НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»
Завідувач кафедри
_____ В.О. РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)
“___” червня 2021 р.

Дипломний проект

на здобуття ступеня бакалавра

зі спеціальності **123 «Комп'ютерна інженерія»**

на тему: Графічний рендерер на базі багатоплатформенного API Vulkan

Виконав : студент IV курсу, групи КВ-71
Денисенко Іван Віталійович
(підпис)

Керівник доц. к.т.н. Павловський В.І.
(підпис)

Консультант з нормоконтролю, доц.каф. СП і СКС, к.т.н. Клятченко Я.М.
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент_____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування та спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.О. РОМАНКЕВИЧ
(підпис)
«___» червня 2021 р.

ЗАВДАННЯ
на дипломний проект студента
Денисенко Івана Віталійовича

1. Тема проекту «Графічний рендерер на базі багатоплатформенного API Vulkan»,
керівник проекту доц. каф. СП і СКС, к.т.н. Павловський Володимир Ілліч,
затверджені наказом по університету від «_____»_____2021р. №_____

2. Термін подання студентом проекту _____

3. Вихідні дані до проекту: див. Технічне завдання.

4. Зміст пояснювальної записки:
- аналіз предметної області;
- аналіз графічних рендерерів та API роботи з графікою;
- проектування графічного рендерера на базі API Vulkan;
- реалізація графічного рендерера;
- тестування графічного рендерера;
- шляхи поліпшення розробки.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо):

- структурна схема програмного додатку. Схема структурна;
- блок-схема алгоритму функціонування додатку. Блок-схема;
- блок-схема алгоритму функціонування 3Д-рендерера. Блок-схема;
- структурна схема 3Д-моделі. Схема структурна.
- Презентація за темою роботи.

6. Консультанти розділів проекту[1]

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., к.т.н., доцент каф. СП і СКС		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів проекту	Примітка
1	Розроблення та узгодження технічного завдання	30.11.2020	
2	Аналіз існуючих рішень	10.01.2021	
3	Підготовка матеріалів першого розділу дипломного проекту	12.04.2021	
4	Підготовка матеріалів другого розділу дипломного проекту	20.04.2021	
5	Підготовка матеріалів третього розділу дипломного проекту	27.04.2021	
6	Підготовка матеріалів четвертого розділу дипломного проекту	10.05.2021	
7	Підготовка матеріалів п’ятого розділу дипломного проекту	14.05.2021	
8	Підготовка графічної частини дипломного проекту	18.05.2021	
9	Оформлення документації дипломного проекту	23.05.2021	
10	Попередній огляд матеріалів диплому	26.05.2021	

Студент _____ Іван ДЕНИСЕНКО
(підпис)

Керівник проекту _____ Володимир ПАВЛОВСЬКИЙ
(підпис)

[1] Консультантом не може бути зазначено керівника дипломного проекту.

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (60 с., 28 рис., 4 додатки).

ГРАФІЧНИЙ РЕНДЕРЕР НА БАЗІ БАГАТОПЛАТФОРМЕННОГО API VULKAN, ПРЯМИЙ РЕНДЕРІНГ, ВІДКЛАДЕНИЙ РЕНДЕРІНГ, ФІЗИЧНО КОРЕКТНИЙ РЕНДЕРІНГ, ДВОПРОМЕНЕВА ФУНКЦІЯ ВІДБИВНОЇ ЗДАТНОСТІ, VULKAN, OPENGL, DIRECT3D, C++, GLSL, GLTF, GLFW.

Об’єкт розробки – графічний рендерер на базі багатоплатформенного API Vulkan.

Мета розробки – графічний рендерер, що дозволяє відображати графічні сцени, що складаються з 3Д-об’єктів та карти оточення, та задавати джерело освітлення.

В ході розробки:

- сформульовані основні вимоги до графічного рендерера;
- розроблена інтерфейс графічного рендерера;
- розроблена структура графічного рендерера;
- розроблено програмний додаток для дослідження і демонстрації можливостей рендерера.

Основні характеристики рендерера:

- генерування зображення в реальному часі;
- можливість маніпуляції з графічною сценою;
- висока швидкодія завдяки використанню сучасного графічного API Vulkan.

В процесі розробки використані наступні технології: мова програмування C++, шейдерна мова програмування GLSL, прикладний програмний інтерфейс Vulkan, математична бібліотека GLM, віконний фреймворк GLFW, бібліотека для роботи з файлами tinygltf.

Поліпшення рендерера можливе за рахунок додавання нових функціональних можливостей графічного рендерера та програмного додатку.

ABSTRACT

Qualification work includes an explanation note (60 p., 28 pictures, 4 appendices).

GRAPHICS RENDERER BASED ON VULKAN CROSS-PLATFORM API, FORWARD RENDERER, DEFERRED RENDERER, PHYSICALLY BASED RENDERING, BIDIRECTIONAL REFLECTANCE DISTRIBUTION FUNCTION, VULKAN, OPENGL, DIRECT3D, C++, GLSL, GLTF, GLFW.

The object of development is graphics renderer based on Vulkan cross-platform API.

The purpose of development is graphics renderer that is capable of rendering graphics scene, which consists of 3D-objects and environment map, and set source of light.

During development:

- the basic requirements have been formulated;
- renderer architecture has been developed;
- structure of renderer has been developed;
- renderer capabilities demonstration application has been developed.

Main renderer characteristics:

- real time image rendering;
- scene manipulation through user input;
- high speed due to Vulkan graphics API.

During development such technologies have been used: C++ programming language, GLSL shading language, Vulkan application programming interface, GLM math library, GLFW window framework, tinygltf file library.

System improvement could be achieved by adding new functional capabilities to the renderer and application.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045480.002 ТЗ	Графічний рендерер на базі багатоплатформенного API Vulkan. Технічне завдання.	4		
	А4	ІАЛЦ.045480.003 ТП	Графічний рендерер на базі багатоплатформенного API Vulkan. Відомість технічного проекту.	2		
	А4	ІАЛЦ.045480.004 ПЗ	Графічний рендерер на базі багатоплатформенного API Vulkan. Пояснювальна записка.	60		

Змін	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045480.001 ОА		
Розробив		Денисенко І.В.			Графічний рендерер на базі багатоплатформенного API Vulkan Опис альбому	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Павловський В.І.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-71	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Романкевич В.О					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045480.005 Д1	Структурна схема	1		
			програмного додатку.			
			Схема структурна.			
	А4	ІАЛЦ.045480.006 Д2	Блок-схема алгоритму	1		
			функціонування			
			програмного додатку.			
			Блок-схема.			
	А4	ІАЛЦ.045480.007 Д3	Блок-схема алгоритму			
			функціонування			
			3Д-рендерера.			
			Блок-схема.			
	А4	ІАЛЦ.045480.008 Д4	Структурна схема	1		
			3Д-моделі.			
			Схема алгоритму			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: «Графічний рендерер на базі багатоплатформенного API Vulkan».

Галузь застосування: графічний рендерер відображення графічних сцен на базі багатоплатформенного API роботи з графікою Vulkan.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на дипломне проектування на здобуття першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою цього проекту є створення графічного рендереру, що має можливість відображення 2Д-, 3Д-об'єктів та мапи середовища, а також дозволяє маніпулювати графічною сценою та переміщувати камеру від першої особи.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Вимоги до програмного продукту, що розробляється

- використання методів відкладеного рендерінгу;
- генерація зображення в реальному часі;
- відображення об’єктів з використання фізично коректної моделі освітлення;
- зміна положення камери від першої особи;
- можливість проводити маніпуляції зі сценою;
- задавати статичне джерело освітлення;
- задавати розсіяне світло від оточення.

5.2 Вимоги до апаратного забезпечення

- процесор: Intel Core i5;
- оперативна пам’ять: 4 Гб;
- простір на диску: 512 мб.

5.3 Вимоги до програмного та апаратного забезпечення користувача

- операційна система Windows 10.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Вивчення літератури за тематикою проекту	15.11.2020
2.	Розроблення та узгодження технічного завдання	30.11.2020
3.	Аналіз існуючих рішень	10.01.2021
4.	Підготовка матеріалів першого розділу дипломного проекту	12.04.2021
5.	Підготовка матеріалів другого розділу дипломного проекту	20.04.2021
6.	Підготовка матеріалів третього розділу дипломного проекту	27.04.2021
7.	Підготовка матеріалів четвертого розділу дипломного проекту	10.05.2021
8.	Підготовка графічної частини дипломного проекту	18.05.2021
9.	Оформлення документації дипломного проекту	23.05.2021
10.	Попередній огляд матеріалів диплому на кафедрі	26.05.2021

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4	ІАЛЦ.045480.000	Завдання на дипломний проект	2	
2	А4	ІАЛЦ.045480.001 ОА	Опис альбому	2	
3	А1	ІАЛЦ.045480.002 ТЗ	Технічне завдання	1	
4	А1	ІАЛЦ.045480.003 ТП	Відомість технічного проекту	1	
5	А1	ІАЛЦ.045480.004 ПЗ	Пояснювальна записка	62	
6	А1	ІАЛЦ.045480.005 Д1	Структурна схема програмного додатку. Схема структурна	1	
7	А1	ІАЛЦ.045480.006 Д2	Блок-схема алгоритму функціонування додатку. Блок-схема	1	
8	А1	ІАЛЦ.045480.007 Д3	Блок-схема алгоритму функціонування 3Д-рендерера. Блок-схема	1	
9	А1	ІАЛЦ.045480.008 Д4	Структурна схема 3Д-моделі. Схема структурна	1	

				ІАЛЦ.045480.003 ТП		
	ПІБ	Підп.	Дата			
Розробн.	Денисенко І.В.			Відомість дипломного проекту	Лист	Листів
Керівн.	Павловський В.І.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. СПіСКС Гр. КВ-71	
Н/контр.	Клятченко Я.М.					
Зав.каф.	Романкевич В.О.					

Пояснювальна записка
до дипломного проекту

на тему: Графічний рендерер на базі багатоплатформенного API Vulkan

Київ – 2021 року

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

АПІ (API) – прикладний програмний інтерфейс.

КГ – комп'ютерна графіка.

Рендерінг (Rendering) – процес генерації зображення об’єкту за допомогою комп’ютерної програми.

Direct3D – прикладний програмний інтерфейс роботи з графікою розробки компанії Microsoft.

GLFW (Graphics Library Framework) – фреймворк для роботи з віконною системою.

GLM (OpenGL Mathematics) – математична бібліотека.

GLTF (Graphics Language Transmission Format) – формат файлів для зберігання графічних сцен.

GPU (Graphics Programming Unit) – графічний процесор.

GPGPU (General Purpose GPU) – графічний процесор загального призначення.

OpenGL – прикладний програмний інтерфейс роботи з графікою.

PBR (Physically Based Rendering) – фізично коректний рендерінг.

Vulkan – прикладний програмний інтерфейс роботи з графікою.

ВСТУП

З розвитком комп’ютерної техніки та її розповсюдженням все більш актуальною проблемою ставало представлення інформації в графічному вигляді. Особливу увагу людство приділяло та приділяє інтерактивним способам представленням інформації, де користувач має можливість проводити різні маніпуляції з відображуваними об’єктами. При чому актуальним завданням залишається збільшення реалістичності відтворюваного зображення.

Світова тенденція представлення інформації в графічному вигляді та забезпечення інтерактивності робить проблему створення засобів відображення графіки актуальною як ніколи. Такі розробки можуть використовуватись як і в розважальних цілях або навчальних, так і в будь-якій сфері людської діяльності вцілому.

Доцільним є створення додатку, що дозволяє демонструвати можливості та переваги графічного представлення інформації, але в той же час є зручним при навчанні інших людей комп’ютерній графіці (КГ).

1. АНАЛІЗ ГРАФІЧНИХ РЕНДЕРЕРІВ ТА ПРИКЛАДНИХ ПРОГРАМНИХ ІНТЕРФЕЙСІВ РОБОТИ З ГРАФІКОЮ

1.1. Аналіз предметної області

Головну роль в сприйнятті інформації людиною грає зір. Заме завдяки зору людина отримує більшість інформації з навколишнього світу. Тому для ефективного сприйняття інформації і постає завдання надавати інформацію в візуальному (графічному) вигляді.

На теперішній час розвиток комп'ютерів та їх розповсюдження лише збільшують потребу у візуальному відображенні інформації. І саме таке завдання і вирішує створена система. До того ж, така система повинна надавати можливість легкого налаштування користувачем.

Розробники програмних додатків вже досить давно реалізують можливість відображення інформації графічно у вигляді статичних картинок чи у вигляді анімованої послідовності. В обох випадках на екран користувача виводиться зображення. Це зображення може як змінюватись в часі, так і буди постійним. Зображення представляє собою масив пікселів – найдрібніших неподільних одиниць зображення.

