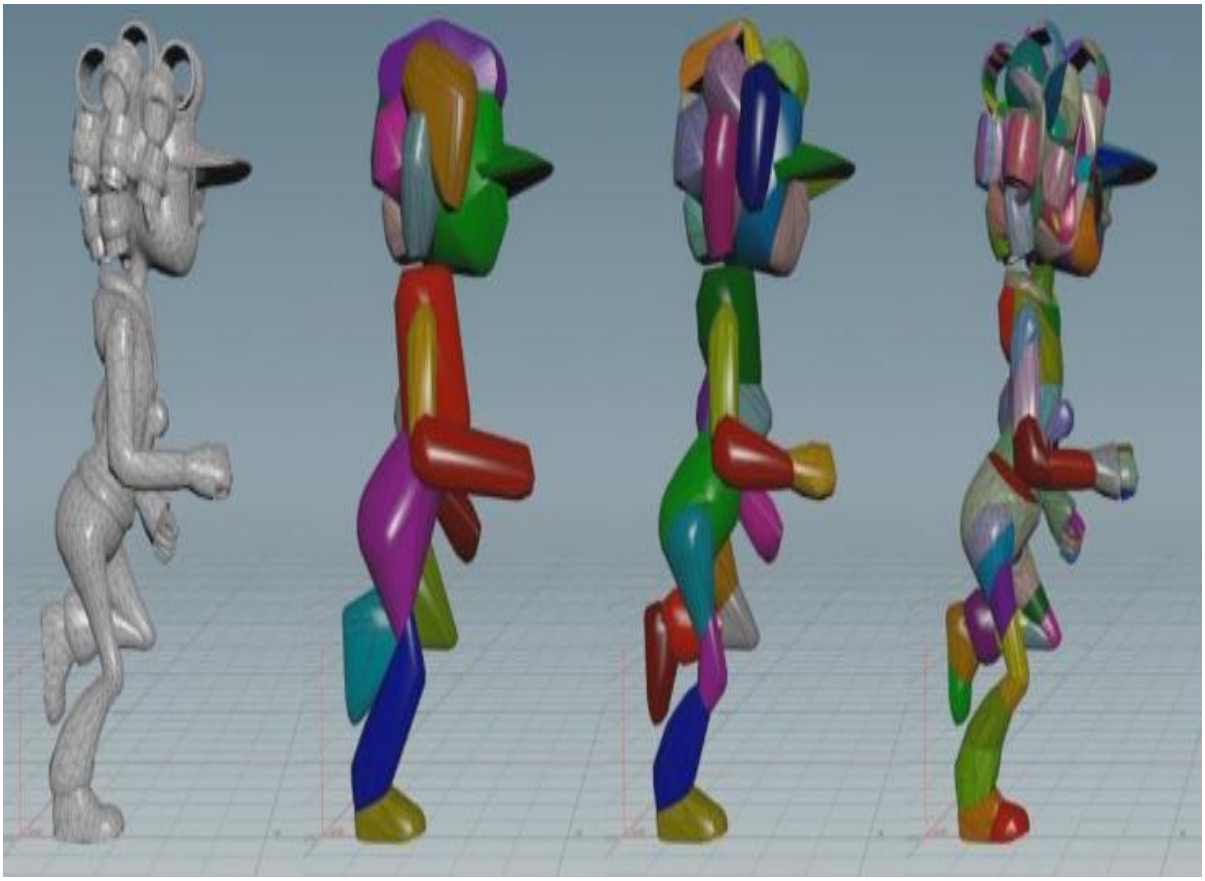

Рис. *7 Опукла декомпозиція та різні її апроксимації*

1.6.6 Метод розрізання

Як вже відомо, 3D об’єкт в іграх представлений у вигляді пустотілої полігональної сітки, яка складається з графічних примітивів, якими є трикутники. Цю сітку щонайменше можна розділити щонайменше два рази. Це робиться методом розрізання, або слайсинг.

Основним інструментом в методі слайсингу виступають площини. Площина — це нескінченна поверхня, до якої належать всі прямі, що проходять через будь-які дві її точки.

Алгоритм дій при слайсингу буде виглядати так:

1. Згенерувати об’єкт, який потрібно розрізати;
2. Згенерувати вручну або автоматично площини, якими буде розрізатися 3D об’єкт;

3. Знайти точки перетину площин з об’єктом та розрізати об’єкт по всім точкам;

4. Заповнити проміжки, які з’явилися на лінії розрізу.

Спосіб слайсингу являється одним із найпростіших у розумінні фрагментизації 3D об’єкту, але має також деякі складності та недоліки. До них можна віднести наступні:

- Час на виконання може бути занадто великим із зростанням полігональності моделі;
- Розрізані об’єкти слід заповнити трикутниками для запобігання візуалізації внутрішнього простору полігональних сіток. Для цього треба застосовувати алгоритм тріангуляції, що не є простою та тривіальною задачею.

Однак, слід зазначити, що даний спосіб генерує доволі цікаві та реалістичні фрагменти і має високу ступінь контролю над їх створенням: площини можна розташовувати або вручну на певних позиціях, змінюючи їх положення в рантаймі, або одразу генерувати. На рисунку 8 проілюстрований метод слайсингу, який застосували до 3D моделі.

Рис. *8* Метод слайсингу, застосований до 3D моделі

1.6.7 Діаграми Вороного

Діаграма Вороного — це розбиття площини з множиною точок S, при якому кожний регіон цього розбиття утворює множину точок, більш близьких до одного із елементів множини, ніж до будь-якого іншого. Точки в діаграмі Вороного називаються сайтами. Ці діаграм названі на честь Георгія Вороного, та має ще назви, такі як розбиття Вороного, розбиття Дирихле, мозаїка Вороного. У загальному, структурному вигляді має вигляд, як наведено на рисунку 9.

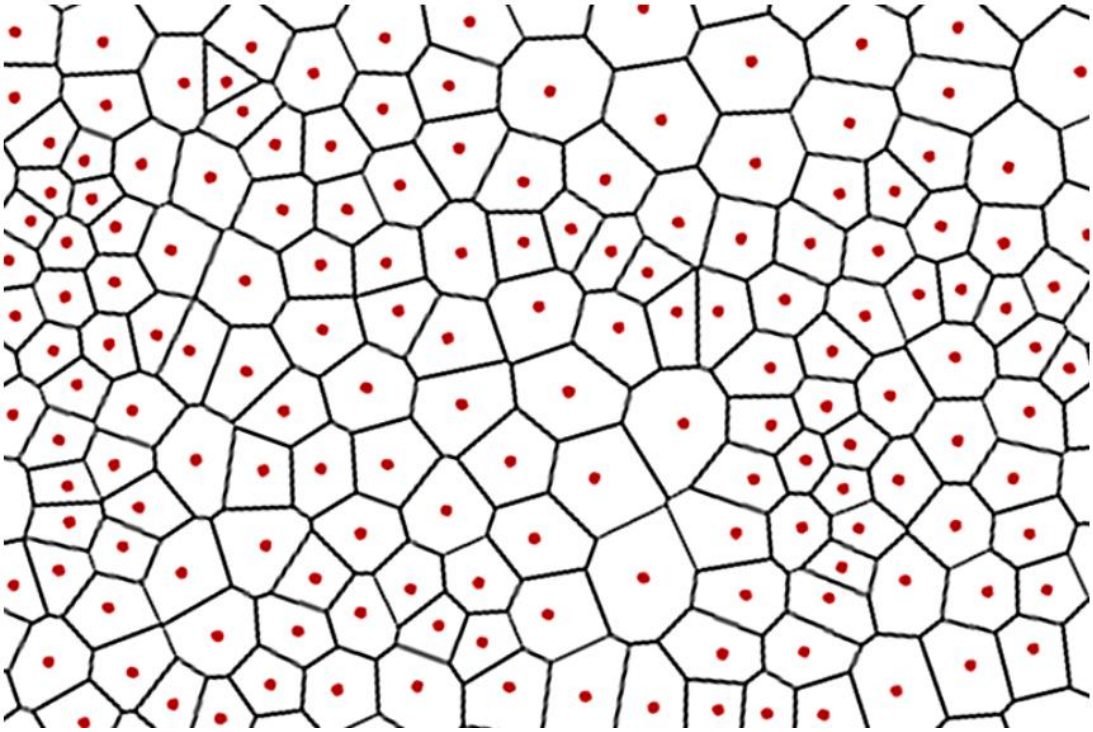

Рис. *9 Діаграма Вороного*

Діаграми Вороного дуже поширені в природі: візерунок на листі або на висушеній землі дуже схожі на діаграми Вороного. Також діаграми Вороного використовують в геодезії, картографії, в архітектурі та дизайні, археології тощо. Так на рисунку 10 можна побачити використання діаграми Вороного в картографії для виділення границь регіонів та подальшого їх аналізу.

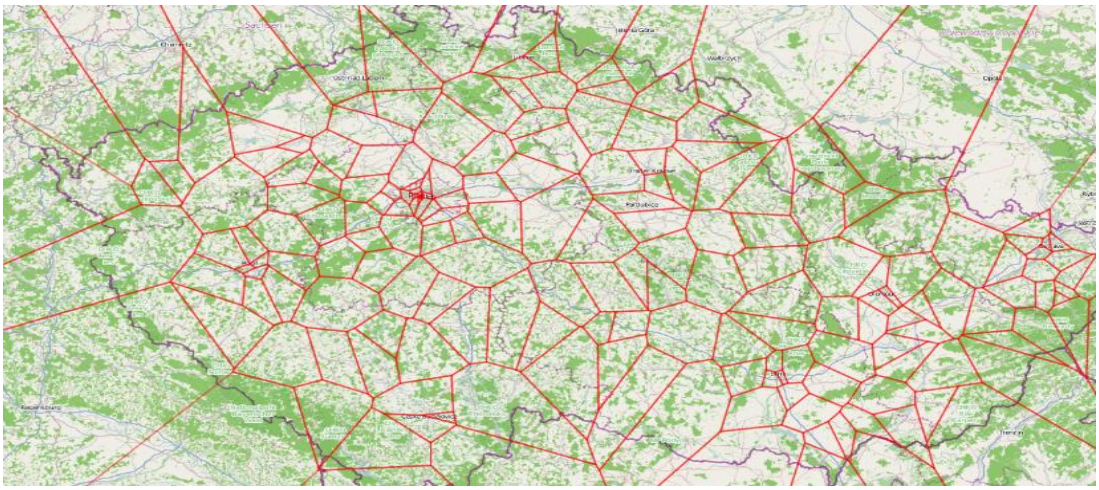

Рис. *10 Використання діаграм Вороного в картографії*

Зовнішній вигляд діаграм Вороного спонукнув розробників ігор використовувати їх при розробці власних систем руйнування. Як було вже сказано раніше, руйнування може використовуватися як центральна особливість механіки ігрового процесу. А з використанням розбиття вороного фрагменти набувають дуже реалістичного і природнього вигляду. Крім того, використання діаграм Вороного може розширити область об’єктів які можна зруйнувати гнучкими тілами, однак для цього потрібно застосовувати ще низку алгоритмів.

Більшість 3D редакторів та ігрових движків вміють розбивати об’єкти за допомогою діаграм Вороного. Наприклад, в ігровому движку Unreal Engine 4 є власний редактор руйнувань Chaos, в якому існує багато способів фрагментувати об’єкт, одним із котрих є використання діаграм Вороного . На рисунку 11 зображено роботу редактору Chaos, який входить до UE4.

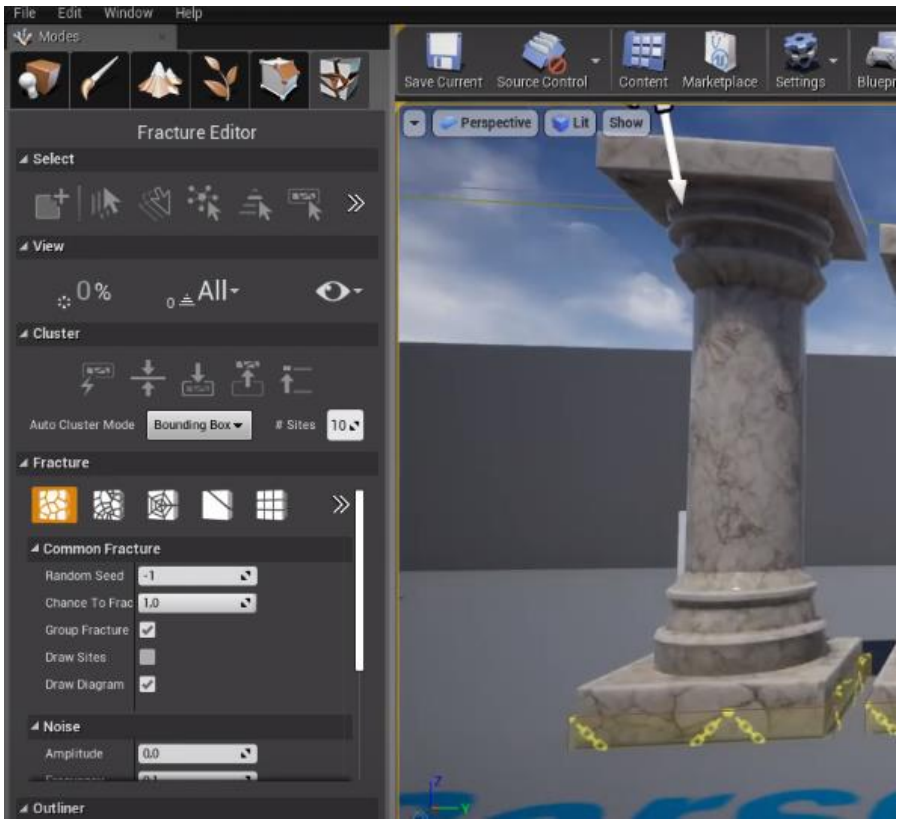

Рис. *11 Система Chaos в UE4*

1.6.8 Воксельна графіка

Voxel Graphics, або воксельна графіка — це засіб моделювання тривимірних об’єктів із вибірковими точками (вокселями) замість полігонів. Переваги цієї технології включають здатність обробляти дуже деталізовані об'єкти, а також об'єкти з нечітко визначеними межами, такі як хмари. Історичними перешкодами для прийняття Voxel Graphics були відсутність обробної потужності та відсутність стандартів для обміну та передачі наборів даних. Ці бар'єри поступово зникають завдяки швидкому розвитку потужності центральних та графічних процесорів комп’ютера, що дозволяє інтерактивно відтворювати воксельні моделі, а також різних технологій стиснення зображень, що забезпечить основу для стиснення великих 3D-зображень.

Графіка на основі вокселів, яку також називають об’ємною графікою, забезпечує альтернативу полігональним моделям. Поширення цієї технології було дуже обмеженим порівняно з багатокутною графікою. Одним із ключових факторів, що сприяють цій технології, є можливість інтерактивного відтворення моделей вокселів.

Об'ємна графіка може описувати об'єкти без чітко визначених меж, необхідних для полігонального опису. Вокселі представляють внутрішні деталі об’єкту, що полегшують такі операції, як відсікання та вирізання. Ці та інші властивості можуть посилити занурення в реальність, наприклад будівля, показана на рисунку 12. На відміну від полігональної графіки, наприклад, втіленої в OpenGL, напівпрозорість обробляється безпосередньо.

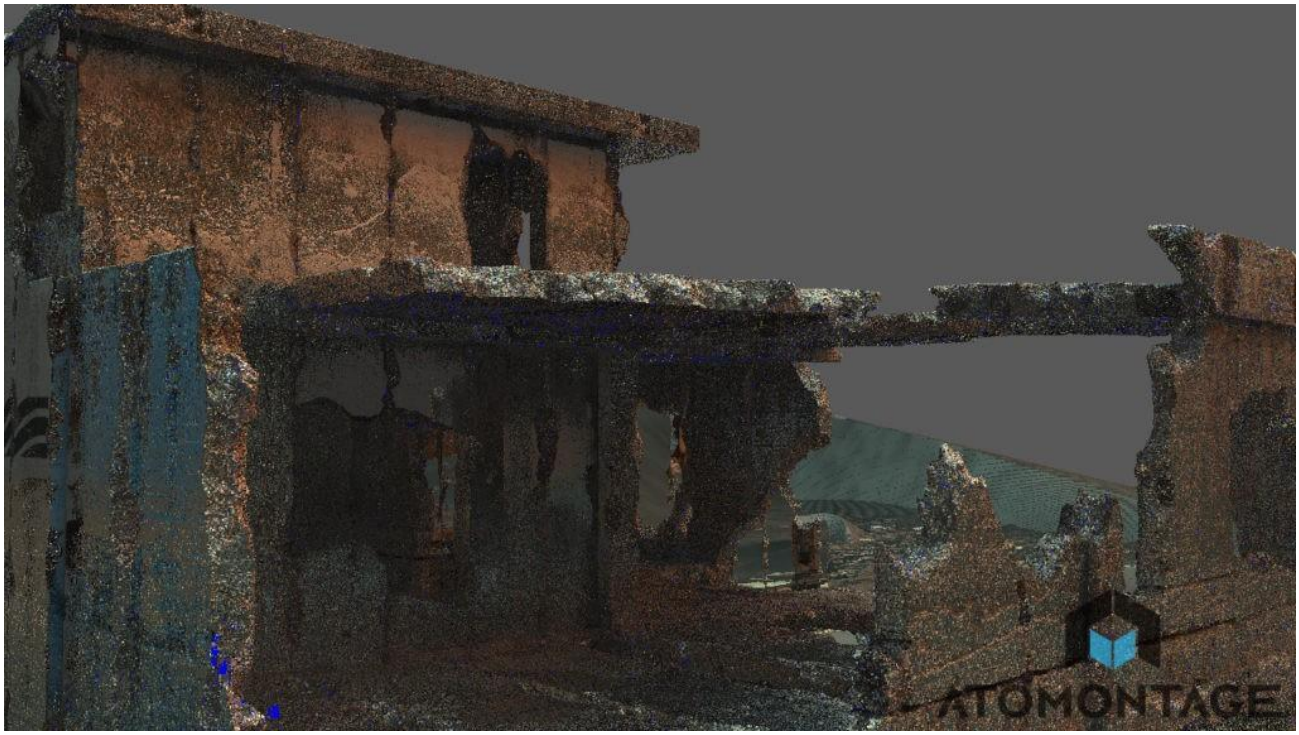

Рис. *12 Воксельне представлення будівлі*

Структури даних вокселів полегшують такі операції, як виявлення зіткнень та модифікація середовища. Здатність обробляти напівпрозорі об’єкти допомагає моделювати аморфні явища, такі як хмари, туман та вогонь [1].

