Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра прикладної математики
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Оптимізація пошуку сусідів
у методі гідродинаміки згладжених частинок
(тема)

Виконав:
студент _2_ курсу, групи _ПМм-18-1
Стрекозов А.Д.
(прізвище, ініціали)
Спеціальність
113 Прикладна математика
(код і повна назва спеціальності)
Тип програми _освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма
Прикладна математика
(повна назва освітньої програми)
Керівник доц. Артюх А.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ПМ ____________ Тевяшев А.Д.
(підпис) (прізвище, ініціали)

2019 р.

Харківський національний університет радіоелектроніки

Факультет інформаційно-аналітичних технологій та менеджменту
Кафедра прикладної математики
Рівень вищої освіти другий (магістерський)
Спеціальність 113 Прикладна математика
(код і повна назва)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма Прикладна математика
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри ПМ ____________
(підпис)
“____” ______________ 2019 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові Стрекозову Антону Дмитровичу
(прізвище, ім’я, по батькові)

1. Тема роботи Оптимізація пошуку сусідів у методі гідродинаміки згладжених частинок

затверджена наказом по університету від 31 жовтня 2019 р. № 1600 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 9 грудня 2019 р.

3. Вихідні дані до роботи математична модель течії в’язкої рідини

4. Перелік питань, що потрібно опрацювати в роботі
   1. Системний аналіз проблеми оптимізації алгоритму пошуку сусідів у методі гідродинаміки згладжених частинок
   2. Вибір і обґрунтування методу розв’язання
   3. Програмна реалізація
   4. Результати обчислювального експерименту
   5. Аналіз можливих застосувань

5. Перелік графічного матеріалу із зазначенням кресленників, схем, плакатів, комп’ютерних ілюстрацій

1. Актуальність теми роботи
2. Постановка задачі
3. Системний аналіз проблеми
4. Метод чисельного аналізу
5. Результати обчислювального експерименту

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Підбір та вивчення технічної літератури за темою роботи	вересень 2019 р.	виконано
2	Вибір та обґрунтування методу	жовтень – листопад 2019 р.	виконано
3	Розробка алгоритму і програми	листопад – грудень 2019 р.	виконано
4	Проведення аналітичних досліджень та розрахунків	листопад – грудень 2019 р.	виконано
5	Робота над текстом пояснювальної записки	грудень 2019 р.	виконано
6	Представлення роботи на рецензію в ЕК	грудень 2019 р.	виконано

Дата видачі завдання _2_ вересня 2019 р.

Студент ________________________________

(підпис)

Керівник роботи ________________ доц. Артюх А.В.

(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 90 с., 5 табл., 54 рис., 2 дод., 18 джерел.

АЛГОРИТМ НАЙБЛИЖЧИХ СУСІДІВ, БОКС, АЛГОРИТМ ПРОСТОРОВОГО ХЕШУВАННЯ, ОПТИМІЗАЦІЯ, ГІДРОДИНАМІКА ЗГЛАДЖЕНИХ ЧАСТИНОК, МОДЕЛЮВАННЯ.

Об’єкт дослідження – оптимізація алгоритму пошуку найближчих сусідів для гідродинаміки згладжених частинок.

Мета роботи – дослідити існуючі алгоритми пошуку сусідів і їх оптимізації, обрати з них найбільш ефективний для обчислень, провести обчислювальний експеримент на моделі в’язкої рідини.

Методи дослідження – боксовий алгоритм та алгоритм просторового хешування, програмна оптимізація.

Атестаційна робота присвячена дослідженню оптимізації існуючих алгоритмів та їх програмній реалізації сучасними засобами. Були розглянуті найбільш поширені алгоритми пошуку сусідів, методи оптимізації програмного коду сучасними засобами. Обчислювальний експеримент було проведено для 1000 тестових частинок у кубі, побудована оцінка кожної варіації алгоритму та оптимізації програмного коду, побудована загальна оцінка оптимізації на прикладі моделювання течії в’язкої рідини за допомогою GLUT.

Висновки роботи та програмний код можуть бути використані як у гідродинаміці згладжених частинок, так і в інших тривимірних моделях із обчисленням найближчих сусідів: відеоігри, комп’ютерне моделювання, машинне навчання тощо.

Дана робота підкреслює необхідність цих алгоритмів та наводить способи їхньої інтеграції у сучасні обчислювальні системи. Програмний код є сумісний з CUDA, тому у майбутньому процес можна буде прискорити завдяки переносу обчислень до відеопроцесора.

ABSTRACT

Introductory note: 90 pages, 5 tables, 54 figures, 2 appendixes, 18 sources.

NEAREST NEIGHBOR SEARCH, BOX, SPATIAL HASHING, OPTIMIZATION, SMOOTHED PARTICLE HYDRODYNAMICS, SIMULATION.

Object of research – optimization of nearest neighbor search algorithm for smoothed particle hydrodynamics.

Purpose of work – research existing algorithms and their modifications, choose the most effective one for computations, perform an evaluation experiment on a fluid simulation.

Methods of research – boxed algorithm and spatial hashing algorithm, program optimization.

This work is dedicated to researching of existing optimization algorithms and their program enhancements by modern approaches. The most widely used nearest neighbor search algorithms, optimization approaches for system applications were considered. Experiment was conducted for 1000 particles within cube and evaluation of each proposed method and optimization has been performed. Overall optimization assessment is calculated using the fluid simulation via GLUT extension.

Results of this work and source code may be applied in smoothed particle hydrodynamics as well as in other fields with NNS use: videogames, 3D simulation, machine learning etc.

This work emphasizes the importance of these algorithms and brings up means of their integration within modern evaluation systems. The written code is CUDA-compatible hence in the future evaluation process may be speeded up by means of GPU.

ЗМІСТ

С.

Вступ ... 8

1 Системний аналіз проблеми оптимізації алгоритму пошуку сусідів у методі гідродинаміки згладжених частинок та постановка задач дослідження ... 9

1.1 Системний аналіз проблеми оптимізації алгоритму пошуку сусідів у методі гідродинаміки згладжених частинок ... 9

1.1.1 Вербальна модель системи ... 9

1.1.2 Морфологічний опис системи ... 10

1.1.3 Функціональна модель системи ... 10

1.1.4 Інформаційна модель системи ... 13

1.2 Аналіз сценаріїв вирішення проблеми оптимізації алгоритму пошуку найближчих сусідів ... 15

1.2.1 Модель аналізу проблеми ... 15

1.2.2 Оцінювання вектора пріоритетів незадоволеностей методом аналізу ієрархій ... 16

1.2.3 Модель вирішення проблеми ... 20

1.3 Змістовна та формальна постановка задачі ... 20

1.4 Постановка задач дослідження ... 23

2 Вибір та обґрунтування методу розв’язання ... 24

2.1 Загальні відомості про алгоритм пошуку найближчих сусідів ... 24

2.2 Наївний алгоритм пошуку найближчих сусідів ... 25

2.3 Боксовий алгоритм пошуку найближчих сусідів ... 26

2.4 Модифікації боксового алгоритму пошуку найближчих сусідів ... 27

2.5 Алгоритм просторового хешування ... 30

2.6 Гідродинаміка згладжених частинок ... 32

3 Програмна реалізація ... 41

3.1 Особливості програмної реалізації на мові C++ ... 41

3.2 Порівняння програмної реалізації алгоритмів ... 42

3.3 Оптимізація коду 44

3.3.1 Оптимізація за допомогою шаблонів 44

3.3.2 Оптимізація за допомогою багатонитковості 45

4 Результати обчислювального експерименту 46

5 Аналіз можливих застосувань 48

Висновки 49

Перелік джерел посилання 50

Додаток А Результати обчислювального експерименту 52

Додаток Б Вихідний код програми 71

ВСТУП

Проблема пошуку найближчих сусідів відома ще з минулого століття, коли треба було організовувати план міста або будинку, але лише починаючи з двохтисячного року, із ростом ринка візуальних технологій і почали з'являтися проблеми колізії об'єктів. Коли є набор тривимірних об'єктів та вони впливають друг на друга треба за найменший час знайти усі об'єкти, на які об'єкт запиту може впливати.

Існує декілька проблем пошуку сусідів, серед них такі як:

а) знайти N найближчих сусідів;

б) знайти усіх сусідів, які схожі с даним із деякою похибкою δ;

в) знайти усіх сусідів, які знаходяться на деякому фіксованому радіусі.

Серед запропонованих методів будемо розглядати наступні:

а) боксовий алгоритм;

б) наївний алгоритм;

в) алгоритм просторового хешування.

Спочатку було дуже зручно розбивати простір на «коробки» із фіксованим радіусом, але потім це було узагальнено на хешування, що і використовується найчастіше у сучасних програмних продуктах.

Будемо намагатись використати деякі припущення щодо розташування даних, на цих припущеннях ми будемо намагатись прискорити алгоритм, з деякою похибкою.

Час обчислення алгоритму – найбільш важлива у цій роботі характеристика. Якщо усі вищезазначені алгоритмі відомі та поширені, їхня реалізація залежить від платформи, під яку цей алгоритм реалізується. Тому у цій роботі будуть розглянуті сучасні підходи програмування, за допомогою яких ми будемо намагатись прискорити процес обчислення.

1 СИСТЕМНИЙ АНАЛІЗ ПРОБЛЕМИ ОПТИМІЗАЦІЇ АЛГОРИТМУ ПОШУКУ СУСІДІВ У МЕТОДІ ГІДРОДИНАМІКИ ЗГЛАДЖЕНИХ ЧАСТИНОК ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Системний аналіз проблеми оптимізації алгоритму пошуку сусідів у методі гідродинаміки згладжених частинок

1.1.1 Вербальна модель системи

Об’єкт аналізу – «Алгоритм пошуку найближчих сусідів».

Предмет аналізу – «Застосування алгоритму пошуку найближчих сусідів у гідродинаміці згладжених частинок».

Точка зору: дослідник.

Ціль: оптимізація алгоритму пошуку сусідів з метою прискорення обчислень для моделювання динамічних об’єктів у методі гідродинаміки згладжених частинок.

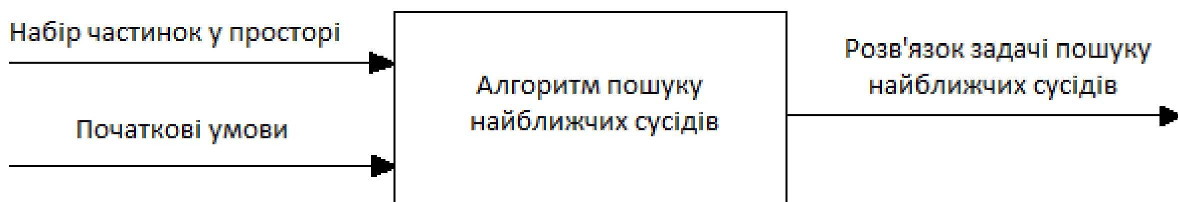

Рисунок 1.1 – Модель типу «чорний ящик»

Щоб відійти від деталей імплементації, використовується модель типу «чорний ящик». Вона змінює акцент від внутрішніх особливостей системи на її взаємодії із зовнішнім середовищем, тому зміст не розкривається. Входом цієї моделі є початкова конфігурація системи (або вплив середовища на неї), а виходом – цільовий продукт. Як можна бачити на рис. 1.1 «чорний ящик» – це алгоритм пошуку найближчих сусідів, його зовнішнє середовище – початкові

умови роботи алгоритму, а виходом є модель розв'язок задачі пошуку найближчих сусідів для заданих частинок.

1.1.2 Морфологічний опис системи

Для морфологічного опису системи будемо використовувати опис зовнішнього середовища. Він представляє собою усі об'єкти за межею системи, які впливають на результат роботи системи та об'єкти на яких впливає безпосередньо сама система. На рис. 1.2 зображено зовнішнє середовище нашої системи. Тобто: як і підприємства, для яких ця система розроблена, так і сучасна теорія алгоритмів і гідродинаміки. Впливають також суб'єктивність дослідника і швидкість обчислювального обладнання.

