System Antyplagiat

Individual clients - tokens

REPORT DATE: 2020-12-05 21:54:53

TITLE:
DIPLOM_FOR_CHECK.docx [ID:7772091]

AUTHOR(S):
Илья Полищук

FILE UPLOAD DATE:
2020-12-05 21:53:08

SKIPPED URL ADDRESSES:

Record of similarities

Please note that high coefficient values do not automatically mean plagiarism. The report must be analyzed by an authorized person.

10.58% SC1

% of 5-words combination found in all available sources except LAD

9.60% SC2

% of 25-words combination found in all available sources except LAD

25	14388	102433
The phrase length for the SC 2	Length in words	Length in characters

List of possible text manipulation attempts

In this section, you can find information regarding text modifications that may aim at temper with the analysis results. Invisible to the person evaluating the content of the document on a printout or in a file, they influence the phrases compared during text analysis (by causing intended misspellings) to conceal borrowings as well as to falsify values in the Similarity Report. It should be assessed whether the modifications are intentional or not.

Characters from another alphabet number of characters from other alphabets which may imitate letters from the alphabet relevant to the document, causing misspellings in the text, please verify their validity	0	show in the text
Spreads number of increased distances between letters - please verify whether they imitate spaces, indicating as merging the words in the Report	0	show in the text
Micro spaces number of spaces with zero length - please verify whether they are placed inside words and cause word division in the text	0	show in the text
White characters number of characters with a white font color - please verify whether they are used instead of spaces, causing merge of the words (in the Report the color of the letters is changed to black in order to show them)	0	show in the text

Active lists of similarities

Scroll the list and analyze especially the fragments that exceed the SC 2 (marked in bold). Use the link "Mark fragment" and see if they are short phrases scattered in the document (coincidental similarities), numerous short phrases near each other (mosaic plagiarism) or extensive fragments without indicating the source (direct plagiarism).

The 10 longest fragments (7.42 %)

Ten longest fragments found in all available resources.

NO	TITLE OR SOURCE URL (DATABASE)	AUTHOR(S)	NUMBER OF IDENTICAL WORDS	
1	https://xreferat.com/33/4358-1-programuvannya-nterfeiysu.html		215	1.49 %
2	https://xreferat.com/33/4358-1-programuvannya-nterfeiysu.html		176	1.22 %
3	https://koloro.ua/ua/blog/3d-tekhnologii/3d-model-vidy-urovni-slozhnosti-sostavnye-chasti.html		145	1.01 %
4	https://xreferat.com/33/4358-1-programuvannya-nterfeiysu.html		110	0.76 %
5	https://koloro.ua/ua/blog/3d-tekhnologii/3d-model-vidy-urovni-slozhnosti-sostavnye-chasti.html		107	0.74 %
6	https://uk.m.wikipedia.org/wiki/Make		85	0.59 %
7	http://www.scs.kpi.ua/sites/default/files/files/2017/%D0%91%D0%B0%D0%BA%D0%B0%D0%BB%D0%B0%D0%B2%D1%80%D0%B8/%D0%9A%D0%B0%D0%BC%D0%BF%D0%BE%D0%B2.pdf		82	0.57 %
8	https://dynamic-design.com.ua/novosti/uk/versia-wikizero/		61	0.42 %
9	https://docmia.com/d/272496		47	0.33 %
10	https://sites.google.com/site/tspp20141218/home		40	0.28 %

from the Internet (10.58 %)

All fragments found in the open access global internet resources.

NO	SOURCE URL	NUMBER OF IDENTICAL WORDS (FRAGMENTS)	
1	https://xreferat.com/33/4358-1-programuvannya-nterfeiysu.html	501 (3)	3.48 %
2	https://koloro.ua/ua/blog/3d-tekhnologii/3d-model-vidy-urovni-slozhnosti-sostavnye-chasti.html	283 (3)	1.97 %
3	https://sites.google.com/site/tspp20141218/home	122 (7)	0.85 %
4	https://dynamic-design.com.ua/novosti/uk/versia-wikizero/	96 (2)	0.67 %
5	https://uk.m.wikipedia.org/wiki/Make	85 (1)	0.59 %
6	http://www.scs.kpi.ua/sites/default/files/files/2017/%D0%91%D0%B0%D0%BA%D0%B0%D0%BB%D0%B0%D0%B2%D1%80%D0%B8/%D0%9A%D0%B0%D0%BC%D0%BF%D0%BE%D0%B2.pdf	82 (1)	0.57 %
7	https://docmia.com/d/272496	69 (2)	0.48 %
8	https://e-tk.lntu.edu.ua/mod/page/view.php?id=4659	64 (2)	0.44 %
9	https://www.wikizero.com/uk/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BA%D0%B5%D1%80%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D1%96%D1%8F%D0%BC%D0%B8	43 (2)	0.30 %
10	https://prezi.com/pk3tvmox3t30/linux/	41 (3)	0.28 %
11	https://uk.gadget-info.com/difference-between-preemptive	38 (2)	0.26 %
12	https://echo.lviv.ua/dev/5394	31 (1)	0.22 %
13	http://bestwebit.biz.ua/pages_02/about_github.php	28 (1)	0.19 %
14	https://f.ua/ua/kingston/ddr4-16gb-3200mhz-pc4-25600-hyperx-predator-black-hx432c16pb3-16.html	20 (2)	0.14 %
15	https://www.vario.bg/gigabyte-geforce-rtx-2070-super-gaming-oc-8g-gv-n207sgaming-oc-8gc	14 (1)	0.10 %
16	https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B2%D0%B0%D0%BD%D1%82%D0%B0%D0%B6%D1%83%D0%B2%D0%B0%D0%BD%D0%B8%D0%B9_%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D1%8C_%D1%8F%D0%B4%D1%80%D0%B0	5 (1)	0.03 %

Verified content - similarities are marked in the text below according to the source:

Please take note of the fact that the system does not give a verdict. If any suspicions arise, the Similarity Report should be subjected to a thorough analysis.

LEGEND

- Internet Sources

РЕФЕРАТ

Об’єктом цього дослідження є відомі програмні інтерфейси OpenGL та Vulkan для роботи з комп’ютерною графікою.

Предметом дослідження є оцінка ефективності впливу збільшення кількості ядер центрального процесору на середній час відображення одного кадру комп’ютерних анімації.

Метою роботи є визначення рівня впливу кількості ядер процесору на час відображення кадрів комп’ютерних анімації з використанням OpenGL та Vulkan.

Результати та їх новизна: дослідження робить внесок у визначення практичної застосовності програмного інтерфейсу Vulkan разом із збільшенням числа процесорних ядер для зменшення часу потрібного на відображення кадрів анімації.

Пояснювальна записка складається зі вступу, 5 розділів, висновків, бібліографічного списку та 4 додатків:

Вступ - описується сутність розробки, її актуальність (3 сторінки).

у першому розділі висвітлено аналіз сучасного стану дослідження проблеми за науковими літературними джерелами також проаналізовано сучасний стану програмно-апаратного забезпечення. Складається з 15 сторінок;

у другому розділі надано обґрунтування експериментального методу дослідження. Складається з 16 сторінок;

у третьому розділі представлене проектування й розробка інструментального забезпечення для дослідження. Складається з 21 сторінки;

у четвертому розділі описано виконані дослідження. Складається з 14 сторінок;

у п'ятому розділі розкриті питання охорона та безпеки праці. Складається з 8 сторінок;

додатки містять технічне завдання й робочий проект

Таблиць - 15, рисунків -21, бібліографія - 74 джерела.

Ключові слова: графіка реального часу, Vulkan, OpenGL, анімації, багатоядерні системи.

ВСТУП

Передові досягнення науки і техніки в області комп’ютерної анімації, такі як імітація руху або відбиття світла загалом є дуже критичними до часу виконання завдань, які на них покладаються. Задачі, які вирішують більшість систем відображення комп’ютерної графіки є задачами так званого жорсткого реального часу (коли перевищення часу виконання поставлених завдань може призвести до невідворотних наслідків) або м’якого реального часу (коли перевищення часу вирішення небажане, але припустиме).

Візуалізація в реальному часі відноситься до швидкого зображень на комп'ютері. Це найбільш інтерактивна область комп’ютерної графіки. На екрані з'являється зображення, глядач діє або реагує, і цей відгук впливає на те, що генерується далі. Цей цикл реакції та візуалізації відбувається з досить швидкою швидкістю, щоб глядач не бачив окремі зображення, а навпаки, занурювався в динамічний процес.

Саме тому ми звертаємо нашу увагу на можливість розподілу таких задач на декілька ядер процесору за для зменшення загального часу обробки кожного кадру анімації перед його відображенням.
Для виводу графіки на екран використовують центральний процесор та графічний процесор. Центральний процесор обчислює позицію, зміщення, форму об’єктів та передає ці дані до графічного процесору. Графічний процесор у свою чергу перетворює дані таких об’єктів у форму придатну для подальшого виведення на екран.
Актуальність роботи. З давніх часів центральний процесор та графічний процесор розвивалися окремо один від одного. Тому зараз, коли центральний процесор досяг успіху у паралельному виконанні задач на декількох ядрах, підходи до обробки та передачі даних на графічний процесор не є ефективними.
Один з таких підходів є використання OpenGL - програмного інтерфейсу до графічного пристрою. Цей інтерфейс розроблявся у 90-их років, коли багатоядерні процесори ще не були досить популярними. Тому методи до обробки та передачі даних які запроваджує цей інтерфейс не дає змогу робити це паралельно, оскільки будь-яка операція з OpenGL може змінювати його глобальний стан, який не передбачає одночасні зміни і не запроваджує синхронізації щоб захиститися від стана гонитви (ситуація в якій робота чи результат операції залежить від послідовності або тривалості виконання).
Щоб виправити такі недоліки може бути використаний інтерфейс Vulkan. Цей інтерфейс не використовує глобальних об’єктів які неможливо синхронізувати. На томість робота з ним є складніша, оскільки об’єкти які були сховані у OpenGL відтепер мають бути створені та використані розробником. Одним із таких об’єктів є командний буфер. Командний буфер це буфер, який зберігає заздалегідь заповнені команди які має виконати графічний процесор. Кожен командний буфер може використовуватись незалежно один від одного, тому кожен потік може обчислювати інформацію об’єктів та заповнювати такі буфери окремо. У кінці, ці буфери мають бути відправлені до графічного процесору, де з них буде створене зображення.
Об’єкт дослідження. Об’єктом дослідження є методи обробки та відображення анімованих моделей використовуючи для цього декілька ядер процесору.
Предмет дослідження. Вдосконалення процесу розробки програмного забезпечення на основі розробки через поведінку звичайного користувача.
Мета і завдання дослідження. В процесі дослідження буде з’ясовано як саме буде змінюватись ефективність відображення анімації використанням OpenGL та Vulkan. А саме:
Як можливі дає Vulkan з обробки об’єктів анімації паралельно на процесорі.
Які методи синхронізації мають бути використані за для такої паралельної обробки.
Залежність ефективності від кількості об’єктів анімації.
Залежність ефективності від кількості ядер процесору.
Наукова новизна. Одночасне використання багатьох ядер процесору пришвидшує обробку кадрів зображень для анімації та робить можливим одночасно відображати більше анімаційних об’єктів, таким чином що би кількість кадрів за секунду була якомога плавніше, хоча би 24 кадрів за секунду, за якими людське око може розрізняти плавність руху.
Практичне значення. Практичне значення роботи викладене наступним чином. Результат проведених досліджень дозволить оцінити продуктивність методу багатоядерного відображення анімаційних об’єктів та в подальшому може бути основою для нових досліджень в даній сфері.
Апробація результатів дослідження. Процес та результати дослідницької роботи доповідались на семінарі кафедри КІТ 09.10.2020р.
Публікації за темою роботи. Підготовлено статтю «Дослідження часової ефективності відображення комп`ютерної графіки реального часу на багатоядерних системах з використанням OpenGL та Vulkan API» до видання у фаховому журналі.

1 АНАЛІЗ ПРОБЛЕМИ МАСШТАБУВАННЯ ОБРОБКИ АНІМАЦІЇ НА БАГАТОЯДЕРНИХ ПРОЦЕСОРАХ

1.1 Історія розвитку методів програмування комп’ютерної графіки

Ранні відеокарти практично не мали пов'язаного з ними коду, що викликається, поняття "драйвери" ще не зовсім стало реальністю. Існувала концепція Video BIOS, яка була розширенням відеосервісів INT 10h BIOS, які фактично обмежувались ініціалізацією та перемиканням режимів відео [1].
Натомість усі графічні карти, принаймні в DOS, мали відображену пам’ять оперативної пам’яті, і була доступна велика документація про те, як саме встановлення різних бітів в оперативній пам’яті вплине на пікселі, що з’являлися на екрані. Не було API для малювання для виклику, якщо ви хочете, щоб щось з’явилося на екрані (будь то піксель, символ, рядок, коло, тощо), ви б написали код для переміщення байтів вправо місця в оперативній пам'яті. Були написані цілі книги про те, як правильно писати код для малювання графіки.
Відеопам'ять була відображена з сегментом A0000h. В реальному режимі середовища DOS ви можете просто сформувати покажчик на цей сегмент і обробити його як 64000-байтний масив, кожен байт відповідає одному пікселю. Встановіть для байта нове значення, піксель змінює колір на екрані. Крім того, реалізація будь-яких функцій малювання вищого рівня залежить від вас або від сторонніх бібліотек [1].
Існували деякі системи, такі як графічний інтерфейс Borland, які абстрагували примітиви графічного малювання в API з різними драйверами, які можна було викликати, щоб малювати речі на різних графічних картах. Однак, як правило, вони працювали повільніше, ніж потрібно для побудови ігор або анімацій.
Дія гри, як правило, оптимізується для певного режиму графічного відображення на певній карті. Наприклад, популярним режимом відображення був VGA 640x480 з 16 кольорами. Це буде вказано у вимогах до програмного забезпечення, і вам потрібно мати відповідне обладнання для підтримки гри. Якщо ви придбали гру VGA, але у вас була лише карта EGA, то гра не працювала б взагалі.
У 80-х і на початку 90-х вам потрібно було зробити все на процесорі, а потім використовувати API відеокарти для показу 2D-зображення [1].

