НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Навчально-науковий інститут прикладного системного аналізу
Кафедра системного проектування

До захисту допущено:

Завідувач кафедри

_____ В. Є. МУХІН

«__»_____ 2022 р.

Магістерська дисертація

на здобуття ступеня магістра
за освітньо-науковою програмою
“Інтелектуальні сервіс-орієнтовані розподілені обчислювання”
зі спеціальності 122 "Комп'ютерні науки"
на тему: «Дослідження моделювання 3D колізій з використанням хмарних систем**»**

Виконав (-ла):
студент (-ка) ІІ курсу, групи ДА-01мн
Сиротюк Олександр Васильович _____

Науковий керівник:
к.т.н., доц.
Харченко Костянтин Васильович _____

Рецензент:
д.т.н., доц.
Аушева Наталія Миколаївна _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2022

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Навчально-науковий інститут прикладного системного аналізу

Кафедра системного проектування

Рівень вищої освіти – другий (магістерський)

Спеціальність – 122 "Комп'ютерні науки"

Освітньо-наукова програма – "Інтелектуальні сервіс-орієнтовані розподілені обчислювання"

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Вадим МУХІН

«___»_____2022 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Сиротюку Олександру Васильовичу

1. Тема дисертації «Дослідження моделювання 3D колізій з використанням хмарних систем» науковий керівник дисертації Харченко Костянтин Васильович к.т.н., доц., затверджені наказом по університету від «___»_____ 20__ р. №_____

2. Термін подання студентом дисертації – 10 червня 2022 р.

3. Об’єкт дослідження
Підходи до розрахунку 3D колізій, а також дослідження архітектур рушіїв, засобів їх моделювання.

4. Предмет дослідження
- Наукові статті до опрацювання
- Існуючі системи зі схожою проблематикою (документація, програмний код, навчальна література)
- Теоретична література за напрямком дослідження:

- Ericson C. Real-Time Collision Detection / Christer Ericson., 2005. – 594 c. – (The Morgan Kaufmann Series in Interactive 3D Technology).
- Eberly D. H. Game Physics / David Eberly., 2010. – 826 c. – (The Morgan Kaufmann Series in Interactive 3D Technology).
- van den Bergen G. Collision Detection in Interactive 3D Environments / Gino van den Bergen., 2003. – 277 c. – (The Morgan Kaufmann Series in Interactive 3D Technology).
- van den Bergen G. Game Physics Pearls / G. van den Bergen, D. Gregorius., 2010. – 352 c.

5. Перелік завдань, які потрібно розробити
   1. Розглянути підходи до визначення колізій в тривимірному просторі
   2. Розглянути підходи до побудови рушіїв симуляцій колізій
   3. Переглянути існуючі засоби та фізичні рушії, а також оцінити можливість їх використання з хмарними технологіями
   4. Оцінити ефективність використання хмарних та розподілених обчислень в фізичних рушіях
   5. Надати архітектуру рушія, котрий би міг бути використаний при роботі з хмарними рішеннями
   6. Оформити результати у вигляді статті.

6. Перелік графічного (ілюстративного) матеріалу
   1. Презентація для захисту роботи.

7. Орієнтовний перелік публікацій
   1. Одна запланована публікація за темою дослідження.

8. Консультанти розділів дисертації*

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

* Консультантом не може бути зазначено наукового керівника

9. Дата видачі завдання

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів магістерської дисертації	Примітка
1.	Отримання завдання	31 січня 2022	
2.	Ознайомлення з теоретичним підґрунтям, необхідним для дослідження алгоритмів визначення колізій в тривимірному просторі.	7 лютого 2022	
3.	Ознайомлення з алгоритмами знаходження та вирішення колізій в тривимірному просторі	14 лютого 2022	
4.	Огляд останніх наукових праць та літератури в предметній області	21 лютого 2022	
5.	Огляд практичних матеріалів для проведення розподілених розрахунків та побудови хмарних застосунків	28 лютого 2022	
6.	Звіт з проходження практики	6 березня 2022	
7.	Ознайомлення з існуючими засобами, котрі використовують розглянуті раніше алгоритми	25 квітня 2022	
8.	Проведення досліджень та вимірів роботи фізичного рушія у різних середовищах	10 травня 2022	
9.	Створення архітектури, котра могла б в повній мірі розкрити потенціал використання хмарних технологій	25 травня 2022	
10.	Оформлення дипломної роботи	5 червня 2022	
11.	Отримання допуску до захисту та подача роботи в ДЕК	10 червня 2022	

Студент О.В. Сиротюк

Науковий керівник дисертації К.В. Харченко

РЕФЕРАТ НА МАГІСТЕРСЬКУ ДИСЕРТАЦІЮ

виконану на тему: «Дослідження моделювання 3D колізій з використанням хмарних систем»
студентом: Сиротюком Олександром Васильовичем

Дана дисертація має об’єм в 131 сторінки, включає в себе 34 ілюстрації, 25 таблиць, а також 25 формул і 3 додатків. В результаті виконання представленої магістерської дисертації було оброблено 21 джерел (літературних та інших посилань).

Актуальність теми

Задача моделювання 3D колізій актуальна зараз в різних галузях: медицина та діагностика захворювань на рівні обробки відео, автоматичне та автоматизоване керування транспортними засобами, моделювання складних тривимірних систем, планування складних автоматизованих процесів з робототехнікою, візуалізація тривимірних зображень і т.д.

Мета і завдання дослідження

Головною метою даної роботи є проведення дослідження алгоритмів проведення фізичних симуляцій, а конкретно – розрахунку колізій в тривимірному просторі, а також аналіз застосування даних алгоритмів при хмарних та розподілених обчисленнях.

Другорядним завданням є аналіз архітектур програмних компонентів для проведення такого роду моделювань та пропозиція що до розробки даного виду програмних застосунків.

Рішення поставлених завдань і досягнуті результати

В результаті виконання даної магістерської дисертації було отримано архітектуру та прототип системи, котра може бути використана для проведення фізичних симуляцій з використанням хмарних систем, а також кластерів ЕОМ.

Розглянуто проблематику побудови такого виду систем, до якої включено як аналіз слабких місць архітектури та інфраструктури, так і аналіз переваг використання різних алгоритмів.

Проведено дослідження не тільки теоретичної, а й практичної бази, до переліку котрої входять не тільки самі засоби та API для проведення фізичних симуляцій, а й застосунки з екосистем (та використання даних застосунків), котрі можуть бути використані в роботі.

Об’єкт досліджень

Проведення фізичних симуляцій з використанням на ЕОМ та підходи до їх використання в хмарних система, а також в розподілених обчисленнях з використанням кластерів.

Предмет досліджень

Предметом дослідження виступають сучасні рушії фізичного моделювання, основні алгоритми, котрі є базовою для проведення фізичного моделювання на ЕОМ а також конкретні архітектури фізичних рушіїв.

Наукова новизна

Новизна роботи полягає у дослідженні використання класичних алгоритмів та підходів знаходження колізій в хмарних та розподілених системах. Також новизною є використання та дослідження конкретних технологій для виконання даних задач. Окремо варто виділити запропоновану архітектуру для вирішення проблеми моделювання фізичних процесів.

Практичне значення отриманих результатів

Головним практичним значенням є отримані результати дослідження (графіки, числові значення та висновки до них), також важливим практичним результатом є отримання модифікації технології Bullet для роботи з кластерами через бекенд OpenCL.

Останнім важливим практичним досягненням є створений план розробки ПЗ, котре могло бути використаним для проведення симуляцій з використанням хмарних технологій та враховувати всі слабкі місця при використанні звичайних рушії фізичних симуляцій при їх використанні в клієнт-серверних архітектурах.

Ключові слова

Тривимірний простір, неперервний простір, колізії, знаходження колізій, вирішення колізій, хмарні технології, розподілені обчислення, високонавантажені обчислення, ігровий рушій, фізика.

ABSTRACT ON MASTER`S THESIS

on topic: «Research of 3D collision modeling using cloud systems»

student Syrotiuk Oleksandr

This master thesis consists of 131 pages, has 34 figures alongside with 25 tables, 25 formulas and 3 additional materials. As a result of the presented master's thesis, 21 external sources (bookish and other references) were processed.

Topicality

The problem of 3D collision modeling is relevant today in various fields: medicine and diagnostics of diseases at the level of video processing, automatic and automated vehicle control, modeling of complex three-dimensional systems, planning of complex automated processes with robotics, and three-dimensional image visualization, etc.

Purpose and objectives

The main purpose of this work is to study algorithms for physical simulations, specifically - the calculation of collisions in three-dimensional space, as well as analysis of the application of these algorithms in cloud and distributed computing.

A secondary goal is to analyze the architectures of software components for such simulations and to propose the development of this type of software application.

Solutions and achieved results

As a result of this master's dissertation, the architecture and prototype of the system was obtained, which can be used to conduct physical simulations using cloud systems, as well as computer clusters.

The problems of building this type of systems are considered, which includes both the analysis of weaknesses of architecture and infrastructure, and the analysis of the advantages of using different algorithms.

A study of not only the theoretical but also the practical basis, which includes not only the tools and API for physical simulations, but also applications from ecosystems (and the use of these applications) that can be used in the work.

The object of the research

Conducting physical simulations using computers and approaches to their use in cloud systems, as well as in distributed computing using clusters.

The subject of the research

The subject of the research are modern engines of physical modeling, basic algorithms, which are the basis for conducting physical modeling on a computer, as well as specific architectures of physical engines.

Scientific novelty

The novelty of the work is the study of the use of classical algorithms and approaches to finding collisions in the cloud and distributed systems. Also new is the use and research of specific technologies to perform these tasks. The proposed architecture should be singled out to solve the problem of modeling physical processes

The practical value of the research

The main practical value is the obtained research results (graphs, numerical values, and conclusions to them), also an important practical result is to obtain a modification of Bullet technology to work with clusters through the backend OpenCL.

The last important practical achievement is the creation of a software development plan that could be used to conduct simulations using cloud technologies and take into account all the weaknesses in the use of conventional physical simulation engines when used in client-server architectures.

Keywords

3D space, continuous space, collisions, collision solution, collision detection, cloud technologies, distributed computing, high-performance computing, game engine, physics.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ЕОМ – електронна обчислювальна машина, комп’ютер.

HPC – high-performance computing, високопродуктивні обчислення.

FPS – frames per second, кадрів за секунду. Величина обернена до довжини кадру, стандартна величина виміру швидкодії симуляцій.

BV – bounding volume, обмежуючий об’єм.

OOBB – object oriented bounding box, BV-прямокутник, вирівняний по осям тіла.

AABB – axis aligned bounding box, BV-прямокутник, вирівняний по осям сцени.

BS – bounding sphere, BV у вигляді сфери.

ADS – acceleration data structures, прискорюючі структури даних.

BVH – bounding volume hierarchy, ієрархія обмежуючих об’ємів, узагальнюючий термін для даного виду/класу структур даних.

DBVT – dynamic bounding volume tree, конкретна древовидна BVH структура даних.

SPA – space partitions algorithms, алгоритми розбиття простору.

BSP – binary space partition, двійкове розбиття простору.

GJK – Gilbert-Johnson-Keerthi, абревіатура прізвищ авторів алгоритму та назва одно з найпопулярнішіх алгоритмів знаходження колізій та відстані між тілами.

SAT – separating axis theorem, теорема осі поділу, використовується для знаходження колізій в простих сценах, є теоретичною основою інших підходів.

CCD – continues collision detection, знаходження колізій в неперервному просторі.

EPA – expanded polytope algorithm, розширений політопний алгоритм, використовується для знаходження глибини проникнення тіл один в одного.

MTV – minimal translation vector, мінімальний вектор трансляції. Вектори, на котрий необхідно перенести одне з тіл, щоб пара тіл більше не стикалися.

C-функція – Constrain function, функція обмежень, дана функція описує обмеження, котрі прикладаються до тіл під час їх симуляції чи взаємодії між собою, повертає нормалізоване значення від 0 до 1, котре описує ступінь задоволеності обмеження (трактовка залежить від типу обмеження).

SDK – software development kit, набір застосунків розробника, набір інструментів та бібліотек, котрі поставляються одним пакетом з ціллю виконання заздалегідь визначеного спектру задач.

AI – artificial intelligence, штучний інтелект, в контексті даної роботи мається на увазі не повноцінний штучний інтелект, а набір алгоритмів та підходів що до побудови інтелектуальних агентів, акторів в симуляції, здатних до певної поведінки.

P2P – pear-to-pear, пряме з’єднання між клієнтами.

HLD – high level description, абстрактна діаграма високого рівня, прийнята опис та базові концепції у вигляді графічного матеріалу.

VCS – version control system, система контролю версій.

API – application programming interface, програмний інтерфейс

ВСТУП

Першим та основним прикладним завданням в історії, котре виникло перед користувачами ЕОМ є вирішення математичних задач, зокрема – їх вирішення з використанням чисельних методів розв’язку. Ще до появи перших ЕОМ дані методи використовувалися інженерами та математиками, проте ефективність використання даних методів в повній мірі розкрилася лише з появою ЕОМ, котра змогла в повній мірі розкрити в основному ітеративну, або ж високо-затратну суть даного сімейства підходів.

Одними з першим задач, до котрих використовувалися чисельні методи та обрахунки на ЕОМ є різного роду симуляції, в основному ці симуляції об’єднує можливість створення чіткої математичної моделі. Дані математичні моделі можуть мати будь який характер: опис проєкту, економічного, соціально явищ. Проте, звичайно, найпершою та найбільш природньою є модель поведінки реального світу: електричні кола, маятники, аеро та гідродинаміка – всі ці предметні області є неймовірно важливими.

Окремим видом моделювання, котре хочеться розглянути в даній роботи є моделювання елементів механіки, а також використання HPC (високо-ефективних обчислень) для вирішення даного типу моделювання в системах реального часу, за можливості та наявності апаратного забезпечення – інтерактивні симуляції.

Проте, перед тим як переходити до використання HPC підходів та архітектур в даному моделюванні необхідно розглянути перш за все основну теоретичну базу даного моделювання з використанням ЕОМ. Перший розділ роботи буде посвячений використанню математичних методів для проведення симуляцій поведінки твердих тіл, а також систем частинок в необмеженому просторі.

Другий розділ роботи є ключовим, в ньому проводяться дослідження існуючих засобів та рішень, надаються посилання на актуальні технології та їх

аналіз, також саме там проведено практичну частину роботи, котра присвячена технічному аналізу хмарних та розподілених обчислень, а також створенню проєкту рушія, котрий буде орієнтований саме на такий вид обчислень.

Більша частина рішень, котрі дозволяють проводити фізичні симуляції на кластерах та хмарах є або зовсім примітивними та в початковому стані розробки, або ж є платними/закритими і не можуть бути використані в повній мірі для проведення наукових досліджень або в роботі малими інженерними компаніями.

Виходячи з тексту вище, фінальною задачею є отримання відповіді що до питання адекватності використання хмарних технологій для роботи з фізичними симуляціями, а також надання прототипу архітектури для створення роботи такого рушія.

1 ОСНОВИ ФІЗИЧНИХ СИМУЛЯЦІЙ ТВЕРДИХ ТІЛ ДЛЯ КОМП'ЮТЕРНОЇ ГРАФІКИ

Фізичне моделювання — це область комп’ютерної науки, яка спрямована на відтворення фізичних явищ за допомогою комп’ютера, також, під даним значенням сприймають сам процес моделювання.

Загалом, ці моделювання застосовують чисельні методи до існуючих теорій, щоб отримати результати, які максимально наближені до того, що ми спостерігаємо в реальному світі. Спектр застосування фізичного моделювання величезний. Найперші ЕОМ використовувалися для фізичного моделювання, наприклад, балістичного руху снарядів у військових.

В цивільному житті, даний підхід дозволяє інженерам симулювати поведінку певних конструкцій, перед тим як втілювати їх в життя: мости, літаки, човни, будівлі. Інженери можуть перевірити супротив споруд до землетрусів, паводків, бурь. Мало того, вчені використовують відповідні фізичні рушії для моделювання поведінки космічних тіл.

Окрім зазначеного вище, дане моделювання широко використовуються і в індустрії розваг, так як фізичний рушій (engine), є одним з основним компонентів ігрового рушія (game engine), котрі також використовуються і при створенні кіно-фільмів, а також будь яких інших видів інтерактивних симуляцій. Виходячи з зазначеного, основними напрямками, котрі розвиваються в фізичному моделюванні є симуляція:

- Твердих тіл (rigid bodies simulation).
- М’яких тіл (soft bodies simulation).
- Частинок (particles simulation).
- Рідких тіл (fluid simulation).

Виходячи з досвіду було прийнято рішення сконцентруватися на розгляданні основних правил симуляції на приклад моделювання поведінки саме твердих тіл, а також частинок й систем твердих тіл, так як:

- Дані види моделювання є концептуально досить простими та інтуїтивними для розгляду.
- Дані види моделювання з іншої сторони є досить комплексними, щоб розкрити суть явища та основні підходи.
- Дані види моделювання використовуються та зустрічаються частіше інших. В той час як інші види симуляцій є скоріше частинними випадками даних видів моделювання.
- В даних видам симуляцій досить наглядно можна показати та візуалізувати поведінку деяких модельованих об’єктів, котрі взаємодіють один з одним.

В процесі симуляції в більшості випадків ми хочемо отримати не лише точний та змістовний результат, в більшості випадків ми хочемо отримати плавне візуальна зображення.

Як було згадано раніше, це досягається з використанням високоефективних алгоритмів, підходів та чисельних методів, прикладених до базових законів фізики, котрі доступні нам ще з шкільних курсів загальної фізики.

Набір даних теоретичних законів та підходів до їх чисельного розрахунку, разом з архітектурою та програмною її реалізацією утворюють так званий **фізичний рушій**, котрий і виконує процес моделювання фізичних явищ на **сцені** (певному визначеному просторі).

Схема, визначена на рисунку 1. підходить до більшості програмного забезпечення, котре виконує певний вид симуляції, особливо дана схема буде знайома людям, котрі активно приймають участь у роботі/розробці інтерактивного програмного забезпечення. Загальна схема фізичного рушія в основному визначається головною петлею, котра представлена на рисунку 1.

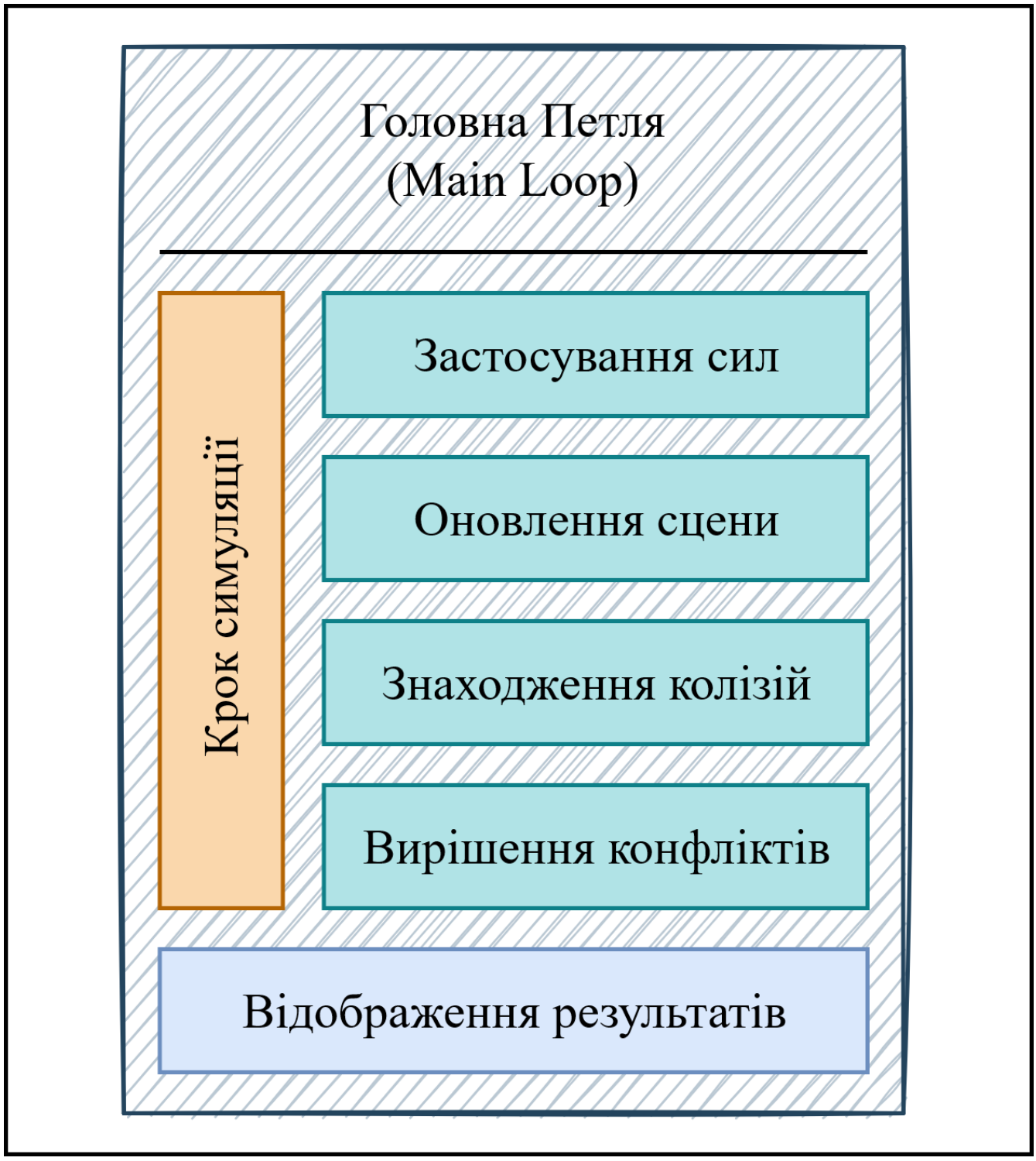

Рисунок 1 – Схематичне зображення головної петлі фізичного рушія

З рисунку 1 можна виділити, що головня петля поділяється на два основні етапи – проведення **кроку симуляції**, а також **відображення результатів симуляції** цього кроку. Відображення результатів симуляції може мати різну

форму, формально цей етап – фіксація результатів роботи рушія на даному кроці та завершення роботи ітерації головної петлі.

Даною фіксацією може бути як просто вивід значень в файл, так і візуалізація з використанням 3D рушія. В останньому випадку все дуже сильно залежить від реалізацію архітектури всього рушія (в такому випадку рушій фізики є окремою складовою загального рушія, рівно як і рендер рушій, котрий займається візуалізацією).

Крок симуляції – це частина головної петлі, котра займається саме симуляцією. Дана частина може бути чітко поділена на 4 основні складові (інколи – 5 складових, але більшість базових архітектур пропонують 4 етапи, зазначені на рисунку, а також нижче):

- **Застосування сили** – перший етап, на ньому до суб'єктів симуляції прикладаються фізичні сили, котрі визначені для впливу в симуляції. Приклад – гравітація, товчки іншими об'єктами. У випадку інтерактивної симуляції – ввід користувача.
- **Оновлення сцени** – наступний етап, на якому з урахуванням впливу сил, визначених на етапі вище вираховуються актуальні параметри об'єктів на сцені (швидкість, позиція та інше).
- **Знаходження колізій** – в залежності від отриманих позицій та інших параметрів об'єктів на даному етапі знаходяться колізії між об'єктами. Даний етап є основним етапом, де відбувається найбільше навантаження на систему.
- **Вирішення конфліктів** – для об'єктів, котрі прийшли в стан взаємної колізії необхідно вирішити конфлікт. На даному етапі відбувається вторинне визначення основних параметрів об'єктів з урахуванням всіх даних, отриманих під час колізії. Тобто, на даному етапі відбувається саме розрахунок результату колізії.

Слід зауважити, що симуляції бувають різного виду в залежності від типу обраного кроку. В досить простих системах/моделях очевидним рішенням є взяття фіксованого кроку, при якому довжина кроку симуляції є фіксованою (приклад – припускаємо, що час між двома кроками симуляції – 10 секунд), в такому випадку симуляція ніяк не прив'язана до реального часу і може використовуватися лише у вигляді розрахунку концепту, або ж математичної моделі.

Для більш практичного використання намагаються зв'язати час виконання кроку симуляції, а також реального часу, при цьому збільшуючи кількість даних кроків. В такому випадку досить реальним є отримання стану, коли довжина кожного кроку різна, і може бути абсолютно мала. В професійному середовищі довжину симуляції позначати як **довжину кадру**, або ж через зворотню величину – **кількість кадрів за секунду**, надалі будемо використовувати прийняту термінологію **FPS**. Крок симуляції будемо називати – **кадр**, а довжину кроку, як було зазначено раніше – **довжина кадра**.

Останнім зауваженням перед переходом до фізичної складової є те, що досліджувані в даній роботі симуляції відносяться до неперервних. Тобто, простір, в котрій буде проводитися симуляція не є дискретним, а координати та інші просторові параметри об'єктів можуть бути визначені довільним числом (з урахуванням максимальної допустимої точності ЕОМ). Такий вид симуляції називаємо – неперервною (continues).

Простір симуляції прийнято називати **сценою симуляції**, або ж просто – **сцена**. Також варто зауважити, що сцени можуть бути вкладені один в одного, проте, що є очевидним, існує принаймні одна головна сцена. Представлення сцен є окремим питанням, котре має суттєвий вплив на ефективність обрахунку, а також на ефективність розпаралелювання задач на кластерах/GPU, або ж при передачі даних при розташуванні рушія фізики в хмарному середовищі.

1.1 Динаміка матеріальної точки

Як було зазначено в попередній частині, в даній роботі нас цікавить в першу чергу симуляція поведінки динаміки твердих тіл, проте для поступового пояснення та розгляду теоретичної бази та основних підходів, потрібно розглянути динаміку матеріальної точки. Даний підхід дозволить індуктивно перейти до розгляду механіки твердого тіла.

Перш за все слід зауважити, що в фізичних симуляція матеріальні точки прийнято називати частинками, або ж – партиклами (англійського – particle), будемо притримуватися даної термінології. Моделювання поведінки частинок є відносно простішим, ніж твердих тіл. Проте, моделювання твердих тіл в тій чи іншій мірі використовує формули та принципи, необхідних для симуляції частинок.

Формально, **частинка** – точка простору, яка повністю описується положенням у просторі, вектором швидкості (скалярною величиною швидкості та вектором напрямку) та масою.

Всі основні принципи при моделюванні частинок відповідають основним постулатам ньютонівської механіки – законам Ньютона:

- **Інерційність** – швидкість та напрямок переміщення тіла стабільні та змінюються лише за рахунок прикладання сил до об’єкта.
- **Протидія сил** – на кожну силу, прикладену до тіла існує відповідна в силі протидія. Значення та вектор напрямку протидійної сили є протилежними до прикладної.
- **Сумарне значення сил** – загальне значення сили, прикладеної до об’єкта рівна добутку маси на прискоренні тіла. Даний закон описує основну формулу, котра використовується на етапі «застосування сил», наведеного на рисунку 1.

Згідно з трьома законами вище визначаємо, що швидкість та положення частинки змінюються лише за наявності прикладеної до неї сил. Проте відсутність прикладеної сили не повинно виключати жоден об’єкт з етапу моделювання, так як він все ще може приймати участь в процесії колізії.

Сила може прикладатися до частинки з різних джерел в залежності від точності моделювання (мається на увазі не точність обрахунків, а відповідність процесу моделювання, симуляції еталонному відповіднику, частіше за все – реальному світу).

Прикладами сил можуть слугувати гравітація, вітер, магнетизм, вплив інших тіл або рідин. Окрім цього, сила може бути комбінацією перечисленнях вище джерел. Також, слід зауважити важливе з точки зору рушія розділення, що сила може бути:

- **Глобальною** – значення, котре діє на всій сцені. Прикладом може слугувати постійна сила гравітації, котра може бути прикладена до будь якого об’єкту, або ж сила притяжіння від якогось великого об’єкту, котрий діє на всю сцену одночасно. Також до таких сил можна віднести виштовхування, якщо вся сцена знаходиться у рідкому середовищі.
- **Локальною** – значення, котре діє в певних виділених областях сцени. В залежності від архітектури рушія може виділятися, або ж не виділятися (в такому випадку локальні сили вважаються глобальними, проте у виділених підсценах).
- **Динамічною** – сили, котрі виникають динамічно в процесі симуляції та діють лише на певні виділені тіла, наприклад – притягання чи відштовхування між двома частинками, що знаходяться поруч. Також до даної категорії відносяться сили, котрі виникають при зштовхуванні об’єктів симуляції.

Припустимо, що ми маємо частинку, її параметри можна обрахувати за допомогою використанням формул 1 – 4, наведених нижче:

$$dt = t_{i+1} - t_i \quad (1)$$

$$v(t_{i+1}) = v(t_i) + \frac{f(t_i)}{m} dt \quad (2)$$

$$p(t_{i+1}) = p(t_i) + v(t_{i+1})dt \quad (3)$$

$$F = ma \quad (4)$$

, де $\boldsymbol{m}$ – маса частинки, $\boldsymbol{t_i}$ – початковий момент часу, $\boldsymbol{t_{i+1}}$ – наступний момент часу, $\boldsymbol{p(t_i)}$ – положення частинки в момент часу $\boldsymbol{t_i}$, $\boldsymbol{v(t_i)}$ – швидкість частинки в момент часу $\boldsymbol{t_i}$,. Для формули 1 та наступних визначено що $\boldsymbol{dt}$ – різниця між значеннями часу $\boldsymbol{t_{i+1}}$ та $\boldsymbol{t_i}$ – називається дельтою часу (в рушії сприймається як довжина кроку симуляції, згадана в початку пункту 1). Для формули 2 та наступних визначено, що $\boldsymbol{v(t_{i+1})}$ – швидкість в момент часу $\boldsymbol{t_{i+1}}$, $\boldsymbol{f(t_i)}$ – сума всіх сил що діє на частинку в момент часу $\boldsymbol{t_i}$. Для формули 3 визначено, що $\boldsymbol{p(t_{i+1})}$ – положення частинки в момент часу $\boldsymbol{t_{i+1}}$. Формул 4 є прямим описом третього закону Ньютона в формульному вигляді, де $\boldsymbol{F}$ – сума всіх сил прикладених до частинки, $\boldsymbol{a}$ – лінійне прискорення частинки. Слід зауважити, що для частинки ми наголошуємо на тому що $\boldsymbol{a}$ є лінійна швидкість.

В формулах 1 – 3 технічно та формально ми проводимо чисельне інтегрування звичайного, диференційного рівняння руху частинки (тіла) в просторі за допомогою неявного методу Ейлера. Дану методологію використовує більшість рушіїв фізики завдяки його простоті, очевидності та задовільної точності при малих значеннях часу (слід зауважити, що в таких рушіях симуляція зазвичай відбувається при значеннях 30 – 60 FPS, що дорівнює проміжку часу від 33.(3) до 16.(6) мілісекунд відповідно.

Примітивний код, в котрому наводиться приклад розрахунку симуляції для декількох попередньо визначених частинок лише з прикладанням сили гравітації

(вертикальне падіння вниз з початково заданим вектором швидкості) можна переглянути в додатку 1.

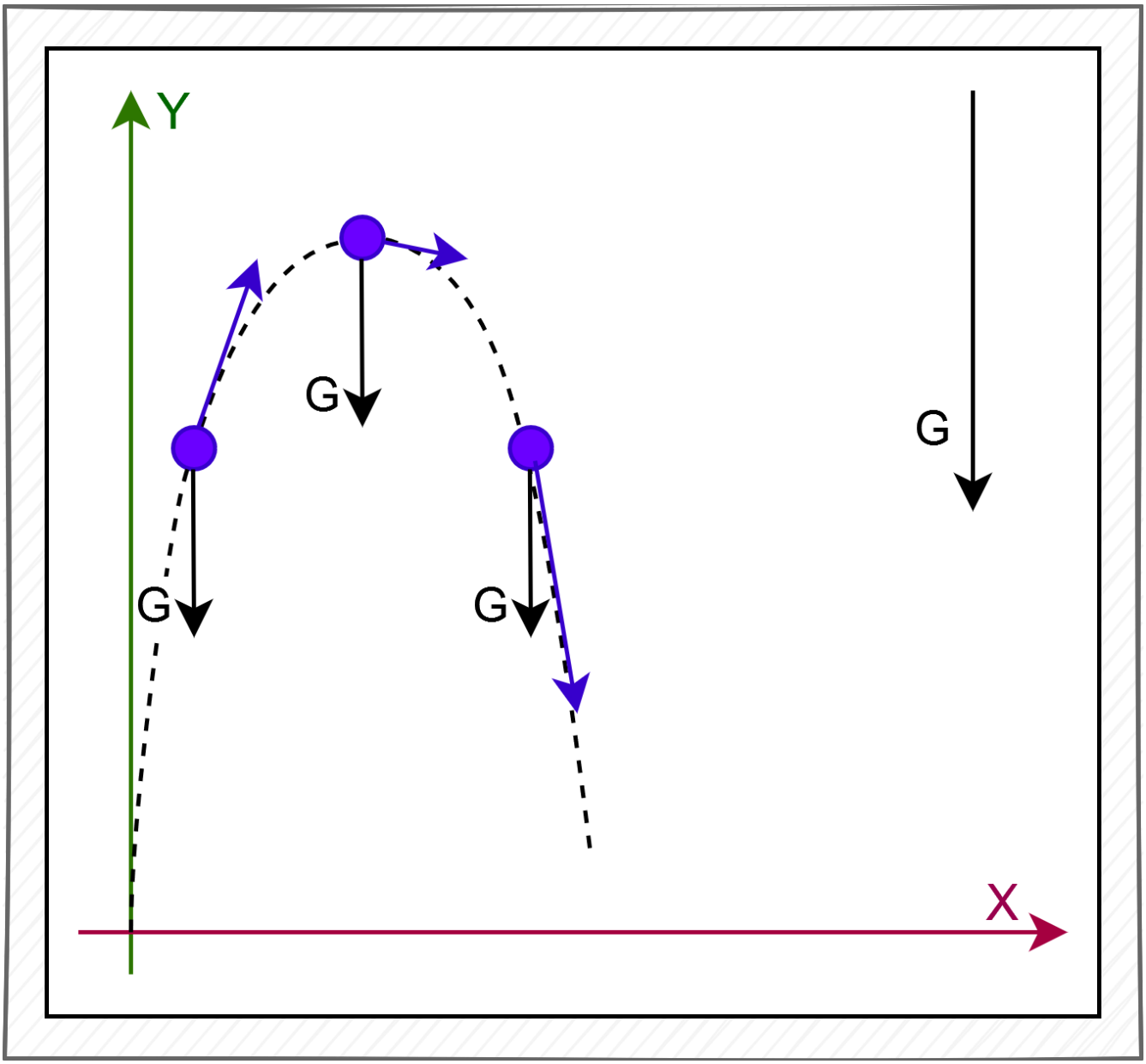

Рисунок 2. Схематичне зображення руху частинки, запущеної з початковим вектором швидкості, направленим вгору.