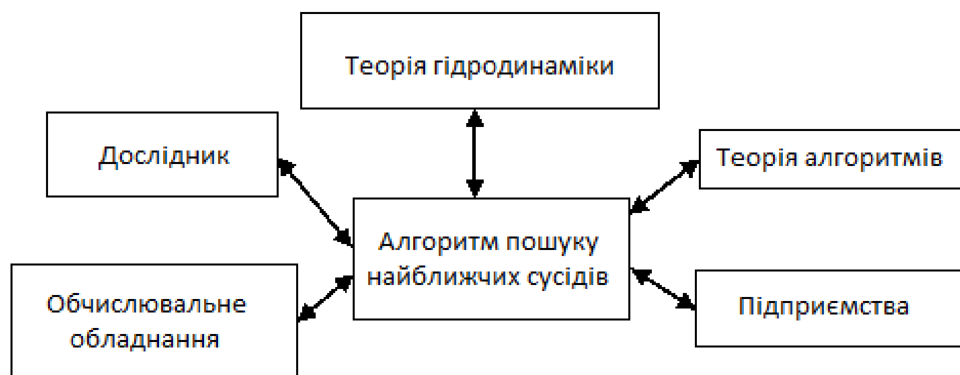

Рисунок 1.2 – Модель зовнішнього середовища системи

1.1.3 Функціональна модель системи

Будемо моделювати функціональну модель системи за допомогою контекстної діаграми IDEF0. Вона відображує функціонування системи у цілому (рис. 1.3) за допомогою розбиття її на функціональні блоки, у кожного з яких

можуть бути ресурси, що споживаються цим блоком та продукт, що цей функціональний елемент виробляє.

Кожен блок цієї діаграми (IDEF0) може бути декомпонований для більш детального опису процесу. Цей процес дроблення може продовжуватися доти не буде досягнутий необхідний рівень деталізації опису бізнес-моделі.

На рис. 1.3 наведена контекстна діаграма IDEF0 на рівні А-0. На цьому етапі ця діаграма нагадує «чорний ящик»: вона теж має входи та виходи, але у IDEF0 показані також елементи із зовнішнього середовища системи, такі як «Теорія алгоритмів» чи «Обчислювальне обладнання». Декомпозиція процесу «Алгоритм пошуку найближчих сусідів» наведена на рис 1.4. Ця контекстна діаграма більш детально описує «чорний ящик» за допомогою інших «чорних ящиків»: «Генерація боксів», «Розподіл частинок по боксах», «Аналіз боксів згідно моделі». Знову, «чорні ящики» можливо знову декомпонувати. Декомпонований елемент «Аналіз боксів згідно моделі» наведений на рис. 1.5.

Скористуємось методом IDEF3 – це метод, який дає аналітикам можливість описати певну послідовність виконання процесів, а також елементи, які беруть участь в одному процесі. Контекстна діаграма розглядуваного процесу зображена на рис. 1.6, а декомпозиція елементу «Аналіз боксів згідно моделі» – на рис. 1.7.

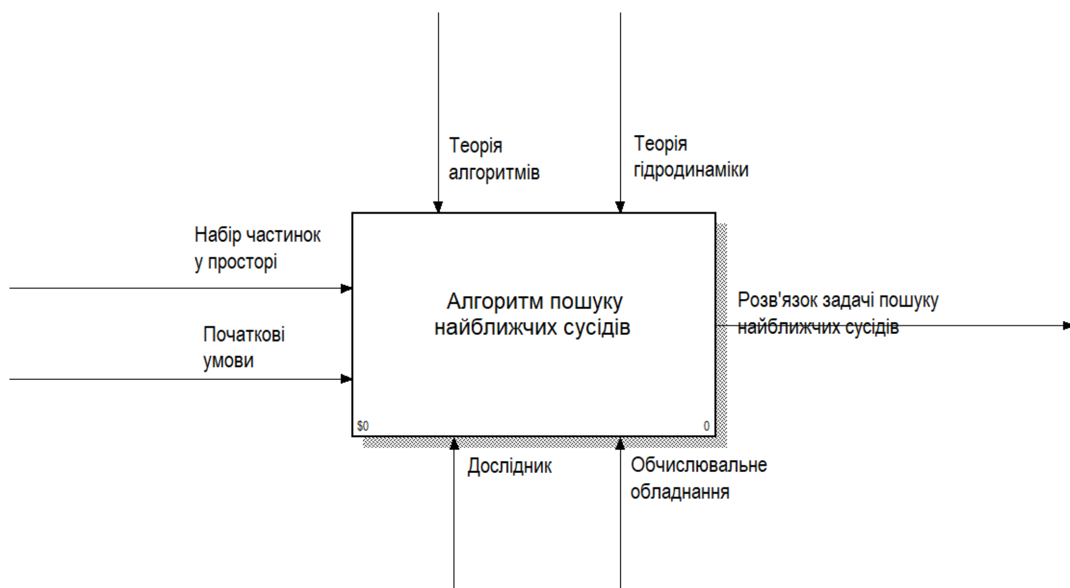

Рисунок 1.3 – Контекстна діаграма «Алгоритм пошуку найближчих сусідів»

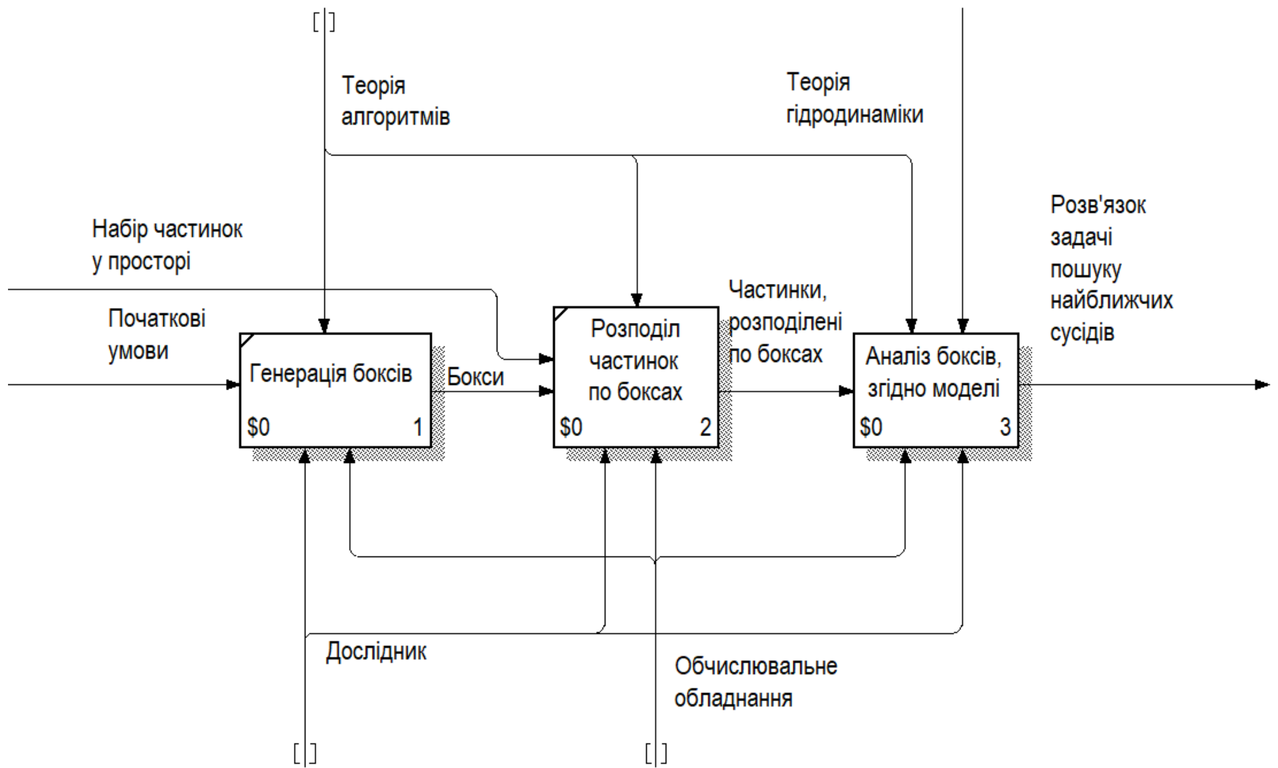

Рисунок 1.4 – Діаграма IDEF0 декомпозиції «Алгоритм пошуку найближчих сусідів»

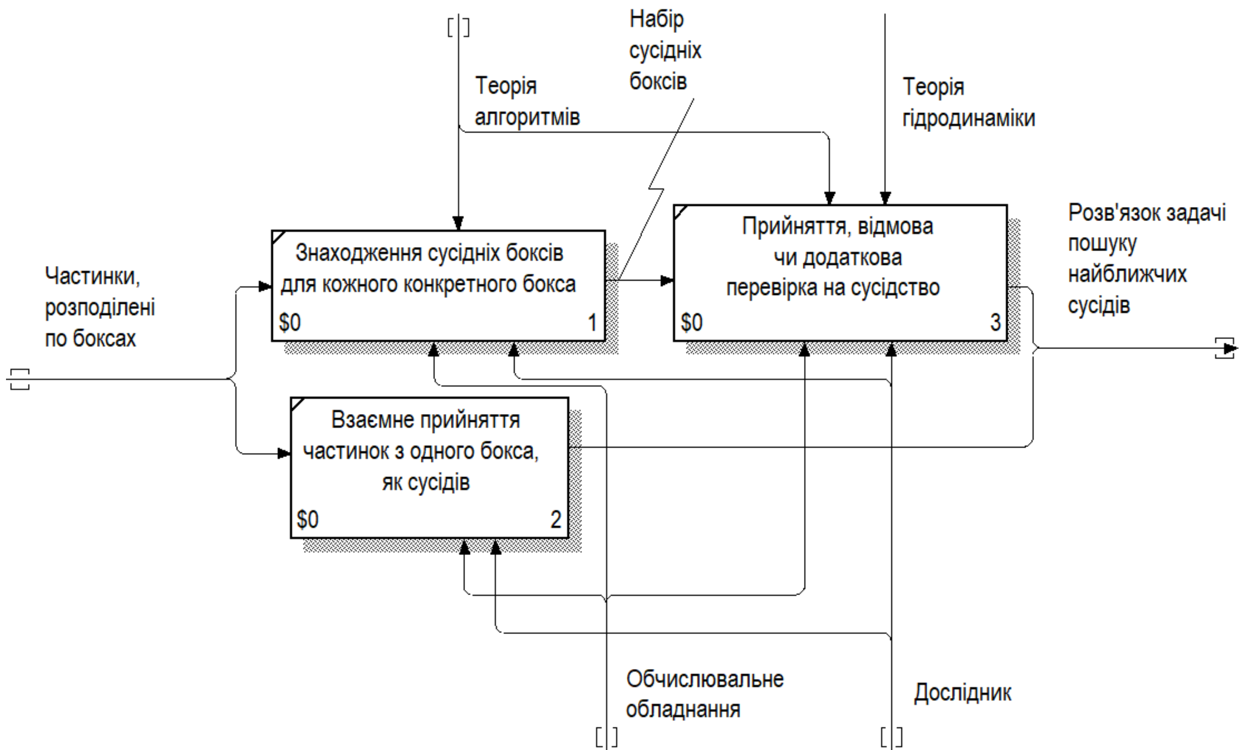

Рисунок 1.5 – Діаграма IDEF0 декомпозиції «Аналіз боксів згідно моделі»

