МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Дніпровський національний університет залізничного транспорту імені академіка В.Лазаряна**

Кафедра Комп’ютерні інформаційні технології

«ДО ЗАХИСТУ»  
Завідувач кафедри КІТ  
проф. Шинкаренко В.І.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ДИПЛОМНА РОБОТА**

на здобуття ОС «Магістр»

Галузь знань 12 Інформаційні технології   
 (шифр) (назва)

Спеціальність 121 Інженерія програмного забезпечення   
 (код) (повна назва)

Освітня програма Інженерія програмного забезпечення   
 (повна назва)

Тема «Дослідження часової ефективності відображення комп`ютерної графіки реального часу на багатоядерних системах з використанням OpenGL та Vulkan API»

Theme «Research of time efficiency of display of computer graphics of real time on multicore systems with use of OpenGL and Vulkan API»

(theme in English)

Керівник дипломної роботи доцент Іванов Олександр Петрович (посада) (підпис) (ПІБ)

Консультант розділу з БЖД ст.викладач \_\_\_\_\_ Музикін Михайло Ігорович   
 (посада) (підпис) (ПІБ)

Консультант економічного розділу доцент \_\_\_\_\_ Гненний Микола Васильович  
  (посада) (підпис) (ПІБ)

Нормоконтролер асистент \_\_\_\_\_ Куроп'ятник Олена Сергіївна  
 (посада) (підпис) (ПІБ)

Студент групи ПЗ1921 (961М). Поліщук Ілля Андрійович  
 (номер групи) (підпис) (ПІБ)

Student Polishchuk Illia   
 (Family name)

Дніпро

2020

Дніпровський національний університет залізничного транспорту імені академіка В. Лазаряна

Факультет Комп'ютерних технологій і систем кафедра Комп’ютерні інформаційні технології

Спеціальність Інженерія програмного забезпечення

«ЗАТВЕРДЖУЮ»  
 Завідувач кафедри

проф. Шинкаренко В.І.

(підпис)

«\_\_\_»\_\_\_\_\_\_\_\_\_2020 р.

**ЗАВДАННЯ**

до дипломної роботи на здобуття ОС Магістр\_\_\_\_\_\_\_\_\_\_\_

(освітній ступень)

студента групи (ПЗ1921) 961-М Поліщука Іллі Андрійовича  
 (номер групи) (ПІБ)

1 Тема дипломної роботи: Дослідження часової ефективності відображення комп`ютерної графіки реального часу на багатоядерних системах з використанням OpenGL та Vulkan API затверджена наказом по університету від «!!!!!TODO!!!!!» листопада 2019 р. № !!!!!TODO!!!!!ст.

2 Термін подання студентом закінченого проекту «!!!!!TODO!!!!!» грудня 2020 р.

3 Вихідні дані до дипломного проекту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4 Зміст пояснювальної записки (перелік питань до розробки) проведення аналізу розробки програмних систем на основі сценаріїв, визначення актуальних проблем, розробка власних методів.

5 Перелік демонстраційного матеріалу презентація формування вимог до систем на основі сценаріїв, результати експериментів, висновки; відео демонстрація роботи розробленого програмного інструментарію для проведення досліджень.

6. Консультанти (з назвами розділів):

|  |  |  |  |
| --- | --- | --- | --- |
| Розділ | Консультант | Підпис, дата | |
| завдання видав | завдання прийняв |
| Техніко-економічні розрахунки | доц. Гненний М.В. |  |  |
| Охорона праці та безпека в надзвичайних ситуаціях | ст. викладач Музикін М.І. |  |  |

КАЛЕНДАРНИЙ ПЛАН

|  |  |  |  |
| --- | --- | --- | --- |
| № п/п | Назва розділів дипломного проекту | Термін виконання розділів проекту (роботи) | Примітка |
| 1 | Вступ | 1.09.2020 – 10.09.2020 |  |
| 2 | Огляд літератури | 10.09.2020 – 15.09.2020 |  |
| 3 | Постановка задачі, технічне завдання | 15.09.2020 – 30.09.2020 |  |
| 4 | Створення тестової програми | 01.10.2020 – 15.10.2020 |  |
| 5 | Перші тестування | 15.10.2020 – 22.10.2020 | 30% |
| 6 | Аналіз результатів | 30.10.2020 – 11.11.2020 |  |
| 7 | Розрахунок економічних показників | 11.11.2020 – 14.11.2020 |  |
| 8 | Охорона праці | 14.11.2020 – 18.11.2020 | 60% |
| 9 | Оформлення пояснювальної записки | 18.11.2020 – 01.12.2020 |  |
| 10 | Демонстраційні матеріали | 5.11.2020 – 16.12.2020 | 100% |

Дата видачі завдання «12» листопада 2019 р.