На рисунку 2 схематично зображено те, як повинна себе вести частина, запущена з вектором швидкості угору, за умови, що вона не піддається ніяким силам, окрім глобальної сили гравітації, направленої вниз. Даний рисунок ілюструє результати виконання коду, наведеного в додатку 1. Як можна побачити

з наведеного коду та рисунку 1, симуляції однієї частинки, котра не піддається жодному динамічному впливу є досить простою задаче, принаймні до тих пір, поки кількість даних частинок не збільшується (а зазвичай стандартна їх кількість при симуляціях може виростати до мільйонів), а також в дію не вступають додаткові сили. (В реальному житті за допомогою частинок симулюють поведінку води, і в даному випадку необхідно враховувати в сотні раз більше параметрів, проте в даному розділі розглядаються базові формули, використовуючи які можна з легкістю перейти до моделювання твердих тіл).

1.2 Динаміка твердого тіла

Розглянувши на приклад симуляції однієї частинки теоретичний базис, необхідний для її симуляції, можна переходити до розгляду теорії, необхідної для симуляції поведінки твердого тіла. Взагалі при роботі з суцільними тілами їх прийнято ділити на дві категорії: м’які тіла та тверді тіла – в даній роботі розглядається використання хмарних та розподілених технологій для обчислення поведінки твердих тіл, як тих, котрі мають адекватну складність теоретичної бази, що дозволяє ознайомитися з нею у виділений проміжок часу. Перш за все слід зауважити, що ми маємо на увазі під твердим тілом.

Тверде тіло – це певний об’єкт довільної форми, котрий володіє тими ж фізичними параметрами, що й частка, проте додатково володіє об’ємом (тобто не є точковим тілом), а також кутом повороту, геометричним центром та центром мас. Положенням тіла прийнято вважати положення його центру мас. Головною ознакою твердого тіла є відсутність деформацій. Тобто **тверде тіло** – це звичайне фізично матеріальне тіло, з однією лише відмінністю, що воно не можу бути деформоване в будь-який спосіб.

Звичайно, такі тіла не існують у реальному світі – навіть найтвердіші матеріали деформуються в тій чи іншій мірі коли до них починають прикладати

деяку достатню за величиною силу – проте тверде тіло є корисною моделлю, припущенням, що спрощує вивчення динаміки твердих тіл, особливо у випадках, коли деформаціями можна знехтувати (а в більшості випадків ними знехтувати можна). Тверде тіло схоже на частинку, оскільки воно також має масу, положення та швидкість. Окрім того таке тіло має об'єм і форму, тому може обертатися навколо осі (або ж осей), що є досить великою проблемою при розрахунках. Це створює додаткові складності, особливо в тривимірному просторі, котрий буде основним простором наших досліджень/розрахунків, проте на даний момент ми сконцентруємося на двовимірному просторі.

Природньо вважати, що тверде тіло обертається навколо свого центру мас, а саме положення твердого тіла визначається положенням його центру мас. Визначимо початковий стан твердого тіла з центром мас у початку координат і нульовим кутом повороту. Тверде тіло можна розглядати як набір нескінченного (у випадку симуляції – кінченого) набору частинок (точок). При симуляціях досить часто прийнято використовувати об'єкти однорідної густини, у таких випадках центр маси припадає на геометричний центр тіла. В загальному випадку прийнято розраховувати центр маси за формулою:

$$cm = \frac{\sum_0^n m_i r_i}{M} \quad (5)$$

, де – $\boldsymbol{m_i}$ це маса частинки $\boldsymbol{i}$ в певній точці з координатами $\boldsymbol{r_i}$, $\boldsymbol{M}$ – загальна маса всього тіла, $\boldsymbol{cm}$ – позиція центру мас. Дана формула вираховує середню позицію всіх частинок тіла, та врівноважує їх за масою. При однорідній густині (а тому і їх однорідній масі) центр мас твердого тіла буде відповідати геометричному центру тіла.

При переміщенні такого тіла під час симуляції слід вважати, що проекції векторів сил, котрі діють на тіла можуть бути прикладені до будь-якої точки тіла, але при прикладанні його до будь-якої точки окрім центру мас виникає обертальна сила та відповідно виникає поворот тіла. При тому нагадується, що центр мас в

даних симуляції прийнято вважати геометричним центром і навпаки. Окремо слід зауважити, що все це залежить від детальності симуляції, і наявність неоднорідних тіл є просто ще одною змінною розрахунку, а не суцільною проблемою. Також окремо зазначаємо, що для систем тіл, все таки центр мас системи буде відрізнятися від геометричного центру. Ілюстрацію переміщення показано на рисунку 3.

Також в формулі 4 зазначено використання суми. В реальному житті при визначення тіла як набору матеріальних точок необхідно використати інтеграл по функції густині від позиція точки в системі координат самого тіла за об'ємом, проте в нашому випадку ми зазначаємо тверде тіло, по-перше однорідним, а по-друге – кінченим набором точок, що дозволяє уникнути даного обрахунку.

При переміщенні твердого тіла виникає не лише зміна координат тіла (як у випадку частинки), а в деяких випадках можливе виникнення повороту тіла. Тому загальна картина руху твердого тіла зображена на рисунку 3.

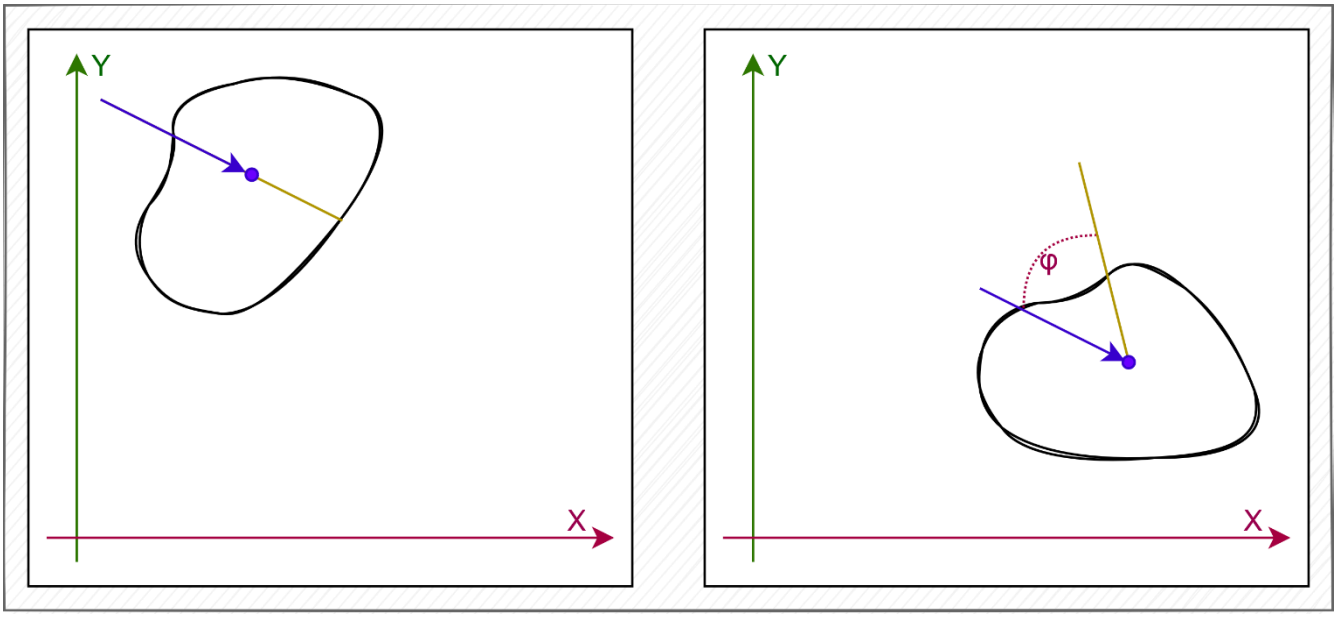

Рисунок 3. Ілюстрація переміщення тіла з поворотом

Насправді, рисунок 3 досить корисний для ілюстрація наступної складової симуляції поведінки твердого тіла – повороту при застосування до тіла сил. Для

цього на рисунок 3 синім показано напрямок прикладання сил, жовтим – фіксована лінія, за якою можна переглядати зміщення. Символом **φ** описується кут повороту тіла. Так як тверде тіло має об'єм, воно може обертатися, тому необхідно ввести також такі поняття як: кутова швидкість, кутове прискорення а також – момент сили. Окрім того, вводимо такий параметр як момент інерції. Момент інерції є скалярною мірою того наскільки важко тіло змусити обертатися навколо осі центра мас (дана величина є відповідником маси при обертальному русі, так як масу можна вважати мірою того, наскільки важко змусити тіло змінити свою позицію в просторі). Вводимо формулу 6:

$$\tau = I\alpha \tag{6}$$

, де **τ** – момент сили (**τ** від англійського – torque), ***I*** – момент інерції, а ***α*** – кутове прискорення. Бачимо, що формула 6 відповідає формулі 4 при лінійному русі за умови, що момент інерції відповідає масі, а кутове прискорення – лінійному прискоренню. Для обрахування моменту інерції існує безліч формул, в залежності від тіла, вони різні і вони є вже попередньо обрахованими.

В додатку 2 наведено код для проведення простої симуляції в двох-вимірному просторі. Для даної симуляції використовувався примітив – квадрат. В даному випадку для квадрат використовувалася вже визначена формула 7.

$$\mathrm{I} = \frac{m(h^2+w^2)}{12} \tag{7}$$

, де ***I*** – момент інерції, ***m*** – маса тіла, ***h*** – висота тіла, ***w*** – ширина тіла. Тепер залишилося вияснити, як завдяки формулі 6 та 7 знайти кутове прискорення, котре необхідне нам для проведення моделювання. Для цього необхідно звернутися до іншого визначення моменту сили, з якого ми дізнаємося, що момент сили є скалярною величиною, котра генерується при прикладанні деякої сили **F (**вектор даної сили визначаємо через ***f***) до довільної точки тіла, координати якої ***r*** є відносними до центра маси. Згідно цього отримуємо, що момент сили можна обрахувати за формулою 8, наведеною нижче:

$$\tau = \bar{f} \times \bar{r} \qquad (8)$$

, де $\boldsymbol{\tau}$ – момент сили, $\boldsymbol{f}$ – вектор сили **F**, $\boldsymbol{r}$ – точка прикладання сили відносно центру маса тіла. Під операцією в даній формулі мається на увазі – векторний добуток векторів. Використовуючи дані формули можна провести симуляцію поведінки твердого тіла. Приклад візуалізації формули 8 для обрахунку моменту сили наведено на рисунку 4.

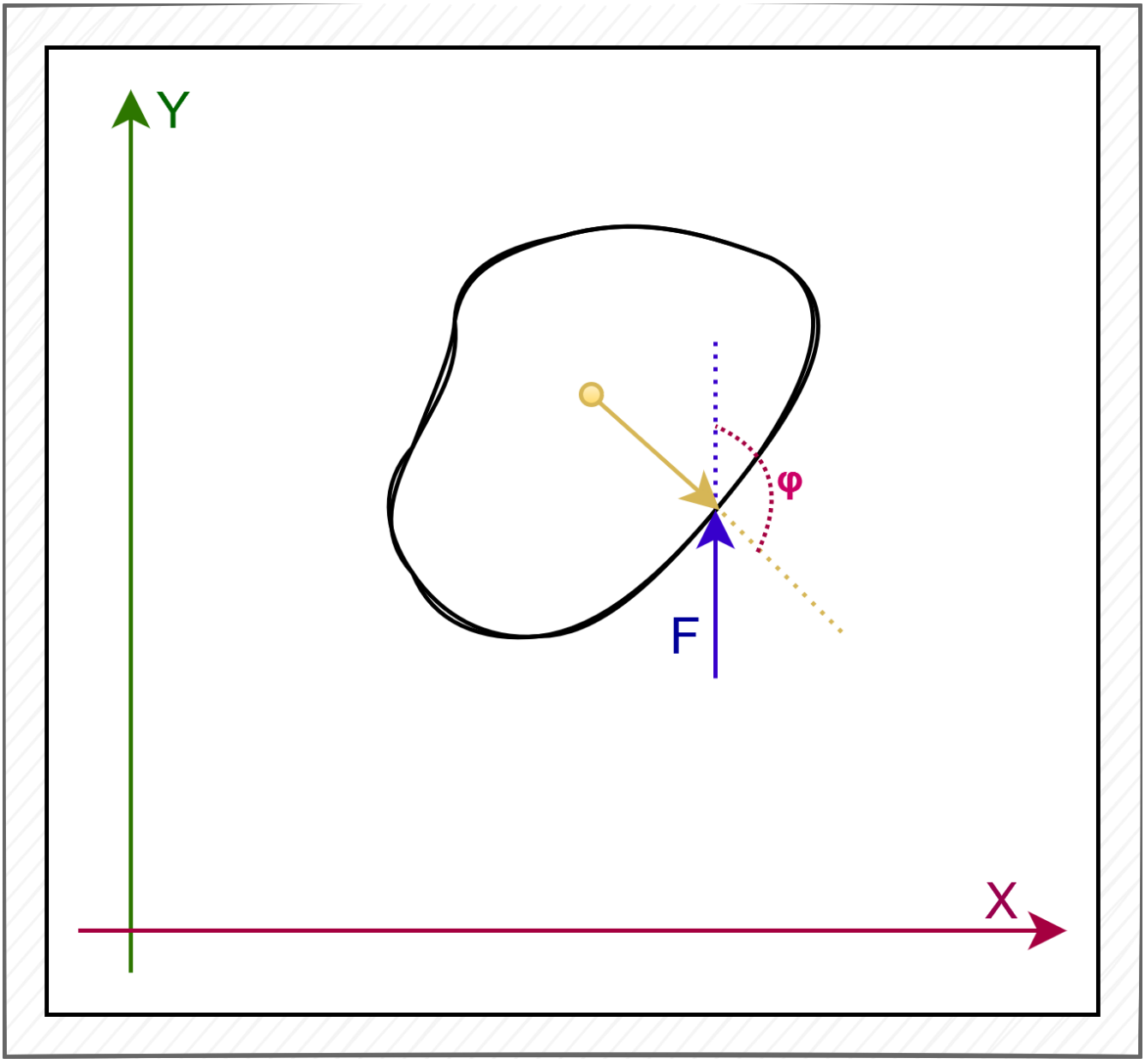

Рисунок 4. Візуалізація формули обрахунку моменту сили

На рисунку 4 чітко видно, що при прикладанні сили **F** до точки з координатами ***r*** у нас виникає обертання навколо центру мас. У випадку тривимірного простору виникає обертання навколо осі центру мас. Приклад простого коду для симуляції поведінки одного тіла під дією гравітації та попередньо заданим вектором швидкості наведено в **додаток 2**.

Двовимірне моделювання можна розглядати як тривимірне моделювання, де всі тверді тіла є тонкими та плоскими, і все це відбувається в площині ***xy***, тобто немає руху по осі ***z*** Дане припущення означає, що вектори ***f*** та ***r*** завжди знаходяться в площині ***xy*** і тому вектор ***τ*** завжди матиме нульові компоненти ***x*** і ***y***, оскільки векторний добуток виливається вектор, котрий буде перпендикулярним до площини ***xy***. Це, у свою чергу, означає, що вона завжди буде паралельна осі ***z***. Таким чином, має значення лише **z**-компонент перехресного добутку. Це призводить до того, що розрахунок крутного моменту в двох вимірах можна спростити до формули 10:

$$\tau = r_x f_y - r_y f_x \tag{10}$$

, де ***τ*** – момент сили, $\boldsymbol{r_x}$ та $\boldsymbol{r_y}$ це відповідні ***x*** та ***y*** компоненти вектору ***r***, $\boldsymbol{f_x}$ та $\boldsymbol{f_y}$ це відповідні ***x*** та ***y*** компоненти вектору ***f***.

Варто зауважити, що додавання лише одного додаткового виміру до симуляції значно підвищує складність моделювання. У тривимірному просторі положення (позиція та нахил) тіла має бути представлена за допомогою кватерніонів, котрі є чотириелементними векторами комплексних чисел. Момент інерції в даному випадку буде представлений матрицею розмірності **3** на **3**, яка називається тензором інерції і не є постійною, оскільки залежить від орієнтації твердого тіла в просторі, а тому змінюється з часом по мірі обертання тіла навколо осі.

1.3 Знаходження колізій

Розібравши як можна досить просто просимулювати поведінку одного твердого тіла в просторі переходимо до більш складних питань симуляції – взаємодії твердих тіл. Найпростішим прикладом взаємодії двох тіл є їх стикання один з одним. Даний процес зіткнення двох об’єктів в термінології прийнято називати **колізією**. Симуляції та знаходження колізій є одною з найважчих задач (уступає хіба що її вирішенню), і на це є декілька причин:

- Першою проблемою є мінімальна алгоритмічна складність пошуку колізій, кота складає собою $\mathbf{O(n^2)}$, де n – кількість предметів на сцені, що симулюються. Дана алгоритмічна складність не може бути зменшена, адже ми повинні перевірити всі пари об'єктів на наявність колізій. Всі оптимізації відбуються скоріше на зменшення числа n, завдяки використанню розвинутих структур опису сцен, а також введення границь симуляції.
- Наступна проблема – можлива складність та різноманітність форм твердих тіл, що підвищує складність при знаходженні перетину між твердими тілами. Саме вирішення даної проблеми є одним з основним напрямків оптимізації та вирішення колізії.

Для вирішення проблеми знаходження колізій в об’єктах зі складними формами використовуються підхід з використанням **широкої** та **вузької** фаз зіткнення. На етапі широкої фази знаходяться потенційні пари зіткнень, на етапі вузької фази знаходяться об’єкти, котрі дійсно переткнулися, а також точки контакту даних об’єктів.

Детальний опис даних фаз та підходів будо розглянуто в пунктах 1.3.1 та 1.3.2. Схематичне зображення різниці між широкою та вузькою фазами знаходження колізій показано на рисунку 5а та 5б. З рисунку 5 видно, що до вузької фази переходять лише ті об’єкти, котрі були визначені як такі, що потенційно зіткнулися (на рисунку позначені червони, об’єкті B0 та B1). Дана

оптимізація виглядає природньою, особливо за уваги, що в реальному використанні об’єкти на сцена можуть вираховуватися десятками тисяч, і обробка вузької фази теж є досить важкою операцією з власною складністю.

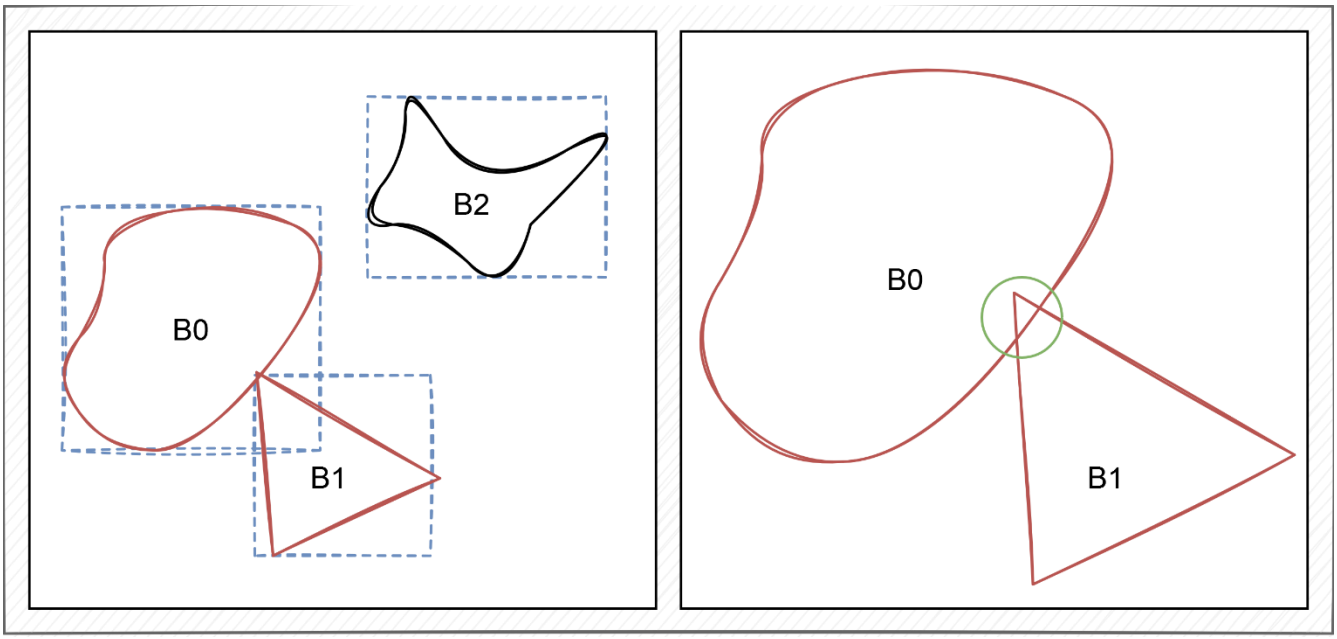

Рисунок 5. а (зліва) – схематичне зображення широкої фази, б (справа) – схематичне зображення вузької фази

1.3.1 Широка фаза при знаходженні колізій

Широка фаза відповідає за пошук потенційних пар об’єктів, котрі перетинаються один з одним. На даному етапі використовуються не справжні форми об’єктів, а так звані обмежувальні об’єми (Bounding volume). Дані об’єми повністю включає в себе тіло. Існують різні видм даних об’ємів, найпопулярніший з них **OOBB**, котрий має форму прямокутника, а також **BS** (Bounding sphere) – котрий має форму сфери. Також інколи використовують **AABB** котрі є обмежувальними коробками вирівняними за осями простору (Axis Aligned). Використання таких об’ємів, як було розглянуто раніше, зумовлена тим що якщо об’єми тіл не перетинаються, то очевидно, що не перетинаються і самі тіла. Слід також зауважити, що всі дані об’єми будуються з припущенням, що вони є

мінімальними за розміром, аби гарантувати ефективність та зменшити кількість хибних тестів на колізію. Візуальне представлення об’ємів відображено на рисунку 6.

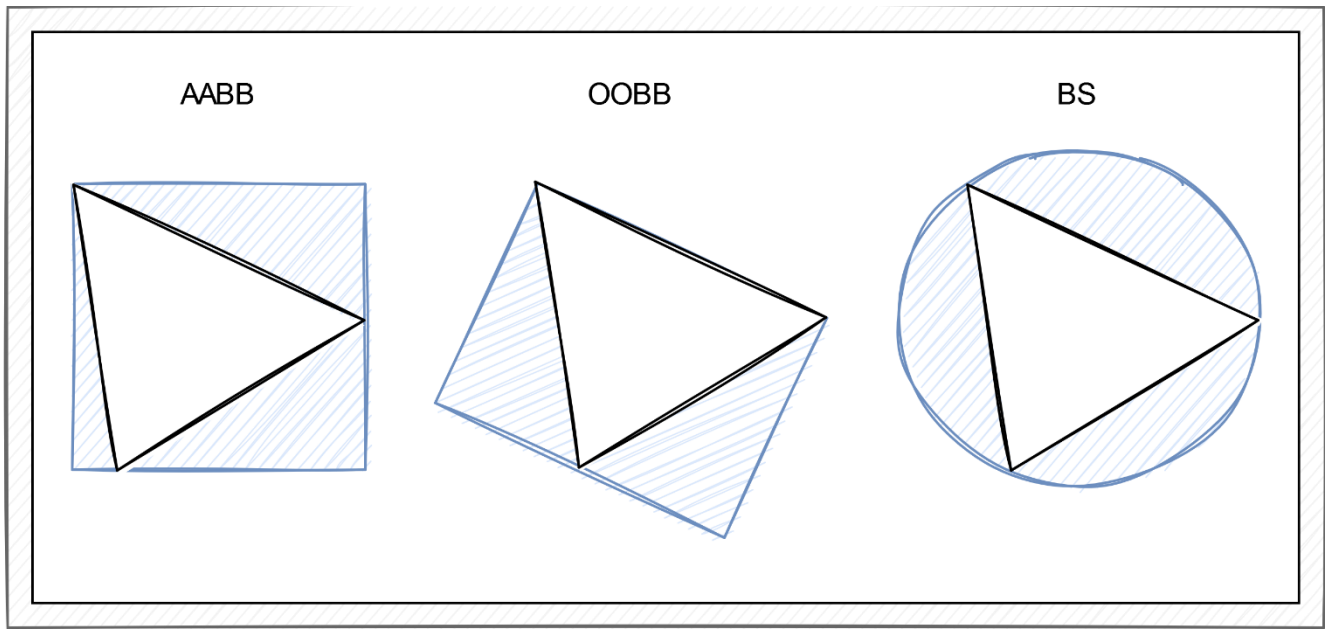

Рисунок 6. Схематичне зображення різних видів об’ємів у двовимірному просторі. Зліва направо: OOBB, AABB, BS

Найбільш вживаними об'ємами є OOBB та AABB, сфера ж використовується досить рідко, зазвичай в цілях відображення, тобто в рендер рушіях. Вподобання до даних об’ємів обумовлюється такими факторами:

– Ці об’єми надають нам досить гарний спосіб оптимізувати пошук колізій.
– Опис даних об’ємів займає дуже малу кількість пам’яті в ЕОМ. В залежності від фігури це можу бути як дві точки в просторі та початкова позиція (AABB та OOBB), так і просто значення радіусу й центр кола (BS).
– Пошук перетину цих об'ємів є досить простою задачею, котра зводиться до вирішення відомих аналітичних формул (вирішення рівняння кола, або ж пошук перетину проекцій прямокутників)

Звичайно, існують інші алгоритми, котрі дозволяють оптимізувати пошук даних перетинів. На даний момент слід зауважити, що важливу роль відіграє

ієрархія об’єктів на сцені та структури даних для їх утворення. Головна з них **DBVT** (Dynamic Bounding Volume Tree). Дана структура даних дозволяє описувати одним об’ємом набір вкладених тіл.

1.3.1.1 Пошук колізій в широкій фазі

Найпростішим алгоритмом, особливо для двовимірного простору є алгоритм **Sweep and Prune.** Суттю даного алгоритму є створення проекції кожної площини OOBB і подальша перевірка, чи перетинаються проекції об’ємів. Об’єми, чиї проекції перетинаються у всіх площинах передаються до вузької фази. Даний оптимізаційний алгоритм є досить простим для використання у будь якому просторі, проте, звичайно ж, простішій для виконання у двовимірному. [1, 2] Приклад роботи даного алгоритму показано на рисунку 7.

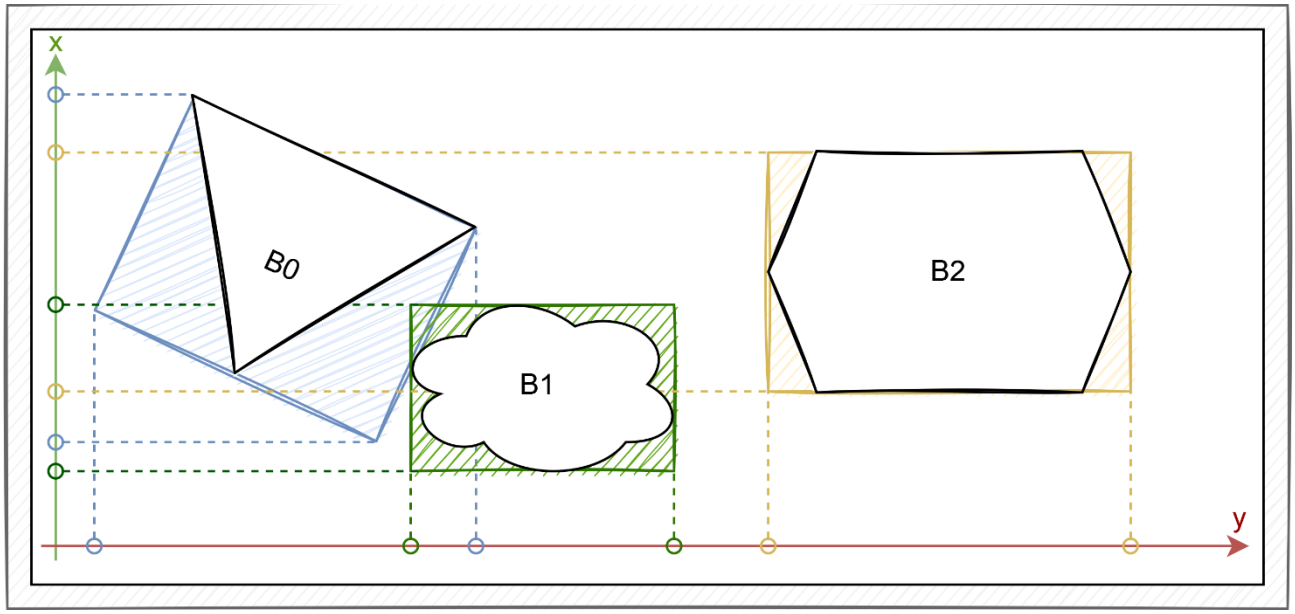

Рисунок 7. Візуальне зображення проекцій за алгоритмом Sweep and Prune

Як і будь який алгоритм, Sweep and Prune чутливий до орієнтації тіла в просторі (саме на даному алгоритму помітно, що використання BS є більш простішим з точки зору обрахунків, проте об’єм BS може буде набагато більшим, ніж OOBB у відповідному тілі). Не дивлячись на те, що Sort and Sweep алгоритм є дешевим при перевірці пари об’єктів, в загальному він не сильно покращає

ситуацію, адже при наївному перебір всіх пар елементів рушій все одно будемо проводити попарне тестування для кожної можливої пари на сцені симуляції, тому навіть з використанням дешевого алгоритму тестування потенціальних колізій, ми все ще маємо небажану тимчасову складність $\mathbf{O(n^2)}$. Тому подальшим продовженням даного алгоритму є додавання структуризації даних в просторі, для цього ми вставляємо всі точки початку та кінця проекцій на вісь (див. рисунок7) в один список і сортуємо його за зростанням. Потім ми обходимо кожен з цих списків (кількість списків залежить від кількості вимірів простору). Обходячи даний список можна нехитрим чином визначити, які інтервали перетинаються, для цього досить знайти дві послідовні точки, котрі не відносяться до одного й того ж самого обмежуючого об'єму. В подальшому даний список отриманих перетнутих інтервалів можна повторно використовувати на кожному кроці моделювання.

В результаті виконання алгоритму зі вказаними вище оптимізаціями ми отримуємо зменшену кількість тестування перетину обмежуючих просторів по кожнйі осі, що дозволяє виконувати алгоритм ефективніше ніж за $\mathbf{O(n^2)}$, так як:

- Тестування кожної наступної осі є набагато простіше, так як проекція вже можу бути відкинути за результатами тестування іншої осі.
- З правильно підібраним алгоритмом сортування можна виконати оптимізації з пошуком по нижній границі.

Для мінімізації кількості тестів перекриття один одним обмежуючих об'ємів було створено безліч алгоритмів, котрі працює за принципом індексації та кешування тіл в просторі. Дані структури даних мають принцип, схожий до звичайних, індексів в базах даних, що використовуються для прискорення запитів читання та запису. Проте, у випадку фізичного моделювання все набагато складніше, адже ми маємо справу з високою волатильністю даних, а також з високою швидкістю оновлення, так як:

- Незважаючи на те що тіла тверді – вони можуть таки змінити свою форму за іншим, не фізичних обставин.
- Обмежуючі об’єми також можуть змінитися. Особливо ця проблема стосується AABB, котрі можуть змінювати свій об’єм в залежності від повороту тіла, OOBB від такої зміни відносно орієнтації – захищені, так як створені вже відного осей об’єкту.
- Об’єкти можуть змінити свій обмежуючий об’єм за рахунок рухомих частин (приклад – людина та руки, якщо об’єкт буде виконувати якусь анімацію, загальний об’єм всього тіла буде змінений).

Також, існують оптимізовані варіанти алгоритму Seep and Prune алгоритмів, один з них основний на комбінації алгоритму Хоара та звичайного розглянутого вище алгоритму Sweep and Sort. [3, 4] В такому випадку даний алгоритм дійсно зменшує кількість тестів, необхідних для повного обходу всіх елементів, так як деякі елементи можуть бути просто відкинуті після поділу сортованого масиву по нижній межі проекції осі.

1.3.1.2 Прискорюючі структури даних та алгоритми розбиття простору

Як було згадано раніше, одним зі способів оптимізувати тестування в широкій фазі є використання певних індексних структур даних. Це досить обширна тема, котра варта окремої роботи та досліджень, тому зараз буде надано перелік окремих типів, котрі є найбільш популярними, та основне правило, котре повинно бути врахованим, при створенні розбиття простору.

Використання кожної окремої структури даних пов'язано з ризиками, перевагами та недоліками. В розбиванні підпростору найкращий підхід залежить від конкретного використання та конкретного типу симуляції, але суть полягає в тому, що метою правильного (з точки зору досвіду) розбиття є досягнення критеріїв:

- Кожен об’єкт має знаходитися рівно в одному підрозбитті (елемент сцени не повинен одночасно знаходитися в двох частинах простору, бути в суперпозиції)
- Кожне підрозбиття простору повинно бути досить великим, щоб тіло в одному підрозбитті могло стикатися тільки з тілами в тому ж самому, або ж в сусідньому до нього підрозбитті.
- Кількість тіл у певному підрозбитті простору повинна бути якомога меншою.
- Кожне підрозбиття простору за об’ємом повинно бути наскільки великі, щоб вміщувати найбільший можливий об’єкт у симульованому світі.

