Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Методи та засоби оптимізації графічних мобільних застосунків

Виконав: студент IV курсу, групи СНзс-42
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

_____ Чучман А.Ю.
(підпис) (прізвище та ініціали)

Керівник _____ Пасічник В.В.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ Пастух О.А.
(підпис) (прізвище та ініціали)

Тернопіль - 2022

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра комп'ютерних наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«__» _____ 2021 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня Бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп'ютерні науки
(шифр і назва спеціальності)

Студенту Чучман Андрій Юрійович
(прізвище, ім'я, по батькові)

1. Тема роботи Методи та засоби оптимізації графічних мобільних застосунків

Керівник роботи Пасічник Володимир Володимирович, д.т.н., проф. каф. КН
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від 23 » 03 2022 року № 4/7-172

2. Термін подання студентом завершеної роботи 12.06.2022р.

3. Вихідні дані до роботи наукові літературні джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
1. Аналіз предметної області. 1.1. Постановка задач оптимізації. 1.2. Аналіз методів рендерингу на мобільних пристроях. 1.3. Класифікація методів оптимізації графічних додатків на мобільних пристроях. 1.4. Засоби профілювання та налагодження графічних програм. 2. Теоретичне дослідження методів оптимізації графічних додатків. 2.1. Методи оптимізації ресурсів. 2.2. Методи оптимізації шейдерних програм та навантаження GPU. 2.3. Методи оптимізації сцени. 2.4. Методи оптимізації освітлення 3. Стратегія оптимізації графічних програм на мобільних пристроях. 3.1. Алгоритм процесу оптимізації додатку. 3.2. Загальні поради щодо оптимізації 3.3 Застосування стратегії оптимізації графічних програм. 4. Безпека життєдіяльності, основи хорони праці.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Титулка. 2. Актуальність. 3. Мета, задачі дослідження. 4. Засоби профілювання та налагодження графічних програм. 5, 6. Методи оптимізації ресурсів. 7. Методи оптимізації шейдерних програм та навантаження GPU. 8,9. Методи оптимізації сцени. 10. Текстурні атласи. 11. Методи оптимізації освітлення. 12. Блок-схема процесу оптимізації додатку. 13. Висновки. Основні результати проведеного дослідження

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи хорони праці	Гурик О.Я., доцент кафедри МТ		

7. Дата видачі завдання 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	23.03 – 25.03	*Виконано*
2.	Підбір джерел про оптимізацію мобільних графічних		
	застосунків	26.03 – 15.04	*Виконано*
3.	Опрацювання джерел про оптимізацію мобільних		
	графічних застосунків	16.04 – 28.04	*Виконано*
4.	Виконання дослідження щодо оптимізації мобільних		
	графічних застосунків	29.04 – 10.05	*Виконано*
5	Розроблення програмного коду	11.05 – 15.05	*Виконано*
6.	Оформлення розділу «Аналіз предметної області»	15.05 – 20.05	*Виконано*
7.	Оформлення розділу «Теоретичне дослідження методів		
	оптимізації графічних додатків»	21.05 – 26.05	*Виконано*
8.	Оформлення розділу «Стратегія оптимізації графічних		
	програм на мобільних пристроях»	27.05 – 31.05	*Виконано*
9.	Виконання завдання до підрозділу «Безпека		
	життєдіяльності, основи хорони праці»	12.05 – 16.05	*Виконано*
10.	Оформлення кваліфікаційної роботи	16.05 – 26.05	*Виконано*
11.	Нормоконтроль	27.05 – 02.06	*Виконано*
12.	Перевірка на плагіат	03.06 – 07.06	*Виконано*
13.	Попередній захист кваліфікаційної роботи	08.06 – 10.06	*Виконано*
14.	Захист кваліфікаційної роботи	13.06	

Студент _____ Чучман А.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Пасічник В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Методи та засоби оптимізації графічних мобільних застосунків // Чучман Андрій Юрійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп’ютерно-інформаційних систем та програмної інженерії, кафедра комп’ютерних наук, група СНзс–42 // Тернопіль, 2022 // С. – 59, рис. – 25, табл. – 12, слайдів – 13, бібліогр. – 23.

Ключові слова: комп'ютерна графіка, мобільний пристрій, стратегія оптимізації графічних додатків, шейдери, opengl es

Кваліфікаційна робота присвячена розробці стратегії оптимізації програмного забезпечення, що використовує комп'ютерну графіку на мобільних пристроях, засновану на існуючих методах оптимізації.

Розглянуто цілі оптимізації графічних додатків для мобільних пристроїв, сформовано завдання оптимізації таких додатків. Проаналізовано сучасний підхід до візуалізації графіки на мобільних пристроях. Сформовані класифікації методів оптимізації графічних додатків за апаратною архітектурою та об'єктом оптимізації. Розглянуті засоби профілювання додатків на мобільних пристроях. Збір та аналіз даних про поведінку програмного забезпечення є першим та найважливішим кроком у процесі оптимізації. Застосування спеціальних утиліт для профілювання програмного забезпечення дозволяє ефективно впоратися з цим завданням.

Проаналізовано ефективність різних методів оптимізації графічних додатків на мобільних пристроїв. На основі отриманих даних було сформовано стратегію оптимізації таких додатків та продемонстровано її ефективність.

ANNOTATION

Methods and means for graphical mobile applications optimization // Chuchman Andriy // Ternopil Ivan Pul'uj National Technical University, Faculty of Computer Information Systems and Software Engineering, Department of Computer Science // Ternopil, 2022 // P. - 59, Fig. - 25, Table - 12, Slide - 13, References - 23.

Keywords: computer graphics, mobile device, optimization strategy of graphic applications, shaders, opengl es

Thesis deals with the development of software optimization strategy that uses computer graphics on mobile devices, based on existing optimization methods.

The goals of optimization of graphic applications for mobile devices are considered, the task of optimization of such applications is formed. The modern approach to graphics visualization on mobile devices is analyzed. Classifications of methods of optimization of graphic applications on hardware architecture and object of optimization are formed. Means of profiling applications on mobile devices are considered. Collecting and analyzing software behavior data is the first and most important step in the optimization process. The use of special utilities for software profiling allows you to effectively cope with this task.

The effectiveness of various methods of optimizing graphics applications on mobile devices is analyzed. Based on the obtained data, a strategy for optimizing such applications was formed and its effectiveness was demonstrated.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ СКОРОЧЕНЬ І ТЕРМІНІВ

ETC (Ericsson Texture Compression) – техніка стиснення текстури з втратами.

FPS (frame per second) - кількість кадрів на секунду.

GPU (graphics processing unit) - графічний процесор.

GUI (graphical user interface) – графічний інтерфейс користувача

HSR (hidden surface removal) - видалення невидимих поверхонь.

IMR (immediate mode rendering) - режим негайної візуалізації.

LOD (level of detail) - рівень деталізації.

PNG (Portable Network Graphics) - растровий формат збереження графічної інформації, що використовує стиснення без втрат.

PVRTC (PowerVR Texture Compression) - формат компресії графічних даних із втратами

TBDR (tile based deferred rendering) - відкладена тайлова візуалізація.

TBR (tile based rendering) - тайлова візуалізація.

UI (user interface) – інтерфейс користувача.

КГ – комп’ютерна графіка.

МП – мобільний пристрій.

МО –методи оптимізації.

ОС – операційна система.

ПК – персональний комп’ютер.

ПЗ – програмне забезпечення.

ЗМІСТ

ВСТУП

Актуальність теми. В останні роки широке зростання отримала мобільна обчислювальна техніка. Повсюдне поширення МП, таких як смартфони та планшети, спричинило розширення сектора ПЗ для цих пристроїв. Одним з найпопулярніших видів ПЗ, що використовується на МП є ПЗ мультимедійного характеру. В його основі лежить застосування КГ.

На поточному рівні розвитку МП їх продуктивність поки не може зрівнятися з продуктивністю ПК. МП балансують між обчислювальною потужністю, терміном служби батареї та вартістю. Це означає, що такі ресурси обмежені в мобільних платформах порівняно з настільними платформами:

– обчислювальна потужність;
– обсяг пам'яті;
– пропускна здатність пам'яті;
– споживання енергії;
– фізичний розмір.

ПК не мають цих обмежень, тому розробники можуть використовувати набагато більше обчислювальних ресурсів. Внаслідок цих обмежень, гостро постає питання оптимізації ПЗ з урахуванням специфіки функціонування МП. Таким чином актуальність роботи зумовлена високою затребуваністю ПЗ з використанням КГ на МП. А також з обмеженнями, що накладаються такими пристроями на склад ПЗ.

У кваліфікаційній роботі розглядається проблема створення ПЗ, що використовує КГ на МП. На сьогоднішній день попит на подібне ПЗ надзвичайно високий, однак рівень існуючих технологій накладає обмеження на обчислювальні можливості МП. Проблеми оптимізації КГ на МП широко досліджуються різними авторами, зокрема Е. Бірсом, Т. Аріано, А.Ліма, К.Пуллі та іншими [2-5].

Мета дослідження – створення стратегії оптимізації ПЗ, що використовує КГ на МП, заснованій на наявних МО.

Для досягнення мети виділено ряд задач:

– класифікація наявних МО;

– виконання порівняльного аналізу існуючих засобів та МО КГ;

– аналіз сучасних підходів до оптимізації графічних програм для МП;

– дослідити різні МО КГ для МП, провести оцінку ефективності цих методів на сучасних МП;

– формування стратегії оптимізації на основі даних аналізу МО.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задач оптимізації

Процес оптимізації КГ може переслідувати різні цілі, такі як:

– збільшення частоти кадрів;

– зменшення енергоспоживання за рахунок меншого розміру даних, що передаються та меншої кількості тактів для кожного кадру;

– скорочення об’єму оперативної пам'яті;

– зменшення розміру даних, що завантажуються.

Різні МО часто взаємопов'язані. Наприклад, можна використовувати оптимізацію частоти кадрів як засіб для економії енергії. Цього можна досягти за рахунок збільшення швидкості рендерингу кадру, але обмежуючи частоту кадрів до нижчого рівня. Це дозволяє заощаджувати енергію, оскільки GPU вимагає менше часу для обчислення кадрів і може залишатися неактивним протягом тривалих періодів часу.

Оптимізація для зменшення обсягу пам'яті програми не є типовим для оптимізації, але це може бути корисним, оскільки невеликі програми більше кешуються. Програми меншого обсягу працюють швидше за рахунок системи кешування.

У оптимізації КГ для МП виділяють два основні завдання оптимізації [2]:

– отримання максимальної продуктивності програми при заданій пам'яті;

– отримання мінімальної оперативної пам'яті при заданій продуктивності.

Критерієм оптимальності максимальної продуктивності є кадрова частота на кінцевому пристрої. Це кількість кадрів, що змінюються за одиницю часу. Загальноприйнята одиниця виміру - кадри в секунду (FPS). Для КГ реального часу загальноприйнято два значення кадрової частоти 30 FPS та 60 FPS.

Обсяг займаної оперативної пам'яті в байтах є критерієм завдання оптимізації пам'яті.

1.2 Аналіз методів рендерингу на МП

Будь-які МО КГ ґрунтуються на існуючих методах рендерингу.

Рендеринг – це процес отримання зображення за допомогою КГ. Класичний метод рендерингу на ПК та консолях – це IMR. На рис. 1.1 показаний графічний конвеєр для IMR.

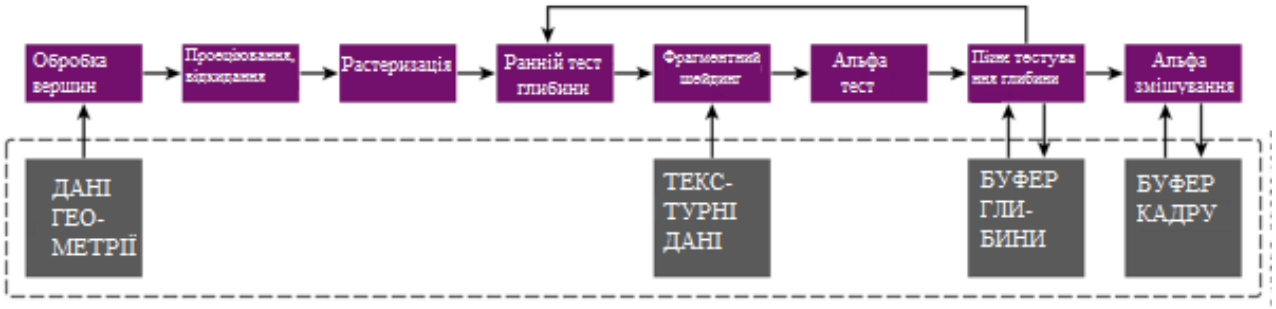

Рисунок 1.1 – Графічний конвеєр IMR

ІМЯ складається з таких основних етапів:

– обробка вершин, вершинний шейдер, та проектування вершин у систему координат дисплея. Також включає читання даних геометрії із системної пам'яті GPU;

– растеризація трикутників;

– фрагментна обробка, фрагментний (піксельний) шейдер – накладання текстур тощо;

– альфа-тестування;

– тест глибини фрагмента та запис фінальної глибини фрагмента в буфер глибини;

– альфа змішування фрагмента та запис вихідних даних у буфер кадру.

