Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Інформаційна технологія візуалізації даних із розрахункових сіток

Виконав: студент IV курсу, групи СТ-41

спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)

_____ Рибенчук В.С.
(підпис) (прізвище та ініціали)

Керівник _____ Матійчук Л.П.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Яцишин В.В.
(підпис) (прізвище та ініціали)

Тернопіль - 2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 126 Інформаційні системи та технології
(шифр і назва спеціальності)

Студенту Рибенчуку Володимиру Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інформаційна технологія візуалізації даних із розрахункових сіток

Керівник роботи Матійчук Любомир Павлович, д.т.н., доц.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «30» 03 2026 року № 4/9-162

2. Термін подання студентом завершеної роботи 21.06.2026 р.

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ
1. Аналіз предметної області.
2. Способи та методи вирішення поставленої задачі
3. Практична частина
4. Безпека життєдіяльності, основи охорони праці
Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титулка. 2. Мета, задачі дослідження. 3. Приклади розрахункових сіток.
4. Використовувані технології. 5. Приклад шейдерів. 6. Приклади відображення сіткових даних. 7. Приклад матеріалу, налаштованого у редакторі вузлів
8. Налаштування сцени. 9. Результати простого відображення.
10. Відображення з параметрами та кольором. 11. Фільтрація сітки за індексами вузлів.
12. Висновки. Основні результати проведеного дослідження.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	26.01..26	*Виконано*
2	Підбір джерел про візуалізацію даних із розрахункових		
	сіток	27.01 – 16.02.26	*Виконано*
3.	Опрацювання джерел про візуалізацію даних із		
	розрахункових сіток	17.02 – 02.05.26	*Виконано*
4.	Виконання дослідження щодо розробки застосунку	03.05 – 10.05.26	*Виконано*
5	Написання програмного коду	11.05 – 18.05.26	*Виконано*
6.	Оформлення розділу «Аналіз предметної області»	19.05 – 22.05.26	*Виконано*
7.	Оформлення розділу «Способи та методи вирішення		
	поставленої задачі»	23.05 – 26.05.26	*Виконано*
8.	Оформлення розділу «Практична частина»	27.05 – 30.05.26	*Виконано*
9.	Виконання завдання до підрозділу «Безпека		
	життєдіяльності, основи хорони праці»	31.05 – 03.06.26	*Виконано*
10.	Оформлення кваліфікаційної роботи	17.05 – 03.06.26	*Виконано*
11.	Нормоконтроль	01.06 – 05.06.26	*Виконано*
12.	Перевірка на плагіат	04.06 – 10.06.26	*Виконано*
13.	Попередній захист кваліфікаційної роботи	11.06 – 18.06.26	*Виконано*
14.	Захист кваліфікаційної роботи	22.06.26	

Студент _____ Рибенчук В.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Матійчук Л.П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Інформаційна технологія візуалізації даних із розрахункових сіток // Рибенчук Володимир Сергійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем та програмної інженерії, кафедра комп’ютерних наук, група СТ–41 // Тернопіль, 2026 // С. – 51, рис. – 22, табл. – 1, слайдів – 12, бібліогр. – 27.

Ключові слова: GLSL, OpenGL, візуалізація даних, редактор вузлів, розрахункова сітка, шейдер

Розроблений графічний пакет є окремим застосунком для персональних комп'ютерів, який реалізований мовою програмування C ++ із застосуванням бібліотеки OpenGL для виведення графічних даних на екран. Для додаткової функціональності також використано низку додаткових бібліотек, що знаходяться у відкритому доступі.

Реалізовані експерименти ґрунтуються на можливості в реальному часі перезавантажувати шейдери – програми, що виконуються на графічних прискорювачах. Вивчаються різні підходи до визначення нових шейдерів та взаємодії з ними через інтерфейс користувача. Зокрема, на одному з етапів роботи були адаптовані результати, отримані при розробці багатьох популярних програм для 3D - моделювання та розробки комп'ютерних ігор. Йдеться про концепцію матеріалів і доповнює її концепцію редактора вузлів, що стало основним результатом роботи.

Розглянуто поетапний процес наближення до результатів роботи та їх технічна реалізація. Описано роботу редактора вузлів, наведено приклади візуальних відображень. Застосунок містить вимоги для компіляції програми.

ANNOTATION

Information Technology for Visualization of Data from Computational Grids // Rybenchuk Volodymyr // Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science // Ternopil, 2026 // P. - 51, Fig. - 22, Table - 1, Slide - 12, References - 27.

Keywords: GLSL, OpenGL, data visualization, node editor, computational mesh, shader

The developed graphics package is a separate application for personal computers, which is implemented in the C ++ programming language using the OpenGL library to display graphic data on the screen. For additional functionality, a number of additional libraries that are in the public domain were also used.

The implemented experiments are based on the ability to reload shaders in real time - programs that run on graphics accelerators. Various approaches to defining new shaders and interacting with them through the user interface are studied. In particular, at one of the stages of the work, the results obtained during the development of many popular programs for 3D modeling and computer game development were adapted. This is about the concept of materials and the concept of the node editor that complements it, which became the main result of the work.

The phased process of approaching the results of the work and their technical implementation are considered. The work of the node editor is described, examples of visual displays are given. The application contains requirements for compiling the program.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ

GLSL (англ. OpenGL Shading Language) – мова програмування високого рівня для написання шейдерів, які керують рендерингом графіки, виконуючись безпосередньо на графічному процесорі (GPU) для створення складних візуальних ефектів, налаштування освітлення, текстурування та анімації в іграх і застосунках.

GPU (англ. Graphics Processing Unit) – графічний процесор, окремий пристрій персонального комп'ютера або ігрової приставки, виконує графічний рендеринг.

OpenGL (англ. Open Graphics Library) — відкрита графічна бібліотека), специфікація, що визначає незалежний від мови програмування крос-платформовий програмний інтерфейс (API) для написання застосунків, що використовують 2D та 3D комп'ютерну графіку.

ПЗ – програмне забезпечення.

Розрахункова сітка - дискретний набір точок (вузлів), що покривають область розв'язку (наприклад, квадрат, об'єм), замінюючи неперервну область на скінченну множину точок.

Шейдер (англ. shader) — це невелика програма для графічного процесора (GPU), яка визначає, як об'єкти та пікселі відображаються на екрані, керуючи світлом, тінями, текстурами, кольором та іншими візуальними ефектами, роблячи графіку реалістичнішою та динамічнішою в іграх та 3D-графіці.

ЗМІСТ

ВСТУП

Актуальність теми. Розрахункові сітки застосовуються для апроксимації різних об'єктів у рамках розв'язання рівнянь математичної фізики методом кінцевих різниць. Результати знаходять активне застосування у практичних інженерних завданнях.

Разом із тим порушується питання візуалізації та інтерпретації одержуваних даних. Якісне спеціалізоване рішення для візуалізації сіткових даних необхідне для забезпечення наочної та зручної роботи дослідників, в т.ч. і прикладних математиків.

Існує ряд програмних пакетів, які частково вирішують поставлене завдання візуалізації. Але вони не задовольняють цілком особливим вимогам вчених із різних причин, які зрештою зводяться до неможливості гнучко розширювати можливості пакета під власні потреби.

Необхідно втілити алгоритм генерації вихідного коду шейдерів на базі напрямлених ациклічних графів, котрі візуально налаштовуються. Розробка не повинна мати значних проблем із продуктивністю за наявності пропонованих об’ємів даних. Необхідність оптимізації власне може виникнути лише при створенні дійсно складних і не шаблонних візуальних відображень за досить великої кількості даних.

Мета роботи – створення інформаційної технології та реалізація графічного пакета для візуалізації розрахункових сіток з можливістю додавання нових візуальних представлень, необхідних для користувача, без додаткового програмування. Мета досягається в рамках **виконання ряду завдань**, що реалізуються у вигляді експериментів, спрямованих на додавання та вивчення можливостей різних концепцій і функцій:

– вивчити формат зберігання сіткових даних, з яким потрібно працювати;

– вибрати набір технологій для розробки;

– реалізувати базовий функціонал для відображення сіткових даних;

– провести ряд експериментів, пов'язаних із створенням можливості розширювати функціонал без зміни вихідного коду та повторної компіляції програми;

– визначити оптимальний варіант із випробуваних у ході експериментів і використовувати його як підсумковий;

– визначити та описати можливі напрями подальшого розвитку розробленої системи.

Практичне значення одержаних результатів. Розробка може бути успішно використана у кінематографі, комп'ютерних іграх, художній комп'ютерній графіці

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Загальні положення про розрахункові сітки

Розрахункова сітка - сукупність вузлів, заданих в області визначення деякої функції, що апроксимують область для застосування чисельних методів. У загальному вигляді побудова сітки означає побудову безперервного відображення з простої обчислювальної області на складнішу фізичну. Приклади розрахункових сіток можна побачити на рисунках 1.1 та 1.2.

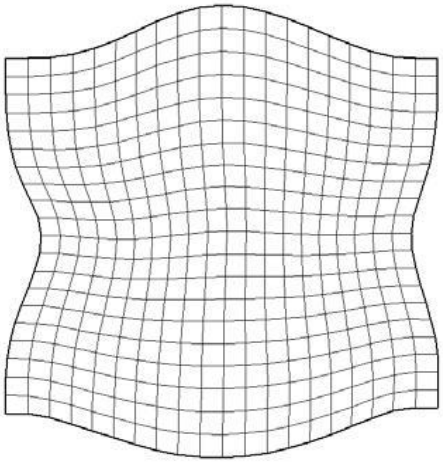

Рисунок 1.1 – Приклад двовимірної розрахункової сітки

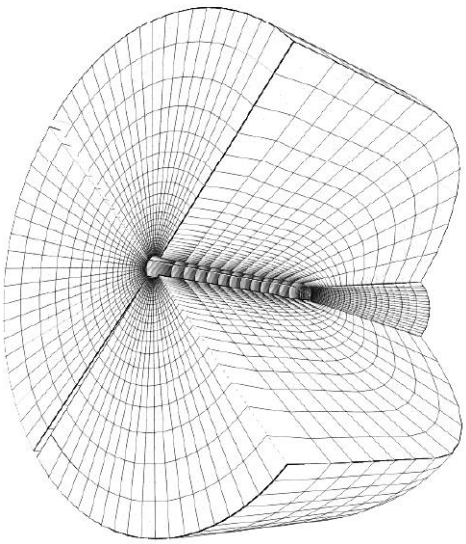

Рисунок 1.2 – Приклад тривимірної розрахункової сітки

Для вирішення диференціальних та інтегральних рівнянь найчастіше застосовуються чисельні методи, у тому числі метод кінцевих різниць, який полягає у заміні похідних різницевими схемами. Перевага - висока продуктивність, що досягається за рахунок паралельних обчислень.

Серед усіх сіток виділяють неструктуровані та структуровані. Умова структурованості сітки - існування відношення порядку на множині вузлів, що задається кусково-гладким відображенням параметричного простору з цілими координатами вузлів у фізичний простір.

Також розрахункові сітки поділяють на вироджені та невироджені. Невиродженість сітки – найважливіший критерій, який доводиться враховувати у дослідженнях. Сітка є невиродженою, якщо відображення обчислювальної сфери у фізичну — бієкція. Ця умова виключає наявність самоперетинів, зазорів та виходів за межі розрахункової області у сітці у фізичній галузі.

Невиродженість сітки залежить від значень якобіана застосовуваного відображення у вузлах сітки. Якобіан обчислюється як визначник матриці часткових похідних відображення. Непозитивні значення якобіана дозволяють говорити про виродженість сітки [1].

1.2 Аналітичний огляд

1.2.1 Попередні результати в галузі візуалізації сіток

В рамках роботи [2] було розроблено систему візуалізації сіткових даних, котра орієнтована в основному на наукових співробітників.

У роботі вивчено перспективи використання веб-середовища як основу системи. Розроблена програма запускається у браузері та дозволяє завантажувати дані з віддаленого сервера. Графічна візуалізація забезпечена за допомогою бібліотеки WebGL.

Програма є модулем для системи, котра визначає особливості організації сцени та реалізації деяких алгоритмів, що застосовуються у розробленій програмі [3].

Також було вивчено питання реалізації підтримки віртуальної реальності

та управління жестами, спрямованих на занурення користувача в абстрактне середовище. Згадано важливість звернення уваги на психофізіологічний фактор віртуальної реальності, забезпечення комфортного сприйняття користувачем моделюється середовища [4].

Набір реалізованих функцій включає розбір сіткових даних у наданому форматі, фільтрацію, розріджування і вибірку груп осередків за індексами. Є можливість експорту ілюстрацій.

Відмінність даної роботи від перерахованих результатів полягає у вивченні способів забезпечення гнучкості та розширюваності набору функцій програми засобами користувача

1.2.2 Актуальність візуалізації розрахункових сіток та аналіз окремих аналогів

Завдання, що вирішуються із застосуванням методу кінцевих різниць та розрахункових сіток, регулярно виникають у ході проведення прикладних досліджень у галузі математичної фізики. Розрахункові сітки у цьому випадку є апроксимацію деякого реального фізичного об'єкта.

