ДОДАТОК А

Перелік джерел посилання за науковими напрямами керівника та науковців кафедри програмної інженерії

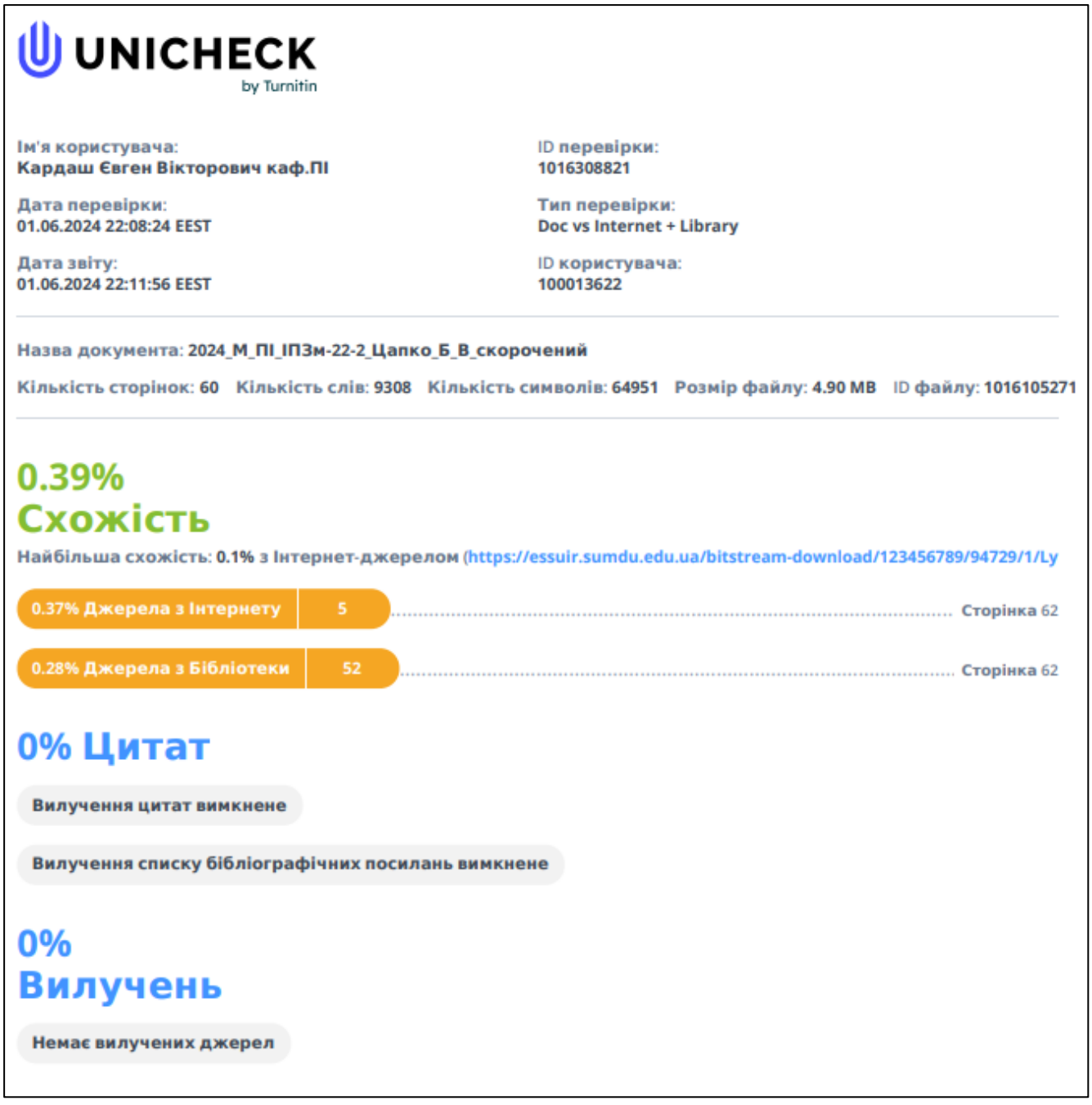
9. Vlasenko, L.A., Rutkas, A.G. On a differential game in a system described by an implicit differential-operator equation // Diff. Equations 2015 №51, pp. 798–807.

10. Vlasenko, L.A., Rutkas, A.G., Chikrii, A.A. On a Differential Game in a Stochastic System Proc. of the Steklov Inst. of Mathematics, 2020, 309, c. S185-S198.

16. Vlasenko L.A., Rutkas A.G., Chikrii, A.O. Functional-Differential Games with Nonatomic Difference Operator// Ukrainian Mathematical Journal. 2022. №74(2), pp. 186–202.

ДОДАТОК Б

Звіт результатів перевірки на унікальність тексту в базі ХНУРЕ



UNICHECK
by Turnitin

Ім'я користувача:
Кардаш Євген Вікторович каф.ПІ

ID перевірки:
1016308821

Дата перевірки:
01.06.2024 22:08:24 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
01.06.2024 22:11:56 EEST

ID користувача:
100013622

Назва документа: 2024_М_ПІ_ІПЗм-22-2_Цапко_Б_В_скорочений

Кількість сторінок: 60 Кількість слів: 9308 Кількість символів: 64951 Розмір файлу: 4.90 MB ID файлу: 1016105271

0.39%
Схожість

Найбільша схожість: 0.1% з Інтернет-джерелом (https://essuir.sumdu.edu.ua/bitstream-download/123456789/94729/1/Ly

0.37% Джерела з Інтернету 5 ..... Сторінка 62

0.28% Джерела з Бібліотеки 52 ..... Сторінка 62

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

ДОДАТОК В

Слайди презентації

Дослідження методів оптимізації OpenGL та Vulkan

Цапко Богдан Вікторович, ІПЗм-22-2
Науковий керівник: доцент, доц. каф. ПІ, Чуприна А.С.

