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 TODO2 додатків:

у вступі описується сутність розробки, її актуальність. Складається із 3 сторінок;

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 TODO22 сторінок;

у третьому розділі представлене проектування й розробка інструментального забезпечення для дослідження. Складається з TODO28 сторінок;

у четвертому розділі описано виконані дослідження. Складається з TODO21 сторінки;

у п‘ятому розділі розкриті питання охорона та безпеки праці. Складається з TODO15 сторінок;

додатки містять технічне завдання й робочий проект

Таблиць – TODO28, рисунків – TODO18, бібліографія – TODO70 джерел.

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

ЗМІСТ

[РЕФЕРАТ 2](#_Toc55749894)

[ЗМІСТ 3](#_Toc55749895)

[ВСТУП 5](#_Toc55749896)

[1 АНАЛІЗ ПРОБЛЕМИ МАСШТАБУВАННЯ ОБРОБКИ АНІМАЦІЇ НА БАГАТОЯДЕРНИХ ПРОЦЕСОРАХ 7](#_Toc55749897)

[1.1 Історія розвитку методів програмування комп’ютерної графіки 7](#_Toc55749898)

[1.1 Огляд старих програмних інтерфейсів комп’ютерної графіки 8](#_Toc55749899)

[1.2.2 Огляд інтерфейсу DirectX 9](#_Toc55749900)

[1.3 Недоліки старих програмних інтерфейсів комп’ютерної графіки 10](#_Toc55749901)

[Щоб виправити такі недоліки може були створені нові графічні програмні інтерфейси. 10](#_Toc55749902)

[1.4 Огляд нових програмних інтерфейсів комп’ютерної графіки 11](#_Toc55749903)

[1.4.1 Огляд інтерфейсу DirectX12 11](#_Toc55749904)

[1.4.2 Огляд інтерфейсу Metal 12](#_Toc55749905)

[1.4.3 Огляд інтерфейсу Vulkan 13](#_Toc55749906)

[1.5 Постановка задачі 14](#_Toc55749907)

[1.6 Огляд літератури 15](#_Toc55749908)

[1.6.1 Огляд мови програмування С++ 16](#_Toc55749909)

[1.6.2 Огляд графічного конвеєру 16](#_Toc55749910)

[ВИСНОВКИ ДО РОЗДІЛУ 1 19](#_Toc55749911)

[2 Обґрунтування експериментального методу дослідження ефективності відображення анімаціЙ на багатоядерних системах 20](#_Toc55749912)

[ВИСНОВКИ ДО РОЗДІЛУ 2 21](#_Toc55749913)

[3 Проектування й розробка інструментального забезпечення для дослідження ефективності відображення анімаціЙ на багатоядерних системах 22](#_Toc55749914)

[ВИСНОВКИ ДО РОЗДІЛУ 3 23](#_Toc55749915)

[4. Дослідження ефективності відображення анімацій на багатоядерних системах 24](#_Toc55749916)

[ВИСНОВКИ ДО РОЗДІЛУ 4 25](#_Toc55749917)

[РЕКОМЕНДАЦІЇ 26](#_Toc55749918)

[література 27](#_Toc55749919)

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 метод 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. 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]

* Значно зменшені накладні витрати на процесор.
* Значно знижене енергоспоживання.
* До (приблизно) 20% покращення ефективності графічного процесора.
* 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:

* Ви хочете відображати 3D-моделі максимально ефективно.
* Ви хочете, щоб ваша гра мала свій унікальний стиль, можливо, з індивідуальним освітленням і затіненням.
* 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.
* У вас є великі числові проблеми, такі як наукове моделювання, які ви можете розділити на незалежні підзадачі, які будуть оброблятися паралельно.
* 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.

Ці програмні графічні інтерфейси були обрані 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. 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.

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 спецификацією |
| 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:

* Які 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.

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:

* Мова шейдерів OpenGL (GLSL)
* Обробка вершин, команди креслення, примітиви, фрагменти та буфери кадрів
* Керування, завантаження та арбітраж доступу до даних
* Створення більших додатків та їх розгортання на різних платформах
* Розширений рендеринг: імітація світла, художні та нефотореалістичні ефекти
* Запобігання та налагодження помилок
* 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 мови програмування С++

Для розробки програми для дослідження була обрана мова програмування С++. Ця мова програмування є об’єктно орієнтованою, має зворотну сумісність з мовою програмування С.

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 С++ ідеально підходить для того, що би використовувати програмні інтерфейси операційної системи без додатків, з можливістю застосування об’єктно орієнтованого підходу

1.6.2 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:
   1. На кожну вершину, отриману з масивів вершин, діє вершинний шейдер. Кожна вершина потоку обробляється по черзі у вихідну вершину.
   2. Необов’язкові примітивні етапи тесселяції.
   3. Необов’язкова примітивна обробка геометричного шейдеру. Вихід - це послідовність примітивів.
2. Поточна обробка вершин, виходи останнього етапу коригуються або передаються в різні місця.
   1. 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.
   2. 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:
   1. Тест на ножиці. Являє собою операцію, яка відкидає фрагменти, що потрапляють за межі певної прямокутної частини екрану.
   2. 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.
   3. Випробування на глибину. Вихідне значення глибини фрагмента може бути перевірено на глибину зразка, на який записується. Якщо тест не вдається, фрагмент відкидається
   4. 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