У ході роботи формується набір даних, що включає опис структури сітки, розташування всіх її вузлів, а також опціональну додаткову інформацію про властивості об'єкта. Можливість візуального відображення та інтерпретації отриманих даних є вкрай важливою для дослідників. Візуалізація повинна дозволяти науковцям зрозуміти, переглянути та отримати представлення про дані.

Існує ряд комерційних та некомерційних графічних пакетів, у функціональність яких включена можливість відображення та опрацювання сіткових даних.

Серед них можна виділити системи Tecplot [5], ParaView [6] та OpenFOAM [7], кожна з яких має дуже широкий набір функцій. Тим не менш, ці системи мають проблеми, які не дозволяють отримати необхідні математикам університету візуальні подання даних.

Tecplot – комерційний продукт, вихідний код якого не поширюється у

вільному доступі, тому користувачі обмежені реалізованими функціями. Приклад візуалізації у цьому середовищі наведено на рисунку 1.3.

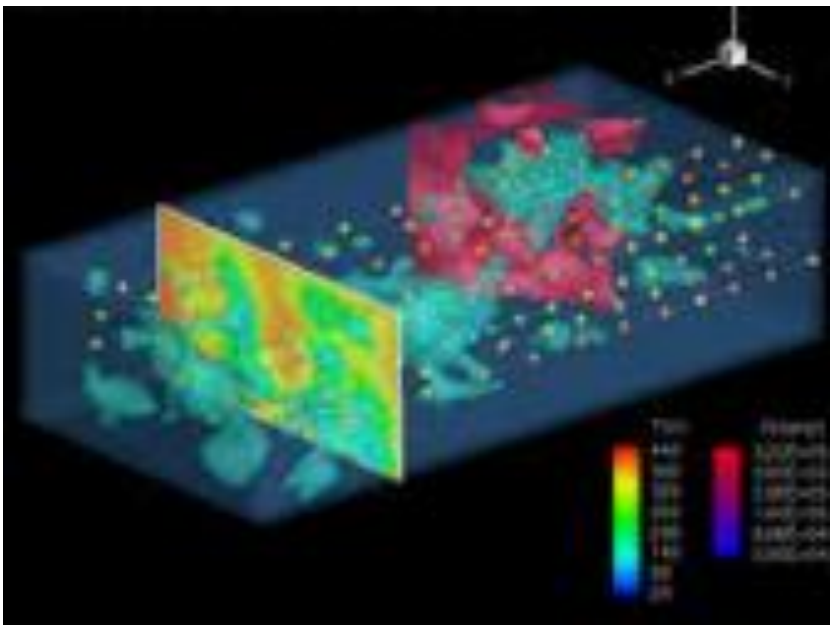

Рисунок 1.3 – Приклад відображення сіткових даних у Tecplot

ParaView – складний та універсальний інструмент загального призначення, недостатньо зручний для роботи з конкретними даними. Приклад візуалізації у ParaView показано на рисунку 1.4.

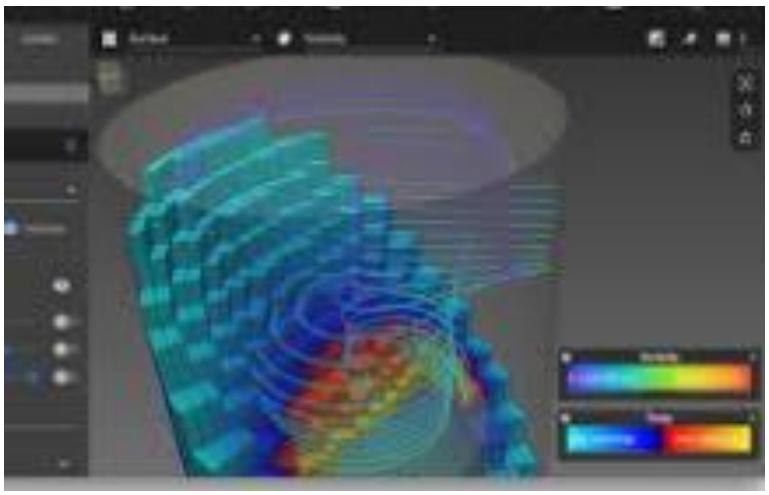

Рисунок 1.4 – Приклад відображення сіткових даних у ParaView

OpenFOAM - складний інструмент для аналізу і візуалізації великих даних (як правило механіки суцільного середовища). Досить часто використовується

спільно з ParaView для постобробки даних. На рисунку 1.5 показано приклад візуалізації сітки даних у програмному середовищі OpenFOAM.

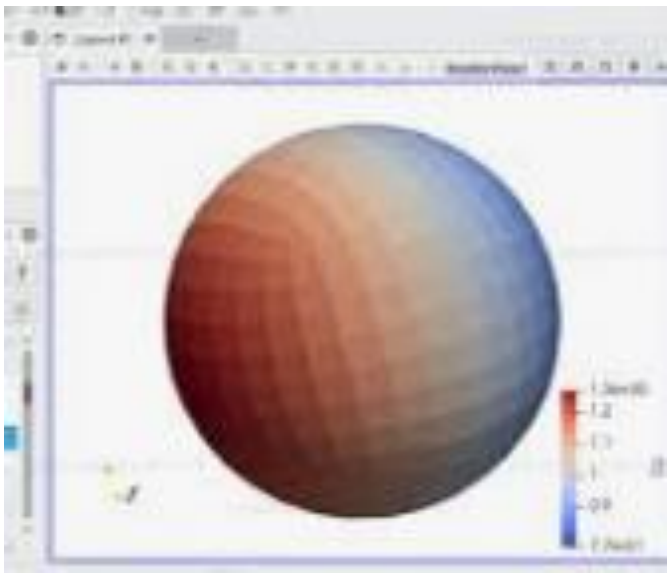

Рисунок 1.5 – Приклад відображення сіткових даних у OpenFOAM

З описаної вище ситуації можна зробити висновок, що візуалізацію розрахункових сіток не можна вважати вирішеним завданням і на ринку ПЗ є простір нових продуктів, що уникають проблем згаданих програмних пакетів

Визначення нових візуальних представлень за допомогою візуального редактора робить поріг входу для нових користувачів досить низьким, дозволяє сконцентруватися на вирішенні своїх завдань.

1.3 Аналіз потреб користувачів

Потреби користувачів не обмежуються застосуванням вже існуючих функцій у доступних графічних пакетах. Загалом можна сформулювати наступну глобальну проблему: у користувача немає можливості реалізувати нове візуальне представлення, або реалізація виявиться надто трудомісткою.

Реалізації нових візуальних уявлень можуть завадити такі проблеми:

– вихідний код програмного продукту не знаходиться у відкритому доступі;

– реалізація нових функцій потребує знання системних мов програмування, програмної інженерії та інших спеціальних знань ;

– робота з програмним продуктом складна і незручна сама по собі, що робить його розширення недоцільним у довгостроковому плані;

– розробка нових функцій займає багато часу та відволікає від вирішення безпосередніх завдань користувача;

– відсутня можливість повторного використання вже одного разу реалізованої функції; не можна комбінувати існуючі функції.

Виходячи з наведених факторів, можна сформулювати вимоги до нового графічного пакету, що вирішує зазначену проблему. Замість створення системи, що володіє певним набором можливостей, слід розробити платформу, що розширюється засобами користувача, не вимагає для цього додаткового програмування самої системи.

1.4 Організація робочого процесу

Процес розробки програмного продукту планується побудувати на основі гнучкої методології розробки. Робота розбита на відносно короткі ітерації, на початку кожної з яких формуються нові вимоги до програми, а до кінця ці вимоги мають бути реалізовані. У межах кожної ітерації коригується курс розвитку проекту з урахуванням останніх результатів. Таким чином, сама програма та функціональні вимоги до неї можуть помінятися радикально після будь-якої з ітерацій.

З поміж переваг такого власне підходу варто виокремити можливість уникати ситуації, коли велику кількість часу було витрачено на реалізацію функцій, які не вирішують поставлені завдання. Можливість гнучко змінювати напрямок розвитку проекту дозволяє зберігати зв'язок із реальними потребами користувачів.

Головним недоліком підходу можна назвати нехтування довгостроковим плануванням, яке може призводити до накопичення технічних недоліків, які в майбутньому можуть завадити внесенню нових змін.

У випадку програми, що обговорюється в рамках цієї роботи, цей підхід є допустимим і корисним, оскільки одним із поставлених завдань є проведення низки експериментів, які повинні допомогти визначити оптимальні способи досягнення мети роботи. Відсутність чіткого розуміння того, які функції необхідні програмі, не дозволяє розробити довгостроковий план розвитку, що виправдовує застосування гнучкої методології розробки.

Важлива частина роботи — отримання зворотний зв'язок кожної ітерації. У цій роботі зворотний зв'язок була організована в рамках проведення наукових досліджень на базі кафедри комп’ютерних наук, де були представлено поточний стан ПЗ, що розробляється, а ряд присутніх фахівців міг задати цікаві для них питання і запропонувати нові ідеї. Запропоновані ідеї стали основним джерелом прогресу, що дозволило на кожній ітерації наближатися до вирішення поставлених завдань.

РОЗДІЛ 2. СПОСОБИ ТА МЕТОДИ ВИРІШЕННЯ ПОСТАВЛЕНОЇ ЗАДАЧІ

2.1 Обговорення ідей, запропонованих на першій ітерації

На початку роботи було запропоновано такі ідеї для реалізації у спеціалізованому ПЗ, яке розробляється в рамках кваліфікаційної роботи:

– візуалізація загального уявлення про сітки на базі наявних даних;

– підтримка сіток, що містять більше мільйона комірок. Повинна зберігатися висока продуктивність;

– аналіз особливостей, їх виділення кольором чи формою;

– підтримка віртуальної реальності із застосуванням спеціалізованих гарнітур. Можлива реалізація таких видів відображення, як загальний обліт та рух над сіткою за просторовими координатами, занурення у сітку з деталізацією інформації про її начинку та виділенням окремих сіток, навігація по сітці за допомогою деяких орієнтирів, рух по сітці зі зміною напрямку та швидкості;

– підтримка за допомогою різних контролерів та природних жестових інтерфейсів.

Так як більша частина роботи проводилася на особистому устаткуванні автора в рамках дистанційного навчання (перебування за межами країни), не вдалося організувати доступ до необхідного обладнання для роботи з віртуальною реальністю. В результаті аналізу потреб користувачів було прийнято рішення сфокусуватися на завданнях, описаних у вступі, оскільки зазначені проблеми здалися найбільш значущими та фундаментальними для побудови системи. Реалізація відкладених завдань, як і раніше, можлива в міру необхідності та появи доступу до спеціалізованого обладнання.

2.2 Формат зберігання даних

Як зразки сіткових даних було надано набір файлів у текстовому форматі Tecplot ASCII [5]. У цьому наборі кожна сітка – результат перетворення

тривимірної прямокутної сітки. Комірки включають по вісім вузлів.

Кожен файл містить опис набору блоків, куди розбита розрахункова сітка. Опис блоку містить наступну іформацію:

– рядок із назвою даного блоку;

– рядок з перерахуванням змінних, що використовуються. Він включає тривимірні координати вузлів, а також будь-які інші значення, які дослідник може додати до даних;

– рядок із зазначенням розмірностей блоку;

– набір рядків, що містять значення для кожної із змінних. Кількість дорівнює добутку розмірностей блоку.

Вузлу відповідає унікальний набір індексів. Вузли, у яких лише один із індексів відрізняється на одиницю, є сусідніми та з'єднані дугами.

Приклад даних наведено у лістингу 2.1.

Лістинг 2.1 – Приклад даних

```
TITLE = "str_data"
VARIABLES = "X", "Y", "Z", "D", "P", "M", "S",
ZONE I=   26, J=   18, K=    9, F=POINT
-3.4740e-01 7.5072e-02 -6.8791e-02 5.1091e-01 4.1617e-01
2.4685e+00 1.0656e+00
-3.6450e-01 7.9131e-02 -7.2510e-02 6.0631e-01 5.2775e-01
2.3225e+00 1.0633e+00
-3.8108e-01 8.1687e-02 -7.4853e-02 1.0448e+00 1.1139e+00
1.8682e+00   1.0476e+00
-3.9749e-01 8.2730e-02 -7.5808e-02 1.0928e+00 1.1826e+00
1.8305e+00   1.0444e+00
...
```

Перед завантаженням даних було розглянуто існуючі рішення з відкритим вихідним кодом. У результаті було прийнято рішення написати свій найпростіший аналіз даних. Так як формат даних досить простий, це не зайняло багато часу, при цьому написаний код вийшов компактним і досить легким для внесення змін.

В даний момент щоразу при завантаженні сітки проводиться повний розбір текстового файлу. Для сіток щодо невеликих розмірів по 10-15 мегабайт час

завантаження може становити 4-5 секунд. На цьому етапі ці значення вважатимуться прийнятними. У майбутньому, якщо виникне потреба у завантаженні дуже великої кількості даних, можна реалізувати збереження завантажених уперше даних у бінарний формат, що дозволить у наступні запуски заощадити час на відкритті сітки.