Воксельна графіка добре працює з дуже деталізованими об’єктами, такими як людське тіло. Приклад моделі на рисунку 13 візуалізує людське обличчя, включаючи майже усі деталі, з точністю до 0,5 мм. Роздільна здатність моделі обмежена лише пристроєм зйомки, а не технологією. Площа пам'яті та швидкість візуалізації залежать від розміру набору даних, а не від його складності. Це є основною відмінністю від моделей багатокутників, в яких складність моделі впливає на кількість полігонів і текстуру та сильно впливає на розмір пам'яті. Як зазначалося у розділі 10 «Об’ємної графіки» (Чен, Кауфман та Ягель), подання багатокутників часто вимагають більше пам’яті, ніж воксельні, що змушує зменшити кількість полігонів, а отже, і сприйнятливу якість зображень.

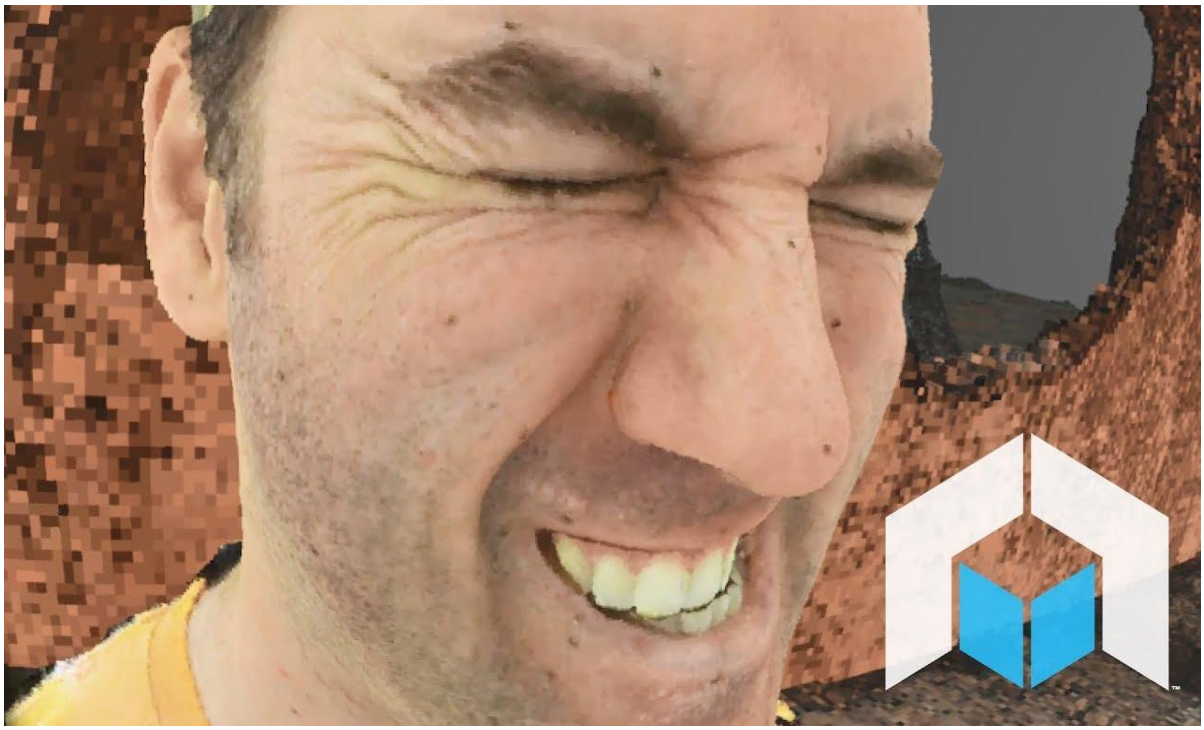

Рис. *13 Воксельне деталізоване представлення обличчя людини*

Як і у випадку з пікселями, самі по собі вокселі не містять інформації про свої координати в просторі. Їх координати обчислюються з їх позиції в тривимірній матриці - структурі, що моделює об'ємний об'єкт або поле значень параметра в тривимірному просторі.

Цим вокселі відрізняються від об'єктів векторної графіки, для яких відомі координати їх опорних точок (вершин) і інші параметри.

Воксельні моделі мають певну роздільну здатність. Кожен воксел має певне значення, наприклад, колір.

Для зберігання воксельної моделі застосовують масив розмірами X × Y × Z. Стиснені воксельні моделі (в порівнянні з векторними) споживають набагато більше місця в пам'яті для обробки. Наприклад, одна незжата модель розміром 256 × 256 × 256 вокселей буде займати пам'ять об'ємом від 32 МБ (256 * 256 * 256 = 16777216 вокселей і як мінімум 2 байта на воксель навіть в 256-ти градаціях сірого, так як до них треба додати 256 градацій прозорості, разом 16777216 * 2 = 33554432 байт = 33554432/1024 = 32768 КБ =

32768/1024 = 32 МБ), в той час як для векторної моделі може знадобитися в десятки або навіть в сотні разів менше об’єму оперативної пам’яті.

Однією з новітніх перспективних технологій, що дозволяє робити ефективну деталізацію воксельних об'єктів, є воксельне октодерево (sparse voxel octree). До переваг такої структури даних можна віднести наступне:

- значна економія пам'яті;
- природна генерація рівнів деталізації;
- висока швидкість обробки рейкастингу.

Перший вузол дерева - корінь - є кубом, що містить весь об'єкт цілком. Кожен вузол або має 8 кубів-нащадків, або не має ніяких нащадків. В результаті всіх розбиттів виходить регулярна тривимірна сітка вокселей, наведена на рисунку 14.

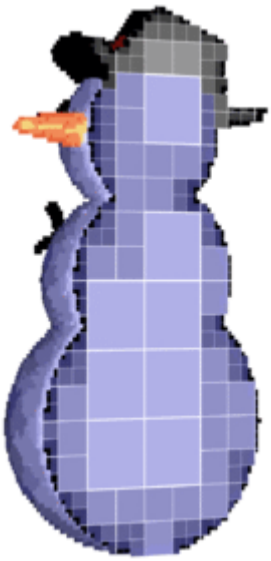

Рис. *14 Побудова воксельного октодерева*

Для воксельних моделей існує безліч алгоритмів візуалізації. Один з найшвидших способів називається «киданням сніжків» (англ. splatting). Вокселі «кидаються» на поверхню перегляду в порядку віддаленості від неї, від далеких до близьких. Утворені «сліди від сніжок» (сплети) рендеряться як диски, колір і прозорість яких змінюється в залежності від діаметра відповідно до нормального (гаусового) розподілу. У різних реалізаціях можуть використовуватися інші елементи або ж інші розподіли.

Для поліпшення якості зображення використовуються більш складні алгоритми відображення: алгоритм Marching cubes та інші. Алгоритм «Marching Cubes» (біжать кубики) будує ізоповерхні, спираючись на дані вокселів. Звичайна реалізація алгоритму використовує значення 8-и сусідніх вокселів, щоб намалювати полігон всередині куба утвореного їх координатами. Оскільки існує всього 256 можливих комбінацій, можна заздалегідь їх підготувати, і використовувати типові «цеглинки» (уже в екранних координатах) для малювання великих обсягів даних в хорошій якості.

Існують і інші алгоритми, наприклад проекція максимальної інтенсивності, яка добре відображає положення в тривимірному просторі найбільш яскравих ділянок тривимірного об'єкту.

1.7 Процес руйнування (Runtime Destruction)

Після завершення фази підготовки геометрії настає час зробити систему більш динамічною. Адже у реальному світі руйнування не відбувається лише тільки одним розбиттям на фрагменти, присутні і інші елементи, такі як динаміка шматочків (розліт, падіння або подальше розщеплення), зв’язки між фрагментами можуть порушуватися, а можуть і залишатися незмінними. Отож процес руйнування — саме той етап, який дозволяє надати розрахованим раніше фрагментам деяких фізичних аспектів. Відслідковується правило — чим реалістичніше відбувається руйнування при найменших затратах ресурсів комп’ютера, тим краще побудована ця система.

Цей етап включає в себе такі методи: заскриптована анімація, булеві операції реального часу, метод скінченних елементів, метод ламких зв’язків, метод складеного твердого тіла.

1.7.1 Заскриптована анімація

Іноді набагато дешевше та краще після взаємодії гравця з об’єктом, після чого останній повинен зруйнуватися, просто підмінити цю модель на її зруйновану версію. Такий спосіб називається заскриптованою анімацією, бо руйнування відбувається тільки завдяки саме анімації підміни незруйнованого об’єкта на зруйнований таким чином, щоб гравець не помітив такого перетворення.

До переваг цього методу слід віднести його обчислювальну дешевизну, високу ступінь контролю над анімацією та високу швидкість візуалізації.

До недоліків даного підходу можна віднести те, що об’єкт завжди анімується за одними правилами и руйнування незмінне. Це досить швидко може набриднути гравцеві і змусити його взагалі не грати в дану гру.

1.7.2 Булеві операції реального часу

Булеві операції роздивлені в етапі підготовки геометрії можуть також застосовуватися в режимі реального часу. Даний підхід на відміну від звичайних булевих операцій не вимагає препідготовки.

Принцип роботи дещо відрізняється: коли в об’єкт потрапляє сила, достатньо велика щоб заподіяти шкоду моделі, форма цієї сили формується процедурно і, виконуючи функцію віднімання, із об’єкту видаляється результат цього самого віднімання.

Однією з переваг булевих операцій реального часу є можливість створення не тільки руйнування об’єкту, але і процедурне моделювання вм’ятин та пошкоджень практично на будь-якому об’єкті, що знаходиться на сцені.

1.7.3 Метод скінченних елементів

Метод скінченних елементів — це чисельний метод вирішування диференційних рівнянь с частковими похідними, а також інтегральних рівнянь що виникають при вирішенні завдань прикладної фізики. На даний момент МСЕ застосовується для вирішування достатньої кількості задач, а саме:

- Стаціонарні задачі розповсюдження тепла, розподілення електричного поля, інші задачі теорії поля;
- Задачі механіки та міцності, в підзадачі яких входять проектування різних просторових оболонок (авто, літак, гелікоптер тощо).

На рисунку 15 наведений приклад візуалізації деформації авто при ударі.

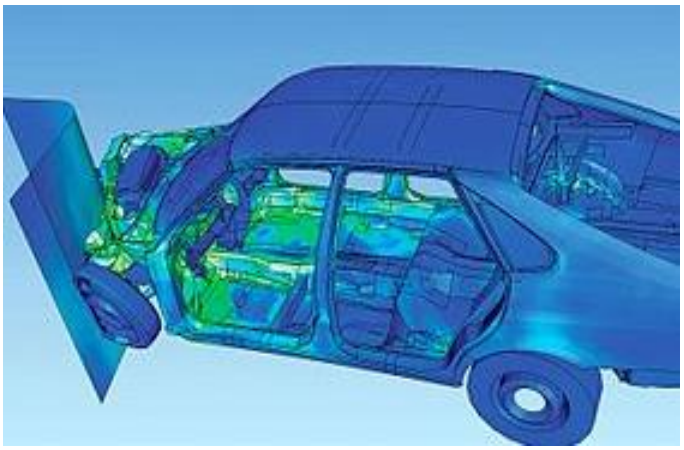

Рис. *15 Візуалізації деформації авто при ударі за допомогою методу скінченних елементів*

Загалом, основною ідеєю МСЕ є те, що будь-яку безперервну величину можна апроксимувати дискретною моделлю, яка будується на множині кусочно-безперервних функцій, визначених на скінченної кількості елементів.

Приблизний алгоритм дій може виглядати так і вварюється від поставленої задачі:

1. В даній області фіксується скінченна кількість точок. Такі точки називають вузлами.

2. Значення безперервної величини в кожному такому вузлі вважається змінною, котра повинна бути знайдена.

3. Область визначення безперервної величини розділюється на скінченну кількість підобластей, котрі називаються скінченними елементами.

4. Безперервна величина апроксимується на кожному СЕ поліномом, який визначається за допомогою значень у вузлах.

Таким чином, використання повністю математичної моделі МСЕ робить використання методу дуже розповсюдженим у симуляціях процесу руйнування та деформації не тільки твердих тіл, але і таких матеріалів, як метал, дерево або скло.

Принцип роботи МСЕ у системі руйнування може полягати у визначенні вузлів, як, наприклад, положеннями фрагментів у просторі ігрової сце-

ни, а значення що повинне бути знайдене — наступне положення шматочку в залежності від прикладених сил до початкового цільного об’єкту.

Наведені переваги роблять метод скінченних елементів одним із найкращих методів для системи руйнування.

1.7.4 Метод ламких зв’язків

Після розбиття цільного 3D об’єкту на фрагменти для надання динаміки отриманій системі частинок можна застосувати так званий метод ламких зв’язків. В даному методі модель, що буде руйнуватися та яка вже складається з менших фрагментів, містить в собі властивості твердого тіла для кожного такого фрагменту. Тверде тіло, або rigid body, дозволяють об’єктам на сцені взаємодіяти за допомогою фізики: сили обертання, тертя, гравітація або взаємодія з іншими об’єктами. Тому цьому методі тверді тіла пов’язані один з одним за допомогою так званих зв’зків.

Суть цього методу полягає в тому, що при дії будь-якої сили на один із фрагментів, ця сила передається на сусідні, зв’язані з ним фрагменти, змушуючи групу зв’язаних твердих тіл діяти як одне ціле. Якщо сила, що подіяла на зв’язок більше ніж деяке встановлене порогове значення, то цей зв’язок розривається.

До недоліків такого підходу можна віднести неоднозначну поведінку цих самих зв’язків. Із-за введення порогового значення, вони можуть бути недостатньо жорсткими і об’єкт, замість руйнування може генерувати інші стани, такі як деформація або навіть симулювати поведінку м’якого еластичного тіла.

1.7.5 Метод складеного твердого тіла

Метод складеного твердого тіла як і метод ламких зв’язків також заснований не використанні зв’язків між фрагментами. Під час виконання обчислюється імпульс і поширюється через об’єкт. Якщо імпульс перевищує

деякий поріг, встановлені звязки або послаблюються, або розриваються. Дуже схоже на метод ламких зв’язків.

Відмінність цього підходу засновується на тому, що при розриванні зв’язку один фрагмент залишається одним фрагментом, тобто не об’єднується в кластери з іншими сусідами.

Ще однією відмінністю є те, що властивості твердого тіла (rigid body) надаються не усім фрагментам одразу, а лише тим, на які подіяла зовнішня сила та в той час, коли ця сама сила подіяла.

1.8 Аналіз існуючих програмних систем руйнування

На сьогоднішній день будь яку гру, в якій можна руйнувати та перебудовувати навколишній світ, можна назвати достойним програмним продуктом, що реалізує механіку руйнування.

Однак, за період з першої появи 3D гри по сьогодні, наявні такі рішення, які були дуже прогресивними для свого часу і залишаються такими і по сьогодні.

1.8.1 Движок Geo-Mod

Однією із перших ігор, яка використала можливості руйнування навколишнього ігрового світу стала гра під назвою Red Faction та її наступник Red Faction 2. Обидві гри використовували спеціальний движок Geo-Mod. Geo-Mod — движок, розроблений Volition Inc. Для серії цих ігор.

Основну функцію цього движка можна побачити вже з назви — Geometry Modification, або модифікація геометрії. Він довзоляв змінювати геометрію рівня за допомогою великого арсеналу зброї. Це надавало гравцеві можливості робити отвори та проходи в поверхнях. Це було досягнуто унікальним на той час методом: замість того, щоб просто замінити об’єкт пошкодженим, Geo-Mod розраховував новий об’єкт з наближеної до реальності фо-

рмою та розміром пошкодження. Результат такої роботи движка можна побачити на рисунку 16.

Рис. *16 Модифікація геометрії у движку Geo-Mod*

Однак це мало певні наслідки: наприклад, у Red Faction гравець міг настільки сильно змінити геометрію рівня, що просто гра становилася неможливою і було необхідно завантажуватися з найближчого чекпоінту.

У Red Faction 2 розробники виправили цю проблему, вимкнувши можливість руйнування у найкритичніших частинах рівня.

1.8.2 Движок Geo-Mod 2.0

Якщо движок Geo-Mod дозволяв робити процедурно згенеровані отвори з деякої анімацією вибуху або розпадання на шматки, то движок Geo-Mod 2.0 та його модифікація 2.5 пішли ще далі. Можливість модифікувати геометрію рівня було вимкнено, однак запроваджено нові можливості.

Движок повністю сконцентрував увагу на симуляції фізики обвалу різних будівель (однак ключові об’єкти все одно не можна було зруйнувати). У двох іграх, що використовують движки версії 2.0 та 2.5, Red Faction Guerrilla

та Red Faction Armageddon, величезна кількість будівель та об’єктів, які можна руйнувати в режимі реального часу, тобто процедурним руйнуванням.

Дуже цікавим моментом є той факт, як саме руйнується звичайна будівля за допомогою цього движка. Якщо гравець вирішив зруйнувати трохи нижньої частини будівлі, то верхня частина не буде зруйнованою, поки є незруйновані опори у нижній частині. Тільки при знищенні усіх ключових точок нижнього поверху уся будівля зруйнується. Приклад такого процедурного руйнування будівлі зображено на рисунку 17.

Рис. *17 Руйнування будівлі у движку Geo-Mod 2.0*

1.8.3 Движок Unreal Engine 4

Unreal Engine — ігровий движок, що розроблюються та підтримується компанією Epic Games. З моменту появи цього ігрового движка він був орієнтований на створення шутерів від першої особи. Однак подальший його розвиток дозволили розробникам робити ігри різних жанрів та напрямків.