Керівник дипломного проекту \_\_\_\_\_\_\_\_\_\_\_ Іванов О.П.

(підпис) (ПІБ)

Завдання прийняв до виконання \_\_\_\_\_\_\_\_\_\_\_ Поліщук І .А.

(підпис) (ПІБ)

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 !!!!!TODO!!!!! 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 !!!!!TODO!!!!! сторінок;

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

у четвертому розділі описано виконані дослідження. Складається з !!!!!TODO!!!!! сторінки;

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

додатки містять технічне завдання й робочий проект

Таблиць – !!!!!TODO!!!!!, рисунків – !TODO!, бібліографія – !!TODO!! джерел.

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

ЗМІСТ

[РЕФЕРАТ 4](#_Toc56359467)

[ЗМІСТ 5](#_Toc56359468)

[ВСТУП 8](#_Toc56359469)

[1 АНАЛІЗ ПРОБЛЕМИ МАСШТАБУВАННЯ ОБРОБКИ АНІМАЦІЇ НА БАГАТОЯДЕРНИХ ПРОЦЕСОРАХ 11](#_Toc56359470)

[1.1 Історія розвитку методів програмування комп’ютерної графіки 11](#_Toc56359471)

[1.1 Огляд старих програмних інтерфейсів комп’ютерної графіки 12](#_Toc56359472)

[1.2.1 Огляд інтерфейсу OpenGL 12](#_Toc56359473)

[1.2.2 Огляд інтерфейсу DirectX 13](#_Toc56359474)

[1.3 Недоліки старих програмних інтерфейсів комп’ютерної графіки 14](#_Toc56359475)

[Щоб виправити такі недоліки може були створені нові графічні програмні інтерфейси. 14](#_Toc56359476)

[1.4 Огляд нових програмних інтерфейсів комп’ютерної графіки 15](#_Toc56359477)

[1.4.1 Огляд інтерфейсу DirectX12 15](#_Toc56359478)

[1.4.2 Огляд інтерфейсу Metal 16](#_Toc56359479)

[1.4.3 Огляд інтерфейсу Vulkan 17](#_Toc56359480)

[1.5 Постановка задачі 18](#_Toc56359481)

[1.6 Огляд літератури 20](#_Toc56359482)

[1.6.1 Огляд мови програмування С++ 21](#_Toc56359483)

[1.6.2 Огляд графічного конвеєру 22](#_Toc56359484)

[ВИСНОВКИ ДО РОЗДІЛУ 1 25](#_Toc56359485)

[2 Обґрунтування експериментального методу дослідження ефективності відображення анімаціЙ на багатоядерних системах 26](#_Toc56359486)

[2.1 Поняття та види анімаційних моделей 26](#_Toc56359487)

[2.1.1 По ключовим кадрам: з точки А в точку Б 28](#_Toc56359488)

[2.1.2 Анімація по траєкторії 28](#_Toc56359489)

[2.1.3 Анімація в динамічному середовищі 29](#_Toc56359490)

[2.1.4 Motion capture: перетворення фільму в мультик 29](#_Toc56359491)

[2.2 Вибір моделей для дослідження 29](#_Toc56359492)

[2.2 Алгоритм обчислювання ефективності анімації 29](#_Toc56359493)

[2.3 Аналіз програмних платформ для досліду 30](#_Toc56359494)

[2.3.1 Операційна система Windows 30](#_Toc56359495)

[2.3.2 Операційна система Linux 30](#_Toc56359496)

[2.3.3 Операційна система MacOS 31](#_Toc56359497)

[2.4 Вибір програмної платформи для досліду 31](#_Toc56359498)

[2.5 Аналіз апаратних платформ для досліду 31](#_Toc56359499)

[2.5.1 Процесори з архітектурою ARM 32](#_Toc56359500)

[2.5.2 Процесори з архітектурою x86 32](#_Toc56359501)

[2.5.3 Процесори з архітектурою x86-64 33](#_Toc56359502)

[2.5.4 Відеокарти AMD Radeon 33](#_Toc56359503)

[2.5.4 Відеокарти Nvidia 34](#_Toc56359504)

[2.6 Вибір апаратної платформи для досліду 34](#_Toc56359505)