Існує три принципових методи генерації зображення. Перший метод – зберігання зображення в растровому вигляді на жорсткому диску для подальшого відтворення його на екрані. Растровий спосіб представлення означає, що зображення являє собою масив пікселів з кінцевою роздільною здатністю, де найдрібніша неподільна частина зображення називається піксель. Саме в такому вигляді зображення відображається на екрані. Другий спосіб зберігання зображень полягає в представленні зображення у вигляді геометричних примітивів: точок, ліній, кривих, полігонів тощо. Цей тип представлення називається векторною графікою. Такий тип зображення може бути перетворений у растровий з нескінченною точністю, а тому знайшов своє використання в тих областях, де масштабування зображення є надзвичайно важливим, наприклад відображення тексту.

Два вище перелічених типи представлення зображень гарно підходять для представлення зображень 2Д-об’єктів, але взагалі не підходять для інтерактивного представлення зображень 3Д-об’єктів. 2Д-об’єкти відрізняються від 3Д-об’єктів тим, що не мають глибини. Але в останні десятиліття представлення 3Д-об’єктів стало надзвичайно важливим. Для відображення 3Д-об’єкту використовують наступний, третій, метод. 3Д-об’єкт зберігається у вигляді полігональної сітки – сукупності вершин, ребр та граней, що визначають форму багатогранного об’єкта. Кожній точці такої сітки відповідає колір об’єкту, який зберігається у спеціальному зображенні, яке називається текстурою. Використовуючи геометрію об’єкту та текстури, можна згенерувати 2Д-відображення такого об’єкту з будь-якого ракурсу, яке і виводиться користувачу. Таке представлення об’єктів дозволяє відображати об’єкти реального світу, що значно підвищує сприйняття користувачем.

Програмна підсистема, що дозволяє відображати як 2Д-, так і 3Д-об’єкти у вигляді зображення, називається графічним рендерером. Створення графічного рендереру і є предметом вивчення та дослідження в чинному дипломному проекті.

1.2. Аналіз відомих графічних рендерерів

Умовно всі рендерери можна поділити на дві категорії. До першої категорії відносяться ті, де зображення генерується в реальному часі, щоб забезпечити плавність картинки, а це приблизно 30 кадрів на секунду або більше, тобто генерація одного кадру займає приблизно 33 мс або менше. Такі рендерери називаються рендерерами реального часу. До другої категорії відносяться ті рендерери, де обмеження в часі не такі суттєві. Рендерери, що відносяться до другої категорії, можуть генерувати зображення стільки часу, скільки завгодно.

В створенні рендерерів першого типу багато зроблено людьми, причетними до ігрової індустрії як головного замовника рендерерів реального часу для забезпечення інтерактивності та плавності ігрового процесу.

В створенні рендерерів другої категорії багато було зроблено компаніями та людьми, що зайняті в кінематографії. Так існує певна закономірність, що генерація одного кадру фільму, де використовується комп'ютерна графіка, займає 15 хвилин.

Розглянемо графічні рендерери реального часу ігрових рушіїв World of Tanks та GTA 5.

1.2.1. Графічний рендерер ігрового рушія World of Tanks

World of Tanks – онлайн-гра від третього лиця жанру казуального танкового симулятору. Особливістю цієї гри є широка база користувачів і, відповідно, дуже широкий спектр комп'ютерний комплектуючих цих гравців. При цьому нижня границя необхідного апаратного забезпечення для комфортної гри досить низька. Тобто, цю гру можна запустити, умовно кажучи, на «калькуляторі».

Такі вимоги до розробників гри втілились і в реалізації рендерера. Рендерер ігрового рушія World of Tanks має два принципово різних налаштування: стандартний та покращений. Зупинимося на стандартному налаштуванні.

Стандартне налаштування World of Tanks реалізує методи прямого рендерінгу (forward rendering). Особливістю такого підходу є його простота як з точки зору реалізації програмістами графіки, так і навантаження на графічний процесор. Ця простота зникає, коли в сцені одночасно потрібно відображати багато джерел освітлення, проте перед розробниками даного рендереру такого завдання не стояло.

Прямий рендерінг дозволяє зменшити загальне навантаження на графічний процесор ціною якості цього зображення. Функціонує він

наступним чином. На вхід рендерер отримує всі об’єкти сцени, що повинні бути відображені. І один за одним відображає кожен з них. Це найпростіший варіант відображення, тому він і отримав назву прямого. Найпростіший та найшвидший. Але за умови, що в сцені не буде складного освітлення.

1.2.2. Графічний рендерер ігрового рушія GTA 5

GTA 5 – гра жанру пригодницький бойовик у відкритому світі. Ключовою особливістю цієї гри є високий ступінь реалізму. Цей реалізм полягає як і в графіці, що інколи складно відрізнити від гарної фотографії, так і в геймплеї. І для досягнення такого ступеня реалізму розробники стикнулись з проблемою освітлення в цій грі, адже в реальному світі одночасно людина може спостерігати сотні та навіть більше джерел освітлення, що досить складно реалізувати в комп’ютерній грі.

Розрахунок освітлення об’єкту є процесом трудомістким. Традиційний метод рендерінгу, так званий «прямий рендерінг» (forward rendering), погано справляється з великою кількістю джерел світла. Суть цього методу полягає в тому, що для розрахунків освітлення кожного об'єкту потрібна інформація про всі джерела світла, які освітлюють даний об’єкт. Але кожен об’єкт освітлюється різною кількістю джерел, що створює додаткові труднощі вирахування цієї множини для кожного об’єкту. Але після того, як об’єкт вже відрендерений в тимчасовий буфер, він може бути перекритим іншим об’єктом, що знаходиться ближче до камери. Таким чином, всі трудомісткі розрахунки освітлення стають марними. І якщо об’єктів в кадрі дуже багато, то вони завжди в тій чи іншій мірі будуть перекривати один одного. До того ж, кількість джерел освітлення завжди обмежена.

Для вирішення вище перелічених проблем було винайдено інший метод рендерінгу – відкладений рендерінг (deferred rendering). Суть цього методу полягає в тому, що розрахунок освітлення «відкладається» на самий останній момент, коли вже оброблена геометрія всіх об’єктів сцени. Для досягнення цієї мети вивід зображення відбувається в декілька буферів, де кожен буфер відповідає певній властивості об’єкту. Кількість та призначення цих буферів буде розглянуто пізніше.

Розглянемо приклад на рисунку 1. В даному випадку вивід зображення відбувається одночасно в 6 буферів (зліва направо): шорсткість (roughness), металевість (metalness), базовий колір (base color), нормаль поверхні (normal), світимість поверхні (emissive) та глибина об’єкту відносно камери (depth). Потім відбувається розрахунок освітлення, де всі буфери поєднуються і ми отримуємо результуюче зображення, як показано на рисунку 1.1 [1] знизу.

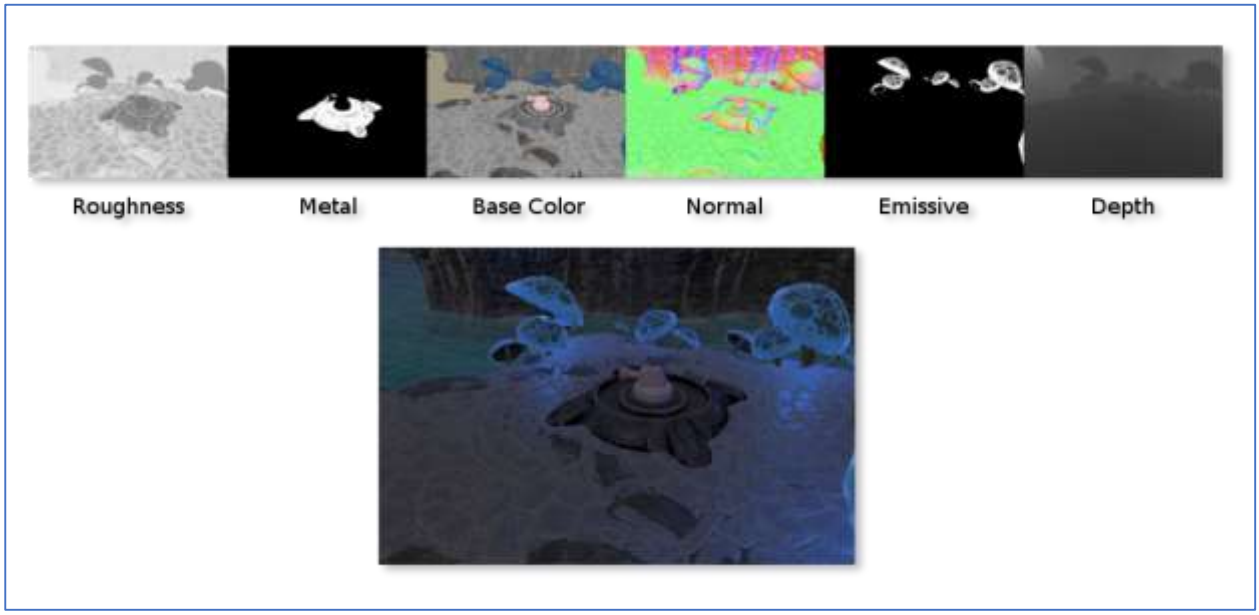

Рисунок 1.1 – Вивід зображення в декілька буферів

Даний метод дозволяє відображати будь-яку кількість джерел освітлень при відносно невеликих додаткових витратах. Але варто зазначити, що навантаження на підсистему пам’яті графічного процесору в такому випадку значно підвищується. Тому такий метод

освітлення не використовується в додатках, що працюють на мобільних пристроях, що пов’язане з тайловою архітектурою мобільних графічних процесорів (tile-based architecture).

Загалом відкладений рендерінг дає значні переваги програмістам графіки, бо фінальний рендерінг сцени (розрахунок освітлення) відкладається до останнього, а до цього йде процес «накопичення» даних про сцену. Далі ці дані можна використовувати, наприклад, для реалізації візуальних ефектів, таких як цвітіння (bloom).

Станом на зараз в середовищі розробників існує певний консенсус – майже всі ігри, що вийшли за останні роки, реалізували відкладений метод освітлення. Тому його можна вважати стандартом де-факто в ігровій індустрії.

1.3. Аналіз відомих графічних API

З самого початку розвитку комп’ютерної графіки як окремої гілки комп’ютерних наук людство хотіло створити візуальну симуляцію реального світу, тобто фізично коректне відображення віртуальних об’єктів на екрані. При вирішенні такої амбітної задачі людство почало стикатись з наступними проблемами: недостатній об’єм пам’яті та обчислювальних ресурсів сучасного на той момент апаратного забезпечення. Якщо першу проблему вдалось якщо і не вирішити, то сильно знизити її прояв, то друга проблема завжди була, є і буде каменем спотикання. Адже досягти ідеального фізично коректного відображення об’єктів неможливо – можливо лише наблизитись до нього з певною точністю. І чим краще наближення, тим більше апаратних можливостей потрібно для комфортної роботи з комп’ютерною графікою.

Все це спонукало до створення спеціалізованого апаратного забезпечення, яке могло б виконувати вузький спектр задач, але виконувати максимально ефективно. Так і з’явились графічні

процесори (GPU). Але одразу постає питання: як ці графічні процесори програмувати? І було знайдено наступний метод. Так як будь-яке стороннє апаратне забезпечення повинно мати свій драйвер, то прикладні програми будуть спілкуватись з графічним процесором не напряму, а з його драйвером, використовуючи спеціальний програмний інтерфейс, який і називається графічним API.

Таких графічних API було створено чимало. І всі вони дуже швидко розвивались та видозмінювались, адже комп'ютерні науки навіть сьогодні є відносно молодою галуззю знань, де багато проблем досі залишаються невирішеними. З тих, що дожили до сьогодні або були створені зовсім недавно, можна виділити наступні: OpenGL, Vulkan, Direct3D. Це, звичайно, не всі, але найбільш популярні.

1.3.1. OpenGL

OpenGL можна по праву вважати ветераном серед графічних API. Хоч перша версія і була випущена в далекому 1992 році, він і досі залишається на слуху. Станом на зараз він вже є неактуальним, але гарно підходить для тих, хто лише починає вивчати комп'ютерну графіку. OpenGL є відкритим платформонезалежним графічним API.