2

Актуальність

У світі, де рендерінг стає все більше важливим компонентом різноманітних застосунків, питання продуктивності та вибору графічного API стає все більш і більш актуальним. Дана робота є важливим кроком у розумінні і актуалізації області графічного моделювання на прикладі API OpenGL та Vulkan, надаючи цінний внесок у вибір оптимального інструментарію для розробників у сучасному світі комп'ютерної графіки.

3

Огляд аналогів

Інтерфейси OpenGL і Vulkan тісно пов’язані між собою історією та підходами до розробки. Тому часто в приклад приводять саме такі порівняння. Але інтернет-ресурси та різні статті дають тільки поверхневий огляд можливостей методів рендерингу та не акцентують увагу на порівняні окремих частин, а тільки готових продуктів. Наше дослідження порівнює різну комбінацію методів на прикладі інтерфейсів OpenGL і Vulkan.

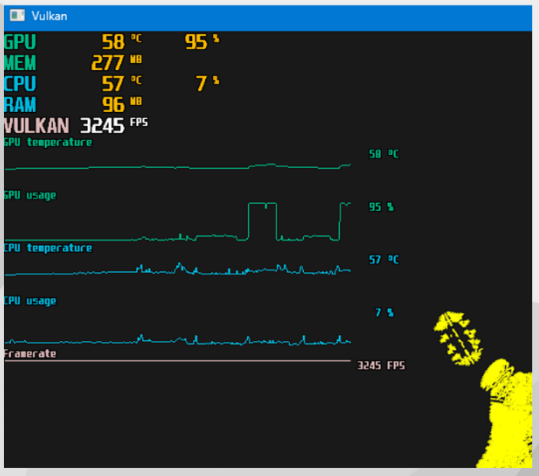
4

Постановка задачі

Постановка задачі ґрунтується на важливості вибору оптимального графічного API для розробки програм з високою продуктивністю графічного моделювання. Вона включає перед собою завдання дослідити та порівняти різні підходи до оптимізації графічного моделювання, зокрема методи рендерингу та обробки графічних об'єктів. Результатами повинні стати порівняння параметрів продуктивності частин пристрою для віртуальних сцен з різним налаштуванням.

5

Методологія



Основним методом, за допомогою якого здійснюється порівняння двох графічних API, є порівняльний аналіз. Досліджуються їх можливості та ефективність реалізації методів для оптимізації графічного моделювання. Але без практичних експеременті в важко дати оцінку продуктивності різним методам.

Тому для успішного порівняння та проведення експеременті в нам потрібні навички з роботою:

- API OpenGL та Vulkan;
- Шейдерами;
- Текстурами;
- Буферами;
- RivaTunerStatisticsServer – засіб для вимірювання продуктивності;
- Visual Studio 2022 – середовище розробки додатку.

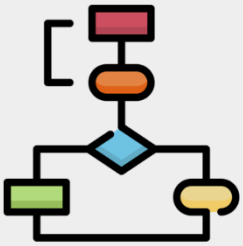
6

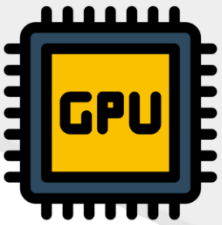
Архітектура систем

Для проведення нашого дослідження будемо використовувати OpenGL 4.6, GLSL 4.5, Vulkan 1.3.275.

Конфігурація системи, на якій розроблялась програма та проходили тести, містить:

- процесор Intel i7-12650H;
- оперативну пам’ять DDR5 4800МГц 16 ГБ;
- відеокарту NVIDIA RTX 4060 Laptop;
- операційну систему Windows 11.





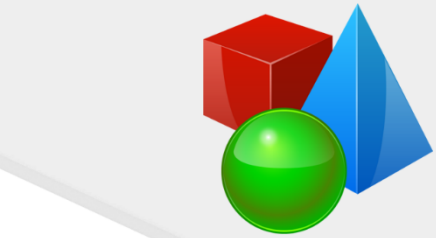
7

Опис програмного забезпечення

Була використана мова програмування C++ для написання коду для роботи інтерфейсів OpenGL та Vulkan, а також мова GLSL для написання шейдерів.

Були використані такі сторонні бібліотеки:

- SOIL і stb_image для завантаження зображень (текстур);
- Assimp і tinyobjloader для завантаження 3D-файлів;
- glm для математичних розрахунків, наприклад, матриць;
- GLEW для кращої взаємодії з OpenGL;
- GLFW для створення вікон, контекстів і поверхонь, обробки подій тощо.

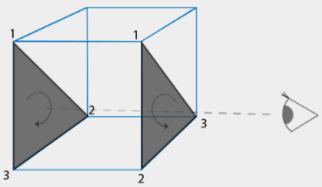


8

Опис програмного забезпечення

Для того, щоб чітко провести дослідження спільних методів, потрібно встановити спільні параметри для рендерингу, використовувати однакові об’єкти, текстури тощо. Параметри, які були встановлені для обох додатків API:

- Використання шейдерів;
- Згладжування;
- Відсікання нелицьових граней;
- Однакове середовище (Windows);
- Розширення екрану, кількість об’єктів для інстансованого рендерингу, вертикальна синхронізація, тести глибини та змішування кольорів;
- Можливість переміщуватися у віртуальному просторі.

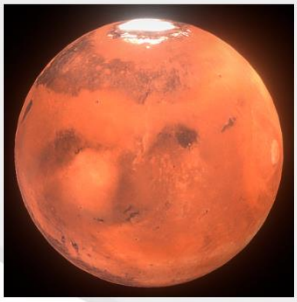


9

Зміст проведеного експерименту

Базовим предметом для тестування стане рюкзак для виживання, а для інстансованого рендерингу – планета Марс.

Для тестування шейдерів ми створили декілька, один з яких симулює освітлення за моделлю Блінна-Фонга, два інших потрібні для тестування геометричного шейдера, інші потрібні для інстансованого рендерингу чи для інших потреб з мінімальним навантаженням.