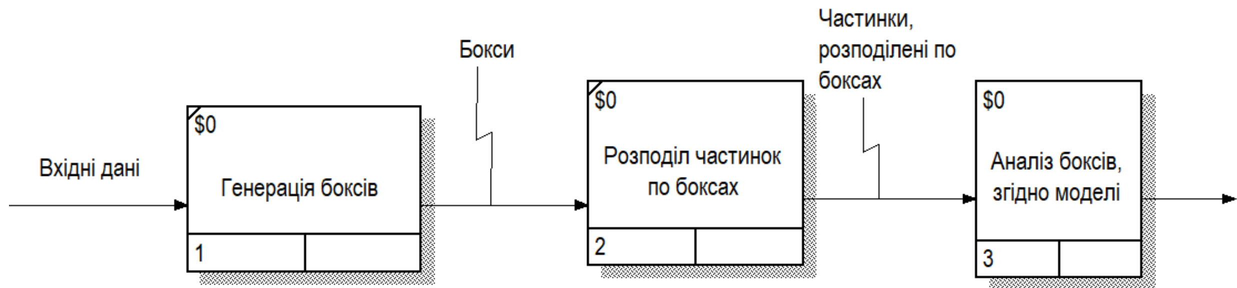

Рисунок 1.6 – Опис роботи «Алгоритм пошуку найближчих сусідів» у нотації IDEF3

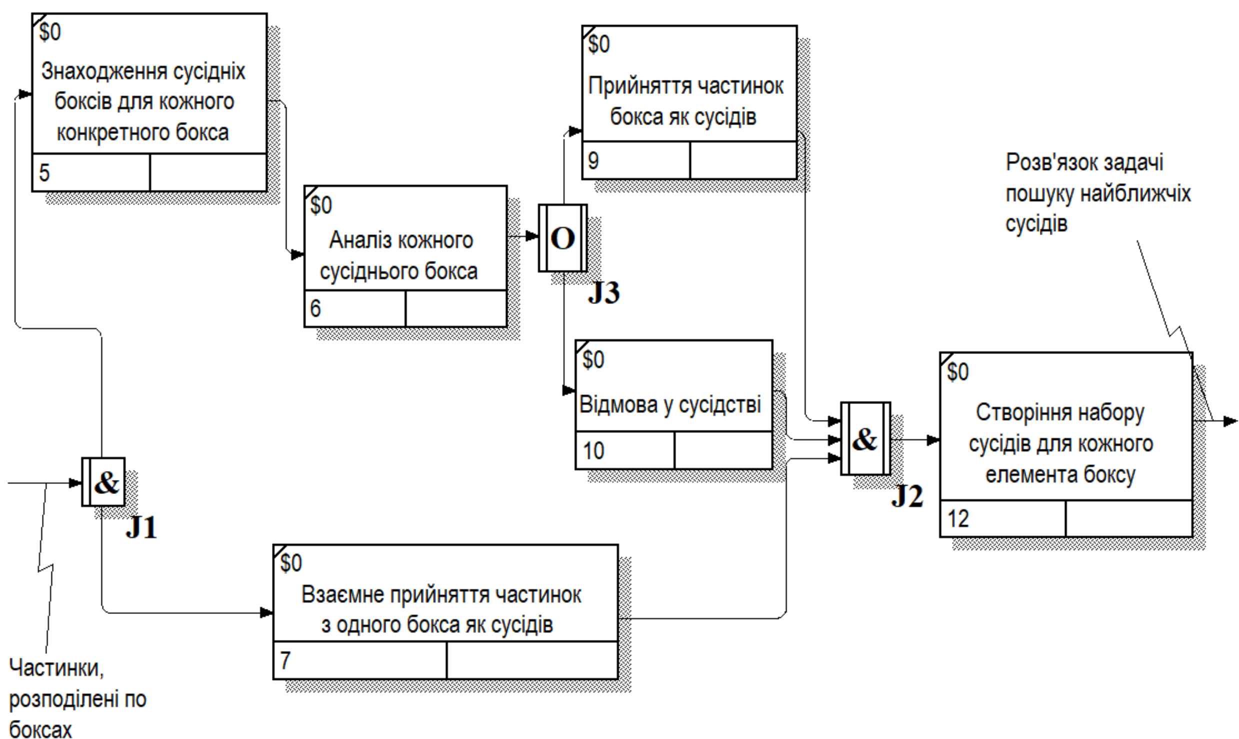

Рисунок 1.7 – Діаграма IDEF3 декомпозиції «Алгоритм пошуку найближчих сусідів» (рівень A1)

1.1.4 Інформаційна модель

Інформаційна модель системи представимо за допомогою діаграми потоків даних DFD. Діаграма DFD описує зовнішні до системи джерела і адресати

даних, логічні функції, до яких здійснюється доступ. Вона акцентує увагу дослідника на складі і взаємозв’язку потоків даних, що зв’язують декілька функціональних блоків між собою.

Для нашого процесу DFD-діаграма наведена на рис. 1.8, декомпозиція елементу «Алгоритм пошуку найближчих сусідів» (DFD-діаграма 1-го рівня) – на рис. 1.9.

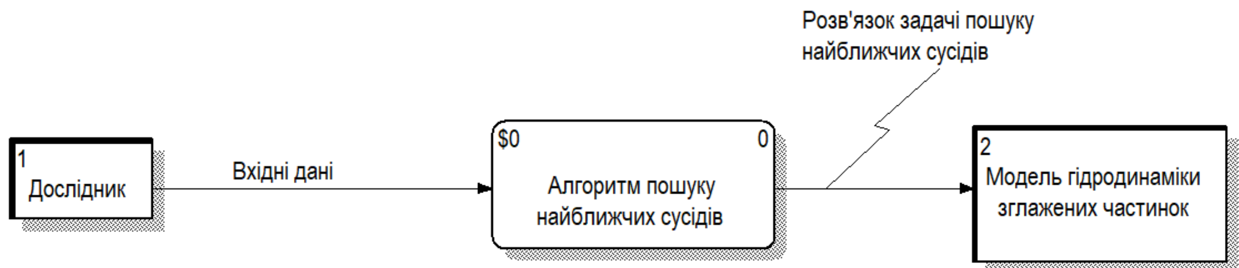

Рисунок 1.8 – DFD-діаграма «Алгоритм пошуку найближчих сусідів»

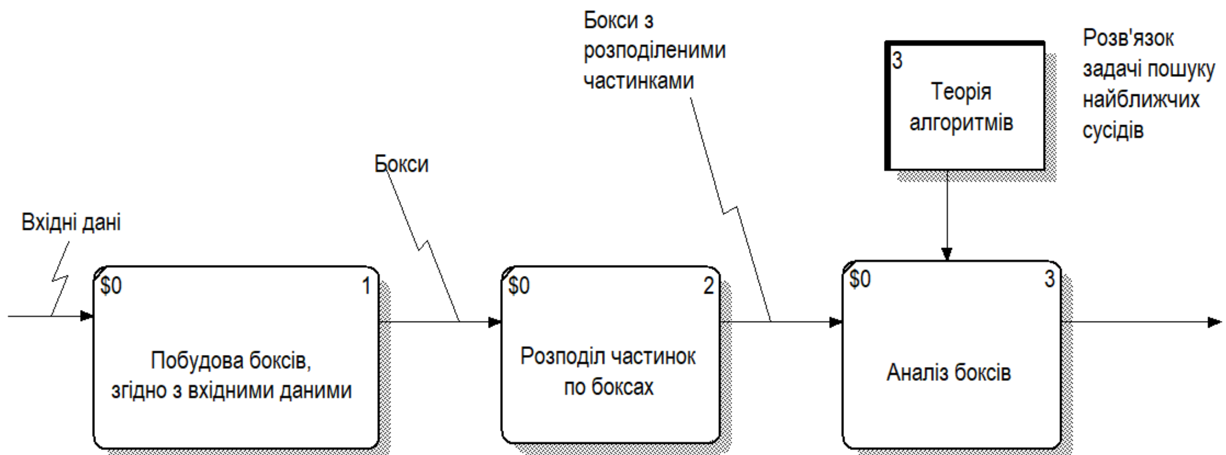

Рисунок 1.9 – DFD-діаграма 1-го рівня декомпозиції «Алгоритм пошуку найближчих сусідів»

1.2 Аналіз сценаріїв вирішення проблеми оптимізації алгоритму пошуку найближчих сусідів

1.2.1 Модель аналізу проблеми

Досліджуємо задачу про вибір мови програмування, яка буде використовуватись для написання алгоритму пошуку найближчих сусідів та за допомогою якої будемо намагатись прискорити обчислення.

Обиратимемо мову програмування, ґрунтуючись на наступних критеріях:

а) перший критерій (К1) – загальний час обчислення;

б) другий критерій (К2) – якість побудови абстракції;

в) третій критерій (К3) – складність написання коду програми та її підтримка.

Обирати будемо з множини альтернатив:

а) перша альтернатива (А1) – C;

б) друга альтернатива (А2) – C++;

в) третя альтернатива (А3) – Java;

г) четверта альтернатива (А4) – Python.

Побудуємо ієрархічну структуру, використовуючи метод парних порівнянь. Ієрархічна структура наведена на рис. 1.10.

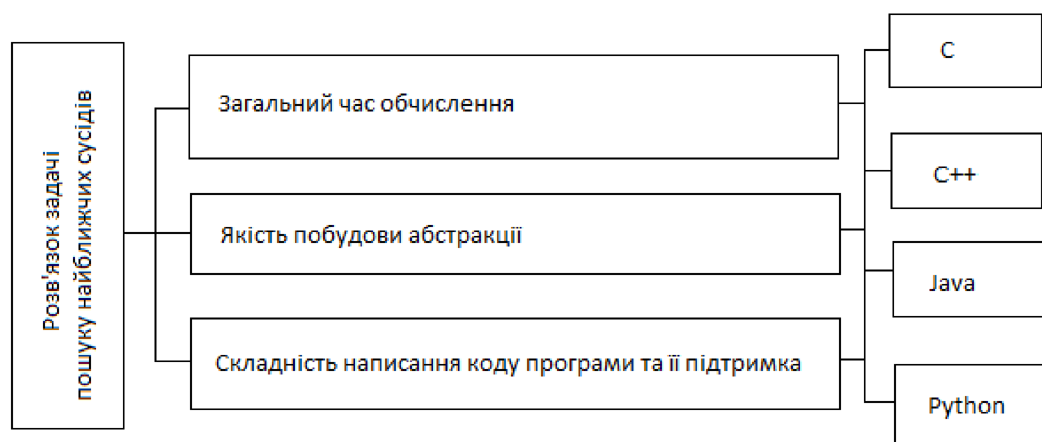

Рисунок 1.10 – Ієрархічна модель процесу аналізу розв’язку задачі

1.2.2 Оцінювання вектора пріоритетів незадоволеностей методом аналізу ієрархій

Тепер побудуємо матрицю парних порівнянь. Віддамо перевагу першому критерію, бо час обчислення – це найважливіший параметр у даній роботі. Будемо намагатись якнайсильніше прискорити час обчислення за допомогою різних оптимізацій. Якість побудови абстракції – це не менш важливий параметр, якісна абстракція необхідна у сучасних програмних системах, для того щоб була можливість виправлення або доповнення програмного коду без зміни великої кількості існуючого. Складність написання коду програми – без виключень, для підприємств цей параметр є найважливішим. Якщо програмний код написаний за допомогою дуже складної або старої мови чи фреймворку, підтримка цього продукту буде коштувати достатньо багато. Але у роботі цей критерій має найменшу важливість. Побудована матриця порівнянь критеріїв наведена у табл. 1.1.

Таблиця 1.1 – Матриця парних порівнянь критеріїв

Критерії оцінювання	К1	К2	К3	Оцінки компонентів	Вектор пріоритетів	Величина значущості
К1	1	3	9	3,000000	0,671620	0,970130
К2	$\frac{1}{3}$	1	5	3,185630	0,265433	1,114820
К3	$\frac{1}{9}$	$\frac{1}{5}$	1	0,281140	0,062941	0,944120
Всього				4,466780		3,029060

Індекс узгодженості дорівнює

$$CI^{K} = \frac{3{,}02906 - 3}{3 - 1} = 0{,}0145319\,.$$

Матриця критеріїв – це матриця третього порядку, її випадковий індекс дорівнює $RI^K = 0,58$.

Відношення узгодженості дорівнює:

$$CR^K = \frac{0,0145319}{0,58} = 0,025055 \approx 2,5\% .$$

Відношення узгодженості є близьким до $0,1$. Вважаємо, що матриця парних порівнянь критеріїв мови програмування побудована вірно.

Тепер проведемо порівняльний аналіз альтернатив. Для прийняття рішення про використання методу, оцінимо їх щодо кожної альтернативи.

Найкращий час обчислення у програм, які написані на таких мовах, як асемблер, C та C++. Вважається, що C++ має гірші показники ніж C, але це не так: написаний код на C, який був скомпільований за допомогою C та C++ компілятора, має однакову швидкість [15]. C и C++ може займати більш часу на ініціалізацію, але при переході до масштабів підприємства чи виробництва будемо мати однакові показники швидкості. Python дуже близький за своїм характером до C та C++, але оскільки він не є компільованою мовою програмування та має багато систематичних недоліків (такі як GIL), він є найгіршим вибором за показниками швидкості, відстає навіть від Java.

Випадковий індекс для матриць четвертого порядку дорівнює $RI^K = 0,9$, порівняння наведене у табл. 1.2.

Індекс узгодженості та відношення узгодженості дорівнюють

$$CI^K = \frac{4,02103 - 4}{4 - 1} = 0,007011, \; CR^K = \frac{0,00701076}{0,90} = 0.00778973 \approx 0,78\% ,$$

що є близьким до $0,1$. Вважаємо, що порівняння побудовано правильно.

Таблиця 1.2 – Порівняння за першим критерієм

Критерії оцінювання	A1	A2	A3	A4	Оцінки компонентів	Вектор пріоритетів	Величина значущості
A1	1	1	5	9	2,590020	0,430101	0,994012
A2	1	1	5	9	2,590020	0,430101	0,994012
A3	$\frac{1}{5}$	$\frac{1}{5}$	1	3	0,588566	0,097738	1,107701
A4	$\frac{1}{9}$	$\frac{1}{9}$	$\frac{1}{3}$	1	0,253279	0,042060	0,925313
Всього					6,021880		4,021031

Якісно побудована абстракція дає можливість зменшити витрати на підтримку програмного продукту. Варто зазначити, що не існує абстракції, яку можливо побудувати за допомогою однієї мови, але неможливо за допомогою іншої. Цей критерій у першу чергу підкреслює якість абстракції. Як показують такі приклади, як Linux Kernel [14], побудувати якісний шар абстракції можна навіть на таких мовах, як C і асемблер. C максимально приближений до мови, за допомогою якої взаємодіє процесор. І Java, і Python пропонують класичний підхід до об’єктно-орієнтованого програмування, але вони програють C++ у багатьох сучасних методах написання коду. Порівняння наведене у табл. 1.3.