2.3 Вибір технологій розробки

Для відображення графічних даних є спеціалізовані апаратні прискорювачі для відображення графіки – GPU [8]. Через обсяг сіткових даних потреба у застосуванні обчислень на GPU при розробці ПЗ очевидна. Робота з GPU має на увазі вибір деякої графічної бібліотеки, через яку організується спілкування між обчислювальним процесором CPU та GPU за принципом клієнт-сервер [9]. Як така графічна бібліотека була обрана OpenGL. Серед її переваг можна виділити підтримку більшості сучасних платформ та мов програмування та відкритий вихідний код.

Було ухвалено рішення про створення програми для персональних комп'ютерів. Як мову вибрано системну мову C++ [10]. Ця мова — найпоширеніша у сфері сучасної комп'ютерної графіки, для неї існує дуже докладна документація та написано безліч корисних допоміжних бібліотек. Також ця мова дозволяє досягти максимальної продуктивності, оскільки відсутні проміжні рівні виконання коду, такі як збирач сміття та інтерпретатор.

Для роботи з операційною системою вибрано популярну в ігровій індустрії бібліотеку Simple DirectMedia Layer (SDL) [11]. В її можливості входить підтримка різних платформ, створення вікна операційної системи, створення контексту для роботи з GPU за допомогою різних графічних бібліотек, обробка введення миші, клавіатури та інших контролерів.

Для створення інтерфейсів користувача вибрано бібліотеку Dear ImGUI, побудовану на технології Intermediate Mode Graphical User Interface [12]. Ця технологія полягає в імперативному підході до оголошення структури та функціоналу інтерфейсу користувача замість більш поширеного в даний момент

декларативного підходу. Бібліотека спеціалізується на застосуванні додатків з візуалізації даних. Принцип її роботи дозволяє легко додавати нові елементи під час розробки та динамічно змінювати стан інтерфейсу під час роботи програми. Приклад коду, який використовує бібліотеку, наведено у лістингу 2.2.

Лістинг 2.2 - Приклад коду із бібліотекою

```
ImGui::Text("Hello, world %d", 123); // Форматований напис
if (ImGui::Button("Save")) // Кнопка збереження
MySaveFunction(); // Функція збереження
ImGui::InputText("string", buf, IM_ARRAYSIZE(buf)); // Поле вводу
ImGui::SliderFloat("float", &f, 0.0f, 1.0f); // Слайдер
```

Результат виконання коду можна побачити на рисунку 2.1. Видно, що в результаті кожного виклику процедури з бібліотеки Dear ImGUI відображається деякий елемент інтерфейсу. Такий принцип роботи став важливим чинником, що дозволило провести майбутній ряд експериментів.

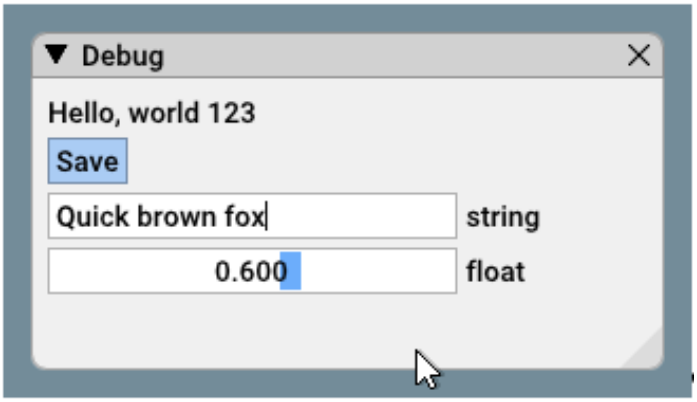

Рисунок 2.1 – Приклад інтерфейсу, створеного за допомогою бібліотеки Dear ImGUI

Розробка велася у середовищі розробки Visual Studio 2019 [13]. Для організації контролю версій було створено репозиторій (сховище вихідного коду та інших даних, що з проектом) з допомогою системи Git [14]. Про те, як отримати вихідний код і запустити його на власному комп'ютері, можна дізнатися у додатку Б.

Для деяких другорядних функцій було використано кілька додаткових

бібліотек. Для редактора вузлів, описаного в одному з наступних підрозділів, застосовано розширення для бібліотеки Dear ImGUI - imgui - node – editor [15]. Форматування рядків реалізовано за допомогою бібліотеки fmt [16]. Для збереження та завантаження структур даних використано бібліотеку ajson [17].

2.4 Базове відображення даних

Перед тим, як розпочати реалізацію експериментів, необхідно було реалізувати базову можливість відображати деяку сітку, представлену в описаному раніше форматі.

Коли сіткові дані завантажені з файлу, їх необхідно подати у відповідному для OpenGL вигляді. Для цього потрібно провести наступні дії::

1. Створити Vertex Array Object (VAO) [18]. Це внутрішня структура даних OpenGL , яка зберігає у собі зв'язки з іншими структурами, які відповідають різні частини процесу відображення даних. Таким чином, VAO допомагає краще організувати роботу з даними у програмі.

2. Для кожного блоку сітки створити Vertex Buffer Object (VBO) [18]. Ця структура зберігає безперервну ділянку пам'яті з довільними даними. У нашому випадку це координати вузлів, значення параметрів, індекси. Після створення VBO необхідно прив'язати до VAO.

3. Для кожного VBO створити Index Buffer Object (IBO) [18]. Ця структура містить список індексів елементів буфера з VBO . Вона вказує, як потрібно відображати вузли. Це дозволяє організувати відображення вузлів, з'єднаних ребрами, оскільки кожен вузол може бути приєднаний до кількох дуг. Для кожної дуги достатньо записати до списку IBO кілька індексів вузлів. IBO також необхідно прив'язати до VAO.

4. Налаштувати атрибути для VBO [18]. У цьому випадку атрибути — частини даних про сітку, серед них позиції вузлів та параметри, додані автором сітки. Налаштування включає опис структури області пам'яті, яку зберігає VBO . Необхідно вказати, де починається і закінчується кожен тип даних [19].

Після виконання описаних дій конфігурація даних для відображення

завершена. У циклі додаток необхідно зробити активним VAO , створений для завантаженої сітки та викликати спеціальну функцію для відображення графіки.

Але спочатку слід визначити, як відображаються надані дані. Для цього використовуються шейдери. Шейдер - програма, що виконується на GPU [3]. Існує різні типи шейдерів, які застосовуються на різних етапах відображення графіки. Кожен тип шейдера працює із певним типом даних.

У розробленій програмі використовуються два типи шейдерів: вершинний та піксельний. Вершинний шейдер визначає положення кожного вузла сітки. Піксельний шейдер визначає колір кожного пікселя екрану.

Для того, щоб застосувати певний набір шейдерів, необхідно завантажити тексти з файлів, скомпілювати засобами OpenGL і об'єднати в шейдерну програму. Тоді цю програму можна використовувати для відображення даних.

Написання шейдерів відбувається із застосуванням спеціалізованої мови програмування GLSL [19]. Приклад вершинного шейдера – лістинг 2.3.

Лістинг 2.3 - Приклад вершинного шейдера

```
layout ( location = 0) in vec 4 inPosition ; // Координати вузла
layout ( location = 1) in float вParam ; // Довільний параметр

out float param ; // Передаємо параметр у піксельний шейдер

uniform mat4 view; // Матриця виду
uniform mat4 проектування; // Матриця проекції

void main(void)
{
     param = inParam;
     gl _ PointSize = 2.0 f ; // Задаємо розмір точки
     // Визначаємо позицію щодо камери
     gl _ Position = projection * view * inPosition ;
}
```

Приклад піксельного шейдера показано у лістингу 2.4.

Лістинг 2.4 - Приклад піксельного шейдера

```
in float param ; // Переданий з вершинного шейдера параметр
out vec 4 outColor ; // Підсумковий колір пікселя
uniform vec 3 defaultColor ; // Настроюваний колір
                              // за замовчуванням

void main ( void )
{
    outColor = vec 4( defaultColor . r , defaultColor . g , de-
    faultColor . b , 1.0) * param ; // Задаємо колір залежно
                                    // від параметра
}
```

Візуальне подання розрахункової сітки можна побачити на рисунках 2.2 та 2.3. Були реалізовані функції відображення окремих вузлів у вигляді точок та ребер між вузлами у вигляді ліній.

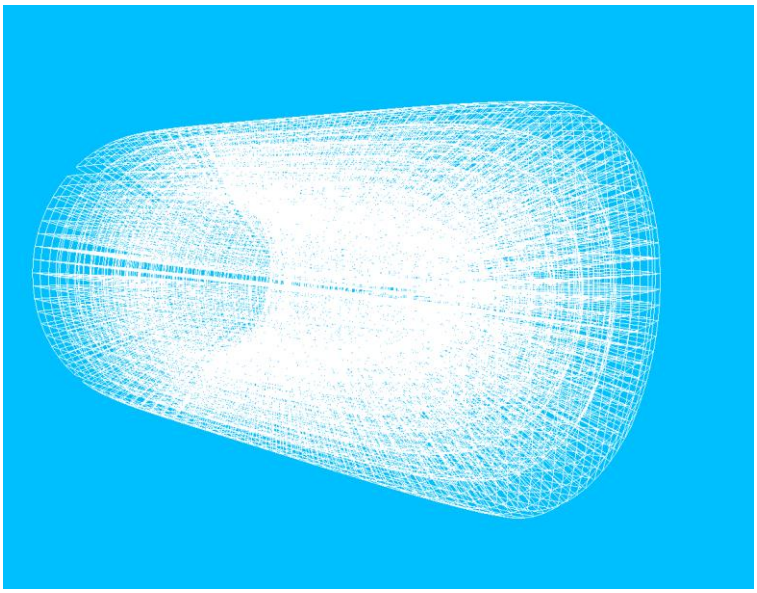

Рисунок 2.2 – Відображення сіткових даних як вузлів, з'єднаних дугами

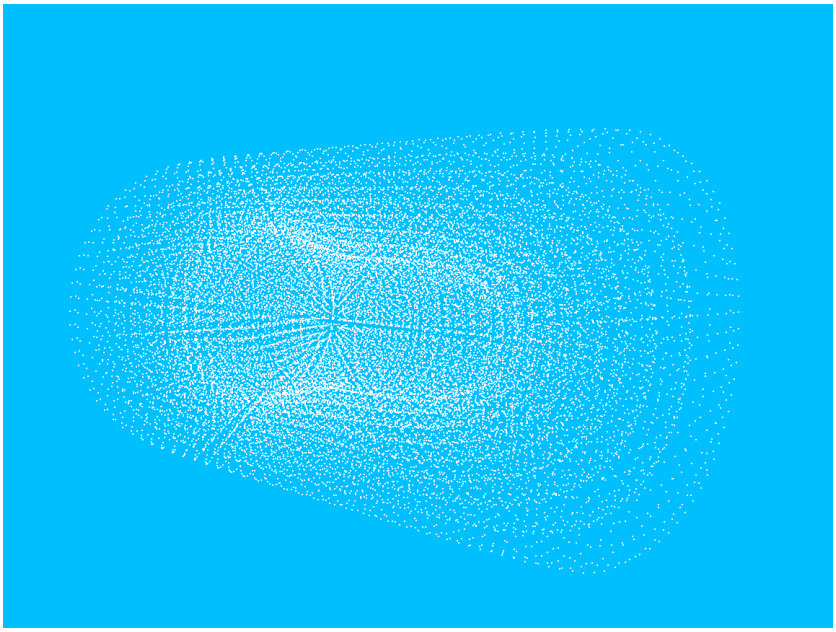

Рисунок 2.3 – Відображення сіткових даних як окремих вузлів

2.5 Керування камерою

Для вивчення сіткових даних з різних ракурсів потрібна можливість керувати положенням камери.

Було реалізовано два типи управління:

– вільна камера - за допомогою миші здійснюється поворот камери навколо власного центру по двох осях. За допомогою клавіатури можна переміщувати камеру у будь-якому напрямку. Цей тип керування підходить для переміщення всередині сітки [20];

– орбітальна камера - все керування здійснюється за допомогою миші. З допомогою руху миші камера повертається навколо заздалегідь обчисленого центру сітки. Позиція камери задається сферичними координатами, які потім перетворюються на декартові [20].

Логічний шлях розвитку системи управління переглядом сітки – підключення підтримки віртуальної реальності у комбінації із спеціальними контролерами. Тим не менш, корисно зберегти базовий підхід для користувачів без необхідного обладнання або мають проблеми зі сприйняттям віртуальної реальності.

2.6 Автоматичне перезавантаження шейдерів

Можливість, реалізована в першому з проведених експериментів, спочатку була спрямована лише на підвищення зручності розробки та не призначена для користувачів пакету. Але в результаті вона стала ключовою для всіх подальших експериментів.

Ця можливість – автоматичне перезавантаження шейдерів під час роботи програми.

Дуже важливим фактором для продуктивної роботи з програмою є швидкість, з якою можна побачити результат змін, що вносяться. Кожна зміна алгоритму візуального представлення матеріалу призводить до необхідності перекомпіляції шейдерів, застосовуваних у матеріалі. Ручна компіляція не дуже зручна і потребує постійного повторення одноманітних дій.