10

Зміст проведеного експерименту

Ми створили 10 сцен для кожного інтерфейсу, але тестів буде 32 (20 головних і 12 додаткових). Сцени, які будуть потрібні для проведення нашого дослідження такі:

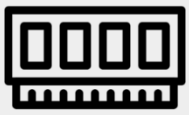
- шейдер освітлення, 2 предмета, згладжування 8x;
- геометричний шейдер, 1 предмет, згладжування 8x;
- інший геометричний шейдер, 1 предмет, згладжування 8x;
- комбінація з 2 геометричних шейдерів, 2 предмета, згладжування 8x;
- комбінація з 1 шейдера освітлення та 2 геометричних, 4 предмета, згладжування 8x;
- шейдер для виведення 3000 планет, згладжування 8x;
- інстансований шейдер для виведення 3000 планет, згладжування 8x;
- комбінація з 1 шейдера освітлення, 2 геометричних, 1 інстансованого, 3000 планет і 4 рюкзака, згладжування 8x;
- комбінація з 1 шейдера освітлення, 2 геометричних, 1 інстансованого, 3000 планет і 4 рюкзака, без згладжування;
- пусте середовище.

11

Зміст проведеного експерименту

Критерії оцінки продуктивності для виявлення кращого оптимізованого методу рендерингу такі:

- завантаженість графічного процесора (у %);
- завантаженість відеопам’яті (в МВ);
- завантаженість центрального процесора (у %);
- завантаженість оперативної пам’яті (в МВ);
- кількість кадрів за секунду.

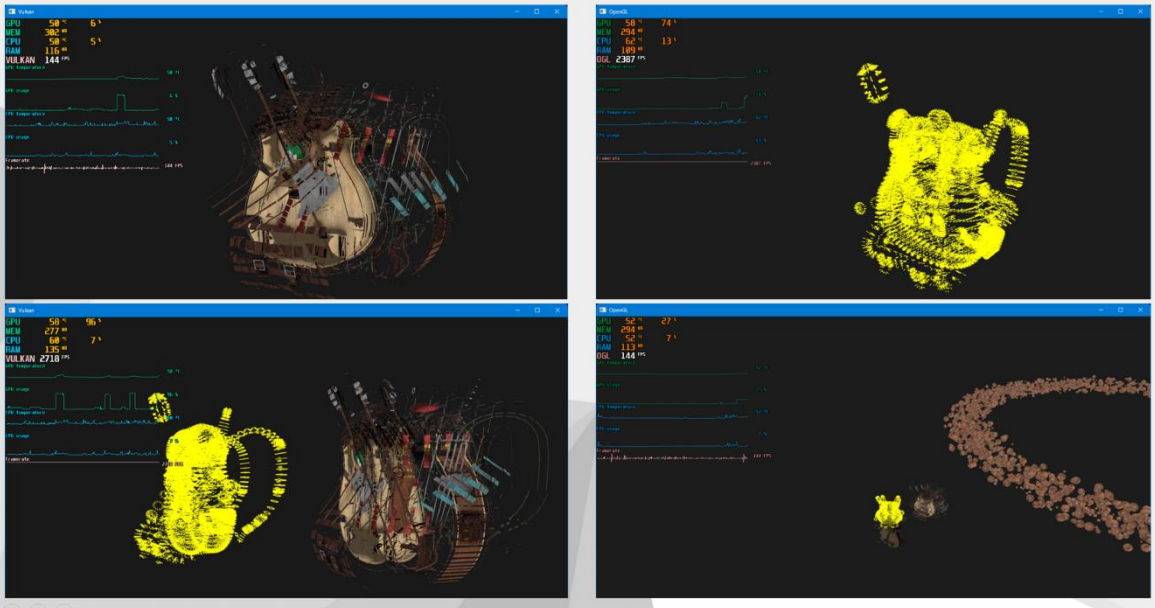


Додатковими критеріями оцінки є:

- температура графічного процесора (в °С);
- температура центрального процесора (в °С).

12

Приклади проведення експериментів



13

Результати головного експерименту

	OpenGL							Vulkan						
	GPU T	GPU	GPU M	CPU T	CPU	RAM	FPS	GPU T	GPU	GPU M	CPU T	CPU	RAM	FPS
Шейдер освітл. з VSync	48	5	294	48	5	125	144	50	6	302	50	5	126	144
Шейдер освітл. без VSync	55	36	294	64	12	125	1800	62	92	277	62	10	96	4700
Геом. шейдер з VSync	49	6	294	51	7	129	144	49	5	302	50	5	116	144
Геом. шейдер без VSync	54	40	294	65	10	129	2160	58	88	277	62	11	97	5350
Інший геом. шейдер з VSync	51	8	294	53	7	109	144	50	8	302	50	6	113	144
Інший геом. шейдер без VSync	57	75	294	64	12	109	2280	57	95	277	58	9	97	3000
Комбін. із геом. шейдерів з VSync	51	9	294	51	5	109	144	50	8	302	51	5	119	144
Комбін. із геом. шейдерів без VSync	58	60	294	51	13	109	1800	58	95	277	60	8	95	3200
Геом. шейдери та освітл. з VSync	51	10	294	53	5	109	144	48	8	302	48	5	119	144
Геом. шейдери та освітл. без VSync	59	45	294	71	11	109	1230	60	95	277	60	8	97	2900
Рендер. планет з VSync	52	23	294	58	8	114	144	50	20	302	50	5	121	144
Рендер. планет без VSync	59	39	294	73	11	114	345	66	98	277	57	7	97	970
Інстанс. рендер. планет з VSync	54	25	294	54	5	109	144	51	18	302	50	5	115	144
Інстанс. рендер. планет без VSync	68	100	294	60	5	109	777	65	98	277	60	7	95	970
Всі предмети з VSync	54	27	294	54	5	109	144	48	23	302	48	5	117	144
Всі предмети без VSync	68	100	294	65	8	109	673	62	99	277	54	8	102	830
Всі об'єкти без згладж., з VSync	55	26	249	55	5	109	144	51	26	249	51	5	118	144
Всі об'єкти без згладж. і VSync	67	100	249	66	8	109	703	59	99	224	53	5	95	710
Порожня сцена з VSync	52	4	294	53	5	110	144	48	5	302	50	5	119	144
Порожня сцена без VSync	55	40	294	66	13	110	3600	52	75	277	60	11	96	5900