Індекс узгодженості та відношення узгодженості дорівнюють

$$CI^{K} = \frac{4{,}10857 - 4}{4 - 1} = 0{,}0361899\,,\; CR^{K} = \frac{0{,}0361899}{0{,}90} = 0{,}040211 \approx 4\%\,,$$

що є близьким до $0{,}1$. Вважаємо, що порівняння побудовано правильно.

За складністю написання, безумовно, мова Python є найпростішою і в розробці, і в підтримці. Мова програмування Java є дещо складнішою, але необхідність контролю за ресурсами у C та C++ значно ускладнює задачу. Порівняння наведене у табл. 1.4.

Таблиця 1.3 – Порівняння за другим критерієм

Критерії оцінювання	A1	A2	A3	A4	Оцінки компонентів	Вектор пріоритетів	Величина значущості
A1	1	$\frac{1}{9}$	$\frac{1}{4}$	$\frac{1}{3}$	0,310202	0,050068	0,851158
A2	9	1	5	6	4,053611	0,654272	0,966868
A3	4	$\frac{1}{5}$	1	2	1,124681	0,181530	1,225320
A4	3	$\frac{1}{6}$	$\frac{1}{2}$	1	0,707110	0,114131	1,065220
Всього					6,195591		4,108571

Таблиця 1.4 – Порівняння за третім критерієм

Критерії оцінювання	A1	A2	A3	A4	Оцінки компонентів	Вектор пріоритетів	Величина значущості
A1	1	2	$\frac{1}{4}$	$\frac{1}{7}$	0,516973	0,083158	1,039470
A2	$\frac{1}{2}$	1	$\frac{1}{6}$	$\frac{1}{9}$	0,310202	0,049898	0,898160
A3	4	6	1	$\frac{1}{3}$	1,681790	0,270526	1,194821
A4	7	9	3	1	3,707790	0,596420	0,946697
Всього					6,216760		4,079150

Індекс узгодженості та відношення узгодженості дорівнюють

$$CI^K = \frac{4,07915-4}{4-1} = 0,026383\,,\ CR^K = \frac{0,026383}{0,90} = 0,0293148 \approx 2,9\%\,,$$

що є близьким до $0,1$. Вважаємо, що порівняння побудовано правильно.

1.2.3 Модель вирішення проблеми

За допомогою розрахунків у табл. 1.1 – 1.4 можемо зробити вибір однієї з даних альтернатив. Кінцеві дані наведені у табл. 1.5. Як видно, була обрана мова C++, бо вона поєднує в собі як велику швидкість обчислень, так і якісну модель.

Таблиця 1.5 – Кінцеві дані

Критерій /Альтернатива	K1	K2	K3	Узагальнені пріоритети
A1	0,430101	0,050068	0,083158	0,307388
A2	0,430101	0,654272	0,049898	0,465670
A3	0,097738	0,18153	0,270526	0,149736
A4	0,042060	0,114131	0,596420	0,096082

1.3 Змістовна та формальна постановка задачі

Існуюча модель для побудови динамічних об’єктів у тривимірному просторі із застосуванням гідродинаміки згладжених частинок [13] не використовує сучасних методів до написання програмного коду та оптимізацій алгоритмів. Будемо намагатися зменшити час обчислення застосовуючи різні методи.

Модель течії в’язкої рідини використовує рівняння Нав’є-Стокса, що складається з двох рівнянь: руху у суцільному середовищі і нерозривності та має наступний вигляд:

$$\rho\left(\frac{\partial}{\partial t}+\mathbf{u}\cdot\nabla\right)\mathbf{u}=-\nabla p+\mu\nabla\cdot\left(\nabla\mathbf{u}\right)+\mathbf{f},$$

$$\nabla\cdot\mathbf{u}=0,$$

де $\mathbf{u}:\Omega\times[0,t]\to\mathbb{R}^3$ – швидкість;

$\rho=\text{const}$ – густина рідини;

$p:\Omega\times[0,t]\to\mathbb{R}$ – гідродинамічний тиск рідини;

$\mathbf{f}$ – вектор зовнішньої сили;

μ – динамічна в’язкість рідини;

∇ – оператор набла.

У загальному випадку оператор набла визначений як векторний диференціальний оператор, компоненти якого є частинними похідними за координатами. У тривимірному випадку оператор має вигляд:

$$\nabla=\left(\frac{\partial}{\partial x},\frac{\partial}{\partial y},\frac{\partial}{\partial z}\right).$$

Після застосування методу Лагранжа спростимо основне рівняння знехтувавши рівнянням нерозривності, після спрощення зовнішні сили перестануть залежати від взаємного розташування між собою після чого отримаємо наступну форму:

$$\rho\frac{\mathrm{d}\mathbf{u}}{\mathrm{d}t}=-\nabla p+\mu\nabla^2\mathbf{u}+\mathbf{F}.$$

Права частина рівняння цього рівняння складається із зовнішніх та внутрішніх сил, які можуть бути записані у вигляді суми $\mathbf{F}=\mathbf{f}^{зовн.}+\mathbf{f}^{внут.}$. На основі цього стає можливим розрахунок прискорення i-ої частинки, що набуває наступного вигляду

$$\mathbf{a}_i=\frac{\mathrm{d}\mathbf{u}_i}{\mathrm{d}t}=\frac{\mathbf{F}_i}{\rho_i},$$

де $\mathbf{a}_i$ – вектор прискорення i-ої частинки;

$\mathbf{u}_i$ – вектор швидкості i-ої частинки;

$\mathbf{F}_i$ – сума зовнішніх та внутрішніх сил i-ої частинки;

ρ_i – густина i-ої частинки;

t – час.

Сучасні обчислювальні системи мають декілька логічних та фізичних ядер, що надає можливість прискорити процес обчислення у стільки разів, скільки логічних ядер надає нам система. Найпотужніший процесор на поточний момент часу – це AMD EPYC 7742 [16], він має 128 логічних ядер, тому обчислення можуть бути прискорені у 128 разів.

За останні роки ринок обчислювальної техніки був керований прагненням досягти більш якісного зображення, тому деякі специфічні обчислення (множення матриць) більш ефективні на відеокартах, ніж на процесорах. У [6] можна побачити як змінювалася теоретична пропускна здатність PCI-E.

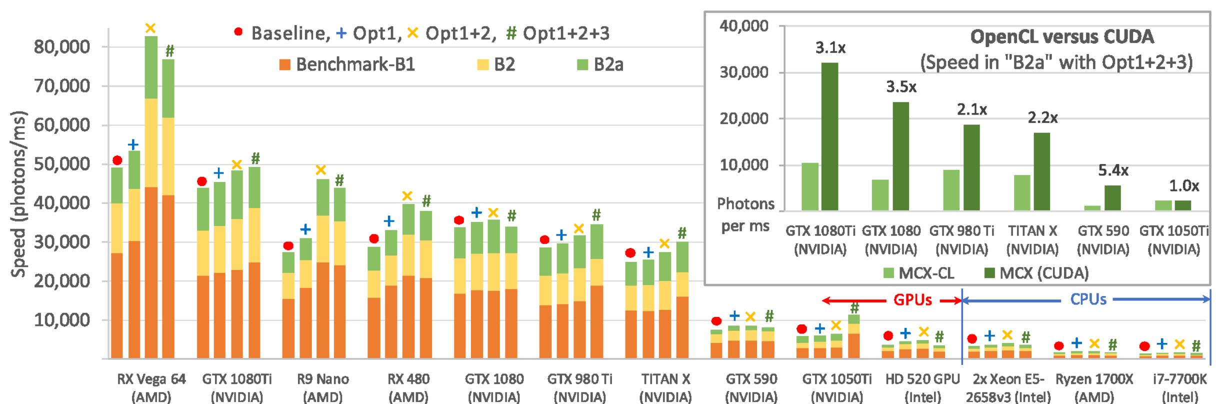

Рисунок 1.11 – Порівняння OpenCL та CUDA [7]

Для обчислень на відеокартах використовуються такі фреймворки, як OpenCL або CUDA. Фреймворк CUDA використовується на відеокартах NVIDIA, і, починаючи з серії Pascal надає низькорівневий програмний інтерфейс для цих обчислень і за швидкістю OpenCL залишається позаду. Варто зазначити, що не усі відеопроцесори GeForce підтримують таке апаратне обчислення. Яскравими прикладами можуть бути такі відеопроцесори, як 1080 Ti та

Titan X (Pascal). На рис. 1.11 наведений графік порівнянь обчислень CUDA та OpenCL для деяких відеопроцесорів GeForce серії Pascal.

У майбутньому, велику частину обчислень може буде робити значно швидше на відеопроцесорах, тому дуже важливо вже зараз намагатися використовувати засоби, сумісні з ланцюгом компіляції CUDA.

1.4 Постановка задач дослідження

Як можна побачити [13], обчислення за допомогою наївного алгоритму є неефективними. При великих обсягах даних вартість цього процесу може дуже швидко збільшитись і це обернеться втратами для підприємства.

Оскільки для обчислень на відеокарті за допомогою CUDA може використовуватися як мова C, так і C++ [10], програма може використовувати не тільки процесор, що прискорить виконання обчислень.

Виходячи з зазначеного вище, завданнями цієї роботи є:

а) побудувати якісну модель сумісну із CUDA для використання алгоритму пошуку найближчих сусідів;

б) провести ретельний аналіз модифікацій алгоритму пошуку найближчих сусідів;

в) провести ретельний аналіз можливих оптимізацій коду сучасними методами;

г) використати побудовану модель для гідродинаміки згладжених частинок.

2 ВИБІР ТА ОБГРУНТУВАННЯ МЕТОДУ РОЗВ'ЯЗАННЯ

2.1 Загальні відомості про алгоритм пошуку найближчих сусідів

Пошуку найближчого сусіда (Nearest Neighbor Search) – це проблема оптимізації знаходження точки (або частинки) з певного набору, найближчої до заданої. Дональд Кнут [4] назвав цю проблему «проблему поштового відділення»: знаходження найближчого сусіда нагадувало пошук найближчого відділення пошти, де громадянин може отримати посилку.

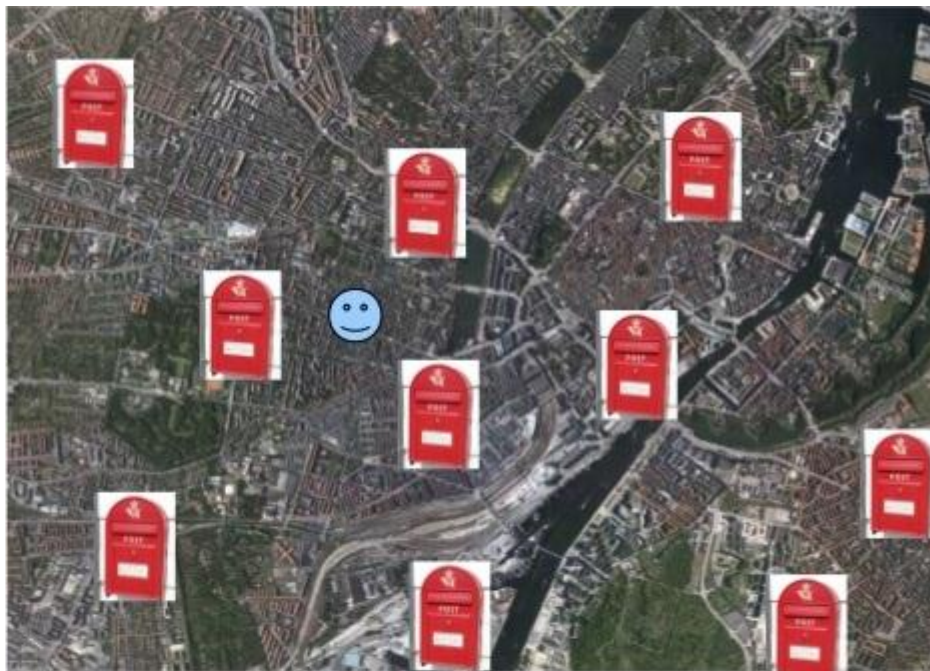

Рисунок 2.1 – Проблема поштового відділення

Одним з уточнень алгоритму є «Пошук найближчих сусідів із фіксованим радіусом» або просто «Пошук найближчих сусідів», який розглянуто у цій роботі.