Спосіб же досягнення згаданих вище критерії залежить від інших параметрів, коли частіше за все визначаються самою симуляцією та тілами в ній, критерії наступні:

- Скільки в сцена знаходиться модельованих тіл.
- Як та яким чином вони рухаються.
- Які вимоги до продуктивності системи в цілому.
- Дедлайнами по роботі над симуляцією (складні алгоритмі вимагають багато досить специфічних знань та часу на імплементацію).

Як приклад слідування за даними правилами наведу наступні приклади:

- Якщо тіла симулюються у відкритому просторі (open world), найпростішим рішення буде розділити сцену шаховою (квадратною) сіткою на сектори, де кожна клітинка (сектор), як мінімум буде більшою за найбільший симульований об’єкт, а потім симулювати поведінку об’єктів в кожні клітинці окремо.
- Якщо мова йде про великий відкритий світ (зазвичай маються на увазі світи де симульованих тіл більше ніж 64 тисячі, або більше максимального значення 16-ти розрядного беззнакового цілого числа), така проста сітка

стане досить дорогою та неефективною, що приведе до падіння швидкодії системи (зменшенню FPS), саме в такому випадку необхідно починати задумуватися не тільки про розбиття у вигляді сітки, а і про спеціальні прискорюючі структури даних. Використання конкретної структури даних залежить вже від власних побажань та специфіки сцени.

Нижче наведено список структур даних, котрі можна використати для розбиття простору при симуляції фізики:

- BSP дерево
- K-D дерево
- R-дерево
- Квадро-дерево
- Окто-дерево
- Кошики (Bins)
- BVH

Взагалі, BVH це не якийсь конкретний спосіб поділу простору, це певний клас алгоритмів та структур даних, котрі допомагають упорядкувати обмежуючі об’єми, що дозволяє легше з ними працювати. Візуальне зображення розбиття деякого простору з обмежуючими об’ємами та візуальне зображення відповідного розбиття у вигляді BVH зображено на рисунку 8.

BVH дерево може бути будь-якої довжини, а також не обмежуватися тільки бінарним виглядом (тобто в кожному вузлі може бути безкінечно велика кількість предметів), проте все ж таки основним правилом є якомога менший об'єм кожного вузла (сектору простору), проте з відсутністю пустих секторів.

Також, бажано, щоб кількість елементів в кожному вузлі/секторі була близькою до однорідної. У випадку використання BVH дерева тестування на колізію відбувається лише в тих елементах, котрі знаходяться в одній гілці дерева. Досить ефективним в такому випадку є використання збалансованих дерев.

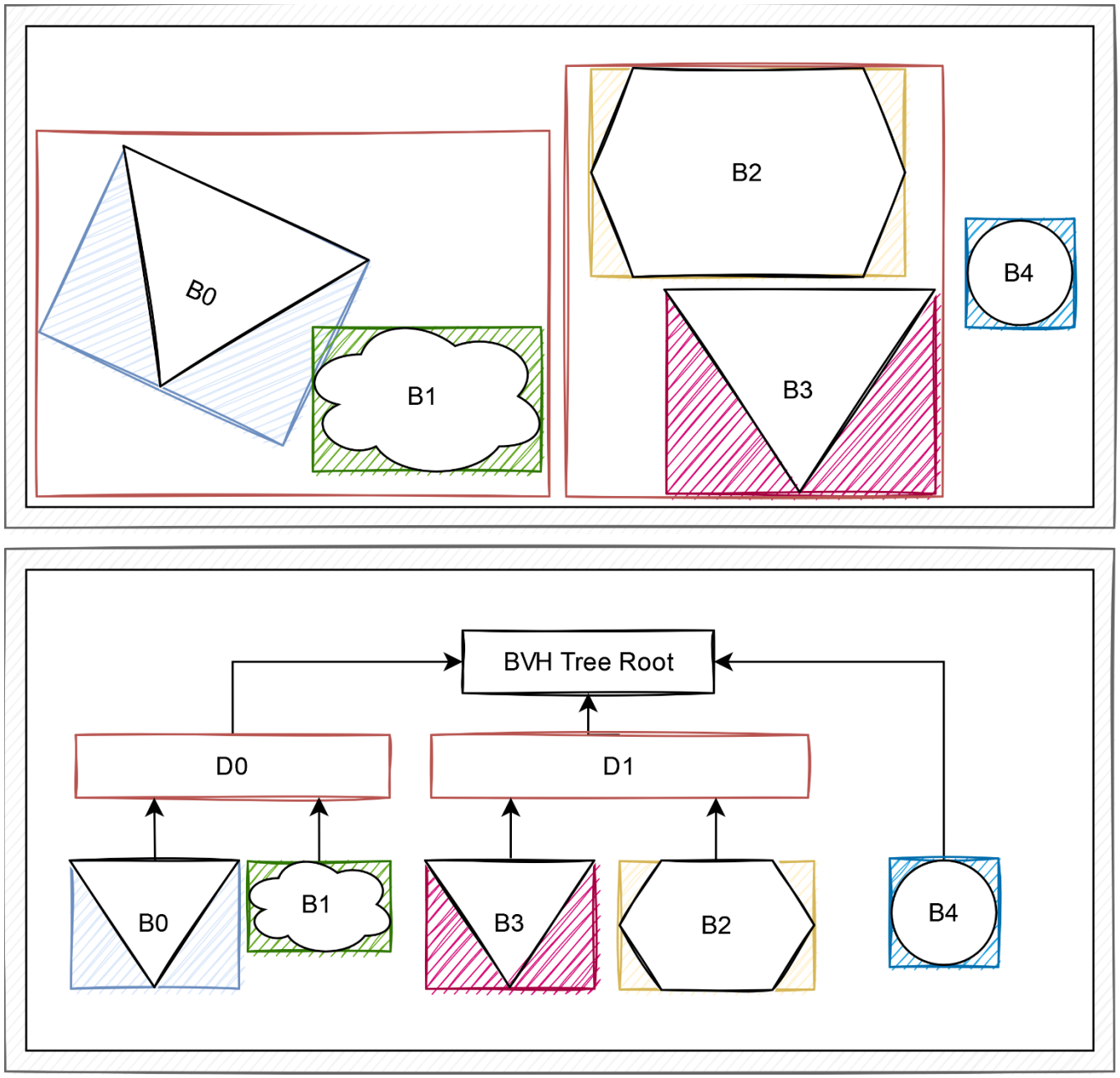

Рисунок 8. а – положення об’єктів в просторі, б – схематичне розбиття об’єктів в BVH структурі даних.

1.3.2 Вузька фаза при знаходженні колізій

Наступною фазою після виконання пошуку потенційних пар є встановлення фінального результату колізії: чи зіткнення таки відбулося, та в яких точках об’єкти перетнулися. Після широкої фази тестування отримуємо список пар

потенційно зіткнувшихся об’єктів, кожну з яких треба перевірити на зіткнення, знайти точки перетину, а також – вирішити, як саме будуть вести себе об’єкти після зіткнення один з одним.

На що потрібно звернути увагу – на відміну від обмежуючих об’ємів, котрі в загальному мають невеликий набір типів, справжній вигляд тіла може приймати безліч форм. При пошуку колізій нас цікавить одна основна властивість тіла, це тип її форми, котрий може бути опуклим, або ж увігнутим. Дані поняття розглядаються в курсі шкільної геометрії, тому детальний опис цих понять ми опустимо, а перейдемо до застосування даних понять під час симуляцій. Перш за все, бажано, що всі фігури в процесі симуляції були саме опуклими, так як найефективніші та досконалі елементи перевірки перетину тіл працюють саме з опуклими формами (ці методи в абстрагованому вигляді проходяться в загальному курсі математики в університеті).

На даному етапі вводимо поняття суцільної оболонки (convex hull). Суцільна оболонка – це мінімальна за розміром множина, котра містить у собі всі розглянуті (обрані) точки і має опуклу форму. В нашому випадку, набір точок – тіло. Тобто суцільна оболонка у випадку фізичного рушія – мінімальна опукла форма, котра покриває все тіло. Очевидно, якщо опукла оболонка буде використовуватися опуклу для представлення увігнутого тіла то оболонка (і саме тіло) втратить властивості увігнутої форми. З втратою тілом властивостей може виникнути ряд візуальних та математичних проблем, основна дана проблема – позитивно-негативні зіткнення об’єктів, коли візуально очевидно, шо вони не повинні стикатися (об’єкт все одно матиме вірне графічне представлення, так як для рендеру увігнуті форми не є проблемою, це той же самий набір полігонів).

В цілому, самих по собі опуклих оболонок буває досить для представлення будь-яких об’єктів, так як випуклість малих об’єктів може бути непомітною та бути зведеною до математичної помилки. Проте існують випадки, коли необхідно

або увігнуте тіло поводилося коректно і візуальні та математичні помилки не можуть бути проігноровані. Особливо це буде помітно у випадках, коли симулюється поведінка контейнерів, або будь яких інших порожніх форм. У такому випадку прийнято використовувати декомпозицію опуклої форму

Опукла декомпозиція працює наступним чином – вона приймає на вхід увігнуту форму, а на виході повертає набір опуклих форм (та їх параметрів), котрі в сумі своїй утворюють увігнуту форму. Просто опукла форма та декомпозиція увігнутої форми показано на рисунку 9.

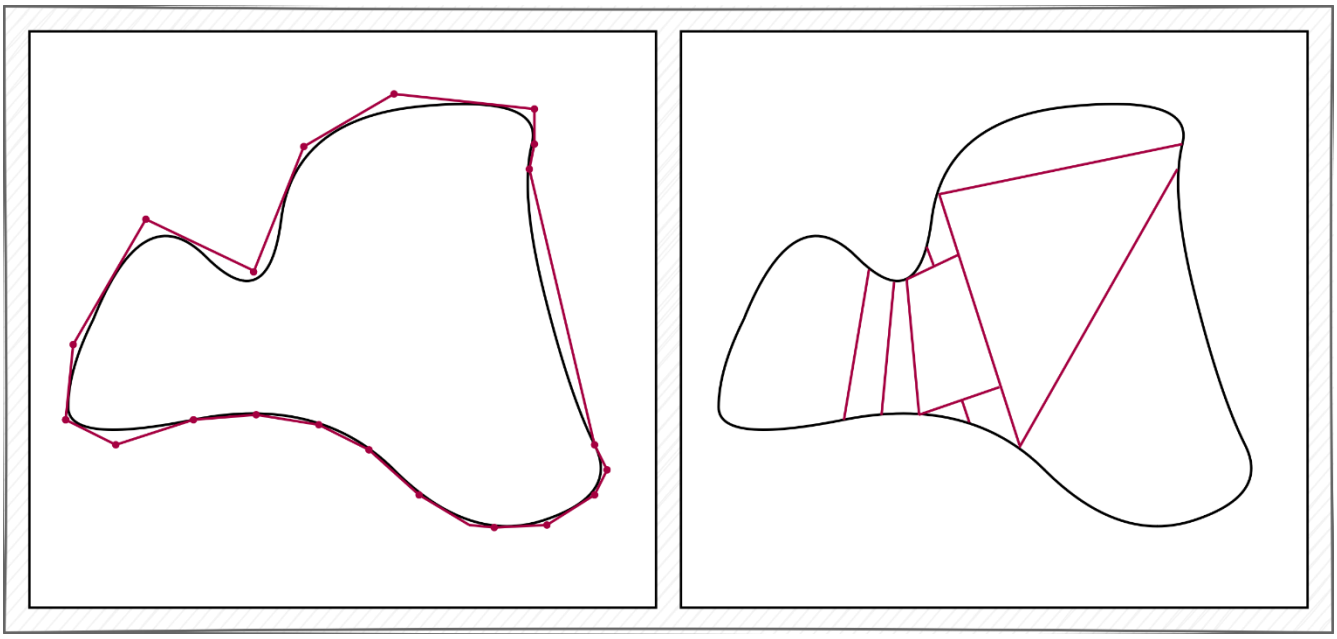

Рисунок 9. а – опукла форму для увігнутої фігури, б – розбиття увігнутої форми

Складність використання даних алгоритмів є досить різною, в залежності від кривизни увігнутої форми, а також від бажаної точності, проте, в загальному є два підходи:

– Автоматизований – алгоритми самі нарізають форми на опуклі форим.
– Ручний – відповідна людина сама проводить нарізку.

В будь-якому випадку, даний процес є може бути оптимізований, а саме розбиття може бути запечене ще на етапі створення тіла, тому основним навантаження є підвищена вартість перевірки на зіткнення. Далі розглянемо

алгоритмі та підходи, необхідні для знаходження колізій у вузькій частині, основними та популярними алгоритмами є GJK, SAT, а також окремі підходити до пошуку CCD (частіше за все – фантомні кроки).

1.3.2.1 Алгоритм SAT

Перш за все слід зауважити, що SAT це не в повній мірі алгоритм, а теорема, проте дана теорема є фундаментом та основою на якій тримається основний алгоритм знаходження перетину між об’єктами. Формально, теорема про вісь поділу (separating axis theorem) стверджує, що пара опуклих форми не перетинаються тоді і тільки тоді коли існує хоча б одна вісь, ортогональні проекції фігур на котру не перетинаються один з одним. Візуалізація ідеї, котра стоїть за даним визначенням можна побачити на раніше представленому рисунку 7.

Варто зауважити, що на відміну від перевірки перетину по конкретним осям, дана теорема стверджує про існування хоча б однієї такої осі, тому задачі алгоритму ускладняється не до перевірки проекцій, а до знаходження осі (котрих є нескінченна кількість), на якій проекції не перетинаються, принаймні якомога раніше. Таким чином, знаходження такої осі якомога швидше грає вирішальну роль для оптимального використання SAT.

Практично та емпірично було знайдено, що спектр всіх осей можна звести до кінцевої кількості, використовуючи в якості осей розподілу нормалі площин фігур, очевидно, що дана множина є конечною (у випадку кіл метод SAT не використовують, хоча і там даний підхід буде працювати). Візуалізація даного підходу зображена на рисунку 10.

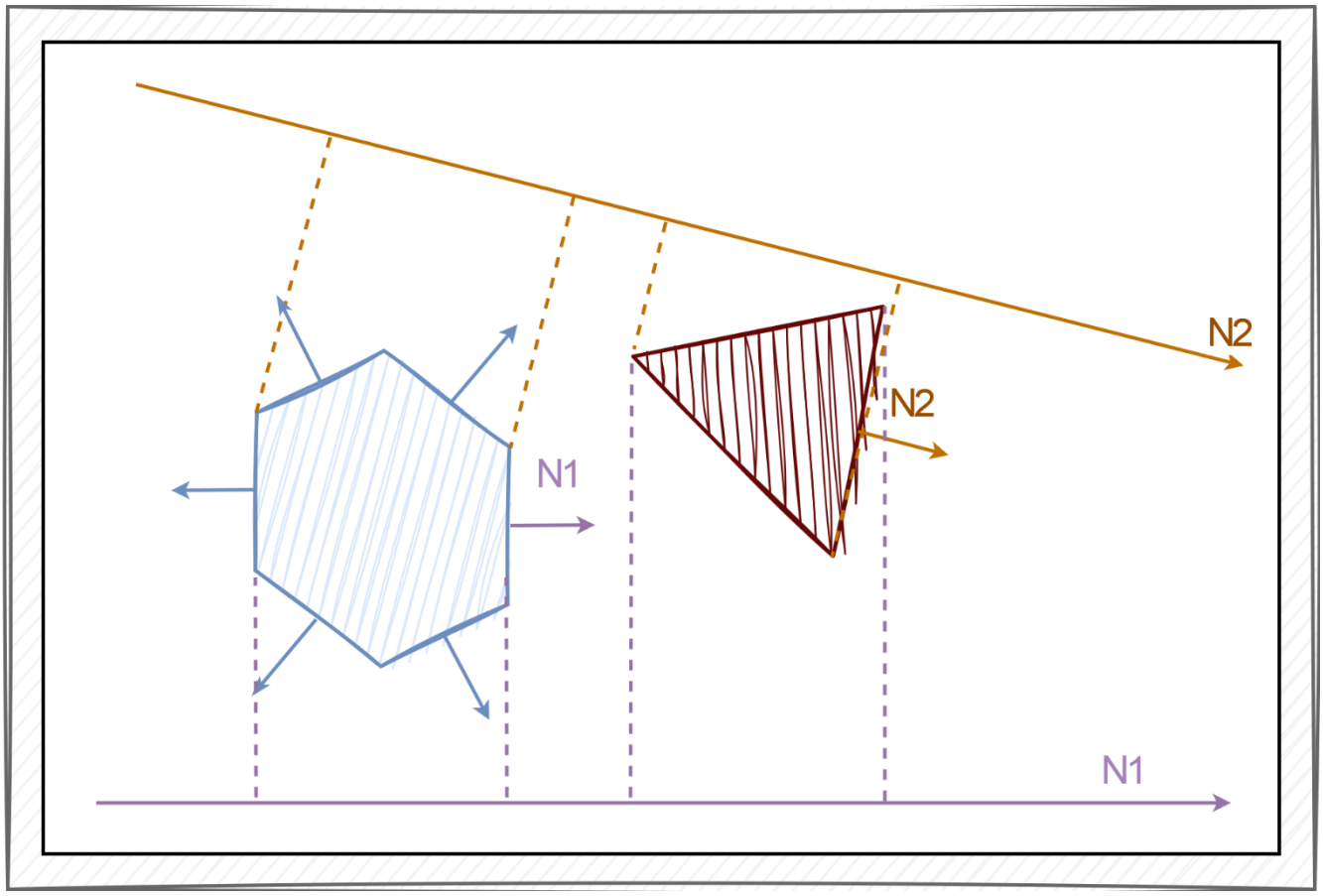

Рисунок 10. Вибір осей за нормалями ребер у двовимірному просторі.

З рисунку 10 видно, що підбір осей можна робити за нормалями поверхностей будь якого тіла з пари. Прагматично використовувати спочатку ой об’єкт, у якого осей – менше. На рисунку 10 показано пару тіл і підбір осей N1 та N2 з нормалей об’єктів N1 та N2 пари тіл. Видно, що проекцій тіл на осі не перетинаються, тому для алгоритму тіла перетинатися не будуть, візуально також не бачимо даного підтвердження, тому все правильно. Також видно, що за деякими осями перетин би відбувся, як наприклад – взяття будь-якої іншої осі трикутника.

Було згадано, що краще починати підбір з тіла, де ребер/площин менше, але починати підбір з тіла де їх більше також може бути корисно, так як менша кількість ребер збільшує ймовірність того що вісь таки матиме перетин проекцій.

Окремо варто зазначити випадок перетину кіл. В даному випадку можна використати алгоритм SAT, проте пошук осей буде дещо складнішим, так як дуже складно знайти початкову вісь, від якої можна відштовхнутися. В такому випадку більш ефективнішим є просто знаходження відстані між центрами кіл та перевірка, чи ця відстань більша ніж сума радіусів двох тіл.

1.3.2.2 Алгоритм GJK

Симуляція неперервного середовища є проблемною в багатьох аспектах (CCD та наступний розділ, одна з таких проблем – технічна обмеженість точності ЕОМ, на котрій відбувається симуляція. Сучасні ЕОМ дозволяють отримувати досить точні результати, проте все ж таки, досить часто вони є недостатніми, або ж вимагають великої затрати ресурсів, в даному випадку прийнято встановлювати граничне значення, при якому відбувається завершення розрахунків – точність обчислень.

У випадку колізій таке значення теж є і описує воно відстань між об’єктами. Попередньо розглянутий SAT дозволяє отримати точне уявлення, чи об'єкти перетнулися, проте він не дає уявлення про те, де саме вони перетнулися, та яка реальна відстань між об'єктами (а також координати найближчих точок). Для вирішення даних проблем було винайдено алгоритм GJK (від прізвищ авторів Гілберт-Джонсос-Кірт), який тут і розглянем, та який є одним з найпопулярніших варіантів вирішення даної проблеми. [4, 5] Класично, при роботі з GJK алгоритмом використовується три основні поняття:

- Опорний вектор та опорна функція.
- Симплекс.
- Сума мінковського.

Використаємо визначення опорного вектора та опорного співвідношення (support mapping), котре надається при розгляданні алгоритму GJK. [5] Опорна функція S (від слова support) від двох аргументів: вектору $\boldsymbol{v}$ та певного набору

точок **A** (точки описують тіло) – котра повертає точку ***a*** з набору **A**, відстань якої до ***v*** є найбільша та ненульова (або ж ще кажуть, що скалярний добуток вектору ***v*** та точки ***a*** є найбільшим). Функція в такому випадку називається опорною, операція називається опорним відображенням, а точка – опорою вектора ***v***. Візуальне зображення показано на рисунку 11.

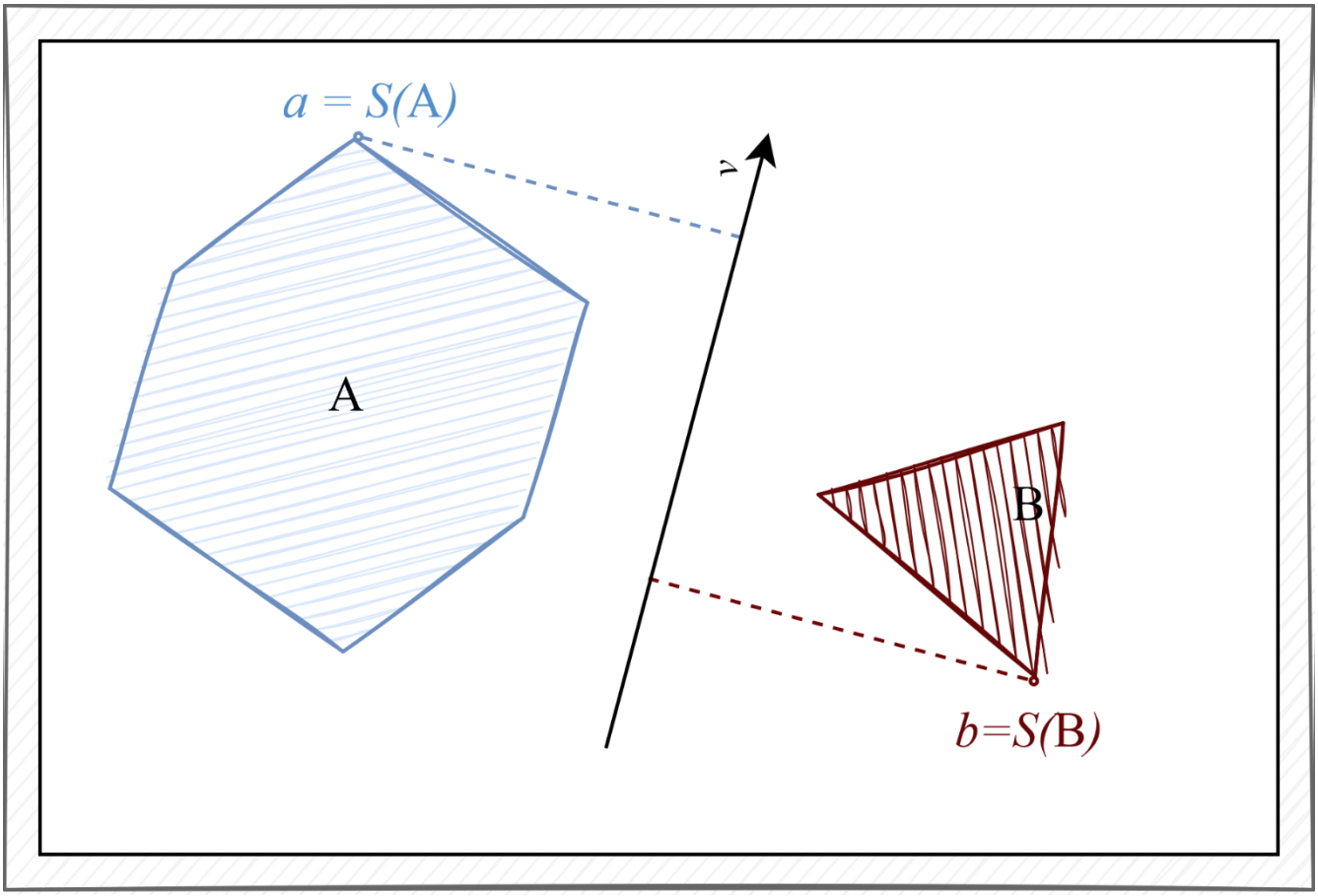

Рисунок 11. Приклад розташування опорних точок вектора ***v*** відносно тіл (множин точок) A та B.

На рисунку 11 тіла **A** та **B** представлені як набір точок (тіла для ЕОМ все ще є дискретним набором ліній та вершин в більшості своїй), тому опорні точки ***a*** та ***b*** відносно вектора ***v*** представлені як функція від набору точок ***S(A)*** та ***S(B)***. Найбільшим же досягненням алгоритму GJK є можливість обраховувати відстань між складними фігурами, відстань між якими неможливо би було порахувати за

звичайних умов без великої затрати ресурсів ЕОМ. В основному дане питання стосується плавним та округлих форм, котрі у звичайному випадку прийшлось би дискретизувати. Дискретизація ж у свою чергу привела би до ряду проблем з реальному фізичною поведінкою даних об’єктів, оскільки б:

1. У випадку низької дискретизації форма об’єкту суттєво би змінилася, що привело би до зміни його поведінки під час моделювання.
2. У випадку високої дискретизації форма об’єкта було б приближеною до реальної, проте висока кількість полігонів (у двовимірному просторі – ребер, але притримуємося однієї термінології) привела би до збільшення вартості обрахунку відстані відносної кожної можливої дискретної частини поверхні тіла.

Більшість способів для знаходження відстані між об’єктами уже давно знайдено то розглянуто в наукових роботах поважних дослідників та інженерів ПЗ. [3 – 5]

Сума та різниця Мінковського це бінарне відображення набору суми та різниці всіх точок двох наборів в інший набір точок, котре визначається за формулами 11 та 12, наведеними нижче:

$$A + B = \{a + b : a \in \mathrm{A}, b \in \mathrm{B}\} \quad (11)$$

$$A - B = \{a - b : a \in \mathrm{A}, b \in \mathrm{B}\} \quad (12)$$

, де $+$ – оператор суми Мінковського, $-$ – оператор різниці Мінковського, $\boldsymbol{a}$ та $\boldsymbol{b}$ – довільні точки з відповідних наборів точок **A** та **B**. Також варто зазначити, що операція суми Мінковського є комутативною, в той час як операція різниці – ні (так як сама операція різниці не є комутативною). Графічне зображення даних операцій показано на рисунку 12.

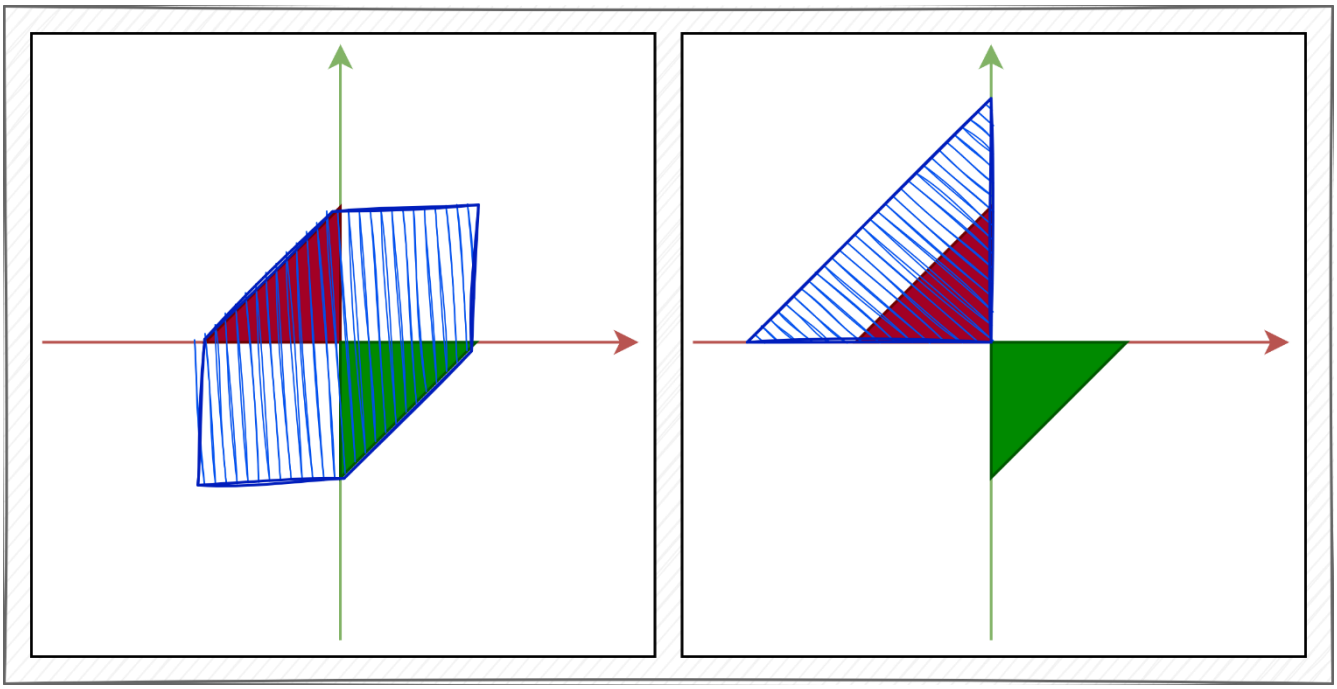

Рисунок 12. а – сума Мінковського та б – різниця Мінковського.

На рисунку 12 для простоти представлення показано суму Мінковського для двох симетричних трикутників. Сума представлена у вигляді суми Мінковського A та B, різниця також – A та B, у випадку виконання різниця у порядку B – A ми отримаємо значення відповідне до суми в порядку A та B. Також у суми Мінковського є три важливі властивості, котрі є ключовими при її використанні:

- Якщо A та B опуклі, то результуюче тіло **суми** Мінковського також опукле.
- Якщо **різниця** Мінковського включає в себе початок координат, то тіла-аргументи даної різниці – перетинаються. Теоретично це доводиться тим що якщо тіла перетинаються, то у них буде принаймні одна точка з однаковими координатами, що при відніманні приводить до того, що результатом становиться саме точка початку координат (0, 0).
- Якщо **різниця** Мінковського не включає в себе початок координат системи, то мінімальна відстань від початку координат та результатом різниці є відстань між тілами-аргументами даної різниці.

Треба згадати, що відстань між двома тілами визначається через формулу 13, наведену нижче.

$$D(\mathrm{A},\mathrm{B}) \coloneqq min\,\{\|a-b\|: a \in \mathrm{A}, b \in \mathrm{B}\} \quad (13)$$

, де ***D*** – символ відображення відстані, ***a*** та ***b*** – довільні точки з відповідних наборів точок **A** та **B**. Як видно з визначення відстані можна постановити твердження 14, що:

$$D(\mathrm{A},\mathrm{B}) \in A - B \quad (14)$$

, де *D(*A, B*)* – відстань між двома тілами, а $A - B$ – різниця Мінковського між цими тілами. З твердження бачимо, що дана відстань входить до різниці Мінковського двох тіл та є мінімальною. Проте проблема полягає в тому що знаходження різниці Мінковського не є тривіальною задачею та вимагає певних обчислювальних потужностей, котрі було б добре зберегти для інших операцій. Проте задачу можна спростити, використавши твердження 15.

$$S_{A-B}(v) = S_A(v) - S_B(-v) \quad (15)$$

, де $S_{A-B}(v)$ – опорна точка різниці Мінковського тіл A та B відносно вектора ***v***, $S_A(v)$ – опорна точка вектора ***v*** для тіла A , та $S_B(-v)$ – опорна точка вектора -***v*** для тіла B.

Взагалі, ми не хочемо будувати всю різницю Мінковсього, так як це дуже складно, а кінцевою метоє є просто перевірка, чи різниця включає в себе початок координат. Тому для перевірки твердження, ми не будуємо всю різницю, а використовуємо більш ефективний підхід, ми намагаємося побудувати піднабір точок різниці Мінковського такий, щоб він включав в себе початок координат, і якщо такий піднабір існує, то і сама різниця Мінковського включає в себе початок координат, а тому тіла перетинаються. Підмножину, котру будемо вибудовувати прийнято називати симплексом, в залежності від кількості вимірів дані симплекси можуть мати різні форми, всього нас цікавлять симплекси від 0 до 4:

- 0-симплекс – точка на площині.
- 1-симплекс – лінія на площині.
- 2-симплекс – трикутник (площина з трьох граней та трьох вершин).

– 3-симплекс – тетраедер.

Ще одна цікава властивість симплекса в тому, що при відніманні від нього кожної вершини ми змінюємо його розмірність (досить важлива властивість для наступного розгляду алгоритму GJK).

GJK повністю спирається на три основні поняття, розгляну вище: опорні точки, сума Мінковського та симплекс. GJK це ітеративний метод, проте котрий досить швидко сходиться (знову ж таки, в залежності від форми об’єкту), котрий може бути використаний лише для опуклих фігур, котрі підтримують опорні точки.

Загальна ціль методу GJK це знаходження такого симплексу, котрий би покривав початок координат і тим самим підтвердив, що фігури різниці Мінковського перетинаються. Алгоритм наступний:

1. Обираємо довільний напрямок для розрахунку за формулою 15 та розраховуємо значення опорної точки.
2. Обираємо протилежний напрямок до обраного на етапі 1 для того щоб гарантовано отримати іншу точку, та також знаходимо значення опорної точки в ній за формулою 15.
3. Після знаходження цих двох точок ми отримуємо відрізок, котрий лежить на вершинах різниці Мінковського. Необхідно обрати в якому напрямку буде проходити опорний вектор, для цього використовуємо припущення, що центр координат повинен лежати лише з однієї сторони від лінії. Тому напрямок наступного опорного вектора можна знайти як перпендикуляр від отриманої лінії до центру координат.
4. За формулою 15 знаходимо опорну точку для опорного вектора з кроку 4 та отримуємо симплекс.
5. Перевіряємо, чи початок координат знаходиться в симплексі. Якщо так, то алгоритм завершує роботу, ні – переходимо до пункту 6.

6. Знаходимо наступний напрямок як перпендикуляр від кожного ребра до початку координат, обираємо той, який напрямлений до центру координат.
7. Повертаємося до пункту 4. Повторюємо алгоритм різку кількість ітерацій в залежності від розмірності простору.