14

Аналіз головного експеримент

	GPU T	GPU	GPU M	CPU T	CPU	RAM	FPS
Шейдер освітл. з VSync	2	1	8	2	0	1	0
Шейдер освітл. без VSync	7	56	-17	-2	-2	-29	2900
Геом. шейдер з VSync	0	-1	8	-1	-2	-13	0
Геом. шейдер без VSync	4	48	-17	-3	1	-32	3190
Інший геом. шейдер з VSync	-1	0	8	-3	-1	4	0
Інший геом. шейдер без VSync	0	20	-17	-6	-3	-12	720
Комбін. із геом. шейдерів з VSync	-1	-1	8	0	0	10	0
Комбін. із геом. шейдерів без VSync	0	35	-17	9	-5	-14	1400
Геом. шейдери та освітл. з VSync	-3	-2	8	-5	0	10	0
Геом. шейдери та освітл. без VSync	1	50	-17	-11	-3	-12	1670
Рендер. планет з VSync	-2	-3	8	-8	-3	7	0
Рендер. планет без VSync	7	59	-17	-16	-4	-17	625
Інстанс. рендер. планет з VSync	-3	-7	8	-4	0	6	0
Інстанс. рендер. планет без VSync	-3	-2	-17	0	2	-14	193
Всі предмети з VSync	-6	-4	8	-6	0	8	0
Всі предмети без VSync	-6	-1	-17	-11	0	-7	157
Всі об'єкти без згладж., з VSync	-4	0	0	-4	0	9	0
Всі об'єкти без згладж. і VSync	-8	-1	-25	-13	-3	-14	7
Порожня сцена з VSync	-4	1	8	-3	0	9	0
Порожня сцена без VSync	-3	35	-17	-6	-2	-14	2300

15

Результати та аналіз додаткового експерименту

OpenGL	GPU T	GPU	GPU M	CPU T	CPU	RAM	FPS
Всі об’єкти з VSync	54	27	294	53	4	109	144
Половина планет з VSync	54	26	294	55	5	109	144
Тільки рюкзаки з VSync	54	26	294	53	6	109	144
Всі об’єкти без VSync	66	100	294	66	7	114	680
Половина планет без VSync	66	100	294	66	8	114	680
Тільки рюкзаки без VSync	65	100	294	70	11	114	680

Vulkan	GPU T	GPU	GPU M	CPU T	CPU	RAM	FPS
Всі об’єкти з VSync	51	23	302	50	6	109	144
Половина планет з VSync	50	19	302	50	7	101	144
Тільки рюкзаки з VSync	50	18	302	52	8	100	144
Всі об’єкти без VSync	65	99	277	57	6	103	800
Половина планет без VSync	66	98	277	70	6	102	1050
Тільки рюкзаки без VSync	64	98	277	69	9	101	1100

	GPU T	GPU	GPU M	CPU T	CPU	RAM	FPS
Всі об’єкти з VSync	-3	-4	8	-3	2	0	0
Половина планет з VSync	-4	-7	8	-5	2	-8	0
Тільки рюкзаки з VSync	-4	-8	8	-1	2	-9	0
Всі об’єкти без VSync	-1	-1	-17	-9	-1	-11	120
Половина планет без VSync	0	-2	-17	4	-2	-12	370
Тільки рюкзаки без VSync	-1	-2	-17	-1	-2	-13	420

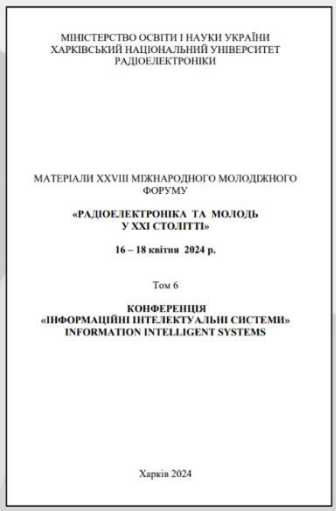
16

Аналіз отриманих результатів

- Vulkan при роботі шейдера освітлення показав завантаженість GPU на 92% та FPS у 4700 кадрів без вертикальної синхронізації, що відповідно на 56% та 2900 кадрів (у 2,6 разів) більше, ніж OpenGL;
- Vulkan при роботі першого геометричного шейдера показав завантаженість GPU на 88% та FPS у 5350 кадрів без вертикальної синхронізації, що відповідно на 48% та 3190 кадрів (у 2,4 разів) більше, ніж OpenGL;
- Vulkan при роботі комбінації геометричних шейдерів показав завантаженість GPU на 95%, CPU на 8% та FPS у 3000 кадрів без вертикальної синхронізації, що відповідно більше на 35% для GPU та 1400 кадрів (у 1,7 разів), менше на 5% для CPU, ніж OpenGL;
- Vulkan при роботі шейдерів для рюкзаків показав завантаженість GPU на 95% та FPS у 2900 кадрів без вертикальної синхронізації, що відповідно на 50% та 1670 (у 2,3 разів) кадрів більше, ніж OpenGL;
- Vulkan при роботі звичайного шейдера для 3000 планет показав завантаженість GPU на 98%, CPU на 7% та FPS у 970 кадрів без вертикальної синхронізації, що відповідно більше на 59% для GPU та 625 кадрів (у 2,8 разів), менше на 4% для CPU, ніж OpenGL;
- Vulkan при захваті екраном менше об’єктів, ніж є у віртуальному просторі, показав зменшену завантаженість GPU з вертикальною синхронізацією, більший FPS на 120, 370 та 420 кадрів відповідно від всіх планет до повного їх зникнення порівняно з OpenGL.