Для вирішення цієї проблеми було реалізовано динамічне підвантаження шейдерів. Воно включає наступні етапи:

– щосекундно проводиться перевірка останнього часу зміни файлів, що містять вихідний код шейдерів. Якщо час зміни виявився пізнішим, ніж остання компіляція шейдера, тоді необхідно повністю перекомпілювати шейдерну програму;

– за необхідності перекомпіляції завантажується вихідний код шейдерів з відповідних файлів, кожен шейдер компілюється окремо;

– окремі скомпіловані шейдери зв'язуються в одну шейдерну програму, яка завантажується на згадку про відеокарту;

– наново формується зв'язок сіткових даних та параметрів матеріалу з шейдерною програмою. З цього моменту можна побачити візуальний результат внесених змін.

Описана функція дозволила швидко намагатися реалізовувати різні візуальні подання даних, просто змінюючи вихідний код шейдера та зберігаючи файл. Виникло припущення, що подібний процес також міг би підійти і для кінцевих користувачів, оскільки зміна та додавання нових шейдерів не потребує перекомпіляції графічного пакета. Мова GLSL має набагато більш лаконічний і

простий синтаксис, ніж C ++, вивчити її можна досить швидко і вже через пару годин написати простий шейдер для відображення сітки.

2.7 Відображення розрахункових сіток за допомогою матеріалів

2.7.1 Концепція матеріалів

Описана вище можливість призвела до появи ідеї про те, як можна її узагальнити та зробити процес зручнішим для користувачів. Йдеться про концепцію матеріалів, яка зустрічається в багатьох програмах для роботи з 3D - графікою, не пов'язаних із областю наукової візуалізації. Матеріали активно використовуються у програмних пакетах, що застосовуються у кінематографі, комп'ютерних іграх, художній комп'ютерній графіці.

Матеріал є комбінацією з алгоритму, що визначає візуальне подання об'єкта, і структури даних, що зберігає параметри для зазначеного алгоритму. Алгоритм може бути заданий за допомогою класичної мови програмування, шейдера або візуального програмування. Значення параметрів матеріалу найчастіше задаються через інтерфейс користувача [19].

Реалізація концепції матеріалів включає наступні підзадачі:

– реалізувати базовий синтаксичний аналіз шейдерів визначення набору параметрів, що є у шейдері;

– реалізувати динамічне відображення параметрів шейдера в інтерфейсі користувача ;

– реалізувати передачу параметрів, що настроюються через інтерфейс, в пам'ять графічного прискорювача.

У повноцінно реалізованій системі матеріалів обов'язково є підтримка збереження та завантаження отриманих результатів у файли. Це дозволяє ділитися своєю роботою з іншими людьми і поступово формувати бібліотеку видів відображень, що часто використовуються. На етапі реалізації описуваного експерименту додавання цієї можливості було відкладено.

2.7.2 Організація параметрів матеріалу

Необхідно було визначити спосіб визначення набору параметрів матеріалу. Так як відображення шейдерів задається за допомогою шейдерних програм, логічно було припустити, що параметрами матеріалу повинні виступати змінні, що використовуються в шейдерах, якщо бути точніше, uniform -змінні. Uniform - змінні спеціально призначені для передачі в шейдери даних, які однакові для всієї розрахункової сітки, та застосовуються при обчисленні позиції кожного вузла або кольору кожного пікселя.

Щоб визначити, які параметри включені в шейдер, необхідно провести простий синтаксичний аналіз вихідного коду. Відомо, що оголошення всіх uniform -змінних починаються з ключового слова " uniform ", потім йде тип даних, опісля того - ім'я змінної. Рядок закінчується крапкою з комою.

У лістингу 2.5 наведено приклад декількох uniform -змінних, визначених в одному з шейдерів.

Лістинг 2.5 – Приклад uniform -змінних

```
uniform mat4 view;
uniform mat4 projection;
uniform mat4 model;

uniform float angleX;
uniform float angleY;
uniform float angleZ;

uniform vec4 time;
```

Серед типів даних змінних можна помітити матриці, вектори та числа з плаваючою комою. У програмі також реалізовано підтримку всіх типів даних, присутніх у шейдерній мові програмування GLSL . Варто окремо звернути увагу на такі змінні, як " view ", " projection ", " model " і " time ". Це зумовлені змінні, які не відображаються в інтерфейсі користувача і не можуть бути налаштовані користувачем, а безпосередньо обчислюються програмою і передаються в шейдер. Наприклад, змінні " projection ", " model " і " view " - матриці проекції, моделі та виду, які змінюються в залежності від розташування та характеристик

камери [21]. Це із огляду на те, що існує набір змінних, які використовуються дуже часто і потрібні в широкому наборі алгоритмів, при цьому потрібні особливі обчислення для отримання значень цих змінних.

2.7.3 Налаштування параметрів матеріалів у інтерфейсі користувача

Описані раніше можливості бібліотеки Dear ImGUI були корисні для налаштування параметрів в інтерфейсі.

На рисуноку 2.4 можна побачити параметри, відображені в інтерфейсі користувача на основі наведеного в попередньому підрозділі вихідного коду.

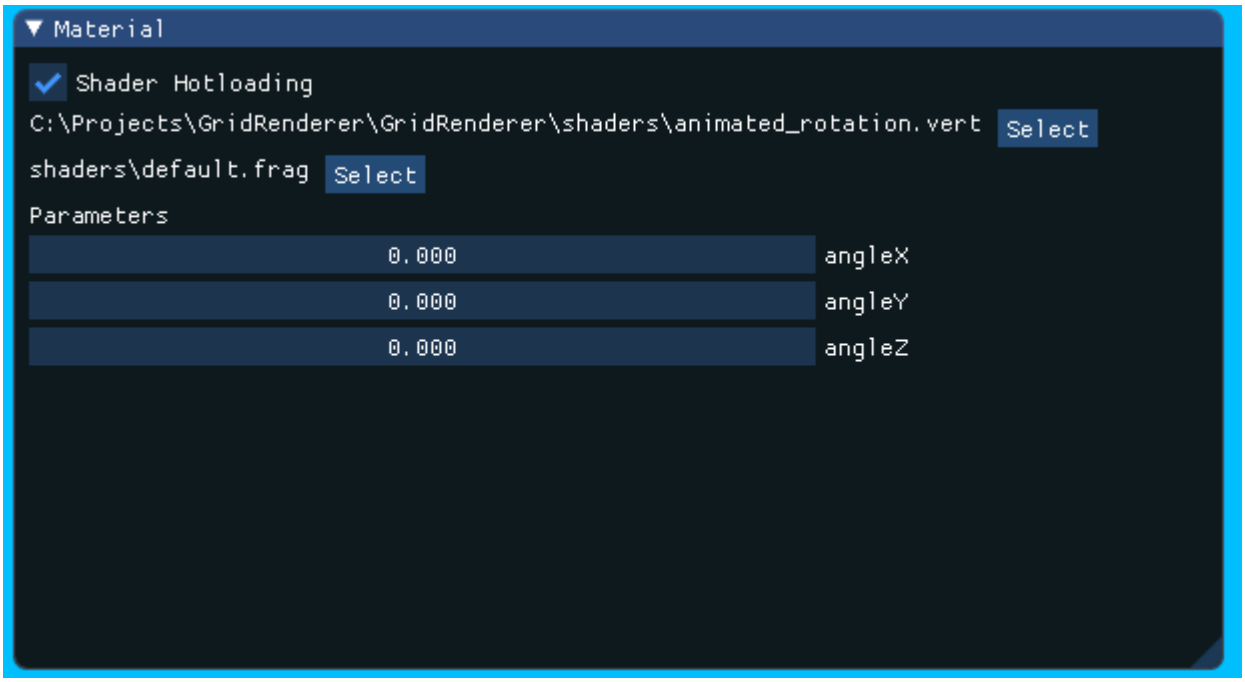

Рисунок 2.4 – Приклад відображення параметрів матеріалу в інтерфейсі користувача

Як і було описано, змінні " view ", " projection ", " model " і " time " не відображаються в інтерфейсі, так як це заздалегідь певні змінні, значення яких обчислюється динамічно.

У інтерфейсі користувача підтримується відображення всіх типів даних, присутніх у мові GLSL , включаючи:

– базові чисельні типи, включаючи цілі числа та числа з плаваючою комою різної точності;

– вектори розмірностей 2, 3 та 4;
– матриці з розмірностями 2, 3 та 4 у різних комбінаціях;
– логічний тип.

Також на рис. 3.4 можна побачити конфігурацію шляхів до файлів з вихідним кодом вершинного та піксельного шейдерів.

2.7.4 Приклад матеріалу

Для демонстрації роботи описаної схеми далі буде наведено приклад одного з тестових матеріалів.

Вихідний код вершинного шейдера наведено у лістигу 2.6.

Лістинг 2.6 - Вихідний код вершинного шейдера

```
#version 450 core

layout(location = 0) in vec4 inPosition;
layout(location = 1) in float inParam1;
layout(location = 2) in float inParam2;
layout(location = 3) in float inParam3;

out float param1;
out float param2;
out float param3;

uniform mat4 view;
uniform mat4 projection;
uniform mat4 model;

uniform vec3 offset;

void main(void)
{
     param1 = inParam1;
     param2 = inParam2;
     param3 = inParam3;

     gl_PointSize = 2.0f;
     gl_Position = projection * view * model * (inPosition +
     vec4(offset, 0.0));
}
```

Вершинний шейдер виконує стандартні дії, такі як передача даних про розрахункову сітку піксельному шейдеру і базовий розрахунок позиції

конкретного вузла сітки в екранному просторі. Додатково шейдер надає можливість налаштувати зміщення сітки щодо початку координат через інтерфейс за допомогою змінної " offset ".

Вихідний код піксельного шейдера представлено у лістингу 2.7.

Лістинг 2.7 - Вихідний код піксельного шейдера

```
#version 450 core

in float param1;
in float param2;
in float param3;

out vec4 outColor;

uniform vec3 color;
uniform float threshold;

void main(void)
{
     if (param1 > threshold)
          discard;

     outColor = vec4(color.r, color.g, color.b, 1.0);
}
```

Піксельний шейдер обчислює кінцевий колір пікселя, де буде відмальовано конкретний вузол. Також шейдер дозволяє налаштувати колір сітки за допомогою змінної " color " і поріг " threshold ", порівнюваний з одним з параметрів, що постачаються в даних разом з розрахунковою сіткою, залежно від якого вузол може бути відсічений з малювання.

Порівняльний результат відображення сітки за допомогою стандартних та наведених вище шейдерів можна побачити на рисунку 2.5.

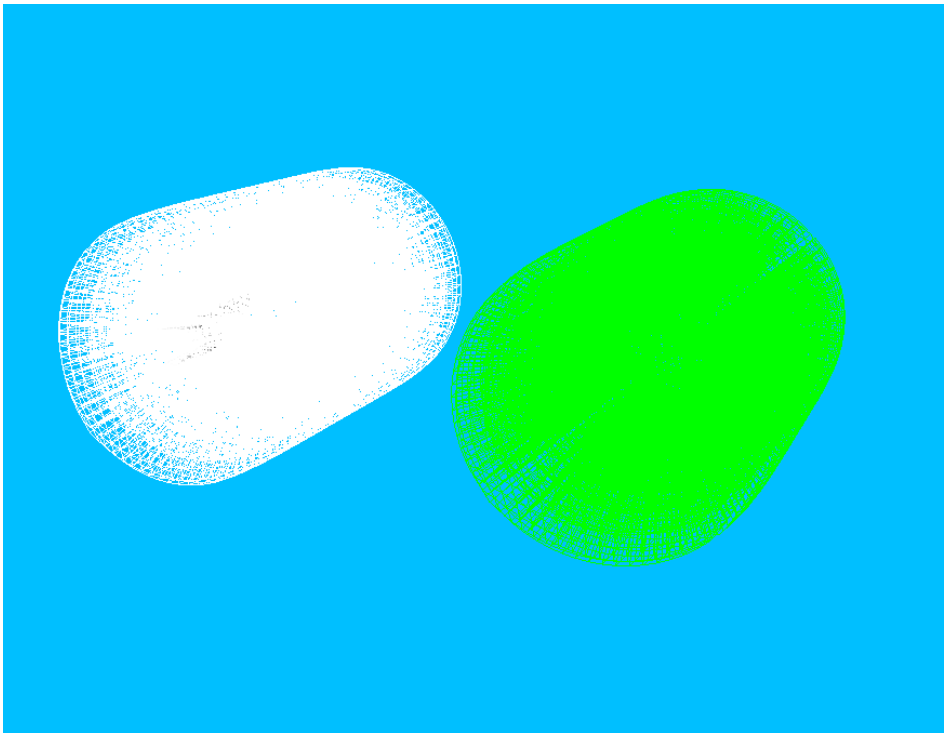

Рисунок 2.5 – Порівняння сітки зі стандартним матеріалом та сітки з окремо створеним матеріалом

Друга сітка зміщена щодо першої та зафарбована у салатовий колір для кращого візуального сприйняття інформації.

РОЗДІЛ 3. ПРАКТИЧНА ЧАСТИНА

3.1 Візуальне створення матеріалів

3.1.1 Концепція редактора вузлів

Описана концепція матеріалів та її реалізація показали, що такий підхід має певний потенціал для вирішення поставлених завдань. При цьому існує недолік, який заважає повній реалізації цього потенціалу.