1.2 Огляд старих програмних інтерфейсів комп’ютерної графіки

На заміну старим методам програмування комп’ютерної графіки прийшли програмні інтерфейсі OpenGL та DirectX. На відміну від MS-DOS, ці інтерфейси не надають прямого доступу до пам’яті відеокарти, що негативно впливає на швидкість роботи з нею. Але на томість ці інтерфейси пропонують стандартизовані методи роботи з 2Д та 3Д графікою. Завдяки такій стандартизації, розробники відеокарт здобули можливість розробити алгоритми обробки зображення які працюють на відеочіпі, а не на процесорі. І розробники мають змогу використовувати одні і тіж виклики до цих інтерфейсів, щоб використовувати їх різні реалізації на від різних постачальників відеокарт. Це надало можливість розробляти такі програми,

які будуть працювати на усіх відеочипах, до яких постачальник розробив реалізацію того чи іншого стандартизованого програмного інтерфейсу.

1.2.1 Огляд інтерфейсу OpenGL

OpenGL - це інтерфейс прикладного програмування, коротше API, який є просто бібліотекою програмного забезпечення для доступу до функцій графічного обладнання. Містить понад 500 різних команд, які використовуються для вказівки об'єктів, зображень та операції, необхідних для створення інтерактивних тривимірних комп'ютерно-графічних додатків.

OpenGL розроблений як спрощений, апаратно-незалежний інтерфейс, який можна реалізувати на багатьох різних типів графічних апаратних систем або повністю в програмному забезпеченні (якщо жодного графічного обладнання немає в системі).

Як такий, OpenGL не включає функції для виконання завдань вікна або обробки вводу користувача; натомість, потрібно буде використовувати засоби, передбачені операційною системою, де додаток буде виконано [2].

Так само OpenGL не надає жодних функцій для опису моделей тривимірних об'єктів або операції зчитування файлів зображень (наприклад, файли JPEG). Натомість, ви повинні побудувати свої тривимірні об'єкти з невеликого набору геометричних примітивів: точок, лінії та трикутників.

З тих пір, як OpenGL існує деякий час - він був вперше розроблений у Silicon Graphics Computer Systems, з версією 1.0, випущеною в липні 1994 р. - існує також безліч бібліотек програм, побудованих з використанням OpenGL для спрощення розробки додатків, наприклад написання відеоігри, створення візуалізації для наукових чи медичних цілей [2].

OpenGL реалізований як система клієнт-сервер, додаток розглядається як клієнт та реалізація OpenGL, надана виробником вашої комп'ютерної графіки обладнання, як сервер. У деяких реалізаціях OpenGL (наприклад, пов'язаних з X Window System), клієнт і сервер можуть виконуватися на різних машинах, підключених до мережі. У таких випадках клієнт видає команди OpenGL, які будуть перетворені в специфічний протокол для віконної системи, який передається на сервер через їх спільну мережу, де вони виконуються для отримання кінцевого зображення [2].

У більшості сучасних реалізацій для реалізації більшості функції OpenGL використовується апаратний графічний прискорювач встановлений на окремій платі та підключений до материнської плати комп'ютера. У будь-якому випадку це так розумно вважати клієнта своїм додатком, а сервер графічним прискорювачем (відеокартою).

1.2.2 Огляд інтерфейсу DirectX

DirectX, колекція інтерфейсів прикладного програмування (API) корпорації Майкрософт, призначена для надання розробникам низькорівневого інтерфейсу до апаратного забезпечення ПК, на якому працюють операційні системи на базі Windows. Кожен компонент забезпечує доступ до різних аспектів апаратного забезпечення, включаючи графіку, звук, обчислювальні технології GPU загального призначення та пристрої введення через стандартний інтерфейс.

Наявність одного стандартного API, якого повинні дотримуватися виробники обладнання, є набагато зручнішим, ніж написання шляхів коду для всіх можливих пристроїв на ринку, тим більше, що нові пристрої, випущені після доставки гри, можуть не розпізнаватися грою, тоді як використання стандартних рішень вирішують цю проблему. DirectX - це сукупність API, що використовуються в основному розробниками відеоігор для вирішення цієї потреби у стандартизації на платформах Windows та Xbox [3].

DirectX 11 є скоріше додатковим оновленням DirectX 10.1, а не основним оновленням DirectX 10, як було з DirectX 9. Microsoft ризикнула, почавши оновлення з Direct3D 10 і вимагаючи не лише нового обладнання, але і Windows Vista як мінімальну вимогу. Це було кілька років тому, і на сьогодні його використовують оскільки не тільки широко розповсюджена апаратна підтримка, але й більшість користувачів Windows зараз охоплюють Windows Vista та Windows 7. DirectX завжди враховував майбутнє, а також скільки років потрібно, щоб розробляти ігри наступного покоління, DirectX 11 буде дуже важливим для ігор на довгі роки [3].

Суперечки Direct3D проти OpenGL часто можуть здаватися релігійними, але справа в тому, що протягом багатьох років OpenGL відставав від Direct3D. Microsoft зробила велику роботу, розвиваючи Direct3D і вдосконалюючи її протягом багатьох років, але OpenGL лише відставав, не дотримуючись своїх обіцянок, оскільки кожна нова версія виходить, і раз за разом розробники страждали від тих самих проблем минулих років. Коли вперше було оголошено OpenGL 3.0, вважалося, що OpenGL нарешті повернеться в позицію, щоб конкурувати з Direct3D. На жаль, світ OpenGL пережив свою частку злетів і падінь, як у самій групі, що стояла за ним, так і через те, як API конкурував з Direct3D, а Direct3D продовжував домінувати [3].

1.3 Недоліки старих програмних інтерфейсів комп'ютерної графіки

З давніх часів центральний процесор та графічний процесор розвивалися окремо один від одного. Тому зараз, коли центральний процесор досяг успіху у паралельному виконанні задач на декількох ядрах, підходи до обробки та передачі даних на графічний процесор не є ефективними.

Старі інтерфейси розроблялися у 90-ті роки, коли багатоядерні процесори ще не були досить популярними. Тому методи до обробки та передачі даних які запроваджує цей інтерфейс не дає змогу робити це паралельно, оскільки будь-яка операція з ними може змінювати його глобальний стан, який не передбачає одночасні зміни і не запроваджує синхронізації щоб захиститися від стана гонитви (ситуація в якій робота чи результат операції залежить від послідовності або тривалості виконання).

Щоб виправити такі недоліки може були створені нові графічні програмні інтерфейси.

1.4 Огляд нових програмних інтерфейсів комп'ютерної графіки

Нові програмні інтерфейси для роботи з комп'ютерною графікою були створені з урахуванням того, що розробники повинні мати можливість розподіляти роботу по обробці та передачі даних на графічний процесор. Тому вони є більш складними, адже усією роботою з пам'яттю повинні керувати програмісти, які розробляють додатки з використанням цих інтерфейсів. Це зроблено для того, щоб користувачі мали можливість розподіляти роботу з пам'яттю графічного чипу по багатьом потокам, та синхронізувати доступ до цієї пам'яті максимально ефективно.

1.4.1 Огляд інтерфейсу DirectX12

DirectX 12 представляє наступну версію Direct3D - 3D-графічний API на основі DirectX. Direct3D 12 швидший та ефективніший, ніж будь-яка попередня версія. Direct3D 12 забезпечує розширені сцени, більше об'єктів, більш складні ефекти та повне використання сучасного обладнання GPU [4].

Direct3D 12 унікальний тим, що забезпечує нижчий рівень апаратної абстракції, ніж у попередніх версіях, що дозволяє значно покращити багатоядерне масштабування процесора вашого додатка. З одного боку,

з Direct3D 12 ваш додаток відповідає за власне управління пам’яттю. Крім того, за допомогою Direct3D 12 ваші додатки та програми отримують вигоду від зменшення накладних витрат на графічний процесор. Direct3D 12 надає розробникам графіки чотири основні переваги (порівняно з Direct3D 11) [4].
1. Значно зменшені накладні витрати на процесор.
2. Значно знижене енергоспоживання.
3. До (приблизно) 20% покращення ефективності графічного процесора.
4. Крос-платформна розробка для пристрою Windows 10 (ПК, планшет, консоль, мобільний).
Direct3D 12 призначений для використання досвідченими графічними програмістами. Це вимагає значної графічної експертизи та високого рівня тонкої настройки. Direct3D 12 розроблений для повного використання багатопоточності, обережної синхронізації процесора / графічного процесора, а також переходу та повторного використання ресурсів від однієї мети до іншої. Це методи, які вимагають значної кількості навичок програмування на рівні пам'яті [4].
Ще однією перевагою Direct3D 12 є його невеликий розмір API. Є близько 200 функцій; і приблизно одна третина з них роблять усі важкі роботи. Це означає, що ви, як розробник графіки, повинні мати можливість навчитися і освоїти повний набір API, не запам'ятовуючи занадто багато імен API.
Для проекту, який використовує всі переваги Direct3D 12, вам слід розробити спеціально налаштований двигун Direct3D 12 з нуля.
Якщо ви, як розробник графіки, розумієте використання та повторне використання ресурсів у ваших програмах, і ви можете скористатися цим, мінімізуючи завантаження та копіювання, тоді ви можете розробити та налаштувати високоефективний механізм для цих програм. Покращення продуктивності могло б бути дуже значним, звільнивши час процесора для збільшення продуктивності [4].
Інвестиції у програмування є значними, і вам слід розглянути можливість налагодження та інструментарію проекту з самого початку. Потоки, синхронізація та інші помилки синхронізації можуть бути складними.
1.4.2 Огляд інтерфейсу Metal
Metal був оголошений на Всесвітній конференції розробників (WWDC) 2 червня 2014 року і спочатку був доступний лише на графічних процесорах A7 або новіших версіях. Apple створила нову мову для програмування графічного процесора безпосередньо за допомогою функцій шейдерів. Це мова затінення металів (MSL), заснована на специфікації C ++ 11. Через рік у WWDC [5].
API продовжував розвиватися, і WWDC 2017 представив захоплюючу нову версію API: Metal 2. Metal 2 додає підтримку віртуальної реальності (VR), доповненої реальності (AR) та прискореного машинного навчання (ML) серед багатьох нових функцій. Осінь 2017 принесла нові оновлення для Metal, включаючи блоки зображень, затінення плиток та обмін нитками, які доступні на пристроях iOS на базі чіпа A11 Bionic, який поставляється з першим графічним процесором, коли-небудь розробленим власноруч від Apple [5].
Вибирайте метал, коли:
1. Ви хочете відображати 3D-моделі максимально ефективно.
2. Ви хочете, щоб ваша гра мала свій унікальний стиль, можливо, з індивідуальним освітленням і затіненням.
3. Ви будете виконувати інтенсивні процеси обробки даних, такі як обчислення та зміна кольору кожного пікселя на екрані кожного кадру, як це було б при обробці зображень та відео.
4. У вас є великі числові проблеми, такі як наукове моделювання, які ви можете розділити на незалежні підзадачі, які будуть оброблятися паралельно.
5. Вам потрібно обробляти декілька великих наборів даних паралельно, наприклад, коли ви тренуєте моделі для глибокого навчання
1.4.3 Огляд інтерфейсу Vulkan
Vulkan - це інтерфейс програмування для графіки та обчислювальних пристроїв. Типово пристрій Vulkan складається з процесора та ряду апаратних блоків із фіксованою функцією для прискорення використовуваних операцій в графіці та обчисленні. Процесор в пристрої, як правило, дуже широкий багатопоточний процесор і тому обчислювальна модель у Вулкані значною мірою заснована на паралельних обчисленнях. Пристрій Вулкан також має доступ до пам'яті, яка може надаватися спільно з основним процесором, на якому ваш додаток запущено. Вулкан також дає контроль над цією пам’яттю вам [6].
Vulkan - це явний API. Тобто майже все - це ваша відповідальність. Драйвер - це програмне забезпечення, яке приймає команди та дані, що формують API, і перетворює їх у щось, що може зрозуміти обладнання. У старих API, таких як OpenGL, драйвери відстежували стан багатьох об’єктів, керували пам’яттю та синхронізацією та перевіряли наявність помилок у вашій програмі під час її роботи. Це чудово підходить для розробників, але спалює цінний час процесора, як тільки ваша програма буде налагоджена та працює належним чином. Vulkan вирішує цю проблему, передаючи майже все відстеження стану, синхронізацію та управління пам’яттю в руки розробника додатків та делегуючи перевірку правильності розширенням, які повинні бути включені. Ці розширення погано впливають на продуктивність, тому мають бути відключені у фінальних збірках.
З цих причин Вулкан і дуже багатослівний, і дещо тендітний. Вам потрібно зробити дуже багато парці, щоб Vulkan працював добре, і неправильне використання API часто може призвести до поломки зображення або навіть збою програми, де у старих API ви могли б отримати корисну помилку повідомлення. В обмін на це Vulkan забезпечує більше контролю над пристроєм, чисту модель і набагато вищу продуктивність, ніж API, які вона замінює [6].
Крім того, Vulkan був розроблений як дещо більше, ніж графічний API. Він може бути використаний для різнорідних пристроїв, таких як графічні процесори (GPU), цифрові процесори сигналів (DSP) та устаткування з фіксованою функцією. Поточна редакція Vulkan визначає категорію передачі, яка використовується для копіювання даних навколо; категорія обчислень, яка використовується для запуску шейдерів над обчислюваними робочими навантаженнями; і категорія графіки, що включає растеризацію, примітивну збірку, змішування, глибинні та трафаретні тести та інші функції, які будуть знайомі програмістам графіки [6].
Якщо підтримка для кожної категорії необов’язкова, можливо мати пристрій Vulkan, який взагалі не підтримує графіку. Як наслідок, навіть API для розміщення зображень на пристрої відображення (що називається презентацією) є не тільки необов’язковими, але вони надаються як розширення Vulkan, а не є частиною основного API [6].
1.5 Постановка задачі
У цій дипломній роботі для порівняння ефективності графічних інтерфейсів на багатоядерних системах