Варто також зауважити, що алгоритм GJK може бути використаний і для знаходження відстані між двома об’єктами, проте підхід там трохи інший та алгоритм підлягає мінорній модифікації.

1.3.2.3 Алгоритм EPA

GJK алгоритм надає нам інформацію про те чи зіткнулися об'єкти, а також симплекс, котрий покриває частину, що зіткнулась. Для того щоб отримати інформацію про глибину зіткнення можна використати інший алгоритм, котрий має абревіатуру **EPA**. [6] За допомогою **EPA** ми бажаємо досягти створення полігона (політипа), котрий буде покритий різницею Мінковського та розширити його до тієї міри, поки він не досягне країв різниці (суми) Мінковського, таким чином ми зможемо проаналізувати даний полігон та отримати глибину проникнення тіла один в одного. (Також варто зазначити, що на даному етапі часто вводять поняття мінімального вектора трансляції, котрий описує відстань, на яку треба перенести одне з тіл, щоб розділити їх).

EPA використовує ту ж саму теоретичну базу, що й GJK, за одним лише виключенням, що кількість точок в симплексі може бути довільною, виходячи з цього, даний термін вживатися не буде, замість нього буде вживаний термін полігон. Алгоритм наступний:

1. Ініціалізуємо початковий полігон за допомогою симплекса, отриманого з алгоритму GJK. Це буде початковий полігон, насправді, він може бути будь-яким, навіть випадковим, проте використання вже отриманої інформації спрощує обрахунки.
2. Визначаємо грань, котра є найближчою до початку координат.

3. Знаходимо нормально даної грані.
4. Використовуючи формулу 15 обраховуємо опорну точку.
5. Якщо відстань від нової точки до початку координат відрізняється від відстані до обраної попередньо грані на певну величину епсилон, то додаємо дану точку до полігону та переходимо до пункту 2, в іншому випадку алгоритм завершує свою роботу.

Зауважу, що точність використовується з тих причин, що ми все ще працюємо на ЕОМ, а алгоритм є ітеративним і може приближуватися безкінечно довго до повного збігу. Результатом глибин буде вважатися відстань від останньої отриманої точки в кроці 5 до початку координат, а нормаллю колізії вважається нормаль грані, котра використовувалася в кроці 5. [6]

1.3.2.4 Застосування CCD

Колізії, котрі оброблюються засобами, розглянутими вище на ЕОМ є дискретними, та ігнорують те що відбувається між кадрами під час симуляції. Досить велика кількість речей може бути невизначена даним видом колізії, особливо це стосується предметів, котрі рухаються на великій швидкості, а також ситуацій, коли крок симуляції є досить великим (такі проблеми часто виникають в ситуаціях, коли FPS знаходиться в районі значення 5-10). Останнє, звичайно є проблемою системи в цілому, та симуляції, проте для предметів з високою швидкістю можна знайти рішення. [7]

Знаходження колізій в проміжки часу, які відбулися між кадрами називається CCD – continues collision detection, неперервний пошук колізій. Засоби CCD дозволяють знайти час зіткнення тіл, котрі зіткнулися між кадрами та обробити їх на початку кадру, до проведення всіх інших симуляцій. Формально, час зіткнення – момент часу, в який відстань між тілами наближена до 0 або дорівнює йому. Взагалі, пошук такого часу є просто проблемою вирішення нелінійного рівняння та пошук його найменшого (позитивного) кореня. Дане

рівняння, звичайно залежить від форм тіл, котрі стикаються, враховуючи, що в парі можуть бути різні форми, то пар рівнянь може бути безліч.

В загальному всі ці системи дають нам рішення у нормалізованому проміжку від 0 до 1 за допомогою якого можна знайти конкретний час зіткнення та інші параметри, прикладаючи даний нормалізований коефіцієнт до дельти параметрів між кадрами. [7, 8]

Так як вирішення лінійних рівнянь є досить складним для довільних тіл, то їх використовують в основному для сферичних тіл, для інших же тіл використовуються алгоритми з підходом підкрокування, на якому ми вибираємо підкрок, котрий є меншим за час кроку симуляції, та ітеративно оновлюємо позиції необхідних тіл на цей підкрок, поки не досягнемо довжини основного кроку. Даний підхід є більш затратним, проте набагато більш універсальним та ефективним в плані різноманітності оброблених форм. [8]

1.3.3 Вирішення колізій

1.3.3.1 Система обмежень

Розглянуті в попередньому пункті симуляції розглядали лише або симуляції одного тіла, або повністю ігнорували взаємодію тіл між собою, формально кажучи, було розглянуто необмежені симуляції (unconstrained simulations), проте розгляд алгоритмів SAT, GJK та EPA приводять нас до заключення, що взаємодію тіл можна обрахувати. Звичайно, зіткнення тіл не є єдиним видом взаємодії, проте це основний її вид.

В загальному симуляції, де на поведінку тіл накладають ті чи інші обмеження називають обмеженими симуляціями (constrained simulations), а правила, котрі до них прикладають – обмеженнями, відштовхування тіл є просто ще одним обмеженням, вирішення якої буде розглянуто в даному розділі. Через використання обмежень в моделюванні на кожному його етапі необхідно

визначати корекційні сили та імпульси, що повинні бути застосовані в протидію до поведінки тіл, за для того щоб обмеження були задоволені та виконувалися. [9]

Згідно написаному вище, в реальному житті обмеження представляють собою функції поведінки тіла або ж його обмеження. Даний клас функції, від англійського слова constrains мають назву – C. Дані функції мають загальний вигляд: вони приймають у вигляді аргументів набір та їх параметрів та повертають в якості результату виконання певне скалярне значення (зазвичай нормалізоване в в проміжку від 0 до 1). Поки значення відповідних C функцій знаходяться в прийнятному діапазоні (для кожної C функції воно своє), то обмеження вважаються задоволеними. Якщо ж вірно протилежне, то система повинна прикладати до набору тіл, що не відповідають задовільним значенням, певні сили та імпульси, за для того щоб пара тіл та система в цілому була приведена до стану рівноваги та задовольняла визначеним через C функції обмеженням. [10, 11]

Приклади розрахунку тих чи інших обмежень є дуже об’ємними, тому в даному розділі ми опустимо конкретні приклади, розглянувши види обмежень, так як вони взаємодіють між собою. Очевидно, що існує безліч видів обмежень, а також очевидно, що вони можуть одночасно бути накладені на симуляцію. Загальна схема використання та представлення обмежень наступна:

$$\bar{q} = \langle q_0, q_1 \dots q_n \rangle = |q_i \coloneqq (\bar{p}_i, \bar{r}_i)| = \langle (\bar{p}_0, \bar{r}_0), (\bar{p}_1, \bar{r}_1) \dots (\bar{p}_n, \bar{r}_n) \rangle \qquad (16)$$

, де $\bar{\boldsymbol{q}}$ – вектор стану всіх тіл на сцені, $\boldsymbol{q_i}$ – стан і-того тіла на сцені, викладка описує, що стан $\boldsymbol{q_i}$є кортеж від двох елементів: $\bar{\boldsymbol{p}}_{\boldsymbol{i}}$ – вектор позиції і-того тіла на сцена, а також $\bar{\boldsymbol{r}}_{\boldsymbol{i}}$ – вектора повороту і-того тіла, тому стан може бути представлений як вектор розмірності 3 та 5 в залежності від вимірності простору (дво та тривимірний простори відповідно).

$$\bar{F} = \langle (\bar{f}_0, \bar{\tau}_0), (\bar{f}_1, \bar{\tau}_1) \dots (\bar{f}_n, \bar{\tau}_n) \rangle \qquad (17)$$

, де $\bar{\boldsymbol{F}}$ – вектор кортежу сил, прикладених до кожного і-того тіла, $\boldsymbol{f_i}$ – вектор сили**,** прикладеної до і-того тіла, $\bar{\boldsymbol{\tau}}_{\boldsymbol{i}}$ – вектор моменту сили, прикладеного до і-того

тіла. Викладка 18 описує, що глобальна сила з твердження 17 прикладена до кожного тіла є результуючою зовнішніх сил, прикладених до тіла а сил обмежень, прикладених до відповідних тіл:

$$\bar{F} = \bar{F_e} + \overline{F_C} \tag{18}$$

, де $\boldsymbol{\bar{F}}$ – вектор кортежу сил, прикладених до кожного і-того тіла, $\boldsymbol{\overline{F_e}}$ – вектор відповідних зовнішніх сил, прикладених до кожного і-того тіла, $\boldsymbol{\overline{F_C}}$ – вектор сил, котрі виникають з обмежень та прикладені до кожного і-того тіла. Використовуючи записи 16, 17 та 18 можна записати Другий закон Ньютона у векторному вигляді з урахуванням лінійної та кутової складової:

$$\ddot{q} = M^{-1}\bar{F} = M^{-1}(\bar{F_e} + \overline{F_C}) = \langle \ddot{q_0}, \ddot{q_1} \dots \ddot{q_n} \rangle \tag{19}$$

$$\ddot{q} = \langle \ddot{q_0}, \ddot{q_1} \dots \ddot{q_n} \rangle = |q_i \coloneqq (\bar{a}_i, \bar{\alpha}_i)| = \langle (\bar{a}_0, \bar{\alpha}_0), (\bar{a}_1, \bar{\alpha}_1) \dots (\bar{a}_n, \bar{\alpha}_n) \rangle \tag{20}$$

, де $\boldsymbol{\ddot{q}}$ – вектор других похідних елементів вектору стану $\boldsymbol{\bar{q}}$, елементами даного вектора других похідних станів є $\boldsymbol{\bar{a}_i}$ – лінійне прискорення тіла, $\boldsymbol{\bar{\alpha}_i}$ – кутове прискорення тіла, $\boldsymbol{\overline{F_e}}$ – вектор відповідних зовнішніх сил, прикладених до кожного і-того тіла, $\boldsymbol{\overline{F_C}}$ – вектор сил, котрі виникають з обмежень та прикладені до кожного і-того тіла. $\boldsymbol{M}$ – діагональна матриця, котра на своїх діагоналях містить послідовно елементи: $\boldsymbol{m_i, m_i, I_i}$, та відповідно $\boldsymbol{M}^{-1}$ – обернена до неї. [11]

Враховуючи описані вище формули тепер опишемо групу обмежень, котрі накладаються для кожного тіла, опис представлено у викладці 21 нижче:

$$\overline{C(q)} = \langle C_0(q), C_1(q) \dots C_k(q) \rangle \tag{21}$$

, де $\boldsymbol{\overline{C(q)}}$ – вектор всіх функцій обмежень, котрий прикладений до тіла та приймає у якості параметру стан тіла $\boldsymbol{q}$, $\boldsymbol{C_i(q)}$ – одна з функцій обмежень, котра прикладається до тіла та приймає стан тіла в якості аргумента. Результатом виконання групи даних є вектор обмежень, котрий повинен бути максимально приближений до значення 0 (мається на увазі, що магнітуда даного вектора має бути близька до 0). Проте рівності нуля самого значення обмеження не є

достатньо, для цього необхідно щоб перша та друга похідна обмеження також дорівнювала 0. Визначимо для цього твердження 22. [12]

$$\dot{C} = \frac{\partial C}{\partial q}\dot{q} = \left| J := \frac{\partial C}{\partial q} \right| = J\dot{q} \quad (22)$$

, де $\dot{\boldsymbol{C}}$ – перша похідна обмеження, $\frac{\partial \boldsymbol{C}}{\partial \mathbf{q}}$ – похідна обмеження за станом тіла, $\dot{\boldsymbol{q}}$ – перша похідна стану тіла. Виходячи з цього та прикладаючи до викладки 21 формулу 22 приходимо до розуміння, що ми отримуємо матрицю Якобі/Якобіан – $\boldsymbol{J}$, котра в свою чергу є узагальненим виглядом градієнту, котрий описує зміну поведінки функції по всіх її аргументах. Використовуючи формулу 22 потрібно вивести формулу для другої похідної обмеження:

$$\ddot{C} = |\dot{J} := \frac{\partial \dot{C}}{\partial q}| = \dot{J}\dot{q} + J\ddot{q} \quad (23)$$

$$\ddot{C} = \dot{J}\dot{q} + JM^{-1}(\overline{F_e} + \overline{F_C}) \quad (24)$$

, де $\ddot{\boldsymbol{C}}$ **–** друга похідна обмежень, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ – перша та друга похідні стану тіла, J – Якобіан, $\dot{\boldsymbol{J}}$ – визначений у додатку до формули 23. Всі інші позначення розглянуті та описані в формулі 19. Наведена вище формула 24 є фінальною для прорахунку обмежень, далі підходи до вирішення даної задачі різняться, проте всі вони полягають в тому щоб поставити $\ddot{C} = 0$ та вирішити рівняння з n невідомими змінними, за даними невідомими змінними ми $\overline{F_C}$, що є фінальною метою вирішення даної системи рівнянь. [12]

1.3.3.2 Вирішення обмеження непроникнення

Вище було надано теоретичне ознайомлення з поняттям обмеження, зараз же потрібно розглянути один конкретний вид обмеження. До цього також розглядалось, що ми повинні тримати значення C якомога ближче до 0, проте обмеження непроникнення відноситься до другого виду обмежень, котра називається **нееквівалентні обмеження**, та на відміну від попереднього типу, в даному виді обмежень значення C дозволяють більш широкий набір значень окрім

0, тобто значення C можуть знаходитися в певному інтервалі довіри. Обмеження непроникнення буде намагатися тримати зіткнені тіла розділеними.

Завдяки алгоритму GJK отримано точку зіткнення тіл, а також нормаль поверхні в точці зіткнення тіл. Слід також нагадати, що алгоритм GJK може використовуватися і для знаходження відстані. Про це слід пам’ятати, так як в багатьох система тіла можуть вважатися зіткнувшимися, якщо відстань між ними досить мала, в такому випадку точкою зіткнення прийнято брати найближчі точки між тілами. Розглянемо приклад вирішення колізії на прикладі двох тіл у двовимірному просторі (взагалі всі приклади досить легко розглядати саме у двовимірному просторі, так як більшість розглянутих алгоритмів досить легко розширюються на третій вимір). Для того щоб розглянути всі крайні випадки та дослідити формулу в цілому, розглянемо випадок зіткнення двох тіл з рисунку 13.

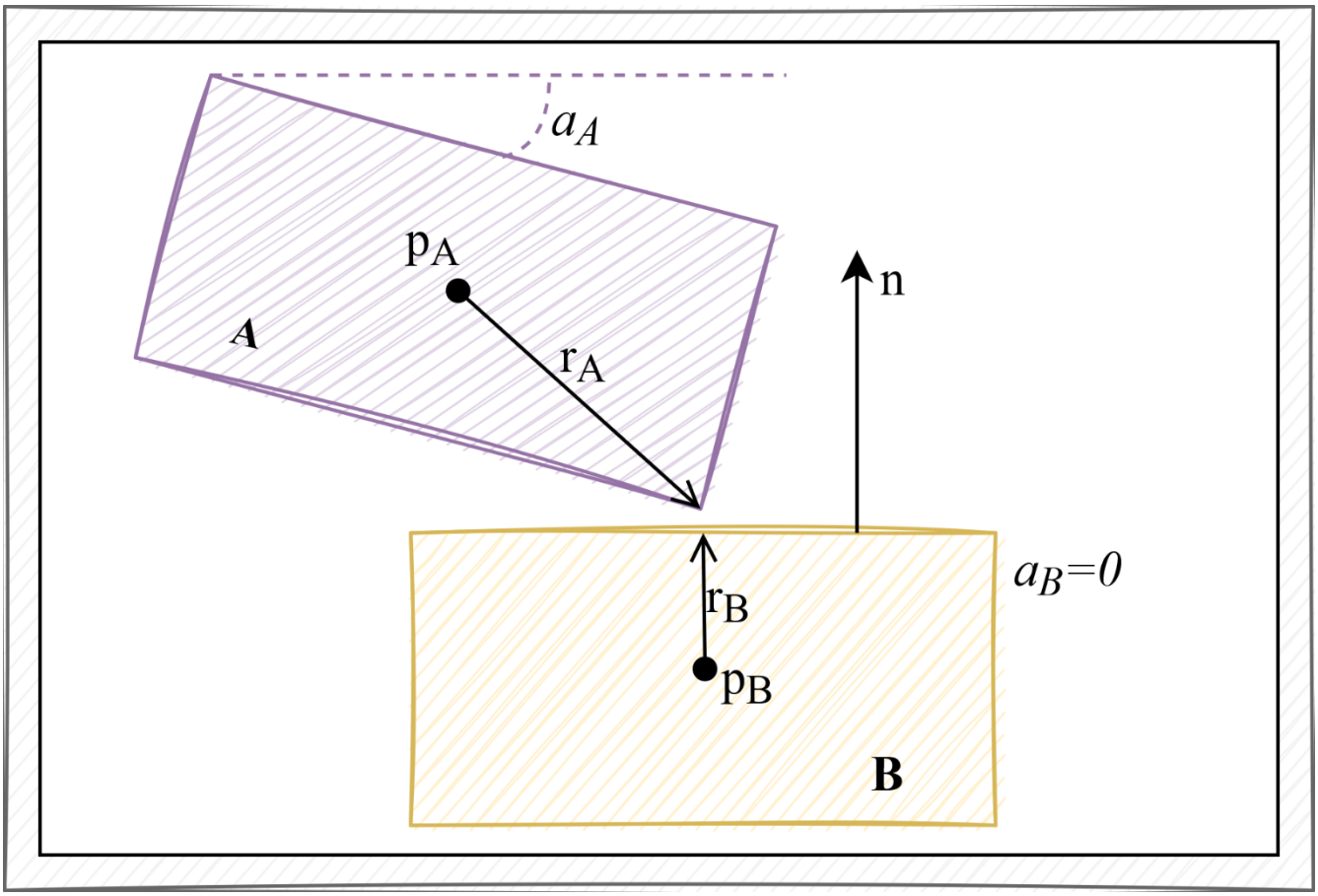

Рисунок 13. Зображення двох тіл при зіткненні та їх параметри

На рисунку 13 наведений є універсальним та загальноприйнятим способом розгляду такого виду обмеження тому є досить розповсюдженим. [1 – 4]. На рисунку вище показано схематичне зображення зіткнення двох тіл, в тривимірному просторі вирішення рівняння, представленого далі буде рівно тим самим, лише компоненти повороту та позиції матимуть на одну розмірність більше. Виходячи з рисунку 13 можемо надати наступну формулу для вирішення обмеження непроникнення:

$$C(p_A, a_A, p_B, a_B) = (p_B + R(a_B)r_B - p_A - R(a_A)r_A) \cdot n \qquad (25)$$

, де $\boldsymbol{C(p_A, a_A, p_B, a_B)}$ – функція від 4-ьох аргументів, котра видає значення величини обмеження, $\boldsymbol{p_A}$ – позиція тіла А (центр мас тіла А), $\boldsymbol{p}_\text{B}$ – позиція тіла В (центр мас тіла В), $\boldsymbol{a}_\text{A}$– поворот тіла А, $\boldsymbol{a_B}$– поворот тіла В, n – нормаль площини зіткнення, $\boldsymbol{R(a)}$ – матриця повороту на кут(и) $\boldsymbol{a}$.

Проаналізувавши отримане далі рівняння, виявиться, що для успішного вирішення даного рівняння потрібно обмежити значення коренів для кожного обмеження у системі (якщо їх декілька). У обмеженнях попередніх типів (обмеження еквівалентності) корені рівняння можуть приймати як позитивні, так і негативні значення, що показує, що обмеження може бути направлене як в позитивне так і в негативне значення, і тому відхилення від еталону може бути у будь-яку сторону. Проте при використанні обмеженості нееквівалентності, а конкретно при обмеженні непроникненні ми бачимо, що значення коренів рівняння може бути лише позитивним (або рівне нулю) так як даний корінь представляє обмеження, котре використовується для виштовхування тіл одне з одного, тобто лише для позитивного вирішення обмеженості. [13]

Вирішення даних систем рівнянь є окремою темою для дослідження, проте в більшості випадків дана проблема вже є розглянутою. [13] Подальший розгляд симуляції фізичних процесів в механіці повинен мати відношення до імпульсної симуляції, розглянуті зараз симуляції проводилися на основі сил. Проте на даний момент достатньо розглянутих симуляцій на основі сили, щоб перейти до

наступної частини роботи – використання хмарних систем та розподілених обчислень для моделювання поведінки твердих тіл.

1.4 Висновки до розділу 1

В процесі роботи над даним розділом було досліджено теоретичний базис, необхідний для проведення фізичних симуляцій з твердими тілами в просторі з дією як постійних сил, так і зовнішніх окремих сил, прикладених до кожного тіла. Були розглянуті ключові терміни фізичної, математичної та інженерної (унікальної саме для проведення симуляцій на ЕОМ) складових моделювання:

– Що до теоретичної фізичної складової було розглянуто наступні твердження:
  - Розглянуто поступовий перелік та розгляд основних теоретичних законів та форму, необхідних для симуляції поведінки частинок та в подальшому – твердих тіл. Наведено простий код для перевірки роботи даних формул та тверджень.
  - Надано категоризацію виду тіл, котрі взагалі можуть бути представлені в процесах моделювання/симуляції поведінки на сцені. Визначено конкретний спектр тіл, з котрими будемо мати справу в подальшому.
  - Було надано визначення поняттю твердого тіла, котре буде використовуватися в подальшому на протязі всієї роботи. Перелічено його властивості та надано фокус на основних причинах вибору саме даного типу об’єктів.
  - Порівняння різниця між проведенням моделювання у двовимірному та тривимірному просторах. Показано, що всі формули та алгоритми, що прикладаються до двовимірного простору можуть бути розширені для використання у тривимірному просторі з додаванням одного додаткового виміру до основних параметрів: позиція, лінійна

швидкість, лінійне прискорення, вектори сили, момент сили, кутова швидкість, кутове прискорення, інерція.

– З математичної складової завдання розглянуто:
  - Поняття чисельних методів та основні принципи, котрі будуть використані при роботі на ЕОМ ті різниця з аналітичним розв'язанням відповідних задач.
  - Розглянуто доведення та виведення більшості математичних викладок, котрі використовуються при моделюванні та розв'язанні рівнянь, що використовуються в механізмах «обмежень» в рушіях.
  - Розглянуто використання поняття градієнту та використання Якобіанів при розрахунках функцій механізму «обмежень».
– З інженерної складової розглянуто наступні елементи:
  - Представлено базову, абстрактну модель, котра включає в себе основні етапи моделювання, котрих повинен притримуватися кожен рушій. Відхилення від даних етапі у більшості випадків веде до унеможливлення використання розробленої теоретичної бази.
  - Для кожного етапу визначення колізії було розглянуто базові концепції та алгоритми, використання котрих достатньо для побудови примітивного рушія: типи фаз, структури даних, алгоритми та підходи. Було надано посилання на матеріали, котрі повинні ознайомити з розвинутими техніками, не розібраними в даній роботі.
  - Розглянуто поняття кроку симуляції, розглянуті мірі визначення ефективності та швидкодії роботи симуляції: полігони, кадри за секунди, час відгуку.
  - Було розглянуто поняття широкої та вузької фази пошуку колізії. Та елементи, що використовуються в них.

- Було розглянуто поняття обмежуючих об’ємів та надано вичерпний список основних їх типів: AABB, OOBB, Bounding Sphere. Надано алгоритми та підходи до перевірки їх зіткнень.
- Було розглянуто поняття ієрархії обмежуючих об’ємів, а також поняття розбиття простору, надано основні критерії, котрих варто притримуватися при розбитті простору.
- Було розглянуто поняття прискорюючих структур (в контексті проведення симуляцій) та надано їх основний список.
- Було розглянуто підходи для знаходження колізій у вузькій фазі: розглянуто SAT, алгоритми GJK та EPA, підходи до вирішення проблеми CCD. Було наведено приклади використання та детальний опис алгоритмів для дво та тривимірного просторів.
- Було розглянуто механізм «обмежень», його застосування, види обмежень, а також розглянуто приклад його застосування для вирішення проблеми проходження тіл один крізь одного. Тобто – розглянуто способи вирішення колізії як такової.

Підсумовуючи вище наведений список заключено, що в результаті виконання роботи було проведено ознайомлення з усім базовим об’ємом знань, необхідних для успішного проведення досліджень у вибраному напрямку.

2 ІНСТРУМЕНТИ ФІЗИНЧХ СИМУЛЯЦІЙ ТВЕРДИХ ТІЛ ДЛЯ КОМПЮТЕРНОЇ ГРАФІКИ

2.1 Перелік основних бібліотек та SDK

На даний момент достеменно відомо велику кількість фізичних рушіїв, котрі ефективно виконують покладені на них функції. Вважається, що на ринку закріпилось декілька основних великих розробок, більшість з них є пропрієтарними та розповсюджуються за досить високу плату. В загальному, список найпопулярніших рішень наступний:

2.1.1 Nvidia PhysX SDK

Nvidia PhysX SDK – найпопулярніше рішення, коли мова заходить до фізичних рушії, станом на 2022-ий рік нараховується п'ята ітерація (версія) даного фізичного **SDK**. На даний момент є складовою рішення Nvidia Omniverse, анонс та концепція якого є основним рушієм до виконання даного дослідження. Безкоштовно розповсюджується, проте платний при використанні в будь-якому проєкті, розрахованому на використання в комерційних цілях. [14] В останніх версіях підтримує сучасні технології та досить складні елементи моделювання:

- Моделювання м’яких тіл.
- Моделювання поведінки тканин (в основному – одягу).
- Симуляції великих об’ємів води.
- Створення довільної геометрії.
- Молекулярна динаміка.

Рішення PhysX поставляється з великою кількістю засобів для відлагодження сцени, профайлінгу поведінки симуляції, а також з детальною документацією що до SDK. Також користувачам SDK надається окрема технічна підтримка у випадку виникнення тих чи інших проблем. Великою перевагою

даного SDK є можливість виконання великої кількості операцій на GPU, що не є великою рідкістю, проте враховуючи спеціалізацію Nvidia на створенні апаратного забезпечення – GPU, вони досягли значних успіхів в утилізації даної периферії при роботі з фізичними симуляціями, особливо це стосується використанням власних GPU Nvidia та CUDA рішень. Використовується як стандартне рішення для рушія Unreal Engine. Набір особливостей можна побачити на рисунку 14. [14]

Latest PhysX Releases

	PHYSX 5	PHYSX 4
Rigid Body Dynamics(TGS or PGS solver)	✓	✓
Scene Query	✓	✓
Joints	✓	✓
Reduced Coordinate Articulations	✓	✓
Vehicle Dynamics	✓	✓
Character Controllers	✓	✓
Soft Body Dynamics (Finite Element Method)	✓	
Cloth (Finite Element Method)	✓	
PBF(liquid/cloth/inflatable/shape matching)	✓	
FLIP(large scale water simulation)	✓	
Custom Geometries	✓	
Material Point Method (MPM)	🔨	
Molecular Dynamics	🔨	

Рисунок 14. Порівняння особливостей PhysX5 та PhysX4

2.1.2 Havok Physics

Havok Physics – даний фізичний SDK/рушій представляється як прямий конкурент PhysX. Його основною перевагою є те що він постачається разом з

Unity, котрий є одним з основних рушіями в індустрії інтерактивних розваг та кінематографу.

Разом з даним пакетом поставляються не лише засоби для симуляції фізики – Havoc Physics, а й окремі засоби для розрахунку симуляції поведінки тканин (Havoc Cloth), а також засоби для розробки AI (Havoc AI). [15] Також поставляються й засоби відлагодження та профайлінгу, як і у випадку прямого конкурента PhysX. Також проблемою є, на відміну від PhysX обмежене використання GPU, котру утилізується лише для Havoc FX. Також є пропрієтарним рішенням.

2.1.3 Bullet Library

Bullet Library – рушій з відкритим кодом, котрий в основному розроблюється однією людиною. [16] В основному орієнтований на виконання на CPU, проте має зачатки OpenCL пайплайну в експериментальний гілці. Великою перевагою на відміну від названих вище конкурентів є:

- Портативність. В ситуації з PhysX це не грає великої ролі, так як в більшості випадків інтеграція PhysX або Havok в окремий рушій це важка задача. Проте взагалі даний рушій є портативним та досить легко запускається на будь-якому апаратному забезпеченні, котре підтримує C++03x компілятор. Багато розробників вважають, що саме Bullet мав би бути вбудованих рушієм для Unity, так як основна платформа даного рушія – мобільні платформи, а рушії Havok та PhysX нативно не підтримуються на даних платформах, проте все одно як де-факто стандарт в Unity обраний Havok.
- Відкритість коду. Bullet є відкритим рушієм, код котрого можна переглянути в будь-який момент часу, що несе велику перевагу для його користувачів. Окрім того, даний рушій гарно структурований та описаний. Також – розповсюдження через систему vcpkg.

– Ліцензія. Дана бібліотека розповсюджується за ліцензією zlib, що дозволяє будь-яке комерційне застосування та модифікацію за умовою згадування автора навіть за умови модифікації початкового коду.
– Поставляється з рішенням для рендеру, котра є досить простим та примітивним, проте добре допомагає при створенні прототипів різних систем.

2.1.4 Box2d Library

Box2D – відкрита бібліотека для проведення симуляцій в двовимірному просторі. Проста та перевірена часом. Дана бібліотека є досить простою та ефективно виконує заявлені функції. Має обширну документацію та вивірений часом програмний код, котрий має мінімальну кількість проблем. Найпопулярніший рушій для використанням у двовимірному просторі. [17] Підтримує наступні особливості:

– Підтримується системою модульних тестів.
– Відкритість коду, як і у випадку Bullets. Також – розповсюдження через систему vcpkg.
– Ліценція. Розповсюджується за MIT ліцензією, котра дозволяє будь-яке використання продукту за згадуванням автора та збереженням оригінального тексту ліцензії.
– Висока ефективність утилізації пам’яті. Бібліотека написана на мові C++ та повністю використовує дану особливість для ефективної утилізації пам’яті.

2.1.5 Порівняння технологій

Так як перші два варіанти закриті до використання в дослідженнях, і хоча автор роботи має доступ до даних рішень, проте їхнє використання не є дозволеним у дослідницьких та приватних цілях, прийнято рішення щодо

використання останніх двох рішень – Bullets та Box2d. В силу того що метою є дослідження саме колізій в тривимірному просторі, то вибір допоміжного рушія, за допомогою якого буде проводитися частина досліджень впав на Bullets.

Також слід зауважити, що жоден з вищезазначених рушіїв сам по собі не є ідеально точним, щоб бути використаним для фізичних симуляцій, точність котрих повинна враховувати всі можливі фактори взаємодії на тіла (як наприклад – сила Коріоліса і т.п.). Проте, дані рушії надають можливість додавати будь-які додаткові способи впливу, тому їх використання може бути доречним при умові додаткового втручання інженера ПЗ. [18] В таблиці 1 наведене порівняння згаданих вище рушіїв для роботи в тривимірному простору (PhysX, Havok, Bullet).

Таблиця 2.1 – Порівняння PhysX, Havok, Bullet

Назва	**PhysX**	**Havok**	**Bullet**
Ліцензія	Пропрієтарна, власник – Nvidia, BDS ліцензія для PC	Пропрієтарна, власник – Intel	Відкрита ліцензія, zlib
Підтримка CPU	PC, MacOS, Linux, PlayStation, Xbox, iOS	PC, MacOS, Linux, PlayStation, Xbox, iOS, Nintendo Switch	PC, MacOS, Linux, PlayStation, Xbox, iOS, Nintendo Switch
Підтримка GPU	PC, MacOS	PC	OpenCL (limited usage)
Потоки	Багатопоточне використання GPU	Багатопоточне використання GPU та CPU	Багатопоточне використання GPU
Сторонні продукти	Maya, Houdini, 3DSMax, Blender	Maya, Houdini, 3DSMax, Blender	Maya, 3DSMax

2.2 Хмарні технології в фізичних рушіях

В загальному, коли розглядається використання фізичних рушіїв в хмарних технологіях то розглядають два основних напрямки використання, один з них – основний, інший – експериментальний та тільки почав активно розвиватися.

2.2.1 Часткові симуляції фізики на хмарах

Першим та найбільш інтуїтивним, основним та часто вживаним є використання віддаленого сервера для розрахунку симуляції клієнта. В даному випадку на сервері відбуваються лише критичні розрахунки, а частина неважливих розрахунків все ще залишається на стороні клієнта. Частіше за все даний підхід використовується саме у відеоіграх, так як даний спосіб дозволяє захиститися від шахрайства. Дана архітектура може працювати в трьох напрямках:

– Сервер є єдиною обчислюючою одиницею симуляції, він повністю займається обробкою фізики: знаходженням колізій, визначення положення тіл, визначення потраплянь снарядів і так далі. Всі параметри клієнт отримує виключно зі сторони сервера. Проте, слід зауважити, що під повною симуляцією мається на увазі симуляції лише критично важливих елементів симуляції. На прикладі багатокористувацької гри: клієнт отримує від сервера своє положення, положення гравців навколо, також клієнт отримує результати потраплянь, припустимо – куль в інших гравців. Проте, в той же час клієнт може проводити симуляції менш важливих деталей, так як: партикли, котрі є частиною візуальної складової, а не гри, або ж симулювати поведінку трави, води і так далі. Тобто, під виразом «єдина обчислювальна одиниця симуляції» мається на увазі, що сервер є єдиним, хто обчислює критичну інформацію.
– Сервер та клієнт обидва займаються обрахунком симуляції. В такому випадку сервер займається все тією ж роботою, що і в попередньому пункті,

проте тепер клієнт також займається обчисленням симуляції. Частіше за все використовується у випадку коли клієнт повинен мати змогу бути стійким до зникнення зв'язку з сервером. Окрім того, в такому випадку дані всіх клієнтів консолідуються, перевіряються та порівнюються з сервером, що значно зменшує вірогідність помилки симуляції.

– Сервер відсутній та клієнти обмінюються даними один з одним напряму. В даному випадку маємо скоріше P2P з'єднання.