Схема, яка вимагає ручного написання шейдерів, має на увазі наявність у користувача базових знань і понять із програмування, а також вимагає вивчення концепцій програмування шейдерів та мови GLSL. Це недостатньо доброзичливий до користувача підхід, який може призвести до виникнення безлічі дрібних помилок, що сповільнить роботу і не дозволить сфокусуватися на цілі дослідника, тобто вивчення розрахункових сіток.

Одним із можливих шляхів міг бути розвиток синтаксичного аналізу написаних шейдерів та створення розширеного діалекту мови GLSL, специфічного для даної системи. Але це не вирішило б зазначену проблему. По-перше, додаткові синтаксичні конструкції не полегшують вивчення роботи з програмою. По-друге, розширення синтаксису шейдерної мови робить пакет, що розробляється, несумісним з іншими програмами.

Необхідний спосіб подальшого спрощення створення нових візуальних відображень. Це призводить до висновку, що потрібний певний візуальний спосіб завдання алгоритмів для матеріалів, тобто деякий аналог візуального програмування.

Якщо звернутися до популярних програм для 3D -моделювання, в них найчастіше можна побачити редагування матеріалів, засноване на створенні графів, що складаються з вузлів та зв'язків між ними. Приклад редактора матеріалу у програмі Blender [22] можна побачити на рис. 3.1.

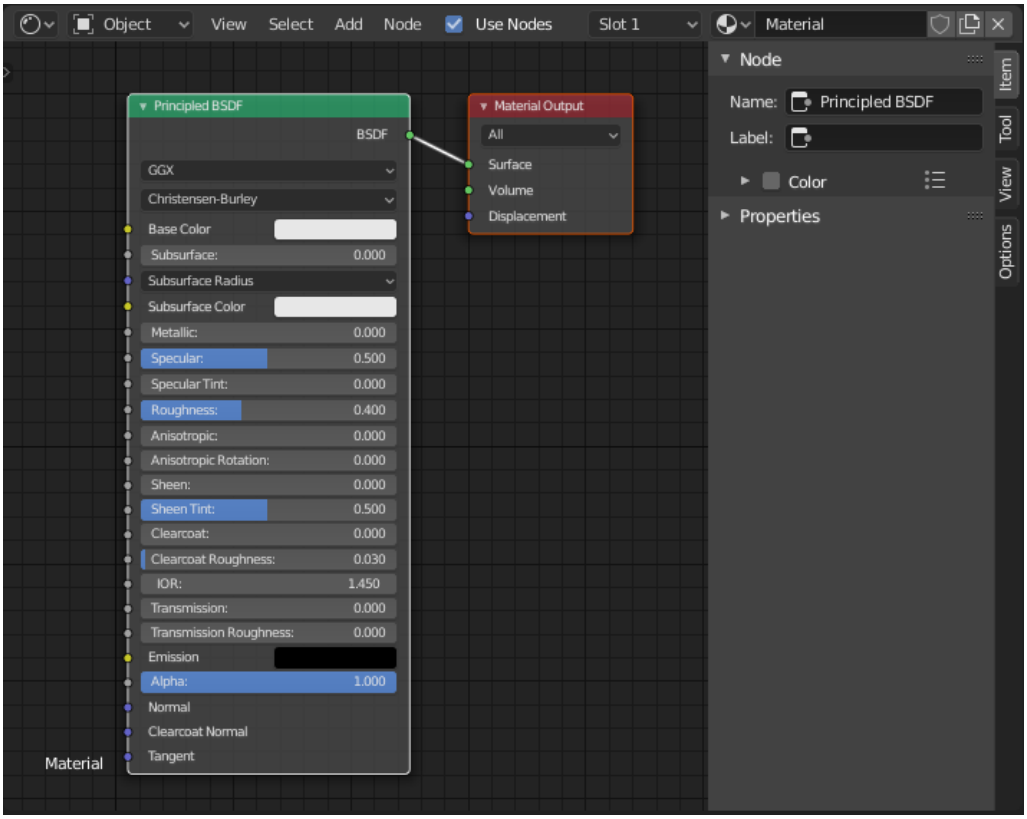

Рисунок 3.1 – Візуальний редактор матеріалів у програмі Blender

3.1.2 Опис редактора вузлів

Як було раніше сказано, редактор вузлів включає два типи сутностей. Це вузли та зв'язки.

Вузол є прямокутним блоком. Вузол має тип, що визначає функціональне призначення вузла та його візуальне відображення.

Є кілька типів вузлів:

– uniform- вузли - призначені для налаштування uniform- змінних, описаних у підрозділі 2.7.2;

– вузли-атрибути - призначені для отримання даних, що зберігаються в атрибутах, описаних у підрозділі 2.4;

– вузли-операції - призначені для перетворення вхідних даних та отримання вихідних;

– вихідні вузли призначені для отримання підсумкового результату роботи матеріалу.

У кожного вузла може бути кілька входів та виходів. У вхідних uniform-вузлів та вузлів-атрибутів є тільки виходи. У вихідних вузлів є лише входи. У

вузлів-операцій може бути довільна кількість входів та виходів. У кожного входу та виходу є тип даних. Набір типів даних обмежений типами даних, доступними у мові GLSL .

Вузли з'єднуються між собою за допомогою зв'язків через входи та виходи. Типи даних з'єднаних входу та виходу повинні збігатися. Зв'язки між вузлами вказують на напрямок руху даних. При створенні зв'язків слід уникати появи циклів, оскільки в такій ситуації немає сенсу у аналізованій концепції, про що буде сказано трохи докладніше пізніше у описі технічної реалізації. Говорячи формально, коректний матеріал є ациклічний напрямлений граф.

Користувач має можливість визначати власні вузли за допомогою створення файлів, що містять конфігурацію нового вузла. Про це написано докладніше.

Приклад матеріалу, налаштованого у редакторі вузлів, можна побачити на рисунку 3.2. Це найпростіший матеріал, який перетворює деякий налаштований колір у вектор, який потім використовується як результат роботи матеріалу. Код, згенерований для матеріалу, можна побачити у додатку А.

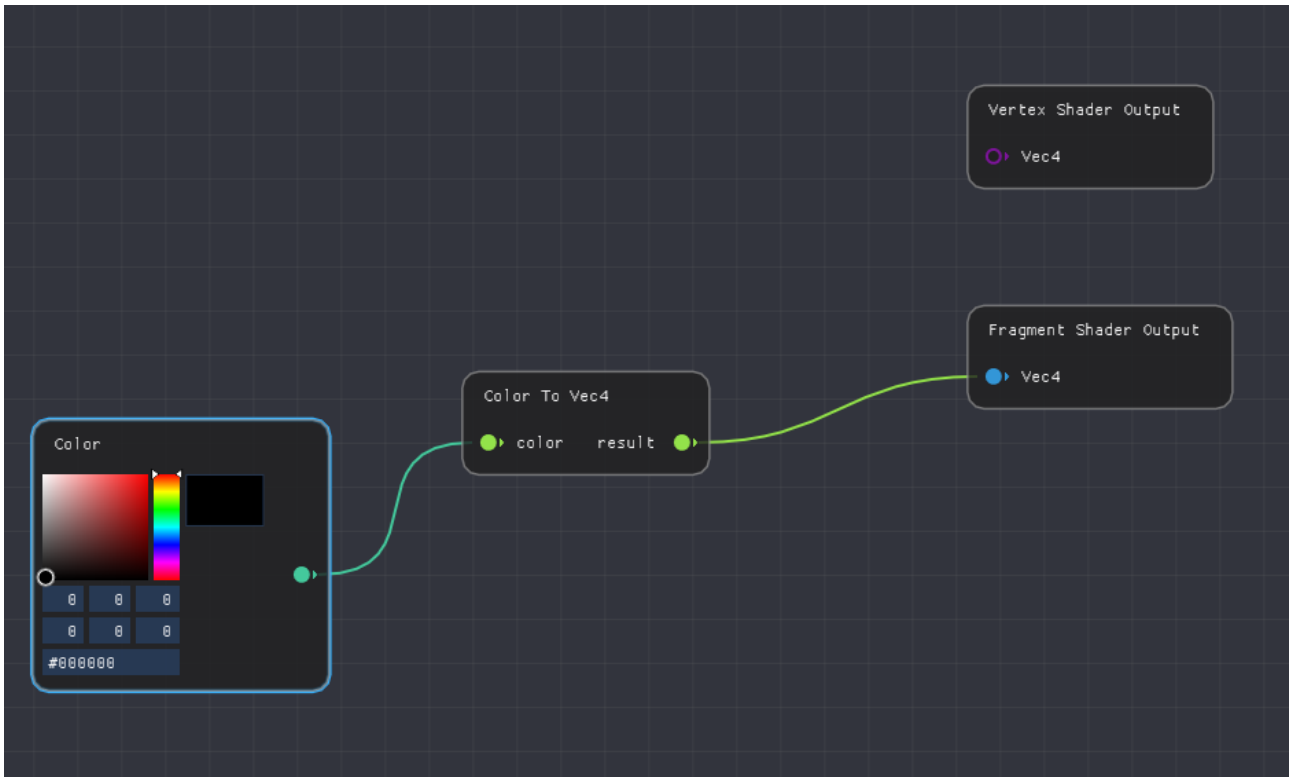

Рисунок 3.2 – Приклад найпростішого матеріалу, створеного у редакторі вузлів

3.1.3 Генерація шейдерів

Графи матеріалів, налаштовані в редакторі вузлів, перетворюються на шейдери. Таким чином досягається відображення матеріалу. Параметри, що налаштовуються через вхідні вузли, реально передаються на графічний прискорювач.

Основна технічна складність полягає у процесі перетворення графа у вихідний код шейдера. Інші елементи системи загалом вже було реалізовано на попередніх ітераціях.

Процес генерації вихідного коду шейдера складається із таких кроків.

1. Знаходиться вихідний вузол матеріалу.
2. Починається рекурсивний обхід ациклічного спрямованого графа у глибину.
3. Якщо відвідуваний вузол - вузол-операція, то в головну функцію шейдера додається нова частина коду відповідно до типу операції, про те, як обробляються вузли операції, можна докладніше прочитати в наступному підрозділі.
4. Якщо відвідуваний вузол - uniform -вузол або вузол-атрибут, то в код шейдера додається оголошення uniform -змінної або атрибута відповідно.
5. При поверненні з рекурсії кожному із вузлів-операцій підставляються змінні коду операції у необхідних місцях з того, з якими вузлами пов'язані входи даного вузла.
6. На останньому етапі окремі секції коду шейдера, такі як оголошення атрибутів, оголошення uniform- змінних, головна функція шейдера, поєднуються в єдиний шейдер, додаються необхідні технічні специфікації.

3.1.4 Бібліотека вузлів-операцій

Кожен тип вузла-операції є деяким перетворенням даних. Важливий момент для забезпечення розширення – можливість додавати нові види перетворень.

Усі вузли-операції організовані у бібліотеку текстових файлів, де кожен файл визначає один вузол-операцію. Додавання, видалення та зміна текстових

файлів — зрозумілий для кожного користувача комп'ютера процес. Деякий набір стандартних вузлів-операцій постачається разом із програмою, користувачі вільні додавати власні та ділитися ними один з одним.

Файл будь-якого коректного вузла-операції містить опис входів, виходів вузла і код, що виконується, підставляється в вихідний код шейдера. Приклад опису простого вузла, що виконує множення вектора на число, наведено у лістингу 3.1.

Лістинг 3.1 – Приклад опису простого вузла

```
Multiply Vec 4
in vec4 vector
in float value
out vec 4 result
vec 4 {2} = {0} * {1};
```

Перший рядок містить назву операції, яку бачить користувач при виборі операції з меню. Далі кілька рядків визначають входи та виходи вузла. Спочатку йдуть входи, що позначаються ключовим словом in на початку рядка, потім виходи, що позначаються ключовим словом out . Для кожного входу та виходу необхідно вказати тип даних та осмислене ім'я. Залишок файлу містить вихідний код, який буде підставлений у вихідний код шейдера. Він може займати довільну кількість рядків. У місцях коду, де мають бути підставлені певні вхідні та вихідні змінні, вказуються числа, що вказують на індекс змінної у списку, укладені у фігурні дужки.

3.1.5 Збереження та завантаження матеріалів

Для збереження та завантаження матеріалів була підібрана спеціалізована бібліотека для мови C ++ під назвою ajson. З її допомогою зберігання структур даних мови можна налаштовувати так, як це показано у лістингу 3.2.

Кожен матеріал зберігається в окремому файлі у форматі JSON.

Лістинг 3.2 – Код збереження структур даних мови

```
struct Person
{
	std::string Name;
	int Age;
};

AJSON(Person , Name , Age)

Person obj;

char * json = "{    \"Name\": \"Boo\",\"Age\": 28}";

ajson::load_from_buff(obj,json);
```

3.1.6 Організація сцени

Організація сцени у проекті дозволяє розташувати у просторі поруч один з одним декілька сіток. У кожній сітці можна налаштовувати позицію поворот і масштаб, а також для кожної з них можна призначити та налаштувати свій матеріал. На рисунку 3.3 можна побачити налаштування сцени в інтерфейсі користувача.