17

Публікація результатів



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ XXVIII МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

16 – 18 квітня 2024 р.

Том 6

КОНФЕРЕНЦІЯ «ІНФОРМАЦІЙНІ ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ» INFORMATION INTELLIGENT SYSTEMS

Харків 2024

УДК 004.92:004.428 DOI: https://doi.org/10.30837/ITSSI.2024.293

ОПТИМАЛЬНЕ ВИКОРИСТАННЯ OPENGL ТА VULKAN

Цапко Б. В.

Науковий керівник – к.т.н., доцент Чуприна А. С.

Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна
e-mail: bohdan.tsapko@nure.ua

The purpose of this work is to make the optimal choice between the OpenGL and Vulkan APIs depending on the needs of developers to obtain the required performance characteristics based on the implementation of similar optimization methods in both APIs. The methods of graphic modeling, their implementation and methods of reproduction, various approaches to the optimization of graphic modeling for certain methods of visualization and processing of graphic objects are considered. Techniques such as using shaders, optimizing textures and buffers have been thoroughly reviewed and compared in terms of performance and how they are implemented in the OpenGL and Vulkan APIs.

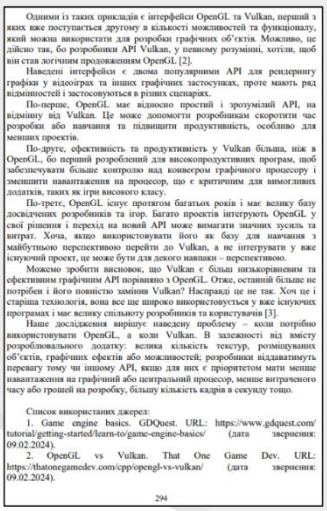
Для моделювання будь-якої графічної сцени потрібно потужне графічне обладнання, а також API для її розробки. Саме вибір останнього та підхід до його використання грає вирішальну роль у відтворенні графіки за допомогою графічного обладнання. І відповідає за це саме розробник програми, в якої він хоче використати свої вміння розробляти графіку.

На сьогодні є безліч інтерфейсів програмування застосунків (API) за допомогою для моделювання графічних моделей чи об'єктів. Велика роль відводиться саме розробнику, бо він пише код, реалізує логіку в програмі та оптимізує її певними функціями або способи, які пропонує обраний API. Але якщо інший інтерфейс пропонує схожий функціонал, але іншу реалізацію, то одне і теж моделювання такої самої сцени може дати різну статистику про рендеринг сцени, тобто швидкість моделювання одного кадру в секунду, займятість відеопам'яті тощо.

Таким чином, правильним рішенням є обрати такий API, який дає найбільшу продуктивність. Але для досягнення цієї мети потрібно докласти неабияких зусиль, бо такі інтерфейси моделювання комп'ютерної графіки хоч і дають більше продуктивності, вони потребують від розробника більшого розуміння та досвіду в розробці рушіїв з використанням певного графічного інтерфейсу через те, що вони стають більш низькорівневими [1].

А чи варто обирати важкий для розуміння API для моделювання невеликих сцен або коли просто відтворюється звичайне вікно з кнопками? Це питання лежить на плечах розробника(ів), бо тільки вони визначають, чи потрібна для власного проекту висока оптимізація, чи через його невелике навантаження необхідність у ній зникає.

293



Одними із таких прикладів є інтерфейси OpenGL та Vulkan, перший з яких вже поступається другому в кількості можливостей та функціоналу, який можна використати для розробки графічних об'єктів. Можливо, це дійсно так, бо розробники API Vulkan, у певному розумінні, хотіли, щоб він став логічним продовженням OpenGL [2].

Наведені інтерфейси є двома популярними API для рендерингу графіки у відеоіграх та інших графічних застосунках, проте мають ряд відмінностей і застосовуються в різних сценаріях.

По-перше, OpenGL має відносно простий і зрозумілий API, на відміну від Vulkan. Це може допомогти розробникам скоротити час розробки або навчання та підвищити продуктивність, особливо для менших проектів.

По-друге, ефективність та продуктивність у Vulkan більша, ніж в OpenGL, бо перший розроблений для високопродуктивних програм, щоб забезпечувати більше контролю над конвеєром графічного процесору і зменшити навантаження на процесор, що є критичним для вимогливих додатків, таких як ігри високого класу.

По-третє, OpenGL існує протягом багатьох років і має велику базу досвідчених розробників та ігор. Багато проектів інтегрують OpenGL у свої рішення і перехід на новий API може вимагати значних зусиль та витрат. Хоча, якщо використовувати його як базу для навчання з майбутньою перспективою перейти до Vulkan, а не інтегрувати у вже існуючий проект, це може бути для декого навпаки – перспективно.

Можемо зробити висновок, що Vulkan є більш низькорівневим та ефективним графічним API порівняно з OpenGL. Отже, останній більше не потрібен і його повністю замінив Vulkan? Насправді це не так. Хоч це і старіша технологія, вона все ще широко використовується у вже існуючих програмах і має велику спільноту розробників та користувачів [3].

Наше дослідження вирішує наведену проблему – коли потрібно використовувати OpenGL, а коли Vulkan. В залежності від вмісту розроблювального додатку: велика кількість текстур, розміщуваних об'єктів, графічних ефектів або можливостей; розробники віддаватимуть перевагу тому чи іншому API, якщо для них є пріоритетом мати менше навантаження на графічний або центральний процесор, менше витраченого часу або грошей на розробку, більшу кількість кадрів в секунду тощо.