Ця проблема може бути розглянута як у двовимірному, так і у тривимірному просторі. У цій роботі для наочності розглядається тривимірний випадок, але наданий код може бути використаний для N-вимірного простору.

Нехай область задана у вигляді $\Omega = \{X_1 \le x \le X_2, Y_1 \le y \le Y_2, Z_1 \le z \le Z_2\}$, а

кожна окрема частинка задана у вигляді $p_i = (x_i, y_i, z_i)^T$, набір частинок – $S = \{p_1, p_2, ..., p_N\}$, де N – кількість частинок у області, радіус пошуку – r. Тоді для кожної частинки p_i потрібно побудувати множину $Z_i = \{q_1, q_2, ..., q_k\}$, де k може приймати значення від 0 до N, елементи якої задовольняють нерівності

$$\left\| p_i - q_j \right\| \le r,\ j = \overline{1,k}.$$

2.2 Наївний алгоритм пошуку найближчих сусідів

Найпростішим розв’язком цієї проблеми буде перебір або наївний алгоритм. Кроки алгоритму виглядають наступним чином:

а) для кожного елемента p_i з множини S розглянемо усі інші елементи z_j, $i \ne j$;

б) якщо $\left\| p_i - z_j \right\| \le r$, то додамо елемент z_j у множину Z_i.

Складність цього алгоритму $O(N^2)$, тому використання цього алгоритму треба уникати. Незважаючи на його складність, цей алгоритм можна прискорити удвічі, якщо брати до уваги симетричність сусідства. Тобто: якщо p_i – це сусід p_j, то p_j – сусід p_i, $i \ne j$. Алгоритм прийме наступний вигляд:

а) для кожного елемента p_i з множини S розглянемо усі інші елементи z_j, $j > i$;

б) якщо $\left\| p_i - z_j \right\| \le r$, то додамо елемент z_j у множину Z_i, а елемент p_i – у множину Z_j.

Ця оптимізація дуже важлива, оскільки зниження часу обчислень майже удвічі може мати великий вплив на підприємстві, якщо за деяких обставин неможливе використання боксового алгоритму. Складність цього алгоритму – теж $O(N^2)$.

2.3 Боксовий алгоритм пошуку найближчих сусідів

З огляду на те, що у більшості випадків відстань між деякими частинками у декілька разів більше, ніж радіус пошуку, метою цього алгоритму є розбиття набору S на підгрупи (або бокси) та аналіз лише тих груп, які межують між собою.

У геометрії це буде означати розбиття на куби, ребро кожного з яких дорівнює r. Якщо повернутися на двовимірний простір, то цей механізм можна інтерпретувати, уявивши сітку, як показано на рис. 2.2.

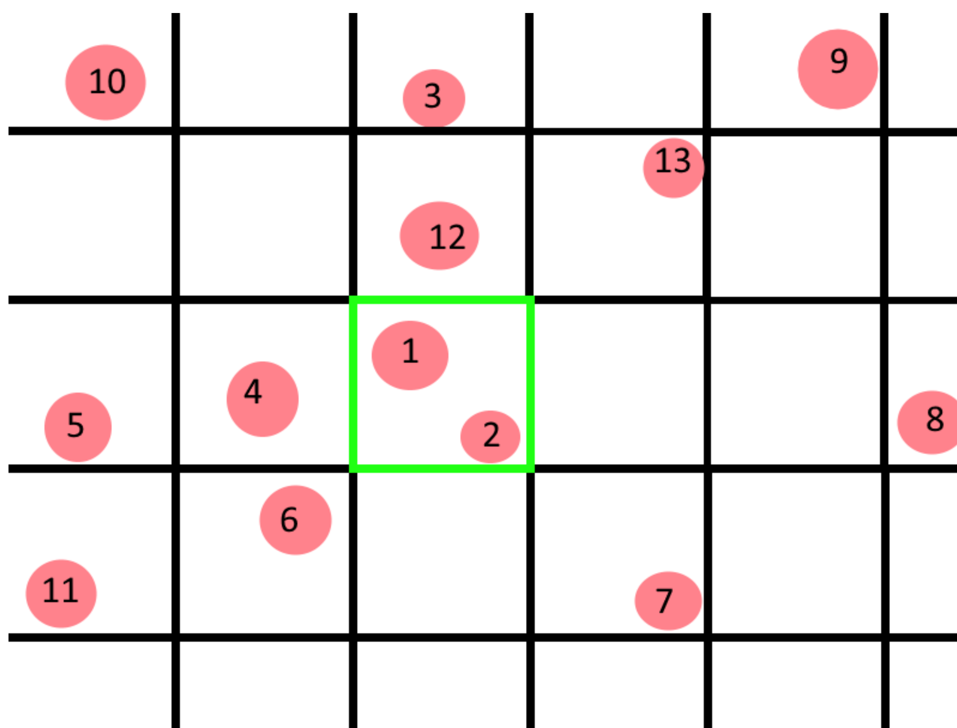

Рисунок 2.2 – Сітка з частинками

На цьому рисунку наочно показаний процес відбору за боксами. Наприклад, будемо шукати сусідів частинки 1, її бокс виділений. У цієї частинки лише 8 сусідніх боксів, тому усі частинки, такі як 3, 5, 8, 9, 10 та 11 автоматично не підходять. Тому розглядаються лише частинки 1, 2, 4, 6, 7, 12 та 13. Далі, усі частинки, які знаходяться всередині, перевіряються наївним алгоритмом. Повний алгоритм має наступний вигляд:

а) розбити область Ω на множину боксів B (куби з ребром r), кількість кубів дорівнюватиме $\left\lceil \frac{X_2 - X_1}{r} \right\rceil$ для осі x (позначимо через l_x), $\left\lceil \frac{Y_2 - Y_1}{r} \right\rceil$ для осі y (позначимо через l_y), $\left\lceil \frac{Z_2 - Z_1}{r} \right\rceil$ для осі z, (позначимо через l_z), тоді множина боксів матиме вигляд $\mathrm{B} = \left\{ B_{i,j,k} \right\}$, $i = \overline{0, l_x}$, $j = \overline{0, l_y}$, $k = \overline{0, l_z}$;

б) кожен елемент $p_i = \left(x_i, y_i, z_i \right)$ множини S додати до відповідного бокса $B_{\left\lceil \frac{x_i - X_1}{r} \right\rceil, \left\lceil \frac{x_i - Y_1}{r} \right\rceil, \left\lceil \frac{z_i - Z_1}{r} \right\rceil}$;

в) для кожного бокса $B_{i,j,k}$ з множини B попарно розглянемо усі елементи z_i і z_j боксів $B_{\tilde{i},\tilde{j},\tilde{k}}$, $\tilde{i} = i-1$, i, $i+1$, $\tilde{j} = j-1$, j, $j+1$, $\tilde{k} = k-1$, k, $k+1$;

г) якщо $\left\| z_i - z_j \right\| \le r$, то додамо елемент z_j у множину Z_i, а елемент z_i – у множину Z_j.

Цей алгоритм також не робить ніяких припущень щодо розташування частинок, тому дає результат без похибки.

2.4 Модифікації боксового алгоритму пошуку найближчих сусідів

Усі наступні оптимізації алгоритму пошуку найближчих сусідів базуються на різниці між точним розв'язком і достатнім. У більшості випадків не потрібно мати точний розв'язок, тому деякі операції ми можемо опустити.

Спочатку візуалізуємо радіус пошуку сусідів для боксу. Як можна побачити на рис. 2.3, коли ребро бокса дорівнює r, то для центрального бокса усі можливі сусіди знаходяться у сусідніх боксах.

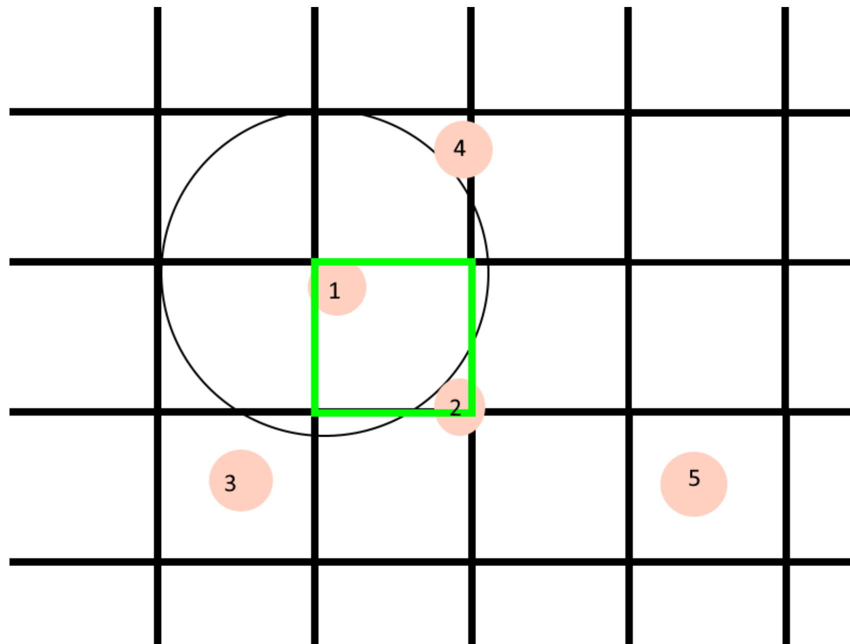

Рисунок 2.3 – Візуалізація радіусу пошуку, коли ребро дорівнює r

Як можна побачити, частинка 2, хоч і знаходиться у одному боксі з 1, не є її сусідом. А усі інші частинки потрібно перевіряти. Якщо допустити, що частинки розподілені за нормальним законом, то відношення площі кругу до площі області пошуку буде дорівнювати

$$p = \frac{\pi r^2}{3r \cdot 3r} = \frac{\pi}{9} = 0,349.$$

Тобто в усій області пошуку лише з третиною частинок буде виконуватися критерій сусідства. Тому якщо автоматично прийняти їх як сусідів, то майже 66% з них буде обрано неправильно.

Розглядаючи лише центральний бокс, можемо автоматично прийняти усі частинки як сусідів, тоді $\frac{1}{9}$ часу (з дев’яти боксів один приймається автоматично) буде заощаджена (на 12,5% швидше). При цьому у найгіршому випадку (частинка у куті) похибка буде дорівнювати:

$$r^2 - \frac{\pi r^2}{4} = r^2 \cdot 0,21.$$

Але помилка існує лише у кутах, і для деяких точок всередині бокса усі його точки будуть сусідами, тому практично цей метод має показати себе більш ефективним.

Наступний підхід базується на зменшені радіусу боксу у $\sqrt{2}$ раз. Це дозволяє уникнути помилки при взаємному прийнятті сусідів у центральному боксі. Натомість ми втрачаємо сусідів, які потрапили у виділену зону на рис. 2.4.

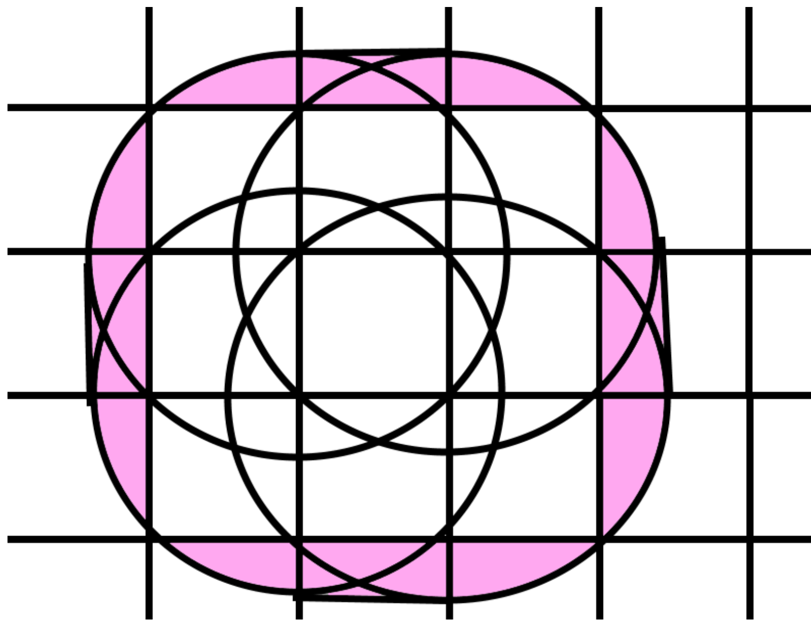

Рисунок 2.4 – Втрати, коли ребро бокса дорівнює $\frac{r}{\sqrt{2}}$

Можна з деякою похибкою прирівняти цю область до описаного кола і тоді при нормальному розподіленні частинок загальне відношення дорівнюватиме:

$$\frac{\left(3\frac{r}{\sqrt{2}}\right)^2}{\pi\cdot\left(\frac{3}{2}r\right)^2}=\frac{2}{\pi}=0,636\,,$$

тобто похибка буде дорівнювати менш ніж 36% у найгіршому випадку. На практиці результати мають бути значно кращі.