буде застосовано старий інтерфейс OpenGL та новий інтерфейс Vulkan оскільки вони є відкритими, та підтримують усі сучасні операційні системи, включно з операційними системами мобільних пристроїв. На відміну від OpenGL, Інтерфейс Vulkan не використовує глобальних об'єктів які неможливо синхронізувати.
Одним із таких об'єктів є командний буфер. Командний буфер це буфер, який зберігає заздалегідь заповнені команди які має виконати графічний процесор. Кожен командний буфер може використовуватись незалежно один від одного, тому кожен потік може обчислювати інформацію об'єктів та заповнювати такі буфери окремо. У кінці, ці буфери мають бути відправлені до графічного процесору, де з них буде створене зображення.
У таблиці 1.1 наведені відмінності OpenGL та Vulkan:
OpenGL Vulkan
Багатоплатформовий - підтримується на Windows та Linux. Для мобільних пристроїв використовується окрема підмножина специфікації OpenGL Багатоплатформовий - підтримується на Windows, Linux і мобільних пристроїв використовуючи одну і ту ж саму специфікацією
Перевіряє вхідні данні та повертає коди помилок в обмін на зниження продуктивності. Не перевіряє вхідні дані. Не дає опису чи гарантій що програма буде працювати і як вона буде працювати
Використовує глобальний стан, який є невеликою машиною станів. Будує напрямок своєї роботи використовуючи надані дані. Не має глобального стану. Розробник сам має проробити усі деталі для свого додатку
Не дає доступу до керування пам'яттю. Явний контроль над керуванням пам'яттю.

Таблиця 1.1 - Відмінності OpenGL та Vulkan
В процесі дослідження має бути з'ясовано, як саме буде змінюватись ефективність відображення анімації з використанням OpenGL та Vulkan. А саме:
1. Які можливі дає Vulkan з обробки об'єктів анімації паралельно на процесорі.
2. Які методи синхронізації мають бути використані за для такої паралельної обробки.
3. Залежність ефективності від кількості об'єктів анімації.
4. Залежність ефективності від кількості ядер процесору.
1.6 Огляд літератури
В даній роботі використана технічна література в якій висвітлюються основні теоретичні основи, що були застосовані при розробці відповідних компонентів системи.
Під час виконання огляду літератури розглядалися наступні питання: основні підходи і методології, що використовуються при проектуванні взаємодії додатків з програмними інтерфейсами комп'ютерної графіки, принципи, що застосовуються при розробці сучасних інтерактивних графічних програм, загальні вимоги до системи та їх різновиди, проблеми оптимізації, найбільш розповсюдженні методи, що використовуються у системах.
Існує дуже багато методів проектування відображення об'єктів комп'ютерної графіки наведені в матеріалі [7]. Ця книга ретельно зосереджується на сучасних методиках, що використовуються для створення синтетичних тривимірних зображень за частки секунд. З появою програмованих шейдерів (маленьких програм зі своєю мовою програмування, що виконуються на графічному процесорі) протягом останніх кількох років виникло і розвивалось широке коло нових алгоритмів. У цьому виданні розглядаються сучасні практичні методи візуалізації, що використовуються в іграх та інших додатках, такі як глобальне висвітлення та методи вибракування, а також пропонує багато детальної інформації про тонкі проблеми, такі як невеликі, але важливі відмінності між форматами текстур. Ця книга також не є керівництвом з програмування - оскільки вона перевищила базовий рівень, вона передбачає, що читач досить добре знайомий із своїм середовищем програмування графіки, щоб мати змогу реалізувати описані методи, не потребуючи покрокових інструкцій. Також представляє міцну теоретичну базу та відповідну математику для галузі інтерактивної комп'ютерної графіки, все у доступному стилі.
Основні підходи та поняття для розробки графічних програм за допомогою Vulkan описані в матеріалі [6]. Усередині є огляд Vulkan на високому рівні, пам'ять та ресурси, черги та команди, бар'єри та буфери пам'яті, презентація, шейдери та конвеєри, графічні конвеєри, креслення, обробка геометрії, обробка фрагментів, синхронізація, запити та багатопрохідність. Також тут можливо знайти пояснення того, що робить кожна функція специфікації Vulkan, наприклад обмеження пристрою (таких як максимальний розмір буфера кадру, кількість байтів у константі push, тощо) та способи їх запиту. Також описано концепції синхронізації в одному з розділів. Це абсолютно важливо для правильної розробки використовуючи багатопоточність. Також надано увагу темі продуктивності та найбільш поширених помилок які можуть на неї негативно впливати.
З ціллю дослідити найпоширеніші підходи до розробки графічних додатків старим підходом був обраний ресурс [2]. Він включає у себе поняття, необхідні для ефективної розробки, такі як:
1. Мова шейдерів OpenGL (GLSL)
2. Обробка вершин, команди креслення, примітиви, фрагменти та буфери кадрів
3. Керування, завантаження та арбітраж доступу до даних
4. Створення більших додатків та їх розгортання на різних платформах
5. Розширений рендеринг: імітація світла, художні та нефотореалістичні ефекти
6. Запобігання та налагодження помилок
7. стиснення текстур
Ресурси [9, 10] надають основи практичного використання OpenGL та Vulkan. Вони розкривають теми, такі як налагодження середи розробки, цикл рендерінгу, робота з шейдерами, буфери вершин, додавання текстур, освітлення, тощо. Також ці ресурси мають програмний код до кожної з тем, за яким можна здобути практичні навички розробки графічних додатків.
1.6.1 Огляд графічного конвеєру
Графічний конвеєр візуалізації ініціюється під час виконання операції візуалізації. Операції візуалізації вимагають наявності правильно визначеного об'єкта масиву вершин та пов'язаного об'єкта програми або об'єкта конвеєра програми, який забезпечує шейдери для програмованих етапів конвеєра [8].
Шейдери являють собою визначену користувачем програму, призначену для запуску на якомусь етапі графічного процесора.
Після запуску трубопровід працює в наступному порядку:
1. Обробка вершин:
- На кожну вершину, отриману з масивів вершин, діє вершинний шейдер. Кожна вершина потоку обробляється по черзі у вихідну вершину.

- Необов’язкові примітивні етапи тесселяції.
- Необов’язкова примітивна обробка геометричного шейдеру. Вихід - це послідовність примітивів.
2. Поточна обробка вершин, виходи останнього етапу коригуються або передаються в різні місця.
- Зворотній зв'язок про перетворення відбувається тут. Являє собою процес захоплення примітивів, згенерованих кроком (-ами) обробки вершин, запис даних з цих примітивів в об’єкти буфера.
- Примітивне відсікання, розділення перспективи та область перегляду перетворюються на віконний простір.
3. Розділення примітивів. Являє собою процес захоплення примітивів, згенерованих кроками обробки вершин, запис даних з цих примітивів в об’єкти буфера.
4. Перетворення сканування та примітивна інтерполяція параметрів, яка генерує ряд фрагментів.
5. Шейдер фрагментів обробляє кожен фрагмент. Кожен фрагмент генерує ряд результатів.
6. Пре-обробка за зразком, включаючи, але не обмежуючись цим:
- Тест на ножиці. Являє собою операцію, яка відкидає фрагменти, що потрапляють за межі певної прямокутної частини екрану.
- Трафаретний тест. Являє собою операцію, виконану після фрагментного шейдеру. Значення трафарету фрагмента перевіряється щодо значення в поточному буфері трафаретів; якщо тест не вдається, фрагмент вибраковується.
- Випробування на глибину. Вихідне значення глибини фрагмента може бути перевірено на глибину зразка, на який записується. Якщо тест не вдається, фрагмент відкидається
- Змішування. Приймає кольори фрагментів, що виводяться з шейдера фрагментів, і поєднує їх із кольорами в кольорових буферах, до яких ці вихідні дані відображають
Схему графічного конвеєру проілюстровано на рисунку 1.1

Рисунок 1.1 - Графічний конвеєр
Висновки до розділу 1
В першому розділі були розглянуті питання мети розробки і її доцільності, можливі способи використання і підходи до розроблюваної системи. Питання доцільності розробки розглядалося при огляді існуючих графічних інтерфейсів, та історії програмування комп’ютерної графіки. Були визначені недоліки і переваги старих та нових програмних інтерфейсів розробки комп’ютерної графіки. Було вирішено, що аналіз порівняння старих підходів, та нових багатоядерних підходів буде проводитися з використанням найбільш відкритих та поширених графічних інтерфейсів OpenGL та Vulkan.
Також було оглянуто відмінності між OpenGL та Vulkan, та оглянуто літературу, за допомогою якої стали зрозуміли такі ключові моменти, як робота графічного конвеєру, шейдери, архітектура рендеру.

2 Обґрунтування експериментального методу дослідження ефективності відображення анімаціЙ на багатоядерних системах
Завдання розробки нових ефективних методів обробки комп’ютерної графіки не втратило своєї актуальності. На теперішній час ЕОМ постійно зазнає значне та прогресуюче зростання потужності. Зокрема збільшується кількість ядер, які можуть обчислювати данні паралельно. Відповідно до їх потужності збільшуються об’єми оброблюваної інформації, зокрема чіткість та кількість моделей комп’ютерної графіки. Чіткість досягається за допомогою збільшення кількості полігонів, тобто графічних примітивів таких як трикутник, з яких складаються тривимірні об’єкти.
Зростання кількості інформації яка може бути оброблена комп’ютером паралельно робить можливим збільшення продуктивності при обчислюванні таких моделей. Зокрема це стосується графічних моделей, які є анімованими. Анімація досягається за рахунок зміни положення у простору частини чи усього об’єкта.
Такі зміни потребують обчислення матриць, які можуть бути застосовані для зміни кута повороту, положення у просторі, чи зміщення окремих вершин одного об’єкта. Анімація скадається з постійну зміну кадрів. Ці обчислення мають бути вирахувані кожен кадр, та можуть бути розпаралелені по багатьом ядрам за для зменшення часу потрібного на відображення одного кадру. Це призводить до того, що одночасно може бути відображено більша кількість анімованих об'єктів не втрачаючи змоги розрізняти плавність у русі. Мінімальна кількість кадрів за секунду які мають бути відображені для досягнення плавності є 24 кадри, які наразі поширені серед майже усіх моделей телевізорів.
2.1 Поняття та види анімаційних моделей
2 3D-модель - це об'ємна фігура в просторі, створювана в спеціальній програмі. За основу, як правило, приймаються креслення, фотографії, малюнки та детальні описи, спираючись на які, фахівці і створюють віртуальну модель [11].
Створення 3Д-моделі об'єкта здійснюється за допомогою 3Д-моделювання. На першому етапі 3D-моделювання проводиться збір інформації: ескізи, креслення, фотографії і відеоролики, малюнки, часто навіть використовують готовий зразок вироба - **в загальному, все, що допоможе зрозуміти зовнішній вигляд і структуру об'єкту. На підставі отриманої інформації 3Д-моделлер або 3Д-дизайнер створює тривимірну модель в спеціальній комп'ютерній програмі. Після того як модель буде виконана, на неї можна буде подивитися з будь-якого ракурсу, наблизити, віддалити, внести необхідні корективи. Сама по собі модель вже готова для подальшого використання - друку на 3Д принтері, 3Д-фрезерування на верстатах з ЧПУ або будь-якого іншого методу прототипування** [11].
Тривимірна модель складається з безлічі точок, які з'єднуються між собою гранями і утворюють полігони.
Вершина - це точка, яка має свої координати в тривимірній системі, тобто X, Y, Z. Свою назву вона отримала через те, що є крайньою точкою плоского багатокутника, або полігону.
Грань, або ребро - відрізок, який з'єднує дві вершини, поняття, узяте знову ж з геометрії. У тривимірній графіці гранню називають обмежувач полігонів.
Основною складовою в тривимірній графіці вважається полігон - плоский багатокутник, безліч яких і утворює тривимірну фігуру. Абсолютно будь-яка фігура буде будуватися з численних простих фігур (причому більшість редакторів використовує трикутники і чотирикутники). Чим більше буде простих фігур у складі складної, тим більш **гладкою буде здаватися** поверхня **3D-моделі (так зване високополігональні моделювання)** [11].
Сукупність полігонів несе інформацію про розмір і форму 3Д моделі, а обрана текстура дозволяє передати достовірну інформацію про зовнішній вигляд об'єкта і являє собою зображення на поверхні фігури.
Анімація **3d** - автоматичне переміщення або трансформація об'єктів у просторі та часі.