Такий підхід містить суттєві недоліки, що не дозволяютьзастосовувати IMR у МП. А саме IMR обробляє весь кадр повністю, що вимагає значних витрат пам'яті, для зберігання буферів, текстур і вершинних даних. Через неможливість ефективно використовувати кеш графічних ядер GPU, збільшується кількість звернень до зовнішньої відеопам'яті, що у свою чергу накладає жорсткі вимоги

щодо мінімальної частоти роботи GPU.

Однак у МП неможливо застосовувати GPU з високою частотою, так як це викликає нагрівання пристроїв і сильну витрату електроенергії.

Тому основний спосіб рендерингу на МП - це тайловий рендеринг. Це процес розбиття зображення на сітку у просторі дисплея та рендеринг кожного осередку сітки, тайлу, окремо. Перевага такого методу рендерингу, у порівнянні IMR в тому, що рендеринг невеликих тайлів зображення дозволяє більш ефективно використовувати кеш пам'ять GPU, і скорочує кількість звернень до зовнішньої відеопам'яті [5]. На рис. 1.2 представлений графічний конвеєр для TBDR.

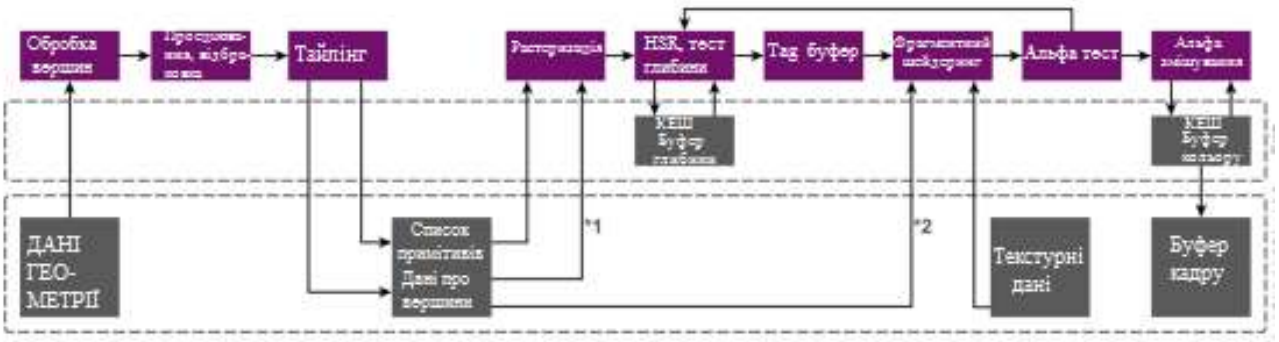

Рисунок 1.2 – Графічний конвеєр TBDR

Після закінчення роботи вершинного шейдера в TBDR відбувається процес розбиття зображення на тайли.

На рис. 1.3 представлений приклад розбиття буфера кадру на тайли.

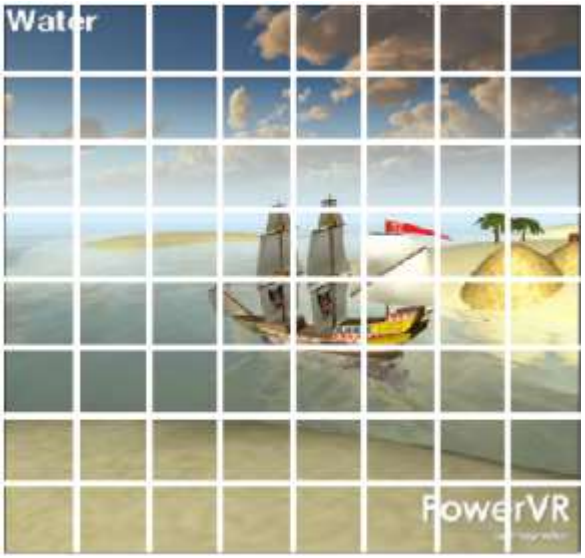

Рисунок 1.3 – Буфер кадра, розбитий на тайли

Під час процесу тайлінгу GPU визначає списки примітивів, що належать до конкретних тайлів. Надалі ці списки використовуватимуться у процесі растеризації. Одна з особливостей тайлового рендерингу - це багаторазова розтеризація примітивів, які потрапляють у кілька тайлів відразу (див. рис. 1.4).

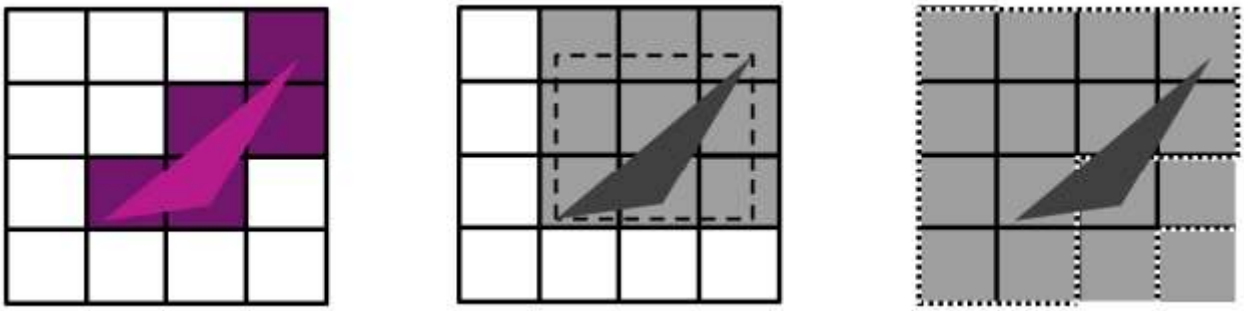

Рисунок 1.4 – Визначення належності примітиву до тайла

Після закінчення процесу растеризації TBDR відбувається процес видалення невидимих поверхонь HSR. У традиційному IMR цей крок графічного конвеєра відсутній. HRS призначений для мінімізації багаторазового перемальовування (overdraw) одних і тих же фрагментів, за рахунок видалення тих фрагментів, котрі приховані. Для непрозорої геометрії застосування HSL дозволяє повністю усунути перемальовку, що у свою чергу істотно зменшує вимоги до використання зовнішньої відеопам'яті. На рис 1.5 – 1.7. показано

порівняння кількості перемальовок із застосуванням IMR та TBDR.

Однак застосування HSR накладає обмеження на роботу з буфером глибини у фрагментному шейдері. При такому підході буфер глибини доступний тільки для читання, але не для запису, тому що всі операції з визначення глибини об'єкта були виконані до фрагментованого фрагмента шейдингу.

Рисунок 1.5 – Оригінальний кадр

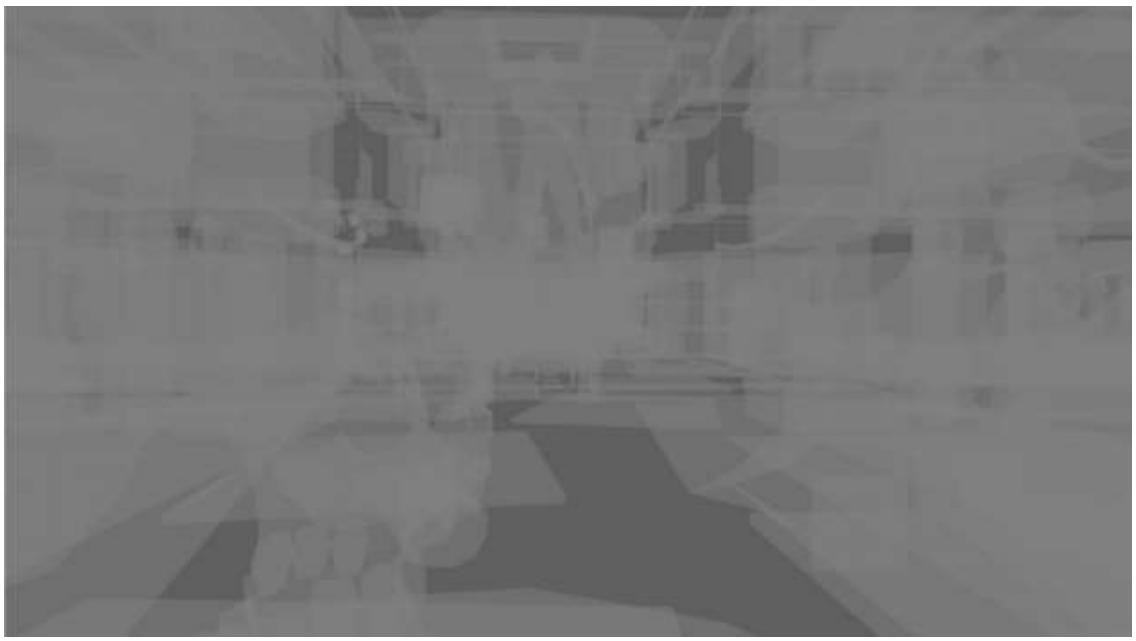

Рисунок 1.6 – Перемалювання із застосуванням IMR

Рисунок 1.7 – Перемалювання із застосуванням TBDR

Загалом застосування тайлового рендерингу з техніками TBDR дозволило суттєво знизити вимоги до GPU, дозволило ефективно застосовувати КГ МП. Проте графічний конвеєр TBDR накладає деякі обмеження щодо застосування технік КГ, такі як робота з буфером глибини, альфа-тест та альфа- змішування. Ці обмеження необхідно враховувати у процесі оптимізації графіки під дану модель рендерингу.

1.3 Класифікація методів опттмізації графічних додатків на МП

За апаратною архітектурою. Графічна складова будь-якого ПЗ заснована на двох апаратних підсистемах обчислювального пристрою: CPU та GPU. Всі МО КГ можна віднести до оптимізації, котра виконується на CPU або GPU. МО цих підсистем сильно відрізняються.

Для візуалізації об'єктів на дисплеї, ПЗ необхідно виконати велику кількість операцій, таких як розрахунок впливу джерел світла на об'єкти, сортування об'єктів у сцені, передача даних про об'єкти, що малюються на GPU при допомозі графічного API.

Зростання числа об'єктів у сцені спричиняє падіння продуктивності програми. Більшість оптимізацій на CPU спрямовані на зменшення числа

об'єктів, що обробляються, спрощення алгоритмів обробки цих об'єктів, а також зменшення кількості команд відображення переданих на GPU.

GPU оптимізації в першу чергу засновані на моделі графічного конвеєра на цільовій платформі. Для мобільних платформ це модель TBDR. Оптимізації, що застосовуються на GPU, спрямовані на спрощення шейдерних програм та використання графіки спеціально підготовленої для рендерингу за допомогою цільової моделі графічного конвеєра.

За об'єктом оптимізації у КГ можна навести таку класифікацію МО:

– ресурсів: моделей, текстур, шрифтів;
– сцени;
– шейдерних програм;
– освітлення;
– роботи з графічним API.

МО ресурсів спрямовані на зменшення обсягу використовуваних ресурсів та їх ефективне використання під час виконання програми. У КГ застосовуються 3 основних види ресурсів – це моделі, текстури та шрифти. Ці види ресурсів можуть займати до 98% усієї споживаної оперативної пам'яті ПЗ. Їх оптимізація один із найважливіших етапів у процесі оптимізації додатку. Основна мета цієї групи методів - зменшення споживання оперативної пам'яті, відеопам'яті, а також зменшення розміру скомпільованого додатка.

МО сцени та ієрархії об'єктів спрямовані на спрощення структури сцени, зменшення кількості об'єктів оброблюваних CPU у кожному кадрі. Основна мета цієї групи методів – зниження навантаження на CPU.

МО шейдерних програм спрямовані на спрощення складності обчислень використовуваних у шейдерах. Методи цієї групи спрямовані на зниження навантаження на GPU.

МО освітлення включають техніки з різним способом обробки освітлення, методи цієї групи направлені однаково на зменшення навантаження на CPU і GPU.

Оптимізація роботи з графічним API використовує методи, котрі направлені на ефективне застосування графічного API МП. Робота з OpenGL ES,

Vulcan та Metal API. Усі ці методи направлені на зменшення навантаження на CPU.

1.4 Засоби профілювання та налагодження графічних програм

Одним із найважливіших пунктів у процесі оптимізації графічних додатків є виявлення проблемних місць у додатку. Для цього застосовуються різні засоби профілювання та налагодження графічних програм.