Список використаних джерел:

1. Game engine basics. GDQuest. URL: https://www.gdquest.com/tutorial/getting-started/learn-to/game-engine-basics/ (дата звернення: 09.02.2024).

2. OpenGL vs Vulkan. That One Game Dev. URL: https://thatonegamedev.com/cpp/opengl-vs-vulkan/ (дата звернення: 09.02.2024).

294

18

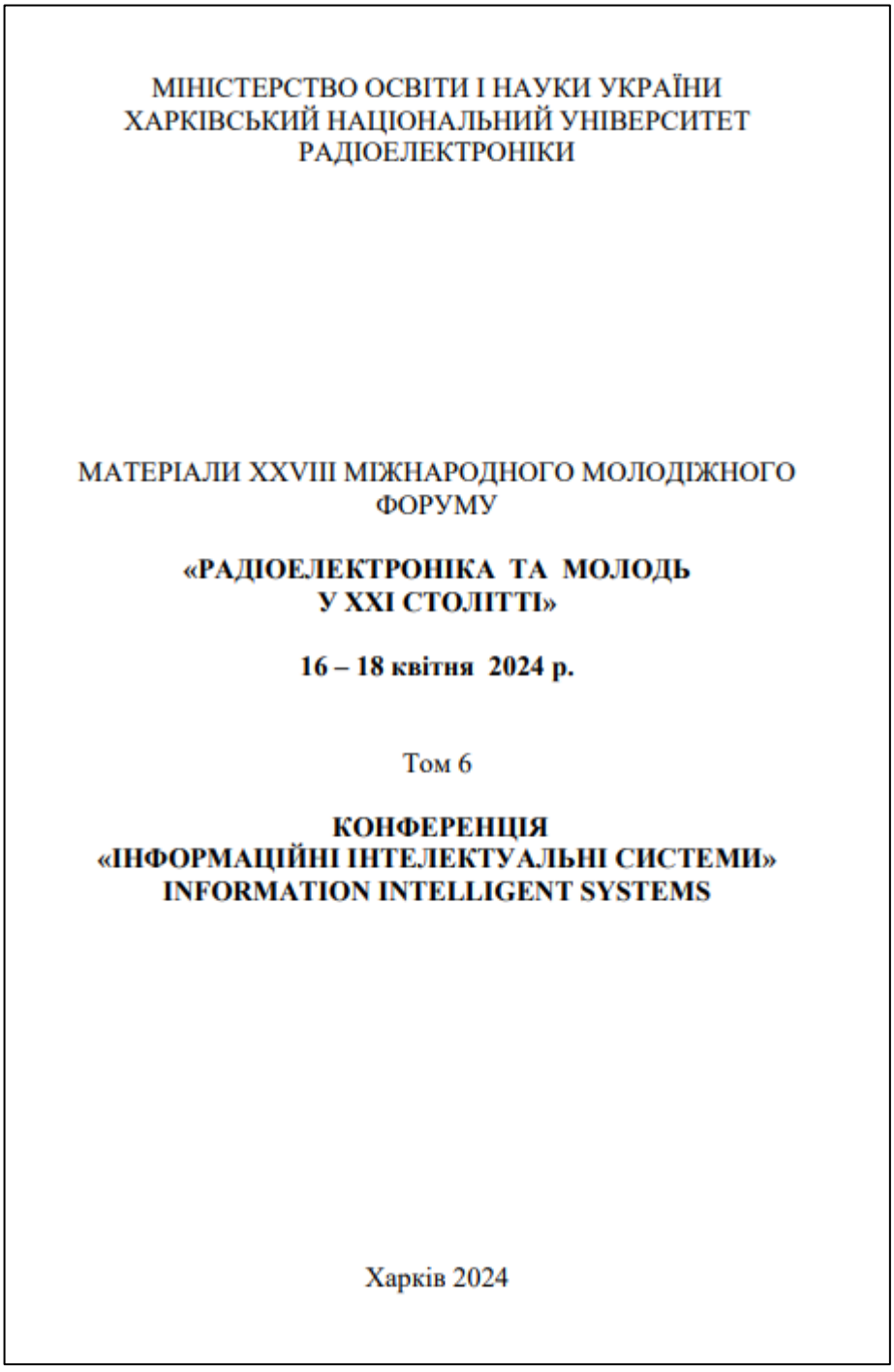
Висновки

Були визначені методи обох інтерфейсів, їх сильні та слабкі сторони. Це дозволило краще зрозуміти, які методи можуть бути найбільш ефективними в конкретних сценаріях розробки графічних додатків.

Результат роботи можна розглядати як досягнення мети аналізу методів для підвищення оптимізації. Також дає можливість розробникам виявити потребу переходу готових продуктів з інтерфейсу OpenGL до Vulkan або потребу обрати певний в залежності від швидкості розробки або продуктивності візуалізації програми. Це відкриває шлях для подальших досліджень та розвитку у цій сфері з метою покращення продуктивності та ефективності графічних додатків.

ДОДАТОК Г

Апробація у вигляді тез у 28-му Міжнародному молодіжному форумі «РАДІОЕЛЕКТРОНІКА І МОЛОДЬ У XXI СТОЛІТТІ»



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
РАДІОЕЛЕКТРОНІКИ

МАТЕРІАЛИ XXVIII МІЖНАРОДНОГО МОЛОДІЖНОГО ФОРУМУ

«РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ У XXI СТОЛІТТІ»

16 – 18 квітня 2024 р.

