ДОСЛІДЖЕННЯ ЧАСОВОЇ ЕФЕКТИВНОСТІ ВІДОБРАЖЕННЯ КОМП'ЮТЕРНОЇ ГРАФІКИ РЕАЛЬНОГО ЧАСУ НА БАГАТОЯДЕРНИХ СИСТЕМАХ З ВИКОРИСТАННЯМ OPENGL ТА VULKAN API

Автор: Поліщук І. А.
Науковий керівник – к.т.н., доц. Іванов О.П
Дніпровський національний університет імені академіка В. Лазаряна

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.

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.

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.

ПОСТАНОВКА ЗАДАЧІ

Розробити систему для дослідження та дослідити як саме буде змінюватись ефективність відображення анімації з використанням OpenGL та Vulkan, а

саме залежність ефективності від кількості об’єктів анімації та залежність ефективності від кількості ядер процесору.

МЕТОДИКА 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 анімовані за допомогою скелетної анімації.

Для обчислення ефективності анімації була обрана величина середньої кількості кадрів які були опрацьовані та намальовані за певний період часу.

Формула для обчислення середньої кількості кадрів за секунду наведена далі:

$$AFPS = \frac{N}{S}, \qquad (1)$$

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.

Щоб замірити ефективність дослідження зі збільшенням ядер процесору потрібно запустити програму та зробити досліди на системах з різною кількістю ядер. Було вирішено обрати спосіб з конфігуруванням кількості ядер яке буде використовувати операційна система через конфігураційне вікно операційної системи, наведене на рисунку 1 нижче.

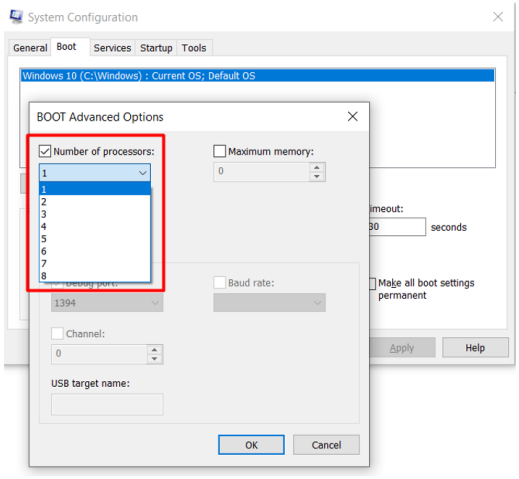

Рисунок 1 – Вікно конфігурації

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.

Однопоточна модель обробки та відображення графіки яка є у OpenGL є дуже простою. У загальному виді вона складається з трьох операцій:

1. Опрацювати ввід користувача
2. Відновити дані для наступного кадру
3. Відобразити кадр на екрані.

Ці операції виконуються у циклі до завершення програми. Схема однопоточного циклу рендерінгу наведена на рисунку 2:

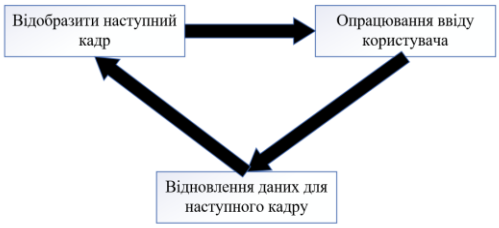

Рисунок 2 – Однопоточний рендер

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 3:

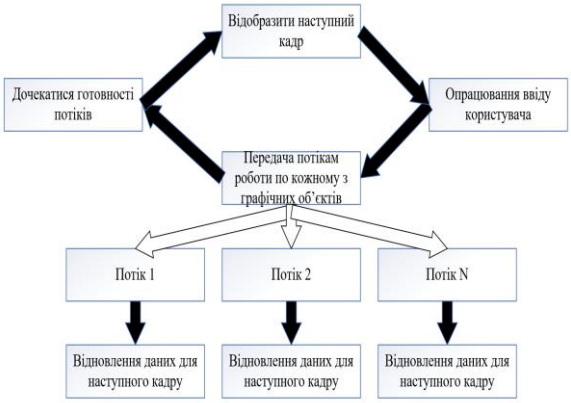

Рисунок 3 – Багатопоточний рендер

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.