Профілювання - це процес збору параметрів роботи програми, таких як час виконання окремих фрагментів, число чітко передбачених умовних переходів, число кеш-промахів, обсяг споживаної оперативної пам'яті і т.д. Інструмент, який використовується для аналізу роботи, називають профайлером (англ. profiler). Процес профілювання виконується разом із оптимізацією програми [7].

Характеристики можуть бути апаратними (час) або викликані ПЗ (функціональний запит). Інструментальні засоби аналізу програми є надзвичайно важливими для того, щоб зрозуміти поведінку програми. Проектувальники ПЗ потребують таких інструментальних засобів, щоб оцінити, як добре виконана робота. Програмісти потребують інструментальних засобів, щоб проаналізувати їхні програми та ідентифікувати критичні ділянки програми.

Основні дані, які необхідно аналізувати в ході оптимізації графічного ПЗ:

- навантаження на CPU;
- навантаження на GPU;
- кількість звернень до графічного API (drawcalls);
- споживання оперативної пам'яті програмою.

1.4.1 Xcode Instruments

Це набір утиліт для профілювання та аналізу продуктивності додатків для ОС X та iOS. Instruments входить до складу середовища розробки Xcode (рис. 1.8).

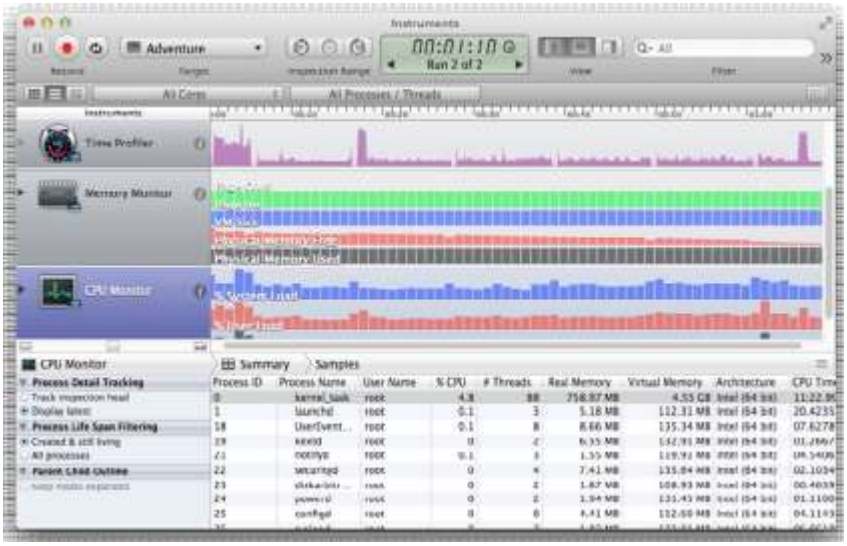

Рисунок 1.8 – Інтерфейс Xcode Instruments

До складу Instruments входять утиліти, які застосовуються при аналізі графічних додатків:

– Time Profiler - дозволяє виміряти навантаження на CPU;

– Allocations - дозволяє відстежувати споживану додатком оперативну пам'ять;

– Leaks - дозволяє відстежувати витоку в оперативній пам'яті;

– OpenGL ES Analytics - дозволяє відстежувати звернення до OpenGL ES API. І навантаження на GPU;

– Metal System Trace - дозволяє відстежувати навантаження на GPU при використанні Metal API.

1.4.2 RenderDoc

Графічний відладчик з відкритим вихідним кодом, для ОС Windows та Linux (рис. 1.9).

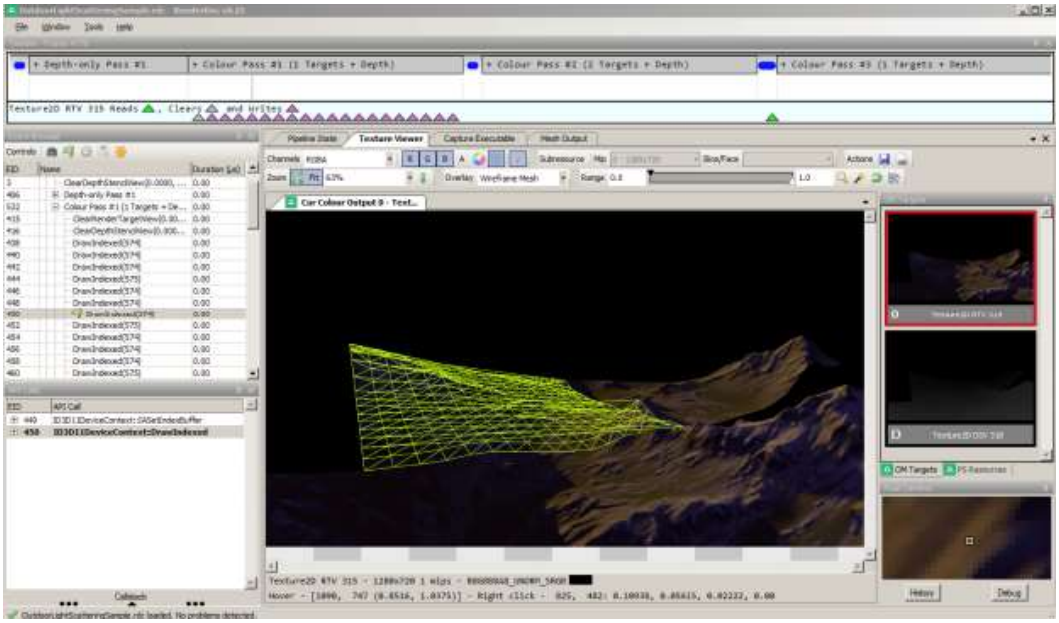

Рисунок 1.9 – Вікно відладки RenderDoc

RenderDoc - дозволяє налагоджувати графічну підсистему програм, що використовують OpenGL, Vulcan і DirectX API.

Перевагами RenderDoc є:

– відкритий вихідний код;
– підтримка сучасних графічних API;
– кросплатформність;
– налагодження шейдерів;
– покадрове налагодження програми.

1.4.3 Unity Profiler

Це утиліта для профілювання графічних програм розроблених на движку Unity (рис. 1.10). Включає в себе утиліти для моніторингу навантаження на CPU, GPU, кількість звернень до графічного API та утиліту для профілювання пам'яті, що споживається. Перевагами цього профайлера є кросплатформеність та можливість профілювання програми на цільовому пристрої.

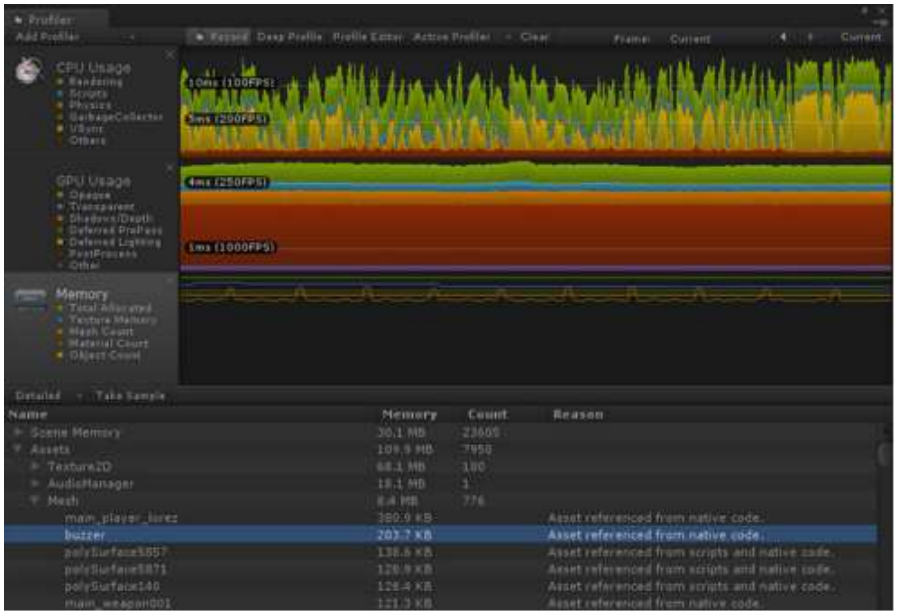

Рисунок 1.10 – Інтерфейс Unity Profiler

РОЗДІЛ 2. ТЕОРЕТИЧНЕ ДОСЛІДЖЕННЯ МО ГРАФІЧНИХ ДОДАТКІВ

2.1 Методи оптимізації ресурсів

У сучасному ПЗ із застосуванням КГ реального часу ресурси можуть займати до 98% всієї споживаної програми оперативної пам'яті [12]. У свою чергу текстури є основним ресурсом, що використовується в таких типах ПЗ. Частка текстур від усіх ресурсів ПЗ з КГ може займати до 99%, а вихідні дані для цих ресурсів варіюються від 1Гб - 2Гб для середніх проектів, і можуть вимірюватися десятками гігабайт для великих [12]. Очевидно, що у вихідному вигляді використовувати такі ресурси на МП неможливо. Тому застосовуються різні способи оптимізації ресурсів.

2.1.1 Компресія текстур

СТ або компресія текстур (Texture compression) - це технологія стиснення зображення, що візуально відображає сукупність властивостей поверхні будь-якого об'єкта, призначене для зберігання текстурного атласу в 3D комп'ютерній графіці систем візуалізації. На відміну від звичайних алгоритмів стиснення зображень алгоритми СТ оптимізовані для випадкового доступу [9]. СТ як засіб оптимізації дозволяє зменшити обсяг споживаної оперативної пам'яті, зменшити кількість текстурних даних, що передаються на GPU.

Для методів стиснення даних текстурних карт істотні дві вимоги: стиснення практично без втрати якості та обробка даних «на льоту» в процесі перетворення та накладання текстур. В основній роботі зі СТ [8], Ендрю Бірс вказав чотири основні особливості СТ для КГ реального часу:

– швидкість декодування. Треба мати швидку декомпресію безпосередньо з стислих даних текстури, щоб не впливати на продуктивність візуалізації;

– довільний доступ. Візуалізація буде утруднена без передбачення порядку доступу текселів, тобто будь-яка схема СТ має дозволити швидкий довільний доступ до декомпресованих даних текстури;

– ступінь стиснення та візуальна якість. В системі рендерингу стиснення з втратами може бути більш терпимим, ніж для інших випадків застосування;

– швидкість кодування.

Варто навести особливості основних форматів стиснення.

PNG - це растровий формат зберігання зображення, особливий тим, що використовується спосіб стиснення графічних даних без втрат інформації. Він підтримує кольорові індексовані зображення (24 біти RGB або 32 біти RGBA), повнокольорові та напівтонові зображення, а також - альфа-канал.

Переваги:

– оскільки графічні дані стискаються без втрат, тому PNG -зображення дуже якісні;

– підтримує і 8-бітну і 16-бітну прозорість.

Недоліки:

– файли мають великі розміри. Це збільшує розмір програм та споживання пам'яті;

– порівняно висока потреба у обчислювальних ресурсах (що веде до погіршення продуктивності).

ETC. Це формат CT, який оперує блоками пікселів розміром 4x4. Спочатку Khronos використовував ETC як стандартний формат для Open GL ES 2.0. (Ця версія ще називається ETC1). В результаті цей формат доступний практично на всіх Android -пристроях. З виходом OpenGL ES 3.0. як новий стандарт використано формат ETC2 - перероблена версія ETC1. Основна відмінність між цими двома стандартами полягає в алгоритмі, що оперує піксельними групами. Поліпшення алгоритму призвели до вищої точності виведення дрібних деталей зображень. Як результат, якість зображень покращилася, а розмір файлів – ні. ETC1 і ETC2 підтримують стиснення 24-бітних даних RGB , але вони не підтримують стиснення зображень з альфа-каналом. Крім того, є два різні формати файлів, що відносяться до алгоритму ETC: це KTX та PKM.

KTX - це стандартний формат файлу Khronos Group, він надає контейнер, в якому можна зберігати безліч зображень. Коли MTP- карта створюється за допомогою KTX, генерується єдиний KTX- файл. Формат PKM -файлу набагато

простіше, такі файли, в основному, використовують для зберігання окремих зображень. Як результат, при використанні PKM в ході створення MTP -карти вийдуть кілька PKM -файлів замість KTX. Тому для зберігання MTP -карток використовувати формат PKM не рекомендується.

Переваги:

– розмір ETC -файлів, помітно менший за розмір PNG -файлів;

– формат підтримує апаратне прискорення практично на всіх Android -пристроях.

Недоліки:

– якість не така висока, як у PNG;

– немає підтримки прозорості.

Для стиснення зображень в ETC можна використовувати Mali GPU Texture Compression Tool та ETC-Pack Tool.

PVRTC - це формат компресії графічних даних із втратами, з фіксованим рівнем стиснення, який використовується переважно у пристроях Imagination Technology PowerVR MBX, SGX і Rogue. Він застосовується як стандартний метод стиснення зображень в iPhone, iPod, iPad.