Ще однім підходом буде прийняття усіх частинок як сусідів. Цей підхід дозволяє зовсім видалити обчислення відстані, що дасть дуже велике прискорення обчислень, натомість похибка буде дуже великою. Теоретично, при нормальному розподілу частинок, відсоток частинок, що відстежуються, у найкращому випадку буде

$$\frac{\pi\left(\frac{3r}{2\sqrt{2}}\right)^2}{\left(\frac{3r}{\sqrt{2}}\right)^2}=\frac{\pi}{4}=0,79\,,$$

а у найгіршому

$$\frac{\pi\left(\frac{3r}{2\sqrt{2}}\right)^2-\frac{1}{2}\left(\pi\left(\frac{3r}{2\sqrt{2}}\right)^2-\left(\frac{2r}{\sqrt{2}}\right)^2\right)}{\left(\frac{3r}{\sqrt{2}}\right)^2}=\frac{\pi\frac{9}{4}-\frac{1}{2}\left(\pi\frac{9}{4}-4\right)}{9}=0,615\,,$$

але у деяких випадках такий алгоритм може мати місце незважаючи на його похибку.

2.5 Алгоритм просторового хешування

Ще один важливий підхід до оптимізації алгоритму побудовано не на припущеннях щодо розташування частинок, а на переході від тривимірних боксів до хеш-боксів (spatial hashing).

Цей підхід є де-факто стандартом для пошуку найближчих сусідів. Уперше ефективний підхід для обчислювань хешу був запропонований у [1] та пізніше був перевірений багатьма іншими дослідниками [2, 8].

Суть цього методу – замість тривимірного масиву використовувати хеш-

таблицю (std::unordered_map у мові C++) та замість фіксованих боксів використовувати абстрактні. Якість результатів цього метода та швидкість обчислень залежать від якості хеш-функції: вірогідність хеш-колізій має бути мінімальна, розміри хеш-таблиці не повинні бути занадто великими (час обчислень збільшиться через кеш-промахи процесору).

У [1] пропонують наступну хеш-функцію:

$$hash(x, y, z) = (x \cdot p_1 \operatorname{xor} y \cdot p_2 \operatorname{xor} z \cdot p_3) \bmod n,$$

де $p_1 = 73856093$, $p_2 = 19349663$, $p_3 = 83492791$.

Тут p_1, p_2, p_3 – достатньо великі прості числа, n – розмір хеш-таблиці.

У роботі [2] погоджується із цією формулою та запропонованими значеннями p_1, p_2, p_3. Для гідродинаміки згладжених частинок він пропонує наступний розмір хеш таблиці:

$$n_H = prime(2n),$$

де n – кількість частинок;

$prime(x)$ – функція пошуку найменшого простого числа, більшого, ніж x.

Офіційне джерело NVIDIA [3] пропонує іншу формулу, основану на побітовому зсуві:

$$hash = \left\lfloor \frac{x}{r} \right\rfloor << XSHIFT \left\| \left\lfloor \frac{y}{r} \right\rfloor << YSHIFT \right\| \left\lfloor \frac{z}{r} \right\rfloor << ZSHIFT,$$

де $XSHIFT$, $YSHIFT$, $ZSHIFT$ – заздалегідь визначені постійні змінні.

Алгоритм обчислень наступний:

а) для кожного елемента p_i множини S обчислити хеш h_i за допомогою формули хешу;

б) додати цей елемент у хеш-бокс H_{h_i};

в) для обчислення найближчих сусідів для кожного елемента $p_i = \left(x_i, y_i, j_i\right)$ множини S додати у множину можливих сусідів K_i усі елементи хеш-боксів із хешами виду

$$hash\left(x_i \cdot h_x \cdot r, y_i \cdot h_y \cdot r, z_i \cdot h_z \cdot r\right),$$

де h може приймати будь-яке зі значень $\{-1,0,1\}$;

г) для кожного елементу p_j множини K_i, якщо додатково перевірити кожен елемент набору елементів K_i на сусідство із p_i, якщо $\left\|p_i - p_j\right\| \le r$ додати p_i до множини Z_j, а елемент p_j – до множини Z_i.

Ця формула дозволяє обчислювати хеш за менший час, тому що операції побітового зсуву швидші ніж множення. Завдяки вбудованим засобам хеш-контейнерів багатьох мов програмування цей алгоритм буде себе добре показувати на будь-якій мові.

Відмітимо, що модифікації, зазначені у пункті 2.4, можна застосовувати і в цьому алгоритмі.

2.6 Алгоритм просторового хешування

Гідродинаміка згладжених частинок (Smoothed Particle Hydrodynamics, SPH) – це лагранжевий (безсітковий) чисельний метод який використовується для отримання розв’язків систем рівнянь з частинними похідними. Проблемна геометрія дискретизується на частинки, які представляють конкретні властивості об’єкта. Цей метод застосовується для моделювання потоків рідини. Завдяки представленню рідини як дискретної кількості частинок, спрощуються рівняння руху та в’язкості, а система набуває властивостей збереження енергії, імпульсу, кутового моменту, маси та ентропії.

В основі методу лежить використання локальних інтерполяцій на оточу-

ючих дискретних частинках для побудови неперервного поля апроксимацій яке може буде використано для дискретизації рівнянь поля. Інтерполяція виконується за допомогою функції ядра. Для даної задачі раціональним буде використання окремого випадку застосування ядерного інтерполянта підсумовування для знаходження густини, через яку буде виражена решта гідродинамічних рівнянь. Функція згладжування W, яку вводять для визначення зв'язку між частинками, має задовольняти наступним умовам:

$$\int_{\Omega} W(\mathbf{r},h)\mathrm{d}\mathbf{r}=1,\ \ W(\mathbf{r},h)\geq 0,$$

$$W(\mathbf{r},h)=W(-\mathbf{r},h),$$

$$W(\mathbf{r},h)=0 \Leftrightarrow \|\mathbf{r}\|>h,$$

де $\mathbf{r}$ – довільна точка у Ω,

h – радіус згладжування, параметр гладкості інтерполяції.

Для будь-якої функції $F(\mathbf{r})$ може бути знайдений її згладжений інтерполянт $F_s(\mathbf{r})$ через згортання ядром $W(\mathbf{r},h)$:

$$F_s(\mathbf{r})=\int F(\mathbf{r}')W(\mathbf{r}-\mathbf{r}',h)\mathrm{d}\mathbf{r}'.$$

Зазвичай за функцію ядра беруть Гаусове ядро, бо воно має незначний вплив на віддалені частинки. Проте для збільшення продуктивності змінимо ядро на сплайн, значення якого за межами двох радіусів згладжування обнуляється. Дане спрощення значно зменшує кількість розрахунків, через що значення будь-якої фізичної величини визначається відповідними значеннями сусідніх частинок, які потрапили у охоплювану радіусом область.

Для використання сплайну у подальших розрахунках вимагаємо симетричність та достатню гладкість, отримавши таким чином двічі неперервно-диференційоване ядро. Відповідний сплайн шостого ступеня приймає вигляд:

$$W(\mathbf{r},h)=\frac{315}{64\pi h^9}\begin{cases}\left(h^2-\|\mathbf{r}\|^2\right)^3, & 0\le\|\mathbf{r}\|\le h,\\ 0, & \|\mathbf{r}\|>h.\end{cases}$$

Припустимо, що нам відома множина усіх точок $\mathbf{r}_i$. Маючи загальну масу та густину частинок, легко знайти загальний об'єм кінцевого елемента. У випадку, коли точки являються достатньо густою вибіркою об'єму ядра, апроксимуємо цей вираз, замінивши інтеграл сумою

$$F_s(\mathbf{r})=\sum_j\frac{\mathrm{m}_j}{\rho_j}F(\mathbf{r}_j)W(\mathbf{r}-\mathbf{r}_j,h),$$

де m_j – маса частинки;

ρ_j – густина частинки;

$\Delta\mathbf{r}_j\approx\frac{\mathrm{m}_j}{\rho_j}$ – кінцевий об'єм елемента.

Важливим є те, що наближення $F_s(\mathbf{r})$ визначене усюди та диференційоване завдячуючи диференційованості ядра, попри значну похибку інтерполяції похідної. У випадку, якщо $F(\mathbf{r})=\rho(\mathbf{r})$, отримаємо наближення густини

$$\rho_s(\mathbf{r})\approx\sum_j\mathrm{m}_jW(\mathbf{r}-\mathbf{r}_j,h),$$

що залежить лише від маси та координати частинки.

В загальному випадку довжина згладжування може залежати від простору, $h=h(\mathbf{r})$, на основі чого виникає можливість розрахунку варіацій у густині вибірки.

Таким чином, оскільки єдиною вимогою є лише значення густини частинки, а загальна маса залишається постійною, то однозначна рівність між інтег-

ралом загального вигляду по об'єму $\rho_s(\mathbf{r})$ та загальною масою є неважливою.

Тому використаємо підхід названий «збиранням», де у якості ядра $\rho_s(\mathbf{r})$ виступає $W(\mathbf{r}_i - \mathbf{r}_j, h(\mathbf{r}_i))$, і потребує лише значення h_i для визначення густини i-ої частинки. Через що, використовуючи підхід «збирання» рівняння значно спрощується і форма $\rho_s(\mathbf{r})$ запису набуває вигляду

$$\rho_i = \sum_{j=1}^{N} \mathrm{m}_j W(\mathbf{r}_i - \mathbf{r}_j, h_i),$$

де N – кількість частинок у межах кола радіусу $2h$ навколо точки $\mathbf{r}_i$.

Застосувавши підхід інтерполяції ядра, виразимо й решту гідродинамічних показників.

Розглянемо внутрішні сили, які налічують у собі тиск та в'язкість. Для вираження тиску скористаємося законом ідеального газу

$$pV = nRT,$$

де $V = \frac{1}{p}$ – об'єм на одиницю маси;

n – кількість частинок на одну моль;

R – універсальна газова константа;

T – температура.

Для ізотермічної рідини із постійною масою права частина рівняння замінюється константою Больцмана

$$pV = \kappa,$$

$$p\frac{1}{\rho} = \kappa,$$

$$p = \kappa\rho.$$

Якщо тиск для кожної частинки відомий, то сила тиску на i-у частинку, представлена через інтерполяцію ядра, набуває вигляду:

$$\mathbf{f}_i^{тиск} = -\sum_{j \neq i} p_i \frac{\mathrm{m}_j}{\rho_i} \nabla W\left(\mathbf{r}_i - \mathbf{r}_j, h\right).$$

Проте отримана сила тиску не є симетричною, що може бути перевірено при взаємодії лише двох частинок. Задля вирішення цієї проблеми можна скористатися властивостями SPH

$$\mathbf{f}_i^{тиск} = -\rho_i \sum_{j \neq i} \left(\frac{p_i}{\rho_i^2} + \frac{p_j}{\rho_j^2} \right) \mathrm{m}_j \nabla W\left(\mathbf{r}_i - \mathbf{r}_j, h\right),$$

або використовуючи загальні фізичні властивості

$$\mathbf{f}_i^{тиск} = -\sum_{j \neq i} \frac{p_i + p_j}{2} \frac{\mathrm{m}_j}{\rho_j} \nabla W\left(\mathbf{r}_i - \mathbf{r}_j, h\right).$$

Таким чином, отримаємо силу тиску, що зберігає імпульс та кутовий момент, тим самим задовольняючи третій закон Ньютона.