Особливістю OpenGL є його простота. Ця простота полягає в тому, що велика частина процесів прихована від програміста, він не має до неї доступу. Найбільш важливим серед них є синхронізація. При чому синхронізація як між центральним процесором (хостом) з графічним процесором (девайсом), так і між командами, що виконуються девайсом, між собою. Для запобігання «гони даних» (data race) реалізація OpenGL додає інколи додає непотрібні очікування між графічними операціями. Але часто так буває, що жодної залежності між операціями немає, і вони можуть бути виконані паралельно. І така «перестраховка» зменшую швидкодію. Таким чином, в загальному випадку реалізація OpenGL справляється з синхронізацією гірше, ніж

міг би справитись програміст, але, з іншого боку, це сильно полегшує роботу програмісту.

Часто так буває, що в системі присутній не один графічний процесор, а декілька, тому потрібно обрати один з-поміж них. Але в OpenGL немає можливості вибору графічного процесору серед наявних в системі.

Також варто зазначити як працює вивід зображення на екран. Очевидно, що для того, щоб вивести зображення на екран, воно повинно бути готовим, тобто згенерованим. Але вивід зображення на екран це дуже повільна операція. І якщо вивід зображення на екран не завершився, то це зображення перевикористовувати заборонено. Таким чином або графічний процесор чекає, доки вивід звершиться, щоб починати генерацію наступного кадру, або використовувати не одне, а два зображення – доки одне зображення виводиться на екран, а в інше здійснюється генерація наступного кадру. Але для того, щоб зображення на екрані завжди було найбільш актуальним і, при цьому, графічний процесор не простоював, бажано використовувати три зображення. Такий спосіб називається потрійною буферизацією (triple buffering). На жаль, OpenGL не дає можливості налаштовувати кількість зображень в буфері та політику оновлень, все на відкуп двайвера.

Також особливість програмного інтерфейсу OpenGL полягає в тому, що всі команди, які викликаються прикладною програмою, накопичуються у внутрішньому буфері реалізації OpenGL. При цьому це накопичення відбувається на одному логічному потоці, що робить неможливим використовувати переваги багатоядерних систем. Також програмісту невідомо, коли ці команди будуть виконані.

1.3.2. Vulkan

API Vulkan розроблялось тою самою організацією, що і OpenGL, але його не можна назвати логічним продовженням розвитку OpenGL. По своїй суті це зовсім інший графічний API, який не має вище перелічених недоліків OpenGL. Так само, як і OpenGL, Vulkan є відкритим платформонезалежним графічним API.

Vulkan надає набагато ширші можливості програмісту. Але з більшими можливостями приходить більша відповідальність. Коректно використовувати Vulkan складно, але виграші, які це дає, варті цього. Перш за все, це виграш в швидкодії. Завдяки кращому розумінню програмістом логіки своєї програми, він краще може налаштувати синхронізацію. Краще налаштована синхронізація – менше непотрібних очікувань.

Також змінилась логіка роботи з API. Новий підхід відкрив можливість використовувати багатоядерні системи, що дозволило вирішити таку проблему, як перевантаження графічного потоку (render thread) – потоку, на якому виконувались всі виклики графічного API. Vulkan дозволяє виконувати виклики функцій графічного API з будь-якого потоку, але проблема синхронізації цих викликів лягає на плечі програміста.

1.3.3. Direct3D

Direct3D є розробкою компанії Microsoft і розповсюджується лише на продукти цієї компанії. Тобто, цей графічний API є платформозалежним.

На даний час вийшло вже багато версій даного графічного API. Хоч вони й мали однакову назву, але інший індекс, проте були за своєю суттю часто зовсім іншими програмними інтерфейсами, які не мали взагалі нічого спільного з попередниками. Пропріетарність цього API дозволяє Microsoft оновлювати свою розробку і впроваджувати

новий функціонал, як їй самій захочеться, без попередньої згоди чи схваленнями іншими гравцями ринку. Така гнучкість, наприклад, дозволила Direct3D стати єдиним на той момент графічним API, що дозволяє використовувати технологію апаратного прискорення трасування промінів (ray tracing). В той же час, в розробці Vulkan задіяні багато компаній та організацій, що накладає певні обмеження на швидкості оновлення цього API. Direct3D таких обмежень не має.

1.4. Постановка задачі на розробку графічного рендерера

Стрімкий розвиток апаратних можливостей сучасних комп'ютерів зумовлює і стрімкий розвиток графічних підсистем додатків для візуального відображення контенту. Тому актуальним є створення рендерера, що відповідає вимогам сучасності, головною особливістю якого є генерація зображень в реальному часі з використанням сучасного графічного API та актуальних методів відображення об'єктів. Такий рендерер повинен відповідати наступним вимогам:

- використання методів відкладеного рендерінгу;
- генерація зображення в реальному часі;
- відображення об'єктів з використання фізично коректних моделей освітлення;
- зміна положення камери від першої особи;
- можливість проводити маніпуляції зі сценою;
- задавати статичне джерело освітлення;
- задавати розсіяне світло від оточення;
- використання сучасного графічного API.

2. ПРОЕКТУВАННЯ ГРАФІЧНОГО РЕНДЕРЕРА НА БАЗІ ГРАФІЧНОГО API VULKAN

2.1. Принцип функціонування графічного рендерера

Графічний рендерер є програмним компонентом будь-якого додатку, що відображає 3Д-об'єкти або генерує зображення. Але кінцевою задачею рендерера є запрограмувати спеціалізоване апаратне забезпечення – графічний процесор. Графічні процесори працюють за схожим принципом і має назву графічного пайплайну (graphics pipeline), який зображено на рисунку 2.1 [2].

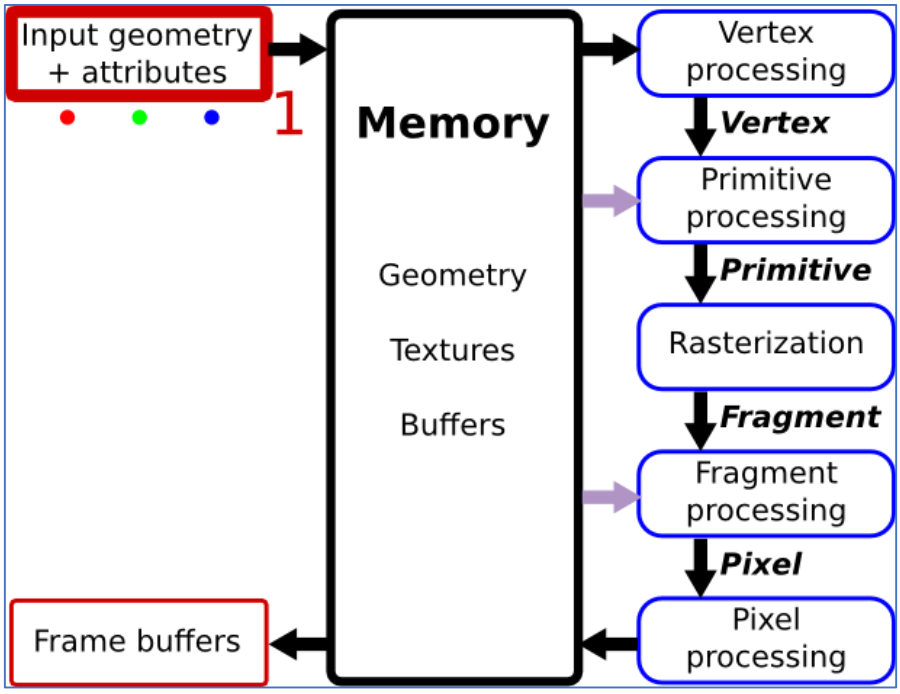

Рисунок 2.1 – Графічний пайплайн

2.1.1. Принцип функціонування графічного пайплайну

Розглянемо графічний пайплайн детальніше. Графічний пайплайн має декілька послідовних стадій: стадія вхідної геометрії (input geometry), обробка вершин (vertex processing), обробка примітивів (primitive processing), растеризація (rasterization), обробка фрагментів (fragment processing), обробка пікселів (pixel processing) та вивід в буфер кадру (framebuffer).

Стадія вхідної геометрії – початок графічного пайплайну. На цій стадії встановлюється формат вхідних даних вершин геометрії об’єкту. Далі ці вхідні дані використовуються для формування примітивів, які складаються з одної або декількох вершин. Кожна вершина має певні атрибути. Атрибут є інформація, що належить певній вершині. Кількість цих атрибутів може змінюватись. Також немає встановленого обов’язкового набору атрибутів. Кількість атрибутів залежить лише від складності об’єкту.

Далі ці вершини, кожна я яких несе певну інформацію, попадає на вершинний шейдер. Шейдер (shader) – програма, що виконується на графічному процесорі. Цю програму створюють програмісти графіки. Завданням вершинного шейдера є модифікація вершин потрібним чином. Наприклад, зміна положення вершини або зміна певних атрибутів за встановленим правилом. Сучасні графічні API дають багато свободи стосовно модифікації вершин. Саме в вершинному шейдері позиція цієї вершини змінює систему координат. Позиція кожної вершини, найчастіше, задається в локальній до об’єкту, до якого відноситься вершина, системі координат, де x, y та z задаються проміжком [-1, 1]. Саме вершинний шейдер спочатку переводить ці позиції в світову систему координат, а потім в систему координат камери. Проте застосувань вершинного шейдеру безліч.

Далі одна або більше вершин формують примітив. Такими примітивами можуть бути як точка, так і лінія або трикутник. Більш складні примітиви майже ніде не зустрічаються, бо будь-який складний примітив може бути представлений як набір трикутників. До того ж, трикутник лежить завжди лише в одній площині, що спрощує подальшу його обробку.

Формування примітивів необхідне для наступного кроку – встановлення відповідності кожного примітиву площі на зображенні та набору пікселів, які він буде займати на зображенні. Для цього кожна

вершина примітиву проектується на зображення. Після цього встановлюється набір пікселів, що перекриваються цим примітивом. Піксель вважається перекритим примітивом, якщо точка семплу (sample point) цього пікселя лежить всередині примітиву. Кількість точок семплів кожного пікселя може бути відмінною від нуля, але не меншою за нуль. Також сусідні пікселі можуть мати спільні точки семплу. Чим більше точок семплу, тим точніше можна встановити яка частина пікселю перекрита примітивом. Але є обов'язкова умова: в часі топологія точок семплу не повинна змінюватись. В іншому разі це призведе до появи видимих оком змін зображення. Вище описаний процес називається растеризацією (ratrerization). Приклад растеризації трикутника можна побачити на рисунку 2.2 [3], де використовується одна точка семплу.

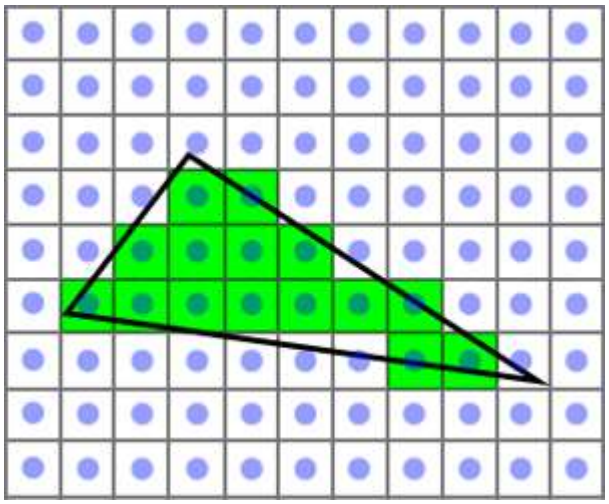

Рисунок 2.2 – Растеризація трикутника

Наступним кроком є обробка тих пікселів, що попадають на зображення, відбувається виклик фрагментного шейдеру. На цьому етапі кожному фрагменту присвоюється певна інформація. Ця інформація передається з вершинного шейдеру. При цьому відбувається інтерполяція цих даних між вершинами примітиву. Перед викликом фрагментного шейдеру може відбуватись певна кількість необов'язкових перевірок, як от перевірка глибини фрагменту. Якщо

фрагмент не проходить якусь з перевірок, він відкидається. Наприклад, потрібно відобразити певний об’єкт на тому місці, де вже був відображений інший об’єкт. В такому разі програміст графіки може ввімкнути тестування глибини фрагменту, і тоді відобразиться лише той фрагмент, який знаходиться ближче до камери.

Після обробки фрагментів формується піксель фінального зображення. Цей піксель може повністю замінити старе значення в тому самому місці зображення або певним чином комбінуватись з ним. Вивід пікселя відбувається в кадровий буфер.

Кожен об’єкт оброблюється на графічному процесорі саме в такому порядку. Жодна зі стадій не може бути пропущена.

2.1.2. Кадровий буфер

Вивід будь-якого зображення будь-якого рендерера відбувається в кадровий буфер. Кадровий буфер складається з цілей виводу (render target) – зображень, в які записуються значення пікселів, згенеровані графічним пайплайном при обробці об’єкту сцени.