На відміну від ETC та S3TC, алгоритм PVRTC не працює з фіксованими блоками пікселів. У ньому використовується білінійне збільшення та змішування з низькою точністю двох зображень низької роздільної здатності. На додаток до унікального процесу стиснення, PVRTC підтримує формат RGBA (з прозорістю) і для варіанта 2-bpp (2 біти на піксель), і для варіанта 4-bpp (4 біти на піксель).

Переваги:

– підтримка альфа-каналів;

– підтримка RGBA для варіанта 2-bpp (2 біти на піксель) та для варіанту 4-bpp (4 біти на піксель);

– розмір файлів набагато менший, ніж у PNG;

– підтримка апаратного прискорення на GPU PoverVR.

Недоліки:

– якість не така висока, як при використанні PNG;

– формат підтримується лише на апаратному забезпеченні PoverVR;

– забезпечується підтримка квадратних POT - текстур, тобто текстур, ширина і висота яких є ступенем числа 2, хоча в деяких випадках є підтримка прямокутних текстур;

– СТ у цей формат може бути повільним.

Для стиснення можна використовувати PVRTexTool.

У таблиці 2.1 представлений порівняльний аналіз ступеня стиснення різних алгоритмів стиснення для текстур розміром 2048 * 2048 пікселів, з альфа каналом і без.

Таблиця 2.1 - Дані тестування ступеня СТ різних форматів

Формат	Роздільна здатність текстури (в пкс)	Альфа-канал	Розмір текстури
PNG	2048	Ні	16 Mb
PNG	2048	Так	21.3 Mb
ETC 4 bit	2048	Ні	2.7 Mb
ETC 8 bit	2048	Так	5.3 Mb
PVRTC	2048	Ні	2.7 Mb
PVRTC	2048	Так	2.7 Mb
PNG	1024	Ні	4 Mb
PNG	1024	Так	5.3 Mb
ETC 4 bit	1024	Ні	0.7 Mb
ETC 8 bit	1024	Так	1.3 Mb
PVRTC	1024	Ні	0.7 Mb
PVRTC	1024	Так	0.7 Mb

З таблиці видно, що застосування алгоритмів СТ із втратами дозволяє значно скоротити обсяг текстур. Слід також зауважити, що через особливості алгоритму стиснення PVRTC, альфа канал не впливає на розмір вихідної текстури.

2.1.2 Комбінування текстурних карт

Це спосіб оптимізації текстур у якому поєднують кілька чорно-білих текстур у різних каналах однієї текстури [13]. На рис. 2.1 – 2.3 показані 3

текстури, об'єднані в різних каналах в одній текстурі - рис. 2.4.

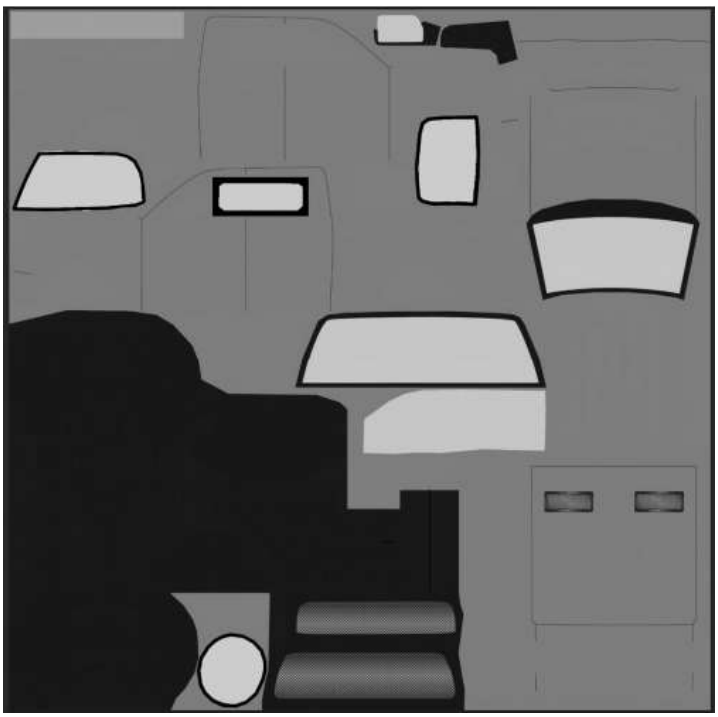

Рисунок 2.1 – Маска відображень (R-канал)

Рисунок 2.2 – Маска області фарбування (G-канал)

Рисунок 2.3 – Маска Ambient Occlusion (B-канал)

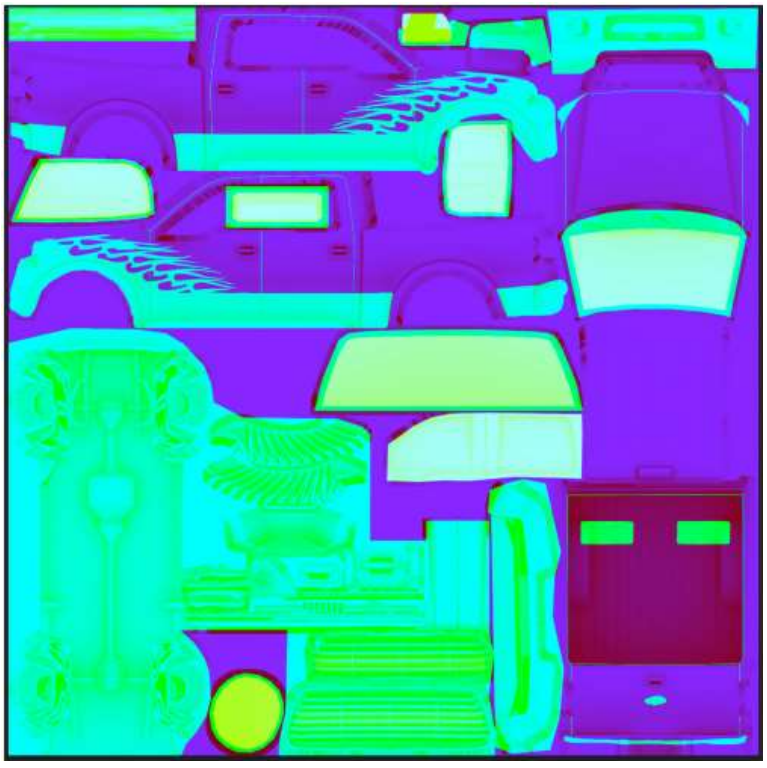

Рисунок 2.4 – Комбінована текстура

Метод комбінування декількох текстур в одній дозволяє скоротити обсяг ресурсів до 4 разів, при використанні всіх 4 каналів комбінованої текстури. Крім

оптимізації пам'яті такий підхід зберігання текстур дозволяє значно оптимізувати фрагментний шейдер.

Приклад неоптимізованого шейдера:

```
fixed reflections = tex2D(_reflections, i.uv).r;
fixed colorMask = tex2D(_colorMask, i.uv).r;
fixed ambientOcclusion = tex2D( ao, i.uv).r;
```

У цьому випадку виконується читання даних із трьох різних текстур, що є витратною операцією при виконанні шейдерної програми, оскільки відбувається звернення до зовнішньої відеопам'яті.

При використанні комбінованої текстури читання з пам'яті виконується один раз:

```
fixed4 combine = tex2D(_MainTex, i.uv);
fixed reflections = col.r;
fixed colorMask = col.g;
fixed ambientOcclusion = col.b;
```

Такий підхід дозволяє значно збільшити швидкість роботи фрагментного шейдера. У таблиці 2.2 наведено дані дослідження швидкості роботи шейдерів з читанням з роздільних та комбінованих текстур на різних МП для 100 об'єктів у сцені.

Таблиця 2.2 – Дані тестування оптимізації комбінованих текстур на швидкість роботи фрагментного шейдера

Тип текстур	Мобільний пристрій	Швидкість відмальовування сцени (FPS)
Роздільні текстури	Samsung Galaxy A7	50
Комбінована текстура	Samsung Galaxy A7	41
Роздільні текстури	iPad Mini 2	35
Комбінована текстура	iPad Mini 2	33
Роздільні текстури	Sony Xperia E1 Dual	70
Комбінована текстура	Sony Xperia E1 Dual	45

2.1.3 Застосування векторної графіки як альтернатива текстурам

Традиційний підхід при відображенні 2D сцен та об'єктів UI полягає у відтворенні об'єктів за допомогою накладання текстур на прямокутні моделі (див. мал. 2.5). Однак такий підхід потребує значних витрат оперативної пам'яті для зберігання текстур та погано підходить для проектів із великою кількістю 2D графіки.

Рисунок 2.5 - Текстура накладена на прямокутну модель, що складається з 2х трикутників

Оптимізація за допомогою векторної графіки дозволяє значно скоротити обсяг текстур у додатку.

Векторна графіка - це спосіб представлення об'єктів та зображень у комп'ютерній графіці, заснований на математичному описі елементарних геометричних об'єктів, які зазвичай називають примітивами, таких як: точки, лінії, сплайни, криві Безье, кола і кола, багатокутники. Об'єкти векторної графіки є графічними зображеннями математичних об'єктів.

Оптимізація із застосуванням векторної графіки полягає в наступному: вихідні ресурси подаються у вигляді векторної графіки: SVG, CDR, CGM, DFX тощо. Наступний етап - це конвертація векторного об'єкта в 3D модель. Сконвертована 3D модель включається додаток як ресурс і використовується для малювання замість текстур. Дані про колір зберігаються у вершинах моделі або спеціально генерованих атласах градієнта. На рис. 2.6 показаний результат конвертації SVG об'єкта в 3D модель з різним рівнем деталізації.

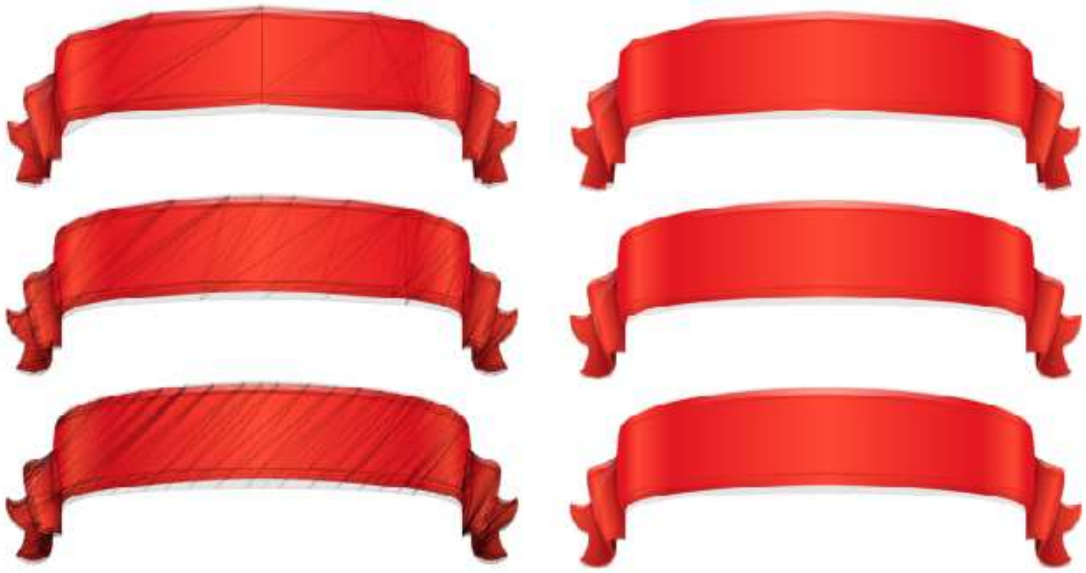

Рисунок 2.6 - Згенерована 3D модель за векторним об'єктом

Цей МО підходить для оптимізації графічного інтерфейсу користувача та 2D об'єктів. Залежно від ступеня деталізації згенерованих моделей обсяг ресурсів можна зменшити на 80% - 90%.

У таблиці 2.3 представлені результати аналізу параметрів способу для різних пристроїв.

Таблиця 2.3 – Дані тестування застосування векторної графіки

Пристрій	Використання традиційних текстур		Використання векторної графіки без напівпрозорих об'єктів		Використання векторної графіки з напівпрозорими об'єктами	
	Об `єм ресурсів	Швидкість відмальовування сцени (ms)	Об `єм ресурсів	Швидкість відмальовування сцени (ms)	Об `єм ресурсів	Швидкість відмальовування сцени (ms)
Samsung Galaxy A7	5.4 Mb	35	890 Kb	37	890 Kb	38
iPad Mini 2	4.1 Mb	33	890 Kb	33	890 Kb	35
Sony Xperia E1 Dual	5.4 Mb	41	890 Kb	41	890 Kb	43
Samsung Galaxy S2	5.4 Mb	38	890 Kb	40	890 Kb	40
ASUS Google Nexus 7	5.4 Mb	35	890 Kb	34	890 Kb	35
Apple iPhone 6 retina	4.1 Mb	33	890 Kb	33	890 Kb	34
Samsung S5	5.4 Mb	37	890 Kb	38	890 Kb	38

Використання векторної дозволяє ефективно економити оперативну пам'ять, однак слід зазначити, що такий підхід збільшує навантаження на GPU внаслідок збільшення кількості вершин, які необхідно малювати. Також слід зазначити, що у зв'язку з тим, що TBDR модель рендерінга не дозволяє ефективно малювати напівпрозору геометрію навантаження на GPU значно вище, ніж при відмальовуванні сцени з використанням текстур.