Використовуючи загальну формулу інтерполяційного ядра та отримане рівняння для сили тиску, отримаємо відповідне інтерполяційне ядро, що описує взаємодію між частинками, та градієнт

$$W_{тиск}\left(\mathbf{r}, h\right) = \frac{15}{\pi h^6} \begin{cases} \left(h - \|\mathbf{r}\|\right)^3, & 0 \le \|\mathbf{r}\| \le h, \\ 0, & \|\mathbf{r}\| > h, \end{cases}$$

$$\nabla W_{тиск}\left(\mathbf{r}, h\right) = -\frac{45}{\pi h^6} \frac{\mathbf{r}}{\|\mathbf{r}\|} \left(h - \|\mathbf{r}\|\right)^2,$$

$$\lim_{\mathbf{r}\to -0} \nabla W_{тиск}\left(\mathbf{r},h\right) = \frac{45}{\pi h^6}, \quad \lim_{\mathbf{r}\to +0} \nabla W_{тиск}\left(\mathbf{r},h\right) = -\frac{45}{\pi h^6}.$$

Загальна формула для сили в’язкості у термінах SPH має вигляд

$$\mathbf{f}_i^{в'язк.} = \mu \sum_{j\neq i} \mathbf{u}_j \frac{\mathrm{m}_j}{\rho_i} \nabla^2 W\left(\mathbf{r}_i - \mathbf{r}_j, h\right).$$

Аналогічно силі тиску, сила в’язкості також не симетрична відносно зміни швидкості від частинки до частинки. В протидію цьому було використано наступне

$$\mathbf{f}_i^{в'язк.} = \mu \sum_{j\neq i} \left(\mathbf{u}_j - \mathbf{u}_i\right) \frac{\mathrm{m}_j}{\rho_i} \nabla^2 W\left(\mathbf{r}_i - \mathbf{r}_j, h\right),$$

де сила в’язкості залежить лише від різниць швидкостей, а не від їх абсолютних значень. Відповідне інтерполяційне ядро та градієнт набувають вигляду:

$$W_{в'язк.}\left(\mathbf{r},h\right) = \frac{15}{2\pi h^3} \begin{cases} -\frac{\|\mathbf{r}\|^3}{2h^3} + \frac{\|\mathbf{r}\|^2}{h^2} + \frac{h}{2\|\mathbf{r}\|} - 1, & 0 \le \|\mathbf{r}\| \le h, \\ 0, & \|\mathbf{r}\| > h, \end{cases}$$

$$\lim_{\mathbf{r}\to 0} W_{в'язк.}\left(\mathbf{r},h\right) = \infty,$$

$$\nabla W_{в'язк.}\left(\mathbf{r},h\right) = \frac{15}{2\pi h^3} \mathbf{r} \left(-\frac{3\|\mathbf{r}\|}{2h^3} + \frac{2}{h^2} + \frac{h}{2\|\mathbf{r}\|^3} \right),$$

$$\lim_{\mathbf{r}\to -0} \nabla W_{в'язк.}\left(\mathbf{r},h\right) = +\infty, \quad \lim_{\mathbf{r}\to +0} \nabla W_{в'язк.}\left(\mathbf{r},h\right) = -\infty.$$

Оскільки зовнішні сили складаються лише із сили гравітації та її формулювання тривіально виражається через другий закон Ньютона

$$\mathbf{f}_i^{тяж.} = \rho_i \mathrm{g},$$

то залишається одна сила – сила поверхневого натягу, яка для певної множини частинок є додатковою зовнішньою силою. Вона не є частиною рівняння Нав'є-Стокса, бо являється граничною умовою. У підході Лагранжа межа представляється множиною частинок.

На молекули рідин впливають сили сусідніх молекул, що знаходяться у ідеальному балансі усередині рідини. На поверхні ж сили розбалансовані – це і спричиняє поверхневий натяг. Сила поверхневого натягу співнапрямлена із поверхневими нормалями відносно рідини, де вони зв'язують її поверхню разом. Сила поверхневого натягу розпрямлює поверхневу кривизну, мінімізуючи площу поверхні. Модель опису поверхневого натягу описується, як

$$\mathbf{f}_i^{пов.} = -\sigma \nabla^2 c_i \frac{\mathbf{n}_i}{\|\mathbf{n}_i\|},$$

де $\mathbf{n}_i$ – напрямлена усередину поверхнева нормаль рідини i-ої частинки;

c_i – згладжена величина кольору частинки;

σ – коефіцієнт тертя.

Згладжена величина кольору частинки є додатковою величиною у SPH і задається у наступному вигляді

$$c_i = \sum_j \frac{\mathrm{m}_j}{\rho_i} W\left(\mathbf{r}_i - \mathbf{r}_j, h\right).$$

Основною властивістю цієї величини є те, що її градієнт є поверхневою нормаллю

$$\mathbf{n}_i = \nabla c_i = \sum_j \frac{\mathrm{m}_j}{\rho_i} \nabla W\left(\mathbf{r}_i - \mathbf{r}_j, h\right),$$

де $\|\mathbf{n}_i\| > 0$ тільки біля або на поверхні рідини. Дивергенція нормалі вимірює

Гаусову кривизну поверхні

$$\kappa = -\frac{\nabla \mathbf{n}}{\|\mathbf{n}\|} = -\frac{\nabla^2 c}{\|\mathbf{n}\|},$$

від'ємність якої необхідна для отримання позитивного об'єму випуклої кривизни. Поверхнева сила зціплення є силою, що діє на одиницю площі даної поверхні рідини і розподіляється по навколишніх частинках

$$\mathbf{t} = \sigma\kappa \frac{\mathbf{n}}{\|\mathbf{n}\|}.$$

Оскільки $\|\mathbf{n}_i\|$ стає зменшується із віддаленням i-ої частинки від поверхні, можемо помножити поверхневе зціплення на нормалізовану скалярну величину $\delta_i = \|\mathbf{n}_i\|$. За рахунок цього впевнимося, що сила густини розподілена по усіх можливих частинках. З огляду на це, виведемо наступну рівність

$$\mathbf{f}_i^{пов.} = \delta_i \mathbf{t}_i = \sigma\kappa_i \mathbf{n}_i = -\sigma\nabla^2 c_i \frac{\mathbf{n}_i}{\|\mathbf{n}_i\|}.$$

Сила поверхневого натягу може бути застосована тільки до часток, що знаходяться біля або на поверхні рідини. Це обмеження, визначене чисельно через відношення поверхневої нормалі до її норми, стає нестабільним при наближенні норми до нуля. Єдиний спосіб уникнення чисельних проблем є введення певного граничного значення $\ell > 0$ концентрації частинок

$$\|\mathbf{n}_i\| \geq \ell.$$

Сила поверхневого натягу асиметрична за своїм задумом. У реальності ця сила є послідовністю взаємодії між молекулами води та повітря. Відсутність

молекул повітря у моделювання не дає змоги зробити силу поверхневого натягу симетричною, через що і сама модель стає дещо віддаленою від дійсності.

Метод SPH є потужним інструментом, який зменшує складність математичних рівнянь потоків рідини, але відсутність нестисливості є одним з основних недоліків методу. Оскільки він може бути отриманий з Лагранжіану і має властивості збереження Лаґранжевої системи, вирішуються труднощі, пов’язані з відсутністю симетрії великих пустот, набагато ефективніше, ніж Ейлерові методи. В результаті природним чином слідують збереження імпульсу і енергії разом з наближеним інваріантом циркуляції.

Однак SPH також зберігає структуру, тобто кожна частинка несе свій склад незмінним, якщо матеріал, який представляє частинка, не піддається хімічним перетворенням. Ще одна особливість методу полягає в тому, динамічний діапазон просторової роздільної здатності регулюється плавно зі зміною щільності. У той же час цей метод має властивості збереження не тільки енергії та імпульсу, але й кутового моменту. Також SPH є інваріантним та нечутливим до будь-яких похибок адвекції.

Немає ніяких обмежень, що накладаються ні на геометрію системи, ні на те, наскільки вона може еволюціонувати від початкових умов, так що початкові умови можуть бути легко запрограмовані без необхідності складних алгоритмів сітки, що використовуються в методах кінцевих елементів.

Проте основним недоліком даного методу є його обмежена точність у багатовимірних потоках. Одне з джерел шуму породжується у апроксимаціях локальних інтерполянтів ядра через дискретні суми ближніх сусідів. Рух часток у багатовимірних просторах має значну більшу ступінь свободи, тому взаємновідштовхуючих сил високого тиску між сусідніми парами частинок не легко позбутися одночасно у всіх просторах.

Зокрема проблемою методу гідродинаміки згладжених частинок є те що метод інтерполяції, дуже простий, і на нього сильно вплине розлад частинок. SPH дає розумні результати для градієнтів першого порядку, але вони можуть бути гірше для похідних більш високого порядку.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Особливості програмної реалізації на мові С++

С++ – це компільована мова програмування високого рівня, розроблена Бьярном Страуступом як розширення мови С [17]. Серед сучасних вендорів, що пропонують С++ компілятори є такі, як Free Software Foundation (g++) [11], LLVM (clang) [12], Microsoft (cl), Intel, Oracle, IBM ті інші.

Мова С++ є стандартизованою Міжнародною Організацією зі Стандартизації (ISO). Класичні стандарти С++98 та С++03 поширені навіть на поточний день. Вони містили багато відомих методів, таких як класи, посилання, шаблони (дженерики), inline assembly, багатонитковість та інші. Але починаючи з стандарту С++11 у мову С++ додались засоби, що повністю змінили більшість підходів до написання коду.

Серед найбільш важливих – додання rvalue семантик, що дозволяють уникнути зайвого копіювання даних. Раніше ця проблема вирішувалася за допомогою CoW (copy on write), тобто доти у змінну явно не запишуть інформацію, копія не буде зроблена. Це дозволяло іноді уникати зайвого копіювання великих даних, таких, як std::string. Гарним прикладом буде фреймворк Qt, п'ята версія якого повністю побудована на принципі CoW (С++03). У шостій версії вони відійдуть від цього принципу та напишуть код за допомогою сучасних стандартів [5].

Іншим важливим аспектом С++11 було додання variadic templates. Це давало багато можливостей для оптимізації, наприклад, деякі класи компілюються лише у кінцевого користувача, що приближує швидкість багатьох засобів стандартної бібліотеки шаблонів (Standard Template Library, STL) до мови С. Грубо кажучи, можна уникнути багатьох викликів допоміжних функцій та у час компіляції генерується одна функція, максимально наближена до С.

У коді програми, що наведена у додатку А, побудована загальна структура. Вона передбачає шаблонний тип даних, шаблонний розмір простору та клас

для тестування цих алгоритмів.

Клас TestNNS використовується для тестування різних алгоритмів та у багатьох випадках він може бути відсутнім. Клас NNSInstance представляє собою чорний ящик, на вхід якого подаються наступні елементи:

а) граничні умови;

б) відстань пошуку сусідів;

в) набір частинок класу DataEntry.

На виході цей клас повинен обчислити сусідів для кожного елементу DataEntry. Сам клас може перевіряти якість обчислень за допомогою функції compare(). Вона приймає в себе «ідеальний» розв'язок задачі пошуку сусідів та обчислює надійність алгоритму за наступною формулою:

$$100\% \cdot \frac{found}{requested \cdot missed},$$

де $found$ – кількість частинок, що були знайдені правильно;

$requested$ – кількість частинок, що було потрібно знайти;

$missed$ – кількість неправильно обраних частинок.

Таким чином, якщо ми вірно обчислили 5 частинок із 20, але 30 знайшли невірно – ефективність не буде від'ємною.

3.2 Порівняння програмної реалізації алгоритмів

Будемо розглядати наступний алгоритми пошуку найближчих сусідів:

а) наївний алгоритм;

б) боксовий алгоритм;

в) алгоритм просторового хешування.

Наївний алгоритм не буде містити ніяких модифікацій, боксовий матиме наступні:

а) припущення щодо сусідства частинок з одного боксу;

б) припущення щодо сусідства усіх частинок;

в) зменшений у $\sqrt{2}$ радіус пошуку.

Для алгоритму просторового хешування будемо використовувати:

а) припущення щодо сусідства частинок з одного боксу;

б) припущення щодо сусідства усіх частинок;

в) зміна алгоритму хешування на запропонований NVIDIA.

Кожна з модифікацій накладається поверх існуючого алгоритму, що дозволяє використовувати за допомогою передачі бітів у останній змінні.

Наївний алгоритм є найпростішим у реалізації на займає лише 10 строчок коду.

Реалізація боксового алгоритму є найбільшою. Клас Box – це один конкретний бокс із цілочисловими координатами. Він містить у собі ці координати, вектор із DataEntry та булевий вектор, із мітками візиту. Ці мітки використовуються задля того, щоб уникнути повторних обчислень. Для спрощення арифметики простору існує клас BoxRelativePosition, який дозволяє проходити по усім сусідами. Тобто, для двовимірного простору, можливі сусіди будуть знаходитися за координатами $\{-1,-1\}$, $\{-1,0\}$, $\{-1,1\}$, $\{0,-1\}$, $\{0,1\}$, $\{1,-1\}$, $\{1,0\}$, $\{1,1\}$.

Задля модифікацій використовуються допоміжні constexpr функції, які приймають до себе змінну, за допомогою якої здійснюється маніпуляція:

а) get_range для вибору відстані;

б) boxed_process_central для вибору обчислень щодо центрального бокса;

в) boxed_process_outer для вибору обчислень щодо сусідніх боксів.