У контексті розробки системи руйнування Unreal Engine може запропонувати плагіни, котрі були розроблені ентузіастами. Однак з почтаку 2020

року UE4 має власну систему проектування кінематографічних руйнувань Chaos.

Chaos — це нова високопродуктивна система фізики і руйнування Unreal Engine. З Chaos Destruction користувачі можуть в режимі реального часу досягати кінематографічної якості в сценах з масштабними руйнуваннями та безпрецедентним контролем над створенням контенту.

В UE4 є власний редактор фрактуризації, де можна вибрати різні методи фрагментування цільного об’єкту, задати поведінку цих об’єктів, задати ключові точки, де в цільній моделі не буде виконуватися руйнування, по суті статична область. Приклад роботи системи Chaos можна побачити на рисунку 18.

Рис. *18 Кінематографічне руйнування на движку Unreal Engine 4*

1.8.4 Движок Digital Molecular Matter

Digital Molecular Matter — це фізичний движок, розроблений швейцарською компанією Pixelux Entertainment у 2008 році. Движок має дві версії. Перша версія була розроблена для використатання в комп'ютерних іграх та інших системах, які повинні працювати в режимі реального часу. Призначенням другої версії було використання його у симуляціях. Був представлений також у вигляді плагінів для 3ds Max та Maya.

Обидві версії движка наділені високою якістю фізичних симуляцій, тому що точність та наближеність до реальних фізичних процесів була одним із ключових концепцій при розробці DMM.

Основною особливістю DMM у порівнянні з іншими фізичними движками реального часу — це використання методу скінченних елементів, що дозволяє DMM симулювати фізичні властивості з дуже високою точністю.

DMM дозволяє візуалізувати руйнування різних об'єктів кожний раз по-різному, в залежності від впливу. На характер руйнування у DMM впливають декілька факторів, такі як сила впливу, напрям та його область. На рисунку 19 зображено використання DMM у грі Star Wars: The Force Unleashed.

Рис. *19 Руйнування у движку DMM*

1.8.5 Voxel based рішення

Завдяки тому, що матриця зберігає значення вокселів для кожного одиничного елементу простору, воксельні моделі підходять для моделювання безперервних середовищ і масивів даних, в той час як векторні більш призначені для моделювання дискретних об’єктів.

Тому воксельна графіка знайшла своє застосування, в першу чергу, в медицині. Ряд медичних пристроїв, як, наприклад, сканери комп'ютерної томографії, тривимірне УЗД, МРТ видають пошарову інформацію при скануванні. По завершенні сканування будується воксельна модель. Значення вокселів в цьому випадку відображають дані з пристрою. У комп'ютерній томографії, наприклад, це прозорість тіла за шкалою Хаунсфілда, тобто прозорість для рентгенівських променів.

Для воксельних моделей (наприклад, медичних даних зі сканера МРТ) просто реалізується висновок будь-якого перетину моделі. Це дає можливість вивчити будь-який зріз даних, чого було б складно добитися за допомогою полігональної графіки.

Також широкого застосування вокселі знайшли у 3D друку. Одним із популярних рішень у цьому випадку виступає GrabCAD Voxel Print, що дозволяє користувачеві точно контролювати друк на рівні вокселя для створення нових структур, текстур, кольорів тощо. Розблоковуючи маніпуляції з концентраціями матеріалів та структурами на рівні вокселів, GrabCAD Voxel Print надає можливість створювати абсолютно нові цифрові матеріали для певних спеціалізованих потреб.

У 3D іграх вокселі використовуються досить давно, проте їх використання обмежено через серйозні вимоги до апаратної частини. Найбільш розповсюдженим способом використання вокселей у 3D іграх є створення моделей. Менш розповсюджений випадок — створення ландшафтів та терейнів, що базуються на вокселях.

Найбільш відомі представники воксельних ігор:

- Вангери, де розробниками використовувались великі багаторівневі безперервні воксельні простори з так званим «живим» ландшафтом, який змінювався в залежності від дій гравця та штучного інтелекту;
- Minecraft використовую подобу воксельної графіки для створення випадкового ландшафту, що генерується, однак слід зазначити, що рендеринг відбувається за допомогою полігонів;
- Cube World — гра використовує воксельну графіку для створення випадкового ігрового світу, моделей гравців та NPC;
- Teardown — гра з можливістю тотального руйнування навколишнього світу, побудованого тільки з вокселів.

Як було описано вище, вокселі більш ресурсозалежні аніж полігони. Тож розмістити звичайним способом декілька тисяч вокселів на сцені, наприклад в ігровому движку Unity, не має великого сенсу, бо продуктивність такої системи буде занадто низькою. Однак Unity Engine має механізм, а точніше стек технологій, який допоможе наблизити продуктивність воксельної графіки до бажаного результату. Цей стек технологій називається Unity DOTS. Новий високопродуктивний стек інформаційно-орієнтованих технологій Unity (DOTS) дозволяє використовувати сучасні багатоядерні процесори на повну потужність. DOTS допомагає розробляти насичені оточення, прискорює розробку за допомогою зручного в роботі універсального коду на C #.

1.9 Результати аналізу існуючих рішень

Розвиток систем руйнувань в іграх проходить доволі цікавий шлях розвитку: від звичайної анімації підміни цільного об’єкту на зруйнований до точних розрахунків поведінки за допомогою математичних методів. За допомогою таких розвинутих систем користувачу надається свій власний світ для експериментів, які неможливо було б виконати, використовуючи тільки заскриптовані події.

Також аналіз існуючих рішень показує, що такі системи, при правильній їх реалізації, не потребують дуже великих комп’ютерних ресурсів при достатньо реалістичній візуалізації руйнувань.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ЗАСОБІВ КОМП'ЮТЕРНОГО МОДЕЛЮВАННЯ РУЙНУВАННЯ

2.1 Методи підготовки геометрії 3D об'єктів

Основним і першим етапом у створенні системи руйнування виступає підготовка геометрії, або Geometry Preparation. Суть алгоритмів і методів полягає у тому, щоб цільний об'єкт, для якого потрібно змоделювати руйнування, спочатку повинен розбитися на фрагменти. Для задачі створення системи руйнування реального часу, або процедурне руйнування, існує низка досить популярних алгоритмів та підходів, які використовуються для поділу 3D моделі на фрагменти. До цих алгоритмів ставляться певні обмеження та вимоги. Основні три вимоги це:

- Алгоритм повинен розбивати 3D модель на фрагменти за мінімальний проміжок часу, такий, що не зменшує якість розбиття та не навантажує комп'ютерну систему;
- Метод повинен повинен моделювати реалістичне розбиття 3D об'єкту на основі полігональної сітки. До основної задачі такої вимоги відноситься заповнення полігомами прогалин, які залишилися після розбиття. *Полігональна сітка* — або «меш» (від анг. *Mesh*) — це сукупність вершин, ребер та граней, котрі визначають форму об'єкта в комп'ютерній графіці та моделюванні. Гранями в меші зазвичай виступають полігони — трикутники, чотирикутники або інші прості опуклі багатокутники. Саме тому метод повинен реконструювати внутрішню частину об'єкту, коли розбиває (фрагментує) модель.

2.1.1 Теорія діаграм Вороного

Діаграми Вороного названі на честь Георгія Феодосевича Вороного, який визначив загальні n-мірні діаграми в 1908 р.

Діаграма Вороного — це розподіл простору на регіони, визначені точками, які задані заздалегідь. Ці точки називають сайтами або насінням. У межах регіону всі точки є ближче до сайту регіону, ніж до будь-якого іншого регіону [11]. На рисунку 1 чорні крапки є точками сайтів, а кольорові ділянки — клітинами Вороного відповідного сайту.

Найчастіше діаграми Вороного використовують евклідову відстань між точками, але можливі й інші показники відстані, такі як Манхетенська відстань.

Евклідова відстань між точками p і q є довжина відрізка, що поєднує їх.

У декартовій системі координат, якщо $p = (p_1, p_2, ..., p_n)$ і $q = (q_1, q_2, ..., q_n)$ дві точки в Евклідовому n-просторі, то відстань **d** від p до q, або від q до p задається наступною формулою:

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

Положення точки в Евклідовому n-просторі є Евклідовим вектором. Таким чином, p і q, можуть бути представлені у вигляді евклідових векторів, починаючи від початку координат простору (початкова точка) з кінцевими точками, що закінчуються в двох точках. Евклідова норма, або евклідова довжина вимірює довжину вектора:

$$\|p\| = \sqrt{p_1^2 + p_2^2 + \cdots + p_n^2} = \sqrt{p * p}$$

Манхетенська відстань, або відстань міських кварталів — метрика, введена Германом Мінковським. Відповідно до цієї метрики, відстань між двома точками дорівнює сумі модулів різниць їх координат. Назва «Манхеттенська відстань» пов'язана з вуличним плануванням Манхеттена.

Манхетенська відстань d_1 між двома векторами $\boldsymbol{p}$ та $\boldsymbol{q}$ в n-вимірному матеріальному векторному просторі із заданою системою координат — сума довжин проекцій відрізка між точками на осі координат. Манхетенська відстань можна задати наступною формулою:

$$d_1(p,q) = \|p-q\|_1 = \sum_{i=1}^{n} |p_i - q_i|,$$

де $p = (p_1, p_2, ..., p_n)$ і $q = (q_1, q_2, ..., q_n)$.

З евклідовою відстанню краї області є перпендикулярними бісектрисами сусідніх ділянок. У випадку побудови системи процедурного руйнування буде використовуватися Евклідова дистанційна метрика для побудови діаграм Вороного. На рисунку 20 також наведено вигляд діаграми Вороного з використанням Евклідової відстані. На рисунку 21 можна побачити, що використання Манхетенської відстані для побудови діаграми Вороного для фрагментизації 3D об’єкту не підходить через занадто нереалістичний вигляд цих самих фрагментів.

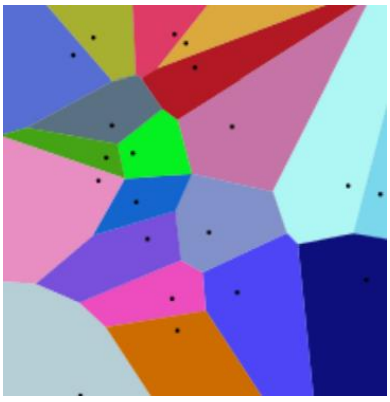

Рис. 20. *Вигляд діаграми Вороного з використанням Евклідової відстані*

Рис. 21. *Діаграма Вороного з використанням Манхетенської відстані*

Більш формально і загалом регіон Ворного R_k, визначений точкою p_k, є набором усіх точок у просторі, відстань яких до p_k не більше їх відстані до p_j, де j — будь-який індекс, відмінний від k. Це відповідає наступному рівнянню:

$$Rk = \{x \in X \mid d(x, p_k) \leq d(x, p_j) \text{ for all } j \neq k\}$$

2.1.2 Алгоритми побудови діаграм Вороного

Першим кроком у розбитті на фрагменти цільного 3D об’єкту у системі руйнування в режимі реального часу за допомогою діаграм Вороного — пошук відповідного, оптимального алгоритму. У цій науковій роботі будуть пояснені три відомі алгоритми для обчислення діаграм Вороного. Кожен з алгоритмів буде проаналізований, щоб перевірити, чи підходить він для обчислення в реальному часі. Найважливішою властивістю є можливість додавання нових вершин до діаграми без необхідності перерахунку всього графу. Для простоти пояснень наступні алгоритми пояснюються в контексті 2D, але їх властивості легко переносяться в 3D, оскільки кінцевий результат в 3D.

Алгоритм перебору

Метод перебору є найпростішим методом в контексті його реалізації. До переваг такого алгоритму слід віднести його простоту, коли реалізація на ранніх етапах та основною ціллю являться досягнення результату, а не продуктивність.

Основним поняттям в алгоритмі перебору являється перпендикулярна бісектриса. Перпендикулярна бісектриса двох точок на площині (A, B) — це лінія, яка перетинає пряму, що утворена точками A та B, під прямим кутом. Розрахувати перпендикулярну бісектрису по двом точкам $A(x_1,y_2)$ та $B(x_2,y_2)$ можна за наступною формулою:

$$y = -\frac{x_2 - x_1}{y_2 - y_1}x + y_1 + y_2 + \frac{(x_2 - x_1)(x_1 + x_2)}{2(y_2 - y_1)}$$

Алгоритм перебору складається з декількох кроків, а саме:

1. Вибрати опорну точку A зі списку точок;
2. Обчислити відстань від A до кожної іншої точки;
3. Вибрати точку, найближчу до A. Цю точку можна назвати B;
4. Знайти перпендикулярну бісектрису між точками A та B;
5. Видалити усі точки на іншій стороні бісектриси, які розташовані далі аніж точка B;
6. Видалити точку B;
7. Повторювати кроки 3-6, поки ще є точки у списку;
8. Знайти точки перетину бісектрис;
9. З’єднати точки перетину за годинниковою або проти годинникової стрілкою, щоб створити багатокутник.

На рисунку 22 наведено графічне представлення кроків алгоритму перебору для побудови діаграм Вороного, яка є більш наглядною.

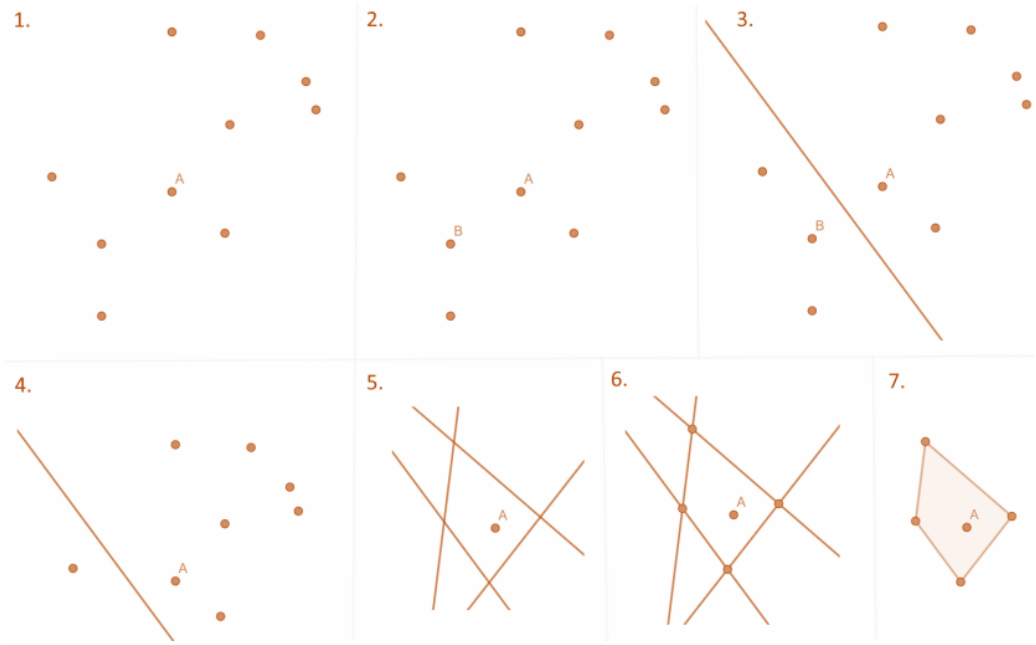

Рис. *22 Графічне представлення алгоритму перебору для побудови діаграми Вороного*

За допомогою даного алгоритму знаходяться лінії, що утворюють ребра комірки Вороного. У кроці 5 видаляються усі точки з іншого боку бісектриси тому що вони точно знаходяться далі від найближчої точки і не можуть розділити ребро з даною точкою. Слід зазначити, що більшість бісектрис будуть перетинатися з іншими бісектрисами, утворюючи замкнутий цикл. Однак винятками слід вважати точки на ребрах діаграми. Ці ребра будуть обрізані межею області.

Недоліком алгоритму перебору для побудови діаграм Вороного є низька продуктивність на останніх ітераціях, коли кількість вершин виміряється тисячами. Комірка Вороного потенційно може мати N-1 сторін, де N — загальна кількість сайтів. Із цього витікає той факт, що часова складність алгоритму для обчислення кожної комірки дорівнює $O(N^2)$. Таким чином, загальна часова складність для побудови всієї діаграми становить $O(N^3)$.

Ще одним недоліком даного алгоритму є неможливість додавати нові точки без чергового перерозрахунку усіх вершин, що впливає на час роботи алгоритму.

Алгоритм Форчуна

Оскільки алгоритм перебору занадто повільний для великої кількості точок, потрібен більш швидкий алгоритм. Одним із способів вирішення існуючих недоліків алгоритму перебору є використання алгоритму Форчуна. Алгоритм Форчуна — це алгоритм замітаючої прямої для генерування діаграми Вороного. Він був опублікований Стівеном Форчуном у 1968 р. у своїй роботі «Алгоритм лінійної розгортки для діаграм Вороного» [12].

Алгоритм Форчуна — алгоритм лінійної розгортки, який має $O(n\log n)$ час і $O(n)$ просторову складність для створення діаграми Вороного на площині [10, 13]. Алгоритм Форчуна використовує замітаючу лінію для вирішення завдань в евклідовому просторі. Замітаюча пряма — це вертикальна пряма, яка рухається зліва направо. Замітаюча лінія рухається через площину і зупиняється щоразу, коли буде досягнута бажана точка. Операції обмежуються об'єктами, які перетинаються або знаходяться поблизу замітаючої лінії. Замітаюча лінія обробляє кожну точку по одному і після завершення її руху формується діаграма Вороного.

Крім того, алгоритм використовує берегову лінію, показану на рисунку 23, яка складається з парабол, які визначаються вхідними точками над замітаючою лінією та самою береговою лінією. Берегова лінія не є прямою, а є складною, що складається зі шматочків парабол, кусочно-заданою кривою зліва від замітаючої прямої. Вона відокремлює порцію площині, всередині якої діаграма Вороного може бути відома, незалежно від інших точок праворуч від замітаючої прямої. Для кожної точки зліва від замітаючої прямої можна визначити параболу для точки, яка рівновіддалена як від цієї точки, так і від замітаючої прямої. Берегова лінія — це межа об'єднань цих парабол. По мірі руху прямої, вершини берегової лінії, в яких дві параболи перетинаються, викреслюють ребра діаграми Вороного. Берегова лінія просувається, зберігаючи основу кожної параболи в точності на половині шляху між початковим станом замітаючої прямої і новою позицією замітаючої прямої. Матема-

тично це означає, що кожна парабола утворюється за допомогою замітаючої прямої як директриси, а задана точка з набору служить фокусом.