Том 6

КОНФЕРЕНЦІЯ
«ІНФОРМАЦІЙНІ ІНТЕЛЕКТУАЛЬНІ СИСТЕМИ»
INFORMATION INTELLIGENT SYSTEMS

Харків 2024

Рисунок Г.1 – Титульна сторінка форуму

УДК 004.92:004.428 DOI: https://doi.org/10.30837/IYF.IIS.2024.293

ОПТИМАЛЬНЕ ВИКОРИСТАННЯ OPENGL ТА VULKAN

Цапко Б. В.
Науковий керівник – к.т.н., доцент Чуприна А. С.
Харківський національний університет радіоелектроніки, каф. ПІ
м. Харків, Україна
e-mail: bohdan.tsapko@nure.ua

The purpose of this work is to make the optimal choice between the OpenGL and Vulkan APIs depending on the needs of developers to obtain the required performance characteristics based on the implementation of similar optimization methods in both APIs. The methods of graphic modeling, their implementation and methods of reproduction, various approaches to the optimization of graphic modeling for certain methods of visualization and processing of graphic objects are considered. Techniques such as using shaders, optimizing textures and buffers have been thoroughly reviewed and compared in terms of performance and how they are implemented in the OpenGL and Vulkan APIs.

Для моделювання будь-якої графічної сцени потрібно потужне графічне обладнання, а також API для її розробки. Саме вибір останнього та підхід до його використання грає вирішальну роль у відтворенні графіки за допомогою графічного обладнання. І відповідає за це саме розробник програми, в якої він хоче використати свої вміння розроблювати графіку.

На сьогодні є безліч інтерфейсів програмування застосунків (API) за допомогою для моделювання графічних моделей чи об'єктів. Велика роль відводиться саме розробнику, бо він пише код, реалізує логіку в програмі та оптимізує її певними функціями або способи, які пропонує обраний API. Але якщо інший інтерфейс пропонує схожий функціонал, але іншу реалізацію, то одне і теж моделювання такої самої сцени може дати різну статистику про рендеринг сцени, тобто швидкість моделювання одного кадру в секунду, зайнятість відеопам'яті тощо.

Таким чином, правильним рішенням є обрати такий API, який дає найбільшу продуктивність. Але для досягнення цієї мети потрібно докласти неабиякі зусилля, бо такі інтерфейси моделювання комп'ютерної графіки хоч і дають більше продуктивності, вони потребують від розробника більшого розуміння та досвіду в розробці рушіїв з використанням певного графічного інтерфейсу через те, що вони стають більш низькорівневими [1].

А чи варто обирати важкий для розуміння API для моделювання невеликих сцен або коли просто відтворюється звичайне вікно з кнопками? Це питання лежить на плечах розробника(ів), бо тільки вони визначають, чи потрібна для власного проекту висока оптимізація, чи через його невелике навантаження необхідність у ній зникає.

Рисунок Г.2 – Перша сторінка тез

Одними із таких прикладів є інтерфейси OpenGL та Vulkan, перший з яких вже поступається другому в кількості можливостей та функціоналу, який можна використати для розробки графічних об'єктів. Можливо, це дійсно так, бо розробники API Vulkan, у певному розумінні, хотіли, щоб він став логічним продовженням OpenGL [2].

Наведені інтерфейси є двома популярними API для рендерингу графіки у відеоіграх та інших графічних застосунках, проте мають ряд відмінностей і застосовуються в різних сценаріях.

По-перше, OpenGL має відносно простий і зрозумілий API, на відмінну від Vulkan. Це може допомогти розробникам скоротити час розробки або навчання та підвищити продуктивність, особливо для менших проектів.

По-друге, ефективність та продуктивність у Vulkan більша, ніж в OpenGL, бо перший розроблений для високопродуктивних програм, щоб забезпечувати більше контролю над конвеєром графічного процесору і зменшити навантаження на процесор, що є критичним для вимогливих додатків, таких як ігри високого класу.

По-третє, OpenGL існує протягом багатьох років і має велику базу досвідчених розробників та ігор. Багато проектів інтегрують OpenGL у свої рішення і перехід на новий API може вимагати значних зусиль та витрат. Хоча, якщо використовувати його як базу для навчання з майбутньою перспективою перейти до Vulkan, а не інтегрувати у вже існуючий проект, це може бути для декого навпаки – перспективою.

Можемо зробити висновок, що Vulkan є більш низькорівневим та ефективним графічним API порівняно з OpenGL. Отже, останній більше не потрібен і його повністю замінив Vulkan? Насправді це не так. Хоч це і старіша технологія, вона все ще широко використовується у вже існуючих програмах і має велику спільноту розробників та користувачів [3].

Наше дослідження вирішує наведену проблему – коли потрібно використовувати OpenGL, а коли Vulkan. В залежності від вмісту розроблювального додатку: велика кількість текстур, розміщуваних об'єктів, графічних ефектів або можливостей; розробники віддаватимуть перевагу тому чи іншому API, якщо для них є пріоритетом мати менше навантаження на графічний або центральний процесор, менше витраченого часу або грошей на розробку, більшу кількість кадрів в секунду тощо.

Список використаних джерел:

1. Game engine basics. GDQuest. URL: https://www.gdquest.com/tutorial/getting-started/learn-to/game-engine-basics/ (дата звернення: 09.02.2024).

2. OpenGL vs Vulkan. That One Game Dev. URL: https://thatonegamedev.com/cpp/opengl-vs-vulkan/ (дата звернення: 09.02.2024).