Простіше кажучи, раніше потрібно було покадрово малювати пересування кожного персонажа. Тепер досить створити тривимірну модель персонажа, після чого її можна рухати в просторі без додаткових зусиль і перемалювань. Але говорити щось просто, а на ділі - пожвавлення 3d моделі персонажів досить складний процес. Щоб змусити фігурку рухатися, мало мати доступ до комп'ютера і розумним програмами. Потрібно ще й уявляти, як може пересуватися герой, які сили на нього при цьому впливають (не ті, які вищі, а, наприклад, гравітація, сила тертя і опору) [12].
Анімація моделей може бути розділена на наступні види:
2.1.1 По ключовим кадрам: з точки А в точку Б
Створення ключових кадрів - один з найбільш поширених способів створення 3д анімації персонажів. Суть методу полягає ось у чому: на шкалі часу задається кілька головних точок, в яких становище або форма об'єкта змінюється. Аніматор задає потрібні параметри моделі в зазначених кадрах, а «проміжні» стану програма розраховує автоматично [12].
Приклад: Для простоти візьмемо гумовий м`ячик, який вдаряється об землю і відскакує вгору. Щоб відобразити один такий «стрибок», процес потрібно розбити на три етапи: м'ячик у верхній точці - м'ячик на землі - м'ячик знову у верхній точці. По-хорошому слід задати більше ключових кадрів, враховувати купу дрібниць. Як то, що при падінні гумовий корпус розтягується, а при ударі - сплющується [12].
2.1.2 Анімація по траєкторії
Не завжди 3d моделі персонажів - це люди або тварини. Нашим героєм може бути будь-який об'єкт, наприклад, літаюча камера або НЛО (в загальному все, на що вистачить фантазії). В такому випадку миготіння лампочок і обертання по осі буде недостатньо - не цікаво. А от змусити об'єкт літати по траєкторії, та ще й «відправити» камеру стежити за переміщенням, вчасно наближаючись і віддаляючись [12].
Анімація по траєкторії і вміле поводження з фокусом (ось вже що точно повинна вміти студія 3d анімації) перетворить просте кружляння об'єкта в просторі - в захоплюючий майже блокбастер.
Суть методу полягає в тому, щоб:
1. задати точку старту (початок шляху об'єкта);
2. позначити траєкторію (шлях, який пробробляє об'єкт);
3. вказати кінцеву точку (де модель повинна зупинитися).
Після того, як персонаж / об'єкт «прив'язується» до траєкторії, програма сама розраховує і створює рух. Якщо при цьому додати анімацію самого об'єкта (помахи крил, відкриття шлюзів, висування шасі) і «програти» з камерою, можна домогтися вельми цікавих ефектів.
2.1.3 Анімація в динамічному середовищі
Наш герой - не знаходиться у вакуумі. Будь-якого персонажа оточує якась реальність, в якій обов'язково є гравітація (якщо справа не в космосі), рух повітряних мас і інші види коливань. Все це варто враховувати, щоб анімація персонажа була досить реалістичною [12].
Строго кажучи, анімація в динамічному середовищі - швидше обчислювальна робота з глибоким зануренням в фізичні характеристики об'єктів. Але без усього цього навіть найдетальніше 3d моделювання з опрацюванням текстур не зробить персонажа жвавіше [12].
2.1.4 Motion capture: перетворення фільму в мультик
Технологія захоплення рухів - молода, але дуже популярна. Суть такого способу:
1. на актора закріплюються датчики;
2. поки актор рухається, камери фіксують положення датчиків;
3. їх зміщення обробляє програма і створює рухомий «скелет» з набором ключових кадрів;
4. отриманий пакет інформації «обтягається» оболонкою - для цього використовується 3d моделювання персонажів.
В результаті дії героя виходять реалістичними, переконливими, а аніматорам не доводиться боротися з фізикою і згадувати, де той гнеться [12].
2.2 Вибір моделей для дослідження
Вибрані моделі для дослідження були розділені на прості та складні.
У якості простих анімованих моделей була обрана модель 3д куба. Також було застосовано метод анімації по траєкторії. Моделі куба кружляють навколо своєї осі.
У якості складних анімованих моделей були обрані моделі тварин. Також ці моделі є анімовані, застосовуючи методи скелетної анімації по ключовим кадрам
2.3 Алгоритм обчислювання ефективності анімації
Для обчислення ефективності анімації була обрана величина середньої кількості кадрів які були опарацьовані та намальовані за певний період часу.
Формула для обчислення середньої кількості кадрів за секунду наведена далі:
AFPS=, (2.1)
де AFPS (Average frames per second) - це середня кількість кадрів за секунду,
N - кількість кадрів які були згенеровані за період тестування
S - кількість секунд за які було проведене тестування
2.4 Аналіз програмних платформ для досліду
Були проаналізовані наступні найпоширініші операціїні системи:
2.4.1 Операційна система Windows
Windows (з англ. - «Вікна») - **4 сімейство комерційних операційних систем (ОС) корпорації Microsoft, орієнтованих на управління за допомогою графічного інтерфейсу. Спочатку Windows була всього лише графічною програмою-надбудовою для поширеної в 1980-х і 1990-х роках операційної системи MS-DOS.**
На червень 2019 року, Windows була встановлена менш ніж на 88,5% персональних комп'ютерів і робочих станцій. За даними компанії Net Applications, на червень 2019 року ринкова частка Windows склала 88,33%. За іншими даними, ринкова частка Windows менше. Падіння частки пов'язано, в першу чергу, з тенденцією до скорочення продажів ПК в світі, а також зі збільшенням частки ОС конкурентів - macOS і Linux [13].
2.4.2 Операційна система **6 Linux**
Linux - сімейство Unix-подібних операційних систем на базі ядра Linux, включають той чи інший набір утиліт і програм проекту GNU, і, можливо, інші компоненти. Як і ядро Linux, системи на його основі як правило створюються і поширюються відповідно до моделі розробки вільного та відкритого програмного забезпечення. Linux-системи поширюються в основному безкоштовно у вигляді різних дистрибутивів - у формі, готової для установки і зручною для супроводу і оновлень, - і мають свій набір системних і прикладних компонентів, як вільних, так, можливо, і власницьких [14].

Станом на середину 2010-х років 10 системи Linux лідирують на ринках серверів (60%), є такими, що превалюють в дата-центрах підприємств і організацій (згідно Linux Foundation), займають половину ринку вбудованих систем, мають значну частку ринку нетбуків (32 % на 2009 рік). На ринку персональних комп'ютерів Linux стабільно займає 3- е місце (за різними даними, від 1 до 5%). Згідно з дослідженням Goldman Sachs, в цілому, ринкова частка Linux серед електронних пристроїв становить близько 42% [14].

2.4.3 8 Операційна система MacOS

Сімейство операційних систем macOS є другим за поширеністю для робочого столу (після Windows). Ринкова частка macOS (враховуються всі версії) за станом на квітень 2020 становить близько 18,99% за оцінками StatCounter. Найпопулярнішою версією macOS є Catalina (48,98% серед усіх версій macOS), слідом йдуть Mojave (21,43%), High Sierra (13,82%), Sierra (5,88%), El Capitan (4,7 %), Yosemite (3,18%). У macOS використовується ядро XNU, засноване на мікроядрі Mach і містить програмний код, розроблений компанією Apple, а також код з ОС NeXTSTEP і FreeBSD. До версії 10.3 ОС працювала тільки на комп'ютерах з процесорами PowerPC. Випуски 10.4 і 10.5 підтримували як PowerPC-, так і Intel-процесори. Починаючи з 10.6, macOS працює тільки з процесорами Intel [15].

2.5 Вибір програмної платформи для досліду

У якості операційної системи на якій проводиться дослід була обрана сучасна та найпоширініша система Microsoft Windows версії 10. 16 Інші операційні системи такі як Linux та MacOS не були розглянуті оскільки вони використовуються здебільшого або як серверні системи, або поширені лише серед багатіїв.

2.6 Аналіз апаратних платформ для досліду

Ключовими апаратними складовими для програм відтворення комп'ютерної графіки є процесор та відеокарта.

Центральний процесор - електронний блок або інтегральна схема, яка виконує машинні інструкції (код програм), головна частина апаратного забезпечення комп'ютера або програмованого логічного контролера. Іноді називають мікропроцесором або просто процесором [16].

Відеокарта (також звана графічною картою, графічною картою, графічним адаптером або адаптером дисплея) - це карта розширення, яка генерує подачу вихідних зображень на пристрій відображення (наприклад, монітор комп'ютера). Часто вони рекламуються як дискретні або виділені відеокарти, підкреслюючи різницю між ними та інтегрованою графікою. В основі обох - блок обробки графіки (GPU), який є основною частиною фактичних обчислень [17].

2.6.1 Процесори з архітектурою ARM

ARM - це сімейство архітектур скорочених обчислювальних наборів (RISC) для комп'ютерних процесорів, налаштованих для різних середовищ. Arm Holdings розробляє архітектуру та ліцензує її для інших компаній, які розробляють власні продукти, що реалізують одну з цих архітектур - включаючи системи на мікросхемах (SoC) та системи на модулях (SoM), що включають пам’ять, інтерфейси, радіостанції, тощо. Він також розробляє ядра, що реалізують цей набір інструкцій, та ліцензує ці конструкції для ряду компаній, які включають ці основні конструкції у свої власні продукти [18].

Процесори, що мають архітектуру RISC, як правило, вимагають менше транзисторів, ніж ті, що мають складну архітектуру обчислювальних наборів команд (CISC) (наприклад, процесори x86, що зустрічаються в більшості персональних комп’ютерів), що покращує вартість, енергоспоживання та тепловіддачу. Ці характеристики бажані для легких, портативних пристроїв, що працюють від акумуляторів - включаючи смартфони, ноутбуки та планшетні комп’ютери та інші вбудовані системи, але певною мірою також корисні для серверів та настільних комп’ютерів. Для суперкомп’ютерів, які споживають велику кількість електроенергії, ARM також є енергоефективним рішенням [18].

2.6.2 Процесори з архітектурою x86

x86 - це сімейство архітектур набору команд, спочатку розроблене Intel на базі мікропроцесора Intel 8086 та його варіанту 8088. 8086 був представлений в 1978 році як повністю 16-бітове розширення 8-бітового мікропроцесора 8080 від Intel, з сегментацією пам'яті як рішенням для адресування більшої кількості пам'яті, ніж може бути покрито звичайною 16-бітною адресою. Термін "x86" виник, оскільки імена кількох наступників процесора Intel 8086 закінчуються на "86", включаючи процесори 80186, 80286, 80386 та 80486 [19].

Станом на 2018 рік більшість проданих персональних комп’ютерів та ноутбуків базуються на архітектурі x86, тоді як у мобільних категоріях, таких як смартфони або планшети, переважає ARM; на високому рівні x86 продовжує домінувати в обчислювальних робочих станціях та сегментах хмарних обчислень [19].

2.6.3 Процесори з архітектурою x86-64

x86-64 (також відомий як x64, x86_64, AMD64 та Intel 64) - це 64-розрядна версія набору інструкцій x86, вперше випущена в 1999 році. Вона представила два нових режими роботи, 64-розрядний режим та режим сумісності, а також новий 4-рівневий режим пейджингового пошуку [20].

Завдяки 64-розрядному режиму та новому режиму підкачки, він підтримує значно більший обсяг віртуальної та фізичної пам’яті, ніж це було можливо у його 32-розрядних попередників, що дозволяє програмам зберігати більший обсяг даних у пам'яті. x86-64 також розширив регістри загального призначення до 64-розрядних, а також збільшив їх кількість з 8 (деякі з яких мали обмежену або фіксовану функціональність, наприклад для управління стеком) до 16 (повністю загальних), а також пропонує безліч інших удосконалень. . Операції з плаваючою комою підтримуються за допомогою обов’язкових інструкцій, подібних до SSE2, а регістри стилів x87 / MMX, як правило, не використовуються (але все ще доступні навіть у 64-розрядному режимі); натомість використовується набір з 32 векторних регістрів, по 128 біт кожен. (Кожен регістр може зберігати один або два числа з подвійною точністю або від одного до чотирьох одиничних номерів точності або різні цілі цілі формати.) У 64-розрядному режимі інструкції модифіковані для підтримки 64-розрядних операндів та 64-розрядного режиму адресації [20].

2.6.4 Відеокарти AMD Radeon

Radeon - торгова марка графічних процесорів, оперативної пам'яті і SSD, вироблених підрозділом Radeon Technologies (колишня ATI Technologies) компанії Advanced Micro Devices (AMD). Торгова марка була створена в 2000 році компанією ATI Technologies (у 2006 році поглиненої компанією AMD). Графічні рішення цієї серії прийшли на зміну серії Rage [21].

2.6.5 Відеокарти Nvidia

Nvidia - американська технологічна компанія, розробник графічних процесорів і систем на чіпі (SoC). Розробки компанії набули поширення в індустрії відеоігор, сфері професійної візуалізації, області високопродуктивних обчислень і автомобільної промисловості, де бортові комп'ютери Nvidia

використовуються в якості основи для безпілотних автомобілів [22].
Компанія була заснована в 1993 році. На IV квартал 2018 року було найбільшим в світі виробником PC-сумісної дискретної графіки з часткою 81,2% (статистика включає всі графічні процесори, доступні для прямої покупки кінцевими користувачами - GeForce, Quadro і прискорювачі обчислень на базі GPU Tesla). [22].
2.7 Вибір апаратної платформи для досліду
Для дослідження була обрана найпоширеніша модель процесору на архітектурі x86-64, та найпоширеніша модель відеокарти від Nvidia.
Детальні характеристики системи для дослідів наведені далі у наступних пунктах.
2.7.1 Процесор
1. Модель: AMD Ryzen 7 2700X.
2. Тип роз'єму: Socket AM4.
3. Кількість ядер: 8.
4. Тактова частота: 3700 Мгц.
5. Об'єм кеш пам'яті 3 рівня: 16 МБ.
2.7.2 Оперативна пам'ять
1. Модель: Kingston 14 HyperX Predator Black (HX432C16PB3 K2/16).
2. Тип пам'яті: DDR4.
3. Максимальна робоча частота: 3200 МГц.
4. Максимальна пропускна здатність: 25600 Мб/с
5. Ємність одного модуля оперативної пам'яті: 8 гігабайт.
6. Загальна кількість встановлених модулів пам'яті: 4.
7. Загальна ємність оперативної пам'яті: 32 гігабайти.
2.7.3 Відеокарта
1. Модель: 15 GIGABYTE GeForce RTX 2070 SUPER GAMING OC 3X WHITE 8G (GV-N207 SGAMINGOC WHITE- 8GD).
2. Тип пам'яті: GDDR6.
3. Інтерфейс: PCI Express 3.0 x16.
4. Обсяг пам'яті: 8 гігабайт.
5. Частоти роботи GPU: 1815 МГц.
6. Частоти роботи пам'яті: 14000 МГц.
2.7.4 Материнська плата
1. Модель: Gigabyte B450 AORUS Pro
2. Тип сокету процесора: AM4.
3. Модель чіпсета: B450.
4. Тип пам'яті: DDR4.
5. Максимальна робоча частота пам'яті: 3200 МГц.
2.7.5 Блок живлення
1. Модель: ATX Seasonic Focus Plus Gold 850W (SSR-850FX).
2. Вихідна потужність: 850W.
3. Напруга: 110/230 В.
4. Частота: 50/60 Гц.