Кількість цілей виводу відрізняється від рендерера до рендерера. Але вона не може бути менше одного. При цьому існують дві спеціалізовані кадрові цілі – буфер глибини (depth map) та шаблонний буфер (stencil buffer). Перший відповідає за зберігання глибини пікселя в системі координат камери. Інший відповідає за зберігання шаблонних значень. Перший буфер використовується для визначення близькості пікселя до камери. Другий може бути використаний як трафарет, наприклад, записувати пікселі в іншу кадрову ціль тільки в тому випадку, якщо відповідне значення пікселя шаблонного буфера дорівнює певному значенню.

Також можна використати кадрову ціль, в яку буде записано швидкість переміщення об’єкту. Потім ці значення використовуються для реалізації техніки розмиття в русі (Motion blur).

Іншим прикладом використання кадрової цілі може бути буфер нормалей поверхні. В такий буфер заноситься значення нормалі поверхні, потім це значення може бути використаним при розрахунках освітлення.

Генерація одного кадру може бути розбита на певні стадії. Наприклад, для реалізації тіней можна використати наступний підхід. Спочатку всі непрозорі об’єкти, що формують сцену, відображаються в спеціальну кадрову ціль. При чому в ній зберігається глибина об’єктів відносно джерела освітлення, як на рисунку 2.3. При цьому чим світліший піксель, тим об’єкт знаходиться далі.

Рисунок 2.3 – Буфер глибини графічної сцени

В наступній стадії відображаються всі об’єкти сцени. При цьому використовується буфер глибини, що був згенерований на попередньому кроці. Глибина об’єкту відносно камери переводиться в систему координат відносно джерела освітлення. Потім це значення порівнюється зі значенням відповідного пікселя буферу глибини. Якщо воно виявилось більше, то об’єкт знаходиться в тіні від світла.

2.1.3. Налаштування графічного пайплайну в залежності від типу об’єкту

Очевидно, що в графічній сцені потрібно відображати принципово різні типи об’єктів. Такими можуть бути: непрозорі об’єкти, небо, прозорі об’єкти, гази, аерозолі, візуальні ефекти та інші. В залежності від типу та складності об’єкту налаштування графічного пайплайну змінюється. Якщо, наприклад, для відображення непрозорих об’єктів вершини формують трикутники, то для відображення диму можливо формувати точкові примітиви, де кожен примітив буде формувати лише один фрагмент, і ці фрагменти будуть затіняти ті об’єкти, що вже були відображені, тобто колір диму буде змішуватись з кольором того об’єкту, що знаходиться за димом.

Варто також відмітити, що зміна графічного пайплайну, за яким працює графічний процесор, процес досить трудомісткий. Тому графічні рендерери можуть створити певну множину графічних пайплайнів для різних типів об’єктів. І відображати всі об’єкти сцени по групам, в залежності до якого типу об’єкту він належить. Таким чином, змін графічних пайплайнів на графічному процесорі буде якомога менше, що зменшить час на відображення сцени.

Але окрім зменшення кількості змін графічних пайпланів важливою є зменшення кількості змін ресурсів, що використовуються цими пайплайнами. Такими ресурсами є текстури та сталі буфери (uniform buffer).

2.1.4. Текстурування об’єктів

В загальному випадку текстура є зображенням з певними правилами вибору фрагмента з цього зображення. Але призначення текстур одне – надавати значення пікселя зображення за певними координатами. При цьому призначення цієї інформації може бути різноманітним: від зберігання кольору об’єкту до карти висот або

нормалі поверхні. Спільного в будь-яких текстур лише одне: в них зберігається значення, що відноситься до певної точки геометрії об’єкту. Використання текстур обмежується вершинними та фрагментними шейдерами, адже лише ці етапи графічного пайплайну є програмованими. Найпопулярнішим варіантом використання текстури є зберігання кольору об’єкту. Приклад такої текстури наведено на рисунку 2.4.

Рисунок 2.4 – Текстура кольору 3Д-моделі автомобіля

Так як більшість опуклих фігур не можна розгорнути в площину без розривів, але можна розбити фігуру на простіші, то фігури

розбиваються на більш прості і розгортаються на площину текстури. Таким чином, не все зображення текстури несе корисне навантаження, але ця проблема не має іншого вирішення.

Зміна текстур в графічному пайплайні також досить важка операція. Тому графічний рендерер може реалізовувати можливість відображати об’єкти, що мають однакові текстури, підряд.

2.1.5. Сталі буфери

Сталим буфером є буфер, значення даних якого не змінюється впродовж відображення об’єкту. Використовувати їх можливо лише в шейдерах. І, зазвичай, сталі буфери використовують для зберігання даних, що використовуються при різноманітних розрахунках.

Наприклад, сталі буфери можна використовувати для зберігання матриці зміни системи координат або трансформації об’єкта.

2.2. Вимоги до графічного рендерера

Розробка графічного рендерера переслідує дві наступні цілі:

- учбова;
- дослідницька.

Суть першої цілі є продемонструвати різні методи та техніки відображення об’єктів та різні підходи до розробки графічних рендерерів.

Суть другої цілі є дослідити ступінь реалізму відображених об’єктів та оцінити швидкодію різних методів відображення об’єктів та підходів до побудови графічних рендерерів.

Актуальність розробки в тому, що за допомогою такого рендереру можна навчати інших комп’ютерній графіці.

2.3. Вибір графічного API

Важливим моментом побудови графічного рендерера є вибір графічного API. Бурхливий розвиток комп’ютерної графіки в останні

роки призвів до появи нових та зміни вже існуючих графічних API. Навіть більше. Не стояли на місці і графічні процесори. Спеціалізація їх ставала все меншою з часом, що призвело до появи графічних процесорів загального призначення (general purpose GPU, GPGPU). Такі графічні процесори можуть виконувати не лише код графічних шейдерів, але й код обчислювальних шейдерів (compute shader).

Використання обчислювальних шейдерів знайшло широке використання. Так, наприклад, симуляція фізики фізичним рушієм PhysX відбувається на графічному процесорі, що дає значне підвищення швидкодії.

Також з кожним роком з'являються все нові графічні технології. В 2016 році компанія Nvidia зробила фурор – вона представила графічні процесори, що мали можливість апаратного прискорення трасування променів (ray tracing) – технології, що дозволяє кардинально змінити принципи відображення в графічних рендерерах реального часу. Також було представлено технологію DLSS. Вона дозволяє збільшувати роздільну здатність фінального зображення, використовуючи технології машинного навчання. Але виникла проблема: підтримка цих технологій була реалізована лише в одному з графічних API – Direct3D. Станом на зараз підтримка цих технологій також присутня і в API Vulkan. Інші графічні API такої підтримки не отримали і, скоріше за все, уже не отримають.

Тому до вибору графічного API варто підійти з розумом. Неправильний вибір може коштувати дуже дорого. Критеріями вибору графічного API є:

- доступність на різних платформах;
- підтримка сучасних технологій;
- гнучкість у використанні;
- швидкодія;
- актуальність в майбутньому.

В залежності від вимог, що ставляться до графічного рендерера, вибір може відрізнятись. Найбільш підходящим під висунуті вимоги є графічний API Vulkan, адже цей графічний API відносно новий, сучасний, підтримує актуальні технології та бурхливо розвивається. Також особливістю його є підтримка різними платформами: Windows, Linux, Android. Але що більш важливо – він низькорівневий, кажуть, що він «наближений до заліза». А саме такий графічний API найкраще підходить для учбових цілей – зрозуміти як працюють графічні процесори.

2.4. Проектування інтерфейсу графічного рендерера

Інтерфейс взаємодії є свого роду контрактом між рендерером та тим програмним компонентом, що використовує можливості рендерера. Цей контракт встановлює результат перетворень вхідних даних рендерером.

Рендерер надає інтерфейс для наступних дій:

- створення графічних примітивів;
- відображення сцени з використанням графічних примітивів.

Графічними примітивами є наступні типи даних:

- текстура;
- семплер;
- матеріал;
- геометричний примітив;
- буфер вершин;
- буфер індексів вершин;
- напрямлене світло;
- розсіяне світло;
- матриця трансформації примітиву.

Умовно кажучи, кожен графічний примітив може бути створений рендерером і кожен графічний примітив може бути задіяний при відображенні сцени.

Графічні примітиви можуть бути використані для відображення наступних об’єктів:

- 2Д-об’єкт;
- 3Д-об’єкт;
- карта середовища.

2.5. Проектування структурних елементів графічного рендерера

Структура графічного рендерера зображена на рисунку 2.5.

Рисунок 2.5 – Структура графічного рендерера

Структурно графічний рендерер складається з трьох компонентів: 3Д-рендерера, графічного контролера та графічного контексту.

Інтерфейс графічного рендерера реалізовано в 3Д-рендерері, а особливості реалізації реалізовані в двох інших компонентах, де кожен має своє призначення.

Стрілка, що виходить з компоненту, показує які структурні елементи він використовує. Дана структура побудована у вигляді певної трирівневої послідовності без розгалужень. При реалізації цих компонентів розгалуження можливі, але кількість рівнів змінена бути не може. Така структура представляє свого роду контракт. При цьому, якщо буде декілька реалізацій будь-якого з компонентів, то той рівень, що використовує даний, може використовувати будь-яку з реалізацій. Тобто, реалізація кожного з рівнів може бути різноманітною, але інтерфейси та сервіс, що вони надають, залишається незмінним.

2.5.1. 3Д-рендерер

В компоненті 3Д-рендерер інкапсульовано всі високорівневі особливості реалізації графічного рендерера.

Саме цей компонент і реалізує вимоги, що поставлені до графічного рендерера. При цьому може бути створений інший рендерер, що реалізує зовсім інші вимоги, при відносно невеликих витратах, адже яким би рендерер не був, він в будь-якому разі використовує той сервіс, що надає графічний контролер.

В цьому і є особливість цієї структури – перевикористання компонентів. Кожен компонент, що знаходиться «нижче», надає власний інтерфейс для взаємодії. Кожен з них, при цьому, є більш низькорівневим, ніж той, що знаходиться вище. Додавання реалізації будь-якого з рівнів лише доповнює структуру, робить її більше гнучкою.

Вимоги до 3Д-рендерера наступні:

- робота в реальному часі;
- реалізація методів відкладеного освітлення;
- створення графічних примітивів;
- використання графічних примітивів для відображення сцени;
- використання фізично коректних методів відображення поверхонь;
- реалізація камери від першої особи.

2.5.2. Графічний контролер

Наступним рівнем є графічний контролер. Цей рівень призначений для більше детального представлення графічних примітивів. Адже поняття 3Д-сцени досить високорівневе, а графічні API оперують більш низькорівневими поняттями. Саме цей рівень інкапсулює ті структури та типи даних, які використовуються кожним з графічних API. Умовно кажучи, кожному з графічних API відповідає свій графічний контролер. Але кожна з реалізацій графічного контролера має однакову архітектуру та надає однаковий сервіс вище розташованому рівню.

Така структура дозволяє додавати підтримку різних графічних API при відносно невеликих витратах – достатньо додати ще одну реалізацію графічного контролера.

2.5.3. Графічний контекст

Графічний контекст є інкапсуляцією графічного процесору та екрану виводу.

Так як в системі може бути декілька графічних процесорів, то саме графічний контекст відповідальний за навантаження потрібного графічного процесору. Також цей рівень відповідальний за відображення результатів роботи графічного процесору на екрані комп’ютера.

Графічний контекст напряму «спілкується» з драйвером відеокарти, яка перетворює виклики Vulkan в пропрієтарні.

2.6. Вибір фізично коректної моделі відображення поверхонь

Дуже важливим аспектом при проектуванні графічного рендерера є вибір фізично коректної моделі відображення поверхонь. Взагалі не будь-яка модель відображення поверхонь є фізично коректною. Фізично коректною вважається така, на яку накладаються певні вимоги, наприклад, закон збереження енергії, тобто, кількість відбитого світла не може бути більшою за кількість падаючого світла, якщо не брати до уваги такий ефект як флюоресценція.

Ідеального фізично коректного відображення об’єктів досягти неможливо. Можливо лише з певною точністю наблизитися до такого результату. І, зазвичай, прийнятний результат навіть не потребує дуже складних моделей.

Існує цілий клас фізично коректних моделей відображення об’єктів, заснованих на теорії мікроповерхонь, яка розглядається нижче.

2.6.1. Теорія мікроповерхонь

З точки зору оптики, поверхнею будь-якого об’єкту є двовимірна площина, що розділяє два середовища, що мають різні індекси заломлення. При цьому поняття поверхні на макроскопічному рівні втрачається – ми не можемо сказати, що в молекули чи атома є поверхня, але також на такому рівні ми змушені розглядати світло як хвилю, що незручно. Тому майже завжди обирають більший масштаб. Надалі світло буде розглядатись з точки зору геометричної оптики – як промінь.