[2.6.1 Процесор 34](#_Toc56359506)

[2.6.2 Оперативна пам’ять 34](#_Toc56359507)

[2.6.3 Відеокарта 35](#_Toc56359508)

[2.6.4 Материнська плата 35](#_Toc56359509)

[2.6.5 Блок живлення 35](#_Toc56359510)

[2.6.6 Пам’ять диску 35](#_Toc56359511)

[2.5 Оцінка ефективності розподілу роботи по ядрам процесору 35](#_Toc56359512)

[2.6 Порівняння збільшення ефективності OpenGL та Vulkan зі збільшенням кількості ядер процесору 37](#_Toc56359513)

[2.7 Проектування методів обробки анімації графічних об’єктів для їх порівняння та аналізу 38](#_Toc56359514)

[2.7.1 Проектування однопоточної моделі обробки графіки OpenGL 38](#_Toc56359515)

[2.7.2 Проектування багатопоточної моделі обробки графіки Vulkan 39](#_Toc56359516)

[2.9 Вибір метода виміру часу у операційній системі Windows 40](#_Toc56359517)

[ВИСНОВКИ ДО РОЗДІЛУ 2 42](#_Toc56359518)

[3 Проектування й розробка інструментального забезпечення для дослідження ефективності відображення анімаціЙ на багатоядерних системах 43](#_Toc56359519)

[ВИСНОВКИ ДО РОЗДІЛУ 3 44](#_Toc56359520)

[4. Дослідження ефективності відображення анімацій на багатоядерних системах 45](#_Toc56359521)

[ВИСНОВКИ ДО РОЗДІЛУ 4 46](#_Toc56359522)

[РЕКОМЕНДАЦІЇ 47](#_Toc56359523)

[Бібліографічний список 48](#_Toc56359524)

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

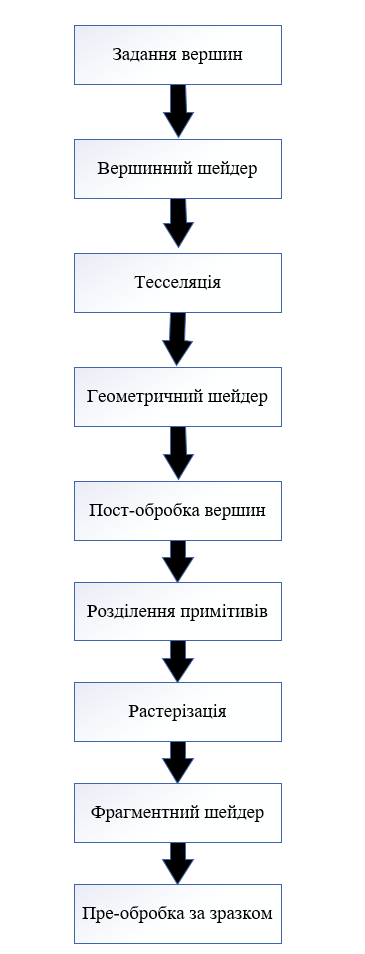


Рисунок 1.1 – Графічний конвеєр

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 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

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 мячик, 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:

* задати точку старту (початок шляху об'єкта);
* позначити траєкторію (шлях, який проробляє об'єкт);
* 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 «пограти» 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:

* на актора закріплюються датчики;
* поки актор рухається, камери фіксують положення датчиків;
* їх зміщення обробляє програма і створює рухомий «скелет» з набором ключових кадрів;
* 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

2.2 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:

(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

2.3 Аналіз програмних платформ для досліду

Були проаналізовані наступні найпошириніші операціїні системи:

2.3.1 Операційна система Windows

Windows (з англ. - «Вікна») - сімейство комерційних операційних систем (OC) 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]

2.3.2 Операційна система 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 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].

2.3.3 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].

2.4 Вибір програмної платформи для досліду

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

2.5 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].

2.5.1 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]

2.5.2 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]

2.5.3 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].

2.5.4 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].

2.5.4 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].

2.6 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.

2.6.1 Процесор

* Модель: AMD Ryzen 7 2700X.
* Тип роз’єму: Socket AM4.
* Кількість ядер: 8.
* Тактова частота: 3700 Мгц.
* Об'єм кеш пам'яті 3 рівня: 16 МБ.