Як було зазначено, діаграма Вороного над (зліва) береговою лінією відома, незалежно від точок, які знаходяться нижче (зправа) замітаючої лінії, тож за створенням діаграм Вороного можна проспостерігати більш ітеративно и послідовно, аніж з використанням алгоритму перебору.

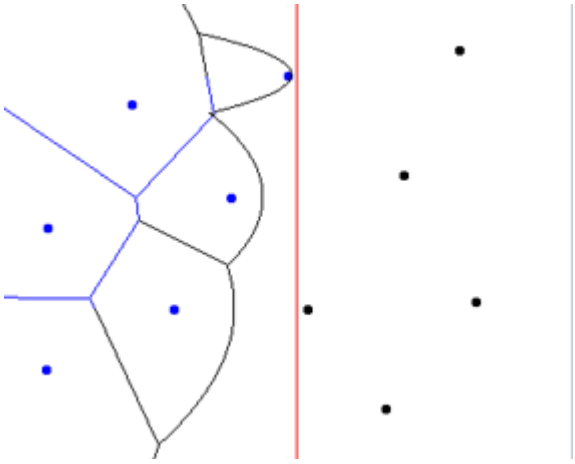

Рис. *23 Замітаюча (червона) та берегова (чорна) лінія в алгоритмі Форчуна*

Слід зазначити, що алгоритм підтримує структуру даних двійкового дерева, що описує комбінаторну структуру берегової лінії, і черга з пріоритетом, в якій перераховується потенційні події в майбутньому, які могли б змінити структуру берегової лінії. Ці події включають додавання іншої параболи в берегову лінію (коли замітаюча пряма проходить через іншу вхідну точку) і видалення параболи з берегової лінії (коли замітаюча пряма стає дотичною до окружності через деякі три вхідних точки, параболи яких утворюють послідовні сегменти берегової лінії). Остання називається подією кола і графічний її приклад наведено на рисунку 24. Кожній такій події може бути присвоєно пріоритет по x-координаті заметаючої прямої в точці, де подія відбулася.

Тож алгоритм складається з послідовного видалення події з черги з пріоритетом, знаходження змін подій в береговій лінії і оновлення структури даних.

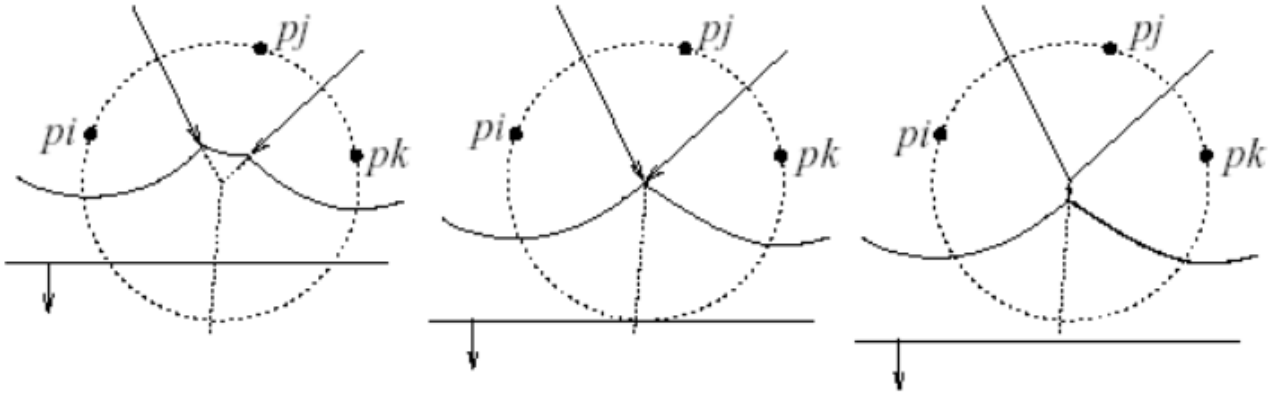

Рис. *24 Утворення серединної параболи та її зхлопування до точки, яка належить діаграмі Вороного*

Алгоритм Боуєра-Ватсона

Алгоритм Форчуна набагато швидше аніж метод перебору і це є його дуже вагомою перевагою. Але відсутність можливості згодом додавати нові вершини робить його поганим вибором для розрахунку знищення в реальному часі. Маючи трохи більший розрахунковий час, цю здатність можна отримати з алгоритму Боуєра-Ватсона. Алгоритм Бойєра-Уотсона, також відомий як алгоритм Боуєра або алгоритм Ватсона, був розроблений незалежно один від одного Адріаном Бойєром (1981) та Девідом Ватсоном (1981) [15]. Цей алгоритм також може бути використаний для обчислення тріангуляції Делоне, про яку поговориться пізніше. За потреби діаграму Вороного можна легко отримати з неї.

В загальному вигляді алгоритм Боуєра-Ватсона працює наступним чином:

1. Створюється трикутник, який містить усі точки, які потрібно обробити.

2. Додається точка до тріангуляції.
3. Знаходяться усі трикутники, у окружності яких є нова точка.
4. Видалення знайдених трикутників.
5. Під’єднання вершин, отвору, що залишився, до нової точки.
6. Повторення кроків 2-6, доки всі точки не будуть оброблені.
7. Видалення усіх трикутників, пов’язаних із вхідним трикутником.

Для початку алгоритму Бойєра-Ватсона потрібна вже існуюча тріангуляція Делоне. Найпростішою дійсною тріангуляцією Делоне є трикутник. Ось чому великий трикутник додано в крок 1. Цей трикутник називається надтрикутником. Як тільки алгоритм закінчиться, це може бути видалено.

Етапи 2-5 повторно тріангулюють частину діаграми, куди була додана точка, щоб зробити її дійсною. Знову тріангуляція Делоне. Умовою є те, що жодна точка не може бути всередині будь-якого трикутника обведене коло.

Крок 7 видаляє трикутники, які поділяють вершину із супер-трикутником. Це очищує графік і залишає лише тріангуляцію точок. Цей крок можна проігнорувати, якщо початковий трикутник не заважає обчислювати, або відомо, що нові вершини будуть додані пізніше. Якщо нові точки додаються до існуючої триангуляції, супертрикутник не буде потрібний, але це потрібно при додаванні точок поза вже триангульованою областю.

У всіх сценаріях алгоритм приймає O (N^2) час [14], де N - кількість точок, але з різними оптимізаціями складність часу можна покращити. Використовуючи зв’язаність трикутників, можна досягти часової складності O ($N\log N$). Це означає, що замість перевіряючи всі трикутники, виконується прогулянка від трикутника у напрямку до нової точки [15].

Додаткової часової ефективності можна досягти, попередньо обчисливши описані кола. Для додатків реального часу додаткові структури даних, такі як quadtree, або дерево квадрантів, можуть бути використані для подальшого вдосконалення час пошуку.

Як висновок, цей алгоритм легко зрозуміти, а оскільки це додатковий алгоритм, він має можливість динамічно змінюватись щоразу, коли додається

нова точка без необхідності кожен раз перераховувати всю діаграму. Розширення Бойєра-Ватсона до вищих розмірів не вимагає великих змін. У 3D трикутники замінюються на тетраедри, ребра з поверхнями, більше нічого не змінюється. А за допомогою декількох методів оптимізації можна досягти більшої ефективності, аніж алгоритм Форчуна.

2.1.3 Тріангуляція

Процес розбиття полігональної області зі складною конфігурацією в набір трикутників називається тріангуляцією. При аналізі або синтезі складних поверхонь їх апроксимують сіткою трикутників, і згодом оперують з найпростішими полігональними областями, тобто з кожним з трикутників. Тріангуляція називається оптимальною, якщо сума довжин всіх ребер мінімальна серед всіх можливих тріангуляцій, побудованих на тих самих вихідних точках. Складність цієї задачі складає *O(NlogN)*. Також відомі алгоритми, що дозволяють в деяких випадках досягнути в середньому *O(N)*. Приклад тріангуляції наведений на рисунку 25.

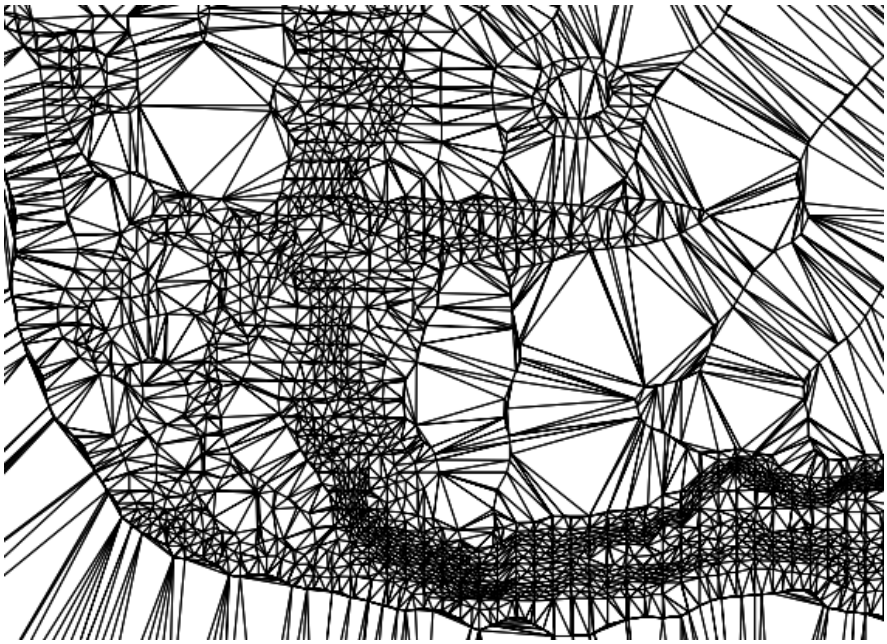

Рис. *25 Приклад тріангуляції*

Задача побудови тріангуляції являється NP-повною. Тому для більшості реальних задач існуючі алгоритми побудови оптимальної тріангуляції неприйнятні через занадто високу складність. При необхідності на практиці застосовують наближені алгоритми. Одним із перших був застосован жадібний алгоритм побудови тріангуляції. Його кроки:

1. Генерується список всіх можливих відрізків, що з'єднують пари вихідних точок, і цей список сортується за довжинами відрізків.

2. Починаючи з самого короткого, послідовно виконується вставка відрізків в тріангуляцію. Якщо відрізок не перетинається з іншими раніше вставленими відрізками, то він вставляється, інакше він не розглядається.

Складність роботи жадібного алгоритму при деяких його покращеннях складає O(N2logN)[17].

2.1.4 Тріангуляція Делоне

Крім оптимальної і жадібної тріангуляції, також широко відома тріангуляція Делоне, що володіє рядом практично важливих властивостей. Вперше задача побудови тріангуляції Делоне була поставлена в 1934 р. в роботі відомого радянського математика Б.Н.Делоне [16]. Тріангуляція називається тріангуляція Делоне, якщо вона являється опуклою та якщо всередину кола, описаного навколо будь-якого побудованого трикутника, не потрапляє жодна з заданих точок тріангуляції [18]. Виконання умови описаного кола наведена на рисунку 26.

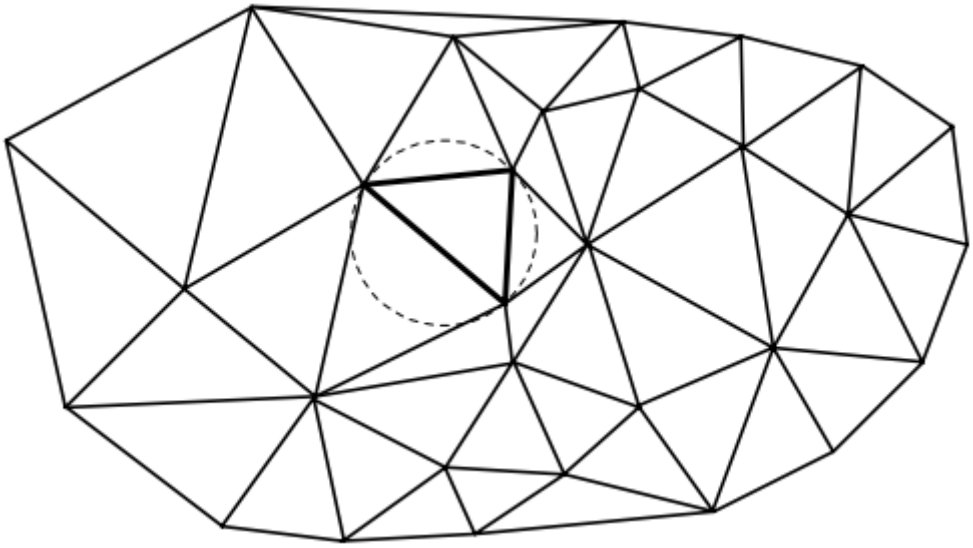

Рис. *26 Виконання умови тріангуляції Делоне*

Дивлячись на тріангуляцію Делоне, видно, що пара сусідніх трикутників тріангуляції задовольняє умові Делоне, якщо цій умові задовольняє тріангуляція, складена тільки з цих двох трикутників. Також трикутник тріангуляції задовольняє умові Делоне, якщо цій умові задовольняє тріангуляція, складена тільки з цього трикутника і трьох його сусідів (якщо вони існують).

Більшість алгоритмів побудови тріангуляції Делоне використовують твердження, що тріангуляцію Делоне можна отримати з будь-якої іншої тріангуляції по тій же системі точок, послідовно перебудовуючи пари сусідніх трикутників ΔABC і ΔBCD, які задовольняють умові Делоне, в пари трикутників ΔABD і ΔACD. Така операція перебудови також часто називається флі-пом [19] та наведена на рисунку 27.

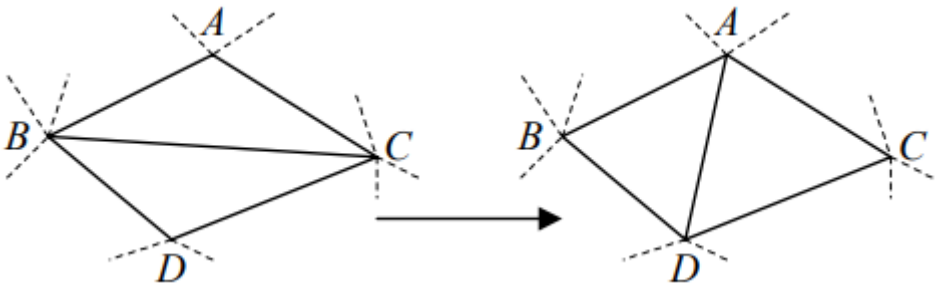

Рис. *27 Перебудова трикутників під тріангуляцію Делоне*

В тріангуляції можна виділити 3 основні види об'єктів: вузли (точки, вершини), ребра (відрізки) і трикутники.

У роботі багатьох існуючих алгоритмів побудови тріангуляції Делоне і алгоритмів її аналізу часто виникають такі операції з об'єктами тріангуляції [20]:

1. Трикутник → вузли: отримання для даного трикутника координат, що утворюють його вузли.

2. Трикутник → ребра: отримання для даного трикутника списку, що утворюють його ребра.

3. Трикутник → трикутники: отримання для даного трикутника списку сусідніх з ним трикутників.

4. Ребро → вузли: отримання для даного ребра координат, що утворюють його вузли.

5. Ребро → трикутники: отримання для даного ребра списку сусідніх з ним трикутників.

6. Вузол → ребра: отримання для даного вузла списку суміжних ребер.

7. Вузол → трикутники: отримання для даного вузла списку суміжних трикутників.

Основні структури тріангуляції Делоне

Основними структурами даних при роботі з тріангуляцію Делоне є:

1. Вузли з сусідами;
2. Подвійні ребра;
3. Вузли та трикутники;
4. Вузли, ребра та трикутники;
5. Вузли, прості ребра та трикутники.

Вузли з сусідами — для кожного вузла тріангуляції зберігаються його координати на площині та список вказівників на сусідні вузли, з якими є спільні ребра [21]. Приклад такої структури наведений на рисунку 28.

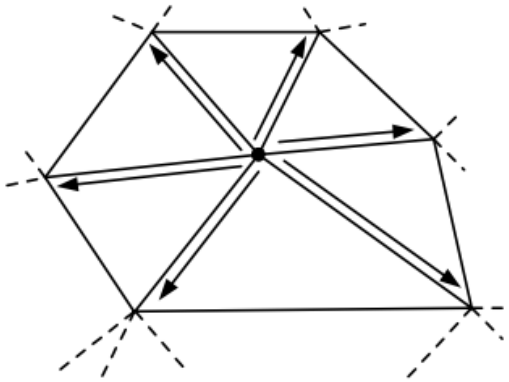

Рис. *28 Вузли структури Вузли з сусідами*

По суті, список сусідів визначає в неявному вигляді ребра тріангуляції. Трикутники ж при цьому не представляються взагалі, що є зазвичай суттєвою перешкодою для подальшого застосування тріангуляції. Крім того, недоліком є змінний розмір структури вузла, що часто приводить до неекономної витрати оперативної пам'яті при побудові тріангуляції.

У структурі Подвійні ребра основою тріангуляції є список орієнтованих ребер. При цьому кожне ребро входить в структуру тріангуляцію двічі, але спрямованими в протилежні сторони.

Для кожного ребра зберігаються такі вказівники [22]:

1. на кінцевий вузол ребра;

2. на наступне за годинниковою стрілкою ребро в трикутнику, що знаходиться праворуч від даного ребра;

3. на «ребро-близнюк», що з'єднує ті ж самі вузли тріангуляції, що і дане, але спрямоване в протилежну сторону;

4. на трикутник, що знаходиться праворуч від ребра.

Приклад структури подвійні ребра наведено на рисунку 29.