Коли світло досягає поверхні, мають значення два аспекти поверхні: середовища з обох сторін від поверхні та геометрія самої поверхні. Після контакту з поверхнею світло розділяється на два

пучки: відбите та заломлене, де кут заломлення залежить від індексів заломлення двох субстанцій. Очевидно, що чим більше світла відбито, то тим більш освітленою виглядає поверхня. Заломлене світло, в свою чергу, починає проникати в середовище. Глибина проникнення залежить від діелектричної проникності середовища. Чим вона вище, тим більш прозорим є середовище, що означає, що світло проникає через середовище. Це світло починає розсіюватись. Але середовище не є однорідним. В реальному світі атоми чи молекули, зазвичай, не утворюють ідеальну структуру з точки зору топології. Також в реальних матеріалах зустрічаються вкраплення інших молекул чи атомів, що відбивають та заломлюють світло всередині середовища. Через таку неоднорідність середовища світло починає розсіюватись. Якщо діелектрична проникність середовища низька, то світло швидко поглинається середовищем. Якщо діелектрична проникність висока, то світло проходить крізь середовище лише з невеликою втратою інтенсивності.

Розсіяне всередині середовища світло через неоднорідність цього середовища в загальному випадку покидає його в випадкових місцях. Так, наприклад, воно може знайти вихід з середовища в тому місці, в якому й проникло в нього. Даний ефект зображено на рисунку 2.6 [4]. Таке світло називається дифузним світлом (diffuse light).

Також важливою є і геометрія поверхні. Кожна реально існуюча поверхня має нерівномірності. При цьому, нерівномірності поверхні, що мають розмір набагато менші, аніж довжина хвилі, майже не мають жодного ефекту. Нерівномірності, що мають розмір набагато більший за довжину хвилі, розглядаються як нахил поверхні, що не впливають на локальну плоскість поверхні. Лише нерівномірності, що лежать в границях від 1 до 100 довжин хвиль змушують світло поводитись інакше, ніж при взаємодії з гладкою поверхнею.

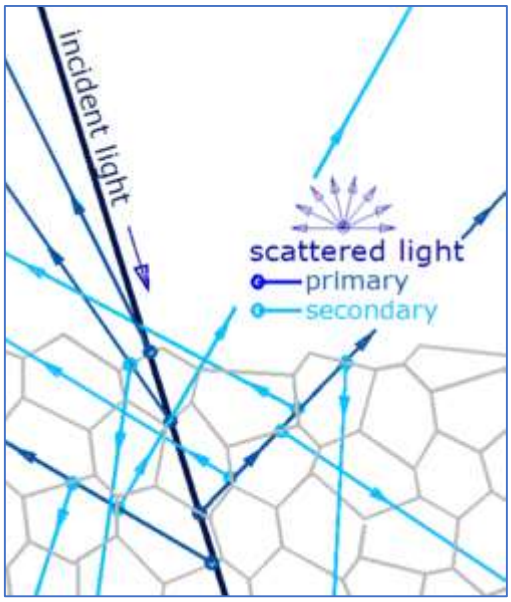

Рисунок 2.6 – Проникність світла в неоднорідному середовищі

Нерівномірності поверхні, що набагато менші за довжину хвилі, називаються мікроповерхнями. Мікроповерхня вважається пласкою та має власну нормаль. Геометрія будь-якої поверхні розглядається як набір мікроповерхонь.

Кожна видима точка поверхні складається з багатьох мікроповерхонь, що відбивають світло в різних напрямках. Орієнтація кожної мікроповерхні є випадковою, що робить зручним представлення поверхні як статистичний розподіл мікронормалей. Для більшості поверхонь розподіл нормалей мікроповерхонь є неперервною з піком в нормалі макроскопічної поверхні. Зручно ввести наступний параметр, який назвемо шорсткість. Шорсткість встановлює наскільки «розтягнутим» є цей пік або, іншими словами, наскільки цей пік є сильним. Візуально помітним ефектом збільшення шорсткості поверхні є збільшення розмиття відбитого світла. У випадку малих точкових джерел світла розмиття призводить до збільшення в розмірах дзеркального відблиску, при цьому края відблиску розмиваються. Цей ефект можна спостерігати на рисунку 2.7 [4].

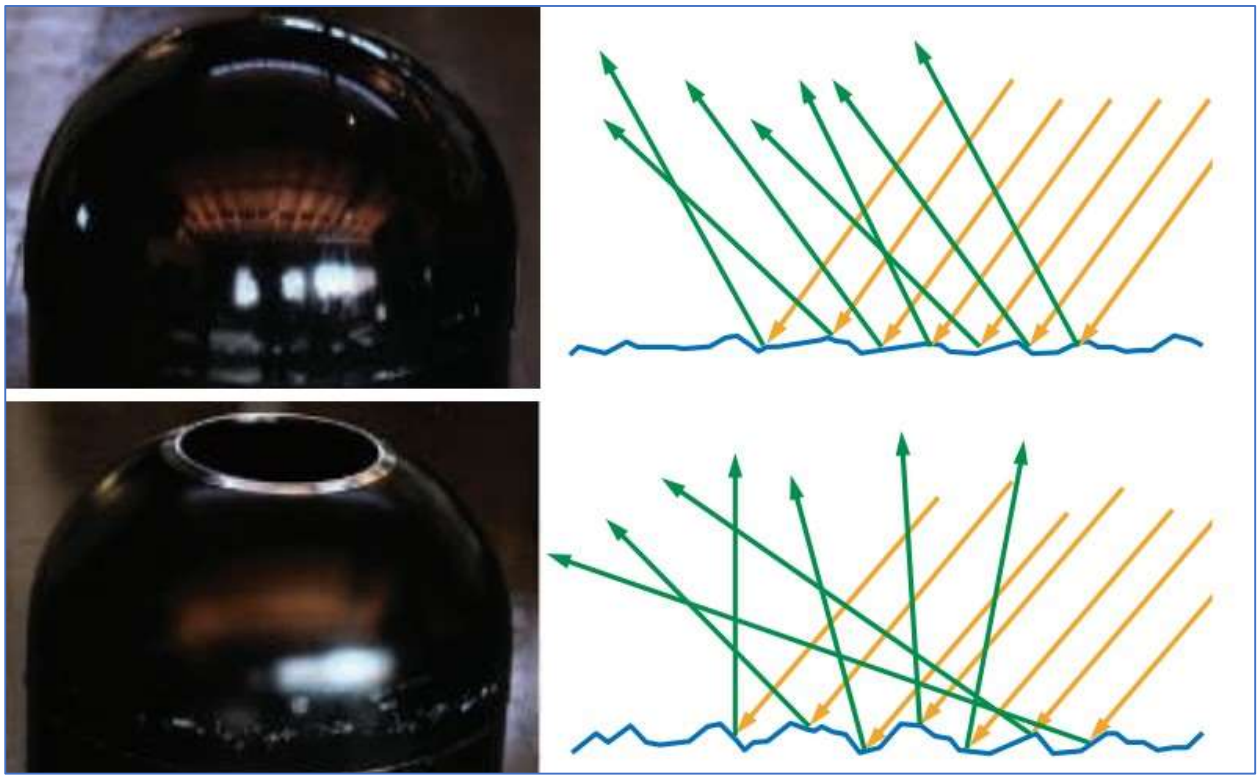

Рисунок 2.7 – Залежність чіткості відображення від шорсткості поверхні

Також варто зазначити, що існує ефект затінювання мікроповерхонь від інших мікроповерхонь. Цей ефект можна спостерігати на рисунку 2.8 [4].

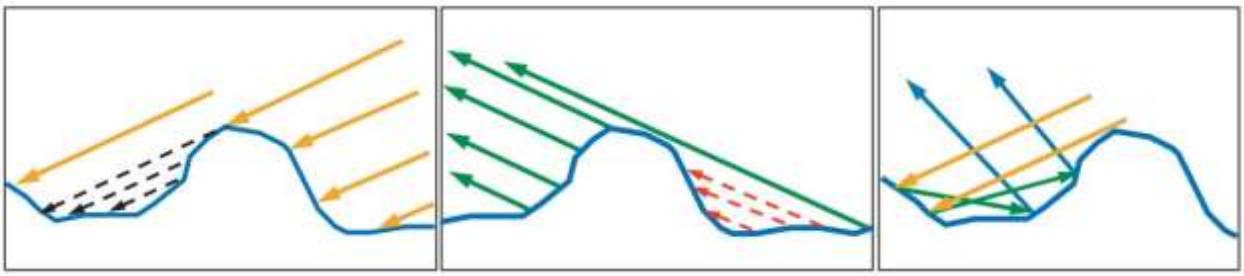

Рисунок 2.8 – Затінювання мікроповерхні

Визначення коефіцієнту відбитого світлу від поверхні залежить від шорсткості поверхні. Шорсткість визначає відсоток мікронормалей поверхні, що вирівняні з макронормаллю поверхні в певному діапазоні та вплив самозатінення на відбите світло. Також необхідна і інша функція, що визначає вплив самозатінення. Обидві ці функції повертають коефіцієнти, що перемножуються між собою. Однією з

найбільш популярних функцій коефіцієнту вирівняних мікронормалей є функція GGX, яка приймає в якості аргументу шорсткість поверхні. Чим більш шорстка поверхня, тим більш розмитими є відблиски. Різні значення шорсткості призводять до іншого зображення, що можна спостерігати на рисунку 2.9 [5].

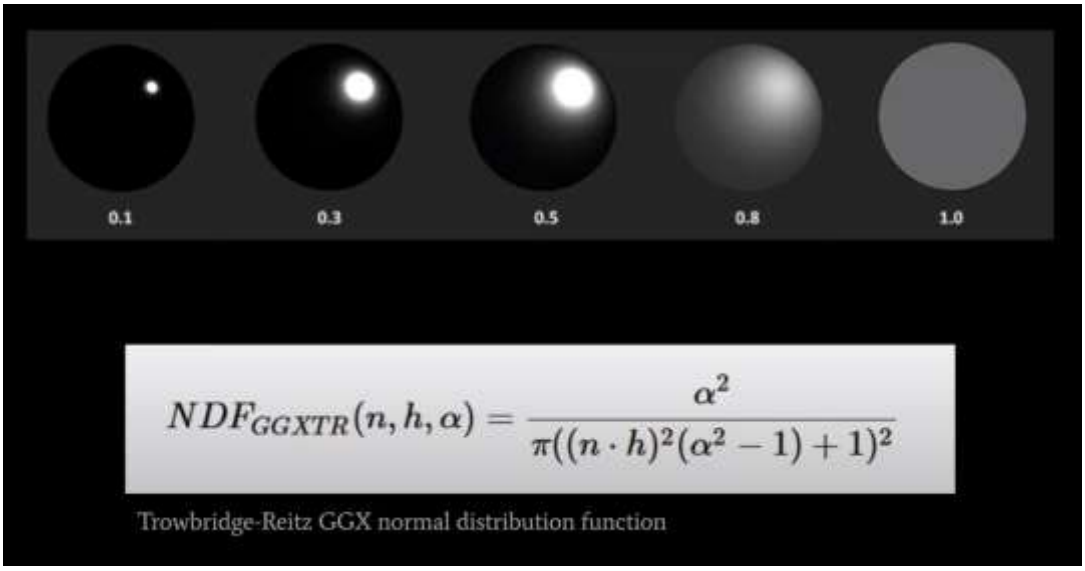

Рисунок 2.9 – Функція GGX від шорсткості поверхні

Найбільш відомоб функцією самозатінення є функція Шліка-Сміта. Вона також приймає в якості аргументу шорсткість об’єкту. Вплив різних значень шорсткості на результат показано на рисунку 2.10 [5].