2.2.2 Повна симуляції фізики на хмарах

Другий спосіб – повна симуляція всієї сцени на віддаленому сервері, разом з усіма ефектами. Даний спосіб використовується в основному закритими рішеннями для створення раніше визначених/заскриптованих сцен. Експериментальним же є використання даного підходу для повної симуляції інтерактивних сцен, даний підхід зовсім не використовується, першим його представила компанія Nvidia разом з продуктом. На рисунку 15 можна побачити частину засобів, котрі інтегрують в себе екосистему Omniverse. [19]

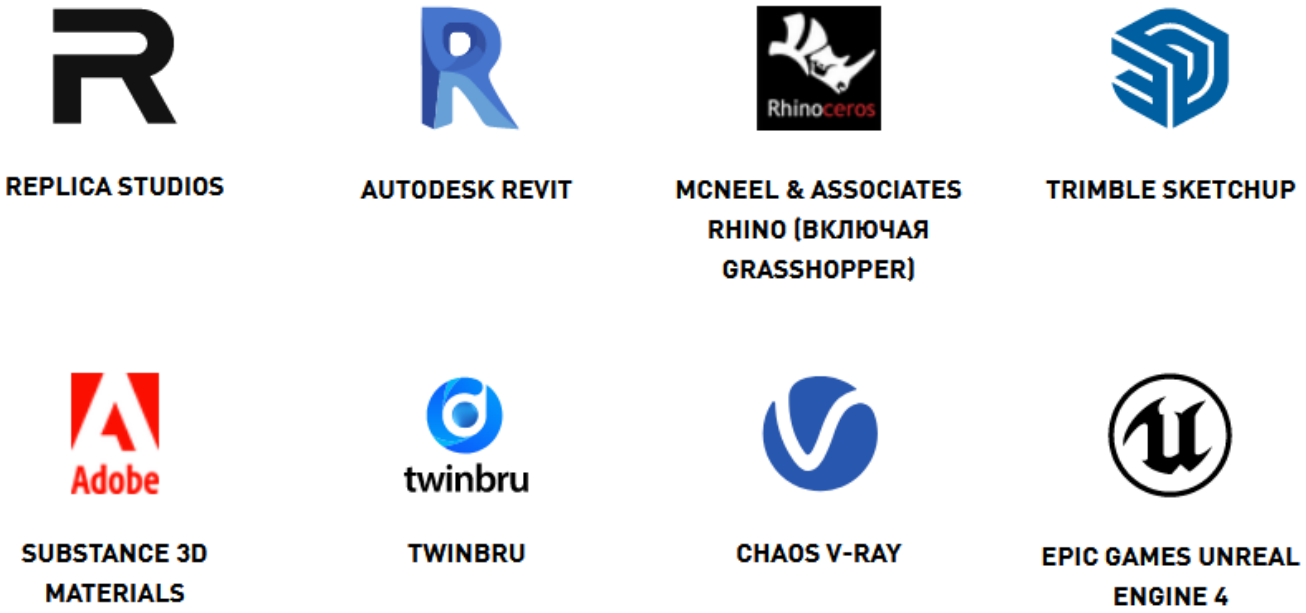

Рисунок 15. Перелік застосунків, котрі сумісні з пакетом Omniverse

Omniverse надає сторонній редактор, котрий може бути під’єднаний до великої кількості популярних застосунків, котрі використовують рішення PhysX, або він може бути інтегрований в будь-яке інше рішення. Список таких рішень можна побачити на рисунку 15, котрий було згадано раніше. [19] Окрім того Omniverse надає набір інструментів, котрі дозволяють створювати власні застосунки на основі Omniverse, список цих інструментів наведено на рисунку 16.

Рисунок 16. Набір інструментів пакету Omniverse

В даній роботі спробуємо оцінити складність створення системи, котра дозволяє проводити симуляції в тривимірному просторі, а також надати приблизний вигляд архітектури, котра б дозволила отримати результат, схожий до нещодавно випущеного пакету Omniverse. Окрім того, порівняємо ефективність використання клієнт-серверного моделювання фізики з повним та частковим використанням симуляції на стороні серверу. Також, розглянемо використання кластерних обчислень для виконання фізичних симуляцій.

Взагалі у випадку таких симуляцій великою проблемою є передача даних та час реакції/відгуку між клієнтом та сервером, котрий прийнято називати просто затримкою, або ж пінгом, від англійського слова **ping**. Також великою проблемою

є формат синхронізації даних та формат сцени, він також грає велику роль. Досить ефективним є формат **USD**, котрий був представлений компанією Pixar ще для власних пропрієтарних застосунків, але став настільки популярним, що є індустріальним стандартом в описі сцен. [20] В заключення можна сказати, що даний вид застосунків страждає рівно від тих самих загроз, що і будь-який інший клієнт-серверний застосунок.

2.3 Висновки до розділу 2

В процесі роботи над даним розділом було проведено практичну та теоретичну роботу щодо дослідження та прототипування архітектури системи, котра повинна використовуватися для подальшої розробки системи. Детальний опис отриманих результатів наступний:

– Проведено порівняння основних технологій, котрі використовуються для симуляції фізики, надано список переваг та недоліків, а також список корисних особливостей даних технологій. Їх перелік:
    - PhysX – універасльне, проте платне для досліджень та комерційного використання рішення.
    - Havok – універсальне, проте повністю закрите рішення.
    - Bullet – рішення з відкритим кодом та базовою підтримкою більшості алгоритмів.
    - Box2D – рішення з відкритим кодом, але виключно для двовимірних просторів.

  Наведено порівняння даних технологій, а також наведено критерії у вигляді таблиці порівнянь, за якими було обрано ту чи іншу технологію для подальшої роботи. В фінальному результаті для подальшої роботи було використано технологію Bullet, так як вон має відкриту ліцензію, відносно легко піддається модифікаціями, має примітивну підтримку стандартного

OpenCL (за допомогою чого потрібно затратити менше технічних зусиль на під’єднання кластеру, хоча це все ще є складною задачею), а також автор роботи попередньо знайомий з даною роботою в професійній площинію.

– Окремо розглянуто технологію Nvidia Omniverse, котра є важливою віхою та проривним досягненням в технологіях симуляції фізики, особливо в напрямку використання хмарних та розподілених обчислень. Анонс даної технології був основним поштовхом для проведення даного дослідження.

3 РОЗРОБКА СИСТЕМИ КОМПЮТЕРНОЇ ГРАФІКИ З ВІДДАЛЕНИМ ФІЗИЧНИМ МОДЕЛЮВАННЯМ

3.1 Постановка задачі

Після розглянутого вище матеріалу перейдемо до прототипування архітектури редактора та рушія з використанням обчислень на віддаленому сервері. Розглянемо дану задачу з точки зору розробки та проєктування будь-якого програмного забезпечення. Перш за все необхідно чітко визначити задачі, котрі повинні бути поставлені перед нами, вимоги, котрі повинні виконуватися системою:

- Система повинна мати клієнт-серверну архітектуру.
- Система повинна мати можливість відображати тривимірному простір.
- Система повинна коректно симулювати переміщення та зіткнення тіл у тривимірному просторі.
- Система повинна моделювати розрахунки на серверній стороні.
- Система повинне мати можливість масштабуватися на різну кількість користувачів та машин в системі.
- Система повинна реалізувати редагування сцени паралельно декільком користувачам.
- Система повинна підтримувати універсальні формати опису сцени таке як USD та Waveform.

Сформульовані вище вимоги отримані в результаті дослідження вже існуючих засобів, а також досвіду їх використання. Можливість досягнення поставлених цілей також є об’єктивно високою в силу обставин, проблематики та рівню розвитку сучасних технологій.

Як буде наведено далі, найбільша проблема, що стримувала інженерів від вирішення поставлених вище задач – недосконаліть технологій, практична

неможливість отримання відповідного апаратного забезпечення, відсутність необхідних досконалих технологій та світової інфраструктури в цілому.

Згідно наведеному вище списку задач сформульовано дерево проблем та дерево рішень, котрі зображено відповідно на рисунку 17 та 18.

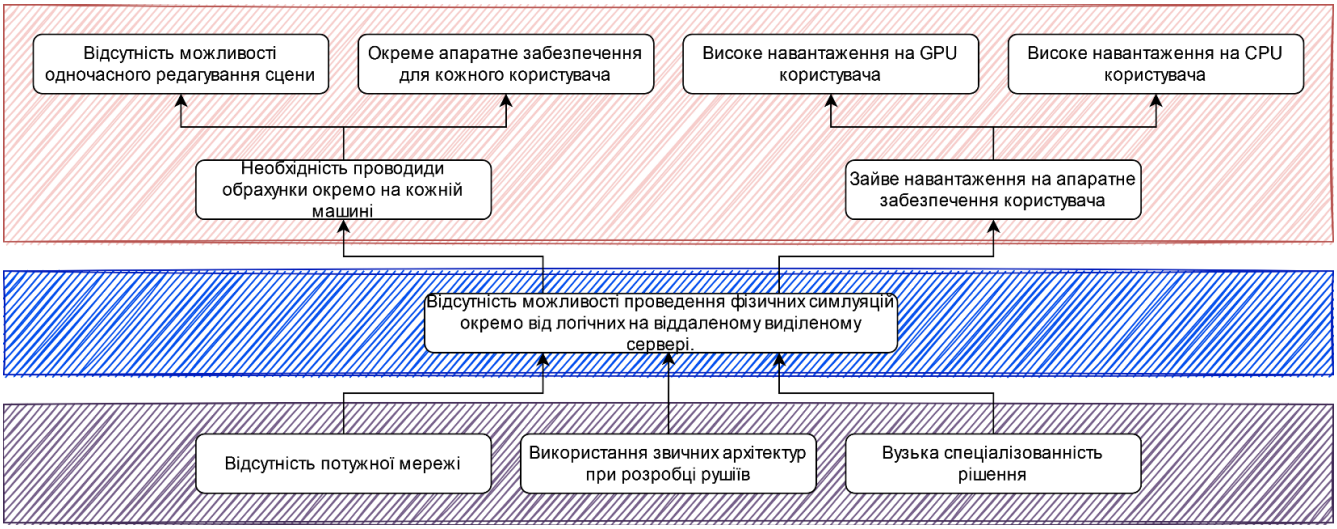

Рисунок 17. Дерево проблем

На дереві проблем, зображеному на рисунку 17 наведено три основні блоки, котрі описують наступне:

- **Синій блок** представляє собою основну проблему, котру ми маємо вирішити при розробці системи.
- **Червоний блок** це наслідки проблеми, котрі будуть вирішені при вирішенні основної. Це не всі наслідки, проте – найважливіші з них.
- **Фіолетовий блок** це причини виникнення основної проблеми. В даному блоці наведені основні причини, це не є повний перелік, проте дані причини є найважливішими.

Після створення дерева проблем можемо перейти до формулювання цілей, котрі повинні бути досягнуті та засобів за допомогою яких дані цілі повинні бути досягнуті. Також наведено список можливостей, котрі ми отримаємо після досягнення поставленої мети.

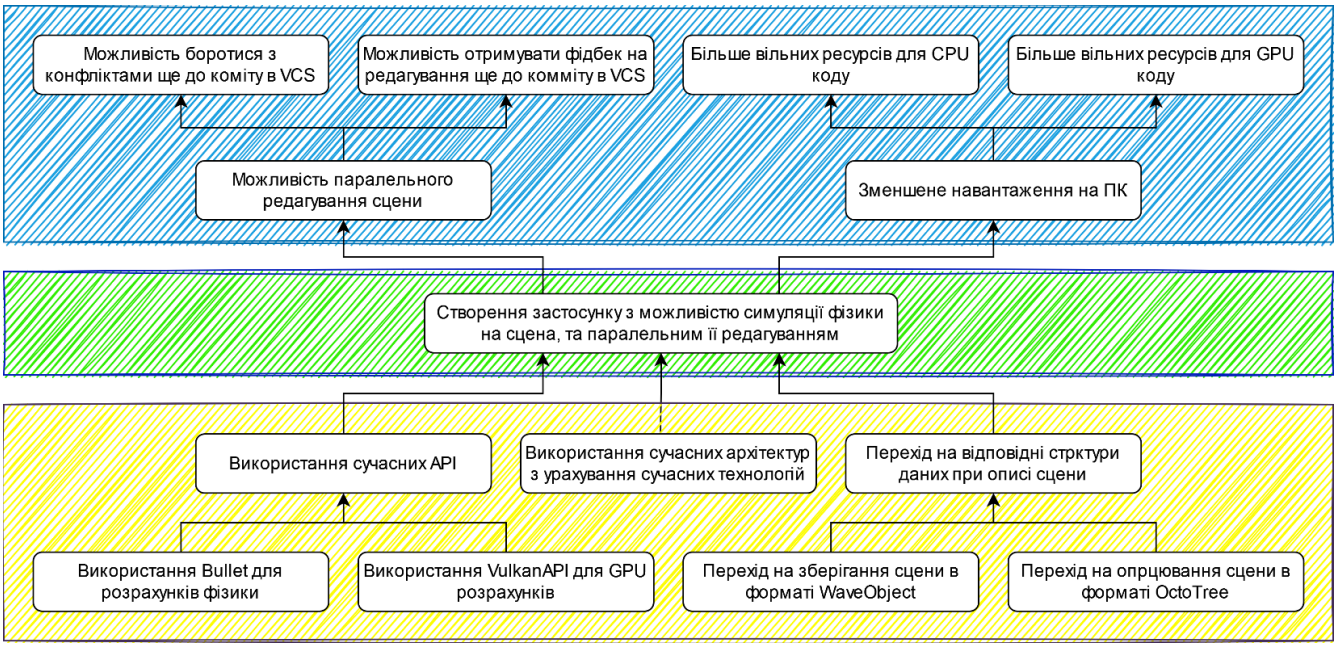

Рисунок 18. Дерево проблем

На дереві рішень, наведеному на рисунку 18 та представленому вище маємо три основні блоки:

- **Зелений блок** – основна мета.
- **Голубий блок** – отримані від досягнення мети переваги.
- **Жовтий блок** – проміжні цілі та мінімальні засоби, котрі ми повинні використати й досягнути для досягнення основної мети.

Дорожня карта розробки проєкту, котра може бути використана як перелік основних ключових віх в розробці проєкту наведена нижче. В ході виконання даного дослідження частина була пройдена, це стосується пунктів 1 – 6 та 9:

1. Зробити огляд подібних рішень у хмарних системах, оцінити вимоги потенційних користувачів такої системи.
2. Розглянути класи хмарних систем, які можуть бути застосовані для вирішення задачі.
3. Розглянути алгоритми, які можна застосувати у хмарних системах для розрахунку колізій.
4. Розглянути необхідні архітектури для реалізації подібних систем.

5. Розглянути можливість застосування мікросервісів для побудови архітектури системи розрахунку колізій.
6. Дослідити параметри і протоколи для передачі даних з- та в- хмарну систему.
7. Дослідити на прикладі хмарної системи параметри, які будуть мати критичні вимоги для побудови системи розрахунку колізій у певних галузях.
8. Створення редактору для редагування сцени.
9. Дослідження структур, необхідних для створення сцен з можливістю паралельного редагування.
10. Перехід до розглянутих структур.
11. Тестування отриманого результату.
12. Створення Gold master – фінального релізу, котрий піде в реліз.

3.2 Проєктування компонентів системи

На основні проведених раніше досліджень можна розробити високорівневу діаграму майбутнього рушія, котра є максимально приближеною до того, що можливо отримати з мінімальними змінами в процесі ітеративної розробки. Згідно рисунку **19**, котрий описує загальний принцип кількісної взаємодії між клієнтами та сервером, виділяється, що в застосунку є дві компоненти найвищого рівня:

– Серверна, або ж віддалена (хмарна) компонента, зображена на рисунку **20** представлена діаграма високого рівня для сервера. Даний сервер є віддаленим застосунком, розміщеним в хмарі, основною роботою якого є фізична симуляція. Інші компоненти зображені на рисунку відіграють службову роль прийому та передачі команд та даних симуляції.
– Клієнтська складова має під собою рушій та редактор на стороні клієнта, котрий займається іншими задачами, пов’язаними з роботою користувача, а також приймає результати фізичної симуляції. HLD на рисунку **21.**

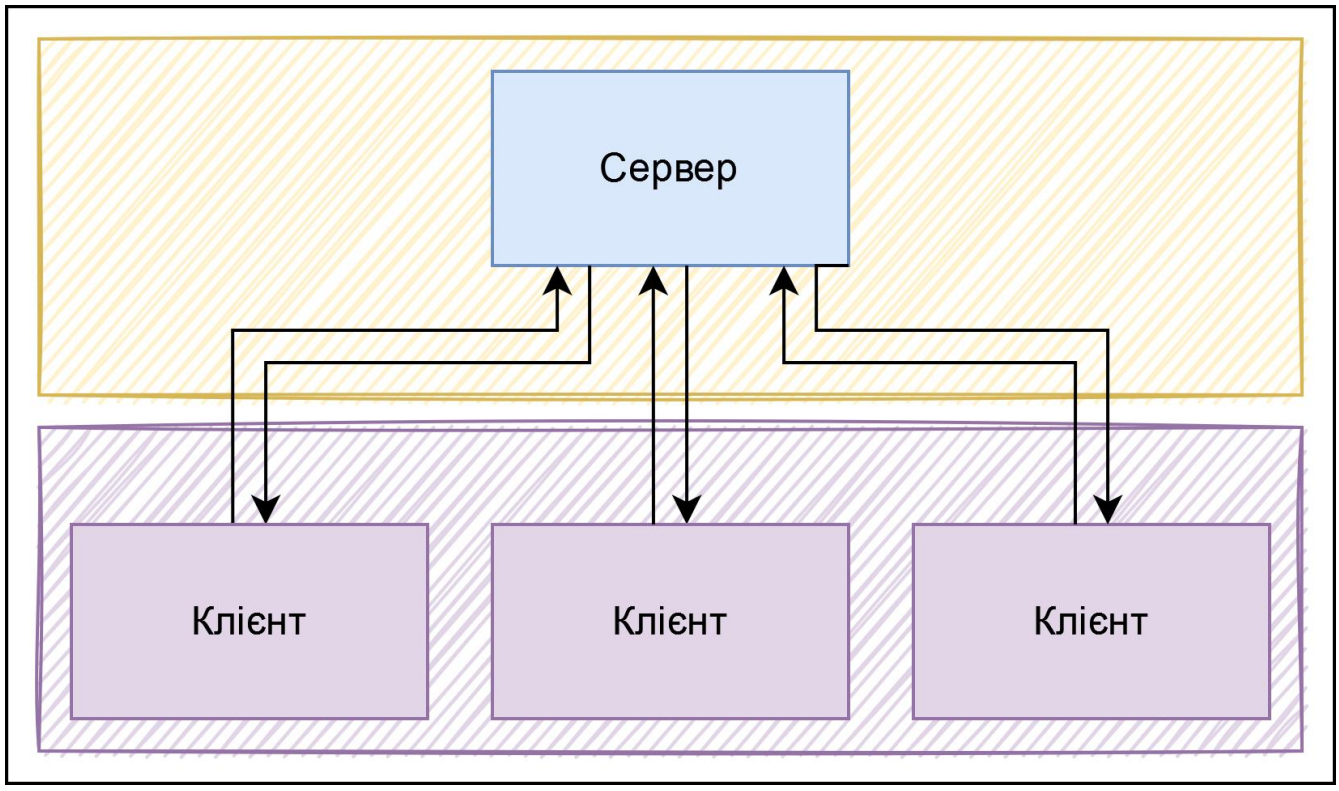

Рисунок 19. Найвищий рівень абстракції системи

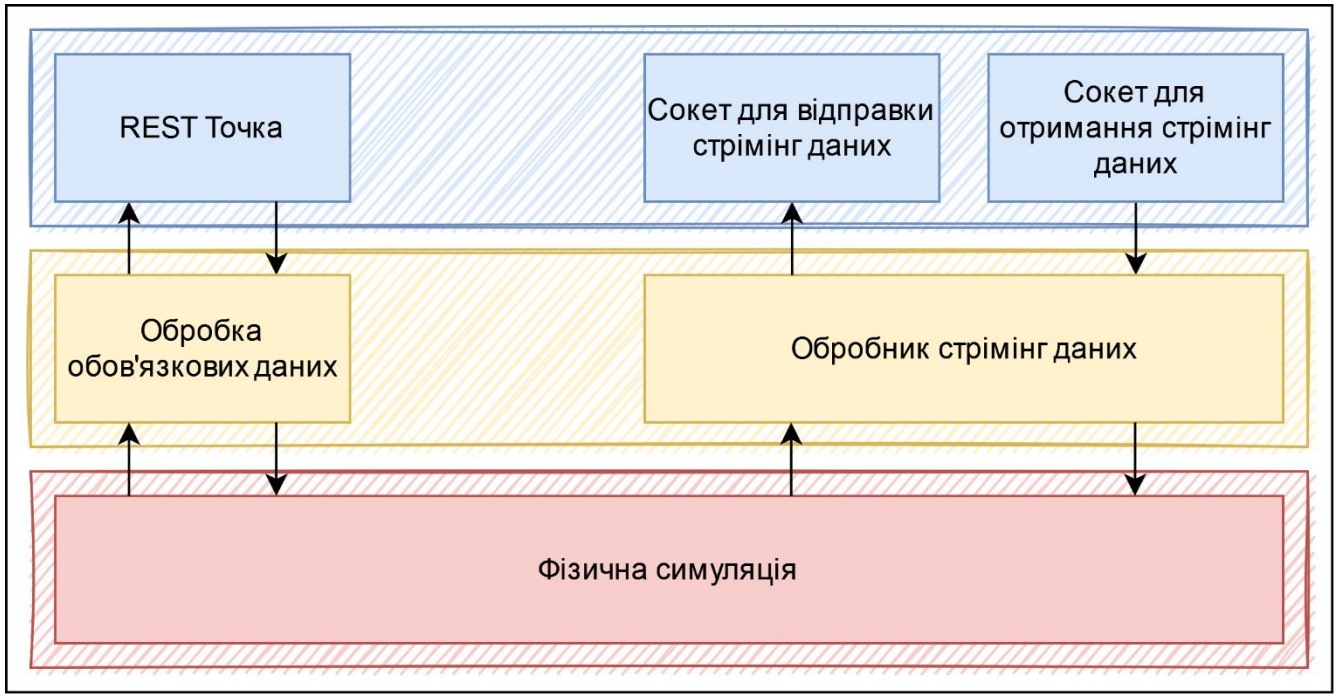

Рисунок 20. Вищий рівень абстракції сервера (хмари)

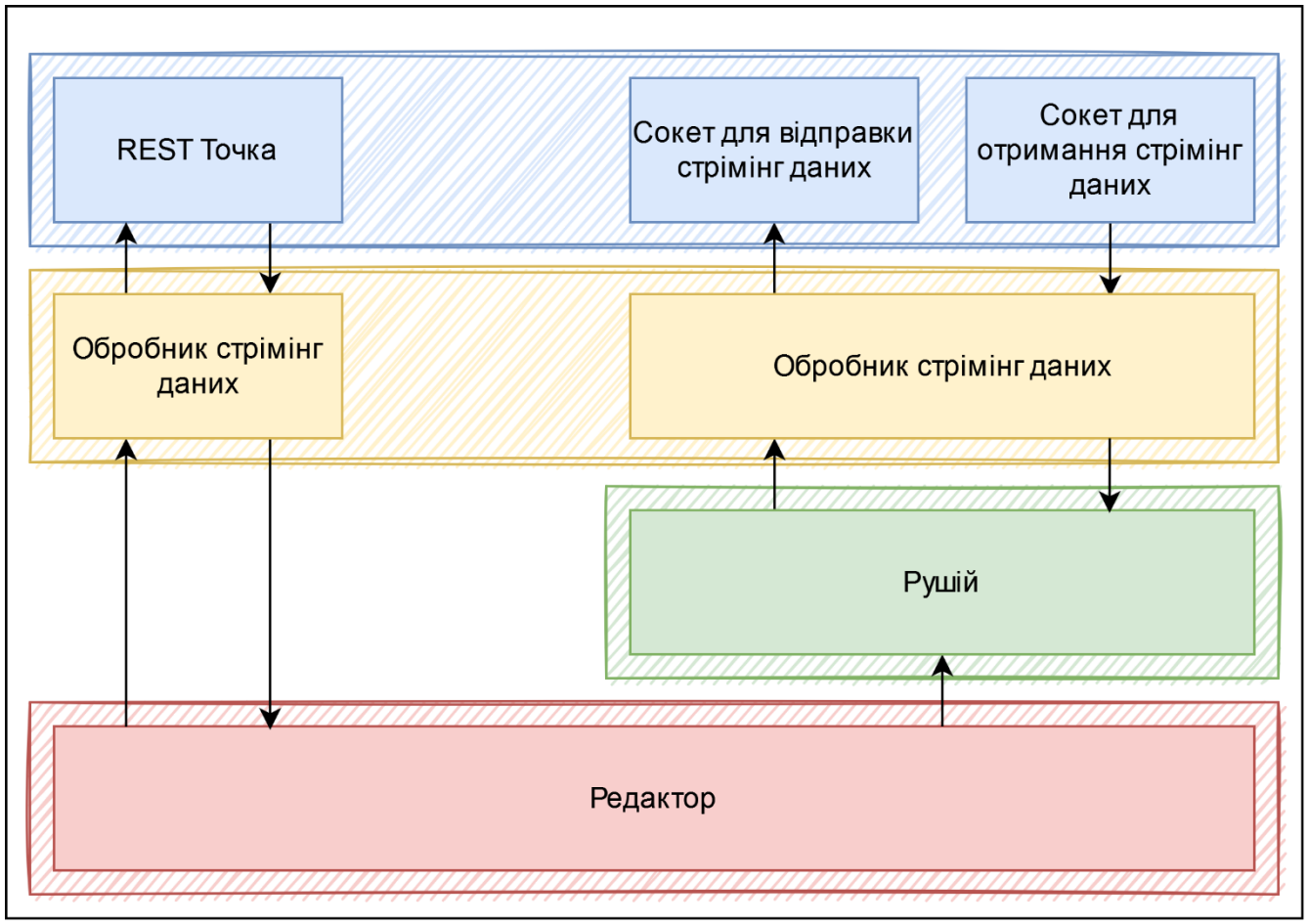

Рисунок 21. Вищий рівень абстракції для клієнта

Слід зазначити, що наведені вище діаграми є концептуальними абстракціями, прийнятими ознайомити та надати ідею щодо того, як повинен виглядати застосунок в цілому, а не конкретно. Для більш конкретних описів необхідно зіставляти спеціальні UML діаграми, даний етап проходиться при конкретному плануванні продукту.

Так як в даному розділі основна увага приділена концептуальному опису, то буде представлена послідовні діаграма, котра повинна надати приблизне представлення про взаємодію елементів на вищому та найвищому рівні між собою та з користувачем. Дана послідовна діаграма представлена на рисунку 22, вона описує процес запуску редактора та підключення до сервера, а також процес

створення та відкриття рівня в редакторі, що є базовими операціями взаємодії користувача з застосунком.

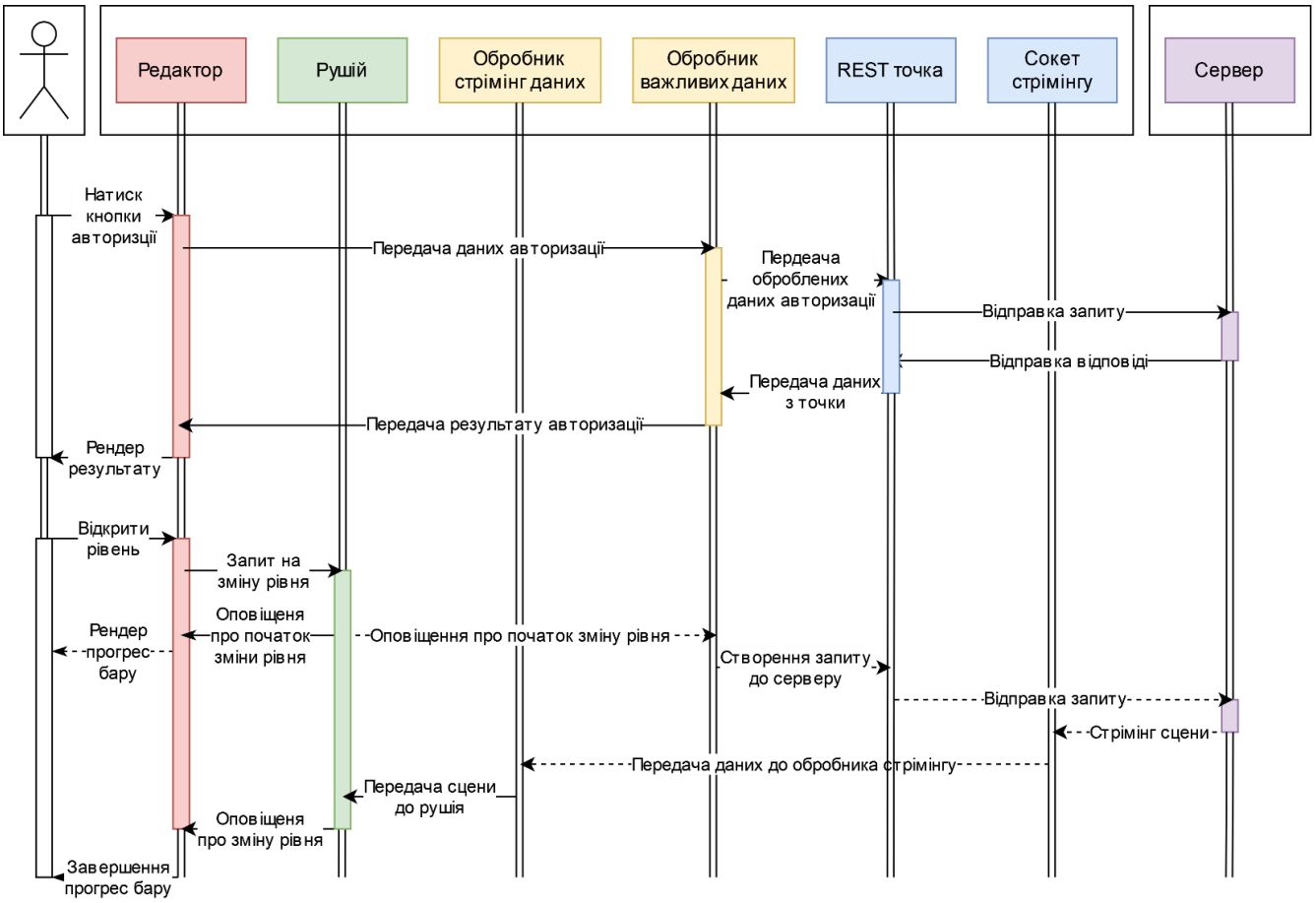

Рисунок 22. Sequence діаграма для частини операцій

Що до об’єктів на сцені слід зауважити, що вони можуть мати різні налаштування через різні підходи до їх симуляції, що вже було згадано в розділі 1 та його підпункту, присвяченому **CCD**.

Засоби редагування та представлення об’єктів на сцені також є важливою складовою, котра має значну роль на роботу та використання рушія, тому їх згадка не може бути проігнорована в контексті виконання даної роботи.

На рисунку 23 зображено Use-Case діаграму, котра описує взаємодію користувача з базовими елементами редактора та рушія для симуляції фізики. Дана діаграма описує саму базову функціональність, котра необхідна для

виконання базових операцій. Дана діаграма не містить повного опису дій, котрі можна виконати з об’єктами на сцені. Для підкріплення слів вище потрібно звернути увагу, що діаграма містить часткові випадки взаємодії користувача з об’єктами на сцені.

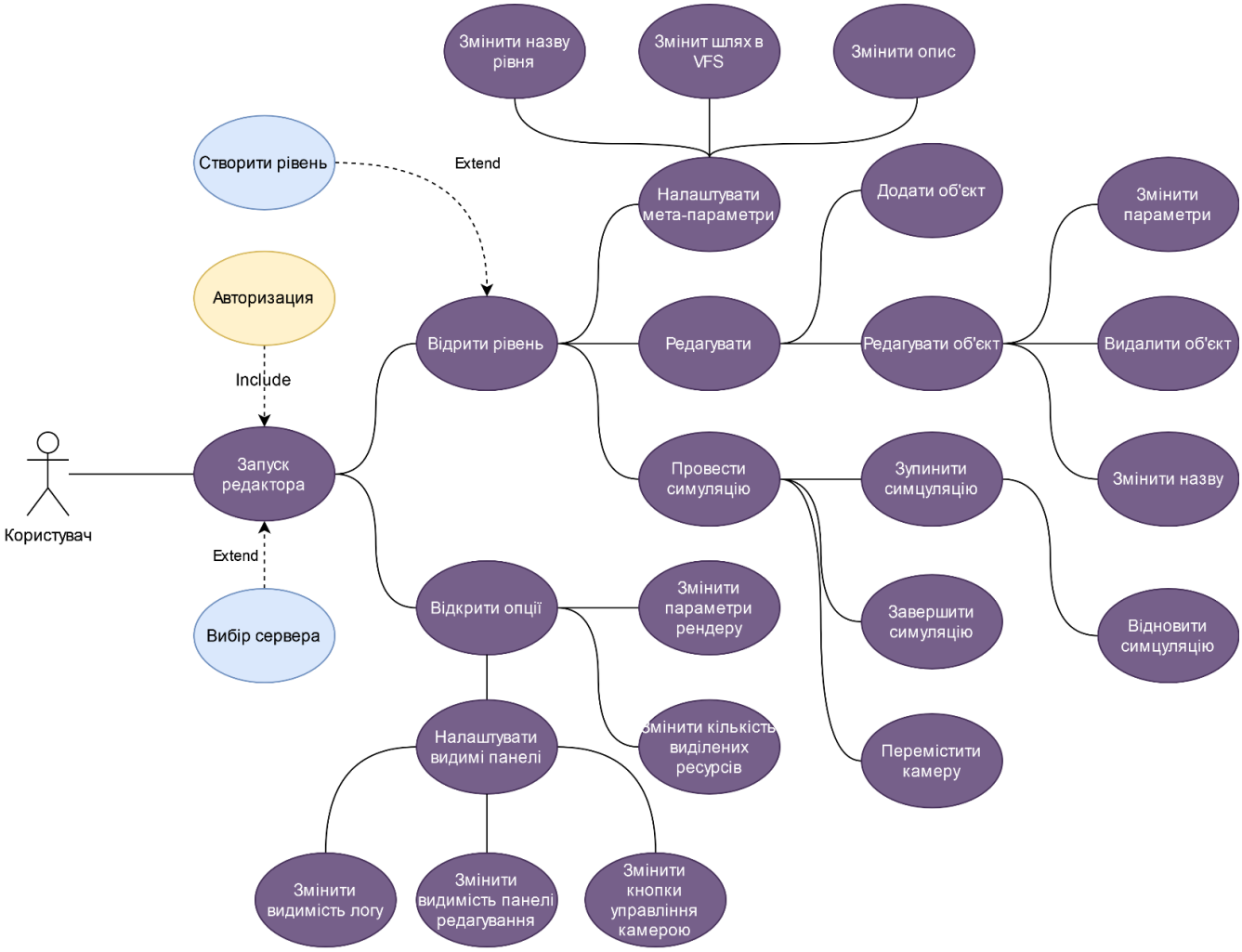

Рисунок 23. Sequence діаграма для частини операцій

3.3 Стек технологій та підтримка якості

Вважається, що система буде розроблятися з використанням мови C++ (стандарт 20) з використанням середовища Visual Studio 2022. Дана мова та середовище буде використане для створення клієнтської та серверної частини системи. Дана мова обрана завдяки доступності великої кількості бібліотек для

роботи з основними елементами системи, вичерпний перелік надано нижче, в списку залежностей:

- **Інструментальний список залежностей:**
    - **C/C++** як стандарт при розробці даного виду систем.
    - **Lua** як мова конфігурацій.
    - **Visual Studio 2022 Community** для розробки застосунку.
    - **Vcpkg** для створення і розповсюдження власних бінарних пакетів бібліотек в мережі, а також за для завантаження та інтеграції відкритих сторонніх бібліотек.
    - **MSVC 143 Toolchain** для останніх версій мови C++ під Win32.
    - **Clang Toolchain** для останніх версій мови C++ під Linux.
    - **CMake** для можливості створення проєкту під Linux сервери.
    - **Perforce VCS** для зберігання коду, контенту та бінарних пакетів.
- **Програмні залежності наступні:**
    - Крос-платформна бібліотека створення графічних інтерфейсі **ImGUI**.
    - Крос-платформна бібліотека прямої роботи з сокетами **SockPP**.
    - Крос-платформна бібліотека створення REST серверів **Casablanca**.
    - Крос-платформна бібліотека для роботи з конфігураціями **TomlCPP**.
    - Крос-платформна бібліотека для роботи з VFS рішеннями **PhysicsFS**.
    - Крос-платформна бібліотека для роботи з YAML **YamlCPP**.
    - Крос-платформна бібліотека для роботи з Lua **Lua Bindings**.
    - Крос-платформне API для роботи з GPU **VulkanAPI.**
    - Windows специфічне API для роботи з мультимедіа **DirectX12**.