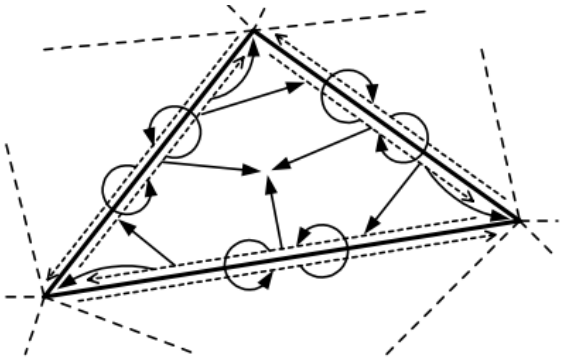

Рис. *29 Ребра в структурі подвійні ребра*

Недоліками даної структури є представлення трикутників в неявному вигляді, а також велика витрата пам'яті, що становить при 8-байтовому представленні координат і 4-байтових вказівниках не менше 64N байт (не враховуючи витрати пам'яті на подання додаткових даних в трикутниках).

У структурі вузли та трикутники для кожного трикутника зберігаються три вказівника на утворюючі його вузли і три вказівника на суміжні трикутники [23], що наведено на рисунку 30.

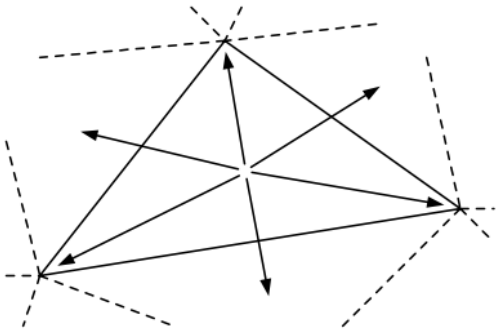

Рис. *30 Трикутники в структурі вузли та трикутники*

Нумерація точок і сусідніх трикутників проводиться в порядку обходу за годинниковою стрілкою, при цьому навпроти точки з номером $i \in \{1, 2, 3\}$ розташовується ребро, відповідне сусідньому трикутнику з таким же номером.

Ребра в даній тріангуляції в явному вигляді не зберігаються. При необхідності ж вони зазвичай подаються як вказівник на трикутник і номер ребра усередині нього.

При 8-байтовому поданні координат і 4-байтових вказівниках дана структура тріангуляції вимагає приблизно 64N байт.

Незважаючи на те, що дана структура поступається «Вузлам з сусідами», вона найбільш часто застосовується на практиці в силу своєї відносної простоти і зручності програмування алгоритмів на її основі.

В структурі «Вузли, ребра та трикутники» в явному вигляді задаються всі об’єкти тріангуляції: вузли, ребра та трикутники. Для кожного ребра зберігаються вказівники на два кінцевих вузла і два сусідніх трикутника. Для трикутників зберігаються вказівники на три ребра, що утворюють трикутник [24].

Дана структура часто застосовується на практиці, особливо в задачах, де потрібно в явному вигляді представляти ребра тріангуляції.

Недоліком цієї структури є велика витрата пам'яті, що становить при 8-байтовому поданні координат і 4-байтових вказівниках приблизно 88N байт. На рисунку 31 наведено приклад структури Вузли, ребра та трикутники.

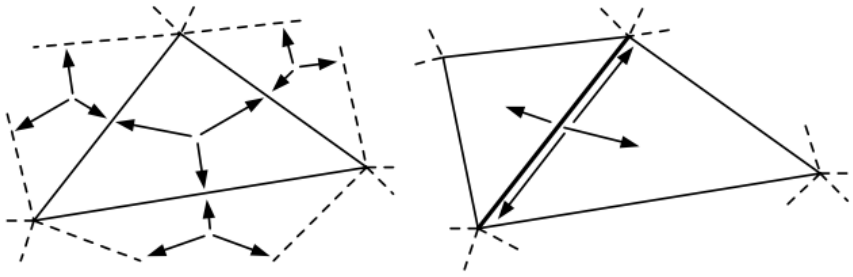

Рис. *31 Трикутники, ребра та вузли в структурі Вузли, ребра та трикутники*

У структурі «Вузли, прості ребра і трикутники» в явному вигляді задаються всі об'єкти тріангуляції: вузли, ребра і трикутники. Для кожного ребра

зберігаються вказівники на два кінцевих вузла і два сусідніх трикутника [24]. Для ребер ніякої спеціальної інформації немає. Для трикутників зберігаються вказівники на три вузла і три ребра, а також покажчики на три суміжних трикутника, що наведено на рисунку 32.

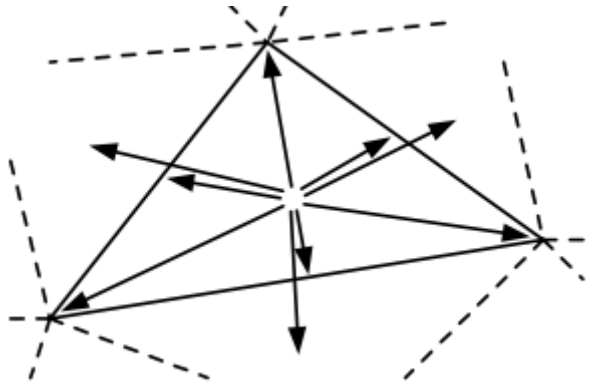

Рис. *32 Структура вузли, прості ребра та трикутники*

Дана структура часто застосовується на практиці, особливо в задачах, де потрібно в явному вигляді представляти ребра тріангуляції.

Недоліком цієї структури є відносно велика витрата пам'яті, що становить при 8-байтовому поданні координат і 4 байтових вказівниках приблизно 80N байт.

Основні алгоритми тріангуляції Делоне

В даний час відомо значна кількість різних алгоритмів побудови тріангуляції Делоне. У таблиці 2 зібрані основні характеристики всіх цих алгоритмів [25].

Таблиця *2*

Основні характеристики алгоритмів тріангуляції Делоне

Назва алгоритму	**Складність в найгіршому випадку**	**Середня складність**
Простий ітеративний алго-	$O\ (N^2)$	$O\ (N^{3/2})$

ритм		
Ітеративний алгоритм «Видаляй та будуй»	$O\ (N^2)$	$O\ (N^{3/2})$
Алгоритм з індексуванням пошуку R-деревом	$O\ (N^2)$	$O\ (N \log N)$
Алгоритм з індексуванням пошуку k-D-деревом	$O\ (N^2)$	$O\ (N \log N)$
Алгоритм з індексуванням пошуку квадродеревом	$O\ (N^2)$	$O\ (N \log N)$
Алгоритм статичного кешування	$O\ (N^2)$	$O\ (N^{9/8})$
Алгоритм динамічного кешування	$O\ (N^2)$	$O\ (N)$
Алгоритм з смуговим розбиттям точок	$O\ (N^2)$	$O\ (N)$
Алгоритм з квадратним розбиттям точок	$O\ (N^2)$	$O\ (N)$
Алгоритм пошарового згущення	$O\ (N^2)$	$O\ (N)$
Алгоритм з сортуванням точок уздовж фрактальної кривої	$O\ (N^2)$	$O\ (N)$
Алгоритм з сортуванням точок по Z-коду	$O\ (N^2)$	$O\ (N)$
Алгоритм «Розділяй і володарюй»	$O\ (N \log N)$	$O\ (N \log N)$
Рекурсивний алгоритм з розрізанням по діаметру	$O\ (N \log N)$	$O\ (N \log N)$
Алгоритм опуклого смуго-	$O\ (N^2)$	$O\ (N \log N)$

вого злиття		
Алгоритм неопуклого смугового злиття	$O(N^2)$	$O(N \log N)$
Покроковий алгоритм	$O(N^2)$	$O(N^2)$
Алгоритм лінійного замітання	$O(N^2)$	$O(N)$
Віяловий алгоритм	$O(N^2)$	$O(N)$
Алгоритм рекурсивного розщеплення	$O(N \log N)$	$O(N \log N)$
Стрічковий алгоритм	$O(N^2)$	$O(N)$

В цілому із всіх алгоритмів найкращими являються *алгоритм динамічного кешування* та *алгоритм пошарового згущення* в силу своєї простоти реалізації та складності.

В алгоритмі тріангуляції з динамічним кешуванням пошуку необхідно завести кеш мінімального розміру, наприклад 2x2. По мірі зростання числа доданих в тріангуляцію точок необхідно послідовно збільшувати його розмір в 4 рази (в 2 рази по обох осях координат), переписуючи при цьому інформацію зі старого кешу в новий. При цьому для збільшення кешу треба виконати наступні пересилки (h - старий кеш, h '- $\forall i,j=\overline{0,\ m-1}:\ h'_{2i,2j},h'_{2i,2j+1},h'_{2i+1,2j},h'_{2i+1,2j+1} := h_{i,j}$ новий): . Даний алгоритм кешування дозволяє однаково ефективно працювати на маленькій і великій кількості точок, заздалегідь не знаючи їх кількості. Для більшості випадкових розподілів вихідних точок даний алгоритм працює значно швидше за всі інші алгоритми [25].

В ітеративному алгоритмі тріангуляції с пошаровим згущенням [26] необхідно розбити площину, на якій містяться точки на $n = (2^u + 1) * (2^v + 1)$ комірок-квадратів однакового розміру. Кожен квадрат відмічається від 0 до 2^u по горизонталі і від 0 до 2^v по вертикалі. На наступному кроці вводиться поняття шару, де точка належить шару *i*, якщо обидва номерів її квадрату

кратні 2^i. По значенням пар номерів всі точки шару *i* поділяються на 4 підмножини:

1. кутові точки;
2. внутрішні точки;
3. X-граничні точки;
4. Y-граничні точки.

Додавання точок тріангуляції відбувається пошарово, починаючи з шару з максимальним номером до нульового. Всередині кожного шару повинно спочатку внести всі точки другого типу, потім третього и насамкінець четвертого типу.

Для довільних наборів точок такий алгоритм, як і всі інші ітеративні, мають трудомісткість $O(N^2)$. Однак, якщо вихідні точки розподілені рівномірно, то складність алгоритму буде наближеною до $O(N)$ [26].

Недоліком більшості ітеративних алгоритмів є те, що при вставці кожної нової точки повинно перераховувати всю тріангуляцію. Однак, в алгоритмі зі згущенням точок вдається позбутися від таких ситуацій, рівномірно та послідовно вставляючи в тріангуляцію вузли. Завдяки цьому алгоритм будує менше вузьких трикутників і саме завдяки цьому швидше виконується на реальних даних, аніж інші алгоритми.

2.1.5. Зв'язок тріангуляції Делоне та діаграми Вороного

Цікавим фактом є те, що тріангуляція Делоне вперше з'явилась в науковому світі як граф, двоїстий діаграмі Вороного — однієї із основних структур обчислювальної геометрії. Саме тому для будь-якою заданої точки $P_i \in \{P_1, \dots, P_N\}$ на площині коміркою Вороного називається геометричне місце точок на площині, які знаходяться до Pi ближче, ніж до будь-якої іншої заданої точки Pj, j ≠I [27]. Сукупність багатокутників Вороного утворює розбиття площини, представляє векторну мережу.

Однією із головних особливостей діаграми Вороного, як говорилось раніше, є її двоїстість тріангуляції Делоне [16]. Основною ознакою є те, що,

з’єднав відрізками ті вихідні точки, многокутники Вороного яких стикаються принаймні кутами, ми отримуємо тріангуляцію Делоне. Приклад такої двоїстої задачі наведено на рисунку 33.

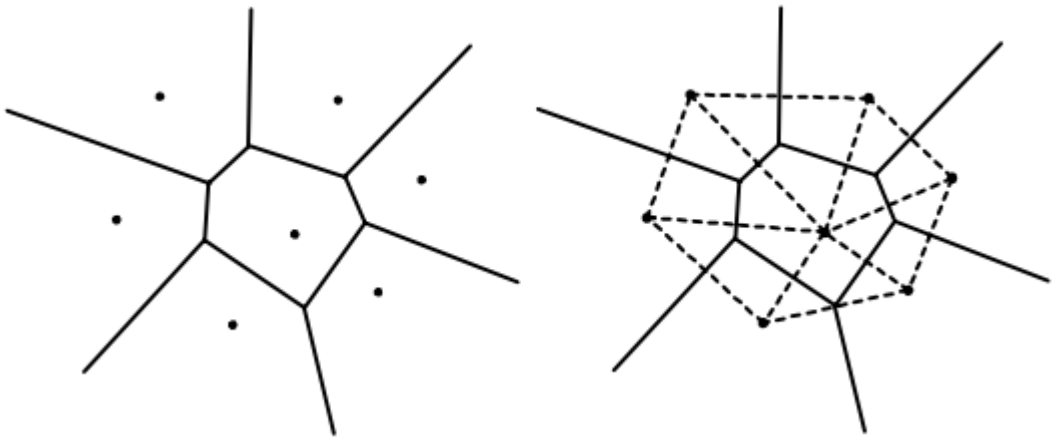

Рис. *33* Діаграма Вороного та двоїста їй тріангуляція Делоне

2.2 Методи realtime руйнування

По завершенні етапу підготовки геометрії потрібно зробити елементи розбитої системи такими, що вони взаємодіють один із одним та з навколишнім світом: розліт, падіння тощо. Зв’язки між фрагментами можуть порушуватися, а можуть і залишатися незмінними. Саме тому процес руйнування – етап, який дозволяє надати розрахованим раніше фрагментам деяких фізичних властивостей. Однією із основних умов є найбільш реалістичне протікання процесу руйнування при найменших затратах ресурсів комп’ютера.

Із багатьох раніше описаних методів основними являються : метод ламких зв’язків та метод складеного твердого тіла, що за своєю суттю являються доволі схожими методами.

Обидва методи засновані на реальній фізиці реальних об’єктів, такі як гравітація, імпульс тощо. Також обидва методи націлені на порушення зв’язків або, як вони ще називаються, обмежень, що притаманні щойно розбитим елементам. Обмеження – це, по суті, правила, які повинні виконуватися під час моделювання, такі як «відстань між цими двома частками не повинно бути більше 2» або «ці дві точки на цій парі твердих тіл повинні завжди збігатися». Іншими словами, обмеження прибирають з модельованого об’єкту деякі ступені свободи твердого тіла. На кожному етапі моделювання ми можемо обчислити коригувальні сили або імпульси, які при застосуванні до тіл будуть зближувати або віддаляти їх (рух буде обмежено), а правила, що накладаються обмеженнями, залишаться виконаними.

На практиці обмеження визначається з точки зору функції поведінки або функції обмеження F, яка приймає стан пари тіл як параметри (наприклад, положення та орієнтацію) і на виході видає скалярне число. Коли значення цієї функції знаходиться в допустимому діапазоні, обмеження виконується. Таким чином, на кожному етапі моделювання повинна прикладатися сила або імпульси до твердих тіл, щоб спробувати утримати значення F у певному діапазоні.

Після розбиття цільного 3D об’єкту на фрагменти для надання динаміки отриманій системі частинок можна застосувати так званий метод ламких зв’язків. В даному методі об’єкт, що буде руйнуватися та який складається з частинок, отриманих за допомогою одного або декількох алгоритмів фрактуризації (підготовки геометрії), містить в собі властивості твердого тіла (rigid body) для кожного такого фрагменту. Тверде тіло, або rigid body, дозволяють об’єктам на сцені взаємодіяти за допомогою фізики: сили обертання, гравітація або взаємодія з іншими об’єктами, такі як зіткнення, зсуви тощо. У зв’язку з цим у цьому методі тверді тіла, що представляють собою частинки цільного об’єкту, пов’язані один з одним за допомогою так званих зв’язків або обмежень.

Суть такого підходу полягає в тому, що при наданні будь-якої сили на один із отриманих фрагментів, ця сила передається на сусідні, зв’язані з ним фрагменти, змушуючи вже отриману групу зв’язаних твердих тіл діяти як одне ціле або окремо. Інакше кажучи один фрагмент (або група фрагментів передають отриманих зовні імпульс усім іншим фрагментам даної системи. Імпульс - це векторна фізична величина, яка являється добутком маси тіла на його швидкість. Слід зазначити, що направлення імпульсу співпадає з направленням вектору швидкості. Якщо сила, що подіяла на зв’язок більше ніж деяке встановлене порогове значення, то цей зв’язок розривається.

До недоліків такого підходу можна віднести непередбачувану в деяких випадках поведінку цих самих обмежень. Із-за введення порогового значення, вони можуть бути недостатньо жорсткими і об’єкт, замість руйнування може генерувати інші стани, такі як деформація або навіть симулювати поведінку м’якого еластичного тіла. Або навпаки, об’єкт не зруйнується загалом, що дуже суперечить поведінці системи руйнування.

Метод складеного твердого тіла як і метод ламких зв’язків також заснований не використанні зв’язків між фрагментами, отриманих на етапі фрактуризації твердого тіла. Під час виконання обчислюється імпульс і поширюється через об’єкт. Якщо імпульс перевищує деяке порогове значення, встановлені зв’язки або розриваються, або послаблюються. Дуже схоже на метод ламких зв’язків з тією відмінністю, що цей метод засновується на тому, що при розриванні зв’язку один фрагмент залишається одним фрагментом, тобто не об’єднується в групи з іншими сусідами-фрагментами.

Ще однією відмінністю є те, що властивості твердого тіла (rigid body) надаються не усім фрагментам одразу, а лише тим, на які подіяла зовнішня сила та в той час, коли ця сама сила подіяла.

РОЗДІЛ 3 ПРОЕКТ ПРОГРАМНОЇ СИСТЕМИ КОМП'ЮТЕРНОГО МОДЕЛЮВАННЯ РУЙНУВАННЯ

3.1 Архітектура системи

Програмна система моделювання руйнування у комп'ютерних іграх складається з класів, діаграма яких наведена на рисунку 34.