2.2 МО шейдерних програм та навантаження GPU

Більшість обчислень у шейдерах виконується на числах із плаваючою комою. Існує кілька варіантів типів з плаваючою комою: Highp, mediump та lowp (у мові CG: float, half та fixed). Ці типи відрізняються точністю і відтак продуктивністю та споживанням енергії на операції з ними.

– Highp – найвища точність. У більшості сучасних GPU – 32 біти. Зазвичай використовується при обчисленнях позицій у просторі та UV-координат;

– Mediump – середня точність. Найчастіше 16 біт. Застосовується здебільшого, якщо не потрібна підвищена точність операцій;

– Lowp - нижча точність, нижча за medium, але достатня для представлення всіх кольорів. У більшості GPU – 11 біт, з діапазоном від -2.0 до +2.0 та точністю 1/256. Застосовується до роботи з текстурними даними.

У сучасних GPU для ПК всі обчислення виконуються у високій точності (32 bit), проте застосування зниженої точності на МП є найпростішою оптимізацією.

Слід зазначити, що реальна точність у бітах може відрізнятися на різних пристроях. У таблиці 2.4 представлений огляд того, як різні лінійки GPU обробляють різні типи з плаваючою комою (у бітах).

Таблиця 2.4 – Точність типів з плаваючою комою у різних лінійках GPU

Лінійка GPU	Highp	Mediump	Lowp
PowerVR Series 6/7	32	16	
PowerVR SGX 5xx	32	16	11
Qualcomm Adreno 4xx/3xx	32	16	
Qualcomm Adreno	32 вершинний шейдер, 16 фрагментний		
ARM Mali T6xx/7xx	32	16	
ARM Mali 400/450	32 вершинний шейдер, 16 фрагментний		
NVIDIA X1	32	16	
NVIDIA K1	32		
NVIDIA Tegra 3/4	32	16	

Зміна точності обчислень дозволяє суттєво знизити навантаження на GPU і знизити обсяг споживаної відеопам'яті.

Приклад оптимізованого фрагментного шейдера наведено в лістингу 2.1.

Лістинг 2.1 - Оптимізований фрагментний шейдер

```
fixed4 frag (v2f i) : SV Target
{
half2 st = i.uv.xy/ Size.xy;
st.x *= Size.y/ Size.x;
half2 pos = st * 3.;
half DF = 0.0;
half a = 0.0;
half2 vel = half2(Time.y* Speed.x, Time.y* Speed.y);
DF += ridgedMF(pos+vel)* Size.z+ GradientSize.z;
DF += ridgedMF(pos+vel*2)* Size.z+ GradientSize.z;
a =snoise(pos*half2(cos(Time.y*0.15),sin(Time.y*0.1))*0.1)*3.1415;
vel = half2(cos(a),sin(a));
half q = smoothstep(GradientSize.x, GradientSize.y,frac(DF));
fixed alpha = clamp(0,1, (q+i.color.r))*i.color.g* wc.a;
fixed3 finalWaves = wc;
return fixed4(finalWaves*alpha+ mc*(1-alpha),alpha+ mc.a);
}
```

Повний вихідний текст оптимізованого шейдера міститься у додатку А.

Таблиця 2.5 – Дані тестування оптимізації шейдера за допомогою кваліфікаторів точності

Пристрій	Швидкість малювання сцени (ms)		
	Highp	Mediump	Lowp
Samsung Galaxy A7	45	35	35
iPad Mini 2	40	33	33
Sony Xperia E1 Dual	55	42	40
Sumsung galaxy S2	50	42	40
ASUS Google Nexus 7	41	34	30
Apple iPhone 0 retina	37	33	33
Samsung S5	40	33	33

mip-текстурування - метод накладання текстур, в якому використовуються кілька копій однієї текстури з різною роздільною здатністю.

Самою кращою якість зображення є тоді, коли роздільна здатність текстури наближається до роздільної здатності екрана. Якщо роздільна здатність екрана висока (текстура дуже маленька/об'єкт дуже близький), зображення буде розмитим. Якщо ж роздільна здатність текстури надто висока, будуть випадкові пікселі, що призведе до втрати дрібних деталей, муару і блимання. Зайва деталізація погана і за продуктивністю - надмірно навантажує відеошину і дає великий відсоток кеш-промахів. З цього випливає те, що краще мати кілька текстур різної роздільної здатності і накладати на об'єкт ту, яка найбільше підходить в даній ситуації. В окремих випадках приріст швидкості текстурування може збільшуватися багаторазово.

Багато сучасного ПЗ дозволяє користувачеві вручну встановити якість текстурування, в налаштуваннях, зробивши вибір між продуктивністю та якістю. У разі встановлення «низької» якості текстур, замість оригінальних текстур ПЗ використовуватиме їх зменшені копії.

Рисунок 2.7 – Приклад mip-текстури

Формується набір текстур із роздільною здатністю від максимального до 1x1. Наприклад: 1x1, 2x2, 4x4, 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512 та 1024x1024. Кожна з цих текстур називається MIP-рівнем чи рівнем деталізації.

На кожній з цих текстур знаходиться одне й теж зображення. З цього випливає, що mip-текстурування тягне за собою зростання витрату пам'яті на третину.

Під час текстури необхідно обчислити відстань до об'єкта. Номер текстури можна отримати згідно з формулою (2.1):

$$miplevel = log_2\left(\frac{dist}{texelsize*resolution}\right) + mipbias \quad (2.1)$$

де *resolution* – роздільна здатність віртуальної камери (кількість пікселів, яка буде в об'єкті розміром в 1 од., розташованому в 1 од. від камери), *texelsize* - розмір текселя в одиницях сцени, *dist* — відстань до об'єкта в тих самих одиницях, *mipbias* — число, що дозволяє вибирати більш менш детальну текстуру, ніж дає формула.

Отримана цифра заокруглюється до цілого числа, і текстура з цим номером (0 - найдетальніша, кожна наступна - вдвічі менша) буде накладатися на об'єкт.

Варто навести недоліки та способи їх вирішення. Як вже було згадано, витрата власне відеопам'яті зростає на 1/3 (проте натомість стає більшою швидкість відображення об'єктів на екрані). MIP -текстурування не дозволяє вирішити проблему текстур, котрі є під гострим кутом до спостерігача (як варіант, траса в автосимуляторі). Для цих текстур роздільна здатність по одній осі сильно корелює з роздільною здатністю по іншій - і, для прикладу, по осі X зображення може бути явно розмито, в той час як по Y видно блимання, властиве підвищеній роздільній здатності текстури. Анізотропна фільтрація — метод текстурування, котрий застосовується, щоб вирішити цю власне проблему.

Існує проблема видимості точної межі між MIP -рівнями. Цю проблему можна вирішити за допомогою трилінійної фільтрації.

Також застосування mip -текстур разом з упаковкою текстур в атласи може спричинити змішування різних текстур в атласі на низьких рівнях mip

Тестування застосування mip-текстур. Застосування mip-текстур у сценах зі значним числом 3D об'єктів з різними розмірами дозволяє суттєво зменшити навантаження на GPU та покращити швидкість відмальовування сцени. У таблиці 2.6 наведено дані про результати тестування застосування mip-текстур для сцени з великим числом б'єктів.

Таблиця 2.6 - Дані тестування застосування mip -текстур

Пристрій	Швидкість малювання сцени (ms)	
	Без mip -текстур	З mip -текстурами
Samsung Galaxy A7	40	33
iPad Mini 2	56	33
Sony Xperia E1 Dual	45	39
Sumsung galaxy S2	46	36
ASUS Google Nexus 7	40	33
Apple iPhone 6 retina	47	33
Samsung S5	49	34

2.3 МО сцени

Кількість видимих на об'єкті деталей значно знижується при значному віддаленні від камери. Однак, при малюванні об'єкта все ще буде використовуватися та ж кількість трикутників, навіть не дивлячись на те, що частина їх візуально буде непомітно. Існує техніка оптимізації, яка дозволяє знижувати кількість трикутників, що відмальовуються, у міру видалення об'єкта від камери - LOD. Якщо весь об'єкт не знаходиться близько до камери, LOD знизить навантаження на обладнання та підвищить продуктивність малювання.

LOD – це прийом програмування тривимірної графіки, що полягає у створенні кількох варіантів одного об'єкта з різними ступенями деталізації, які перемикаються в залежності від видалення об'єкта від віртуальної камери (рис 2.8) [15].

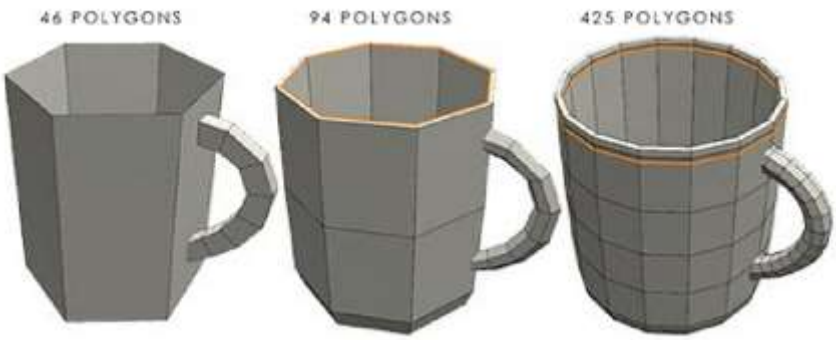

Рисунок 2.8 – Рівні деталізації об’єкта

Сенс методу полягає в тому, що відображати високодеталізовані об'єкти, що знаходяться на великій відстані від віртуальної камери, недоцільно з обчислювальних ресурсів. Використання LOD здатне суттєво знизити вимоги до ресурсів комп'ютера при виведенні графіки на екран, будь то рендеринг або виведення у реальному часі.

Таблиця 2.7 містить результати тестування застосування LOD об'єктів у сцені.

Таблиця 2.7 – Дані тестування застосування LOD об'єктів

Пристрій	Швидкість малювання сцени (ms)	
	Без застосування LOD	Із застосуванням LOD
Samsung Galaxy A7	40	33
iPad Mini 2	56	33
Sony Xperia E1 Dual	45	39
Sumsung galaxy S2	46	36
ASUS Google Nexus 7	40	33
Apple iPhone 6 retina	47	33

Метод композиції буфера глибини дозволяє досягти значного приросту продуктивності на мобільних платформах [14].

Суть методу полягає в об'єднанні пре-рендеренної статичної сцени з динамічними об'єктами в сцені. Статична сцена запікається у кілька текстур:

– текстура кольорів (RGB) - базова текстура, що містить основний колір сцени;

– текстура глибини (Grayscale) - необхідна для композиції динамічних об'єктів зі статичною сценою (рис. 2.9);

– текстура освітлення (RGB) – опціонально, застосовується для накладання освітлення на динамічні об'єкти.

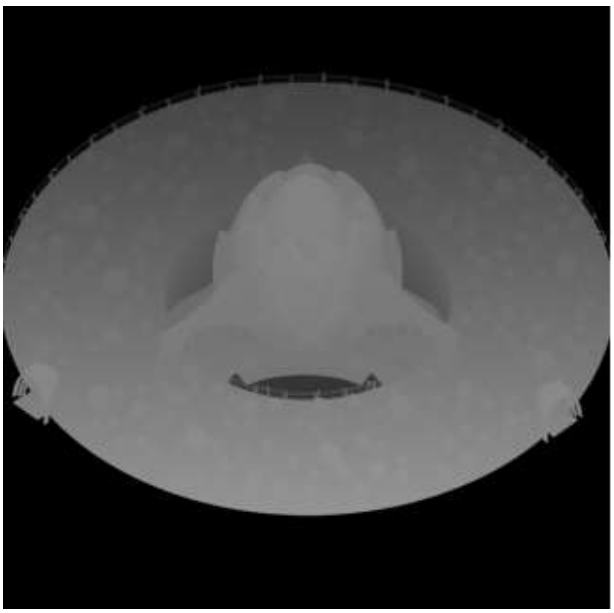

Рисунок 2.9 – Текстура глибини статичної сцени

Під час виконання програми статична сцена поєднується з динамічними об'єктами за допомогою шейдерів. На рис. 2.10 – 2.11 продемонстровано результати такого об'єднання.