ПРОГРАМНЕ ЗАБЕЗБЕЧЕННЯ ДЛЯ ДОСЛІДЖЕННЯ

Для цього дослідження було спроектоване та розроблене програмне забезпечення для рендеру анімованих об’єктів.

У головному меню можливо настроювати кількість об’єктів, тип рендеру та тип об’єкта. Головне меню інструментального засобу показане на рисунку 4:

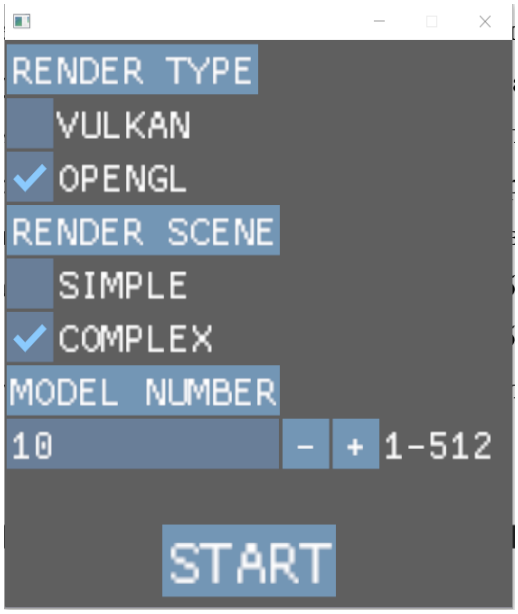

Рисунок 4 – Головне меню ПЗ

Процес рендеру після налаштування розробленого ПЗ показано на рисунку 5 та 6:

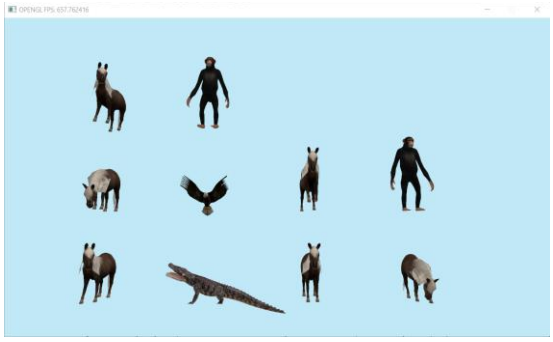

Рисунок 5 – Рендер складних моделей

Рисунок 6 – Рендер простих моделей

Після закінчення рендеру буде показано вікно з вимірами необхідними для дослідження. Вікно з результатами наведено на рисунку 7:

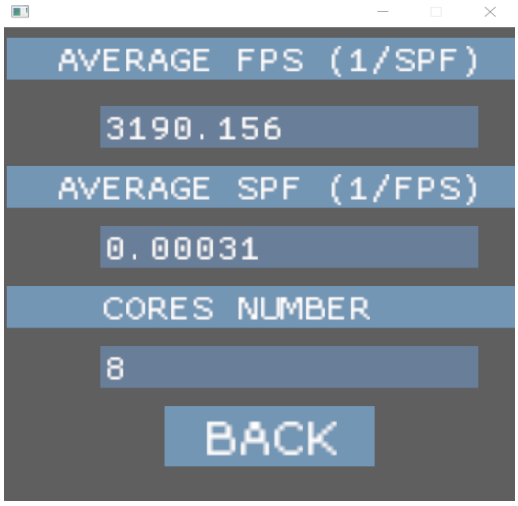

Рисунок 7 – Вікно результатів

РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

У даній роботі для оцінки зміни ефективності використовувалися 10, 100 та 500 анімованих графічних моделей, які одночасно відрисовувалися на екрані під час рендеру. Тести проводилися двічі. Один раз на рендері за допомогою програмного інтерфейсу до комп'ютерної графіки OpenGL. Другий раз на рендері за допомогою програмного інтерфейсу до комп'ютерної графіки Vulkan.