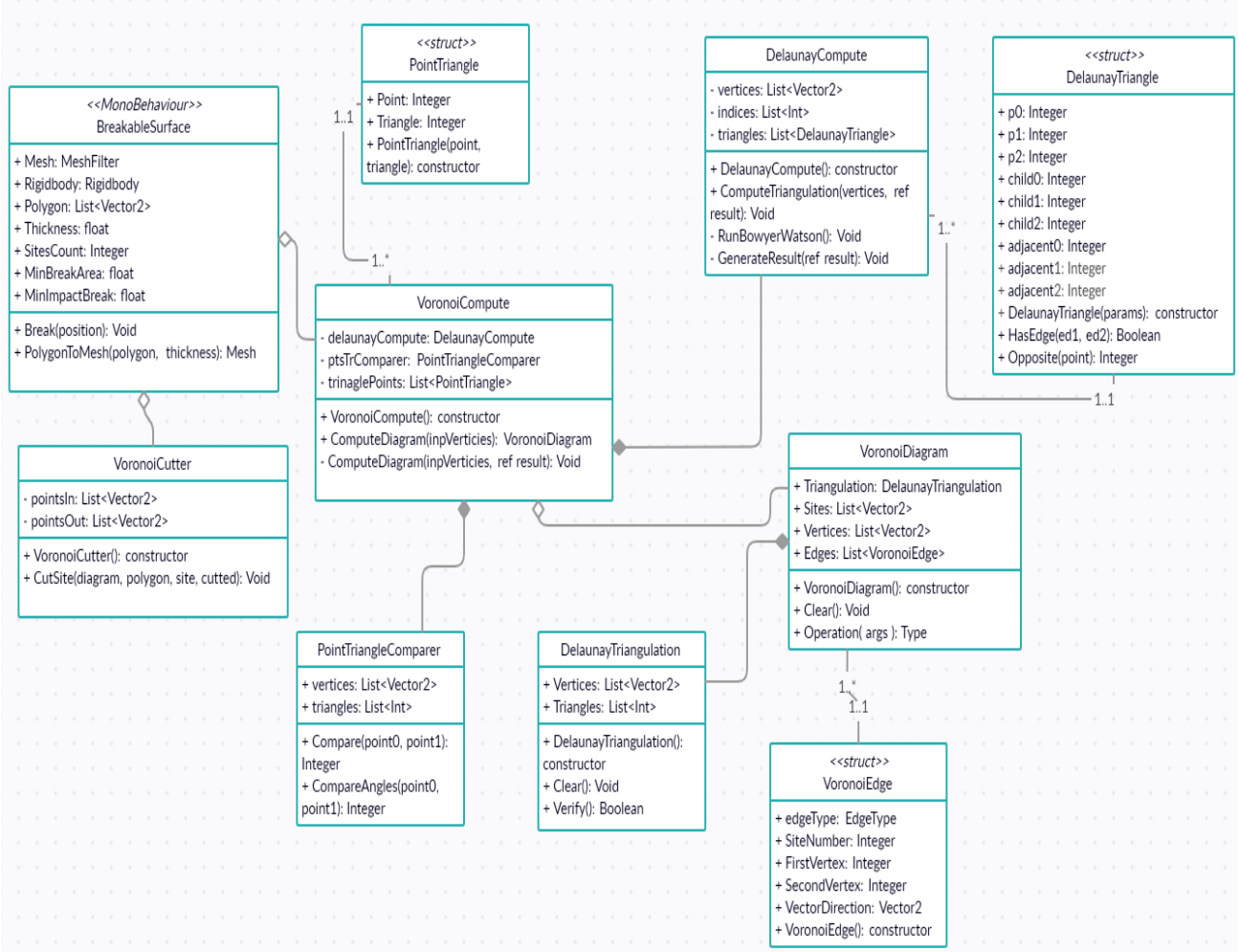

Рис. *34 Архітектура системи*

Головним класом гри є клас "BreakableSurface". Він містить в собі метод для детектування зіткнення з об'єктом, що може зруйнувати поверхню, та метод, що руйнує цю саму поверхню.

Клас “VoronoiCompute” призначений для розрахунку діаграми Вороного, яка заснована на тріангуляції Делоне, яка, в свою чергу, розраховується за допомогою класу “DelaunayTriangulation” та “DelaunayCompute”.

Застосунок написаний на мові програмування C# з використанням компонентно-орієнтованого підходу, що надає ігровий движок Unity.

Компонентний підхід програмування

У світі програмування поняття компонентів та роз'єднання йдуть рука об руку. Компонент можна сприймати як меншу частину більшої машини. Кожен компонент має свою конкретну роботу, і в цілому (і оптимально) може виконати своє завдання або мету без допомоги будь-яких зовнішніх джерел. Крім того, компоненти рідко належать одній машині, і їх можна поєднувати з різними системами для виконання свого конкретного завдання, але досягати різних результатів, коли справа доходить до загальної системи. Це пов’язано з тим, що компоненти не тільки не цікавляться цією ще більшою системою, але й навіть не знають, що вона існує.

Класичний приклад компонентів — це частки автомобіля. Не тільки весь автомобіль сам по собі є компонентом, але кожен окремий його аспект є компонентом.

3.2 Засоби реалізації

Під час розробки програмної системи, було вирішено використовувати наступні засоби:

1. Ігровий движок Unity2019;
2. Середовище розробки Visual Studio 2019;
3. Мова програмування C# та фреймворк .NET;
4. Universal Render Pipeline, а саме різновид Light Weight Render Pipeline або LWRP;
5. Android SDK.

3.2.1 Ігровий движок Unity 2019

Unity3D — це "екосистема розробки ігор", вона включає середовище для розробки інтерактивного 2D та 3D вмісту, включаючи механізм візуалізації та фізики, інтерфейс сценаріїв для програмування інтерактивного вмісту , експортер вмісту для багатьох платформ (настільних, веб-, мобільних) та зростаючого співтовариства, що ділиться знаннями.

Unity3D поставляється в двох варіантах — безкоштовна версія, яка і використовувалася в проекті, і про-версія, що коштує 1500 доларів.

Основні елементами ігрового движка Unity являються система координат, користувацький інтерфейс (так званий Editor), базові компоненти фізики та освітлення.

Система координат

Система координат в Unity складається з трьох координат: X, Y та Z. В системі Unity координати поділяють на локальні та глобальні.

Локальні координати ще називають координатами об’єкту, бо вони відносяться лише до координат змодельованого об’єкту, який був отриманий у будь-якому 3D редакторі, наприклад Blender.

Глобальні координати, або world space координати — це положення об’єкту відносно інших об’єктів. World space — це простір, в якому живуть ваші об’єкти. Вони мають 3D-координату. Світовий простір не має обмежень щодо свого розміру. Він тягнеться від (-infinity, - infinity, - infinity) до (infinity, infinity, infinity) [28]. Об'єкти та камера можуть знаходитись де завгодно в цьому просторі. Розмір одиниці виміру у світовому просторі довільний. Приклад локальних та глобальних координат в ігровому движку Unity наведено на рисунку 35.

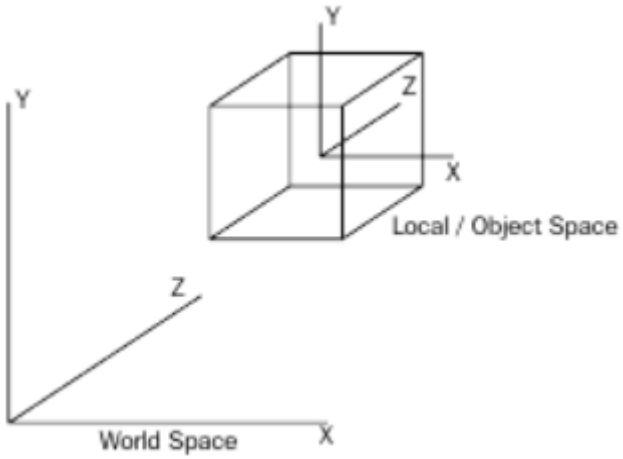

Рис. *35* Локальні та глобальні координати Unity3D

Базові концепції користувацького інтерфейсу

Користувацький інтерфейс Unity 2019 представляє собою зручну оболонку над движком, завдяки якій можливе управління компонентами, на якій заснована, в основному, розробка на Unity. Користувацький інтерфейс в Unity3D називається едітор, приклад якого наведено на рисунку 36.

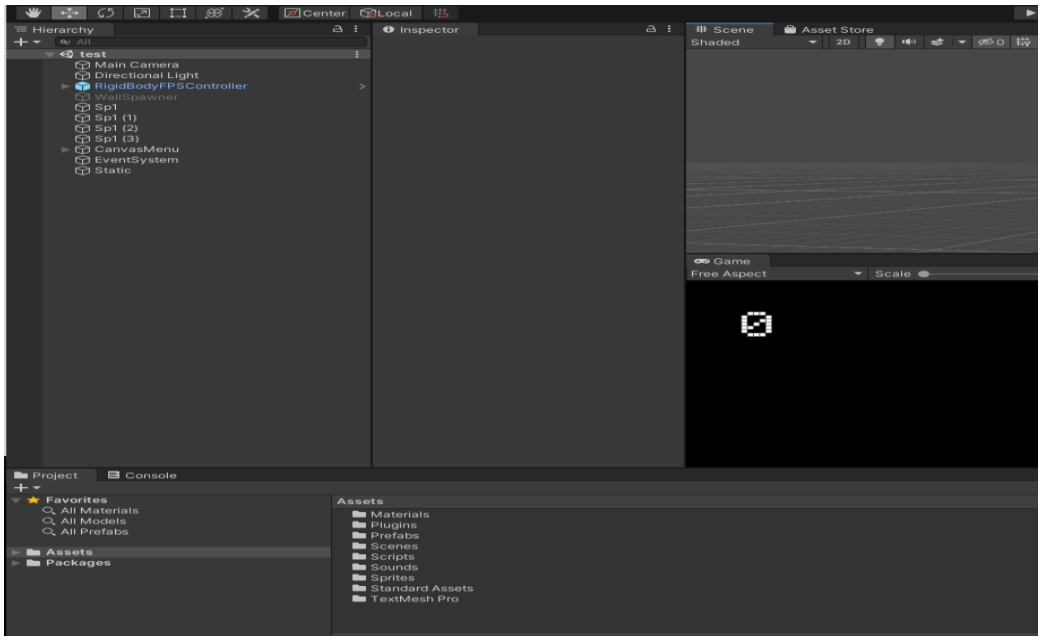

Рис. *36* Editor в Unity3D

Едітор, наведений на рисунку 36, складається з наступних елементів:

1. **Scene**: В цій частині розміщені будь-які візуальні ресурси у своєму середовищі Unity. Він буде оновлюватися в режимі реального часу під час попереднього перегляду гри. Зверху праворуч розташовується маніпулятор; це дозволяє перемикатися між кількома стандартними поданнями: ізометричне (2D) та перспективне (3D). Хоча це не має великого значення, це дозволяє розглядати сцену з будь-якою точки, що є стандартним способом відображення ігор Unity.

2. **Game**: Коли гра не запущена, ця вкладка відображатиме відображення того, як буде виглядати гра, ігноруючи графічні ефекти, які потрібно обчислити під час виконання, з точки зору основної камери. Під час попереднього перегляду гри взаємодія з грою буду проходити через це вікно. Коли сцена порожня, то це вікно відображається кольором тла.

3. **Hierarchy**: у цьому списку перераховані всі об’єкти на поточній завантаженій сцені та всі дочірні об’єкти, які вони можуть мати. Діти — це об’єкти, які можна вважати підлеглими батьківському об’єкту; куди б не рухався верхній об’єкт, вони будуть слідувати, зберігаючи поточний зсув, який вони мають до цього об’єкта. Це важлива концепція в розробці ігор за допомогою движку Unity.

4. **Project/Assets view**: Це перелік усіх спеціальних ассетів для поточної гри, включаючи графічні ресурси, звук, сценарії, префаби (попередньо зібрані ігрові об’єкти) та багато іншого.

5. **Inspector**: Інспектор дозволяє переглядати та налаштовувати окремі налаштування різних ігрових об'єктів та ассетів, а також коригувати деякі загальні налаштування. Інспектор чутливий до вмісту та змінює свої параметри, виходячи з того, який ігровий об’єкт / ассет вибрано. Це також місце, де можна продемонструвати налаштування проекту та уподобання, вибравши їх у меню Редагувати.

6. **Графічні значки для переміщення сцени та її вмісту.** Рука дозволяє рухатись навколо сцени; в поєднанні з іншими елементами керування

сюжетною камерою, Unity стає дуже простим для навігації (див. нижче). Значок праворуч, схожий на чотири стрілки, дозволяє переміщати вибраний об’єкт навколо. Це називається перетворенням об'єкта. Наступна піктограма дозволяє обертати об'єкт, а остання - рівномірне масштабування об'єкта.

7. **Панель відтворення**. Це дозволяє розпочинати гру, призупиняти та припиняти запуск гри в редакторі Unity. Це найшвидший і найпростіший спосіб тестування та налаштування гри.

3.2.2 Середовище розробки Visual Studio 2019

Разом з інсталяцією Unity 2019 інсталюється, також, середовище розробки Visual Studio 2019 з предінстальованим плагіном для розробки в парі з Unity 2019. Це допомагає швидко писати стандартні методи Unity, такі як Start, Update, OnColissionEnter тощо, тому що, в такому випадку, середовище розробки доповнює назву методу та автоматично заповнює параметри.

Звісно, якщо предінстальована стандартна Visual Studio, можна використовувати її. Однак такий підхід не гарантує правильності написання стандартних Unity методів. Це може привести к тому, що метод, який написано неправильно просто не буде працювати, не видаючи ніяких помилок і відстежити таку проблему буде дуже проблематично.

3.2.3 Мова програмування C# та .NET

C# (вимовляється як "Сі Шарп") — це багатоцільова мова комп'ютерного програмування, придатна для всіх потреб у розробці [29].

Хоча C # походить від мови програмування C, він має такі функції, як збір сміття, які дозволяють новачкам освоїти C# швидше, ніж C або C ++. Подібно до Java, вона об'єктно-орієнтована, постачається з великою бібліотекою класів і підтримує обробку винятків, різні типи поліморфізму та відділення інтерфейсів від реалізацій. Ці функції в поєднанні з його потужними інструментами розробки, підтримкою мультиплатформ та загальними засо-

бами роблять C# хорошим вибором для багатьох типів проектів розробки програмного забезпечення: швидке застосування проекти розвитку, проекти, що реалізуються приватними особами чи великими чи малими командами, інтернет-додатки та проекти, що мають суворі вимоги до надійності. Тестові фреймворки, такі як NUnit, роблять C# придатним для тестової розробки і, отже, гарною мовою для використання з екстремальним програмуванням (XP). Його сильний набір тексту допомагає запобігти багатьом помилкам програмування, які часто зустрічаються в слабо введених мовах.

.NET Framework — це загальне середовище для створення, розгортання та запуску веб-служб, веб-додатків, служб Windows та додатків Windows. .NET Framework містить загальнодоступні бібліотеки класів — такі як ADO.NET, ASP.NET та Windows Forms - для надання розширених стандартних служб, які можуть бути інтегровані в різні комп'ютерні системи [29].

.NET framework — це набір інструментів, розроблений для платформи Windows, щоб дозволити розробнику взаємодіяти з хост-системою або будь-яким зовнішнім об'єктом, будь то інший процес чи інший комп'ютер. Платформа .NET — це особлива реалізація платформи Windows. Інші операційні системи мають власні реалізації через різницю в операційній системі управління введенням-виводом, моделях безпеки та інтерфейсах.

3.2.4 Universal Render Pipeline (URP)

Render pipeline (або конвеєр візуалізації) визначає спосіб відображення об’єктів на сцені у три основні етапи:

1. Culling, або кулінг — у ньому перелічені об'єкти, які потрібно відтворити, бажано ті, які видно камері (фокусний кулінг вибракування) та неприховані іншими об'єктами (оклюзійний кулінг).

2. Render, або рендеринг — це отрисовка цих об’єктів з правильним освітленням і деякими їх властивостями в буфери на основі пікселів.

3. Post processing, або пост-процесінг може бути виконаний з цими буферами, наприклад, застосовуючи градацію кольорів та глибину різкості, для формування кінцевого вихідного кадру, який надсилається на пристрій відображення. Ці операції повторюються багато разів на секунду, залежно від частоти кадрів. Unity підтримує кілька конвеєрів візуалізації і навіть дозволяє легко розширити деякі з них за допомогою мови програмування C#.

До початку 2018 року в Unity був доступний лише один конвеєр рендерингу; він був перейменований на "Вбудований рендеринг" [30]. Цей візуалізатор пропонує вибір між прямим та відкладеним рендерингом.

У режимі прямого рендерингу (multi-pass) усі об'єкти на сцені відображаються один за одним послідовно, потенційно за кілька проходів, залежно від кількості світла, що впливає на кожен об'єкт, отже, вартість рендерингу може різко зрости, коли об'єкти освітлюються кількома джерелами світла. Цей тип візуалізації зазвичай пропонує широкий вибір шейдерів і легко справляється з прозорістю (opaque).

У відкладеному режимі всі (непрозорі) геометрії спочатку відображаються в буфери, які зберігають інформацію про їх матеріали (колір, дзеркальність, гладкість тощо). У подальшому проході кожен піксель затіняється послідовно: час візуалізації буде залежати головним чином від кількості джерел світла, що впливають на кожен піксель. Прозорі об’єкти та певні об’єкти зі складними шейдерами все одно потребуватимуть додаткових переходів вперед. Відкладений рендеринг зазвичай рекомендується застосовувати при роботі зі сценами, що містять багато динамічних джерел світла, наприклад, штучно освітленими інтер’єрами або проектами із поєднанням зовнішнього та внутрішнього освітлення.

На даний момент Unity пропонує два конвеєри візуалізації, розроблені з урахуванням продуктивності та сучасного апаратного забезпечення: Hight-Definition Render Pipeline (HDRP) і Lightweight Render Pipeline (LWRP), що зараз, в поточних версіях Unity називається Universal Render Pipeline.

Universal Render Pipeline, раніше Lightweight Render Pipeline (UniversalRP) — це заздалегідь побудований сценарій для рендерингу, виготовлений Unity. Технологія пропонує графіку, яка масштабована до мобільних платформ, і ви також можете використовувати її для консолей вищого класу та ПК. Ви зможете досягти швидкої візуалізації у високій якості, не потребуючи обчислювальної технології шейдерів. UniversalRP використовує спрощене фізичне освітлення та матеріали.