294

Рисунок Г.3 – Друга сторінка тез

ДОДАТОК Д

Експертний висновок результатів перевірки кваліфікаційної роботи на відповідність оформлення вимогам ДСТУ 3008: 2015

Експертний висновок результатів перевірки кваліфікаційної роботи

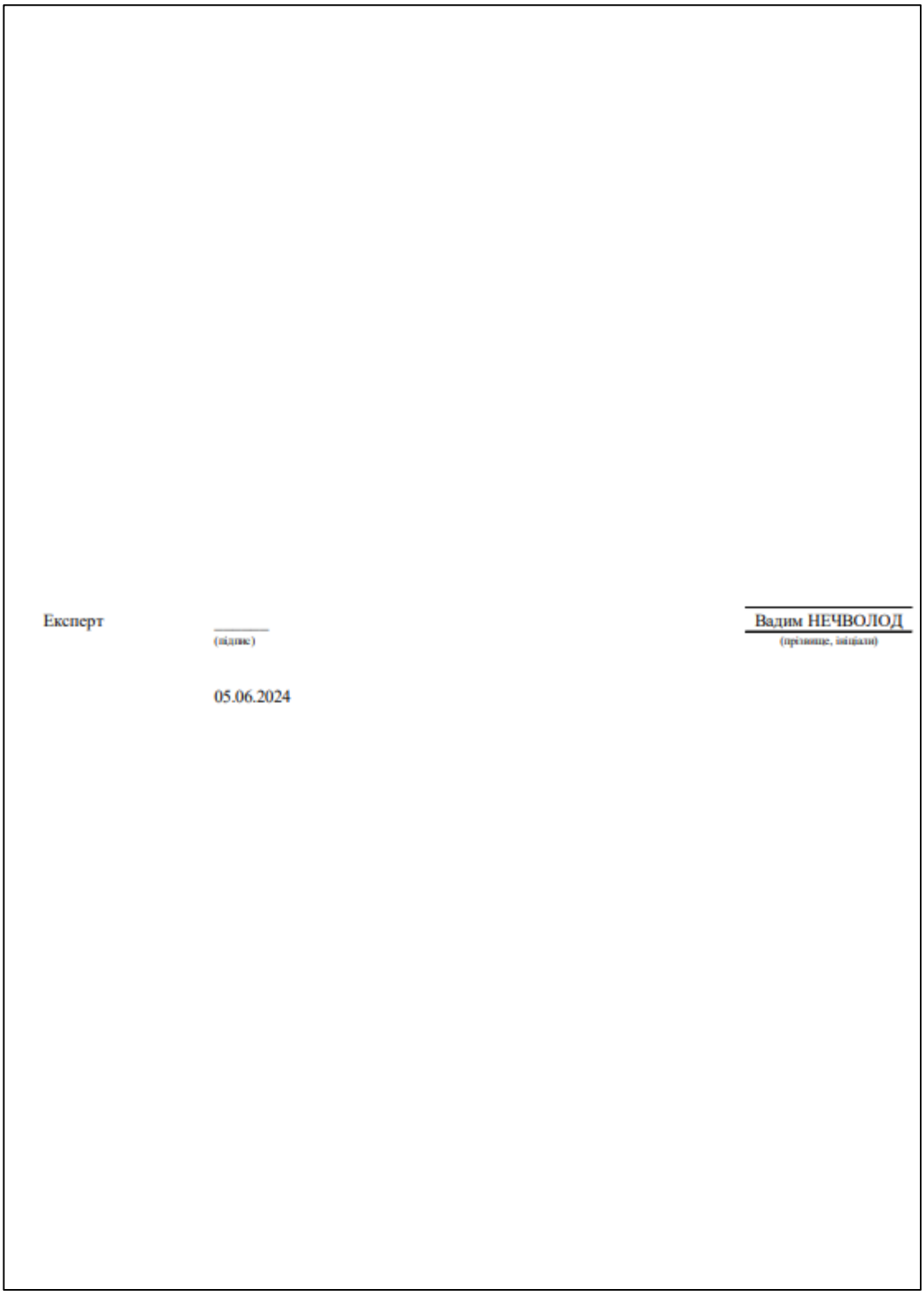
студент (посада) | програмної інженерії (кафедра) | ІПЗм-22-2 (група)

Цапко Б.В.
(прізвище, ім'я, по-батькові)

Зауваження

Пункт ДСТУ 3008-2015	Зміст пункту	Сторінка кваліфікаційної роботи
1	2	3
	7.1 Загальні положення	
	7.3 Нумерація сторінок звіту	
	7.4 Нумерація розділів, підрозділів, пунктів, підпунктів	
	7.5 Рисунки	
	7.6 Таблиці	
	7.7 Переліки	
	7.8 Примітки	
	7.9 Виноски	
	7.10 Формули та рівняння	
	7.11 Посилання	
	7.13 Список авторів	
	7.14 Скорочення та умовні познаки	
	7.15 Додатки	
Методичні вказівки до виконання кваліфікаційної роботи магістра… **ЗАТВЕРДЖЕНО** кафедрою ПІ протокол № 5 від 13.11.2023р. 3.2 Оформлення пояснювальної записки згідно з ДСТУ 3008:2015 Звіти у сфері науки і техніки. Структура та правила оформлювання. **Шаблон** затверджений засіданням кафедри №3 від 16.10.2023.	Рисунок повинен розміщуватися одразу після його згадування у тексті, або на наступній сторінці. **Під рисунком повинен бути підпис із словом Рисунок, порядковим номером цього рисунку, через тире з великої літери – назва рисунку та в круглих дужках вказується джерело з якого взятий цей рисунок, або то, що його виконано самостійно.**	25, далі за текстом
Методичні вказівки до виконання кваліфікаційної роботи магістра… **ЗАТВЕРДЖЕНО** кафедрою ПІ протокол № 5 від 13.11.2023р. 3.2 Оформлення пояснювальної записки згідно з ДСТУ 3008:2015 Звіти у сфері науки і техніки. Структура та правила оформлювання. **Шаблон** затверджений засіданням кафедри №3 від 16.10.2023.	Назву таблиці друкують з великої літери і розміщують над таблицею з абзацного відступу та в круглих дужках вказується джерело з якого взята ця рисунокаблиця. ПРИКЛАД: шаблон, стор.15	40, дали за текстом.

Рисунок Д.1 – Перша сторінка експертного висновку


Експерт _____ (підпис) Вадим НЕЧВОЛОД (прізвище, ініціали)

05.06.2024

Рисунок Д.2 – Друга сторінка експертного висновку