Рисунок 3.3 – Налаштування сцени

Тут представлено список завантажених сіток, налаштування їх позицій, поворотів, масштабів, вибір режиму подання та налаштування видимих частин сітки.

3.2 Результати та їх обговорення

3.2.1 Приклади відображення

Нижче наведено приклад відображення сітки за допомогою графа, показаного на рисунку 3.4.

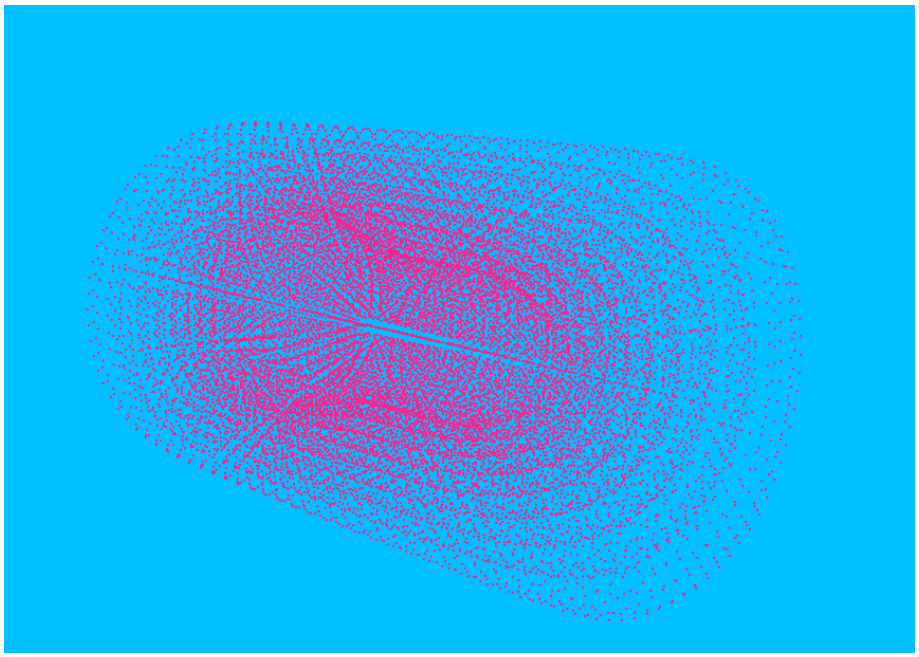

Рисунок 3.4 – Найпростіше відображення сітки одним кольором

Це мінімальний приклад, який дозволяє просто відобразити дані якимось одним кольором, що налаштовується.

Приклад відображення деякого параметра, заданого для кожного вузла сітки, можна побачити на рисунку 3.5. На рисунку 3.6 проілюстровано граф у редакторі вузлів.

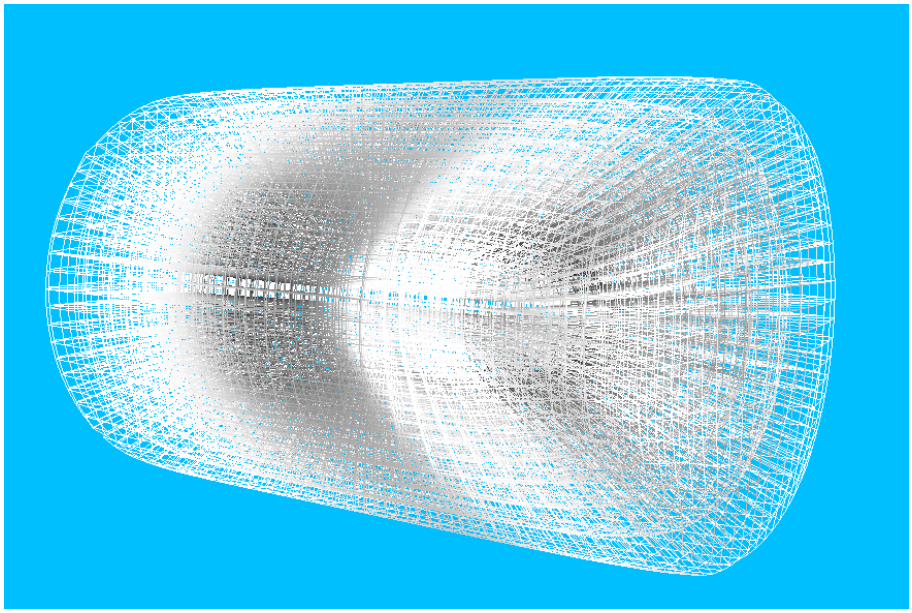

Рисунок 3.5 – Відображення параметра

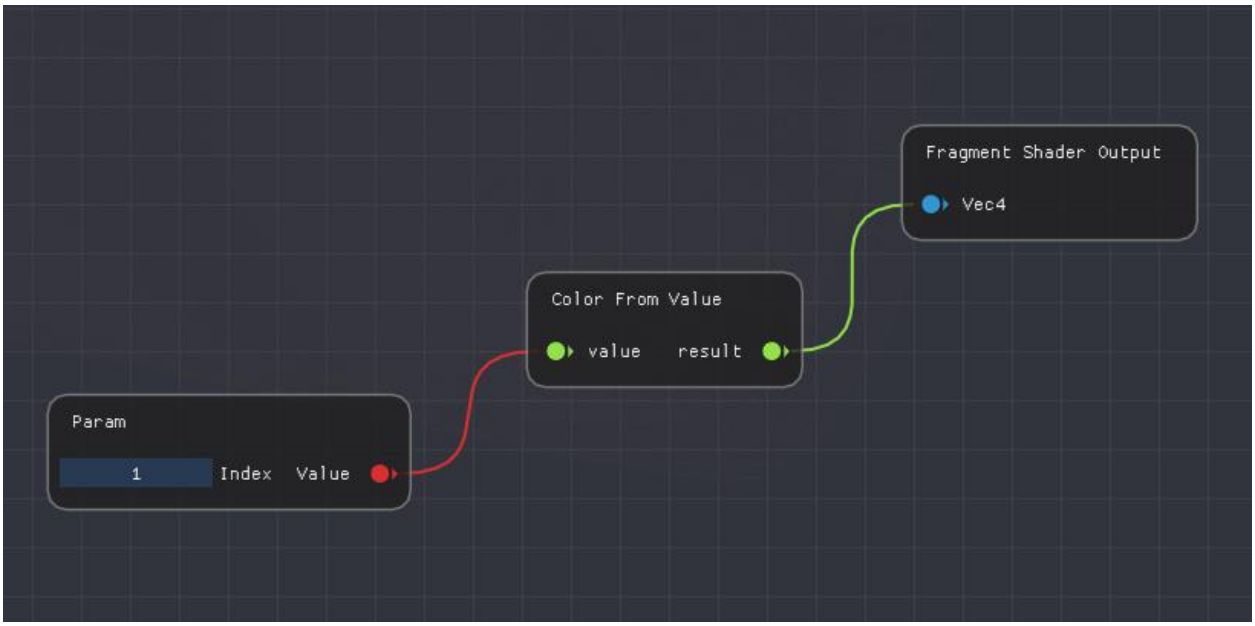

Рисунок 3.6 – Налаштування відображення параметра

Якщо змінити параметр, що відображається, відповідно змінюється відображення, як видно на рисунку 3.7 (помітні невеликі вкраплення темних відтінків сірого кольору всередині сітки).

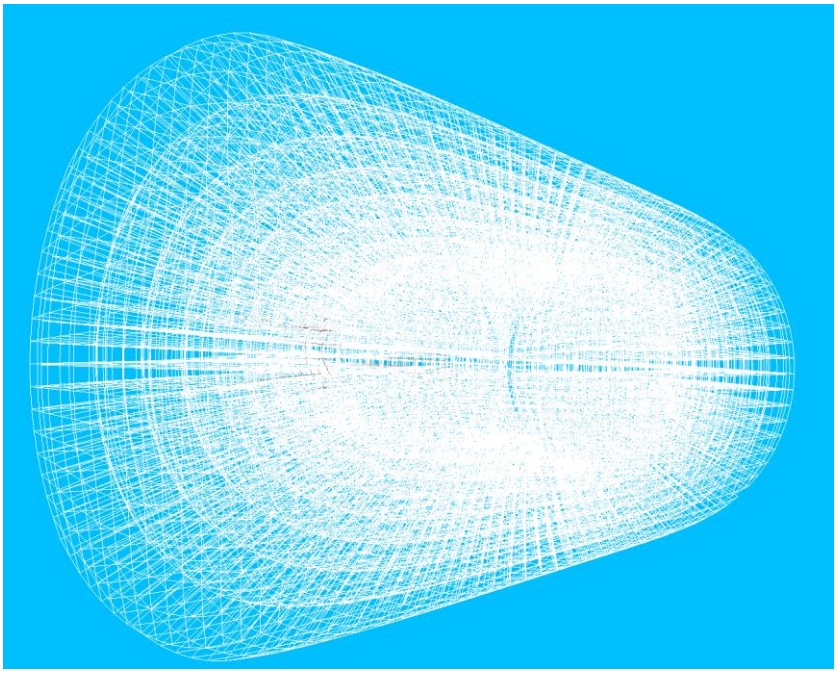

Рисунок 3.7 – Відображення з іншим параметром

Об'єднавши відображення даних кольором та параметром, як показано на рисунку 3.8, ми отримаємо результат, який видно на рисунку 3.9. Як видно, комбінування різних операцій відбувається за допомогою з'єднання заздалегідь створених відображень за допомогою деякого вузла, що об'єднує.

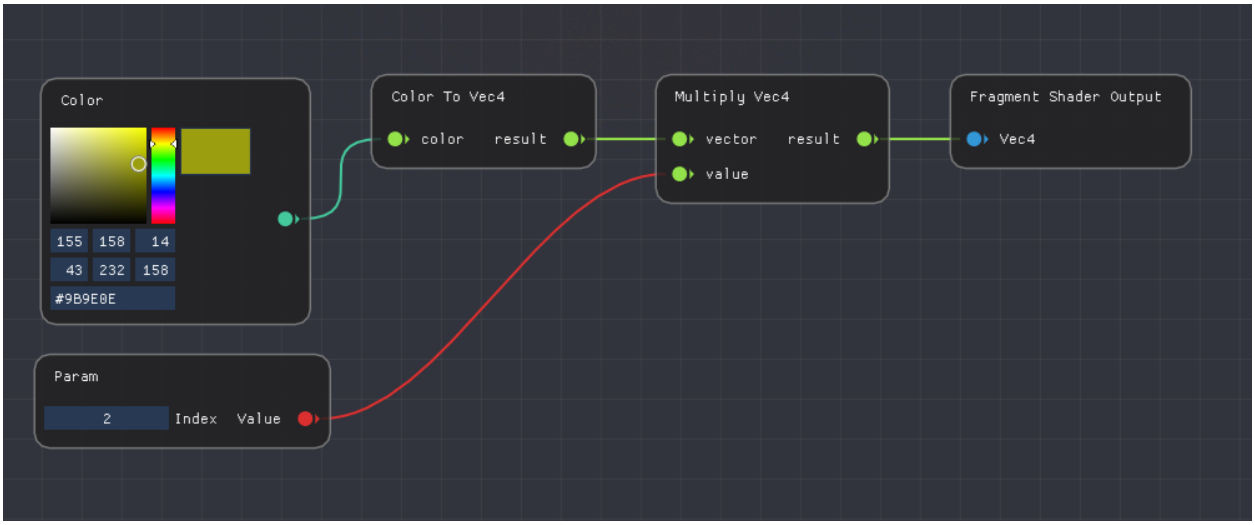

Рисунок 3.8 – Налаштування відображення з параметром та кольором

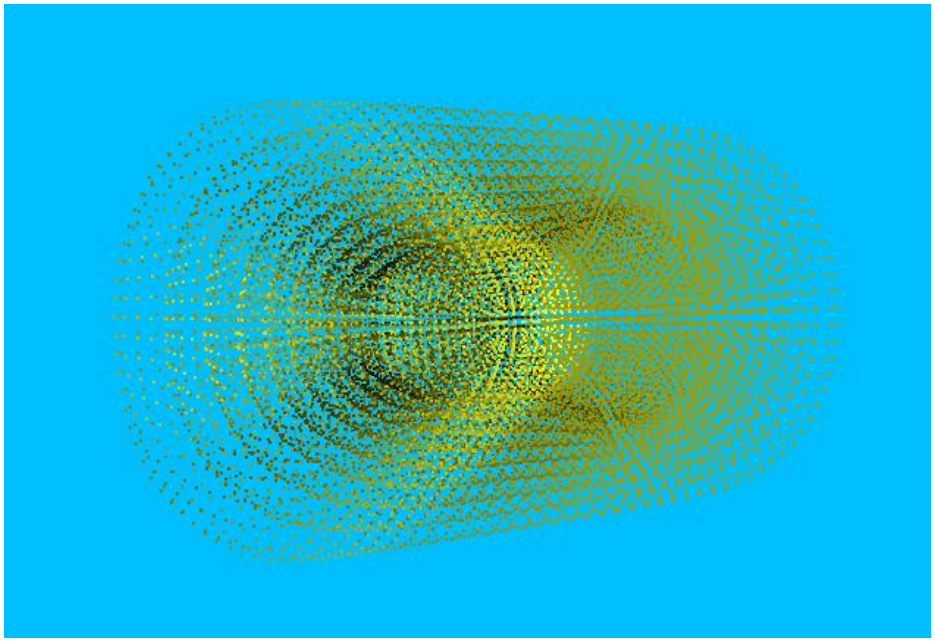

Рисунок 3.9 – Відображення параметрів і кольорів

За допомогою умовних вузлів ми можемо певним чином фільтрувати дані, маніпулюючи значення індексів вузлів. Приклад показаний на рисунках 3.10 та 3.11. Якщо граф стає громіздким, можна виділити набори операцій, що часто використовуються, в окремі вузли.