2.7.6 Пам'ять диску
1. Модель: Samsung 850 Pro.
2. Об'єм: 256 гігабайт.
3. Інтерфейс підключення: SATAIII.
4. Швидкість читання: до 550 МБ/с.
5. Швидкість запису: до 520 МБ/с.
2.8 Оцінка ефективності розподілу роботи по ядрам процесору
Операційна система Windows на якій буде проводитися тестування використовує попереджувальне планування задач.
Попереджувальне планування використовується, 11 коли процес переходить із запущеного стану в стан готовності або із стану очікування в стан готовності. Ресурси (переважно цикли процесора) виділяються процесу протягом обмеженого періоду часу, а потім забираються, і процес знову розміщується в черзі готових, якщо в цьому процесі залишається час спалаху процесора. Цей процес залишається в черзі, поки він не отримає наступний шанс на виконання [23].
Процеси програми для досліду, або точніше потоки процесів для обробки та відображення анімованих графічних моделей будуть використовувати ядра процесору на які їх відправить операційна система. Взагалі, для інтенсивних потоків, операційна система рідко змінює ядро на якому цей потік виконується, тому ми можемо досить впевнено замірити навантаження на ядра процесору при виконанні досліду. Найбільш доцільна програма для виміру розподілу це системна програма «Resource Monitor» яка поставляється з операційною системою.
«Resource Monitor» - це новий службовий компонент, що з'явився в Windows 7 і Windows Server 2008 R2, за допомогою якого можна переглядати відомості про використання апаратних ресурсів (процесора, оперативної пам'яті, фізичних дисків і мережі) і програмних ресурсів (дескрипторів файлів і модулів) в режимі реального часу. Монітор ресурсів Windows дозволяє фільтрувати результати для обраних процесів або служб, за якими можна вести моніторинг. Крім цього, завдяки монітору ресурсів можна запускати, зупиняти, припиняти і відновлювати процеси і служби, а також усувати помилки тоді, коли програма не відповідає [24].
У даній дослідницькій роботі програма «Resource Monitor» була обрана для замірювання навантаження роботи на кожне ядро у реальному часі. Скріншот програми «Resource Monitor» наведено далі.

Рисунок 2.1 - Скріншот програми «Resource Monitor»
У правій частині вікна на рисунку 2.1 видно навантаження для кожного ядра. Навантаження рахується від 0 до 100 процентів, де 0 процентів це мінімальна робоча частота, 100 процентів це максимальна робоча частота. Кожну секунду данні обновлюються та показується графік з робочою частотою ядра минулої секунди.
2.9 Порівняння збільшення ефективності OpenGL та Vulkan зі збільшенням кількості ядер процесору
Щоб замірити ефективність дослідження зі збільшенням ядер процесору потрібно запустити програму

та зробити досліди на системах з різною кількістю ядер. Є три способи змінити кількість ядер на системі:
1. Фізична зміна процесору процесорами з різною кількістю ядер.
2. Використання віртуальних машин та виділення такій машині різної кількості ядер.
3. Використання конфігураційного меню Windows.
Спосіб з фізичною заміною процесору є дуже дорогим, оскільки потрібно придбати декілька процесорів з різною кількістю ядер. Також, цей спосіб буде не дуже точним, оскільки у різних процесорів можуть бути різні об'єми кешу та різні частоти на яких він працює.
Спосіб з віртаульною машиною не може бути використаний у цьому дослідженні, оскільки віртуальні машини не підтримують новітні програмні інтерфейсі для роботи з комп'ютерною графікою, такі як Vulkan.
Отже було вирішено обрати спосіб з конфігуруванням кількості ядер яке буде використовувати операційна система через конфігураційне вікно операційної системи, наведене на рисунку 2.2 нижче.

Рисунок 2.2 - Конфігураційне вікно зміни кількості ядер процесорів
Змінюючи кількість процесорів у конфігураційному вікні та перезавантажуючи комп'ютер, операційна система бачить лише задану кількість ядер процесору. Саме так й буде зроблені виміри дослідження на різних кількостях ядер.
2.10 Проектування методів обробки анімації графічних об'єктів для їх порівняння та аналізу
2.10.1 Проектування однопоточної моделі обробки графіки OpenGL
Однопоточна модель обробки та відображення графіки яка є у OpenGL є дуже простою. У загальному виді вона складається з трьох операцій:
1. Опрацювати ввід користувача
2. Відновити дані для наступного кадру
3. Відобразити кадр на екрані.
Ці операції виконуються у циклі до завершення програми. Схема однопоточного циклу рендерінгу наведена на рисунку 2.3:

Рисунок 2.3 - Однопоточний цикл рендерінгу
2.10.2 Проектування багатопоточної моделі обробки графіки Vulkan
Багатопоточна модель обробки графіки Vulkan складається з основного потоку виконання, та допоміжних потоків. Основний потік дає завдання допоміжним потікам на обробку графічних об'єктів. Кожен допоміжний потік відновлює буфера матриць об'єкту та команди відрисовки об'єкта.
Головний потік у свою чергу чекає завершення роботи усіх допоміжних потіків, збирає їх, та виконує відрисовку на екрані на основі даних які були створені чи змінені у допоміжних потіках для кожного графічного об'єкту анімації.
Схема багатопоточного циклу рендерінгу наведена на рисунку 2.4:

Рисунок 2.4 - Багатопоточний цикл рендерінгу
Саме так досягається перевага у використанні багатьох ядер процесору яка є у нових програмних інтерфейсів комп'ютерної грфіки.
2.11 Вибір метода виміру часу у операційній системі Windows
З моменту введення набору інструкцій x86 P5 багато розробників ігор використовували лічильник позначок часу читання, інструкцію RDTSC, для виконання синхронізації з високою роздільною здатністю. Мультимедійні таймери Windows є достатньо точними для обробки звуку та відео, але з часом кадрів, що становить дюжину мілісекунд або менше, вони не мають достатньої роздільної здатності для надання дельта-часу інформації. Багато ігор досі використовують мультимедійний таймер під час запуску для встановлення частоти процесора, і вони використовують це значення частоти для масштабування результатів від RDTSC, щоб отримати точний час. Через обмеження RDTSC, API Windows надає більш правильний спосіб доступу до цієї функціональності за допомогою процедур QueryPerformanceCounter та QueryPerformanceFrequency [25].
Тож для заміру часу буде використано рекомендований Microsoft спосіб з викликом функції програмного інтерфейсу до операційної системи Windows QueryPerformanceCounter з роздільною точністю менше ніж одна мікросекунда. Цього більш ніж досить для заміру часу виконання одного кадру. Для точності було визначено конкретні характеристики комп'ютеру на якому проводяться досліди.
Висновки до розділу 2
В другому розділі були освітлені та обґрунтовані напрямки дослідження ефективності відображення анімацій на багатоядерних системах.
Були вибрані моделі для дослідження розділені на прості та складні.
У якості простих анімованих моделей була обрана модель 3д куба з анімаційним методом по траєкторії.
У якості складних анімованих моделей були обрані моделі тварин які застосовують методи скелетної анімації по ключовим кадрам.
Для обчислення ефективності анімації була обрана величина середньої кількості кадрів які були опрацьовані та намальовані за певний період часу. Також було розглянуто формулу 2.1 за якою вираховується ця величина.
Після аналізу найпопулярніших програмних та апаратних платформ було обрано операційну систему Windows, процесор на базі архітектури x86-64 та відеокарту від Nvidia.
Далі було проаналізовано та обрано системну програми «Resource Monitor» за якою буде визначатися розподіл роботи по ядрам процесору. Також визначено, що для зміни кількості ядер буде використовуватися вікно конфігурації Windows, яке дозволяє змінювати максимальну кількість ядер які використовує операційна система.
Останні методи дослідження які були висвітлені це архітектура однопоточної та багатопоточної моделі відображення комп'ютерної графіки. У якості методу для вимірювання часу було обрано системну функцію QueryPerformanceCounter з роздільною точністю менше ніж одна мікросекунда.

3 Проектування й розробка інструментального забезпечення для дослідження ефективності відображення анімаціЙ на багатоядерних системах
3.1 Зовнішнє проектування
3.1.1 Формалізація задачі
Формалізація задачі на рівні зовнішнього проектування представлена у вигляді діаграми варіантів використання .

Користувач представлені у вигляді актора, що взаємодіє з системою за допомогою варіантів використання. Варіанти використання надають опис можливостей, які система надає акторам.
На діаграмах можуть бути використані наступні типи відношень між варіантами використання та акторами:
1. відношення асоціації - відображає зв'язок між акторами та варіантом використання. Відображається лінією зі стрілкою між акторами і варіантом використання;
2. відношення включення - показує, що варіант використання включається в базову послідовність, позначається стрілкою з поміткою «include»
Користувач може виконувати наступні варіанти використання:
1. Задати налаштування рендерінгу
2. Запустити рендерінг
3. Зупинити рендерінг
4. Подивитися результати рендерінгу
Схема варіантів використання (Use-case diagram) [26] наведена на рис. 3.1

Рисунок 3.1 - Схема варіантів використання
3.1.2 Вхідні дані
Вхідними даними програми є:
1. Тип рендерінгу (OpenGL або Vulkan).
2. Складність об'єктів рендерінгу (прості або складні).
3. Кількість об'єктів для рендерінгу (від одного до 512)
3.1.3 Вихідні дані
Результатом роботи програми є наступні вихідні дані:
1. Середня кількість кадрів за секунду за час рендерінгу
2. Середня кількість секунд за один кадр за час рендерінгу
3. Кількість використовуваних ядер процесору

3.1.4 Проектування динаміки системи
Для кожного варіанту використання можна побудувати діаграму послідовності (Sequence Diagram). Даний тип діаграми дозволяє розглядати динаміку взаємодії об'єктів у часі.
Для проектування розвитку подій в часі була розроблена діаграма послідовності для користувача (див. рис. 3.2), що відображає динаміку взаємодії об'єктів під час виконання. Актор є ініціатором послідовності дій.
Для відображення станів, в яких може знаходитись об'єкт або система в цілому, та умови переходу із одного стану в інший, використовується діаграма послідовності (Sequence Diagram). Діаграма використовується для характеристики поведінки елементу системи в межах її життєвого циклу у вигляді станів цих елементів та послідовності їх змін.

Рисунок 3.2 - Діаграма послідовності

3.1.5 Проектування інтерфейсу користувача
Проектування програмного інтерфейсу користувача грає важливу роль в етапі розробки програми, оскільки невдало спроектований інтерфейс ускладнить роботу з нею і може викликати у користувача небажання працювати із програмою.
Оскільки програмний продукт націлений на користувачів, що можуть не мати досвіду у роботі з подібними програмами, при розробці інтерфейсу необхідно дотримуватись цілого ряду правил:
1. природність інтерфейсу. Не бажано змінювати звичні шляхи вирішення виробничої задачі та термінологію, що використовується у даній сфері діяльності;
2. простота інтерфейсу. Інтерфейс повинен забезпечувати легкість у його вивченні та у використанні. Всі команди, повідомлення мають бути небагатослівними. Треба розміщувати та представляти елементи на екрані з урахуванням їх смислового значення та логічного взаємозв'язку;
3. естетична привабливість. Проектування візуальних компонентів є найважливішою складовою;
4. заголовки 3 повинні знаходиться біля або зверху редагованих полів;
5. стандартні елемента, такі як кнопки «ОК» та «Скасувати» повинні розташовуватися згідно з замовчуваннями операційної системи;
6. діалогові вікна повинні відповідати потоку даних між користувачем і системою;
7. треба пам'ятати про надсилання підтверджень користувачеві. У випадку об'ємних команд користувач повинен отримувати інформацію про відправлення йому команди;
8. у системи повинна бути проста обробка помилок. Помилка повинна бути показана, а правильні дані повинні використовуватися для виконання наступного завдання;
9. у системи повинна бути операція "відміна". У найпростішому випадку система повертається до останньої операції. У складніших - до попередніх;
10. зв'язані операції повинні бути об'єднані в один діалог. Якщо це неможливо, операції повинні бути розділені таким чином, щоб зв'язані діалоги були доступні;
11. потрібно дотримуватися правила Міллера. Правило Міллера говорить, що людина може зосередитися на 5-9 елементах. Правило повинне застосовуватися при проектуванні меню, підменю, діалогових полів і т .д. Правило може бути реалізоване шляхом декомпозиції інтерфейсу і його подальшим угрупуванням в об'єднані групи.
1 Принцип обліку знань користувача припускає наступне: інтерфейс повинен бути настільки зручний при реалізації, щоб користувачам не знадобилося особливих зусиль, щоб звикнути до нього. У інтерфейсі повинні використовуватися терміни, зрозумілі користувачеві, а об'єкти, керовані системою, повинні бути безпосередньо пов'язані з робочим середовищем користувача.
Наприклад, якщо розробляється система, призначена для авіадиспетчерів, то керованими об'єктами в ній повинні бути літаки, траєкторії польотів, сигнальні знаки і тому подібне . Основну реалізацію інтерфейсу в термінах файлових структур і структур даних необхідно приховати від кінцевого користувача. Принцип узгодженості інтерфейсу користувача припускає, що команди і меню системи повинні бути одного формату, параметри повинні передаватися у всі команди однаково і пунктуація команд повинна бути схожою [27].
Час на навчання користувачів з таким інтерфейсом скорочується. Також знання, отримані при роботі з одним компонентом системи можна застосувати з іншими частинами системи.