Також іспити подвіювалися через те, що у якості анімованих моделей було використано два типи: прості та складні.

Для кожного типу кожної кількості моделей було зібрано інформацію, а саме:

– Середня кількість кадрів за секунду.
– Кількість використаної пам’яті графічного процесору.
– Кількість використаної оперативної пам’яті

На рисунку 7 та 8 зображено графіки зміни середньої кількості кадрів відображаємих за секунду зі зміною кількості ядер для однопоточного OpenGL та багатопоточного Vulkan типів рендерів для простих та складних моделей:

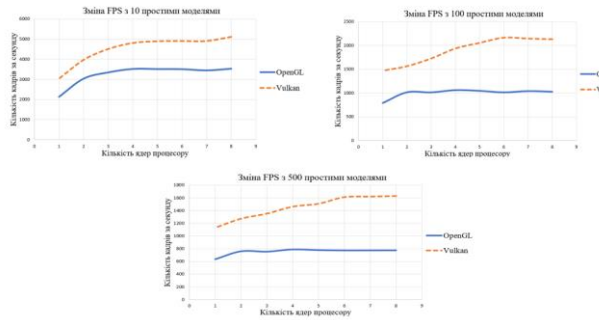

Рисунок 7 – Зміна FPS з простими моделями

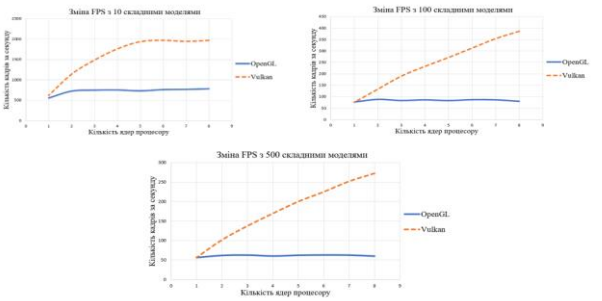

Рисунок 8 – Зміна FPS з складними моделями

Рисунок 7 показує що для десяти простих моделей зріст кількості кадрів відображаємих за секунду застосовуючи багатопоточний рендер на Vulkan майже відсутній починаючи з п’яти ядер. Для ста моделей зріст зупинився на шостому ядрі, для п’ятиста на сьомому ядрі.

Це можна пояснити тим, що прости моделі, які являють собою 3д куб який обертається навколо своєї осі, навантажує процесор досить слабко, так що декілька моделей встигають обробитися лише на одному ядрі, і таким чином навіть для п’ятиста простих моделей, останні ядра не задіяні, і не впливають на ефективність.

Рисунок 8 показує що для десяти складних моделей ріст кількості кадрів відображаємих за секунду зупиняється на шостому ядрі, майже як і для простих моделей. Але для ста та навіть п’ятиста моделей, ріст не зупиняється, та є лінійним. Це пояснюється тим, що для скелетної анімації, яка застосовується для складних моделей, треба набагато більше процесорного часу для вирахування матриць та зміщень вершин, також кількість цих вершин значно більша ніж у простих моделей. Таким чином, для усіх ядер є робота, адже одне чи декілька ядер вже не можуть вчасно обробити велику кількість складних моделей, так що би не залишати роботи для решти ядер.

Що стосується однопоточного рендеру на OpenGL, то його зріст незначно збільшується тільки на двох ядрах. Це пояснюється тим, що операційна система не перериває обробку об’єктів на одному ядрі своїми службовими процесами, а використовує вільне ядро.

На рисунку 9 відображено розподіл роботи по ядрам процесору використовуючи програмний інтерфейс до комп’ютерної графіки OpenGL.

На рисунку 10 відображено розподіл роботи по ядрам процесору використовуючи програмний інтерфейс до комп’ютерної графіки Vulkan.