Для підтримки застосунку та його безпечної розробки в рамках методології TDD, заздалегідь описано піраміду тестування, котре необхідно буде провести, а також список учасників, необхідних для проведення. Даний список є досить важливим, так як він дозволив описати та знайти найбільш проблемні ділянки в

архітектурі, котрі потребують уваги, перелік вразливих місць системи показано на рисунку Д3.1 та Д3.2 (див. додаток 3).

1. **End-2-End** тести – тестування відбувається відповідальним QA інженером, або ж просто Manual QA. В такому випадку відповідальний запускає редактор і намагається провести основні операції з метою створення рівня і запуску простої симуляції: авторизація, запуск рівня, редагування рівня, запуск симуляції, взаємодія з симуляцією.
2. **Smoke –** тестер запускає редактор і виконує з редактором sanity-check і перевіряє систему на неочевидні способи її використання, котрі були визначні заздалегідь.
3. **Інтеграційне –** проводиться простий запит до пари цільових елементів системи, котрий в цілому перевіряє коректність тестування модулів при взаємодії один з одним.
4. **Контрактне –** тест перевіряє, чи запити відправляються та надаються в правильному форматі. В моїй системі це може бути тестування передачі даних на REST endpoints: перевірка авторизації, а також перевірка того, чи правильно передаються команди при натиску кнопок в редакторі. Також окремо необхідно перевіряти, чи в правильному форматі передаються дані з потому даних. Такі ж тести проводяться окремо і на вихідні дані, для обох: клієнта та сервера.
5. **Компоненте** – автоматизоване тестування перевіряє, що запити, направлені в систему відбуваються коректно, залежності всіх модулів замінені тестовими аналогами**.** Дане тестування можливе для всіх елементів системи, окрім, напевно – рушія, котрий повинен тестуватися окремо меншими частинами.
6. **Модульне –** тестування окремих функцій, класів в системі. Кількість таких тестів в даній роботі є неймовірно велика. Повинно бути протестовано окремо рушій, редактор та сервер.

3.4 Дослідження використання розподілених технологій

Використання розподілених технологій зводиться до використання кластеру машин, а також використання специфічних бібліотек, придатних до цього. В більшості випадків кластери використовуються для наукомістких ситуацій, котрі:

- Не є інтерактивними. Вони є так званими офлайн симуляціями. В даному вигляді симуляцій, роль користувача зведена до мінімуму, умови симуляції повністю детерміновані та залежать від початкових параметрів.
- Такі симуляції досить часто мають досить великий крок моделювання, що сильно зменшує основну розглянуту характеристику виміру швидкодії – FPS. В такому випадку майже ніколи не йдеться про інтерактивність симуляції, або ж про можливість приємно її спостерігати.
- В більшості випадків кластери надають дуже велику обчислювальну потужність, що приводить нас до можливості використовувати більш ресурсоємкі та точні алгоритми. Можливо навіть замінити більшість припущень на пряме використання фізичних формул, без глибокого їх аналізу та створення припущень на основі спостережень та аналізу.

Для дослідження роботи з кластерами було взято рушій Bullet, а також його імплементацію з використанням OpenCL. OpenCL в основному використовується для обчислення на GPU, проте загальна ідея OpenCL полягає в абстрагуванні обчислювальних програм від місця їх виконання, тому існує можливість підміни GPU на кластер, саме такий варіант і розглянемо. Для даної роботи з OpenCL доступно два основних пакети Clara та VCL. [21]

Для того щоб приблизити експеримент до умов, в яких він дійсно може проводитися (припущення, що рушій використовує компанія, котра займається розробкою інтерактивного продукту з фізичним рушієм в своїй основні, та для симуляцій використовує не спеціально виділений кластер машин, а робочі машини працівників, об'єднаний в кластер), то кластер було побудовано з 4-ьох машин

(максимальна можлива кількість машин до використання автором) з наступними конфігураціями CPU, котрі будуть обраховувати задачі:

Таблиця 3.1 – Перша конфігурація машини в кластері

Параметр	Значення
CPU	Intel Celeron J4125
GPU	Відсутній
Кількість ядер CPU	4
Наявність технології Hyperthread	Hi
Тактова частота	2.0 – 2.7
ОЗУ	4 Gb

Таблиця 3.2 – Друга конфігурація машини в кластері

Параметр	Значення
CPU	Intel i7-10700K
GPU	NVidia 2060 Super
Кількість ядер CPU	8
Наявність технології Hyperthread	Так
Тактова частота	3.8
ОЗУ	32 Gb

Для виконання обчислень було використано три машини з таблиці 3.1 та одну машину з таблиці 3.2. ЕОМ з таблиці 3.2 є також хостом обчислень і також є частиною кластеру.

Для проведення експерименту було обрано стандарту сцену з пакету демо-сцен бібліотеки Bullet під назвою API\Kinematic Body, сцена зображена на рисунку 24.

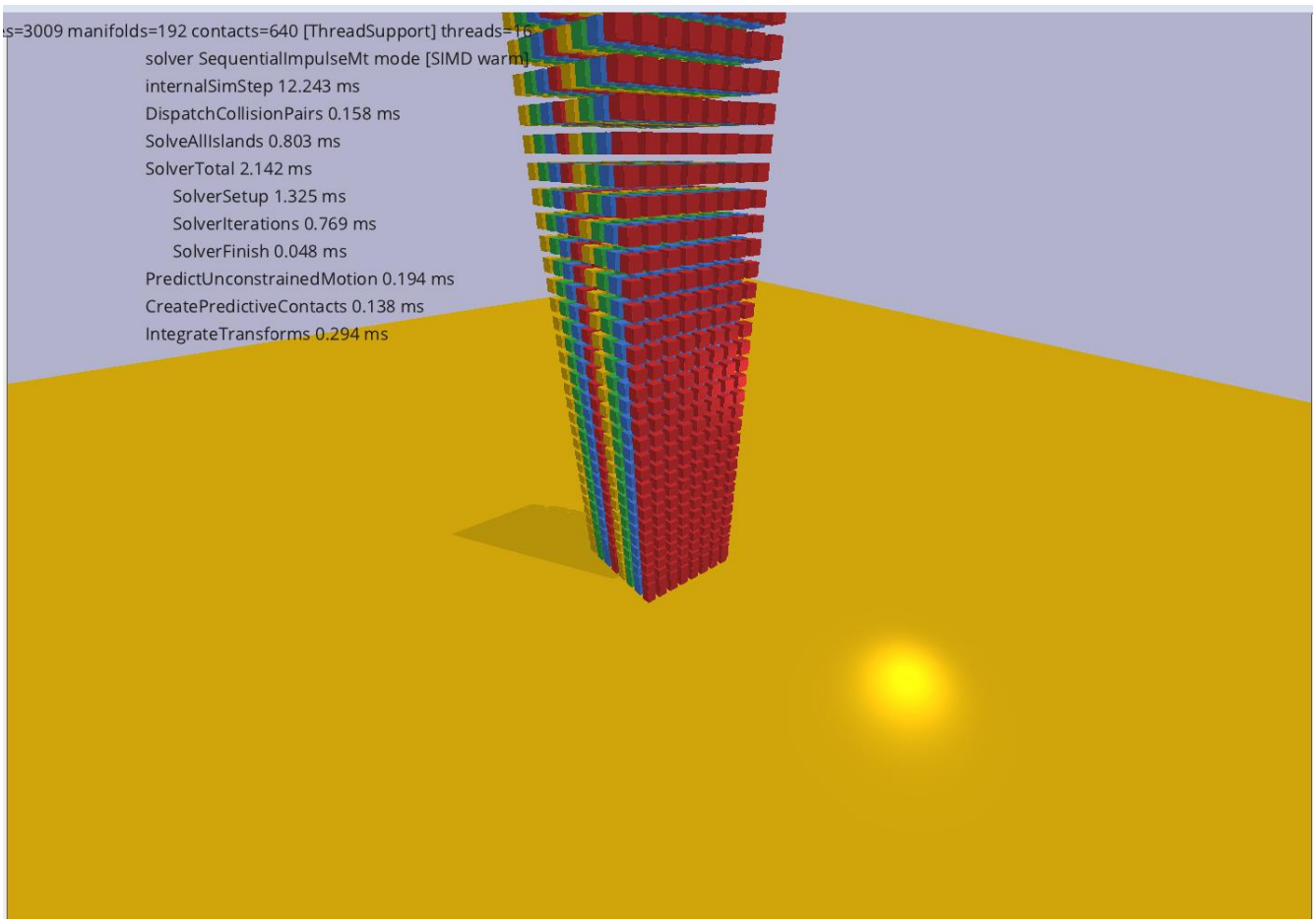

Рисунок 24. Сцена на момент початку симуляції. (з обрахунку на кластері)

На сцені з рисунку 24 відбувається наступне: куб котрий складається з 3000 менших кубічних тіл падає на землю та поступово розпадається на шматки, спричиняючи експоненціальне збільшення кількості контактів за кадр. Після розпадання на шматки у користувача з’являється можливість перетаскувати кубічні тіла.

Переглянути фінальний результат моделювання сцени Benchmark можна на рисунку 25, наведеному нижче (результати кожного моделювання є детермінованими та однаковими, до тих пір поки користувач не втрутиться сам). Час проведення симуляції – 10 секунд.

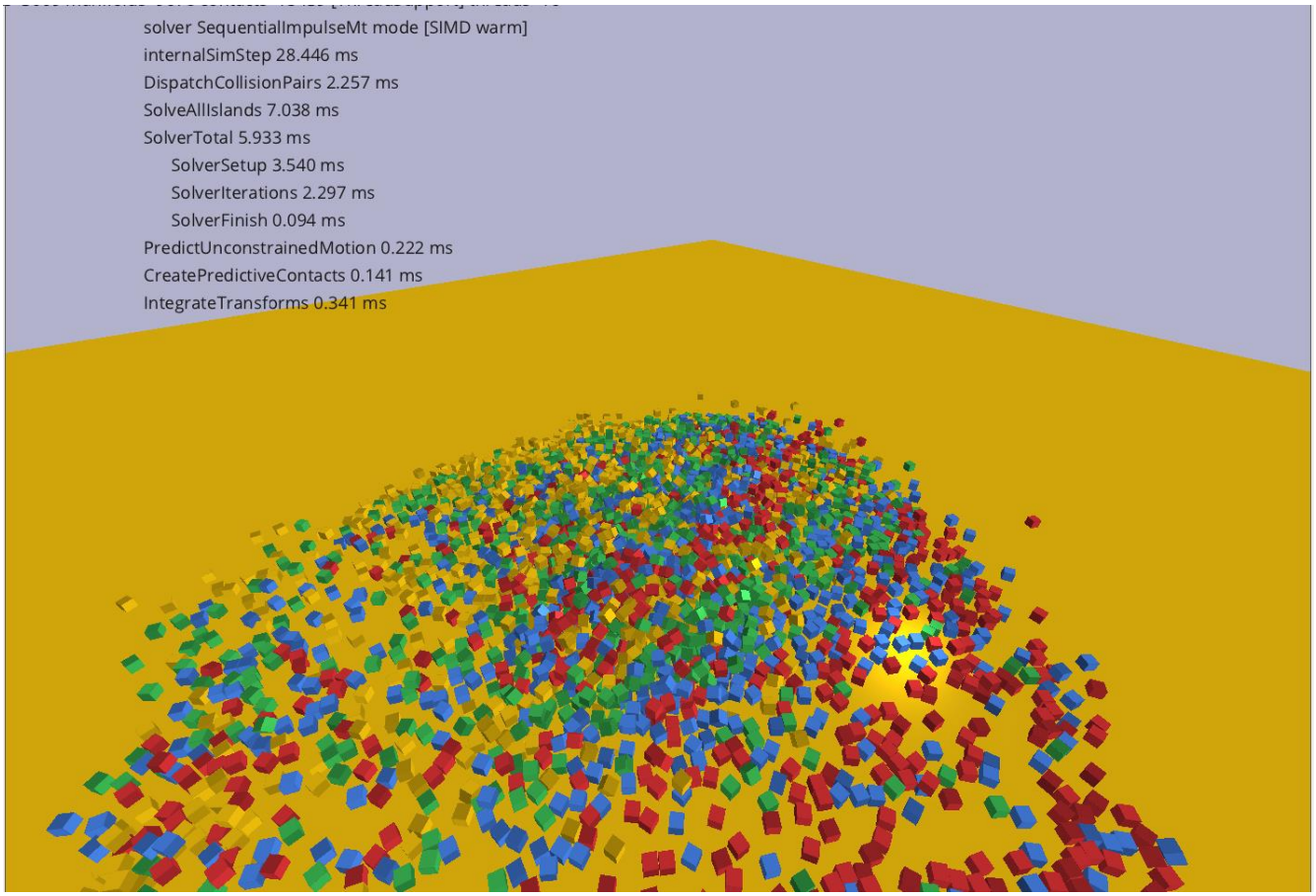

Рисунок 25. Кінець симуляції сцени, тіла в стані спокою (з обрахунку на кластері)

На наведеному вище прикладі було виміряно частоту кадрів на різних конфігураціях. Конфігурації наступні:

- Виконання на одній машині з таблиці 3.2 (виконання на наявній ЕОМ)
- Виконання на одній машині з таблиці 3.1.
- Виконання на кластері з чотирьох машин (хост та три сервера з таблиці 3.1)
- Виконання на кластері з трьох машин (три сервера з таблиці 3.1)

Кроки експерименту наступні:

1. Модифікація бібліотеки Bullet для роботи з кластером OpenCL.
2. Розгортання кластеру OpenCL.
3. Запустити виконання бенчмарк-сцени з рисунку 24 на кластері з однієї машини з характеристиками з таблиці 3.2.
4. Повторити крок 3 для кластеру з одного серверу.

5. Повторити крок 3 для кластеру з чотирьох машин.
6. Повторити крок 3 для кластеру з трьох серверів.
7. Звірити отримані результати у формі графіку та надати висновки за отриманими даними та результатами спостереження за симуляцією бенчмарк сцени.

Історія FPS зберігалася в історії (в пам'яті програми) та виводилася в файл формату **.csv** по завершенню її симуляції. Далі результати було відрендерено за допомогою бібліотеки мови **python** – **matplotlib**. Графіки представлено на рисунках 26 та 27. На графіках можна побачити, як у випадку розподілених обчислень під час симуляції поведінки змінюється FPS на протязі часу.

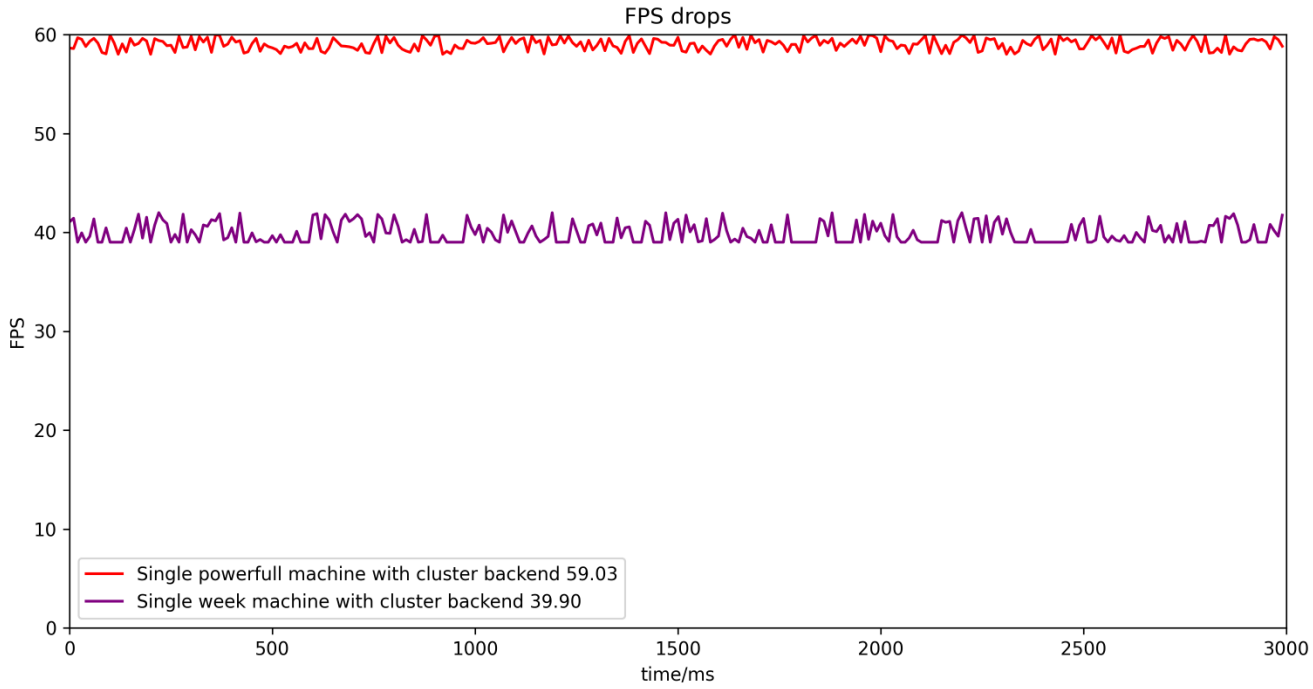

Рисунок 26. Графік залежності FPS від часу симуляції при обчисленні на різниз машинах з використанням OpenCL бекенду.

На графіку, представлено на рисунку 26 показано зміну FPS з часом при обрахуну симуляції з використанням OpenCL бекенду для однієї машини з

різними конфігураціями (червоний – один потужний ЕОМ з використанням OpenCL, пурпурний – один слабший ЕОМ з використанням OpenCL бекенді).

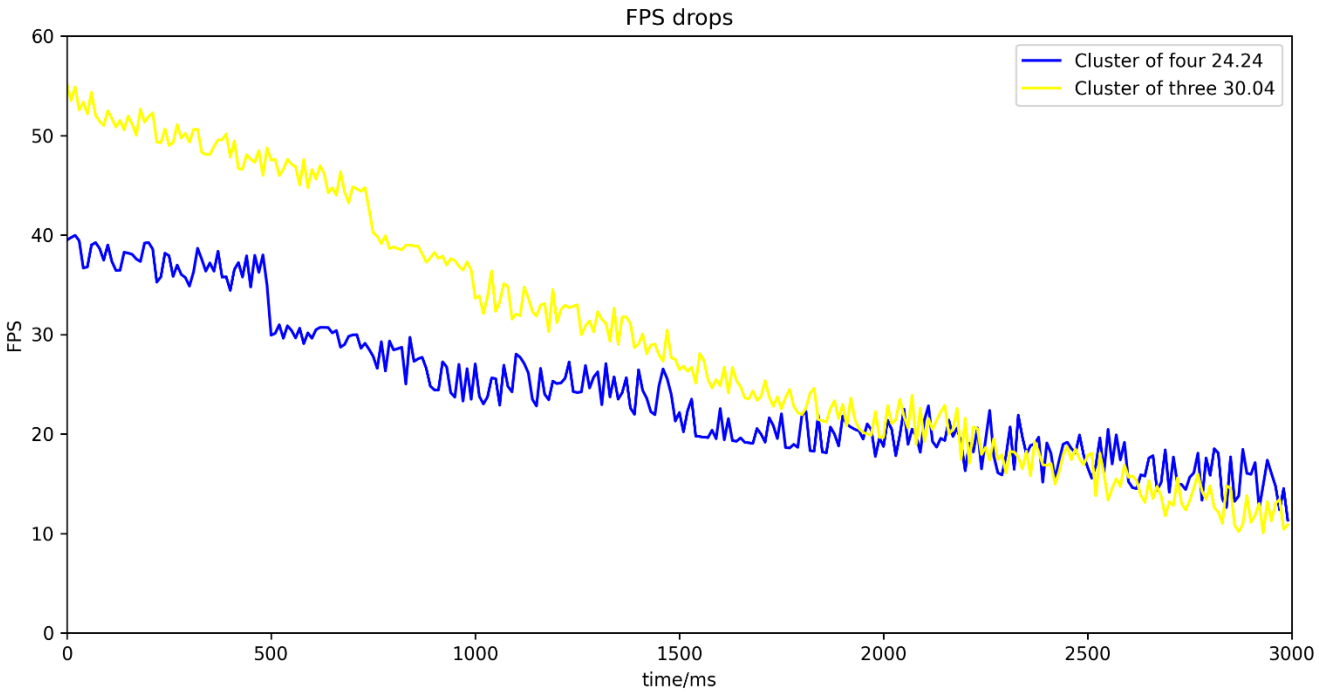

Рисунок 27. Графік залежності FPS від часу симуляції при обчисленні на кластері

На графіку з рисунку **27** показано зміну FPS з часом при обрахунку на кластерах з 3 (три машині з таблиці 3.1, жовтий колір) та 4 (три машини з таблиці 3.1 та одна машина з таблиці 3.2, сині колір) машин.

Виходячи зі спостереженнь за симуляцією було заключено, що найкращий результат отримується при запуску симуляції в режимі роботи на CPU без кластеру, також гарний результат показує такий же режим при запуску симуляції на сервері. Видно, що симуляції в обох випадках є стабільними та коливаються в рамках декількох кадрів – 1-4, що є допустимим, особливо на слабкій конфігурації сервера. Ситуація набагато гірша при використанні кластеру, можна заключити, наступне:

1. Багато ресурсів витрачається на передачу даних, до цього заключення можна прийти з таких спостережень: виконання на одній машині є стабільним. При чому на машинах з різними конфігураціями, зі збільшенням

кількості машин просадка FPS збільшується. У такому випадку, можливо, треба зменшувати кількість даних, що передаються між машинами.

2. Видно, що відбувається сильна просадка під час розпадання кубу (500 – 1000 мілісекунд), в даний момент починає оброблюватися особливо багато колізій та передаватися особливо багато інформації.
3. Видно, що на початку FPS є високим, так як тіла рухаються відносно лінійно та не стикаються, спричиняючи мінімальні колізії.
4. В кінці симуляції FPS стабілізується, що викликане станом спокою об’єкту, проте деякі об’єкти все ще можуть переміщатися та рухатися, деякі об’єкти знаходяться постійно поверх інших та вимагають обрахунку обмежень, що не дозволяє отримати ті ж самі значення FPS, що й на початку симуляції. Відображення графіку зупинено після 3000 мілісекунд через стабілізацію результатів.

Всього на сцені знаходилось близько 3000 об’єктів, що є досить великою кількістю об’єктів з повноцінною фізичною симуляцією. В кінці симуляції тримається приблизно 18500 активних контактів між тілами, в той час як на початку симуляції їх було 650. В режимі роботи на одній машині використовується приблизно 16 потоків для підтримки симуляції в 60 FPS. Незважаючи на те що результати кластеру були не дуже втішні, при оптимізації передачі даних та вибору більш однорідних та трохи потужніших агентів можна отримати достойні результати.

3.5 Дослідження використання хмарних технологій

Тепер дослідимо використання серверної архітектури, для цього візьмемо приклад з рисунків 24 – 25 та спробуємо просимулювати його поведінку, розмістивши симуляцію повністю на серверній стороні. Для цього відмикаємо всі зайві та косметичні ефекти, щоб залишити лише симуляції, котрі нас цікавлять. В

даному пункті метою є дослідити вплив таких чинників на симуляцію з використанням технології віддаленого сервера:

- Вплив віддаленості сервера за однакових пропускних спроможностей на симуляцію, тобто – перевірка залежності симуляції від затримки.
- Перевірка потужності сервера на якість симуляції, та наскільки погіршення параметрів в цілому змінить якість симулювання.

В попередньому пункті цього не було згадано, але в якості алгоритму розбиття простору було обрано розглянутий в даній роботі **DBVT**. Кількість тіл на даний момент все теж саме – 3000. Тепер спробуємо перевірити вплив затримки на швидкість роботи. Кроки експерименту наступні:

1. Провести еталонні тестування на потужній ЕОМ з таблиці 3.2 та слабкому сервері з таблиці 3.1. Використовувати дані з пункту 3.4 (рисунок 26) не є правильним кроком, так як в розглянутому раніше експерименті було використано бекенд OpenCL, котрий впливав на стабільність.
2. Провести тестування, розгорнувши на потужному ЕОМ сервер, а на іншій ЕОМ розгорнути клієнт, котрий буде отримувати результати симуляції. Заміряти результати, виміряти затримку в кожен момент симуляції, за прикладом експерименту в пункті 3.4
3. Уповільнити якість з’єднання та повторити крок 2 з новою затримкою.
4. Повторити кроки 2 та 3 зі слабкішим сервером з характеристиками з таблиці 3.1 та порівняти результати у графічному вигляді, проаналізувати отримані дані та надати висновок по ним.
5. Повторити крок 2 з сервером на комерційній платформі, наприклад, Heroku.

Результати виконання кроків, наведених вище, представлені на рисунку 28. На рисунку 28, червоним кольором позначено FPS при запуску симуляції на потужній клієнтській ЕОМ (таблиця 3.2), синім – на слабкій ЕОМ (таблиці 3.1). Помітно, що симуляція стабільна в плані знаходження в межах деякого значення FPS, в межах 60-ти та 43-х FPS. Результат кроків 2 та 3 показано на рисунку 29.

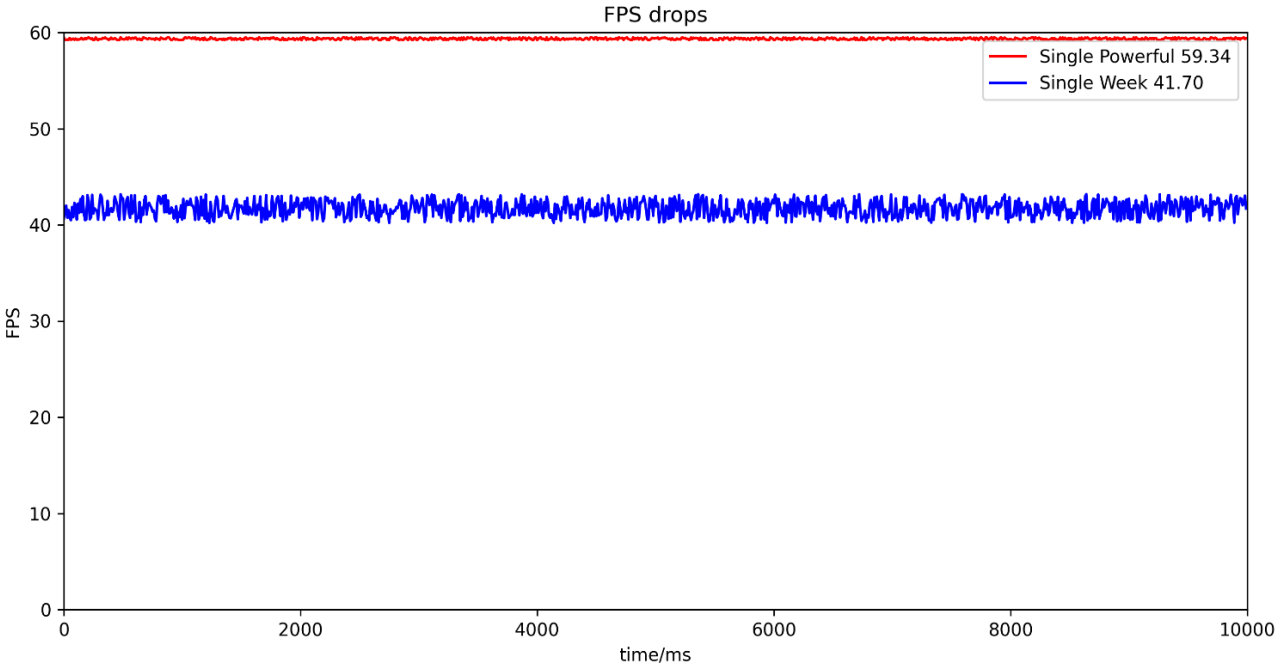

Рисунок 28. Проведення симуляції на клієнтській машині

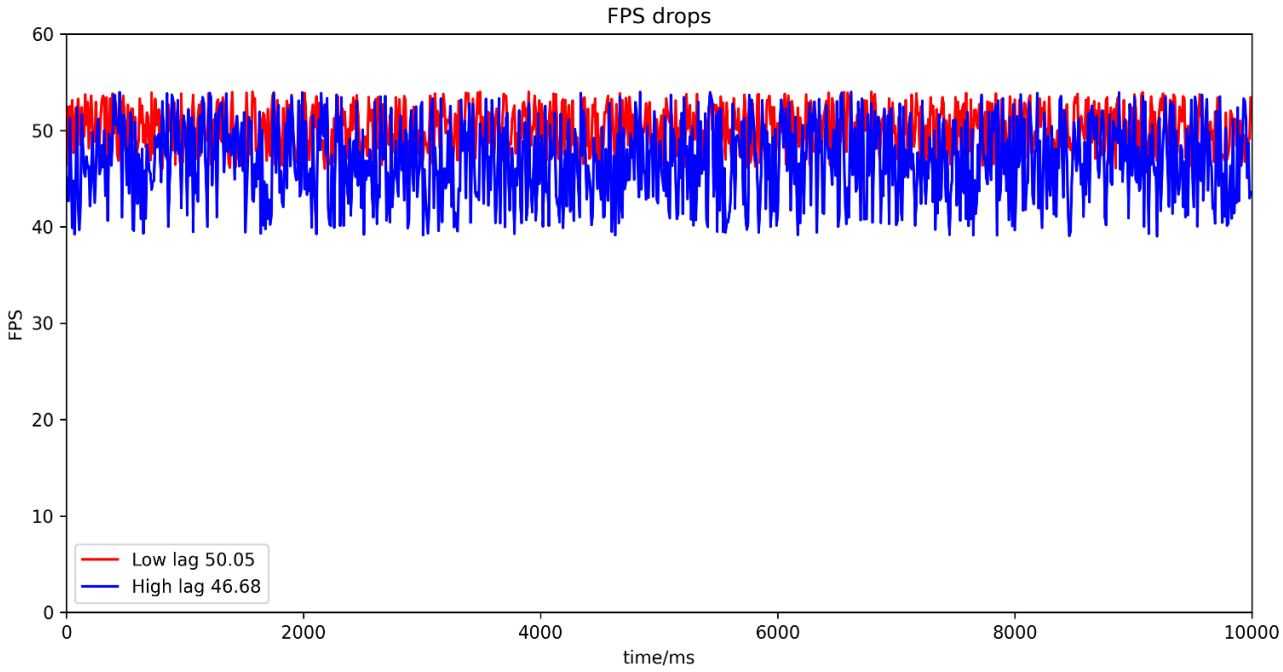

Рисунок 29. Проведення симуляції на потужній машині з різною затримкою

На рисунку 30 зображено графік затримок, котрі виникали відповідно на рисунку 29. Сині та червоні кольори відповідають тим же значенням, що і раніше. Проаналізуваваши рисунки 29, 30 стане помітно, що зі збільшенням затримки

медіана майже не змістилась, девіація збільшилась. Ріст затримки впливає на симуляцію: збільшується час очікування, що спричиняє зростання магнітуди FPS.