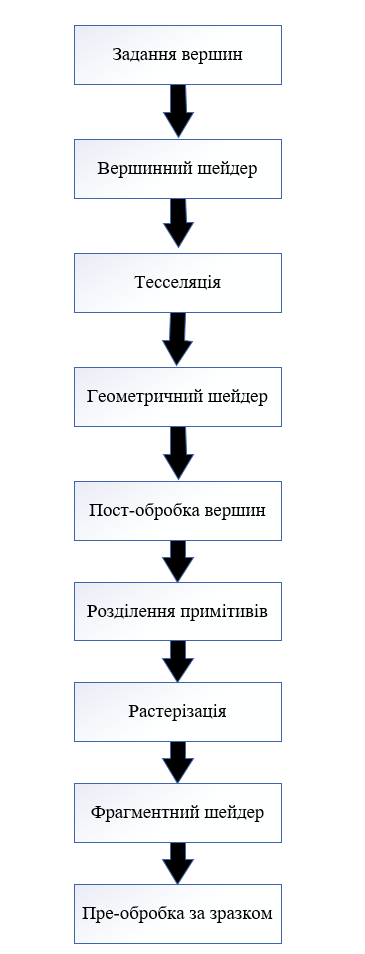


Рисунок 1.1 – Графічний конвеєр

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.

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 скадаєтся 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 Вибір моделей для досліду

h

2.2 Алгоритм обчислювання ефективності анімації

H

2.3 Програмна платформа для досліду

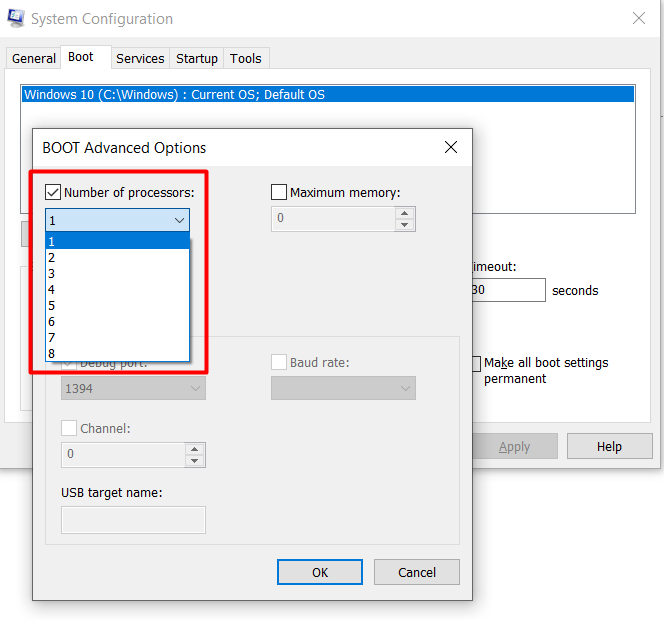
2.4 Апаратна платформа для досліду

2.5 Оцінка ефективності розподілу роботи по ядрам процесору

h

2.6 Порівняння збільшення ефективності OpenGL та Vulkan

h



ВИСНОВКИ ДО РОЗДІЛУ 2

3 Проектування й розробка інструментального забезпечення для дослідження ефективності відображення анімаціЙ на багатоядерних системах

РАСПИСАТЬ ПРО ВСЕ ВНЕШИНЕ БИБЛИОТЕКИ, СТУДИЮ, АПИ, PIMPL…………

ВИСНОВКИ ДО РОЗДІЛУ 3

4. Дослідження ефективності відображення анімацій на багатоядерних системах

ВИСНОВКИ ДО РОЗДІЛУ 4

РЕКОМЕНДАЦІЇ

література

1. MS-DOS графічне програмування [Електронний ресурс] — Режим доступу: <https://retrocomputing.stackexchange.com/questions/11219/how-did-old-ms-dos-games-utilize-various-graphic-cards>
2. Kessenich J. OpenGL Programming Guide Ninth Edition J. Kessenich, G. Sellers, D. Shreiner – Boston: Addison–Wesley, 2016. — 976 с.
3. Sherrod A. Beginning DirectX R 11 Game Programming A. Sherrod, W. Jones – Boston: Cengage Learning PTR, 2011. — 384 с.
4. DirectX 12 [Електронний ресурс] — Режим доступу: <https://docs.microsoft.com/en-us/windows/win32/direct3d12/what-is-directx-12->
5. Begbie С. Metal by Tutorials Second Edition С. Begbie, M. Horga – Boston: Razeware LLC, 2019. — 735 с.
6. Seller G. Vulkan Programming Guide G. Seller – Boston: Addison-Wesley Professional, 2016. — 478 с.
7. Tomas A. Real-Time Rendering 3rd edition А. Tomas, E. Haines, N. Hoffman – Massachusetts: A K Peters, Ltd, 2008. — 1045 с.
8. Графічний конвеєр [Електронний ресурс] — <https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview>
9. <https://learnopengl.com/>
10. <https://vulkan-tutorial.com/>