2.6.2 Оперативна пам’ять

* Модель: Kingston HyperX Predator Black (HX432C16PB3K2/16).
* Тип пам’яті: DDR4.
* Максимальна робоча частота: 3200 МГц.
* Максимальна пропускна здатність: 25600 Мб/с
* Ємність одного модуля оперативної пам’яті: 8 гігабайт.
* Загальна кількість встановлених модулів пам’яті: 4.
* Загальна ємність оперативної пам’яті: 32 гігабайти.

2.6.3 Відеокарта

* Модель: GIGABYTE GeForce RTX 2070 SUPER GAMING OC 3X WHITE 8G (GV-N207SGAMINGOC WHITE-8GD).
* Тип пам’яті: GDDR6.
* Інтерфейс: PCI Express 3.0 x16.
* Обсяг пам'яті: 8 гігабайт.
* Частоти роботи GPU: 1815 МГц.
* Частоти роботи пам'яті: 14000 МГц.

2.6.4 Материнська плата

* Модель: Gigabyte B450 AORUS Pro
* Тип сокету процесора: AM4.
* Модель чіпсета: B450.
* Тип пам’яті: DDR4.
* Максимальна робоча частота пам’яті: 3200 МГц.

2.6.5 Блок живлення

* Модель: ATX Seasonic Focus Plus Gold 850W (SSR-850FX).
* Вихідна потужність: 850W.
* Напруга: 110/230 В.
* Частота: 50/60 Гц.

2.6.6 Пам’ять диску

* Модель: Samsung 850 Pro.
* Об’єм: 256 гігабайт.
* Інтерфейс підключення: SATAIII.
* Швидкість читання: до 550 МБ/с.
* Швидкість запису: до 520 МБ/с.

2.5 Оцінка ефективності розподілу роботи по ядрам процесору

Операційна система Windows на якій буде проводитися тестування використовує попереджувальне планування задач.

Попереджувальне планування використовується, 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.