В даному випадку мова йде про узгодженості низького рівня. І творці інтерфейсу завжди повинні прагнути до нього. Проте бажана узгодженість і більш високого рівня. Наприклад, зручно, коли для всіх типів об'єктів системи підтримуються однакові методи (такі, як друк, копіювання і тому подібне). Проте повна узгодженість неможлива і навіть небажана. Наприклад, операцію видалення об'єктів робочого столу доцільно реалізувати за допомогою їх перетягання в корзину. Але в текстовому редакторові такий спосіб видалення фрагментів тексту здається неприродним.
Завжди потрібно дотримувати наступний принцип: кількість несподіванок повинна бути мінімальною, оскільки користувачів дратує, коли система раптом починає поводитися непередбачуваною. При роботі з системою у користувачів формується певна модель її функціонування. Якщо його дія в одній ситуації викликає певну реакцію системи, природно чекати, що таке ж дія в іншій ситуації приведе до аналогічної реакції. Якщо ж відбувається зовсім не те, що очікувалося, користувач або дивується, або не знає, що робити. Тому розробники інтерфейсів повинні гарантувати, що схожі дії справлять схоже враження.
Дуже важливий принцип відновлюваності системи, оскільки користувачі завжди допускають помилки. Правильно спроектований інтерфейс може зменшити кількість помилок користувача (наприклад, використання меню дозволяє уникнути помилок, які виникають при введенні команд з клавіатури), проте всі помилки усунути неможливо. У інтерфейсах повинні бути засоби, що по можливості запобігають помилкам користувача, а також що дозволяють коректно відновити інформацію після помилок.
Підтвердження деструктивних дій - якщо користувач вибрав потенційно деструктивну операцію, то він повинен ще раз підтвердити свій намір.
Можливість відміни дій - відміна дії повертає систему в той стан, в якому вона знаходилася до їх виконання. Не зайвою буде підтримка багаторівневої відміни дій, оскільки користувачі не завжди відразу розуміють, що зробили помилку.
Наступний принцип - підтримка користувача. Засоби підтримки користувачів повинні бути вбудовані в інтерфейс і систему і забезпечувати різні рівні допомоги і довідкової інформації. Повинне бути декілька рівнів довідкової інформації - від основ для початкуючих до повного опису можливостей системи. Довідкова система повинна бути структурованою і не перенавантажувати користувача зайвою інформацією при простих запитах до неї. Принцип обліку різнорідності користувачів припускає, що з системою можуть працювати різні їх типи. Частина користувачів працює з системою нерегулярно, час від часу.
Але існує і інший тип "досвідчені користувачі", які працюють з додатком щодня по декілька годин. Випадкові користувачі потребують такого інтерфейсу, який "керував" би їх роботою з системою, тоді як досвідченим користувачам потрібний інтерфейс, який дозволив би їм максимально швидко взаємодіяти з системою.
Крім того, оскільки деякі користувачі можуть мати різні фізичні вади, інтерфейс повинен мати інструменти, які допоможуть їм переналаштувати інтерфейс для себе. Це можуть бути інструменти, які дозволяють відображати збільшений текст, замінювати звук текстом, створювати великі кнопки тощо.
3.1.6 Створення ескізів форм
На етапі проектування до остаточного затвердження інтерфейсу користувача зручно оперувати ескізами екранів. При розробці ескізів багато уваги уділялося внутрішньопрограмній узгодженості інтерфейсу. Назви управляючих кнопок, розташування ключових елементів форм і загальний стиль по можливості уніфікувалися або робилися схожими, з метою полегшення навчання користувачів при роботі з програмою.
Програма повинна мати простий інтерфейс і складатись з екранів та діалогів кількох типів:
1. екран задання налаштувань рендерінгу рис - 3.2
2. екран рендерінгу рис - 3.3
3. екран результатів рендерінгу - рис 3.4
Розроблений інтерфейс є інтуїтивно зрозумілим для користувача, що зменшує ймовірність помилок при роботі.
Екран задання налаштувань містить в собі 2 пари радіокнопок.
Радіокнопка (Radio button), або перемикач - елемент інтерфейсу, який дозволяє користувачеві вибрати одну опцію (пункт) з визначеного набору (групи). Радиокнопки представляють собою елемент круглої (рідше - квадратної або ромбовидної) форми, а вибраний елемент виділяється найчастіше точкою всередині. Поруч з кнопкою розташовується опис обраного елемента. Радиокнопки розташовують групами по кілька штук, причому в будь-який момент обрана може бути тільки одна кнопка з групи. При ініціалізації програми будь-яку кнопку з групи, як правило, вже вибрана, але технічно можливо залишити поза обраної жодну. В такий стан групу радіокнопок привести засобами тільки самих радіокнопок неможливо [28].
Перша пара радіокнопок використовується як засіб вибору програмного інтерфейсу комп’ютерної графіки який буде використовуватися під час рендерінгу. Це може бути OpenGL або Vulkan
Друга пара радіокнопок використовується як засіб вибору складності анімованих графічних об’єктів, які будуть відображатися під час рендерінгу. Доступні два типи об’єктів. Прості являють собою прості 3д куби. Складні об’єкти являють собою анімовані за допомогою скелетної анімації моделі тварин.
Також на екрані задання налаштувань присутнє поле редагування для задання кількості відображаємих анімованих графічних об’єктів при запуску рендерінгу.
Поле редагування (Textbox, Edit field) - елемент графічного інтерфейсу користувача, що дозволяє виробляти введення і виведення текстової інформації в певній галузі інтерфейсу. Якщо записана в поле редагування інформація перевищує його розміри, необхідно використовувати вертикальну або горизонтальну смугу прокрутки, рухаючи повзунок вертикально або горизонтально відповідно за допомогою миші. У деяких інтерфейсів для зручного відображення інформації можна змінювати розміри поля редагування. Введення даних в текстове поле відбувається за допомогою алфівітно-цифрового блоку клавіатури [28].
Мінімальна кількість відображаємих анімованих графічних об’єктів складає один. Максимальна - 512.
Після задання налаштувань користувач має змогу нажати на кнопку.
Кнопка (кнопка) - базовий елемент інтерфейсу комп'ютерних програм, принцип дії та вид аналогічної кнопки в техніці. При натисканні на кнопку відбувається програмно пов'язане з цим натисканням дія або подія. Опис пов'язаних з кнопкою дій пропонує запропонований розробником користувальницького вмісту разом із документацією до додатка. Для того, щоб ініціювати подію викликаєме кнопкою необхідно перемістити курсор на робочу область кнопки і створити однократне або двократне (в

залежності від функціональної специфікації) натискання на ліву кнопку миші. Проста кнопка має два стану - "натиснути" і "віджато". Кнопка може змінювати вигляд залежно від стану та положення курсору або фокусу [28].
На формі задання налаштувань є тільки одна кнопка яка запускає рендерінг.
При натисканні на кнопку старту рендерінгу, у робочому вікні програми з'являється задані анімовані об'єкти. У рядку стану на верхній частині робочого вікна відображається поточна кількість кадрів за секунду які були відображені за останній час. Це поле відновлюється кожну секунду і надає оперативну інформацію оператору персонального коп'ютера.
Рядок стану (Status bar) - елемент графічного інтерфейсу, призначений для виведення повідомлень. Рядок стану зазвичай, має прямокутну форму і знаходиться в нижній частині робочого вікна. Рядок стану не несе ніяких інших функцій, крім виведення оперативної інформації [28].
Екран результатів рендерінгу містить три текстових поля з результатами рендерінгу такими як середнє число кадрів відображаємих за секунду, середня кількість секунд потрібних на відображення одного кадру, та кількість застосованих ядер процесору.
Поле редагування (Textbox, Edit field) - елемент графічного інтерфейсу користувача, що дозволяє виробляти введення і виведення текстової інформації в певній галузі інтерфейсу. Якщо записана в поле редагування інформація перевищує його розміри, необхідно використовувати вертикальну або горизонтальну смугу прокрутки, рухаючи повзунок вертикально або горизонтально відповідно за допомогою миші. У деяких інтерфейсів для зручного відображення інформації можна змінювати розміри поля редагування. Введення даних в текстове поле відбувається за допомогою алфівітно-цифрового блоку клавіатури [28].
Також екран результатів містить у собі кнопку при натисканні на яку користувач повертається до першого вікна налаштування рендерінгу.

Рисунок 3.3 - Ескіз екрану налаштувань

Рисунок 3.4 - Ескіз екрану рендерінгу

Рисунок 3.5 - Ескіз екрану результатів
3.2 Внутрішнє проектування
3.2.1 Вибір мови програмування
Для розробки програми для дослідження була обрана мова програмування C++. Ця мова програмування є об'єктно орієнтованою, має зворотну сумісність з мовою програмування C.
Оскільки програма дослідження передбачає роботу з програмними інтерфейсами комп'ютерної графіки OpenGL та Vulkan, які в свою чергу офіційно та без додаткових налаштувань можуть бути використані через програмний інтерфейс операційної системи, мова C++ ідеально підходить для того, що би використовувати програмні інтерфейси операційної системи без додатків, з можливістю застосування об'єктно орієнтованого підходу. Це пояснюється тим, що операційні системи самі працюють на мові програмування C та C++, та експортують свої інтерфейси на таких мовах.
3.2.2 Вибір **13** **системи контролю** версіями
Система керування версіями (СКВ, **англ. source code management, SCM) - програмний інструмент для керування версіями одиниці інформації:** початкового **коду програми, скрипту, веб-сторінки,** вебсайту, **3D-моделі, текстового документу тощо.** [29]
9 Система контролю дозволяє зберігати попередні версії файлів та завантажувати їх за потребою. Вона зберігає повну інформацію про версію кожного з файлів, а також повну структуру проекту на всіх стадіях розробки. Місце зберігання даних файлів називають репозиторієм. [29]
Системи контролю версії є необхідним інструментом **для роботи** розробника програмного забеспечення. Вони дають такі переваги:
1. Створення різних варіантів одного файлу;
2. Документування змін до файлів, таких як автора зміни, рядок зміни, тощо;
3. Є контроль доступу до читання чи запису у файли підконтрольні системі контролю версії.
4. Може документувати проект чи кожну версію проекту;
5. Дозволяє пояснювати зміни та переглядати такі пояснення;
Системи контролю версії діляться на централізовані та розподілені.
Централізовані системи версії використовують сервер який слідкує за усіми змінами. Кожен розробник має підключитися до такого серверу що би мати змогу читати чи вносити зміни до файлів проекту. Такі системи мають свої недоліки та переваги. Недоліками є те, що в разі аварійного припинення роботи серверу уся історія змін стає недоступною, та зупиняється усяка можливість роботи з проектом. Перевагою є те, що розробники зберігають значно меньше кількості локальної інформації, адже уся вона їде з сервера.
Розподілені системи у свою чергу не мають жорстко прив'язаного серверу, але не виключають його використання. Уся інформація про усі зміни у проекті зберігається у кожного розробника, який працює з проектом. Кожен розробник може змінювати та читати файли проекту навіть без доступу до серверу чи мережі Інтернет. Але це було би не зручно без серверу. Сервер потрібен для того, що би декілька розробників мали змогу завантажити на сервер, або з серверу свої чи інші зміни.
У якості системи контролю версії для цієї роботи була обрана розподілена система контролю версії Git.
7 Git - розподілена система керування версіями файлів та спільної роботи. Проект створив Лінус Торвальдс для керування **розробкою ядра Linux, а сьогодні підтримується Джуніо Хамано** (англ. Junio C. Hamano). **Git є однією з найефективніших, надійних і високопродуктивних систем керування версіями, що надає гнучкі засоби нелінійної розробки, що базуються на відгалуженні і злитті гілок. Для забезпечення цілісності історії та стійкості до змін заднім числом використовуються криптографічні методи, також можлива прив'язка цифрових підписів розробників до тегів і комітів** [30].
Ця система контролю версії є досить зручною та сучасною. Також корпорація Microsoft надає безкоштовні сервера для зберігання проектів у сервісі Github.
3.2.3 Вибір системи збірки програмного забеспечення
Система збірки є дуже важливою частиною будь якого складного проекту.
Прикладом таких систем є програма make.
5 make це **утиліта для автоматичної побудови програм. Дії, що повинна виконати make, описують в спеціальних файлах, які називаються Makefile. Make вказує послідовність дій, які повинні бути виконані заради побудови програми. В основному ця послідовність складається з викликів команд компілятора,**

компонувальника та файлової системи. Основною особливістю утиліти make є те, що вона не просто виконує послідовність кроків, що може привести до великого часу побудови програми. Make виконує дії лише над тими файлами, які змінилися з часу попереднього виклику програми. Отже час на перебудову програми стає значно менший [31].
Також розповсюдженим прикладом у операційній системі Windows є система msbuild. Ця система використовує файли з описом збірки проекту на мові XML.
У якості системи збірки для цього проекту була обрана система CMake.
Взагалі кажучі, ця система сама по собі не є системою збірки. Ліпше сказати що це є генератор інших систем збірок.
CMake описує збірку проекту у файлі CMakeLists.txt використовуючи власну мову програмування схожу на паскаль.
Перед початком роботи треба запустити програму cmake яка використовуючи описання збірки проекту у файлі CMakeLists.txt згенерує обрані файли до іншої системи збірки. Таким чином, використовуючи одні і тіж сами файли опису збірки можливо використовувати їх з різними системами збірки. Наприклад, на операційній системі Linux найчастіше генерують файли до системи збірки make. А на операційній системі Windows йде генерація файлів до системи збірки msbuild. Також є можливість запустити Visual Studio з згенерованого проекту. Це є досить зручним тому й застосовано у цьому дипломному проекті.
3.2.4 Ієрархія та взаємодія класів системи
Виконаємо проектування системи. Для початку визначимо основні обов'язки системи.
1. Задання налаштувань до рендерінгу.
2. Рендерінг використовуючи OpenGL.
3. Рендерінг використовуючи Vulkan.
4. Рендерінг складних об'єктів.
5. Рендерінг простих об'єктів.
6. Відображення результатів рендерінгу
Зробимо інтерфейс для рендерінгу, та реалізуємо його класами рендерінгу з OpenGL та Vulkan. Також зробимо клас сцени, яка буде ініціалізувати класи рендерінгу з об'єктами заданою кількістю та типом. Для вікна результатів та налаштування теж розробимо відповідні класи.
Схема взаємодії класів наведена на рисунку 3.6

Рисунок 3.6 - Схема взаємодії класів
На рисунку 3.6 можливо побачити що програма була спроектована з використанням об'єктно орієнтованого підходу.
Було використано інтерфейс IRender. Його реалізують клас для відображення графічного інтерфейсу користувача результатів та налаштувань RenderGui. Також його реалізують класи для рендерінгу анімованих графічних моделей RenderOpengl та RenderGUI.
У свою чергу клас Scene використовує інтерфейс IRender для налаштувань рендеру та запуску не знаючи про те який саме тип рендеру це є.
Класи RenderVulkan та RenderOpengl використовують внутрішні допоміжні класи для роботи з 3д моделями, шейдерами та камерою.
Задача класу Main створити об'єкт RenderGUI, на основі вибраних налаштувань створити одну з реалізації інтерфейсу IRender, та віддати цей об'єкт у створений об'єкт класу Scene, який в свою чергу зробить підготовку 3д моделей, їх типу та кількості, та запустить рендер.