UniversalRP використовує однопрохідний прямий рендеринг. Використовуйте цей конвеєр, щоб отримати оптимізовану продуктивність у реальному часі на кількох платформах.

UniversalRP підтримується на таких платформах:

- Windows та UWP;
- Mac і iOS;
- Android;
- XBox One;
- PlayStation4;
- Nintendo Switch;
- Усі поточні VR-платформи.

Universal Render Pipeline доступний за двома шаблонами: UniversalRP та UniversalRP-VR. UniversalRP-VR поставляється з попередньо ввімкненими налаштуваннями спеціально для VR. Документація для обох трубопроводів візуалізації однакова. Для будь-яких питань щодо UniversalRP-VR потрібно дивитися документацію UniversalRP.

Вбудовані та спеціальні шейдери Lit не працюють із універсальним конвеєром рендеринга. Натомість UniversalRP має новий набір стандартних шейдерів. Якщо оновити поточний проект до UniversalRP, можна оновити вбудовані шейдери до нових.

Проекти, створені за допомогою UniversalRP, не сумісні з конвеєром рендерингу високої чіткості або вбудованим конвеєром рендеринга Unity.

Перш ніж розпочати розробку, треба вирішити, який конвеєр рендерингу використовувати у своєму проекті.

3.2.5 Android SDK

Для портування гри на мобільні пристрої під управлінням операційної системи Android було використано Android SDK. Набір для розробки програмного забезпечення для Android (SDK) включає повний набір засобів розробки. [31] Сюди входять відлагоджувач, бібліотеки, емулятор на основі QEMU, документація, зразок коду та навчальні посібники. На даний момент підтримувані платформи розробки включають комп'ютери під управлінням Linux (будь-який сучасний настільний дистрибутив Linux), Mac OS X 10.5.8 або новішої версії та Windows 7 або новішої версії. Станом на березень 2015 року SDK недоступний для самого Android, але розробка програмного забезпечення можлива за допомогою спеціалізованих програм для Android.

Покращення SDK для Android поєднуються із загальним розвитком платформи Android. SDK також підтримує старі версії платформи Android на випадок, якщо розробники бажають націлити свої програми на старі пристрої. Інструменти розробки — це завантажувані компоненти, тому після завантаження останньої версії та платформи старі платформи та інструменти також можна завантажити для перевірки сумісності.

3.3 Модулі та алгоритми

Під час розробки програмної системи було розроблено два модулі, що забезпечують стабільну роботи застосунку:

1. Модуль розрахунків руйнування;
2. Модуль геймплею (гри).

Діаграму модулів можна побачити на рисунку 37.

Рис. *37 Модулі програмної системи*

3.3.1 Модуль руйнування

На рисунку 38 зображена схема, на основі якій відбуваються розрахунки діаграми Вороного та тріангуляції Делоне, за допомогою яких потім будуються частки зруйнованого об’єкту.

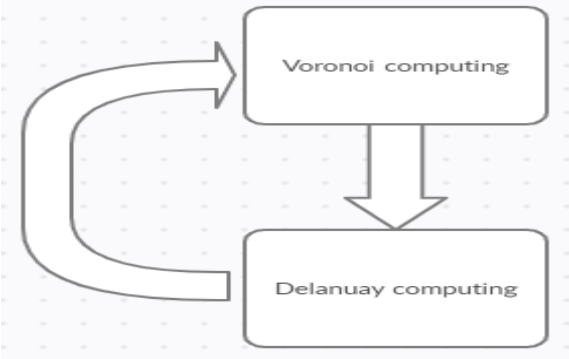

Рис. *38 Модуль руйнування*

Основними алгоритмами модулю руйнування є інкрементний алгоритм побудови діаграми Вороного — алгоритм Боуєра-Ватсона та двоїста задача побудови тріангуляції Делоне за діаграмою Вороного.

В лістингу 1 наведено метод алгоритму Боуєра-Ватсона для довільного набору точок.

Лістинг 1 *Алгоритм Боуєра-Ватсона*

```
void RunBowyerWatson() {
for (int i = 0; i < vertices.Count; i++) {
var pi = i;
if (pi == highest) continue;
     // Index of the containing triangle
          var ti = FindTriangleNode(pi);

          var t = triangles[ti];

               // The points of the containing triangle
in CCW order
          var p0 = t.P0;
          var p1 = t.P1;
          var p2 = t.P2;

          // Indices of the newly created triangles.
          var nti0 = triangles.Count;
          var nti1 = nti0 + 1;
          var nti2 = nti0 + 2;

          // The new triangles! All in CCW order
          var nt0 = new DelaunayTriangle(pi, p0, p1);
          var nt1 = new DelaunayTriangle(pi, p1, p2);
          var nt2 = new DelaunayTriangle(pi, p2, p0);

          nt0.Adjacent0 = t.Adjacent2;
          nt1.Adjacent0 = t.Adjacent0;
          nt2.Adjacent0 = t.Adjacent1;

          nt0.Adjacent1 = nti1;
          nt1.Adjacent1 = nti2;
          nt2.Adjacent1 = nti0;
          nt0.Adjacent2 = nti2;
```

```
                nt1.Adjacent2 = nti0;
                nt2.Adjacent2 = nti1;
                t.Child0 = nti0;
                t.Child1 = nti1;
                t.Child2 = nti2;
                triangles[ti] = t;
                triangles.Add(nt0);
                triangles.Add(nt1);
                triangles.Add(nt2);
                if (nt0.Adjacent0 != -1) LegalizeEdge(nti0,
nt0.Adjacent0, pi, p0, p1);
                    if (nt1.Adjacent0 != -1)
LegalizeEdge(nti1, nt1.Adjacent0, pi, p1, p2);
                    if (nt2.Adjacent0 != -1)
LegalizeEdge(nti2, nt2.Adjacent0, pi, p2, p0);
                }
            }
```

3.3.2 Модуль геймплею

Діаграма модулю геймплею представлена на рисунку 39 і включає в собі такі елементи як зіткнення твердого тіла з поверхньою, її руйнування та вплив вже зруйнованої поверхні з іншими поверхнями.

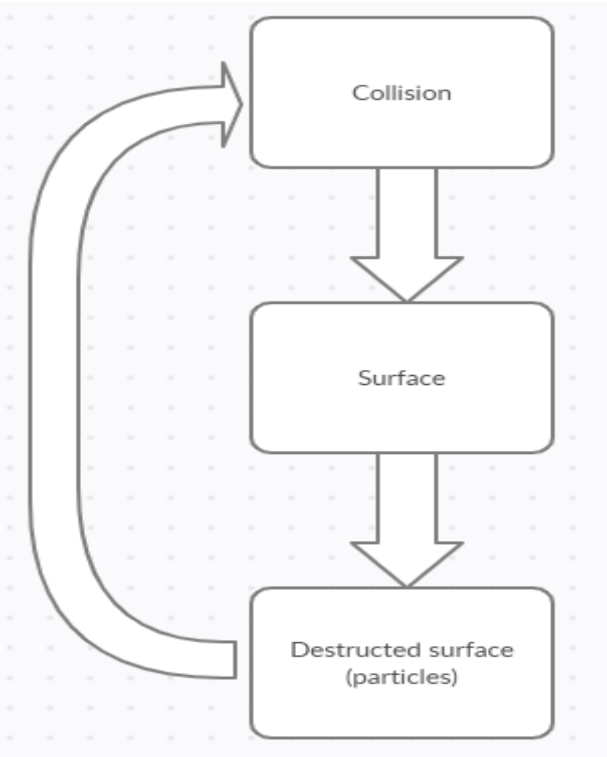

Рис. *39 Модуль геймплею*

Для відслідковування зіткнення одного об’єкту з іншим в ігровому движку Unity застосовується так звана фізика твердих тіл. Основними її компонентами являються тверде тіло (rigidbody) та колайдер (collider). Якщо ці два компоненти присутні на обох об’єктах, що стикаються, то система Unity аналізує точки зіткнення та подальшу фізику об’єктів за допомогою методу OnCollisionEnter, лістинг якої, в контексті існуючого геймплею, наведено в лістингу 2.

Лістинг 2 *Відстежування зіткнень твердих тіл*

```
     void OnCollisionEnter(Collision coll) {
     if (age > 5 && coll.impactForceSum.magnitude >
MinImpactToBreak) {
     var pnt = coll.contacts[0].point;
     Break((Vector2)transform.InverseTransformPoint(pnt));
     if (MainSurface)
     {
          GameManager.Instance.AddPoint();}}}
```

Розрізання (фрактурізація) відбувається завдяки можливості Unity модифікувати меш в рантаймі. Метод фрактурізації на основі отриманих діаграм наведено в лістингу 3.

Лістинг 3 *Модифікування мешу*

```
    Mesh MeshFromPolygon(List<Vector2> polygon, float
thickness) {
        var count = polygon.Count;
        var verts = new Vector3[6 * count];
        var norms = new Vector3[6 * count];
        var tris = new int[3 * (4 * count - 4)];
        var vi = 0;
        var ni = 0;
        var ti = 0;
        var ext = 0.5f * thickness;
        for (int i = 0; i < count; i++) {
            verts[vi++] = new Vector3(polygon[i].x,
polygon[i].y, ext);
            norms[ni++] = Vector3.forward;
        }

        for (int i = 0; i < count; i++) {
            verts[vi++] = new Vector3(polygon[i].x,
polygon[i].y, -ext);
            norms[ni++] = Vector3.back;
        }

        for (int i = 0; i < count; i++) {
            var iNext = i == count - 1 ? 0 : i + 1;
            verts[vi++] = new Vector3(polygon[i].x,
polygon[i].y, ext);
            verts[vi++] = new Vector3(polygon[i].x,
polygon[i].y, -ext);
            verts[vi++] = new Vector3(polygon[iNext].x,
polygon[iNext].y, -ext);
            verts[vi++] = new Vector3(polygon[iNext].x,
polygon[iNext].y, ext);

            var norm = Vector3.Cross(polygon[iNext] -
polygon[i], Vector3.forward).normalized;
            norms[ni++] = norm;
            norms[ni++] = norm;
            norms[ni++] = norm;
            norms[ni++] = norm;
        }
        for (int vert = 2; vert < count; vert++) {
```

```
            tris[ti++] = 0;
            tris[ti++] = vert - 1;
            tris[ti++] = vert;
        }

        for (int vert = 2; vert < count; vert++) {
            tris[ti++] = count;
            tris[ti++] = count + vert;
            tris[ti++] = count + vert - 1;
        }

        for (int vert = 0; vert < count; vert++) {
            var si = 2*count + 4*vert;

            tris[ti++] = si;
            tris[ti++] = si + 1;
            tris[ti++] = si + 2;

            tris[ti++] = si;
            tris[ti++] = si + 2;
            tris[ti++] = si + 3;
        }
        Debug.Assert(ti == tris.Length);
        Debug.Assert(vi == verts.Length);
        var mesh = new Mesh();
        mesh.vertices = verts;
        mesh.triangles = tris;
        mesh.normals = norms;
        return mesh;
    }
```

За надання фізичних властивостей отриманим з діаграми Вороного фрагментам відповідає фізичний движок, вбудований в Unity Engine. програмний движок, що переносить закони фізики реального світу у віртуальний (наприклад, ігри). Умовно поділяються на два види: наукові та ігрові. Ігровий тип фізичного движку використовується в комп'ютерних іграх як основний компонент, ланка ігрового движку. Це накладає деякі зобов'язання на нього: він повинен працювати у режимі реального часу, щоб відтворювати всі фізичні процеси у грі с той самою точністю, що і у реальному світі. Однак разом з

тим від ігрового фізичного движка не потребується висока точність розрахунків. Основна вимога до ігрового фізичного движка — візуальна реалістичність, тобто для досягнення цієї цілі не потрібно виконувати надточні розрахунки.

Так як Unity побудована на компонентному підході, то існує і компонент, що відповідає за фізику об’єктів. Такий компонент називається Rigidbody. За допомогою цього компоненту фрагментам надається початкова швидкість та направлення, по якій ці фрагменти будуть здійснюватися. Компонентом, що доповнює функціональність Rigidbody елементами взаємодії з іншими об’єктами, являється компонент Collider. Завдяки цьому об’єкти можуть зтикатися один із одним і в той самий час відбудеться перерахунок всіх векторів направлення та швидкості. Такі зіткнення називаються колізіями. Колізія в движку Unity — це зіткнення двох твердих тіл, в результаті якого виникають нові рушійні сили для обох цих тіл при наявному фізичному компоненті хоча б на одному із твердих об’єктів.

3.4 Проект інтерфейсу

Мобільний застосунок, представлений у вигляді гри, надає можливість користувачеві (гравцеві) руйнувати поверхні, що летять на гравця та заробляти бали та бонусні снаряди. Поточний прогрес гри відображається завдяки виводу поточних балів на користувацький інтерфейс, зображений на рисунку 40.

Рис. *40 Користувацький інтерфейс*

На даному екрані три головних елементи: лічильник балів, кнопка паузи/меню, та чорне тло, в якій відбувається основний геймплей. На рисунку 41 зображене меню паузи, де гравець може вибрати тип (в даному випадку зовнішній вигляд) об’єкту, що буде руйнувати літаючі поверхні.

Рис. *41 Меню паузи*

При досягненні гравцем певної кількості балів, його рівень майстерства підвищується, що інформується грою підказкою зверху (див. рис. 42).

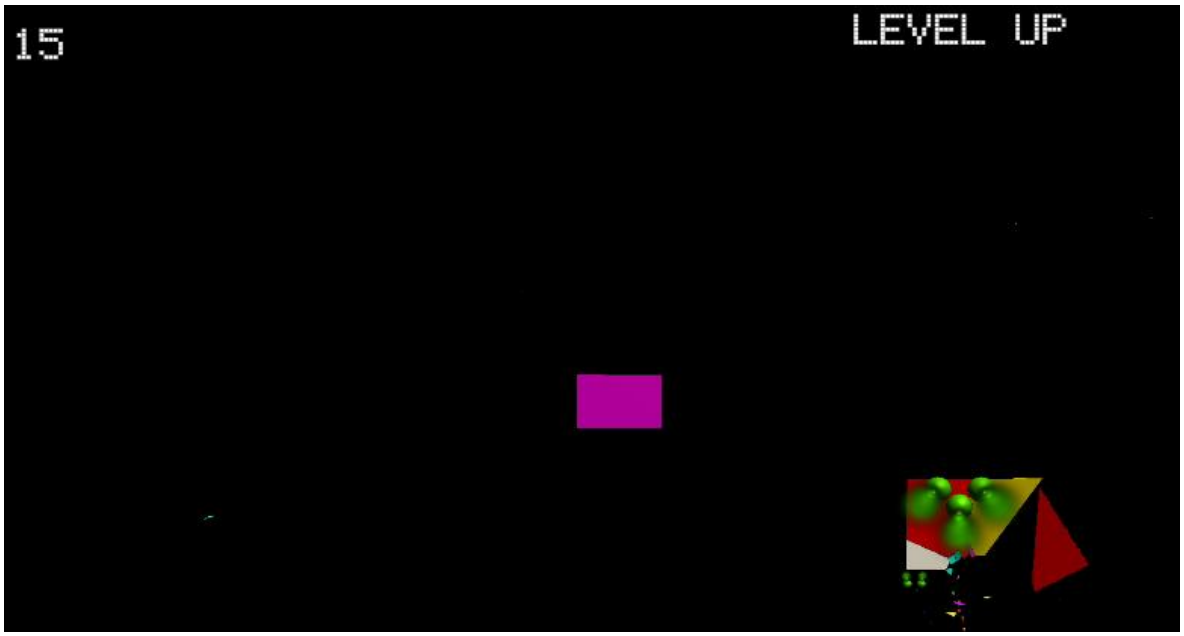

Рис. *42 Інформування про підвищення рівня майстерства*

3.5 Вимоги до апаратного забезпечення

Основною задачею розробленої програмної системи, яка представляє собою мобільну гру, є управління гравцем снарядами, що руйнують поверхні. Тому були визначені наступні критерії до застосунку.

Мінімальні вимоги до апаратного забезпечення:

- чотириядерний процесор з базовою частотою 1.2 GHz;
- оперативна пам’ять ємністю не менше 3 Гб;
- графічний прискорювач Adreno 505 або аналогічний йому.

Рекомендовані вимоги до апаратного забезпечення:

- восьмиядерний процесор з базовою частотою 1.8 GHz;
- оперативна пам’ять ємністю не менше 6 Гб;
- графічний прискорювач Adreno 630 або аналогічний йому.

3.6 Опис функціональних можливостей

Функціональні можливості застосунку направлені на отримання користувачем (гравцем) максимального задоволення від гри та захоплення реалістичністю руйнування поверхонь. За допомогою даної гри гравець може з легкістю та простотою проводити свій час у грі, не задумуючись над рейтингом або купівлею будь-якого цифрового контенту у грі. Даний застосунок знаходиться у бета доступі, який можна знайти в магазині ігор від Google — Google Play Store.

РОЗДІЛ 4 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ СИСТЕМИ РУЙНУВАННЯ

4.1 Аналіз пікового впливу моделювання руйнування на продуктивність мобільної системи в ігровому процесі

При розробці програмного застосунку, що представаляється у вигляді тривимірної гри для мобільних пристроїв під управлінням операційної системи Android, була протестована швидкодія генерації руйнування поверхонь при активному геймплеї. Взагалом, на продуктивність в даній системі впливає кількість генерованих часток, що залишились від початкового об’єкту та кількість одночасних колізій твердих тіл. Перший показник впливає на кількість споживаної оперативної пам’яті пристрою, завантаженість центрального процесору та графічного прискорювача. Виміри продуктивності велися на мобільному пристрої під управлінням операційної системи Android 10 та вирішено вимірювати у кількості кадрів за секунду (frames per second, FPS). У комп'ютерних іграх під кадровою частотою розуміється частота, з якою процес гри оновлює зображення в кадровому буфері [32]. При цьому гри можна розділити на два класи: ігри з постійною кадровою частотою і гри зі змінною кадрової частотою. Так, у таблиці 3 та 4 наведені експерименти в ході яких можна визначити пікові навантаження на систему при різній кількості генерованих часток.