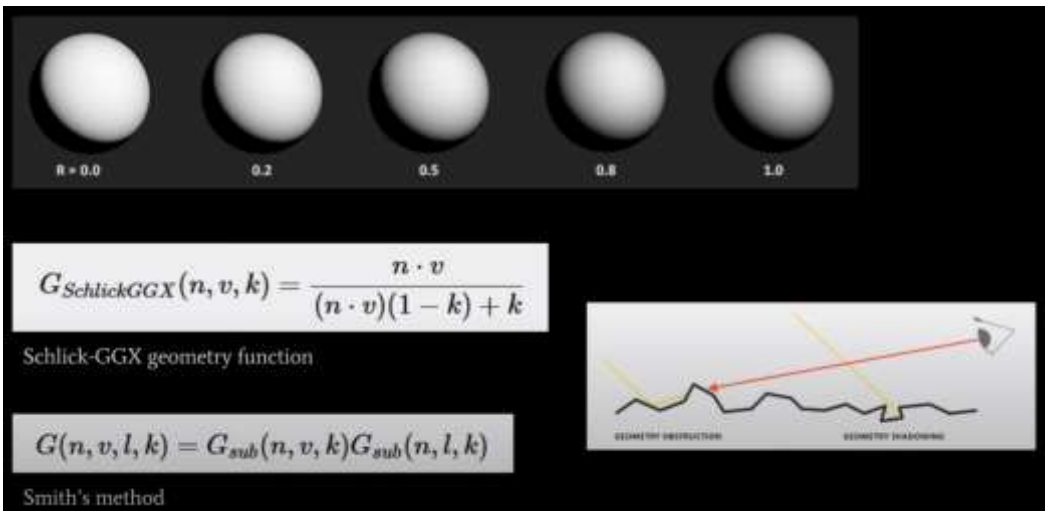

Рисунок 2.10 – Функція Шліка-Сміта самозатінення поверхні від шорсткості

До цього моменту розглядався лише аспект геометрії поверхні. Ще одною важливою характеристикою поверхні є матеріл.

2.6.2. Двопроменева функція відбивної спроможності

Відображення поверхонь за допомогою фізично коректних моделей відображення зводиться до визначення енергетичної яскравості (radiance), що потрапляє на камеру з певного набору можливих напрямків.

В залежності від матеріалу поверхні коефіцієнт відбитого світла змінюється. Також змінюється і кількість дифузного світла. Ба більше – коефіцієнт відбитого світла для кожної довжини хвилі різний. Але моделювання пучка світла як множину багатьох пучків світла з різними довжинами потребує занадто багато обчислювальних ресурсів, щоб використовувати такі методи в рендерерах реального часу. Тому світло розбивається на три проміжки: червоний, зелений та синій. Таке розбиття обумовлене будовою людського ока, яке має три різних типу колбочок, де кожен тип відповідає за сприйняття одного з цих проміжків кольору. Кожен інший колір, що може бути сприйняти людиною, є комбінацією цих трьох в різних пропорціях.

Коефіцієнти відбиття світла від поверхні визначаються в лабораторіях в ідеальних умовах. Надалі ці значення використовуються в рендерерах для визначення частки відбитого світла в трьох діапазонах.

Але інтенсивність відбитого світла не може бути вищою за інтенсивність світла, що потрапило на поверхню. Функція, що визначає цей коефіцієнт, називається двопроменевою функцією відбивної спроможності. Існує багато варіацій цієї функції, де розглядається не лише відбите світло. Всі вони придатні для використання в рендерерах, але в різних умовах.

2.6.3. Вибір PBR workflow

Всі вище описані методи симуляції поверхонь знайшли втілення в такому понятті як PBR workflow – метод представлення об’єкту в поняттях фізично коректного відображення. На даний момент існує два найбільш розповсюджених PBR workflow: metal/roughness та specular/glossiness. Вони досить схожі та дають майже однаковий результат відображення. Але процес створення об’єктів за цими методами відрізняється.

Для представлення об’єкту в metal/roughness workflow потрібно три наступних компоненти: базовий колір об’єкту (albedo), металевість (metalness) та шорсткість (roughness). Колір об’єкту задається для кожного з трьох діапазонів кольорів окремо. Металевість є скалярним значенням від 0 до 1, де 1 означає ідеальний метал, а 0 – діелектрик. Шорсткість також є скалярним значенням від 0 до 1. Вплив цього параметру на відображення об’єкту описано вище.

Для specular/glossiness workflow кожен об’єкт має наступний набір параметрів: базовий колір об’єкту (albedo), блискучість (glossiness) та дзеркальність (specular). Колір об’єкту також задається для кожного з діапазонів окремо, як і у випадку з metal/roughness workflow. Блискучість об’єкту є зворотнім від шорсткості і також представлене у вигляді скалярного значення. А от дзеркальність об’єкту задається для кожного з діапазонів кольору окремо.

Для представлення об’єкту з використанням metal/roughness workflow потрібно мінімум 2 текстури. В одній буде базовий колір, а в інший в різних каналах будуть знаходитись шорсткість та металевість. Для представлення об’єкту з використанням specular/glossiness workflow потрібно не менше двох текстур: одна для базового кольору, одна для дзеркальності та ще одна для блискучості. Але блискучість можна записати в альфа-канал текстури дзеркальності, що призведе до економії місця. Але як би то не було, specular/glossiness workflow

вимагає більше місця для зберігання і, відповідно, створює більше навантаження на систему пам’яті графічного процесору.

Порівняння текстури цих двох workflow зображено на рисунку 2.11 [6].

Рисунок 2.11 – Порівняння metal/roughness workflow та specular/glossiness workflow

Тож постає вибір: який з цих workflow використовувати. В індустрії рендерерів реального часу використання metal/roughness workflow переважає над specular/glossiness workflow. В першу чергу це пов’язане з тим, що фінальні зображення в залежності від використаного workflow майже не відрізняється. Але specular/glossiness workflow створює додаткове навантаження на графічний процесор, що в рендерерах реального часу не є бажаним. Тому доцільним вважаю використання саме metal/roughness workflow.

2.7. Проектування програмного додатку для дослідження і демонстрації можливостей рендерера

Графічний рендерер не може бути самодостатньою програмою за визначенням. Графічний рендерер, як вже було сказано, являє собою програмний компонент. Виконаний він може бути або у вигляді бібліотеки, або у вигляді фреймворку, або в якомусь іншому вигляді.

Тому, щоб продемонструвати можливості рендерера, потрібно створити програмний додаток, частиною якого буде графічний рендерер. Такий програмний додаток повинен реалізовувати наступні вимоги:

- обробка вводу користувача;
- робота зі стандартизованими типами файлів зберігання графічних об’єктів та сцен;
- забезпечення можливості змінювати налаштування рендерера на льоту.

Зупинимось детальніше на виборі типу файлів зберігання графічних об’єктів та сцен. В загальному випадку формат такого файлу і, відповідно, можливості повинні відповідати тим вимогам, що постають перед розробниками програмного додатку. Адже кожен додаток реалізує власні функціональні можливості. І деякі стандарти форматів зберігання графічних об’єктів дозволяються реалізовувати поставлені задачі, а деякі ні.

Станом на зараз розроблено багато різних форматів. Кожен з яких має власні переваги та недоліки. Деякі з них направлені на універсальність використання, деякі заточені під специфічне використання. Найкращим варіантом було б розробити власний стандарт зберігання графічних об’єктів, що відповідав би власним вимогам. При цьому такий формат мав би всі необхідні функціональні можливості і не мав би надлишкових. Але в такому випадку одразу постала би проблема: як створювати графічні об’єкти?

Використовувати існуючи засоби моделювання дуже сильно ускладнилось, адже вони використовують або власні формати зберігання об’єктів, або найбільш поширені в індустрії. Створювати власний редактор графічних об’єктів не представляється можливим. Тому єдиним виходом є обрати один або декілька форматів, що будуть підтримуватись додатком.

Найбільш підходящим на роль універсального формату зберігання графічних об’єктів є glTF (Graphics Language Transmission Format). Цей формат з’явився в публічному доступі в 2015 році, що робить його досить новим та актуальним. Особливістю цього формату є його компактність та зрозумілість. Зберігання інформації в файлі даного формату можливе в двох варіантах: текстовому (розширення файлу .gltf) або бінарному (розширення файлу .glb). Перший варіант потребує більше місця, але є простим для розуміння людиною. Також в такий файл легко вносити зміни, а не заново експортувати графічну модель з графічного редактору. Іншим варіантом зберігання інформації є бінарний. Такий варіант потребує менше місця та його обробка триває менше часу, ніж обробка аналогічного текстового. При цьому в даних файлах зберігається лише головна інформація про графічний об’єкт. Всі дані, як от буфери вершин, індексні буфери, текстури тощо зберігаються в бінарному вигляді або всередині цих файлів, або в інших файлах. Тож головний файл лише посилається на цю інформацію.

Іншою ключовою особливістю даного формату зберігання файлів є підтримка фізично коректних моделей освітлення. Даний формат підтримує як metal/roughness workflow, так і specular/glossiness workflow. При цьому підтримка першого є пріоритетною і бажаною, що сказано в специфікації до цього формату.

Також присутня підтримка графічних сцен, що зберігаються у вигляді графу.

Також присутня підтримка анімації об’єктів. Проте дана можливість в чинному рендерері використана не буде, тому дана можливість є надлишковою.

Структура файлу типу glTF зображена на рисунку 2.12.

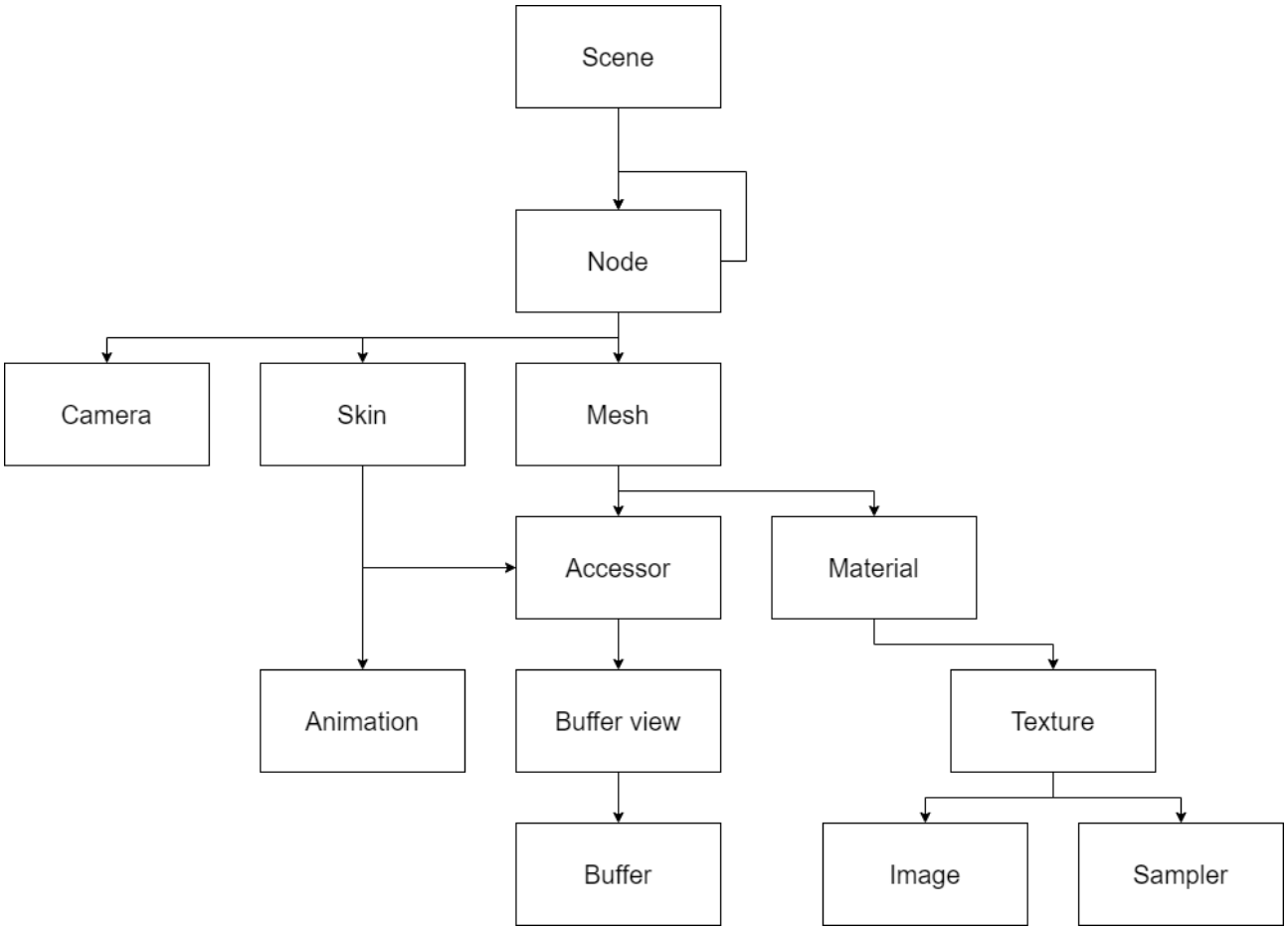

Рисунок 2.12 – Структура файлу формату glTF

3. РЕАЛІЗАЦІЯ ГРАФІЧНОГО РЕНДЕРЕРА

Графічний рендерер складається з трьох наступних класів:

- Renderer;
- VulkanGraphicsController;
- VulkanContext.

Схематичне зображення взаємодії цих класів між собою зображено на рисунку 3.1.