3.2.5 Використані принципи проектування
Під час проектування були використані принципи KISS та SOLID.
Принцип SOLID розшифровується таким чином:
1. Single responsibility - принцип єдиної відповідальності
2. Open-closed - принцип відкритості / закритості
3. Liskov substitution - принцип підстановки Барбари Лисков
4. Interface segregation - принцип розділення інтерфейсу
5. Dependency inversion - принцип інверсії залежних
Принцип єдиної відповідальності обозначає, що кожен об'єкт повинен мати одне зобов'язання та рівень обов'язковості бути повністю інкапсульованим у класі. Усі його послуги повинні бути зроблені виключно на забезпеченні цих зобов'язань [32].
Принцип відкритості / закритості декларує, що програмні сутності (класи, модулі, функції та інше) повинні бути відкритими для розширення, але закритими для змін. Це означає, що ці сутності можуть змінити своє введення без змін їх вихідного коду [32].
Принцип підстановки Барбари Лисков у формуліровці Роберта Мартина: «функції, які використовують базовий тип, повинні мати можливість використовувати підтипи базового типу не знайомих за цим» [32].
Принцип розділення інтерфейсу у формуліровці Роберта Мартина: «клієнти не повинні залежати від методів, які вони не використовують». Принцип розділення інтерфейсів говорить про те, що занадто «товсті» інтерфейси необхідно розподіляти на більш маленькі та специфічні, щоб клієнти маленьких інтерфейсів знали лише про методи, які необхідні їм у роботі. У тому, що при зміні методу інтерфейсу не слід змінювати клієнтів, які цей метод не використовують [32].
Принцип інверсії залежних - модулі верхніх рівнів не повинні залежати від модулів нижчих рівнів, а обидва типи модулів повинні залежати від абстракцій; самі абстракції не повинні залежати від деталей, а ось деталі повинні залежати від абстракцій [32].
KISS - це принцип проектування та програмування, при якому просторова система декларується у якості основної сукупності або цінності. Існує два варіанти розшифровки аббревіатури: «будь простим, дурним» і більш коректним є «тримай коротко і просто» [33].
У проектуванні наступних принципів KISS виражається в тому, що:
Не має смислу реалізувати додаткові функції, які не будуть використовуватися зовсім або їх використання крайні маловірогідні, як правило, для більшості користувачів достатньо базової функціональності, а також умовне використання лише для зручності додатків [33];
Не варто перенавантажувати інтерфейс опціями, які не будуть потрібні більшості користувачів, значно простіше передбачити для них окремий «розширенний» інтерфейс (або зовсім відмовитися від відповідної функції) [33];
Безглуздо робити реалізацію складної бізнес-логіки, яка виконує абсолютно всі можливі варіанти впровадження систем користувача та навколишнього середовища, - по-перших, це просто неможливо, а

по-других, така фанатичність заставляє збирати «зореліт», що найчастіше ірраціонально з комерційною зору [33].
3.2.6 Використані шаблони проектування
У дипломній роботі були використані такі шаблони проектування такі як інтерфейс та стратегія.
Інтерфейс це основний шаблон проектування, що представляє собою загальний метод для структурування комп'ютерних програм для того, щоб їх було більше зрозуміти. У загальному інтерфейс - це клас, який забезпечує програму простою або більш програмно-специфічну можливість доступу до інших класів [34].
У даному випадку було використано інтерфейс IRender з яким працює клас Scene. Клас Scene нічого не знає про те, яку саме реалізацію інтерфейсу IRender на даний момент він використовує.
Стратегія це **12 поведінковий шаблон проектування, призначений для визначення сімейства алгоритмів, інкапсуляції кожного з них** та **забезпечення їх** взаємозамінності. **Це дозволяє вибирати алгоритм шляхом визначення відповідного класу. Шаблон Strategy дозволяє** змінювати виборний **алгоритм** незалежності **від** об'єктів- **клієнтів, які його використовують** [35].
У цій роботі можливо сказати, що стратегія це клас Scene. Передаючи різні об’єкти класів-реалізацій від класу IRender ми тим само змінюємо алгоритм яким робиться рендер.
3.2.7 Технологічна платформа
Під час реалізації системи використовувалася така технологічна платформа.
GLAD - бібліотека для динамічної загрузки графічних програмних інтерфейсів. Вона потрібна для того що би загрузити вказівники функції OpenGL та Vulkan в час виконання програми. Це потрібно що би здобути доступ до цих програмних інтерфейсів до комп’ютерної графіки. Вони роблять це за допомогою програмного інтерфейсу до операційної системи, яка в свою чергу дає змогу загрузити функції з файлу динамічної бібліотеки (dll). Ці функції реалізуються розробником графічних карт.
GLFW - бібліотека для керування вікном операційної системи. Також вона може створювати начальне налаштування графічних інтерфейсів, потрібне для того що би вказати їм на яке вікно та у якому форматі має відрисовуватися комп’ютерна графіка.
Також служить для того, що би керувати та обробляти ввід з клавіатури та миші.
GLM - Бібліотека для математичних операції. Надає класи та методи для операцій над векторами та матрицями. Спеціально розроблена для того, що би бути сумісною з форматами графічних інтерфейсів комп'ютерної графіки.
ASSIMP - Бібліотека для загрузки файлів з анімованими об’єктами. Ключовою функцією є можливість завантаження різних форматів 3д моделей, які перетворюються у загальний формат визначений цією бібліотекою.
IMGUI - Бібліотека для графічного інтерфейсу. Надає змогу відрисовувати елементи керування, такі як кнопка, перемикач, та інші.
3.3 Стратегія тестування
Тестування програмного забезпечення охоплює цілий ряд видів діяльності, дуже аналогічний послідовності процесів розробки програмного забезпечення. Сюди входять постановка задачі для тесту, проектування, написання тестів, тестування тестів і, нарешті, виконання тестів і вивчення результатів тестування. Вирішальну роль грає проектування тесту. Можливий цілий спектр підходів до вироблення стратегії проектування тестів [36].
Стратегія тестування повинна відповідати на такі питання:
1. що тестувати;
2. якими методами.
Усі методи тестування можна поділити на дві групи: тестування за вхідними даними - «чорного ящика» [37, 38], та тестування логіки програми «білого ящика» [39]).
Тестування чорної скриньки може бути використано для методів, які мають лінійну структуру. Для того, щоб визначити, який метод тестування на чорну скриньку є більш ефективним у цій ситуації, спочатку слід розглянути кожен метод окремо.
Деякі методи з нелінійною структуру, особливо ті, що обробляють дані зі складною структурою, слід перевірити двома методами: еквівалентним методом розділу ("чорний ящик"), оскільки він дозволяє класифікувати можливі помилки, а іноді дозволяє виявляти такі помилки, які не були виявлені під час тестування методами «білого ящика», а також вхідні та вихідні дані не обмежуються специфікацією програми, а також залежать одна від одної та способу охоплення рішень.
Тестування методів класів, які мають нелінійну структуру, тобто мають умови, цикли, необхідно провести з використанням тестування методом покриття рішень («білого ящика»), у разі коли умови прості, та методом покриття умов, якщо умови складні. Для тестування білим ящиком виберемо метод покриття рішень та покриття умов. Для тестування чорним ящиком - метод припущення про помилку та метод еквівалентного розбиття.

3.3.2 Вибір стратегії налагодження програми
Розроблена система досить об’ємна і складається із взаємопов’язаних частин. Налагодження буде виконуватися за допомогою методів індукції та просування від місця виникнення помилки до місця помилки.
3.3.3 Результати налагодження програми
При налагодженні програми було виявлено деякі помилки. Наприклад, помилка при загрузці файлу текстури.
До місця виникнення ймовірної помилки ставиться точка зупину («breakpoint») зображена на рис. 3.7, на якій програма зупиниться і чекатиме покрокового виконання. На кожному наступному кроці існує можливість перегляду змісту змінних та результатів виконання функцій за допомогою інструменту «Evaluate Expression» (див. рис. 3.8). Таким чином легко відстежити, в якому місці програми виникає помилка.

Рисунок 3.7 - Використання точки зупину програми

Рисунок 3.8 - Перегляд значення змінної
Інструмент «Evaluate Expression» також дозволяє викликати методи та проглядати результат, який вони повертають, що є дуже корисним для тестування. За допомогою перерахованих утиліт були налагоджені майже всі функції програми. Знайдені помилки було ліквідовано.
Висновки до розділу 3

У третьому розділі було розглянуте зовнішнє та внутрішнє проектування.
При зовнішньому проектуванні було формалізовано задачу, їй вхідні та вихідні дані та схему способів використання. Також було спроектовано динаміку системи за допомогою діаграми послідовності. Спроектовано ескізи інтерфейсу користувача.
При внутрішньому проектуванні було обрано мову програмування C++. Також було обгрунтовано та обрано систему контролю версії, систему збірки програмного забезпечення. Було розроблено ієрархію класів та їх взаємодію згідно з принципами парадигми об'єктно орієнтованого програмування. Описано використані бібліотеки.
Також було описано стратегію тестування та результати налагодження програми.
Програма є придатною до використання.

4. Дослідження ефективності відображення анімацій на багатоядерних системах
4.1 Збір даних для аналізу
Збір даних проводився з використанням комп'ютеру на базі операційної системи Windows. Детальні характеристики описані у пункті 2.6.
Починаючи з одного ядра, та змінюючи кількість ядер на одне ядро, було виміряно дані для різної кількості моделей.
У даній дипломній роботі для оцінки зміни ефективності використовувалися 10, 100 та 500 анімованих графічних моделей, які одночасно відрисовувалися на екрані під час рендеру. Тести проводилися двічі. Один раз на рендері за допомогою програмного інтерфейсу до комп'ютерної графіки OpenGL. Другий раз на рендері за допомогою програмного інтерфейсу до комп'ютерної графіки Vulkan.
Також іспити подвіювалися через те, що у якості анімованих моделей було використано два типи: прості та складні. Простий тип анімованих моделей це 3д куб який обертається навколо своєї вісі. Складний тип анімованих моделей це 3д моделі тварин зі скелетною анімацією.
Для кожного типу кожної кількості моделей було зібрано інформацію, а саме:
1. Середня кількість кадрів за секунду.
2. Кількість використаної пам'яті графічного процесору.
3. Кількість використаної оперативної пам'яті
Зібрані дані для десяти простих моделей наведені у таблиці 4.1. Для 100 моделей у таблиці 4.2. Для 500 моделей у таблиці 4.3.
Зібрані дані для десяти складних моделей наведені у таблиці 4.4. Для 100 моделей у таблиці 4.5. Для 500 моделей у таблиці 4.6.