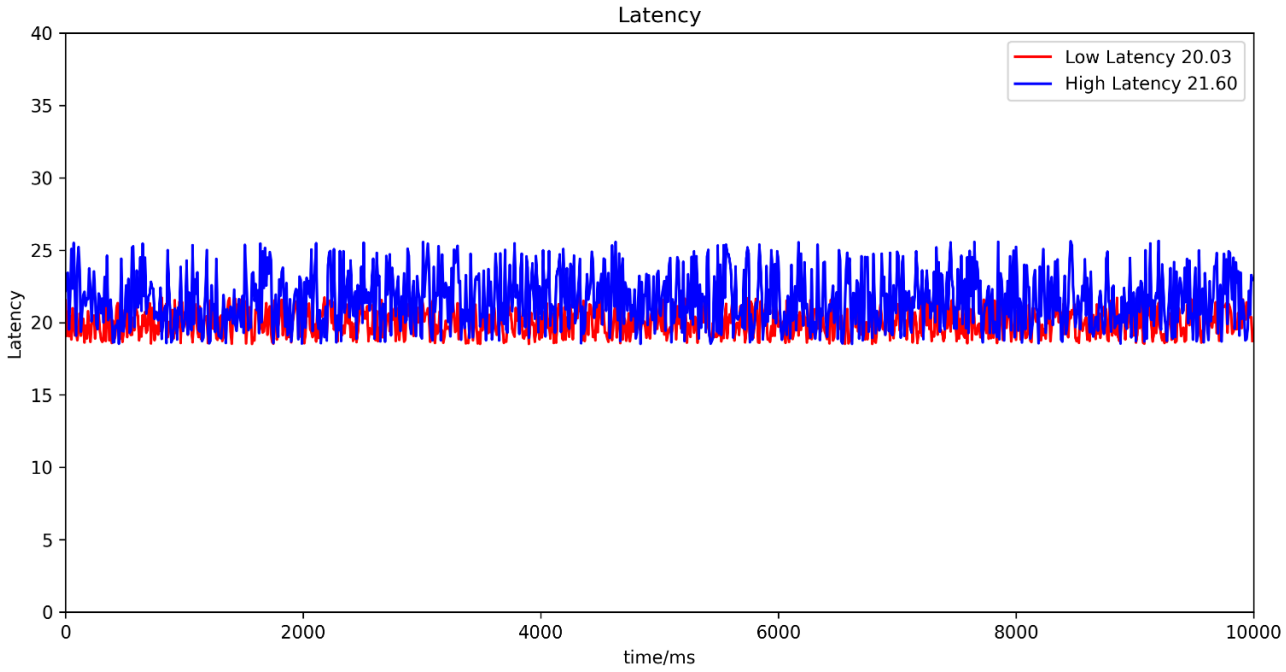

Рисунок 30. Графік зміни затримки з часом для потужної машини

На рисунках 31, 32 наведені відповідні графіки FPS та затримки для слабшої ЕОМ.

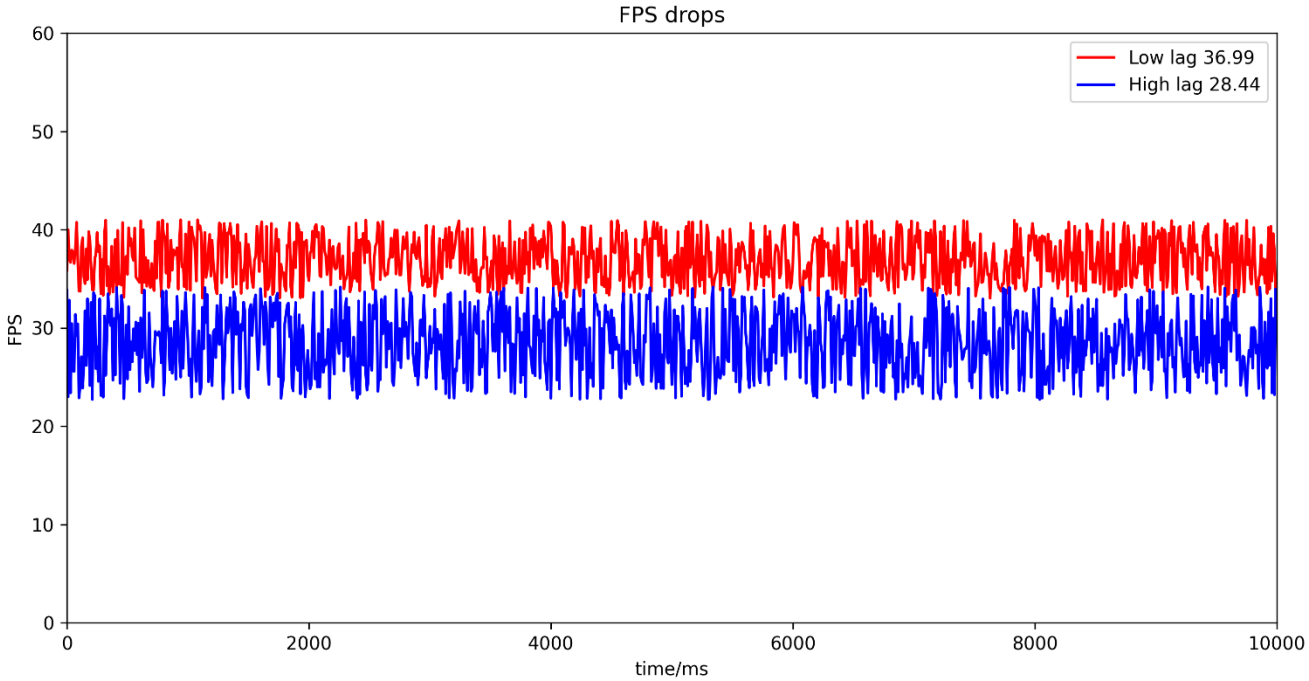

Рисунок 31. Проведення симуляції на слабшій машині з різною затримкою

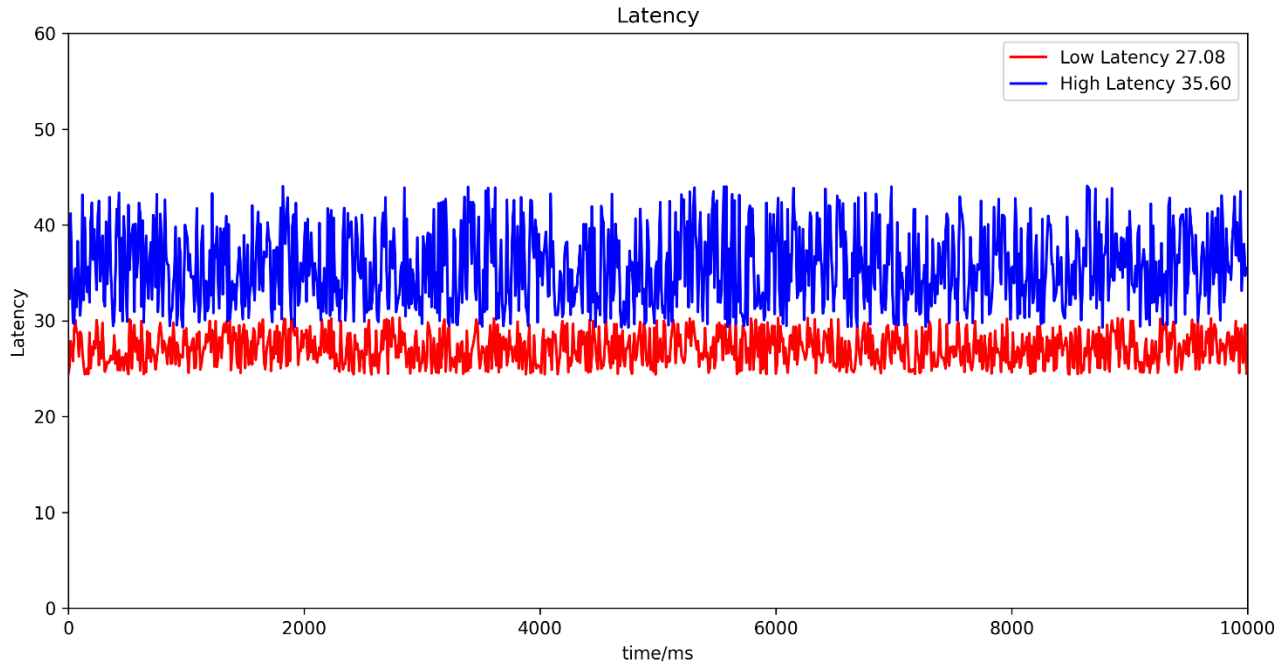

Рисунок 32. Графік зміни затримки з часом для слабшої машини

Тепер необхідно повторити даний експеримент використавши досить віддалений сервер, для цього використаємо комерційний сервер Heroku. Результати виконання експериментів на ньому представлені на рисунках 33 та 34.

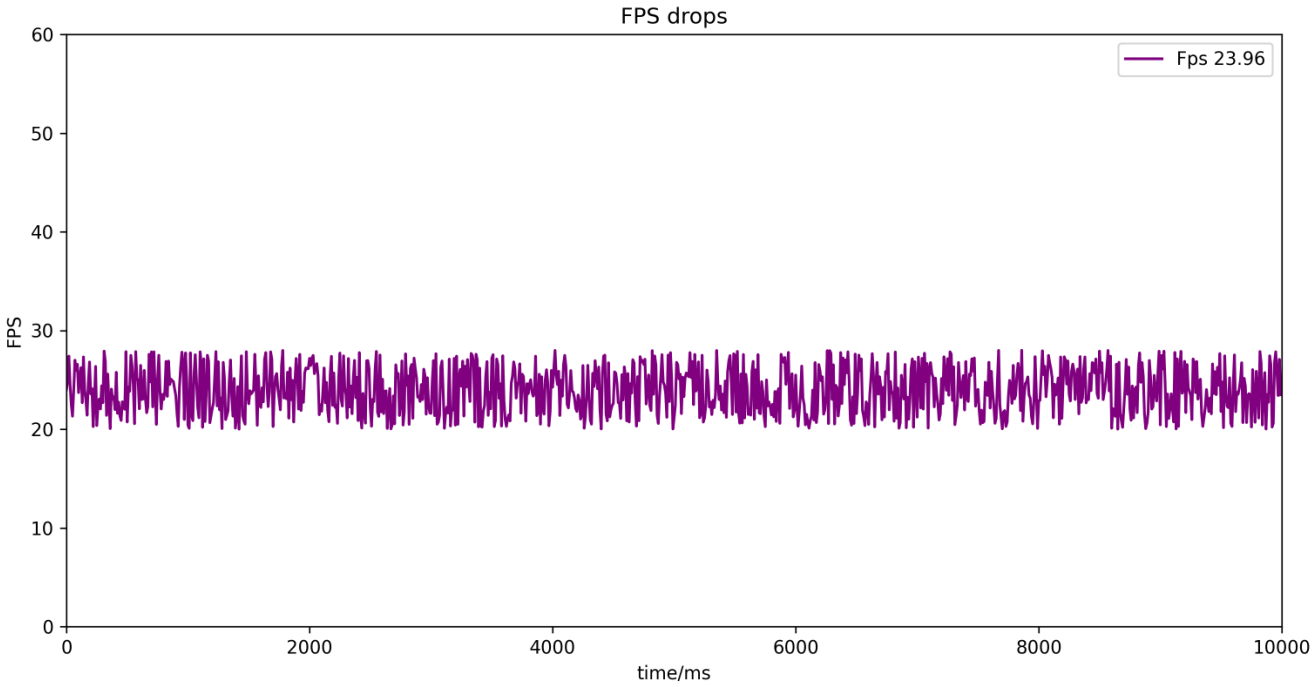

Рисунок 33. Проведення симуляції в хмарі Heroku (FPS)

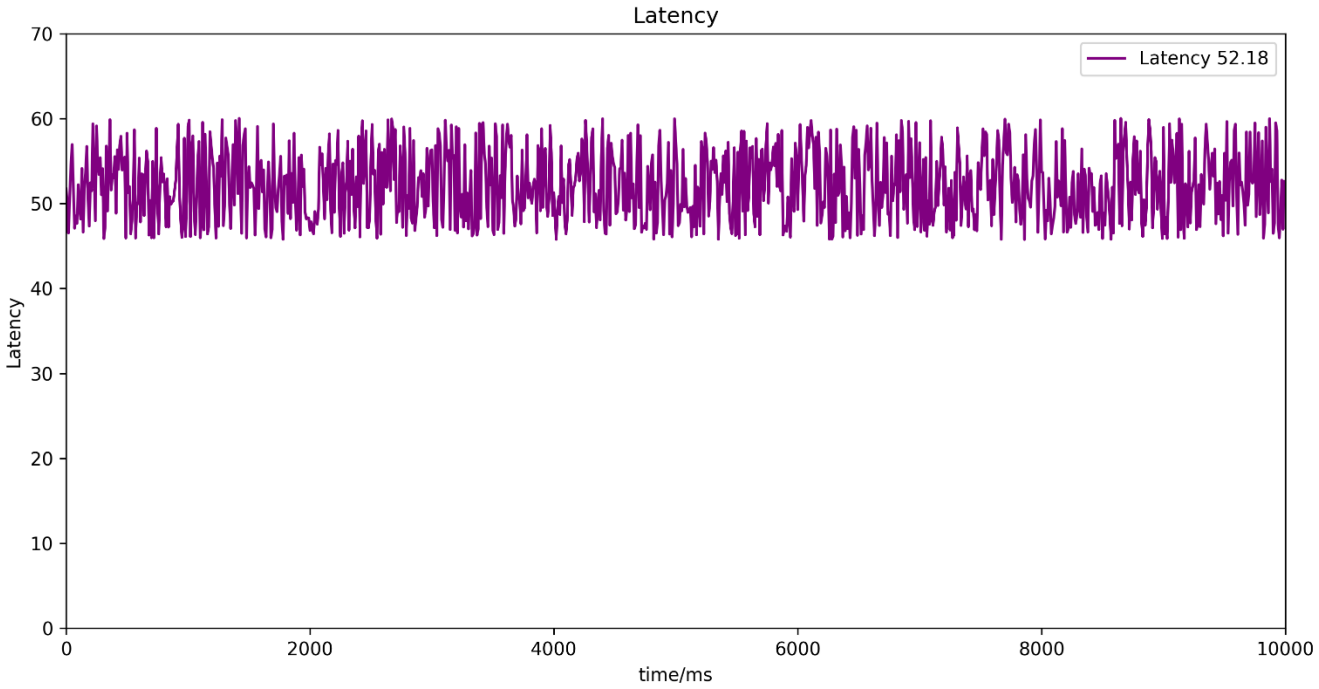

Рисунок 34. Проведення симуляції в хмарі Heroku (затримка)

3.6 Висновки до розділу 3

В процесі роботи над даним розділом було проведено практичну та теоретичну роботу щодо дослідження та прототипування архітектури системи, котра повинна використовуватися для подальшої її розробки. Детальний опис отриманих результатів наступний:

- Завдяки попереднім розділам було визначено конкретні цілі, котрі повинні бути досягнуті, а також надані конкретні критерії, котрі можна виміряти в тому чи іншому вигляді, за якими можна оцінювати стан готовності системи.
- Було надано план розробки повноцінного рушія та редактора з можливістю проведення симуляції з використанням хмарних технологій. За даним планом можна визначити ключові віхи та елементи. Частина наведених віх вже пройдена в результаті даної роботи.

- Надана проєктна та технічна документація щодо планування розробки. Також надано документацію користувача у вигляді послідовних діаграм та use-case діаграм.
- Наведено дерево проблем та рішень, котрі є підсумком всього дослідження, також надано діаграми проблеми та критичних точок архітектури, надано діаграми високого рівня для опису архітектури.
- Надано детально описаний стек технологій, котрий є оптимальним для виконання поставленої задачі, в даному стеку було перераховано: мови програмування, бібліотеки та фреймворки, інструментарій та платформи, на котрих бажано проводити розробку.
- Разом зі стеком технологій представлено опис етапів контролю якості системи у вигляді піраміди тестування з описом конкретних проблем та способів їх виявлення на кожному етапі піраміди.
- Створено на надано список критичних точок архітектури, також надано їх категоризацію за типом, а також за складністю боротьби з ними. Поставлено, що більшість з них є проблемами, котрі виникають у всіх застосунках з клієнт-серверною архітектурою.
- Проведено модифікацію та доопрацювання обраної технології Bullet для проведення подальшого дослідження. В результат було проведено дослідження на тестовій сцені з симуляцією 3000 тіл та в середньому 18000 контактів за кадр.
- Описане вище дослідження проводилося як для розподілених обчислень (зв’язок з кластером відбувалася через бекенд OpenCL), так і за допомогою клієнт серверної архітектури, котра намагалася в тій чи іншій мірі симулюватися поведінку системи Omniverse та принцип її роботи.
- Проведено порівняння між використання хмарних та розподілених технологій при моделюванні фізичних явищ. Наведено теоретичні плюси та

мінуси використання того чи іншого підходу, які в подальшому дослідженні були підкріплені практично.

– Отримані результати наведені у вигляді графіків, котрі описують зміну FPS з часом для кластерів різного розміру, а також для одиночних ЕОМ різної потужності. Для хмарних обчислень (власні сервери різної потужності, віддалені сервери на основі Heroku) також наведено дані графіки, в додаток до чого наведені графіки впливу затримки передачі даних на FPS. Перед аналізом даних варто зазначити, що 3000 одночасно активних фізичних об'єктів на сцені це багато, середня симуляція на практиці має до 1000 активних фізичних об'єктів в кадрі, кількість контактів між ними значно менша 18000. В результаті зроблено наступні висновки:

  - Кластерні обчислення вимагаються знаходження на сцені великої кількості об’єктів, набагато більшого порядку ніж використовується на практиці, більше ніж було використано в наведеному вище експерименті. При дослідженні на 3000 тілах багато часу витрачається на передачу даних між вузлами кластеру, про що свідчить зменшення швидкодії при збільшенні кількості вузлів. Тому потрібно або оптимізувати кількість даних, що передаються, для проведення симуляції, або збільшувати швидкість каналів, або ж використовувати ще більшу кількість вузлів, для зменшення кількості даних на один вузол за рахунок їх розподілу між більшою кількістю вузлів. В останньому випадку можна буде зменшити вимоги до апаратного забезпечення, адже кількість операцій на вузол зменшиться. Також варто зазначити на деяку недосконаліть технології, використаної для зв’язку з кластером, якщо порівняти графіки з рисунків 19 та 20 то витрати на використання кластерного АПІ наступні: для потужної ЕОМ втрата FPS складає 0.25 середнього FPS, що є приблизно 0.004%, а для слабшої ЕОМ складає 0.84 середнього FPS, що є приблизно

0.02%. Також варто зазначити що у випадку кластеру FPS є менш стабільним. Девіація від середнього для кластерного бекенду навіть з однієї ЕОМ складає близько 6 FPS, а просто запуск на симуляції має відхилення в менш ніж 1 FPS. Для слабшої ЕОМ дані значення мають показники в 7 FPS та 3 FPS. Тобто можна заключити, що саме по собі використання API OpenCL додає затрати на абстракцію.

- Хмарні ж обчислення мають стабільні результати, котрі показують, що вже навіть зараз при використанні навіть трохи модифікованої відкритої технології Bullet можна досягнути непоганих результатів при симуляції навіть більшої ніж практично використовуваної великої кількості об’єктів. Бачимо, що використовуючи сервер, котрий знаходиться в локальній мережі однієї будівлі та при досить невисокій затримці (на затримку впливає не тільки стан мережі, а й стан ЕОМ, використаної для серверу) в 20 – 50 мс можна отримати практичні результати в районі 50 та 40 FPS в залежності від потужності самого сервера. У випадках обох (потужної та слабшої ЕОМ з таблиць 3.2 та 3.1. відповідно) можна констатувати втрату в середньому 15-ти FPS при переході до серверної обробки фізики, ще приблизно 25%, проте навіть з такими великими числами слід вважати, що великий відсоток викликаний скоріше недосконалістю та станом прототипу використаної технології. Дані результати є дещо гіршими при використанні комерційного серверу Heroku, проте тут грає роль декілька факторів: навіть при високій затримці сервер знаходиться досить далеко від клієнту, а по-друге – потужності, виділені інфраструктурою Heroku є меншими ніж на ЕОМ з таблиці 3.1. Також варто розглядати ці дані в контексті того, що зазвичай фізичні симуляції на ЕОМ користувача проводяться в стабільних 30 FPS, тому якщо середні значення не виходять за дані рамки, то результати є

успішними. В додаток до вище наведеного – в більшості великих проєктів час оновлення фізики дорівнює взагалі 1 – 5 FPS в секунду.

4 РОЗРОБКА СТАРТАП ПРОЄКТУ

4.1 Опис ідеї стартапу

Використання клієнт-серверних архітектур в системах з використанням фізичних рушіїв в загальній своїй ідеє не є новою, проте вона досить довго зводилася лише до використання у вигляді необхідності у випадку коли створювався медіаконент, частиною якого є взаємодія між багатьма людьми. Іде я ж використання даного підходу в процесі самої розробки продукту є досить нова і на даним момент вона виходить на ринок досить активно. Поява таких систем може значно скоротити затрати компаній різного масштабу на закупівлю робочих станцій для спеціалістів різного гатунку: дизайнери (технічні), артисти (художники, аніматори), технічні спеціалісти – всі вони потребують потужних робочих станцій з потужними GPU та CPU на борту. Використання даного клієнт-серверно орієнтованого підходу дозволить централізувати використання даних ресурсів, та зменшити оптимізувати їх в тих відділах, котрі частіше за все працюють разом над спільними елементами. На даний момент відомо про одного потужного розробника, котрий зацікавлений в розвитку даного напрямку, а також декількох менших компаній, котрі надають послуги в схожій площині. В подальшому в даному розділі буде наведено економічно-технічний аналіз можливості створення стартапу на основі даної ідеї в умовах надвисокої конкуренції. Даний розділ буде слугувати звітом щодо такої можливості, матиме перелік стратегій у випадку різної поведінки та динаміку ринку, зацікавленості клієнтів та інших факторів, котрі можливо прорахувати. Першим кроком при проведенні аналізу можливості створення проєкту це формалізація ідея та змісту, для цього побудовано таблиці 4.1, котра є формальним описом ідеє стартапу, та котра містить основні напрямки використання продукту користувачем та вигоди, котрі він отримає від використання.

Таблиця 4.1 – Опис ідеї проєкту

Зміст та ідея	**Напрямки застосування**	**Вигоди для користувача**
Створення системи для проведення фізичних симуляцій з можливістю одночасного редагування віддаленими клієнтами в режимі реального часу. Система має клієнт-серверну архітектуру, де сервер відповідає за проведення симуляції та синхронізацію стану між різними клієнтами, а клієнтська частина надає зручний інтерфейс для внесення змін та процеси, що керують симуляцією.	Звільнення ресурсів машин користувачів системи, зменшення апаратних вимог що до ЕОМ користувачів	Великі компанії зможуть зекономити на покупці GPU/CPU, та замінити їх більш дешевими аналогами
	Одночасна робота над складними сценами	Користувачі зможуть одночасно редагувати безліч сцен не боячись заблокувати роботу інших користувачів невдалими змінами (залежить від пайплайну проєкту у користувача)

Основним конкурентом проєкту даного стартапу вважається розробка **Nvidia Omniverse**, також, вірогідним супротивником є інші продукти, котрі просто надають можливість роботи з серверами – **Unreal Engine**, **Unity**. Присвоймо нашому проєкту ім’я **WMO engine**. Легенда умовних позначення для таблиці 4.2 наступні:

- **S** – strong side of the project, сильна сторона проєкту.
- **N** – neutral side of the project, нейтральна сторона проєкту.
- **W** – week side of the project, слабка сторона проєкту.

Таблиця 4.2 – Порівняння сильних сторін конкуруючих проєктів

Економічні та технічні характеристики розробленої системи	**Стартапи для порівняння**				**S**	**N**	**W**
	WMO engine	**Nvidia Omniverse**	**Unity Engine**	**Unreal Engine**			
Відкрита реалізація	Так	Ні	Так	Ні		+	
Умови використання	Безкоштовно для некомерційного	Безкоштовно при 2-х користувачах	Безкоштовно за певних умов	Безкоштовно за певних умов	+		
Внесення змін в реальному часі	Так	Так	Ні	Варіюється	+		
Постійна технічні підтримка	Обмежена	Відповідно до ліцензії	Обмежена	Обмежена		+	
Масштабування хмарних потужностей	Так	Так	Обмежена	Обмежена	+		
Вартість використання	100$ в місяць за клієнта	2000$ в рік та окрема	Варіюється	5% прибутку		+	
Особливі вимоги до системи	Ні	Наявність відеокарти Nvidia RTX	Ні	Варіюється		+	

Перелік даних в таблиці 4.2 дає уявлення про те, що головними властивостями проєкту є можливість отримати безкоштовний доступ до функціональності продукту, принаймні для персонального користування, або для тестування необхідності, можливість одночасного редагування сцени, а також можливість до масштабування хмарних потужностей.

До нейтральних сторін відноситься відкритість коду системи, наявність обмеженої технічної підтримки, вартість використання продукту для комерційних користувачів, а також відсутність особливих вимог до користувача.

4.2 Технологічний аудит проєкту

Для створення рушія та редактора було обрано мову програмування C++, а також набір інструментів, наведений в пункті 3.3. Також буде використаний засіб Apache Mesos управління кластером з ЕОМ, даний кластер буде розгортатися за вимоги користувача та додаткову оплату з його сторони. Технології та спосіб реалізації проєкту показано в таблиці 4.3.

Таблиця 4.3 – Технологічний аудит проєкту.

Ідея проєкту	Технології	Наявність	Доступність
Створення рушія та редактору	C++, Lua, HLSL	Наявні	Безкоштовні
Розгортання інфраструктури	Apache Mesos	Наявні	Apache License, Version 2.0

4.3 Аналіз ринкових можливостей

Основні ринкові можливості стартап-проєкту та визначення ринкових характеристик представляють собою головний етап. Розглянуті характеристики потрібно використати при релізі проєкту, або при дослідженні вірогідності успішного релізу продукту до вільного ринку.

Окрім того аналіз ринкових можливостей дозволяє уникнути широкого спекту загроз та ризиків. В першу чергу керівники проєкту повинні провести дослідження наведених нижче характеристик (опис також наведено в таблиці 4.4): попит до продукту, обсяг можливих продаж, динаміка зміни стану ринку.

Таблиця 4.4 – Характеристики ринку.

Показник	**Значення характеристики**
Число основних гравців на ринку	3
Сумарний обсяг продажу на ринку	Від 2400 до 240000 грн за користувача в місяць.
Динаміка зміниу стану ринку	Повільно зростає
Обмеження для входу на ринок	Висока технологічна база
Вимоги стандартизації	Стандартизації Microsoft, Sony

В таблиці 4.4. наведено показники попиту на ринку, обсягу на ринку та динаміка розвитку ринку проєкту. З таблиці можна побачити, що головних обмеженням для входу на ринок є необхідність великої технологічно бази, адже сам по собі проєкт є технологічно вимогливим, окрім того деякі послуги потребують додаткової сертифікації.

Вслід за проведення ринкового аналізу необхідно провести аналіз категорій основних клієнтів. Дослідження та поділ потенційних клієнтів на групи для розробленого проєкту наведено в таблиці 4.5.

Варто зауважити, що в таблиці 4.5 проводиться аналіз груп комерційних клієнтів, насправді кількість груп клієнтів може бути значно більшою, так як використання продукту можливе в безкоштовному режимі приватними особами, в майбутньому дані клієнту будуть приносити прибуток, але тоді вони перейдуть в одну з описаних категорій, тому вирізняти окремо їх не потрібно. На даний момент потрібно сконцентруватися на прибуткових елементах ринку.

Таблиця 4.5 – Дослідження клієнтів стартапу

Ринкова потреба	**Потенційні клієнти**	**Відмінності груп**	**Вимоги до товару**
Потреба в прискорені процесів розробки та налаштування віртуальних фізичних середовищ	Користувачі (підприємства), котрі потребують виконання фізичних симуляцій при змінюваних умовах, чи налаштування складної сцени декількома користувачами в режимі реального часу	Використання існуючих підходів до подібних задач, використання власних рішень	Зручна інтеграція, ціна, швидкість інтеграції, можливість підтримки та модифікації, одночасна модифікація
Необхідність гнучкого редагування віртуальних фізичних середовищ	Дослідники та інженери, яким необхідно дослідити ті, чи інші умови взаємодії об’єктів з динамічним чи статичним оточенням, розробники ігор чи працівники кінематографу	Наявність рішень від конкурентів. Відмінність у потребах та вимогах до візуалізації, чи редагування.	Зручні інструменти візуалізації й налаштувань, малий (незначний) час відгуку системи, можливість внесення модифікацій згідно поточних потреб

В таблиці 4.6 наведено аналіз/перелік загроз при релізі проєкту на цільовий ринок. Окрім ризиків, наведених в таблиці 4.6 потрібно розглянути позитивні можливості, їх дослідження та детальний опис наведено в таблиці 4.7, розміщеній нижче.

Таблиця 4.6 – Фактори виникнення загроз при виході до ринку.

Фактор	**Опис**	**Можливі реакції компанії**
Конкуренція	Неочікувана поява нового конкурента, або зміна стратегії старого	Покинути ринок, продавши продукт конкуренту, домовитися про поділ ринку, переглянути стратегію
Мала зацікавленість продуктом	Продукт є малоцікавим для користувачів	Покинути ринок, переглянути рекламну стратегію, переглянути бізнес стратегію
Зміна цільової групи користувачів	Зовнішня динаміка ринку змінила потреби користувачів	Покинути ринок, змінити концепцію проєкту, переглянути бізнес стратегію
Втрата цільового курсу	Продукт не відповідає початковому задуму	Повернутися до старого курсу, змінити цільову групу продукту, змінити рекламну стратегію

Таблиця 4.7 – Фактори виникнення можливостей при виході до ринку.

Фактор	**Опис**	**Можливі реакції компанії**
Поглинення конкурентної компанії	Можливість поглинути компанію конкурента	Утворення конгломерації, відмова від поглинання
Виявлення додаткової цільової групи клієнтів	В процесі розвитку проєкту їм зацікавилась нова/інша цільова група	Підтримувати розвиток продукту у даному напрямку

Продовження таблиця 4.7

Фактор	Опис	Можливі реакції компанії
Послаблення конкурентної компанії	Конкурент втратив позицію на ринку	Утворення конгломерації, ігнорування конкурента, допомога, заняття частини ринку конкурента

В таблицях 4.8 та 4.9 наведено аналіз конкуренції на ринку за різними підходами. Таблиця 4.8 показує класичний підхід до дослідження конкуренції на ринку, таблиці 4.9 показує дослідження ринку за М. Портером. Також в таблиці 4.10 за результатами дослідження в таблицях 4.8 та 4.9 проводиться дослідження конкурентоспроможності середовища.

Таблиця 4.8 – Дослідження конкуренції на ринку.

Особливості середовища	Опис характеристики	Вплив на курс діяльність компанії
Висока конкуренція на ринку	Високий рівень конкуренції на ринку, або високий рівень появи сильного конкурента на ринку	Повторний аналіз ринку, на котрий компанія бажає вийти
Наявність потужної монополії чи конгломерації на ринку	На рисунку знаходиться потужній монополіст, або потужна конгломерація компаній	Підтримка від інших конкурентів на ринку, звернення в антимонопольній комітет
Відсутність виразності серед конкурентів	Цільовий ринок сильно перенасичений, тому складно створити унікальний продукт	Знаходження унікальної складової, або створення сильної пір компанії.

Таблиця 4.9 – Аналізу конкуренції на ринку за М. Портером.

	Прямі конкуренти	**Потенційні конкуренти**	**Постачальники**	**Клієнти**	**Замінники**
Складові	Nvidia Omniverse, Unity, Unreal Engine	-	-	Компанії з розробки ігор, кіно або наукові установи	Неочікувані конкуренти
Висновки	Кількість конкурентів незначна, проте вони мають сильні позиції	-	-	Частина клієнтів користується	Передбачити замінники подальших конкурентів неможливо

Таблиця 4.10 – Обґрунтування факторів конкурентоспроможності

Назва фактору	**Обґрунтування фактору**
Зручний та інтуїтивний інтерфейс редактора	Користувачі будуть взаємодіяти з рушієм виключно завдяки інтерфейсу, необхідно зробити його якомога зручнішим, особливо в порівнянні з конкурентами, а також використати останні практики, так як UI/UX вже давно встановлений через Unreal/Unity
Можливість масштабування	Бекенд системи повинен легко масштабуватися на різну кількість користувачів
Можливість надання допомоги при власному розгортанні інфраструктури	За бажання користувачами розгорнути власну інфраструктуру ми повинні надати максимальну технічну допомогу для вирішення даного питання

Опираючись на попередню інформацію з таблиці 4.6 – 4.10 переходимо до дослідження сильних сторін продукту. Дане дослідження проводимо з використанням числових оцінок, саме дослідження проведено в таблиці 4.11, наведеній нижче.

Таблиця 4.11 – Порівняльне дослідження сторін проєкту

Фактор конкурентно-спроможності	**Оцінка**	**Оцінка рейтингу конкурентів відносно розробленого продукту та стартапу в цілому**						
		-3	-2	-1	0	+1	+2	+3
Зручний та інтуїтивний інтерфейс редактора	16						+	
Витрати на інтеграцію у вже існуючі системи	6			+				
Можливість масштабування системи	20							+
Швидкодія системи	8					+		
Ефективність підтримки	10				+			
Можливість паралельного редагування сцен	20							+

Далі необхідно провести так званий SWOT (Strength, Weakness, Opportunities, Threats) аналіз. Даний вид аналізу досить абстрактно описує сильні та слабкі проєкту.

Результат даного виду аналізу показано на рисунку 4.12. В подальшому результати даного виду аналізу буде використовуватися для проведення інших досліджень перспектив продукту на конкурентному ринку. Саме завдяки SWOT аналізу в подальшому створюються альтернативні моделі виведення продукту на ринок, котрі наведені в таблиці 4.13.