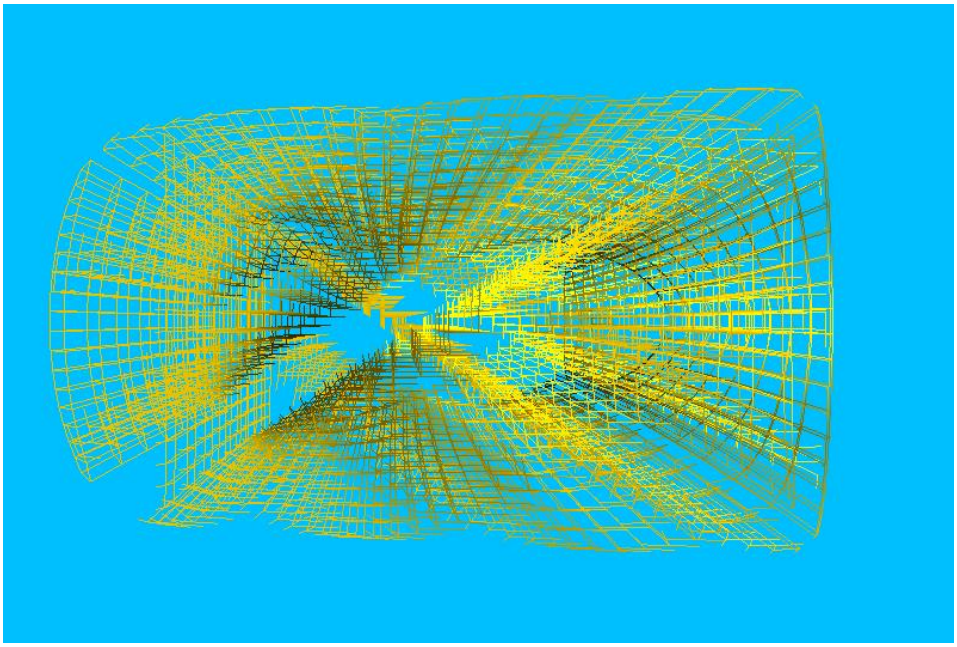

Рисунок 3.10 – Найпростіша фільтрація сітки за індексами вузлів

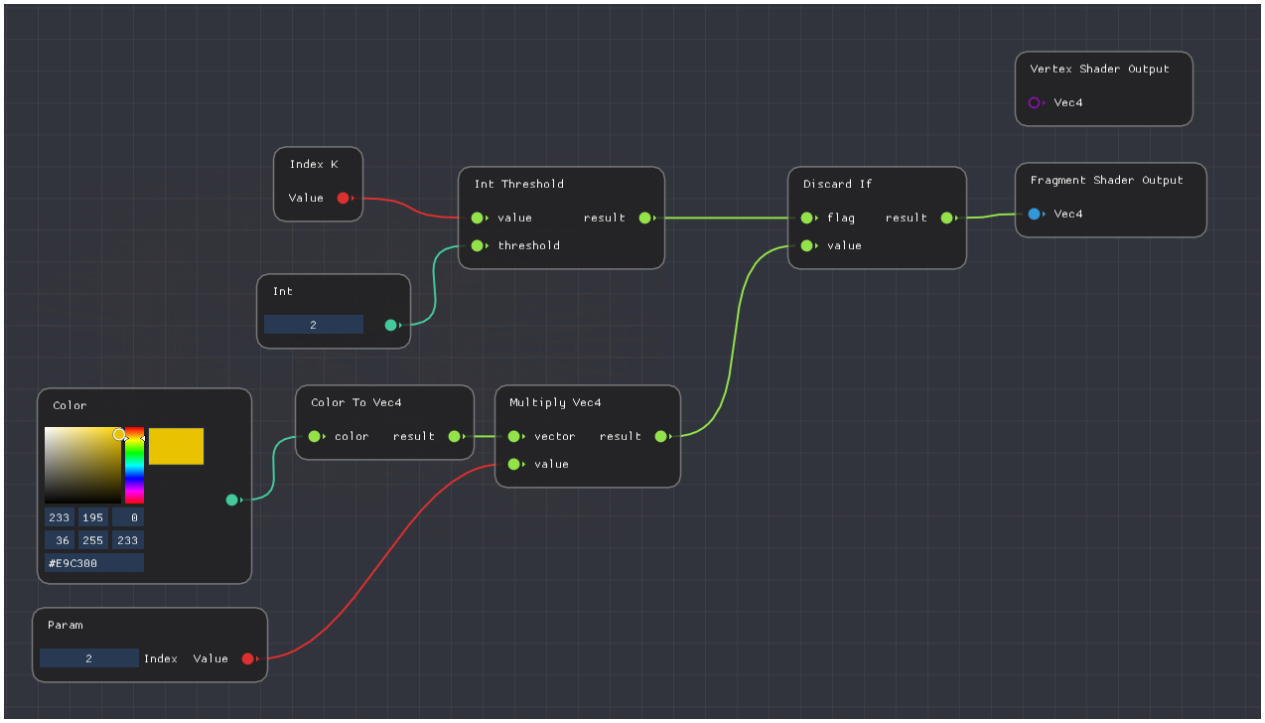

Рисунок 3.11 – Налаштування фільтрації сітки за індексами вузлів

Один і той самий матеріал можна застосовувати для різних наборів даних, що видно на рисунку 3.12.

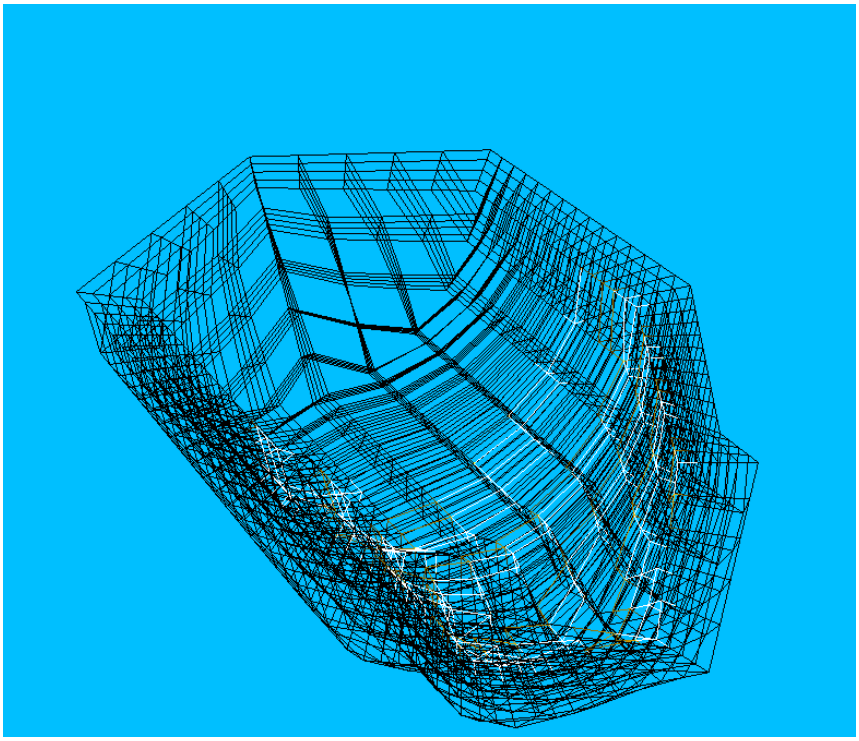

Рисунок 3.12 – Матеріал, застосований для іншої сітки

3.2.2 Напрямки подальшої роботи

Серед напрямів подальшої роботи можна виділити наступні:

1. Наведення типів. На момент написання тексту вузли можуть бути з'єднані, тільки якщо біля входу та виходу один і той самий тип даних. Це призводить до необхідності визначати додаткові вузли-операції, призначені для перетворення типів даних. Цього можна уникнути, якщо автоматично наводити тип, якщо це можливо. Наприклад, ціле число до плаваючої коми.

2. Повноцінна підтримка вершинних шейдерів. Це дозволить вільно маніпулювати положенням вузлів, комірок сіток у тій самій манері, як і зараз реалізовано із кольорами.

3. Підтримка геометричних шейдерів. Геометричні шейдери дозволяють генерувати додаткову геометрію на етапі виконання шейдера, що значно розширює набір можливих функцій, які можна реалізувати.

4. Покращена бібліотека вузлів-операцій. Було б зручно розподіляти вузли за категоріями, маючи файли в ієрархії папок.

5. Вкладені графи вузлів. Введення нового рівня абстракції дозволить підвищити можливість повторного використання готових графів вузлів, які виконують функціональність, котра часто повторюється.

6. Використання віртуальної дійсності. Даний напрямок можна назвати окремим і таким, що не відноситься безпосередньо до описаної концепції редактора вузлів. Тим не менш, у довгостроковій перспективі видається цікавою спроба реалізації створення нових візуальних відображень у віртуальній тривимірній реальності, ґрунтуючись на ідеях, що використовуються у двовимірному редакторі вузлів.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Навчання працюючих і інструктажі з охорони праці

Однією із складових ефективної роботи з профілактики виробничого травматизму є належна підготовка, навчання та підвищення кваліфікації працівників з питань охорони праці. Загальний порядок проведення навчання з питань охорони праці встановлений Законом України «Про охорону праці» (ст. 18. «Навчання з питань охорони праці»).

Виконання вимог Закону України «Про охорону праці» в частині проведення навчання та перевірки знань з питань охорони праці здійснюється відповідно до Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці, затвердженого наказом Держкомітету України з нагляду за охороною праці 26 січня 2005 р. № 15 (далі — Типове положення).

Нагляд за дотриманням вимог Типового положення здійснюють органи державного нагляду за охороною праці, а координацію та методичний супровід — Головний навчально-методичний центр та навчальні підрозділи експертно-технічних центрів Держгірпромнагляду.

Вивчення предмета «Охорона праці» при підготовці, перепідготовці та підвищенні кваліфікації працівників, які залучаються до виконання робіт з підвищеною небезпекою, на підприємстві регламентується п. 2.3 Типового положення. На підприємствах згідно з п. 1.1 Додатку 3 Типового положення навчання та перевірку знань з питань охорони праці повинні проходити керівники, заступники керівників, головні спеціалісти, керівники основних виробничих та технічних служб, безпосередньо пов’язані з організацією безпечного ведення робіт. Крім цього, згідно з п. 5 Додатку 3, навчання та перевірку знань з питань охорони праці мають проходити керівники, спеціалісти служб охорони праці, члени комісій з перевірки знань з питань охорони праці, особи, відповідальні за технічний стан і безпечну експлуатацію об’єктів

підвищеної небезпеки підприємств.

Типове положення встановлює порядок та місце проведення навчання та перевірки знань з питань охорони праці посадових осіб (п. 5.2 та п. 5.3). Посадові особи, перелік яких наведено в п. 5.2, проходять навчання у Головному навчально-методичному центрі Держнаглядохоронпраці. Перевірка знань цієї категорії посадових осіб проводиться комісією, створеною наказом Держгірпромнагляду.

Організацію навчання та перевірки знань з питань охорони праці працівників на підприємстві здійснюють працівники служби кадрів або інші спеціалісти, яким роботодавець доручив організацію цієї роботи. Навчання та перевірка знань з питань охорони праці працівників (виконавців і посадових осіб), які не залучаються до виконання робіт підвищеної небезпеки, проводиться не рідше ніж один раз на три роки. Посадові особи та працівники, які виконують роботи підвищеної небезпеки, проходять спеціальне навчання та перевірку знань відповідних нормативно-правових актів з охорони праці не рідше одного разу на рік.

Посадові особи малих підприємств (п. 5.4), які не мають можливості створити власні комісії з перевірки знань з питань охорони праці та провести навчання з питань охорони праці, проходять навчання та перевірку знань в навчальних закладах, які мають відповідний дозвіл.

Спеціальне навчання з питань охорони праці може проводитись безпосередньо на підприємстві або навчальним закладом, який має відповідний дозвіл. При проведенні такого навчання на підприємстві навчальні плани та програми розробляються з урахуванням конкретних видів робіт, виробничих умов і функціональних обов’язків працівників і затверджуються наказом керівника підприємства.