Таблиця *3*

Порівняння продуктивності системи при різній кількості генерованих часток

Номер експерименту	Кількість часток	Кадрів за секунду (FPS)
1	5-10	Стабільно, 60
2	10-20	Стабільно, 60
3	20-25	Стабільно, 60
4	30-50	Пікові зниження, 30-50
5	60-70	Пікові зниження, 28-39
6	100-120	Пікові зниження, 25-30
7	>200	Пікові зниження, 15-20

На рисунку 43 можна побачити залежність частоти кадрів від кількості часток, що генеруються.

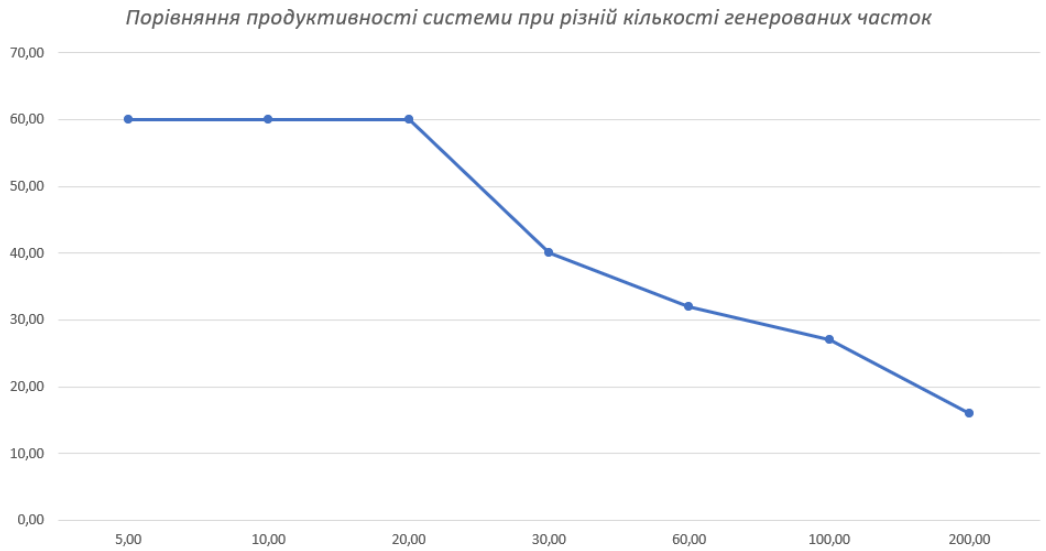

Рис. *43 Залежність частоти кадрів від кількості часток, що генеруються*

Таблиця *4*

Пікові навантаження на CPU та GPU мобільного пристрою

Номер експериме-нту	**Кількість час-ток**	**Завантаженість CPU, %**	**Завантаженість GPU, %**
1	5-10	55%	9%
2	10-20	58%	11%
3	20-25	58%	12%
4	30-50	59%	15%
5	60-70	63%	17%
6	100-120	75%	25%
7	>200	>80%	>30%

На рисунку 44 навдений графік, що показує залежність пікового навантаження на систему.

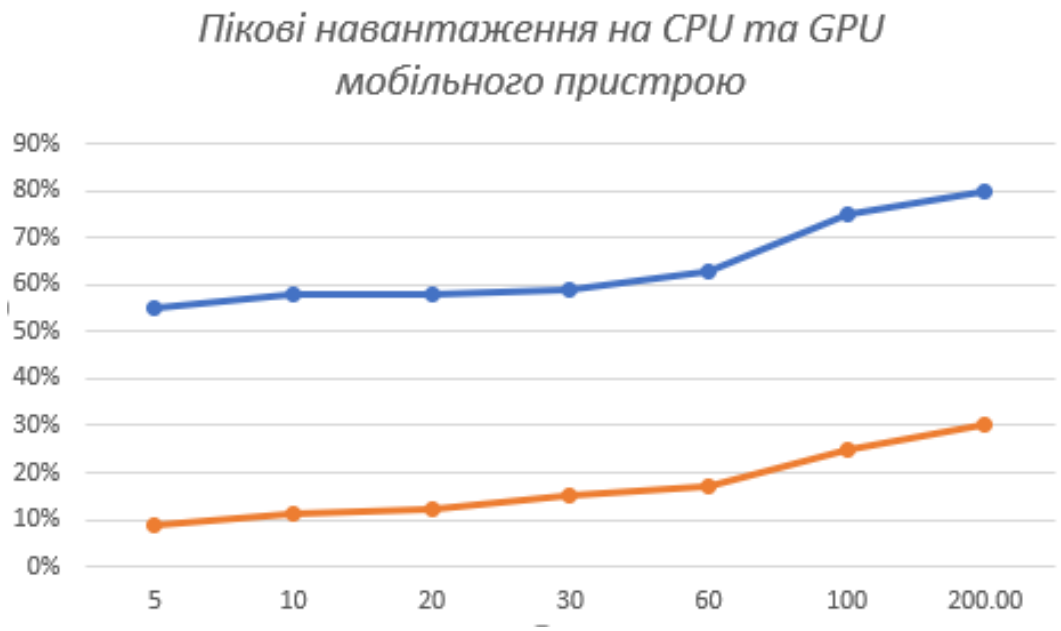

Рис. *44* Графік *пікового навантаження на CPU та GPU мобільного пристрою*

Заміри проводилися за допомогою мобільного додатку GameTurbo, що вбудований в фірмову оболонку MIUI, починаючи з 10 версії. Приклад такого заміру показано на рисунку 45.

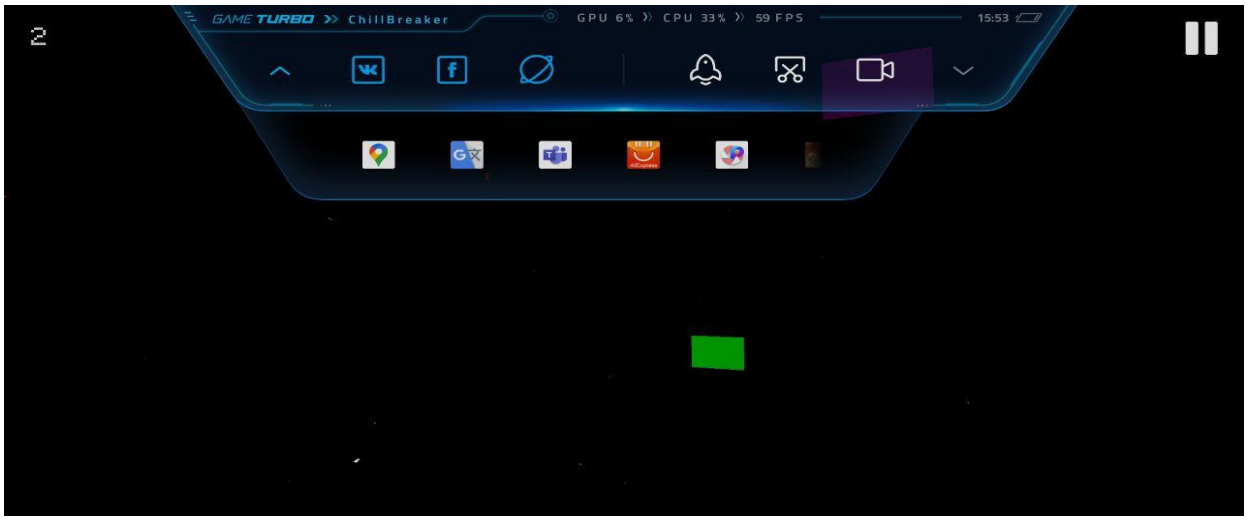

Рис. *45 GameTurbo для заміру продуктивності системи*

4.2 Аналіз довгого впливу моделювання руйнування на продуктивність мобільної системи в ігровому процесі

При довготривалій сесії гри, як комп'ютерної так і мобільної, виникає, зазвичай, нагрів компонентів пристрою, зокрема центрального процесору. Якщо своєчасно не охолодити центральний процесор, то подальші навантаження на нього можуть привести до перегріву та виходу з ладу пристрою. Для запобігання такого випадку процесор знижує, іноді кажуть «скидає», частоти, що призводить не тільки до його охолодження, а і до зниження загальної продуктивності системи. Такий процес називається пропус тактів, або дроселювання тактів. Частіше за все це поняття зустрічається під назвою тротлінг.

Тротлінг — механізм пропуску частини машинних тактів (циклів) в цифровій електроніці з метою синхронізації роботи різних компонентів (наприклад, в інтерфейсі SCSI) або їх захисту, в тому числі процесора, від термічного пошкодження при перегріванні [33]. У таблиці 5 наведені дані тестів

на кількість кадрів за секунду (FPS) в залежності від часу, проведеного за грою.

Таблиця *5*

Дані про тротлінг процесору

Час, хв	**FPS**	**Температура CPU, °C**
5	60	35
10	60	36
15	60	37
30	56	40
45	54	41
60	52	43

На рисунку 46 наведений графік залежності температури та кількості кадрів від часу у грі.

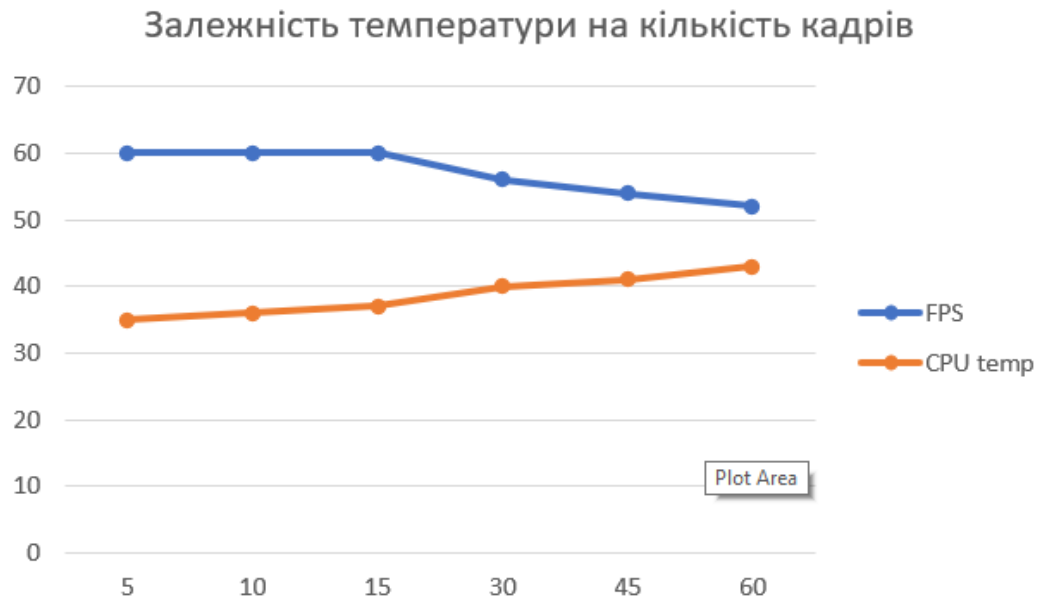

Рис. *46 Графік залежності температури та кількості кадрів від часу у грі*

ВИСНОВКИ

1. Встановлено актуальність розробки системи руйнування реального часу у вигляді тривимірної гри. Виходячи за аналізу можливостей ігрових движків з підтримкою симуляції фізики, теоретичних та практичних досліджень була сформована низка методів для дослідження, було проведено їх порівняння та аналіз, а також були визначені технології та засоби розробки, визначено їх продуктивність, простоту та зручність використання.

2. Досліджено сучасні відеоігри та ігрові движки з підтримкою руйнування в реальному часі, визначено їх переваги та недоліки, а також можливість використання у сучасній ігровій індустрії.

3. Досліджена можливість використання систем руйнування у тривимірних іграх на різних ігрових платформах, таких як персональні комп’ютери та мобільні телефони під управлінням операційної системи Android.

4. У ході дослідження і розробки основним інстурментом для реалізації системи було використано мову програмування C#, яка являється основною мовою розробки в ігровому движку Unity.

5. Розроблено програмний застосунок, що представляє собою тривимірну гру, демонструючи можливості руйнування поверхонь в реальному часі. Розробка велась під мобільну платформу на операційній системі Android.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Рендеринг с помощью вокселей: новый уровень графики в игpax?, URL: http://www.thg.ru/graphic/voxel_ray_casting/onepage.html (дата звернення: 10.10.2020)

2. Владимиров Д.А. Булевы алгебры. Москва: «Наука», 1969. С. 320.

3. 7 причин в пользу Maya, URL: https://animationclub.ru/blogs/2569/3031/7-maya (дата звернення 03.09.2020)

4. Келли Мэрдок. Autodesk 3ds Max 2013. Библия пользователя Москва: «Диалектика», 2013. С. 816.

5. Blender Feauters, URL: https://www.blender.org/features-gallery/ (дата звернення 04.09.2020)

6. What is Blender, URL: http://www.blender.org/blenderorg/blender-foundation/ (дата звернення 04.09.2020)

7. Asteroids, Retro Gamer journal. Live Publishing, 2006. С. 106—107.

8. The Pong Story, URL: http://www.pong-story.com/intro.htm (дата звернення 04.09.2020)

9. Games Now Legally Considered an Art Form, URL: https://v1.escapistmagazine.com/news/view/109835-Games-Now-Legally-Considered-an-Art-Form-in-the-USA (дата звернення 05.09.2020)

10. Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf. Computational Geometry. Springer-Verlag, 2000. С. 151—160.

11. David Austin. Voronoi Diagrams and a Day at the Beach, URL: http://www.ams.org/publicoutreach/feature-column/fcarc-voronoi (дата звернення 25.09.2020)

12. Steven Fortune. A sweepline algorithm for Voronoi diagrams. ACM Digital Library Proceedings of the second annual symposium on Computational geometry, 1986. С. 313—322.

13. K. Hinrichs, J. Nievergelt, and P. Schorn. Plane-sweep solves the closest pair problem elegantly, 1988. С. 255–261.

14. Rebay, S. Efficient Unstructured Mesh Generation by Means of Delaunay Triangulation and Bowyer-Watson Algorithm. Journal of Computational Physics Volume 106, 1993. С. 127.

15. Liu, Yuanxin, and Jack Snoeyink A comparison of five implementations of 3D Delaunay tessellation. Combinatorial and Computational Geometry, 2005. С. 439—458.

16. Делоне Б.Н. О пустоте сферы. Изв. АН СССР. ОМЕН, 1934. С. 793–800.

17. Gilbert P.N. New results on planar triangulations. Tech. Rep. ACT-15, Coord. Sci. Lab., University of Illinois at Urbana, 1979.

18. Ильман В.М. Экстремальные свойства триангуляции Делоне. Алгоритмы и программы, 1985. С. 57–66.

19. Lawson C. Transforming triangulations. Discrete Mathematics, 1972. С. 365–372.

20. Sibson R. Locally equiangular triangulations. Computer Journal. 1978. С. 243–245.

21. McCullagh M.J., Ross C.G. Delaunay triangulation of a random data set for isarithmic mapping. The Cartographic Journal, 1980. С. 93–99.

22. Heller M. Triangulation algorithms for adaptive terrain modeling. Proc. of the 4th Intern. Symp. on Spatial Data Handling, 1990. С. 163–174.

23. Фукс А.Л. Изображение изолиний и разрезов поверхности, заданной нерегулярной системой отсчётов. Программирование, 1986. С. 87–91.

24. Bjorke J.T. Quadtrees and triangulation in digital elevation models. International Archives of Photogrammetry and Remote Sensing, 16th Intern. Congress of ISPRS, Commission IV. Part B4, 1988. С. 38–44.

25. Скворцов А.В., Костюк Ю.Л. Эффективные алгоритмы построения триангуляции Делоне. Геоинформатика. Теория и практика., 1998. С. 22–47.

26. Фукс А.Л. Предварительная обработка набора точек при построении триангуляции Делоне. Геоинформатика. Теория и практика., 1998. С. 48–60.

27. Voronoi G. N New applications of continuous parameters to the quadratic form theory. Second Thesis: Research on primitive parallelohedrons, 1908. С. 198–287.

28. Unity Transform Docs, URL: https://docs.unity3d.com/ru/current/Manual/Transforms.html (дата звернення 15.11.2020)

29. C# and .NET Fundamentals, URL: https://www.kau.edu.sa/Files/830/Files/55533_C_Sharp_Programming.pdf (дата звернення 17.11.2020)

30. Render pipelines in Unity, URL: https://pydonzallaz.files.wordpress.com/2018/11/unity_lighting_rendering_pineline_donzallaz.pdf (дата звернення 18.11.2020)

31. Android SDK Docs, URL: https://developer.android.com/studio/command-line (дата звернення 19.11.2020)

32. Understanding and Optimizing Video Game Frame Rates, URL: https://www.lifewire.com/optimizing-video-game-frame-rates-811784 (дата звернення 19.11.2020)

33. Функции управления производительностью и энергопотреблением процессоров, URL: https://www.ixbt.com/cpu/intel-thermal-features-p4.shtml, (дата звернення 19.11.2020)

34. Гармаш Олексій, студента магістратури ФЕЕІТ ІІ ЗНУ. Наук. кер.: проф. Вербицький В. Г. «КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ ПРОЦЕСУ РУЙНУВАННЯ ТІЛ У 3D ІГРАХ». Збірник наукових праць студентів, аспірантів і молодих вчених «Молода наука-2020» : у 5 т. Запорізький національний університет. Запоріжжя: ЗНУ, 2020. Т.5. С. 82-84.

35. Гармаш О. Р., магістрант, Гармаш О.Р., Вербицький В.Г., проф. – науковий керівник. Комп’ютерне моделювання процесу руйнування тіл у 3d іграх. МАТЕРІАЛИ XXV науково-технічної конференції студентів, магістрантів, аспірантів, молодих вчених та викладачів. Запоріжжя : ЗНУ, 2020. С. 161.

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Гармаш Олексій Романович, студент _2_ курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти sp115-08@stu.zsea.edu.ua, — підтверджую, що написана мною кваліфікаційна робота на тему «**Комп'ютерне моделювання процесу руйнування тіл у 3D іграх**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2020 Підпис _____ Гармаш Олексій Романович (студент)

Дата 30.11.2020 Підпис _____ Вербицький Володимир Григорович (науковий керівник)