Таблиця 4.12 – SWOT аналіз проєкту

Сильні сторони: – Інтуїтивний інтерфейс – Можливість масштабування під різні кількості користувачів – Можливість редагування сцен в реальному часі	**Слабкі сторони:** – Загальна швидкодія система деякий час може бути низькою – Інтеграція системи у вже існуючі процеси може бути складною
Можливості: – Зацікавити широкий спектр клієнтів у середньому та великому бізнесу – Створити принципіально новий продукт	**Загрози:** – Висока конкуренція – Неможливість реалізувати проект через неочікувані технічні складнощі

Таблиця 4.13 – Альтернативи впровадження проекту на ринок

Альтернативний спосіб виведення	Успішність	Витрати часу
Надання безкоштовної підтримки	40%	6 місяців
Розширення можливостей безкоштовної ліцензії	80%	-
Інформування через конференції	30%	12 місяців
Розповсюдження завдяки відкритому коду, та підтримка проекту через пожертвування	40%	-

4.4 Розробка ринкової стратегії проєкт

Ключовим етапом розробки ринкової стратегії є дослідження груп користувачів, котрі можуть бути зацікавлені, або ж не зацікавлені у продукту. Результати в таблиці 4.14. Знаючи параметри групи проводимо створення базової стратегії та стратегії конкурентної поведінки, вони показані в таблиці 4.15 та 4.16

Таблиця 4.14 – Цільові групи споживачів

Опис	Готовність споживачів	Попит	Конкуренція	Складність входу
Розробники медіаконтенту	Системи зі схожим інтерфейсом та можливостями широко розповсюджені, тому з точки зору користувача нічого не змінюється. Проблемою є складність інтеграції та відбиття клієнтів від вже звичних рішень.	Середній	Надвисока	Надвисока

Таблиця 4.15 – Стратегія розвитку

Обрана альтернатива розвитку	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції	Базова стратегія розвитку
Розширення можливостей безкоштовної ліцензії	Підтримка середньо та малаго бізнесу, котра не може позволити собі конкурентні продукти	Ключовою позицією є продовження розвитку багатокористувацької взаємодії	Розширення команди інтеграції, розширення технічної команди та надання нових можливостей

Таблиця 4.16 – Стратегія конкурентної поведінки

Першість продукту на ринку	**Позиція що до переманювання клієнтів**	**Позиція що до копіювання ознак конкурентів**	**Позиція щодо конкурентної поведінки**
Ні	Так, повністю, так як ринок повністю занятий.	Частково, основною характеристикою є UI/UX. Базова функціональність повинна бути однаковою.	Ведення адаптивної цінової політики, переманювання клієнтів

Таблиці 4.15 та 4.16 описували стратегію відносно конкурентів та клієнтів, дослідження стратегії позиціонування має в своїй меті дослідження того, як проект позиціонує себе на ринку. Дослідження наведено в таблиці 4.17.

Таблиця 4.17 – Стратегія позиціонування продукту

Вимоги цільової аудиторії до товару	**Базова стратегія розвитку**	**Ключові конкурентоспроможні позиція власного стартапу**	**Вибір асоціацій, котрі повинні сформуватися**
Ефективність, наявність загальноприйнятні функціональності та UI/UX	Покращення можливості паралельної роботи за рахунок отримання відгуків та збільшення технічної компетенції	Надання можливості паралельної розробки сцени, доступні ліцензія	Масштабованість, інтуїтивність, ефективність, простота, доступність

4.5 Розробка маркетингової програми

Слідкуючи за асоціаціями з таблиці 4.17 необхідно розробити план концепцій, котрих треба притримуватися при маркетинговому позиціонуванні продукту. Список концепцій показано в таблиці 4.18.

Таблиця 4.18 – Визначення ключових переваг концепцій

Потреба	Запропонована вигода	Ключові переваги
Можливість паралельного редагування сцен	Система дає можливість одночасно працювати декільком користувачам над сценою та моделювати фізики в ній	Лише основний конкурент має дану можливість
Доступність	Доступний план що до доступності системи	Основний конкурент з тими ж технічними можливостями має високу вартість
Технічна підтримка при розгортанні власної інфраструктури	Допомога при розгортанні інфраструктури	Основний конкурент з тими ж технічними можливостями на надає даних послуг
Зрозумілий та звичний інтерфейс	Надається звичний та зрозумілий інтерфейси, схожий за UI/UX до звичних	Основний конкурент має власну концепцію UX, зовсім не схожу до звичних ідей інших конкурентів та прийнятих індустріальних стандартів.

Далі описується трирівнева модель продукту. Дана модель відповідає за опис: ідеї продукту, його фізичної складової та процесу надання товару користувачам. Таблиця з описом моделі на рисунку 4.19.

Таблиця 4.19 – Трирівнева модель товару продукту з описом

<table>
<tr><th>Рівні товару</th><th colspan="2">Сутність та складові</th></tr>
<tr><td rowspan="6">I. Товар за задумом

II. Товар у реальному виконанні</td><td colspan="2">Система з рушія та редактора, що дозволяє за доступну ціну редагувати паралельно великі сцени з повним фізичним моделюванням</td></tr>
<tr><td>Властивості/характеристики</td><td>Значення характеристики</td></tr>
<tr><td>1. Максимальна доступність
2. Паралельне редагування є ефективним
3. Швидкодія є задовільною</td><td>Сервер повинен мати 16 Gb RAM, 8Gb VRAM. Клієнт повинен мати 4Gb RAM та 2Gb VRAM</td></tr>
<tr><td colspan="2">Якість: способи контролю якості описані в пункту 3.3, а слабкі точки систему в додатку 3. Контроль буде відбуватися за даними піраміди тестування.</td></tr>
<tr><td colspan="2">Пакування: товар розповсюджується у вигляді бінарних пакетів через систему Vcpkg, або Connan</td></tr>
<tr><td colspan="2">Марка: назва розробника, назва конгломерату, торгова марка</td></tr>
<tr><td rowspan="2">III. Товар із підкріпленням</td><td colspan="2">Додатково: технічна підтримка з боку компанії-розробника</td></tr>
<tr><td colspan="2">Захист потенційного товару
– Ліцензія
– Товарний знак
– Репутація</td></tr>
</table>

Наступним етапом є вирішення цінової політики і меж даних цінової політики, як і наведення специфіки закупівлі продукту клієнтами, його маркетинг в плані закупівлі та отримання прибутку. Результати в таблицях 4.20 – 4.22.

Таблиця 4.20 – Визначення фінальних цін

Значення цін на продукт	**Рівень цін на продукт конкурентів**	**Рівень доходів цільової групи**	**Верхня та нижня межі на послуги.**
Залежно від пакету. 25000 – 40000 за користувача в компанії	67000 гривень	Високий. Від 350000 гривень.	Безкоштовно. 40000 гривень за користувача в компанії

Таблиця 4.21 – Визначення специфіки закупівлі товару

Специфіка закупівельної поведінки	**Функції постачальника**	**Глибина каналу збуту**	**Система збуту**
Надання послуги покупки системи	Викласти бінарний пакет в системі розповсюдження пакетів	2-ий рівень. Напряму з репозиторію бінарних пакетів компанії. Проте не на сайті компанії	Конференції, репутація, сайт компанії
Надання послуг з інтеграції в інфраструктуру	Відправити команду спеціалістів, надати канал комунікації	1-ий рівень. Напряму через ресурси компанії.	Конференції, репутація, сайт компанії

Продовження таблиці 4.21

Специфіка закупівельної поведінки	Функції постачальника	Глибина каналу збуту	Система збуту
Прийняття пропозицій щодо покращення продукту	Надати форму зв’язку з командою	1-ий рівень. Напряму через ресурси компанії	Сайт компанії
Прийняття інвестицій	Надати форму інвестицій	1-ий рівень та 2-ий. Напряму через ресурси компанії, або інші сторінки в соц. мережах.	Сайт компанії, конференції

Таблиця 4.22 – Концепція маркетингових комунікацій

Специфіка поведінки цільових клієнтів	Канали комунікацій	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення
Маркетинг через статті на виділених ресурсах та порівняння	Виділені канали зв’язку, сайти компанії	Ефективність, нова функціональність, надійність, підтримка	Познайомити з товаром, зацікавити
Маркетинг через конференції та інших клієнтів, активних членів ком’юніті	Конференції, мітинги, туторіали	Ефективність, нова функціональність, надійність, підтримка	Познайомити з товаром, зацікавити

4.6 Висновки до розділу 4

За результатами даного дослідження з проведення ефективності реалізації стартап проекту, котрий включає в себе елементи розробки, викладені в пунктах 1 – 3, було отримано економічно-технічний звіт щодо даного стартапу. Весь даний розділ присвячений створенню плану розробки успішного продукту, а також дослідженню альтернативних способів, за отриманим економічно-технічним дослідженням можна постановити наступне:

- Надано чітке ідейне та формальне обґрунтування ідеї виникнення стартапу, мета його створення, актуальність задачі котру вирішуватиме створений продукт, а також визначені ключові характеристики, упор на розвиток яких допоможе вивести даний продукт на ринок з надзвичайно високим рівнем конкуренції.
- Надано перелік стратегій, котрі можуть бути використані при виході продукту на відповідний цільовий ринок з 3 вже існуючими сильними конкурентами та можливістю до виникнення нових. Надано декілька альтернативних стратегій.
- Надано конкретну маркетингову стратегії, спрямовану на конкретні цільові групи, зацікавлені в основних ключових ідеях продукту.
- Завдяки проведеному технічному аудиту було виведено базові технічні характеристики, списки апаратного, програмного та інструментального забезпечення, необхідного для успішного створення, функціонування та розвитку продукту.
- Досліджено нагальну ситуацію на ринку та отримано список слабких та сильних сторін наявних конкурентів.
- Отримано готовий та фінальний план розвитку проєкту. Основною даного проєкта буде вже готовий та завершений продукт. Було проведено аналіз ринку, з чого було отримано, що в найгіршому випадку вихід на ринок

триватиме близько 12-ти місяців. Час виходу на ринок залежить від підходу до просунення продуктів, в кращому випадку час виходу на ринок становитиме 2 місяці, але шанс цього досить низький, а ймовірність бути поглинутими конкурентами – досить висока.

ВИСНОВКИ

В процесі проведення досліджень в рамках магістерської дисертації було проведено аналіз предметної області даної роботи, які є підходи та алгоритми проведення моделювання фізичних явищ (а конкретно колізій твердих тіл) в тривимірному просторі з використання ЕОМ і подальше їх застосування разом з хмарними технологіям та розподіленими обчисленнями. Дана робота була виконання з такими цілями:

- Детально ознайомитися з об’єктом, а також предметом дослідження.
- Отримати практичні результати застосування вже існуючих технологій (в процесі адаптувати їх до стану прототипу) разом з хмарними та розподіленими обчисленнями. Дати висновок що до того, чим дані підходи відрізняються від загальноприйнятних.
- На основі отриманих результатів зробити висновок що до доцільності проведення подальшого дослідження. В результаті ствердувальної відповіді – описати архітектуру рушія, котрий би був конкретно створений для використання разом з хмарними технологіями, провести аналіз даної архітектури.

Враховуючи цілі, наведені вище, а також загальну структуру дослідження та даної роботи (чотири розділи, присвячені освітленню проблематики з різних точок зору), загальний висновок наступну та наведений нижче.

В першому розділі було проведено ґрунтовний аналіз теоретичного базису (як з точки зору математики, фізики, так і з точки зору програмного інженера), а також надано опис самої проблеми проведення моделювання твердих тіл. Було формально надано та виведено весь список базових формул математики, та приведено весь список теорій та фізичних законів, необхідних для розуміння та вирішення проблематики. Окрім математичної та фізичної точки зору на проблему

велику увагу було приділено розгляду проблеми з точки зору комп’ютерних наук, в даній площині було проведено аналіз таких явищ як:

- Тверде тіло та частинка, було надано визначення в контексті проведення фізичних симуляцій на ЕОМ.
- Обмежуючі об’єми (**BV**), прискорюючі структури даних (**ADV**), алгоритм розбиття простору (**SPA**), ієрархії обмежуючих коробок (**BVH**), бінарне розбиття простору (**BSP**)
- Основні метрики, котрі використовуються для виміру ефективності роботи даних алгоритмів: довжина фрейму (**delta time**) та кадри за секунду (**FPS**).
- Загальна схему принципу пошуку колізій: крок, широка фаза, вузька фаза, пошук неперервних колізій (**CCD**).
- Алгоритми пошуку колізій в широкій та вузькій фазах: **SAT**, **GJK**, **EPA**.
- Розглянуто систему обмежень, поняття С-функції, а також ланцюга С-функцій, розглянуто їх взаємодія. Окрему увагу було приділено С-функції непроникнення та її використанню.

В другому розділі роботи було проведене дослідження самих сучасних технологій для проведення фізичних симуляцій: надано короткий опис, порівняння у вигляді таблиці, а також наведені конкретні критерії, за якими було обрано найбільш доцільний для подальшого дослідження. В подальшому обрану технологію (Bullet), було модифіковано для отримання кращої підтримки розподілених обчислень з використанням OpenCL API в якості систему зв'язку з кластером. Також були внесені певні модифікації в програмний код для отримання більшої стабільності при роботі з мережевим кодом (дослідження в хмарах). Детальний аналіз отриманого рішення було проведено в кінці третього розділу.

В третьому розділі було проведено практичну роботу щодо створення архітектури рушія, котрий буде розрахований саме на використання з хмарними технологіями, саме навколо цієї ідеї базувалася вся описана в розділі архітектура.

Було наведено конкретні задачі, котру повинна виконувати система за заданою архітектурою, рівно як і наведені конкретні критерії оцінки ефективності. Було надано проектну та технічну документацію, перелік основних віх. Архітектуру проаналізовано на предмет критичних точок в ній, надано список конкретних елементів піраміди тестування, котрий допоможе нівелювати наявність даних точок. В кінці розділу було надано набір інструментів для реалізації даної архітектури, а також проведено ряд досліджень з використанням модифікації Bullet. Використовуючи модифікований код та збудовану сцену з 3000 активними об’єктами, котрі генерують в середньому 18000 контактів за секунду було отримано практичні чисельні результати у вигляді графіків, на основі яких було отримані висновки що до доцільності використання технологій. З отриманими результатами було зроблено окремий детальний висновок, розміщений в кінці розділу 3. Узагальнюючі тезиси наступний:

- Використання розподілених технологій є доцільним при симуляції великих за об’ємом сцен, котрі виходять за рамки практично використовуємих та виникають лише в процесі наукоємних обчислень. Окрім того, використання кластеру потребує досить розумного ставлення до кількості вузлів, так як їх кількість напряму впливає на швидкодію системи. При малій кількості потужних вузлів витрати на передачу даних занадто між вузлами занадто великі та нівелюють всю потужність кластера як такого, тому у випадку фізичних симуляцій потрібно рухатися в сторону більшої кількості вузлів з меншою потужністю кожного окремого вузла.
- Використання хмарних технологій в той же час показало себе доцільніше. Отримані практичні результати свідчать про те що навіть з використанням не пристосованих до хмар технологій можна отримати працюючу стабільну систему. Було отримано, що деградація системи з затримкою в 20 – 40 мілісекунд становить приблизно 25%, проте на відміну від розподілених обчислень середній FPS в кожній симуляції був стабільний і мав стабільніше

значення відхилення. Також слід зауважити, що середній в жодному експерименті не падав нижче 29-30 FPS.

Узявши до увагу, що на практиці фізичні симуляції виконуються при стабільних 30 FPS було прийняте рішення про доцільність подальшого розгляду підходу з використанням віддалених, хмарних обчислень.

В четвертому розділі було проведено економічно-технічний аналіз продукту, котрий розробляється за архітектурою, представленою в розділі 3. В результаті роботи над розділом було надано конкретний набір стратегій, котрий допоможе продукту вийти на ринок з надзвичайно високим рівнем конкуренції. Згідно з даним планом продукт має помірно-оптимістичний шанс на закріплення на ринку. Розроблено план подальшого розвитку системи після досягнення нею можливості вирішення ключових задач.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Avril Q. A Broad Phase Collision Detection Algorithm Adapted to Multi-cores Architectures [Електронний ресурс] / Q. Avril, V. Gouranton. – 2010. – URL: https://www.researchgate.net/publication/43174906_A_Broad_Phase_Collision_Detection_Algorithm_Adapted_to_Multi-cores_Architectures. (дата звернення 02.01.2022)
2. I-COLLIDE: An Interactive and Exact Collision Detection System for Large-Scale Environments [Електронний ресурс] / J. D.Cohen, L. C. Ming, M. Dinesh, M. K. Ponamgi. – 1995. – URL: https://www.cs.jhu.edu/~cohen/Publications/icollide.pdf. (дата звернення 04.01.2022)
3. Ericson C. Real-Time Collision Detection / Christer Ericson., 2005. – 594 с. – (The Morgan Kaufmann Series in Interactive 3D Technology).
4. van den Bergen G. Game Physics Pearls / G. van den Bergen, D. Gregorius., 2010. – 352 с.
5. van Den Bergen G. A Fast and Robust GJK Implementation for Collision Detection of Convex Objects [Електронний ресурс] / Gino van Den Bergen. – 1999. – URL: http://www.dtecta.com/papers/jgt98convex.pdf. (дата звернення 07.01.2022)
6. van Den Bergen G. A Proximity Queries and Penetration Depth Computation on 3D Game Objects [Електронний ресурс] / Gino van Den Bergen. – 2001. – URL: https://graphics.stanford.edu/courses/cs468-01-fall/Papers/van-den-bergen.pdf. (дата звернення 13.01.2022)
7. Mirtich B. V. Impulse Based Dynamic Simulation of Rigid Body Systems [Електронний ресурс] / Brian Vincent Mirtich. – 1996. – URL: http://www.kuffner.org/james/software/dynamics/mirtich/mirtichThesis.pdf. (дата звернення 20.01.2022)

8. Coumans E. Continuous Collision Detection and Physics [Електронний ресурс] / Erwin Coumans. – 2005. – URL: https://www.gamedevs.org/uploads/continuous-collision-detection-and-physics.pdf. (дата звернення 23.01.2022)
9. Chappuis D. Constraints Derivation for Rigid Body Simulation in 3D [Електронний ресурс] / Daniel Chappuis. – 2013. – URL: https://danielchappuis.ch/download/ConstraintsDerivationRigidBody3D.pdf. (дата звернення 01.02.2022)
10. Effective Constrained Dynamic Simulation Using Implicit Constraint Enforcement [Електронний ресурс] / M.Hong, C. Min-Hyung, J. Sunhwa, S. w j Welch. – 2005. – URL: https://www.researchgate.net/publication/221067910_Effective_Constrained_Dynamic_Simulation_Using_Implicit_Constraint_Enforcement. (дата звернення 11.02.2022)
11. Tamis M. Constraint based physics solver [Електронний ресурс] / M. Tamis, M. Giuseppe. – 2015. – URL: http://mft-spirit.nl/files/MTamis_ConstraintBasedPhysicsSolver.pdf. (дата звернення 17.02.2022)
12. Witkin A. Physically Based Modeling: Principles and Practice Constrained Dynamics [Електронний ресурс] / Andrew Witkin. – 1997. – URL: https://www.cs.cmu.edu/~baraff/sigcourse/notesf.pdf. (дата звернення 23.02.2022)
13. Hecker C. The Mixed Linear Complementarity Problem [Електронний ресурс] / Chris Hecker. – 2007. – URL: https://chrishecker.com/The_Mixed_Linear_Complementarity_Problem. (дата звернення 24.05.2022)
14. PhysX. Офіційний сайт Nvidia PhysX SDK: веб-сайт. URL: https://developer.nvidia.com/physx-sdk (дата звернення 01.06.2022)
15. Havoc. Офіційний сайт Havoc Physics: веб-сайт. URL: https://www.havok.com/havok-physics/ (дата звернення 01.06.2022)

16. Bullet Real-Time Physics Simulation. Офіційний сайт Bullet: веб-сайт. URL: https://pybullet.org/wordpress/ (дата звернення 01.06.2022)
17. Box2d. Офіційний сайт Box2d: веб-сайт. URL: https://box2d.org/documentation/ (дата звернення 01.06.2022)
18. Simulation Tools for Model-Based Robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX [Електронний ресурс] / T. Erz, T. Yuvala, E. Todorov. – URL: https://homes.cs.washington.edu/~todorov/papers/ErezICRA15.pdf (дата звернення 01.06.2022)
19. Nvidia Omniverse. Офіційний сайт Nvidia Omniverse: веб-сайт. URL: https://www.nvidia.com/ru-ru/omniverse/ (дата звернення 02.06.2022)
20. Universal Scene Description. Офіційний сайт Pixar: веб-сайт. URL: https://graphics.pixar.com/usd/release/intro.html (дата-звернення 02.06.2022)
21. The MOSIX Virtual OpenCL (VCL) Cluster Platform [Електронний ресурс] / B. Amnon, S. Amnon, – URL: https://developer.amd.com/wordpress/media/2013/06/2913_2_final.pdf (дата звернення 03.06.2022)

ДОДАТОК 1

Приклад частини коду для симуляції поведінки частинок в двовимірному просторі з впливом гравітації та випадково заданим початковим вектором швидкості.

```
#pragma once

#include <iostream>
#include <cinttypes>
#include <vector>

#include "utils.hpp"
#include "world.hpp"
#include "vector.hpp"

constexpr uint8_t max_particle_count       = std::numeric_limits<uint8_t>::max();
constexpr uint8_t particle_count           = 1;
constexpr int min_position                 = 0;
constexpr int max_position                 = 50;

struct                                     particle
{
	vector                             m_position     { 0.f, 0.f };
	vector                             m_velocity     { 0.f, 0.f };
	float                              m_mass         { 0.f };
};

vector compute_particle_acceleration(particle &target)
{
	auto compute_gravity_force         = [] (const particle &target) -> vector { return
vector::multiply(constant::gravity_direction, (target.m_mass * constant::gravity_acceleration)); };
	auto compute_acceleration          = [] (const particle &target, const vector force) ->
vector { return vector::divide(force, target.m_mass); };
	return                             compute_acceleration(target,
compute_gravity_force(target));
}

void initialize_particles(std::vector<particle> &particles)
{
	for (particle &target : particles)
	{
		target.m_position          = { static_cast<float>(random_number(min_position,
max_position)), static_cast<float>(random_number(min_position, max_position)) };
		target.m_velocity          = { 0.f , 0.f };
		target.m_mass              = 1.f;
	}
}

void draw_particles(std::vector<particle> &particles)
{
	for (std::size_t i = 0; i < particles.size(); ++i)
	{
		particle &target           = particles[i];
		std::printf                ("Particle[%llu] : (%.2f, %.2f)\n", i,
target.m_position.m_x, target.m_position.m_y);
	}
}

void simulate_particles()
```

```
{
	std::vector<particle> particles;
	particles.resize                    (particle_count);

	initialize_particles                (particles);

	constexpr float delta_time          = 1;
	constexpr float max_time            = 10;
	float current_time                  = 0;

	for (float current_time = 0; current_time < max_time; current_time += delta_time)
	{
		for (particle &target : particles)
		{
			const vector acceleration    = compute_particle_acceleration(target);
			const vector velotics_delta  = vector::multiply(acceleration, delta_time);
			target.m_velocity.add        (velotics_delta);

			const vector position_delta  = vector::multiply(target.m_velocity,
delta_time);
			target.m_position.add(position_delta);
		}

		draw_particles                      (particles);
		sleep                               (static_cast<int>(delta_time));
	}
}
```

ДОДАТОК 2

Приклад частини коду для симуляції поведінки твердого тіла в двовимірному просторі з впливом гравітації та випадково заданим початковим вектором швидкості.

```
#pragma once

#include <iostream>
#include <cinttypes>
#include <vector>

#include "utils.hpp"
#include "world.hpp"
#include "vector.hpp"

constexpr uint8_t max_body_count = std::numeric_limits<uint8_t>::max();
constexpr uint8_t body_count = 1;
constexpr int min_body_position     = 0;
constexpr int max_body_position     = 50;

namespace                                           shape
{

struct                                              box
{
public:
       float                                        m_width;
       float                                        m_height;
       float                                        m_mass;
       float                                        m_inertia_moment;

public:
       inline                                       box                  ();
       inline                                       box                  (const float width, const
float height, const float mass);
       inline float                          inertia       () const                    { return
(m_mass * (m_width * m_width + m_height * m_height)) / 12; }
};

box::box                                            () : m_width(0.0f), m_height(0.0f), m_mass(0.0f),
m_inertia_moment(0.0f)
{
       m_inertia_moment                     = inertia();
}

box::box                                            (const float width, const float height, const
float mass) : m_width(width), m_height(height), m_mass(mass), m_inertia_moment(0.0f)
{
       m_inertia_moment                     = inertia();
}

}

template <typename shape_t>
class                                               rigid_body
{
```

```
public:
	inline 					rigid_body	();
	inline 					rigid_body	(const shape_t shape, const vector
&position, const float angle);

public:
	shape_t 				m_shape;

	vector 					m_force;
	vector 					m_position;
	vector 					m_linear_velocity;

	float 					m_torque;
	float 					m_angle;
	float 					m_angular_velocity;
};

template <typename shape_t>
rigid_body<shape_t>::rigid_body() :
							m_shape(shape_t()),
							m_force(constant::zero_vector),
m_linear_velocity(constant::zero_vector), m_position(constant::zero_vector),
							m_torque(0.0f), m_angle(0.0f),
m_angular_velocity(0.0f) { }

template <typename shape_t>
rigid_body<shape_t>::rigid_body(const shape_t shape, const vector &position, const float angle) :
							m_shape(shape),
							m_force(constant::zero_vector),
m_linear_velocity(constant::zero_vector), m_position(position),
							m_torque(0.0f), m_angle(angle),
m_angular_velocity(0.0f) { }

template <typename shape_t>
void initialize_rigid_bodies(std::vector<rigid_body<shape_t>> &bodies)
{
	auto initialize_rigid_body	= [] (rigid_body<shape_t> &body) -> void
	{
		const shape_t shape		{ static_cast<float>(random_number(1, 2)),
static_cast<float>(random_number(1, 2)), 10.0f };
		const vector position { static_cast<float>(random_number(min_body_position,
max_body_position)), static_cast<float>(random_number(min_body_position, max_body_position)) };
		const float angle		{ (random_number(0, 360) / 360.0f) * constant::pi_value *
2 };
		body 					= { shape, position, angle };
	};

	for (rigid_body<shape_t> &body : bodies)
	{
		initialize_rigid_body (body);
	}
}

template <typename shape_t>
void compute_force			(rigid_body<shape_t> &body)
{
	const vector force		= { 0.0f, 100.0f };
	body.m_force			= force;
}

template <typename shape_t>
void compute_torque			(rigid_body<shape_t> &body)
{
```

```
	const vector arm_vector		= { body.m_shape.m_width / 2.0f, body.m_shape.m_height / 2.0f };
	body.m_torque			= arm_vector.m_x * body.m_force.m_y - body.m_force.m_x * arm_vector.m_y;
}

void draw_bodies(std::vector<rigid_body<shape::box>> &bodies)
{
	for (std::size_t i = 0; i < bodies.size(); ++i)
	{
		const rigid_body<shape::box> &body = bodies[i];
		std::printf			("body[%lld] p = (%.2f, %.2f), a = %.2f\n", i, body.m_position.m_x, body.m_position.m_y, body.m_angle);
	}
}

void simulate_rigid_bodies()
{
	std::vector<rigid_body<shape::box>> bodies;
	bodies.resize			(body_count);

	initialize_rigid_bodies		(bodies);

	constexpr float delta_time	= 1;
	constexpr float max_time	= 10;
	float current_time		= 0;

	for (float current_time = 0; current_time < max_time; current_time += delta_time)
	{
		for (rigid_body<shape::box> &body : bodies)
		{
			compute_force		(body);
			compute_torque		(body);
			vector linear_acceleration = vector::divide(body.m_force, body.m_shape.m_mass);
			body.m_linear_velocity.add(linear_acceleration.multiply(delta_time));
			body.m_position.add	(vector::multiply(body.m_linear_velocity, delta_time));

			const float angular_acceleration = body.m_torque / body.m_shape.m_inertia_moment;
			body.m_angular_velocity = body.m_angular_velocity + angular_acceleration * delta_time;
			body.m_angle		= body.m_angle + body.m_angular_velocity * delta_time;
		}

		draw_bodies			(bodies);
		sleep				(static_cast<int>(delta_time));
	}
}
```

ДОДАТОК З

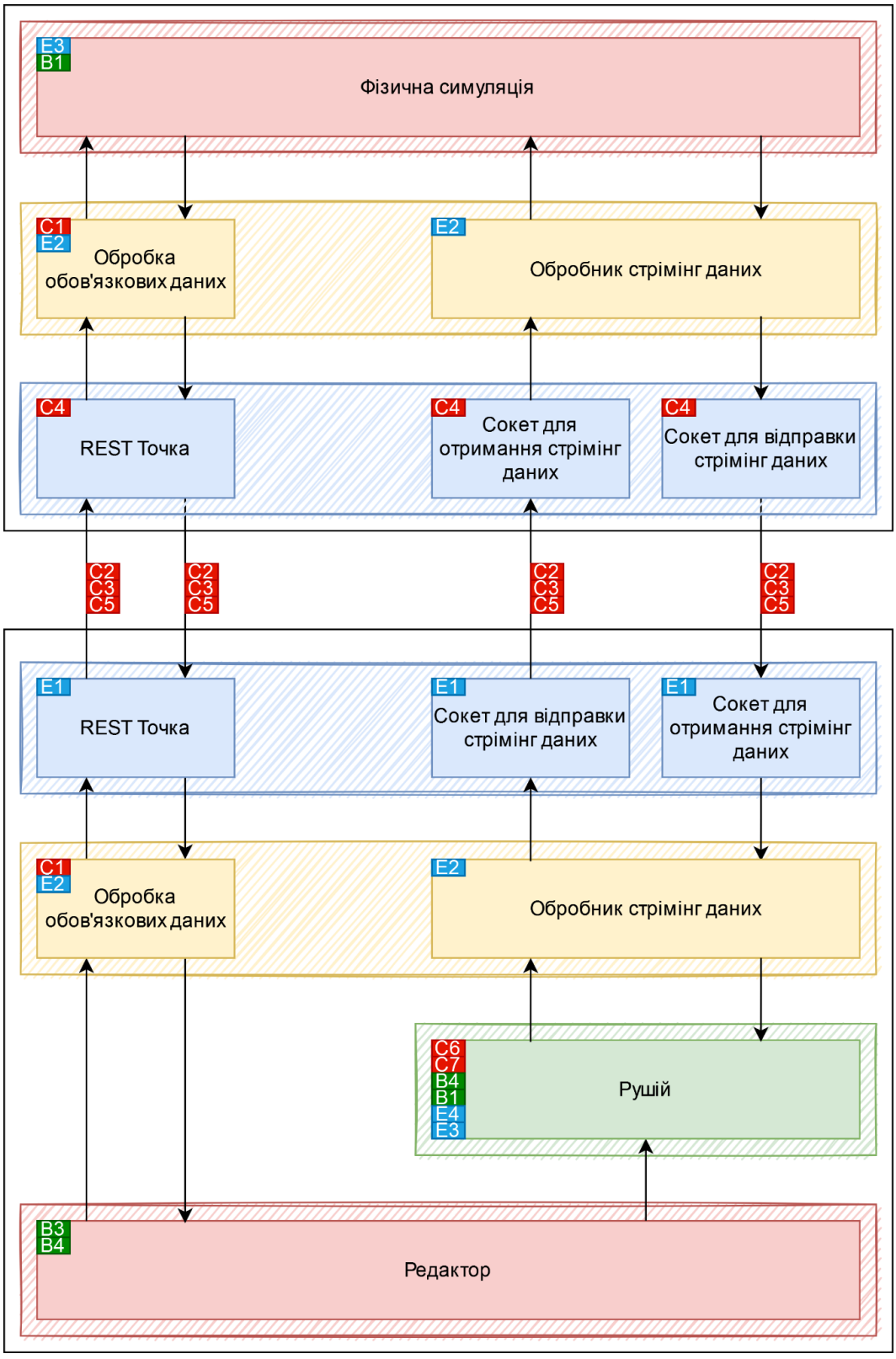

Рисунок ДЗ.1 Загальна HLD з помітками вразливих місць та загроз, без легенди

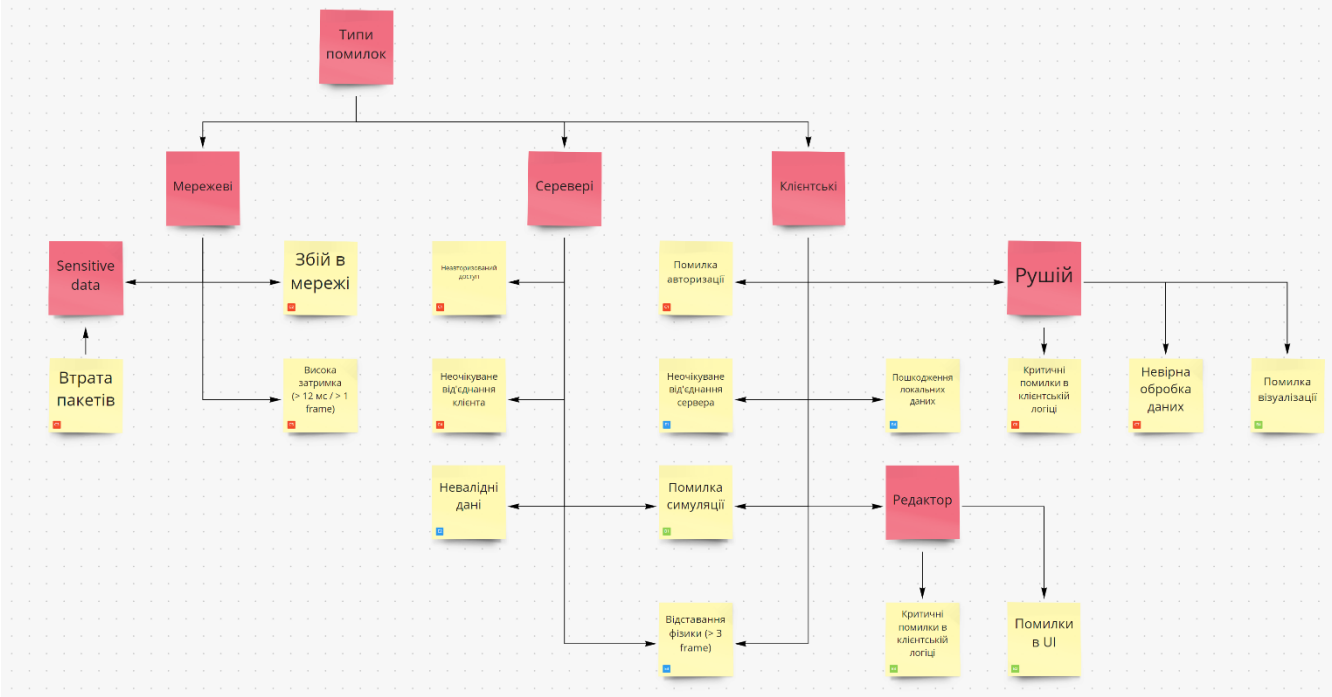

Рисунок Д3.2 Легенда та переліз загроз до рисунку Д3.1