Таблиця 4.1 - Зібрані дані для десяти простих моделей.
Кількість ядер FPS OpenGL GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб) FPS Vulkan GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб)
8 3524.58 79648 38844 5114.068 99272 59376
7 3442.121 79636 38840 4910.444 99268 58468
6 3501.287 79636 38832 4908.323 99268 57420
5 3506.833 79636 38944 4898.079 99224 57260
4 3512.181 79636 38872 4810.297 99268 56812
3 3341.337 79636 39280 4512.096 99268 56504
2 3033.547 79648 38392 3973.538 99268 56020
1 2135.311 79640 31484 3041.443 99272 49044

Таблиця 4.2 - Зібрані дані для ста простих моделей.
Кількість ядер FPS OpenGL GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб) FPS Vulkan GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб)
8 1025.618 79636 40148 2132.973 105448 76256
7 1038.978 79648 40196 2145.989 105404 76816
6 1012.066 79648 40204 2165.39 105400 75464
5 1044.569 79636 40156 2055.611 105448 76540
4 1059.232 79636 40176 1937.76 105404 72992
3 1011.612 79648 40048 1727.357 105448 72180
2 1013.952 79648 40756 1565.177 105440 68412
1 788.656 79632 32656 1471.663 105428 58104

Таблиця 4.3 - Зібрані дані для п'ятиста простих моделей.
Кількість ядер FPS OpenGL GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб) FPS Vulkan GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб)
8 774.727 81684 41448 1631.82 107476 84736
7 773.917 81684 41372 1622.579 107468 85160
6 773.562 81696 41332 1614.145 107468 81104
5 777.731 81684 40748 1509.396 107468 81128
4 788.579 81684 41448 1463.737 107472 78512
3 753.69 81696 41508 1353.444 107520 77472
2 758.99 81684 41336 1273.405 107520 76932
1 632.555 81680 33080 1130.3 107504 65628

Таблиця 4.4 - Зібрані дані для десяти складних моделей.
Кількість ядер FPS OpenGL GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб) FPS Vulkan GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб)
8 784.51 223360 199976 1966.821 241952 252816
7 769.067 223388 208104 1944.66 241948 252996
6 762.868 223376 203960 1968.191 241904 253304
5 732.205 223380 203904 1936.505 241896 252776
4 754.518 223368 205188 1758.827 241944 252184
3 750.211 223364 196740 1483.39 241936 247520
2 725.34 223364 195332 1137.627 241948 252452
1 551.729 223352 188768 615.572 241956 245324

Таблиця 4.5 - Зібрані дані для ста складних моделей.
Кількість ядер FPS OpenGL GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб) FPS Vulkan GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб)
8 79.59 511868 486492 386.435 530358 665228
7 86.224 511860 488072 354.583 530364 671836
6 86.977 511880 489164 313.158 530308 665024
5 82.812 511860 490944 272.014 530328 667076
4 86.203 511852 492136 233.191 530280 654472
3 83.153 511864 495544 190.856 530368 681192
2 88.262 511856 483776 133.587 530360 663656
1 76.501 511844 490944 76.183 530332 649552

Таблиця 4.6 - Зібрані дані для п'ятиста складних моделей.
Кількість ядер FPS OpenGL GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб) FPS Vulkan GPU пам'ять OpenGL (Кб) Оперативна пам'ять OpenGL (Кб)
8 60.445 642636 624000 272.45 660776 847344
7 63.065 642608 625128 251.901 660724 848680
6 63.431 642620 620492 224.799 660792 835352
5 62.724 642616 619788 199.902 660696 840936
4 60.713 642616 619764 169.251 660768 851912
3 63.156 642616 622500 137.17 660764 832220
2 62.207 642632 629444 101.311 660760 831908
1 56.73 642596 610924 55.931 660760 833124

4.2 Аналіз розподілу роботи по ядрам процесору
Для того, що б відобразити розподіл роботи по ядрам процесору було обрано лише один випадок - 500 складних моделей. Цього більш ніж досить що би робити висновки про розподіл роботи по ядрам.
За для виміру розподілу роботи по ядрам була використана утиліта «Resource Monitor» описана у розділі 2.5.
На рисунку 4.1 відображено розподіл роботи по ядрам процесору використовуючи програмний інтерфейс до комп'ютерної графіки OpenGL.
На рисунку 4.2 відображено розподіл роботи по ядрам процесору використовуючи програмний інтерфейс до комп'ютерної графіки Vulkan.

Рисунок 4.1 - Розподіл роботи по ядрам процесору з OpenGL

Рисунок 4.2 - Розподіл роботи по ядрам процесору з Vulkan
Як видно з рисунка 4.1, при однопоточному рендері з OpenGL лише п'яте ядро постійно навантажено майже на 100%. А усі інші ядра майже нічим не навантажені. Це не є добре, оскільки рендер сповільнюється тому що повинно чекати на обробку інформації ядром, яке перенавантажено, замість того щоб використовувати інші ядра, як це відбувається при багатопоточному рендері на Vulkan на рисунку 4.2.
4.3 Аналіз зміни ефективності FPS зі збільшенням кількості ядер
На рисунку 4.3 та 4.4 зображено графіки зміни середньої кількості кадрів відображаємих за секунду зі зміною кількості ядер для однопоточного OpenGL та багатопоточного Vulkan типів рендерів для простих та складних моделей.

Рисунок 4.3 - Графіки зміни FPS з простими моделями

Рисунок 4.4 - Графіки зміні FPS з складними моделями
Рисунок 4.3 показує що для десяти простих моделей зріст кількості кадрів відображаємих за секунду застосовуючи багатопоточний рендер на Vulkan майже відсутній починаючи з п'яти ядер. Для ста моделей зріст зупинився на шостому ядрі, для п'ятиста на сьомому ядрі.
Це можна пояснити тим, що прости моделі, які являють собою 3д куб який обертається навколо своєї осі, навантажує процесор досить слабко, так що декілька моделей встигають обробитися лише на одному ядрі, і таким чином навіть для п'ятиста простих моделей, останні ядра не задіяні, і не впливають на ефективність.
Рисунок 4.4 показує що для десяти складних моделей ріст кількості кадрів відображаємих за секунду зупиняється на шостому ядрі, майже як і для простих моделей. Але для ста та навіть п'ятиста моделей, ріст не зупиняється, та є лінійним. Це пояснюється тим, що для скелетної анімації, яка застосовується для складних моделей, треба набагато більше процесорного часу для вирахування матриць та зміщень вершин, також кількість цих вершин значно більша ніж у простих моделей. Таким чином, для усіх ядер є робота, адже одне чи декілька ядер вже не можуть вчасно обробити велику кількість складних моделей, так що би не залишати роботи для решти ядер.
Тут можна зробити висновок, що для 3д моделей навіть значної кількості, у яких мало вершин та прості операції, велика кількість ядер є зайвою. Для малого числа складних моделей велика кількість ядер теж є зайвою, але зі збільшенням таких моделей, можна досить швидко навантажити усі ядра, так що би вони опрацьовувалися паралельно. І залучивши усі ядра процесора, ми досягнемо піку продуктивності, коли усі ядра однаково навантажені, та максимальна кількість анімованих об'єктів обробляється паралельно а не великою чергою.
Що стосується однопоточного рендеру на OpenGL, то його зріст незначно збільшується тільки на двох ядрах. Це пояснюється тим, що операційна система не перериває обробку об'єктів на одному ядрі своїми службовими процесами, а використовує вільне ядро.
Таблиця 4.7 показує різницю у кількості відображаємих кадрів за секунду у процентному відношенні між OpenGL та Vulkan різної кількості ядер для простих моделей. А таблиця 4.8 для складних моделей.
Таблиця 4.7 - Різниця FPS для простих моделей
Кількість ядер Різниця FPS при 10 простих моделей Різниця FPS при 100 простих моделей Різниця FPS при 500 простих моделей
8 45% 108% 111%

7 43% 107% 110%
6 40% 114% 109%
5 40% 97% 94%
4 37% 83% 86%
3 35% 71% 80%
2 31% 54% 68%
1 42% 87% 79%

Таблиця 4.8 - Різниця FPS для складних моделей
Кількість ядер Різниця FPS при 10 простих моделей Різниця FPS при 100 простих моделей Різниця FPS при 500 простих моделей
8 151% 386% 351%
7 153% 311% 299%
6 158% 260% 254%
5 164% 228% 219%
4 133% 171% 179%
3 98% 130% 117%
2 57% 51% 63%
1 12% 0% -1%

Як видно з таблиці 4.7, для простих моделей вулкан ефективніше ніж OpenGL на у середньому на 40% для десяти моделей та на 90% для 100 та 500 моделей. При цьому зміна ефективності зі зміною кількості кадрів є незначною, приблизно 5-10%.
Таблиця 4.8 показує що для складних моделей, одне ядро по ефективності однакове для OpenGL та Vulkan. Але далі зріст ефективності йде приблизно 50% з кожним додатковим ядром. Тож для складних моделей при одному ядрі Vulkan не є ефективним, як при простих моделей. А починаючи з двох та більше ядер Vulkan є досить ефективним, значно збільшуючи кількість кадрів видображаємих за секунду.
4.4 Аналіз зміни GPU пам'яті зі збільшенням кількості ядер
На рисунку 4.5 та 4.6 зображено графіки зміни кількості використаної пам'яті графічної карти зі зміною кількості ядер для однопоточного OpenGL та багатопоточного Vulkan типів рендерів для простих та складних моделей.

Рисунок 4.5 - Графіки зміні пам'яті графічної карти з простими моделями

Рисунок 4.6 - Графіки зміні пам'яті графічної карти з складними моделями
Як видно з рисунків 4.5 та 4.6 пам'ять графічної карти для рендеру анімаційних моделей не збільшується зі збільшенням ядер процесору.
Таблиця 4.9 показує різницю у кількості пам'яті графічної карти у процентному відношенні між OpenGL та Vulkan для різної кількості ядер для простих моделей. А таблиця 4.10 для складних моделей.
Таблиця 4.9 - Різниця пам'яті графічної карти для простих моделей
Кількість ядер Різниця GPU пам'яті при 10 простих моделей Різниця GPU пам'яті при 100 простих моделей Різниця GPU пам'яті при 500 простих моделей
8 25% 32% 32%
7 25% 32% 32%
6 25% 32% 32%
5 25% 32% 32%
4 25% 32% 32%
3 25% 32% 32%
2 25% 32% 32%
1 25% 32% 32%

Таблиця 4.10 - Різниця пам'яті графічної карти для складних моделей
Кількість ядер Різниця GPU пам'яті при 10 складних моделей Різниця GPU пам'яті при 100 складних моделей Різниця GPU пам'яті при 500 складних моделей
8 8% 4% 3%
7 8% 4% 3%
6 8% 4% 3%
5 8% 4% 3%
4 8% 4% 3%
3 8% 4% 3%
2 8% 4% 3%
1 8% 4% 3%

Як видно з таблиці 4.9 Vulkan застосовує на 25% більше пам'яті для 10 простих моделей та на 32% для 100 та 500 моделей.
Для складних моделей, дані для яких наведені у таблиці 4.10, Vulkan застосовує на 8% більше пам'яті для 10 простих моделей та на 3-4% для 100 та 500 моделей.
У середньому для простих моделей графічної пам'яті Vulkan використовується на 25 мегабайт більше ніж для OpenGL. Для складних моделей на 18 мегабайт більше.
4.5 Аналіз зміни оперативної пам'яті зі збільшенням кількості ядер
На рисунку 4.7 та 4.8 зображено графіки зміни кількості використаної оперативної пам'яті зі зміною кількості ядер для однопоточного OpenGL та багатопоточного Vulkan типів рендерів для простих та складних моделей.

Рисунок 4.7 - Графіки зміні оперативної пам'яті з простими моделями

Рисунок 4.8 - Графіки зміні оперативної пам'яті з складними моделями
Як видно з рисунків 4.7 та 4.8 оперативна пам'ять для рендеру анімаційних моделей не збільшується зі збільшенням ядер процесору, так само як і пам'ять графічної карти.
Таблиця 4.11 показує різницю у кількості оперативної пам'яті у процентному відношенні між OpenGL та

Vulkan для різної кількості ядер для простих моделей. А таблиця 4.12 для складних моделей.
Таблиця 4.11 - Різниця оперативної пам’яті для простих моделей
Кількість ядер Різниця RAM пам’яті при 10 простих моделей Різниця RAM пам’яті при 100 простих моделей Різниця RAM пам’яті при 500 простих моделей
8 53% 90% 104%
7 51% 91% 106%
6 48% 88% 96%
5 47% 91% 99%
4 46% 82% 89%
3 44% 80% 87%
2 46% 68% 86%
1 56% 78% 98%

Таблиця 4.12 - Різниця оперативної пам’яті для складних моделей
Кількість ядер Різниця RAM пам’яті при 10 складних моделей Різниця RAM пам’яті при 100 складних моделей Різниця RAM пам’яті при 500 складних моделей
8 26% 37% 36%
7 22% 38% 36%
6 24% 36% 35%
5 24% 36% 36%
4 23% 33% 37%
3 26% 37% 34%
2 29% 37% 32%
1 30% 32% 36%

Як видно з таблиці 4.11 Vulkan застосовує на 50% більше пам’яті для 10 простих моделей та на 90% для 100 та 500 моделей.
Для складних моделей, дані для яких наведені у таблиці 4.12, Vulkan застосовує на 25% більше пам’яті для 10 простих моделей та на 30-35% для 100 та 500 моделей.
У середньому для простих моделей оперативної пам’яті для Vulkan використовується на 20 мегабайт більше ніж для OpenGL для 10 моделей. На 35 мегабайт більше для 100 моделей та на 42 мегабайти більше для 500 моделей.
Для складних моделей на 50 мегабайт більше для 10 моделей, на 174 мегабайти більше для 100 моделей та на 218 мегабайт більше для 500 моделей.
Висновки до розділу 4
У четвертому розділі для кожного типу кожної кількості моделей було зібрано інформацію експериментальним шляхом, таку як середня кількість кадрів за секунду, кількість використаної пам’яті графічного процесору та кількість використаної оперативної пам’яті.
Проаналізувавши цю інформацію можна стверджувати що ріст продуктивності зі збільшенням ядер процесору з багатопоточною моделлю рендеру на Vulkan є досить значним для використання. Але не у всіх випадках. Як що графічні моделі для обробки занадто прості, як 3д куби з декількома вершинами, велика кількість ядер не буде грати ніякої ролі, тому що декілька ядер буде досить для обробки усіх таких моделей.
Для складних моделей зі скелетною анімацією та великою кількість вершин кожне ядро додає суттєвий зріст продуктівності, оскільки навантаження на процесор досить велике, паралельна обробка таких моделей на різних ядер з кожним ядром збільшує кількість кадрів за секунду на 50%.
Також було проаналізовано та показано, що багатопоточний рендер навантажує усі ядра процесору однаково, а однопоточний рендер перевантажує лише одне ядро.
Проаналізувавши споживання пам’яті було встановлено що вона не збільшується зі збільшенням кількості ядер, тільки зі збільшенням кількості моделей.