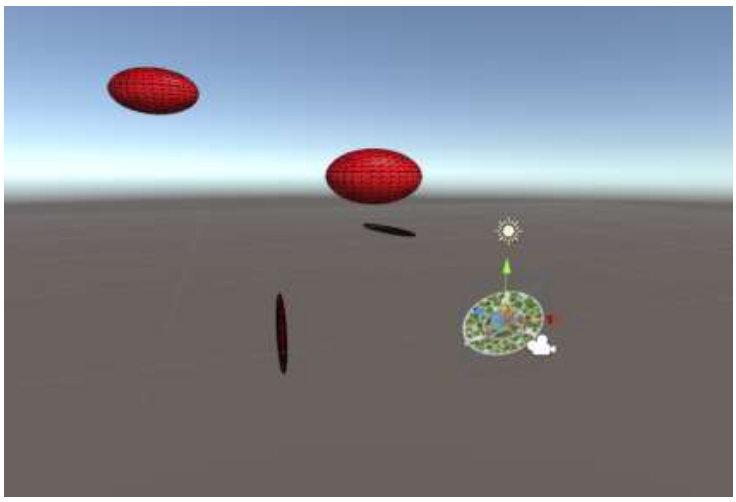

Рисунок 2.10 – Композиція статичної сцени з динамічними об'єктами (вид із редактора)

Рисунок 2.11 – Композиція статичної сцени із динамічними об'єктами (realtime)

Повний текст шейдера, що виконує змішування об'єктів у сцені, представлений у додатку Б.

Даний МО дозволяє створити ілюзію 3D-сцени без реального прорахунку великої кількості об'єктів, висвітлення, відображення тощо.

Однак такий підхід має суттєві недоліки:

– можна використовувати тільки у разі фіксованої камери та постійного кута огляду в сцені;

– працює тільки на пристроях із підтримкою OpenGL ES 3.0, оскільки у ранніх версіях OpenGL ES відсутня підтримка запису в буфер глибини із фрагментного шейдера.

Цей метод рекомендується застосовувати лише у випадку візуалізації складних 3D-сцен з невеликою кількістю динамічних об'єктів та фіксованою камерою

Текстурні атласи. Одним із МО КГ є застосування текстурних атласів. Цей метод більшою мірою застосовується для оптимізації швидкості відображення GUI [1].

Перемикання між безліччю окремих текстур негативно позначається на швидкості відображення кадру. У програмах нерідко застосовують множину дрібних текстур під час відображення GUI (кнопки, іконки, тощо). Тому в таких випадках буває доцільним використовувати одне велике зображення, а не замість множину маленьких.

Текстурний атлас — це зображення, що містить набір підзображень, кожне з яких є текстурою для 2D або 3D об'єкта. Для відображення підтекстури на об'єкті застосовується UV -перетворення, при цьому UV-координати в такому атласі задають, яку власне частину зображення треба використовувати.

Атласи можуть містити як підтекстури однакових розмірів, так і різних. Для складання атласів використовують s програми-генератори, і ручне складання.

На рис. 2.12 показаний приклад згенерованого текстур атласу для GUI комп'ютерної гри.

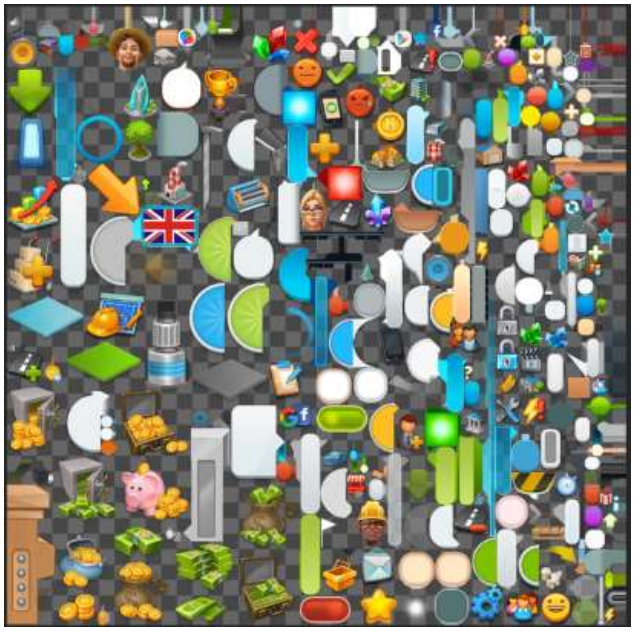

Рисунок 2.12 – Текстурний атлас

Переваги використання текстурних атласів [1,3]:

– дозволяє скоротити кількість змін станів до одного для всього атласу;

– зменшує кількість зайнятих текстурних слотів до одного для атласу;

– мінімізується фрагментація відеопам'яті;

– з'являється можливість використання NPOT "текстур, тобто атлас буде дотримуватися розмірності кратного ступеня 2 (256, 512, 1024 пікс.), А його елементи можна робити довільного розміру.

Однак, використання атласів може призводити до виникнення нових проблем:

– поєднання країв підтекстур стає нетривіальним завданням;

– стає складно або неможливо використовувати міпмапінг;

– при використанні фільтрації текстури необхідно додавати відступи, щоб сусідні підтекстури не змішувалася з потрібною;

– створення атласів вручну може бути трудомістким, тому потрібно використовувати спеціальні програми для генерування атласів;

– виникають невеликі втрати пам'яті, оскільки частина атласу може бути зайнята текстурами.

У таблиці 2.8 представлені дані оптимізації GUI із застосуванням текстурних атласів.

Таблиця 2.8 - Дані оптимізації GUI із застосуванням текстурних атласів

Пристрій	Швидкість малювання сцени (ms)	
	Без текстурних атласів	З текстурними атласами
Samsung Galaxy A7	48	33
iPad Mini 2	45	33
Sony Xperia E1 Dual	53	39
Sumsung galaxy S2	58	36
ASUS Google Nexus 7	38	33
Apple iPhone 6 retina	45	33
Samsung S5	42	36

У цілому нині застосування текстурних атласів для оптимізації GUI значно полегшує рендеринг кадру і застосовується у більшості програм КГ як у МП і ПК.

2.4 МО освітлення

Карти освітлення - це МО освітлення, в якому освітлення для статичних об'єктів у сцені запікається в текстури. Даний метод дозволяє позбавитися прорахунку динамічного освітлення і тіней для оточення в сцені.

Під час створення картка заповнюється чорними пікселями. Далі, для кожного текселя карти освітлення знаходяться тривимірні координати точки на полігоні. Для цієї точки необхідно побудувати список усіх джерел світла, які впливають на її освітлення: вектори з даної точки до джерел світла перевіряються на перетин з геометрією сцени, і якщо перетин має місце - то це джерело світла не висвітлює точку (щодо нього точка в тіні) . Інші джерела збільшують значення текселя карти освітлення на величину, яка залежить від моделі освітлення і положення джерела світла щодо точки. З метою покращення зовнішнього вигляду картинки, до карт освітлення часто застосовується білінійна фільтрація [10]. Ці операції повторюються для кожного освітлюваного полігону сцени. На рис. 2.13 показаний приклад запеченого освітлення сцени карти освітлення.

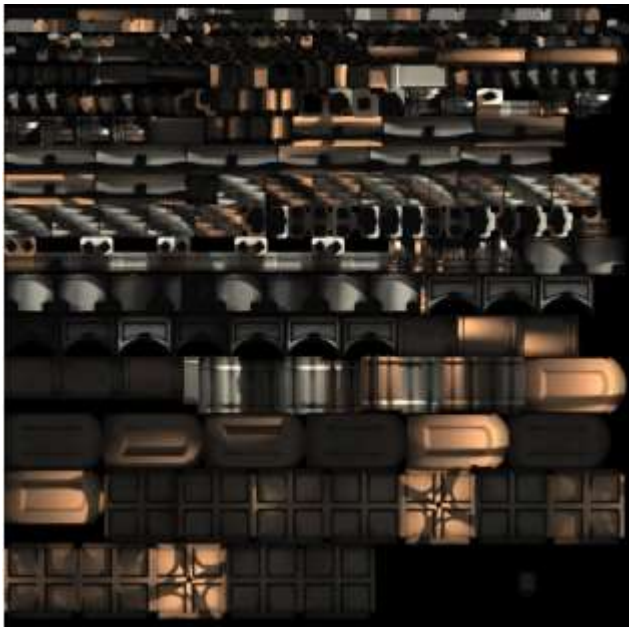

Рисунок 2.13 – Карта освітлення

Однак такий підхід має один суттєвий недолік – неможливість створити карти освітлення для об'єктів в динаміці у сцені. Застосування комбінованих джерел освітлення дозволяє частково оминути це обмеження. У такому підході створюються карти освітлення статичного оточення і виконується прорахунок освітлення для об'єктів в динаміці в реалі. У таблиці 2.9 представлені результати тестування повністю динамічного освітлення та використання карт освітлення для статичного освітлення.

Таблиця 2.9 – Дані тестування ефективності карт освітлення

Пристрій	Швидкість малювання сцени (ms)	
	Динамічне освітлення	Карти освітлення
Samsung Galaxy A7	75	35
iPad Mini 2	64	33
Sony Xperia E1 Dual	93	39
Sumsung galaxy S2	87	36
ASUS Google Nexus 7	60	33
Apple iPhone 6 retina	62	33
Samsung S5	73	36

У тому чи іншому вигляді карти освітлення використовуються практично у всіх сучасних 3D-додатках реального часу. Цей метод застосовується до створення всього статичного освітлення сцени. Освітлення генерується для статичної геометрії на початок циклу рендеринга, і під час рендерингу переважно змінюється. На сучасному обладнанні реалізація повністю динамічного освітлення з використанням карток освітлення неможлива через велику ресурсоємність процесу створення карток освітлення. Цей підхід сприймається як основа більшості інших алгоритмів рендерингу тіней у часі.

РОЗДІЛ 3. СТРАТЕГІЯ ОПТИМІЗАЦІЇ ГРАФІЧНИХ ПРОГРАМ НА МОБІЛЬНИХ ПРИСТРОЯХ

3.1 Алгоритм процесу оптимізації додатку

Процес оптимізації включає виявлення проблемних місць застосування і використання різних методів для усунення цих місць.

Процес оптимізації складається з кількох кроків:

– збір даних про продуктивність програми;

– аналіз цих даних та виявлення проблемних місць;

– визначення відповідних МО;

– вибір та застосування МО;

– повторний збір даних про продуктивність для перевірки результатів оптимізації.

Цей процес представлено на блок-схемі на рис. 3.1.

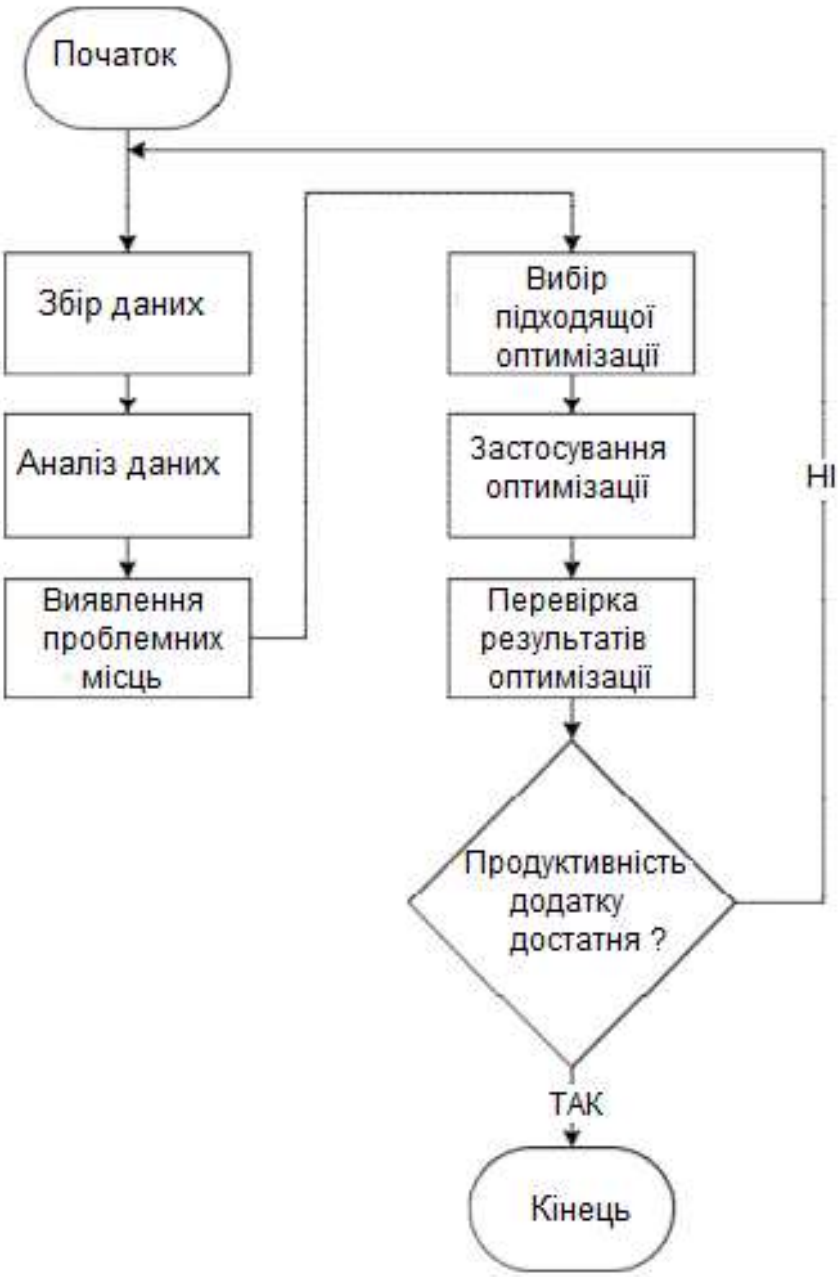

Рисунок 3.1 – Блок-схема процесу оптимізації додатку

Збір даних – це перший етап у процесі оптимізації. На цьому етапі застосовуються ПЩ – профайлери. При зборі даних необхідно виконувати такі правила:

– дані потрібно збирати лише з цільових пристроїв. Тільки справжнє апаратне забезпечення дозволить отримати точні дані щодо продуктивності програми.