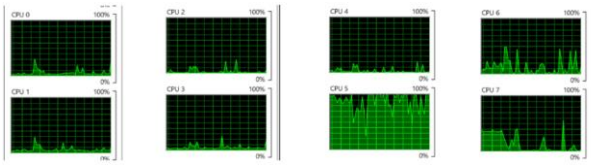

Рисунок 9 – Розподіл роботи по ядрам з OpenGL

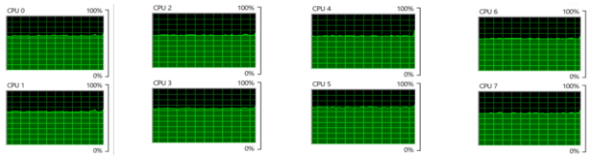

Рисунок 10 – Розподіл роботи по ядрам з Vulkan

Як видно з рисунка 9, при однопоточному рендері з OpenGL лише п’яте ядро постійно навантажено майже на 100%. А усі інші ядра майже нічим не навантажені. Це не є добре, оскільки рендер сповільнюється тому що повинно чекати на обробку інформації ядром, яке перенавантажено, замість того щоб використовувати інші ядра, як це відбувається при багатопоточному рендері на Vulkan на рисунку 10.

На рисунку 11 та 12 зображено графіки зміни кількості використаної пам’яті графічної карти зі зміною кількості ядер для однопоточного OpenGL та багатопоточного Vulkan типів рендерів для простих та складних моделей.

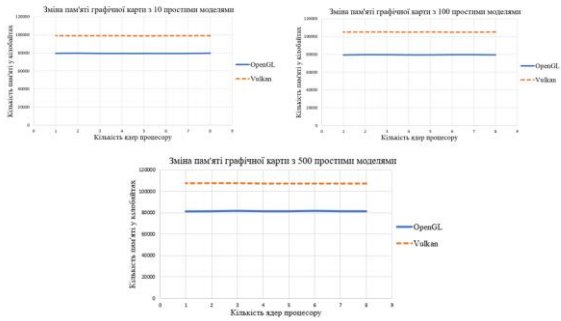

Рисунок 11 – Зміна GPU пам’яті з простими моделями

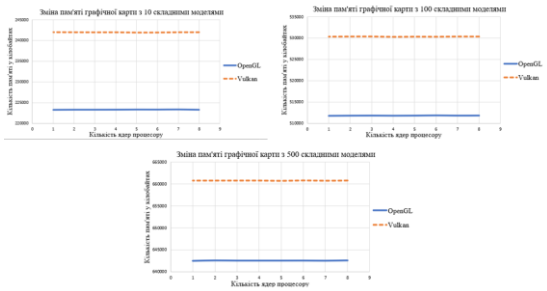

Рисунок 12 – Зміна GPU пам’яті з складними моделями

Як видно з рисунків 11 та 12 пам’ять графічної карти для рендеру анімаційних моделей не

збільшується зі збільшенням ядер процесору, теж саме було встановлено і для оперативної пам’яті.

ВИСНОВКИ

Для 3д моделей навіть значної кількості, у яких мало вершин та прості операції, велика кількість ядер є зайвою. Для малого числа складних моделей велика кількість ядер теж є зайвою, але зі збільшенням таких моделей, можна досить швидко навантажити усі ядра, так що би вони опрацьовувалися паралельно. І залучивши усі ядра процесора, ми досягнемо піку продуктивності, коли усі ядра однаково навантажені, та максимальна кількість анімованих об’єктів обробляється паралельно а не великою чергою. Оскільки навантаження на процесор досить велике, паралельна обробка таких моделей на різних ядер з кожним ядром збільшує кількість кадрів за секунду на 50%.

БІБЛІОГРАФІЧНИЙ СПИСОК

1. Kessenich J. OpenGL Programming Guide Ninth Edition J. Kessenich, G. Sellers, D. Shreiner – Boston: Addison–Wesley, ‘. — 976 с.
2. Seller G. Vulkan Programming Guide G. Seller – Boston: Addison-Wesley Professional, 2016. — 478 с.