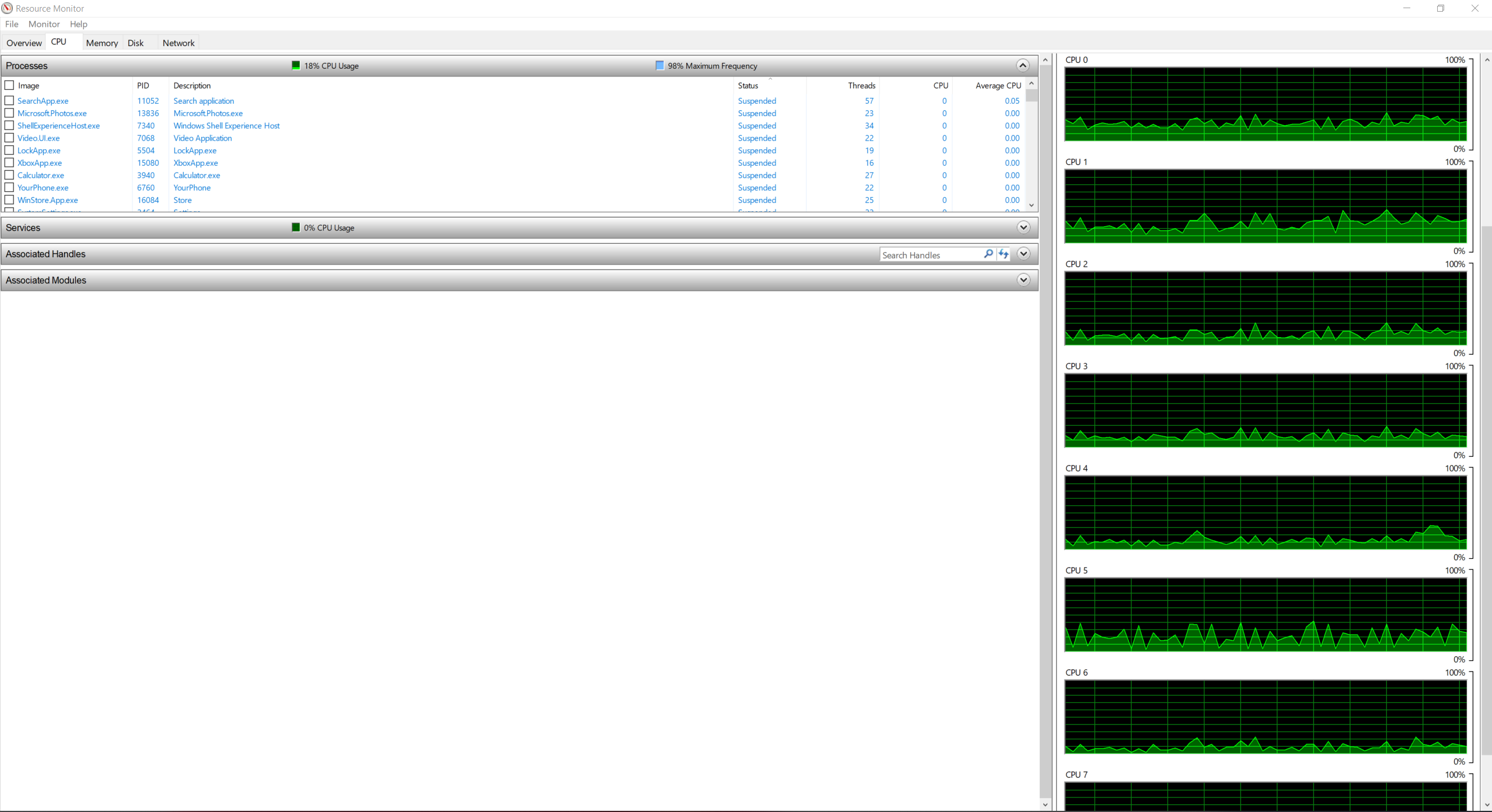


Рисунок 2.1 – Скріншот програми «Resource Monitor»

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.

2.6 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.

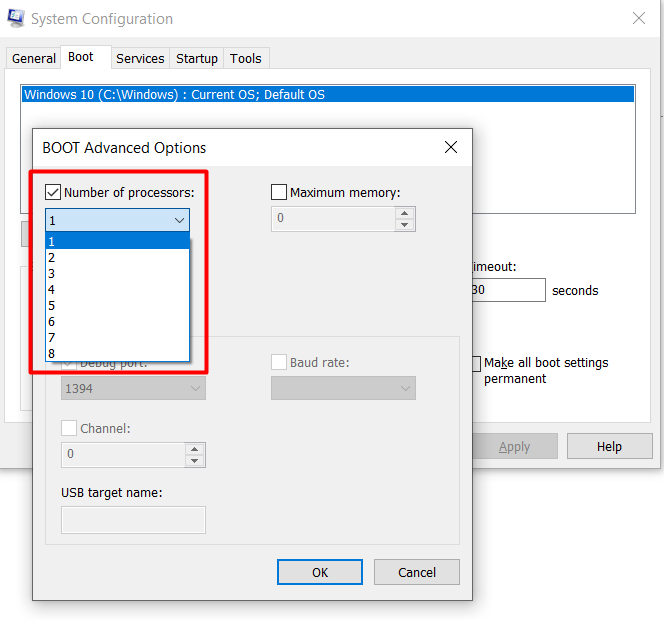


Рисунок 2.2 – Конфігураційне вікно зміни кількості ядер процесорів

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.

2.7 Проектування методів обробки анімації графічних об’єктів для їх порівняння та аналізу

2.7.1 Проектування однопоточної моделі обробки графіки OpenGL

Однопоточна модель обробки та відображення графіки яка є у OpenGL є дуже простою. У загальному виді вона складаєтся 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 2.4 – Однопоточний цикл рендерінгу

2.7.2 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 чі 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.

2.9 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

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

Бібліографічний список

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/>
11. <https://koloro.ua/blog/3d-tekhnologii/3d-model-vidy-urovni-slozhnosti-sostavnye-chasti.html>
12. <https://klona.ua/blog/3d-animaciya/kak-ozhivit-geroya-3d-animatsiya-personazhei>
13. <https://ru.wikipedia.org/wiki/Windows>
14. <https://ru.wikipedia.org/wiki/Linux>
15. <https://ru.wikipedia.org/wiki/MacOS>
16. <https://ru.wikipedia.org/wiki/Центральный_процессор>
17. <https://en.wikipedia.org/wiki/Video_card>
18. <https://en.wikipedia.org/wiki/ARM_architecture>
19. <https://en.wikipedia.org/wiki/X86>
20. <https://en.wikipedia.org/wiki/X86-64>
21. <https://ru.wikipedia.org/wiki/Radeon>
22. <https://ru.wikipedia.org/wiki/Nvidia>
23. <https://www.geeksforgeeks.org/preemptive-and-non-preemptive-scheduling/#:~:text=In%20preemptive%20scheduling%20the%20CPU,or%20switches%20to%20waiting%20state>.
24. <http://www.oszone.net/10487#01>
25. <https://docs.microsoft.com/en-us/windows/win32/dxtecharts/game-timing-and-multicore-processors>