– вертикальна синхронізація повинна бути вимкнена, так як ця процедура впливає на показники навантаження на GPU, і не дозволяє зібрати

точні дані щодо продуктивності програми.

Для визначення проблемних місць у програмі зібрані дані необхідно проаналізувати. Більшість профайлерів дозволяють уявити дані у вигляді графа або діаграм.

Вибір МО залежатиме від якості аналізу проблемних місць у програмі. Більшість із них пов'язана з такими областями:

- код програми;
- неправильне використання API;
- вершинні шейдери;
- фрагментні шейдери;
- пропускна здатність пам'яті;
- обсяг пам'яті;

Після застосування оптимізації необхідно виконати повторний збір даних про продуктивність програми та провести їх аналіз. Існує ймовірність того, що оптимізація може мати незначний ефект на продуктивність програми. Причини цього:

- існують інші проблемні місця у додатку, які обмежують продуктивність
- вимірювання та аналіз даних були проведені неправильно, тому були обрані неправильні способи оптимізації

Оптимізація програми – ітеративний процес. Для досягнення необхідних показників продуктивності необхідно повторювати кроки збору, аналізу даних, вибору методів та їх реалізації. Також слід зазначити що у процесі оптимізації вузьких місць у додатку можуть створюватися нові проблемні місця оскільки більшість МО мають як позитивні і негативні ефекти.

3.2 Загальні поради щодо оптимізації

Експерименти з різними підходами до оптимізації. Різні реалізації GPU мають різні ресурси та можуть використовувати різні версії драйверів. Ці відмінності впливають на продуктивність по-різному, тому важливо

експериментувати з різними підходами до графічного програмування та оптимізації для досягнення максимальної продуктивності.

Різні програми можуть реагувати на оптимізацію по-різному. В одному додатку конкретна оптимізація може мати більший вплив на продуктивність, ніж в іншому.

Не всі МО підвищують продуктивність програми. Графічний конвеєр складається з кількох компонентів та різних ресурсів, кожен з яких може бути проблемним місцем. Оптимізації, які не усувають проблемне місце, не впливають на продуктивність, доки вузькі місця не будуть усунені.

Часто існують кілька варіантів оптимізації проблемного місця, тестування та аналіз дозволяють вибрати метод, котрий найбільше підходить для вирішення даної проблеми.

Використання часу відображення кадру замість частоти кадрів. FPS - це найпростіший і базовий спосіб виміряти продуктивність програми, проте час малювання кадру (Frame time) краще підходить для вимірювання ефективності оптимізації [8].

Час малювання – це лінійна величина, а частота кадрів – нелінійна.

Графік на рис. 3.2 показує, як FPS залежить від швидкості відображення кадру.

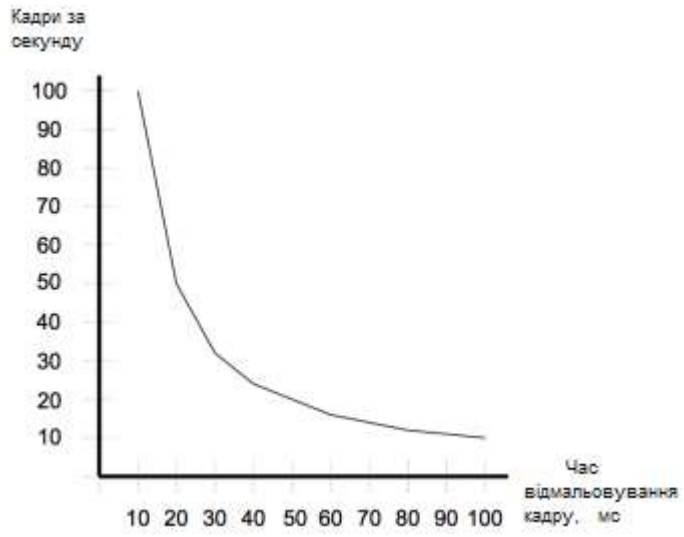

Рисунок 3.2 – Залежність кількості кадрів за секунду від швидкості відображення кадру

При використанні FPS неможливо прорахувати ефект від кількох МО, через нелінійність цієї величини. Однак це можна зробити за допомогою часу кадру.

У таблиці 3.1 наведено приклад зміни часу кадру на однакову величину та зміни FPS.

Таблиця 3.1 – Нелінійна зміна FPS при зміні часу кадру

Зміни FPS	Різниця у FPS	Різниця часу кадру
20 до 22.2	2.2	5 ms
50 до 66.6	16.6	5 ms
100 до 200	100	5 ms

3.3 Застосування стратегії оптимізації графічних програм

Запропонована стратегія оптимізації графічних програм була використана в процесі розробки комерційного проекту My Nano World [18]. Проект My Nano World це відеогра для МП під керуванням Android та iOS, у якій гравцеві необхідно розвивати власне місто.

У цьому проекті були використані такі МО:

– компресія текстур;
– текстурні атласи;
– mip-текстурування;
– зниження точності обчислень шейдерів;
– комбінування текстурних карт.

У таблиці 3.2 наведено результати процесу оптимізації проекту.

Таблиця 3.2 – Дані про результати процесу оптимізації проекту My Nano World

	До процесу оптимізації	Після процесу оптимізації
Об'єм займаних ресурсів на жорсткому диску	2.5 Гб	250 Мб
Обсяг займаних ресурсів в оперативній пам'яті	2.7 Гб	320 Мб
Швидкість малювання кадру в мс.	200 мс.	33 мс.

У процесі оптимізації проекту вдалося знизити обсяг ресурсів у 10 разів, а швидкість відображення кадру в 6 разів. Що доводить ефективність використаних МО та запропонованої стратегії оптимізації.

Таким чином, було проаналізовано ефективність різних МО графічних додатків на МП. На основі отриманих даних було сформовано стратегію оптимізації таких додатків та продемонстровано її ефективність.

РОЗДІЛ 4. БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

4.1 Санітарно-гігієнічні вимоги до умов праціз ПК

Санітарні правила і норми влаштування і обладнання кабінетів комп'ютерної техніки в навчальних закладах та режиму праці учнів на персональних комп'ютерах встановлюють нормативи фізичних чинників, що створюються комп'ютерами при їх роботі, та гігієнічні вимоги до проектування, виготовлення і експлуатації вітчизняних та експлуатації імпортних персональних комп'ютерів, що застосовуються в навчально-виховному процесі.

Вимоги до приміщень та розташування робочих місць з ПК: приміщення, призначені для роботи з ПК, повинні мати природне освітлення. Орієнтація вікон повинна бути на північ або північний схід, вікна повинні мати жалюзі, які можна регулювати, або штори; не дозволяється розміщувати кабінети обчислювальної техніки у підвальних приміщеннях будинків; кабінети, обладнані комп'ютерною технікою, в навчальних закладах повинні розміщуватись в окремих приміщеннях з природним освітленням та організованим обміном повітря; стіни, стеля і підлога та обладнання кабінетів комп'ютерної техніки повинні мати покриття із матеріалів з матовою фактурою з коефіцієнтом відбиття: стін — 40- 50 %, стелі — 70 - 80 %, підлоги — 20-30 %, предметів обладнання — 40-60 % (робочого столу — 40-50 %, корпуса дисплею та клавіатури — 30-50 %, стелажів — 40-60 %); поверхня підлоги повинна мати антистатичне покриття та бути зручною для вологого прибирання; забороняється використовувати для оздоблення інтер'єру приміщень комп'ютерних кабінетів полімерні матеріали (дерев'яно-стружкові плити, шпалери, що придатні для миття, плівкові та рулонні синтетичні матеріали, шаровий паперовий пластик та ін.), що виділяють у повітря шкідливі хімічні речовини, які перевищують гранично допустимі концентрації; вміст шкідливих хімічних речовин в повітрі дошкільних та учбових приміщень з комп'ютерною технікою не повинен перевищувати середньодобові концентрації [22].

Вимоги до освітлення приміщень та робочих місць: приміщення з ПК повинні мати природне та штучне освітлення; штучне освітлення в приміщеннях з ПК повинно здійснюватись системою загального освітлення; як джерела світла при такому освітленні повинні застосовуватись переважно люмінесцентні лампи; яскравість світильників загального освітлення в зоні кутів випромінювання від 50° до 90° з вертикаллю в поздовжній та поперечній площинах повинна складати не більше 200 кд/м^2, захисний кут світильників повинен бути не менше 40; загальне освітлення повинно бути виконано у вигляді суцільних або переривчастих ліній світильників; штучне освітлення повинно забезпечувати на робочих місцях в кабінетах з ПК освітленість не нижчу, а на екранах дисплеїв — не вище приведених в таблиці 4.1; коефіцієнт запасу для освітлювальних установок загального освітлення приймається рівним 1,4; необхідно проводити чищення скла вікон та світильників не менше двох разів на рік, а також заміну перегорілих ламп по мірі їх виходу з ладу; в кабінетах з ПК слід обмежити нерівномірність розподілу яскравості в полі зору учнів [22]. Співвідношення яскравості між робочим екраном та близьким оточенням не повинно перевищувати 5:1, між поверхнями робочого екрану і оточенням (стіл, обладнання) — 10:1; величина коефіцієнту пульсації освітленості не повинна перевищувати 5 %. Газорозрядні лампи повинні застосовуватись в світильниках загального та місцевого освітлення з високочастотними пускорегулюючими апаратами; необхідно передбачити обмеження прямого блиску від джерел природного та штучного освітлення; яскравість великих поверхонь (вікна, світильники і таке інше), що знаходяться у полі зору, не повинна перевищувати 200 кд/м^2, мірою захисту від прямого блиску має бути зниження яскравості видимої частини джерел світла застосуванням спеціальних розсіювачів, відбивачів та інших світлозахисних пристроїв, а також правильне розміщення робочих місць відносно джерел світла; повинні передбачатись заходи щодо обмеження відбитого блиску на робочих поверхнях (екран, стіл, клавіатура); яскравість полисків на екрані не повинні перевищувати 80 кд/кв. м. Яскравість стелі при застосуванні системи відбитого освітлення не повинна перевищувати 200 кд/кв. м.

Таблиця 4.1 — Норми освітленості в кабінетах з ПК

Характеристика роботи	Робоча поверхня	Площина	Освітленість,лк	Примітка
Робота переважно з екранами дисплеїв ПК (50 % та більше робочого часу)	Екран	вертикальна	200	не вище
	Клавіатура	горизонтальна	400	не нижче
	Стіл	горизонтальна	400	не нижче
Робота переважно з екранами дисплеїв ПК (менше 50 % робочого часу)	Екран	вертикальна	200	не вище
	Клавіатура	горизонтальна	400	не нижче
	Стіл	горизонтальна	500	не нижче
	Дошка	вертикальна	500	не нижче
Проходи основні	Підлога	горизонтальна	100	

Вимоги, що забезпечують захист від впливу іонізуючих та неіонізуючих електромагнітних полів та випромінювань: ВДТ на електронно-променевих трубках можуть бути потенційними джерелами гігієнічно значимих рівнів електромагнітних випромінювань в діапазоні частот 50Гц-300 МГц; інтенсивність ультрафіолетового випромінювання на відстані 0,3м від екрану не повинна перевищувати в діапазоні довжин хвиль 400 - 320 нм — 2 Вт/м^2, 320 - 280 нм — 0,002 Вт/м^2, ультрафіолетового випромінювання в діапазоні 280 - 200 нм — не повинно бути.