Періодичність інструктажів, навчання та перевірки знань з питань охорони праці залежить від видів виконуваних робіт та встановлюється Типовим положенням. Перевірка знань з питань охорони праці після проведення спеціального навчання проводиться комісією підприємства.

Якщо на підприємстві неможливо створити комісію з перевірки знань з

питань охорони праці (п. 4.4 Типового положення), перевірка знань проводиться комісією спорідненого підприємства або Теруправління Держгірпромнагляду.

Всі працівники та посадові особи підприємства, включаючи посадових осіб, відповідальних за виконання робіт підвищеної небезпеки (крім зазначених в п. 5.2 та п. 5.3 Типового положення), проходять навчання та перевірку знань з питань охорони праці на підприємстві. Типове положення не зобов'язує, але й не забороняє проводити навчання всіх виконавців та посадових осіб (особливо тих, що виконують роботи підвищеної небезпеки) в навчальних закладах. У нашій країні є багато підприємств, де таке навчання проводиться, і це має позитивні наслідки. Ті витрати, які при цьому несуть підприємства, перекриваються створенням більш безпечних умов праці і в результаті збереженням життя та здоров'я працівників.

Також в навчальних закладах проходять навчання та перевірку знань із загальних питань охорони праці всі посадові особи та фахівці, які проводять інструктажі або навчання підлеглих працівників з питань охорони праці, виконують роботи з проектування об'єктів, а також інші працівники, незалежно від того, передбачено таке навчання Типовим положенням чи ні.

4.2 Санітарно-гігієнічні вимоги до умов праці

На робочих місцях працівників, які відповідальні за експлуатацію сервісу управління механізмом авторських прав на мультимедійні файли, необхідно забезпечити дотримання вимог, затверджених Наказом Мінсоцполітики від 14.02.2018 за № 207 «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями».

Приміщення для роботи з ЕОМ мають бути обладнані системами опалення, кондиціонування повітря, або припливно-витяжною вентиляцією. У приміщеннях на робочих місцях мають забезпечуватись оптимальні значення параметрів мікроклімату: температури, відносної вологості та рухливості повітря відповідно до норм та правил, а також ДБН В.2.5-67:2013 «Опалення, вентиляція та кондиціонування», затверджених наказом Мінрегіону від 25.01.2013 р. № 24.

Відповідно до санітарних норм мікроклімату виробничих приміщень ДСН 3.3.6.042-99 в офісних приміщеннях, обладнаних ЕОМ, температура повітря повинна становити 22-25°С, відносна вологість повітря – 40-60 %, швидкість руху повітря – не більше 0,1 м/с [27].

Приміщення, призначені для роботи з ЕОМ, повинні мати природне освітлення. У виробничих приміщеннях, обладнаних ЕОМ, необхідно створити належне освітлення. При експлуатації сервісу управління механізмом авторських прав на мультимедійні файли, важливим, з точки зору охорони праці, є забезпечення достатньої величини природного та штучного освітлення, які визначені у НПАОП 0.00-7.15-18 [28]. Природне світло повинно бути бічним, зорієнтованим, як правило, на північ чи північний схід, і забезпечувати коефіцієнт природної освітленості не нижче 1,5%. При виробничій потребі дозволяється експлуатувати ЕОМ у приміщеннях без природного освітлення за узгодженням з органами Держпромгірнагляду та органами й установами санітарно-епідеміологічної служби. Вікна приміщень повинні мати регулювальні пристрої для відчинення, а також жалюзі, штори тощо. Штучне освітлення приміщення з робочими місцями, обладнаними відеотерміналами ЕОМ загального та персонального користування, має бути всеосяжним і рівномірним. У випадку, коли переважають роботи з документами, допускається комбіноване освітлення (додатково до загального освітлення встановлюється світильники місцевого освітлення). Світильники розміщуються збоку від робочих місць (переважно ліворуч), або локально над робочим місцем (при розташуванні відеотерміналів ЕОМ за периметром приміщення). Як джерело світла при штучному освітленні застосовуються, як правило, люмінесцентні лампи. У світильниках місцевого освітлення допускається застосування ламп розжарювання. Рівень освітленості на робочому місці має становити 300-500 лк. При використанні комбінованого освітлення не допускається відблисків на поверхні екрана та збільшення освітлення екрана вище 300 лк.

Орієнтація вікон повинна бути на північ або північний схід, вікна повинні мати жалюзі, які можна регулювати, або штори; не дозволяється розміщувати кабінети обчислювальної техніки у підвальних приміщеннях будинків; кабінети,

обладнані комп'ютерною технікою, в навчальних закладах повинні розміщуватись в окремих приміщеннях з природним освітленням та організованим обміном повітря; стіни, стеля і підлога та обладнання кабінетів комп'ютерної техніки повинні мати покриття із матеріалів з матовою фактурою з коефіцієнтом відбиття: стін — 40- 50 %, стелі — 70 - 80 %, підлоги — 20-30 %, предметів обладнання — 40-60 % (робочого столу — 40-50 %, корпуса дисплею та клавіатури — 30-50 %, стелажів — 40-60 %); поверхня підлоги повинна мати антистатичне покриття та бути зручною для вологого прибирання; забороняється використовувати для оздоблення інтер'єру приміщень комп'ютерних кабінетів полімерні матеріали (дерев'яно-стружкові плити, шпалери, що придатні для миття, плівкові та рулонні синтетичні матеріали, шаровий паперовий пластик та ін.), що виділяють у повітря шкідливі хімічні речовини, які перевищують гранично допустимі концентрації; вміст шкідливих хімічних речовин в повітрі дошкільних та учбових приміщень з комп'ютерною технікою не повинен перевищувати середньодобові концентрації [28].

Організація робочого місця фахівця із експлуатації сервісу повинна забезпечувати відповідність усіх елементів робочого місця та їх розташування ергономічним вимогам ДСТУ 8604:2015 «Дизайн і ергономіка. Робоче місце для виконання робіт у положенні сидячи. Загальні ергономічні вимоги».

Відстань від екрана до ока фахівців, які працюють за комп’ютером визначається згідно з вимогами ДСанПіН 3.3.2.007-98.

Рівень шуму не повинний перевищувати: на місцях, де працюють програмісти та оператори ЕОМ, 55 дБА, у лабораторіях, де складаються алгоритми та ведеться робота з документацією – 60 дБА, у машинному залі – 65 дБА, у приміщеннях, де розміщені гучні агрегати обчислювальних машин – 75 дБА.

ВИСНОВКИ

Упродовж проведеної роботи було розроблено інформаційну технологію та створено спеціалізований графічний пакет для візуалізації розрахункових сіток. Використання редактора вузлів є логічним продовженням попередніх експериментів, спрямованих на забезпечення гнучкості та розширюваності функціональності системи з боку користувача.

Реалізовано алгоритм генерації вихідного коду шейдерів на основі спрямованих ациклічних графів, що налаштовуються візуально. Система не має проблем із продуктивністю на заявлених обсягах даних. Потреба оптимізації може виникнути тільки при створенні справді складних і нетривіальних візуальних відображень на великій кількості даних.

Подальший розвиток цього напряму є перспективним, адаптація додаткових функцій з популярних програмних пакетів для 3D - моделювання з урахуванням потреб наукової візуалізації може принести значну користь у довгостроковій перспективі.

Можливе поєднання цього підходу з різними суміжними напрямками, такими як віртуальна реальність, управління жестами, онлайн-візуалізація та іншими.

.

ПЕРЕЛІК ДЖЕРЕЛ

1. Ushakova O.V. Advances in Grid Generation // Nova Science Publishers, New-York, 2007

2. Наквасюк В.В. Розробка системи візуалізації конкурентних і паралельних процесів програмного забезпечення: магістерська робота. К.: НУ КМА , 2021. 53 с.

3. Муляр В. П. Візуалізація даних та інфографіка. Харків: ФОП Панов А. М., 2020. 200 с.

4. Olsannikova E., Ometov A., Koucheryavy Y. and Olsson T., Visualizing Big Data with augmented and virtual reality: challenges and research agenda. // Journal Of Big Data, 2015, pp.1-27.

5. Tecplot [[Електронний ресурс] – Режим доступу: https://tecplot.com (дата звернення: 28.04.2026).

6. ParaView [Електронний ресурс] – Режим доступу: https://paraview.org (дата звернення: 28.04.2026).

7. OpenFOAM [Електронний ресурс] – Режим доступу: https:// openfoam.com (дата звернення: 28.04.2026).

8. Shreiner D., OpenGL Programming Guide // Addison-Wesley, Boston, 2013

9. Meiri E. OpenGL Step by Step [Електронний ресурс] – Режим доступу: http://ogldev.atspace.co.uk/index.htm (дата звернення: 29.04.2026).

10. Stroustrup B., The C++ Programming Language // Addison-Wesley, Boston, 2013.

11. SDL Wiki [Електронний ресурс] – Режим доступу: https://wiki.libsdl.org (дата звернення: 30.04.2026).

12. Dear ImGUI [Електронний ресурс] – Режим доступу: https://github.com/ocornut/imgui (дата звернення: 30.04.2026).

13. Visual Studio [Електронний ресурс] – Режим доступу: https://visualstudio.microsoft.com (дата звернення: 01.05.2026)

14. Git [Електронний ресурс] – Режим доступу: https://git-scm.com (дата звернення: 01.05.2026).

15. Imgui-node-editor [[Електронний ресурс] – Режим доступу: https://github.com/thedmd/imgui-node-editor (дата звернення: 03.05.2026).

16. Fmt [Електронний ресурс] – Режим доступу: https://github.com/fmtlib/fmt (дата звернення: 03.05.2026).

17. Ajson [Електронний ресурс] – Режим доступу: https://github.com/lordoffox/ajson (дата звернення: 05.05.2026).

18. Wright R.S., Haemel N., Sellers G., Lipchak B., OpenGL SuperBible // Addison-Wesley, Boston, 2010.

19. Bailey M., Cunningham S., Graphics Shaders. Theory and Practice. Second Edition // CRC Press, New York, 2012.

20. Lengyel E., Mathematics for 3D Game Programming and Computer Graphics // Course Technology PTR, Boston, 2012.

21. Lengyel E., Foundations of Game Engine Development Volume 1: Mathematics // Terathon Software LLC, Lincoln, 2016.

22. Blender [Електронний ресурс] – Режим доступу: https://blender.org (дата звернення: 10.05.2026).

23. Методичні вказівки до виконання кваліфікаційної роботи ор Бакалавр для студентів спеціальності 126 – Інформаційні системи та технології, всіх форм навчання / укладачі: Готович В.А., Дуда О.М. Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2024. 43 с.

24. Bodnarchuk, I., Skorenkyy, Y., Kramar, T., Duda, O., & Nykytyuk, V. (2022). Use of Analytical Hierarchy Process in Scenarios Design for a Digital Museum with XR components. ITTAP, 414–425.

25. Zagorodna, N., Skorenkyy, Y., Kunanets, N., Baran, I., Stadnyk, M. Augmented Reality Enhanced Learning Tools Development for Cybersecurity Major. CEUR Workshop Proceedings., 2022, 3309, pp. 25–32.

26. Skorenkyy, Yu., Kozak, R., Zagorodna, N., Kramar, O., Baran, I. Use of augmented reality-enabled prototyping of cyber-physical systems for improving cyber-security education. Journal of Physics: Conference Series., 2021, 1840(1), 012026.

27. Заікіна Д., Глива В. Основи охорони праці та безпека життєдіяльності. 2019. URL: https://doi.org/10.31435/rsglobal/001 (дата звернення: 14.05.2026).

28. Безпека в надзвичайних ситуаціях. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання / укл.: Стручок В. С. Тернопіль: ФОП Паляниця В. А., 2022. 156 с.

ДОДАТКИ

Додаток А

Вихідний код вершинного шейдера

```
#version 450 core

layout(location = 0) in vec4 inPosition;
layout(location = 1) in float inParam1;
layout(location = 2) in float inParam2;
layout(location = 3) in float inParam3;
layout(location = 4) in int inIndexI;
layout(location = 5) in int inIndexJ;
layout(location = 6) in int inIndexK;

out vec4 outPosition;
out float param1;
out float param2;
out float param3;
flat out int indexI;
flat out int indexJ;
flat out int indexK;

uniform mat4 view;
uniform mat4 projection;
uniform mat4 model;

void main(void)
{
  outPosition = inPosition;
  param1 = inParam1;
  param2 = inParam2;
  param3 = inParam3;
  indexI = inIndexI;
  indexJ = inIndexJ;
  indexK = inIndexK;
  gl_Position = projection * view * model * inPosition;
  gl_PointSize = 2.0f;
}
```

Вихідний код піксельного шейдера

```
#version 450 core

in vec4 position;
in float param1;
in float param2;
in float param3;
flat in int indexI;
flat in int indexJ;
flat in int indexK;

uniform vec4 var_7;

out vec4 outColor;

void main(void)
{
  outColor = var_7;
}
```

Додаток Б

Репозиторій із вихідним кодом проекту знаходиться у відкритому доступі на сайті Github.

Для відкриття та компіляції проекту потрібна наявність Visual Studio 2019 на комп'ютері розробника. Під час розповсюдження програми потрібне лише встановлення актуального розповсюджуваного пакета Microsoft.