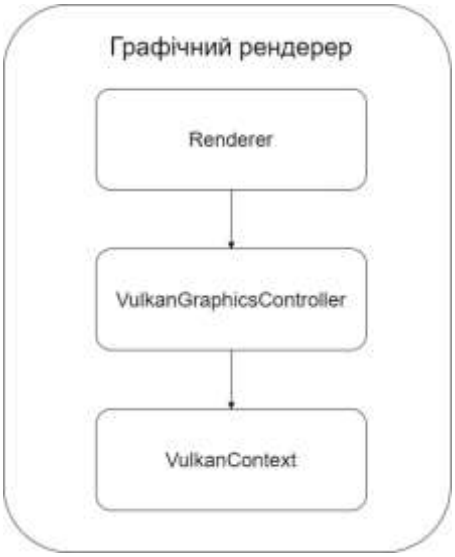

Рисунок 3.1 – Зв’язок модулів графічного рендерера

3.1. Реалізація графічного контексту

Компонент графічний контекст виконано у вигляді класу VulkanContext.

Завданням графічного контексту є коректне навантаження графічного процесору та вивід фінального зображення на екран користувача. Зважаючи на використання графічного API Vulkan, постають два задання: синхронізація між генерацією зображення та виводом цього зображення на екран і питання швидкодії.

По-перше, недоцільно виводити зображення, що ще не згенероване, і недоцільно генерувати нове зображення в те, що в даний час виводиться на екран.

По-друге, недоцільно чекати, доки зображення буде виведено на екран, адже в цей час графічний процесор може використовувати інше зображення, яке потім буде виведено на екран. Оновлення зображення на сучасних моніторах займає близько 1-5мс, що досить суттєво.

Для вирішення першого завдання було використано ті примітиви синхронізації, що надає Vulkan. Для синхронізації хосту з девайсом використовуються VkFence, а для синхронізації операцій на девайсі між собою використовуються VkSemaphore.

Для вирішення другої задачі було зроблено наступне. Одночасно використовуються три різних зображення. Одне з них може використовуватись для виводу зображення на екран, друге може використовуватись для одночасного генерування зображення 3Д-сцени. Третє зображення використовується тоді, коли генерація наступного зображення вже завершилась, але вивід попереднього зображення ще ні. Тому в третьому зображенні зберігається найбільш актуальне зображення. Зрозуміло, що при частоті кадрів більшою за частоту оновлення монітору частина згенерованих зображень ніколи не буде відображена на екрані. Це призводить до надлишкового навантаження на графічний процесор. На персональних комп’ютерах така проблема не є критичною. Вона проявляється лише на пристроях, де тепловиділення має жорсткі рамки, як от смартфони чи ноутбуки, де бажано використовувати іншу політику.

При ініціалізації графічного контексту відбувається наступна послідовність дій:

1) створюється екземпляр VkInstance;
2) створюється екземпляр VkSurface;
3) обирається графічний процесор із наявних;
4) створюється екземпляр VkDevice;
5) створюється екземпляр VkSwapchain;
6) створюються екземпляри VkFence та VkSemaphore;

3.2. Реалізація графічного контролеру

Компонент графічний контролер виконано у вигляді класу VulkanGraphicsController.

Умовно кажучи, графічний контролер реалізує ті можливості, що надає API Vulkan, але в більш загальному вигляді, без прив’язки до конкретного API, адже реалізацій графічного контролера може бути декілька.

Додатково графічний контролер відповідальний за створення буферів команд – VkCommandBuffer. В ці буфери відбувається запис команд на проведення операцій на девайсі. Потім ці командні буфери надсилаються на графічний процесор для виконання.

Графічний рендерер надає інтерфейс для створення та подальшого використання наступних типів даних:

- RenderPass;
- Framebuffer;
- Image;
- Buffer;
- Shader;
- Pipeline;
- Sampler;
- UniformSet.

Кожне з цих понять має свій аналог в API Vulkan, але ці типи даних є фундаментальними для будь-якого графічного API, адже графічний пайплайн в кожному графічному процесорі однаковий, хоч один і той самий графічний процесор може підтримувати декілька графічних API.

Графічний контролер відповідальний за наповнення буферів команд, які потім передаються графічному контексту, який додає синхронізацію та відправляє їх графічному процесору на виконання.

Для наповнення буферів команд графічний контролер надає множину методів. Частина цих методів змінює стан (state) буферу команд, але на графічному процесорі не виконується. Інша частина є командами на виконання безпосередньо на графічному процесорі.

Командами на зміну стану командного буферу є:

- зміна кадрового буферу;
- зміна графічного пайплайну;
- зміна вершинного буферу;
- зміна індексного буферу;
- зміна набору юніформ;
- зміна динамічних станів графічного пайплайну.

Командами, що безпосередньо виконуються на графічному процесорі, є:

- відображення об’єкту з використанням індексного буферу;
- відображення об’єкту без використання індексного буферу;
- оновлення даних буферу;
- оновлення даних зображення.

3.3. Реалізація 3Д-рендерера

Компонент 3Д-рендерер реалізовано у вигляді класу Renderer. Блок-схема алгоритму функціонування даного компоненту зображено на рисунку 3.2.

Стрижневим в реалізації 3Д-рендерера є реалізація методу відкладеного освітлення. Послідовність дій при відображенні 3Д-сцени наступна:

- G pass: відображення кожного об’єкту в спеціальному G-буфері, де зберігаються значення metal/roughness workflow;
- depth copy pass: копіювання даних з буферу глибини в інший, дублювання даних;

- lightning pass: використання даних з G-буфера та буферу глибини для освітлення об’єктів;
- present pass: tone mapping та вивід фінального зображення.

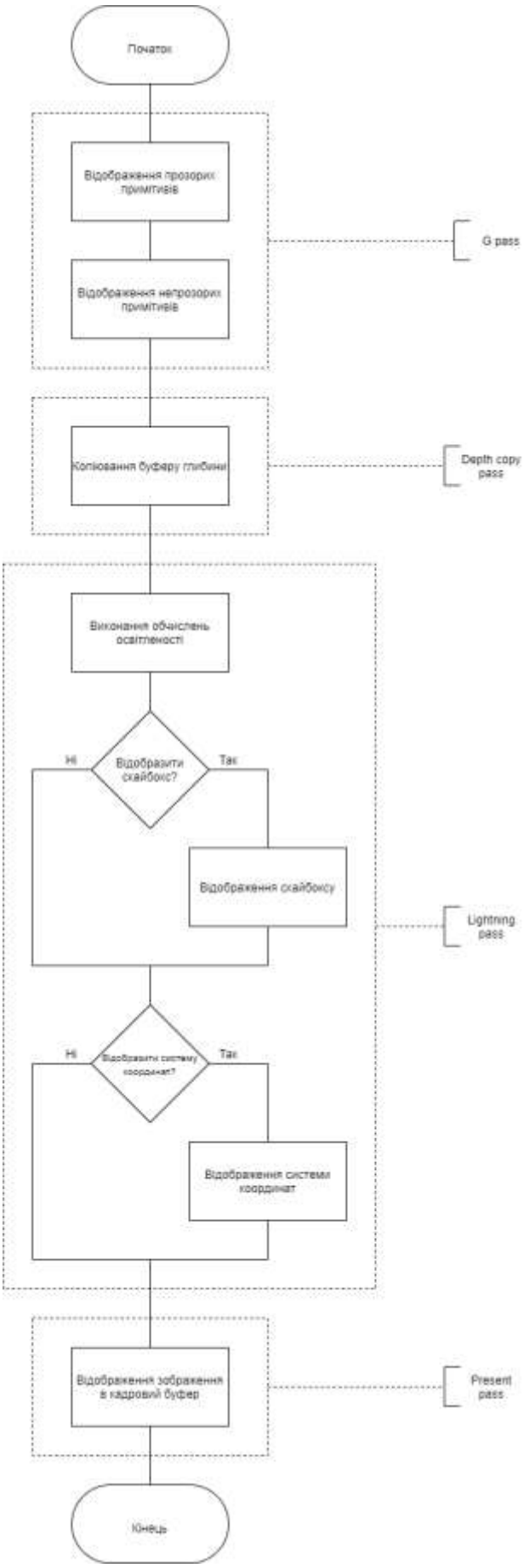

Рисунок 3.2 – Блок-схема алгоритму функціонування 3Д-рендерера

Графічно ця послідовність зображена на рисунку 3.3.

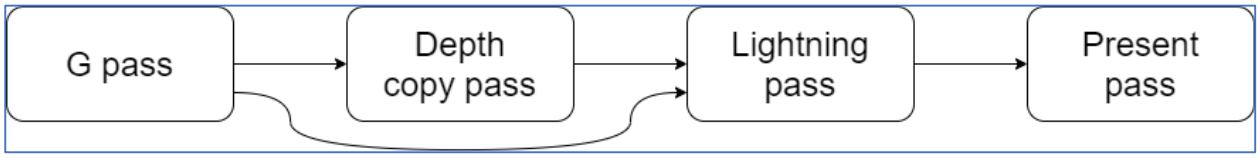

Рисунок 3.3 – Послідовність операцій 3Д-рендерера

Методи класу Renderer, що надають можливості створення графічних примітивів або відображення графічної сцени, наведені на рисунку 3.4.

Renderer
1. set_resolution()
2. begin_frame()
3. end_frame()
4. draw_primitive()
5. draw_skybox()
6. materials_create()
7. skybox_create()
8. vertex_buffer_create()
9. index_buffer_create()

Рисунок 3.4 – Інтерфейс класу Renderer

1. Метод, що встановлює роздільну здатність згенерованого зображення.
2. Метод, що встановлює початок кадру.
3. Метод, що встановлює кінець кадру.
4. Метод, що дозволяє відобразити примітив з використанням матеріалу.
5. Метод, що дозволяє відобразити скайбокс.
6. Метод, що дозволяє створили матеріали.
7. Метод, що дозволяє створити скайбокс.
8. Метод, що дозволяє створити вершинний буфер.
9. Метод, що дозволяє створити індексний буфер

3.4. Реалізація програмного додатку

Для реалізації програмного додатку потрібно вирішити наступні завдання:

- реалізувати можливість створення вікна додатку;
- реалізувати обробку користувацького вводу.

Для вирішення обох завдань було використано бібліотеку glfw. Це платформонезалежна бібліотека, що дозволяє створювати, змінювати та проводити різноманітні маніпуляції з вікном операційної системи. Також вона дозволяє обробляти користувацький ввід.

При цьому, варто зазначити, що роздільна здатніть зображення, що генерується рендерером, статична, і не змінюється зі зміною роздільної здатності вікна операційної системи. Це значно спрощує графічний рендерер та взагалі процес взаємодії з графічною підсистемою і, в той же час, дозволяє реалізувати алгоритм згладжування зображення SSAA (Super Sample Anti-Aliasing). Суть цього методу полягає в генеруванні зображення, що має роздільну здатність вище, ніж у екрана користувача. Таким чином, прибираються «зубці», що виникають на краях об’єктів. Вплив згладжування SSAA продемонстровано не рисунку 3.5 [7], де ліва частина зображення згенерована без використання будь-яких технологій згладжування, а права – за допомогою згладжування SSAA.

Рисунок 3.5 – Вплив згладжування SSAA на відображення об’єктів

Для вирішення поставлених завдань було розроблено наступну структуру програмного додатку, яка зображена на рисунку 3.6. Також з точки зору програмного додатку графічний рендерер є окремим модулем, функціональні можливості якого він використовує. При цьому інтерфейс взаємодії з графічним рендерером надає 3Д-рендерер.