4.2 Вимоги до виробничого освітлення та його нормування

Приміщення для роботи з ВДТ повинні мати природне та штучне освітлення відповідно до ДБН В.2.5-28-2006 (на заміну СНиП II-4-79).

Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід і забезпечувати коефіцієнт природної освітленості не нижче ніж 1,5%. Розраховується він за методикою, викладеною в ДБН В.2.5-28-2006. За виробничої потреби дозволяється експлуатувати ЕОМ у приміщеннях без природного освітлення за узгодженням з органами державного нагляду за охороною праці та органами і установами

санітарно-епідеміологічної служби. Вікна приміщень з ВДТ повинні мати регулювальні пристрої для відкривання, а також жалюзі, штори, зовнішні козирки та ін.

Штучне освітлення приміщення з робочими місцями, обладнаними ВДТ ЕОМ загального та персонального користування, має бути обладнане системою загального рівномірного освітлення. У виробничих та адміністративно-громадських приміщеннях, де переважають роботи з документами, допускається вживати систему комбінованого освітлення (додатково до загального освітлення встановлюються світильники місцевого освітлення).

Загальне освітлення має бути виконане у вигляді суцільних або переривчатих ліній світильників, що розміщуються збоку від робочих місць (переважно зліва) паралельно лінії зору працівників. Допускається застосовувати світильники таких класів світлорозподілу: світильники прямого світла – П; переважно прямого світла – Н; переважно відбитого світла – В. При розташуванні ВДТ за периметром приміщення лінії світильників штучного освітлення повинні розміщуватися локально над робочими місцями. Для загального освітлення необхідно застосовувати світильники із розсіювачами та дзеркальними екранними сітками або віддзеркалювачами, укомплектовані високочастотними пускорегулювальними апаратами (ВЧ ПРА). Застосування світильників без розсіювачів та екранних сіток забороняється [23].

Як джерело світла при штучному освітленні повинні застосовуватися, як правило, люмінесцентні лампи типу ЛБ. При обладнанні відбивного освітлення у виробничих та адміністративно-громадських приміщеннях можуть застосовуватися металогалогенні лампи потужністю до 250 Вт. Допускається у світильниках місцевого освітлення застосовувати лампи розжарювання.

Яскравість світильників загального освітлення в зоні кутів випромінювання від 50º до 90º відносно вертикалі в подовжній і поперечній площинах повинна складати не більше 200 кд/м2, а захисний кут світильників повинен бути не більшим за 40º. Коефіцієнт запасу відповідно до ДБН В.2.5-28-2006 для освітлювальної установки загального освітлення слід приймати рівним 1,4. Коефіцієнт пульсації повинен не перевищувати 5 % і забезпечуватися

застосуванням газорозрядних ламп у світильниках загального і місцевого освітлення. За відсутності світильників з ВЧ ПРА лампи багатолампових світильників або розташовані поруч світильники загального освітлення необхідно підключати до різних фаз трифазної мережі.

Рівень освітленості на робочому столі в зоні розташування документів має бути в межах 300…500 лк. У разі неможливості забезпечити даний рівень освітленості системою загального освітлення допускається застосування світильників місцевого освітлення, але при цьому не повинно бути відблисків на поверхні екрану та збільшення освітленості екрану більше ніж до 300 лк. Світильники місцевого освітлення повинні мати напівпрозорий відбивач світла з захисним кутом не меншим за 40º. Необхідно передбачити обмеження прямої блискості від джерела природного та штучного освітлення, при цьому яскравість поверхонь, що світяться (вікна, джерела штучного світла) і перебувають у полі зору, повинна бути не більшою за 200 кд/м2. Необхідно обмежувати відбиту блискість шляхом правильного вибору типів світильників та розміщенням робочих місць відносно джерел природного та штучного освітлення. При цьому яскравість відблисків на екрані відеотерміналу не повинна перевищувати 40 кд/м2, яскравість стелі при застосуванні системи відбивного освітлення не повинна перевищувати 200 кд/м2 [22].

Необхідно обмежувати нерівномірність розподілу яскравості в полі зору осіб, що працюють з відеотерміналом, при цьому відношення значень яскравості робочих поверхонь не повинно перевищувати 3:1, а робочих поверхонь і навколишніх предметів (стіни, обладнання) – 5:1.

Необхідно використовувати систему вимикачів, що дозволяє регулювати інтенсивність штучного освітлення залежно від інтенсивності природного, а також дозволяє освітлювати тільки потрібні для роботи зони приміщення.

Для забезпечення нормованих значень освітлення в приміщеннях з відеотерміналами ЕОМ загального та персонального користування необхідно очищати віконне скло та світильники не рідше ніж 2 рази на рік, та своєчасно проводити заміну ламп, що перегоріли.

ВИСНОВКИ

В кваліфікаційній роботі запропоновано загальну стратегію оптимізації графічних програм для МП, з урахуванням існуючих технологій, методів та засобів оптимізації КГ.

Основні результати, отримані при виконанні кваліфікаційної роботи:

– проаналізовано методи та засоби оптимізації графічних додатків на МП;

– виконано класифікацію МО графічних додатків;

– проведено аналіз сучасних підходів до оптимізації графічних програм для МП;

– розглянута архітектура графічного конвеєра, що застосовується на мобільних GPU;

– досліджено різні МО КГ для МП, проведено оцінку ефективності цих методів на сучасних МП;

– виконано класифікацію МО;

– сформовано стратегію оптимізації та доведено її ефективність.

Статистичні дані, отримані в ході досліджень характеристик МО, дозволили скласти загальну стратегію оптимізації графічних додатків для МП.

Ефективність стратегії оптимізації була перевірена на реальному проекті МП.

ПЕРЕЛІК ДЖЕРЕЛ

1. Прокопчук О. П. Інформаційна технологія компоновки колекцій текстур у атласи зображень із компактифікацією // АПКН-2019, 14-15 листоп. 2019 р. – Хмельницький. – Т. 1. – С. 164–168.
2. Ginsburg D., Purnomo B., Shreiner D., Munshi A. OpenGL ES 3.0 Programming Guide 2nd, - 2014. - 322 c.
3. Pulli K., Aarnio T., Roimela K., Vaarala J. Designing graphics programming interfaces for mobile devices // IEEE Computer Graphics and Applications. - 2005. – p. 66 – 75.
4. Alex de Souza Campelo Lima, Edson Alves da Costa. Experimental Approach of Asymptotic Computational Complexity of Shaders for Mobile Devices with OpenGL ES // 2014 Brazilian Symposium on Computer Games and Digital Entertainment - 2014. – p. 173-182.
5. Chun-Fa Chang, Shyh-Haur Ger. Enhancing 3D Graphics on Mobile Devices by Image-Based Rendering // Advances in Multimedia Information Processing - PCM 2002. - p. 1105-1111.
6. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов. – М.:«ДМК Прес», 2015. – 368 с.
7. Rideout P. iPhone 3D Programming: Developing Graphical Applications with OpenGL ES, Publisher: O'Reilly Media, 2010. - 440 p.
8. OpenGL ES Application Optimization Guide, ARM, 2013. - 176 с.
9. A. Beers Rendering з Compressed Textures // Computer Graphics, Proc. 1996. – p. 373-378.
10. Jacson S. Unity3D UI Essential. - TR Press, 2015. - 280 p.
11. McDermott W. Creating 3D Game Art для iPhone з Unity: Featuring modo and Blender pipelines., - Focal Press, 2010. - 272 p.
12. Hughes J., Dam A., McGuire M., Computer Graphics: Principles and Practice (3rd Edition) 3rd Edition., Addison-Wesley Professional, 2013. - 1264 p.
13. Malizia A. Mobile 3D Graphics, Springer, 2006. – 162 p.
14. Guha S. Computer Graphics Через OpenGL: З Theory to Experiments,

Second Edition., - Focal Press, 2014, - 951 p.

15. Lengyel E. Mathematics for 3D Game Programming and Computer Graphics, Third Edition., - Cengage Learning PTR, 2011, - 576 p.

16. Marschner S. Fundamentals of Computer Graphics. - AK Peters/CRC Press, 2015 - 748 p.

17. Xcode Instruments [Електронний ресурс]. - Режим доступу: https://developer.apple. com/library/content/documentation/DeveloperT ools/Conceptual/InstrumentsUserGuide (Дата звернення 15.04.2022).

18. My Nano World [Електронний ресурс]. - Режим доступу: https://play.google.com/store/apps/details?id=com.nanorealitygames.mynanoworld&hl=uk (Дата звернення 15.04.2022).

19. OpenGL Registry [Електронний ресурс]. - Режим доступу: https://www.khronos.org/registry/OpenGL/index_gl.php (Дата звернення 16.04.2022).

20. Shadertoy [Електронний ресурс]. - Режим доступу: https://www. shadertoy.com (Дата звернення 16.04.2022).

21. OpenGL for MacOS [Електронний ресурс]. - Режим доступу: https://developer.apple.com/opengl/ (Дата звернення 09.05.2022).

22. Яремко З. М. Безпека життєдіяльності: Навч. посіб. — Львів., 2005. – 301 с.

23. Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями. [Електронний ресурс] - Режим доступу: https://zakon.rada.gov.ua/laws/show/z0508-18 – (дата звертання: 01.06.2021).

ДОДАТКИ

ДОДАТОК А
Фрагмент програмного коду шейдера зі зниженою точністю обчислень та генерацією процедурною текстури

```
Shader "Unlit/River"
{
Properties
{
     _mc("River Main Color", COLOR) = (1,1,1,1)
     _wc("Waves Color", COLOR) = (1,1,1,1)
     _Size("xy - size of the waves, z - waves count, w - waves
     speed", VECTOR) = (1,1,0,0)
     _GradientSize("Gradient size (xy) FractalSize(zw)", VECTOR)=
     (0.1,0.9,0,0) _Speed("River speed (xy)", VECTOR) = (1,1,0,0)
     [Toggle] _Invert("Invert color mesh color?", Float) = 0
}
SubShader
{
Tags { "RenderType"-'Transparent" "Queue"-'Transparent"}
LOD 100
Blend SrcAlpha OneMinusSrcAlpha
Pass
{
CGPROGRAM #pragma vertex vert #pragma fragment frag #pragma target
3.0 #include "UnityCG.cginc"
struct appdata {
float4 vertex: POSITION; float2 uv : TEXCOORD0; fixed4 color:
COLOR;
};
struct v2f {
float2 uv : TEXCOORD0; float4 vertex: SV_POSITION; fixed4 color:
COLOR;
};
fixed4 _wc; fixed4 _mc; half4 _Size; half2 _Speed; fixed _Invert;
fixed4 _GradientSize;
half3 mod289(half3 x)
{ return x - floor(x * (1.0 / 289.0)) * 289.0; }
 half2 mod289(half2 x)
{ return x - floor(x * (1.0 / 289.0)) *289.0; }
 half3 permute(half3 x)
{ return mod289(((x*34.0)+1.0)*x); |}
half snoise(half2 v)
const half4 C = half4(
0.211324865405187, // (3.0-sqrt(3.0))/6.0
0.366025403784439, // 0.5*(sqrt(3.0)-1.0)
-0.577350269189626 ,/0.0//.
// First corner
half2 i = floor(v + dot(v, C.yy)); half2 x0 = v - i + dot (i, C.
XX);
int2 i1 = (x0.x > x0.y)? half2(1.0, 0.0): half2(0.0, 1.0); half4
x12 = x0.xyxy + C.xxzz; x12.xy -= i1;

……..
```

ДОДАТОК Б
Фрагмент програмного коду шейдера композиції буфера глибини статичної сцени

```
Shader "Custom/Depth From Texture" {
Properties {
_DepthMultipler("Depth Multipler", Float) = 0 _MainTex("Base (RGB),
Opacity (A)", 2D) = "white" _DepthTexture("Depth (R)", 2D) =
"white" _ViewDir("Camera Direction", Vector) = (0,0,1,0)
}
SubShader
{
Tags {
"Queue"-'Transparent"
"IgnoreProjector"-'True"
"RenderType"="Transparent"
}
LOD 200
Pass
{
ZWrite On
Blend SrcAlpha OneMinusSrcAlpha // use alpha blending
CGPROGRAM
#pragma vertex vert
#pragma fragment frag
#include "UnityCG.cginc"
uniform float4 _MainColor; uniform float _DepthMultipler; uniform
float4 _ViewDir;
sampler2D _MainTex; sampler2D _DepthTexture;
struct vertOut {
float4 pos: POSITION; float4 worldPos : TEXCOORD1; float2 uv :
TEXCOORD;
};
struct fragOut {
float4 color: COLOR; float depth : DEPTH;
};
vertOut vert(float4 pos : POSITION, float2 uv : TEXCOORD) {
vertOut OUT;
OUT.pos = mul(UNITY_MATRIX_MVP, pos); OUT.worldPos =
mul(_Object2World, pos);
OUT.uv = uv; return OUT;


……
```