Клас Box агрегується у n-вимірний вектор у класі Boxes, яким у алгоритмі і маніпулюють.

Реалізація алгоритму просторового хешування значно простіша: використовується лише одна абстракція SHBox. Лише у цьому випадку для відстеження боксів що ми вже обробили використовується std::set із хешами.

Задля модифікацій використовуються допоміжні constexpr функції, які приймають до себе змінну, за допомогою якої здійснюється маніпуляція:

а) get_ calculate_hash для вибору хеш-функції;

б) boxed_process_central для вибору обчислень щодо центрального бокса;

в) boxed_process_outer для вибору обчислень щодо сусідніх боксів.

3.3 Оптимізація коду

3.3.1 Оптимізація за допомогою шаблонів

На рис. 3.1 наведена реалізація класу Point. У ньому добре розкривається могутність fold expressions та std::index_sequence із стандарту C++17.

```
template<typename _T, size_t _Dim, typename = std::make_index_sequence<_Dim>>
class Point;

template<typename _T, size_t _Dim, std::size_t ... _Idx>
class Point<_T, _Dim, std::index_sequence<_Idx ...>> : public std::array<_T, _Dim>
{
    public:
        bool within(const Point<_T, _Dim> & _other, const _T & _range) const
        {
            _T accumulated = ( std::pow(_other[_Idx] - (*this)[_Idx], 2) + ... );
            if ( accumulated > _range * _range )
                return false;
            return true;
        }
        bool operator ==(const Point<_T, _Dim, std::index_sequence<_Idx ...>> & _other) const
        {
            bool res = (true & ... & ((*this)[_Idx] == _other[_Idx]) );
            return res;
        }
};
```

Рисунок 3.1 – Реалізація класу Point

Це дозволяє під час компіляції для N-мірного простору зробити функції, що, наприклад, у випадку для булевої функції within(), що перевіряє дві частинки на сусідство, замість циклу обходу кожного вимірювання робити це за один раз. Тобто, для двовимірного класу Point<double, 2> строчка коду

```
_T accumulated = ( std::pow(_other[_Idx] - (*this)[_Idx], 2) + ... )
```

перетвориться на

```
_T accumulated = ( std::pow(_other[0] - (*this)[0], 2) +
                   std::pow(_other[1] - (*this)[1], 2)) ,
```

що значно прискорює обчислення. Порівняння швидкості обчислення наведені у розділі 4.

3.3.2 Оптимізація за допомогою багатонитковості

Тут вважаємо, що кількість ниток – 12. Це число може бути будь-яким, але для наочності будемо використовувати саме його.

Самий простий підхід – кожен із 12 ниток буде виконувати роботу обчислення найближчих сусідів для усіх частинок, номер яких співпадає з залишком по модулю 12. На перший погляд усе просто, але із початком додання багатонитковості з'являється проблема атомарності. Операція додання покажчика DataEntry до вектора сусідів – не атомарна. Проблема атомарності вирішується засобами синхронізації, тобто доданням N м'ютексів. Цей процес досить важкий з боку зору обчислень та не має сенсу для наших операцій.

Інший підхід – це зробити цю операцію атомарною, тобто гарантувати те, що ніякі два потоки не будуть додавати сусідів до одного елемента DataEntry. Це досить легко зробити, а саме замінити операцію взаємного додання сусідства (що попередньо прискорила процес обчислень майже удвічі). Тобто, якщо кількість ядер – два, то багатонитковість не має сенсу. Рекомендованою кількістю є 3 і більше потоків (логічних ядер).

4 РЕЗУЛЬТАТИ ОБЧИСЛЮВАЛЬНОГО ЕКСПЕРИМЕНТУ

Таким чином, за допомогою оптимізації алгоритму та оптимізації програмного коду отримано значний виграш у часі (у порівнянні з наївним алгоритмом). В експерименті використовувались 1000 частинок та тривимірна область $\Omega = \{0 \le x \le 50, 0 \le y \le 50, 0 \le z \le 50\}$.

На рис. А. 1 – А.3 наведено симуляцію в'язкої рідини у [13], написаного із використанням технології GLUT.

На рис. А.4 наведено часову статистику різних класичних алгоритмів (50 тестів, для $r = 8$). Вони дають 100% точність, тому графіки порівнянь точності відсутні.

На рис. А.5 – А.7 наведені графіки залежності часу виконання від радіусу пошуку. Чим ближче радіус пошуку до розмірів області, тим ближче за швидкістю боксовий алгоритм до алгоритму просторового хешування.

На рис. А.8 – А.11 наведені графіки часу виконання модифікацій боксового алгоритму, на рис А.12 – А.15 – точність (50 тестів, $r = 8$).

На рис. А.16, А.17 наведені графіки часу виконання модифікацій алгоритму просторового хешування, на рис А.18, А.19 – точність (50 тестів, $r = 8$).

Графік порівнянь середнього часу обчислення, середньої точності для різних радіусів пошуку наведені на рис. А.20 і А.21 для боксового алгоритму, та на рис А.22 і А.23 для алгоритму просторового хешування.

Також наведені графіки «точність помножена на час обчислення» для модифікацій боксового алгоритму – на рис А.24, для алгоритму просторового хешування – на рис А.25, що показують ефективність кожної модифікації у порівнянні із класичним.

Порівняння швидкості обчислень для різних хеш-функцій наведені на рис А.26. Функція, що була запропонована NVIDIA показала досить непогані результати, тому надалі будемо використовувати її.

Оскільки точність алгоритму не залежить від випадкових величин (на відміну від часу обчислення), то в усіх випадках функції точності лінійні.

Найкращим себе показав боксовий алгоритм, він добре показує себе в усіх випадках, окрім дуже малого радіусу пошуку, де алгоритм просторового хешування зменшує кількість боксів.

Варто відмітити, що якщо радіус пошуку буде достатньо великим, то боксова оптимізація зведеться до наївного пошуку і час обчислень буде майже у два рази більшим.

Різні модифікації показали себе дуже погано, тому не варто робити ніяких припущень і перевіряти кожну частинку. Завдяки тому, що із припущенням ми беремо дуже багато зайвих частинок, час обчислень навіть збільшується через багато записів до пам'яті.

У алгоритмі просторового хешування велике значення має хеш-функція. У деяких випадках він обчислював навіть швидше за боксовий, як показують результати обчислень за допомогою хеш-функції запропонованої NVIDIA.

На рис. А.38 наведено порівняння часу обчислень у [13] попереднього алгоритму пошуку найближчих сусідів та оптимізованого. Таким чином, обчислення було значно прискорено, що зменшує загальну кількість системних ресурсів, необхідних для симуляції.

Пояснимо, чому алгоритм просторового хешування працює довше ніж боксовий. У просторовому хешуванні, кожному боксу надається хеш, його унікальний ідентифікатор, та за допомогою std::unordered_map ми обчислюємо його позицію. Ця процедура значно довша, ніж просто обчислення координат бокса – вони і являються хешом у нашому випадку. Але адресація до боксу за лінійний час відбувається швидше ніж та сама, але із додатковим обчисленням хешу.

Фактично, алгоритм просторового хешування може вироджуватися у боксовий алгоритм при винятковій хеш-функції.

Програмні засоби оптимізації обчислювань надали значне прискорення, особливо багатонитковість. Неможливо уявити сучасну обчислювальну систему, яка працює лише у одному потоці.

5 АНАЛІЗ МОЖЛИВИХ ЗАСТОСУВАНЬ

Результати дипломної роботи можуть бути застосовані у сучасних системах моделювання гідродинаміки згладжених частинок та дають прискорення обчислювального процесу за допомогою оптимізації. Колізія об'єктів трапляється у багатьох комп'ютерних симуляціях, не тільки у моделюванні рідини, а, наприклад для виявлення колізії двох ігрових об'єктів у грі, чи скальпеля хірурга із пацієнтом.

Алгоритм просторового хешування також займає почесне місце у машинному навчанні, бо сусідство може бути визначено не тільки за допомогою евклідової метрики, а, наприклад, за допомогою схожості текстів за вмістом, що може застосовуватися у аналізах текстів чи постів у соціальних мережах.

Програмний продукт будує абстракції над простором та наведені алгоритми можуть бути використані для різних об'єктів.

Прийому оптимізації на мові C++ дають можливість створювати програмний продукт із великою кількістю абстракцій: Завдяки шаблонам ми можемо відійти від типу даних, розмірності простору, визначеності операторів і писати програмний код алгоритму для будь-яких об'єктів, у просторі яких задана метрика.

Реалізація багатониткового дає змогу використовувати різну кількість робочих потоків для обчислень, що робить оптимізованою як для двох ядерних процесорів, так і для 128-ядерних.

ВИСНОВКИ

У атестаційній роботі були розглянуті три основні алгоритми пошуку найближчих сусідів та можливі модифікації цих алгоритмів. Найважчим для обчислення є наївний алгоритм, а два інших – добре знаходять себе показали.

Різні запропоновані модифікації цих алгоритмів погано себе показали, тому у прикладній сфері вони не мають сенсу. Отже, два алгоритми, що показали найшвидші результати – класичний боксовий алгоритм та алгоритм просторового хешування із використанням хеш-функції NVIDIA.

Ці алгоритми вже успішно використовуються у відеоіграх, системах моделювання (медицина, аеродинаміка, гідродинаміка) тощо.

Розглянути були засоби програмної оптимізації, що дали значне прискорення обчислювального процесу. Існує багато інших засобів прискорити процес обчислення, але отримані результати доводять, що якісно побудована система складається не лише з добре побудованої математичної моделі, а й з якісного програмного продукту та обчислювальної техніки.

Програмний код написаний на мові C++, компільований за допомогою Microsoft Visual Compiler за стандартом C++17, що є сумісним із CUDA [10]. Тому у майбутньому цей процес обчислень можна буде перенести на відеопроцесор. Відеопроцесори досить часто використовуються у сучасних обчислювальних системах (наприклад, ферма криптовалют), тому внутрішня архітектура надає можливість прискорити деякі специфічні обчислення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Optimized Spatial Hashing for Collision Detection of Deformable Objects. Teschner M., Heidelberger B., Muller M., Pomeranets D. Computer Graphics Laboratory, ETH Zuric, 2003. 8 p.

2. Kelager M. Lagrangian fluid dynamics using smoothed particle hydrodynamics. University of Copenhagen, 2006. 88 p.

3. GPU gems 3. URL : https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch32.html (дата звернення: 05.11.2019).

4. Кнут Д. Е. Искусство программирования. Т. 3. Москва : М.: ООО "И.Д. Вильямс", 2013. 960 с.

5. Technical vision for Qt 6. URL : https://www.qt.io/blog/2019/08/07/technical-vision-qt-6 (дата звернення: 05.11.2019).

6. Do we really need GPU for Deep Learning? URL : https://mc.ai/do-we-really-need-gpu-for-deep-learning (дата звернення: 05.11.2019).

7. Significant speed gap between CUDA and OpenCL. URL : https://devtalk.nvidia.com/default/topic/1029272/significant-speed-gap-between-cuda-and-opencl-how-to-debug-/ (дата звернення: 05.11.2019).

8. Baciu G., Wong W., Sun W. RECODE: an image–based collision detection algorithm // The Journal of Visualization and Computer Animation, 1999. V. 10. P. 181–192.

9. Erin J. Hastings, Jaruwan Mesit, Ratan K. Guha. Optimization of Large-Scale, Real-Time Simulations by Spatial Hashing, College of Engineering and Computer Science. University of Central Florida, 2005. 9 p.

10. CUDA C++ Programming Guide. URL : https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#c-cplusplus-language-support (дата звернення: 05.11.2019).

11. The Free Software Foundation (FSF). URL : https://www.fsf.org/ (дата звернення: 05.11.2019).

12. The LLVM Compiler Infrastructure. URL : https://llvm.org/ (дата звер-

нення: 05.11.2019).

13. C++ implementation of Smoothed Particle Hydrodynamics. URL : https://github.com/aartiukh/sph-sdk (дата звернення: 05.11.2019).

14. The Linux Kernel Archives. URL : https://www.kernel.org/ (дата звернення: 05.11.2019).

15. C gcc versus C++ g++ fastest programs. URL : https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/c.html (дата звернення: 05.11.2019).

16. AMD EPYC™ 7742. URL : https://www.amd.com/en/products/cpu/amd-epyc-7742 (дата звернення: 05.11.2019).

17. The C++ Programming Language. URL : http://www.stroustrup.com/C++.html (дата звернення: 05.11.2019).

18. Recent milestones: C++17 published, C++20 underway. URL : https://isocpp.org/std/status (дата звернення: 05.11.2019).