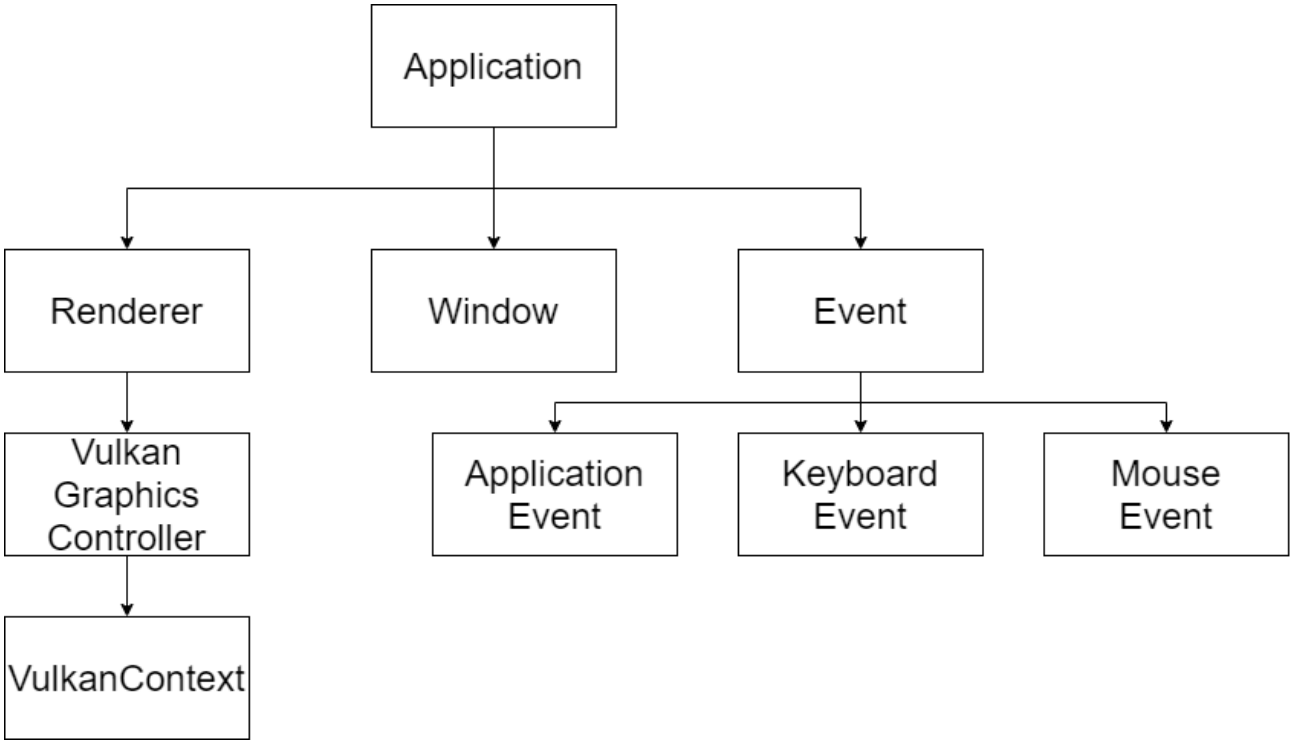

Рисунок 3.6 – Структура програмного додатку

Кожен модуль з цієї структури представлений у вигляді окремого класу. Блок-схема алгоритму функціонування програмного додатку зображена на рисунку 3.7.

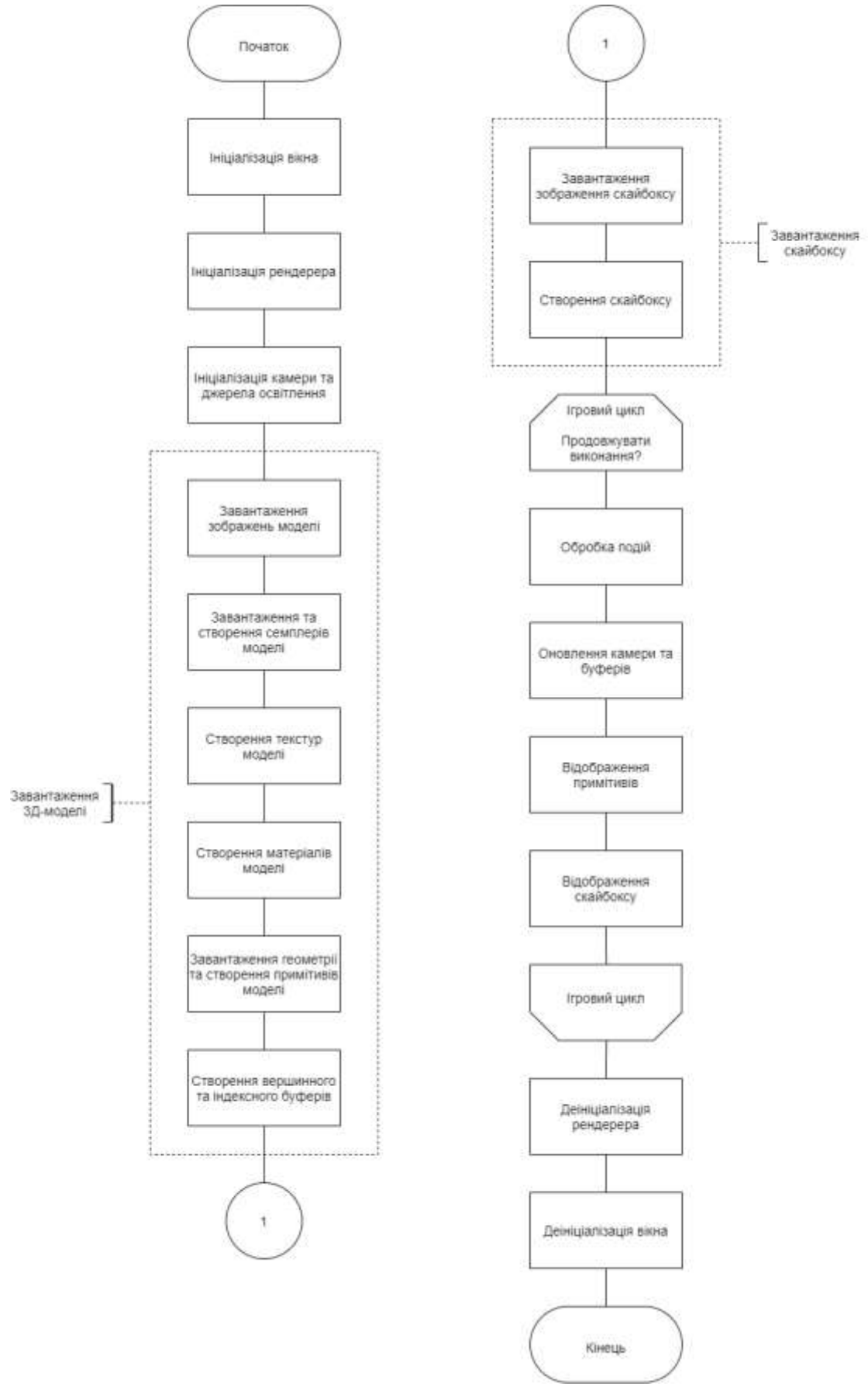

Рисунок 3.7 – Блок-схема алгоритму функціонування програмного додатку

4. ТЕСТУВАННЯ ГРАФІЧНОГО РЕНДЕРЕРА

4.1. Тестування рендерера в залежності від графічної складності сцени

Тестування проводились з встановленою роздільною здатністю графічного рендерера 1280 на 720 пікселів. Також використовувався скайбокс з роздільною здатністю 1024 на 1024 пікселів.

Використовувалось декілька 3Д-моделей:

- автомобіль – 8338 полігонів, 5412 вершин, рисунок 4.1;
- шолом – 11306 полігонів, 8870 вершин, рисунок 4.2;
- будівля – 57043 полігонів, 61238 вершин, рисунок 4.3;
- мавзолей – 199966 полігонів, 191640 вершин, рисунок 4.4.

Рисунок 4.1 – 3Д-модель автомобіля

Рисунок 4.2 – 3Д-модель шолома

Рисунок 4.3 – 3Д-модель будівлі

Рисунок 4.4 – 3Д-модель мавзолея

Було проведено ряд тестувань, де швидкодія вимірювалась в частоті кадрів на секунду. Результати наступні:

- 3Д-модель автомобіля: середня частота кадрів склала 151 кадр на секунду. Частота кадрів зі схованим об’єктом склала 246 кадрів на секунду;
- 3Д-модель шолома: середня частота кадрів склала 101 кадрів на секунду. Частота кадрів зі схованим об’єктом склала 242 кадри на секунду;
- 3Д-модель будівлі: середня частота кадрів склала 115 кадрів на секунду. Частота кадрів зі схованим об’єктом склала 232 кадри на секунду;
- 3Д-модель мавзолея: середня частота кадрів склала 121 кадр на секунду. Частота кадрів зі схованим об’єктом склала 207 кадрів на секунду.

Варто пояснити отримані результати. Графічна складність визначається двома параметрами: кількістю полігонів 3Д-моделей та об’ємом обчислень освітленості поверхонь. Перший параметр напряму залежить від 3Д-об’єкта. Другий параметр залежить від кількості

пікселів, які цей об’єкт займає. Чим їх більше, тим більше обчислень потрібно провести. Тому для кожного об’єкту обирався такий ракурс, щоб він займав приблизно таку саму кількість пікселів, що і інші об’єкти. І з результатів чітко видно, що чим більша кількість полігонів об’єкту, тим нижча частота кадрів.

Також було проведено наступний дослід: об’єкт «ховався» з поля зору камери. Таким чином, жодних обчислень освітленості поверхонь не було. З такого досліду можливо встановити вплив кількості полігонів на швидкодію графічного рендерера. З цього досліду чітко видно, що чим більше полігонів обробляється графічним процесором, тим нижче швидкодія. При цьому, навіть якщо об’єкт не відображається на сцені. Таке порівняння корисне, щоб оцінити максимально теоретичну швидкодію графічного рендерера з даним об’єктом. Тому, таким чином, можна оцінити вплив обчислень освітленості на швидкодію.

Також було проведено інший дослід: скайбокс було прибрано і зроблено всі ті самі тести. Результати наступні:

- 3Д-модель автомобіля: середня частота кадрів склала 161 кадр на секунду. Частота кадрів зі схованим об’єктом склала 262 кадри на секунду;
- 3Д-модель шолома: середня частота кадрів склала 106 кадрів на секунду. Частота кадрів зі схованим об’єктом склала 255 кадрів на секунду;
- 3Д-модель будівлі: середня частота кадрів склала 118 кадрів на секунду. Частота кадрів зі схованим об’єктом склала 239 кадрів на секунду;
- 3Д-модель мавзолея: середня частота кадрів склала 124 кадри на секунду. Частота кадрів зі схованим об’єктом склала 213 кадрів на секунду.

З результатів видно, що швидкодія збільшилась. Але це збільшення несуттєве. При цьому, зображення без скайбоксу виглядає нереалістично. Приклад наведений на рисунку 23.

Рисунок 4.5 – Графічна сцена без скайбоксу

4.2. Тестування рендерера в залежності від налаштувань

Чинний рендерер дозволяє налаштовувати роздільну здатність згенерованого зображення. Було проведено ряд тестів з такою графічною сценою: скайбокс з роздільною здатністю 1024 на 1024 пікселів та 3Д-модель будівлі. В кожному з тестів змінювалась роздільна здатність згенерованого зображення. Результати наступні:

- роздільна здатність 640 на 480 пікселів: середня частота кадрів склала 179 кадрів на секунду;
- роздільна здатність 800 на 600 пікселів: середня частота кадрів склала 146 кадрів на секунду;
- роздільна здатність 1280 на 720 пікселів: середня частота кадрів склала 121 кадр на секунду;
- роздільна здатність 1920 на 1080 пікселів: середня частота кадрів склала 83 кадри на секунду;

- роздільна здатність 2560 на 1440 пікселів: середня частота кадрів склала 59 кадрів на секунду;

- роздільна здатність 3840 на 2160 пікселів: середня частота кадрів склала 37 кадрів на секунду.

Очевидно, що зі збільшенням частоти кадрів знижується швидкодія. Але сприйняття зображення людиною також зростає. Тому при виборі роздільної здатності потрібно шукати той оптимальний варіант, де забезпечується плавність зображення, але від цього не страждає швидкодія.

5. ШЛЯХИ ПОЛІПШЕННЯ РОЗРОБКИ

5.1. Шляхи поліпшення графічного рендерера

Комп’ютерна графіка розвивається з шаленим темпом. Розвиваються як алгоритми та методи відображення, так і підходи до генерації зображення. Тому можливих шляхів поліпшення графічного рендереру дуже багато. Звичайно, не всі вони виправдані з точки зору корисності. Але є перелік покращень, які дають суттєве покращення зображення при відносно невеликих витратах на розробку. В першу чергу це реалізація методу image-based lightning (IBL).

Технологія IBL полягає у захопленні реальних значень освітленості певної точки простору за допомогою камер, що дозволяють знімати панорамні фото. Потім таке зображення проектується на сферу і використовується для симуляції освітлення об’єктів в графічній сцені.

Наступним важливим кроком може бути реалізація точкового динамічного освітлення. Станом на зараз графічний рендерер має можливість відображення освітлення лише від статичного джерела – сонця, яке не змінює своє направлення та інтенсивність.

Ще однією технологією збільшення реалістичності зображення є візуальний ефект, що має назву bloom. Цей ефект спостерігається і в реальному світі, коли на однорідному з точки зору яскравості зображенні знаходиться дуже яскравий об’єкт. Тоді цей яскравий об’єкт немов би засліплює собою інші. Цей ефект зображено на рисунку 5.1 [8].

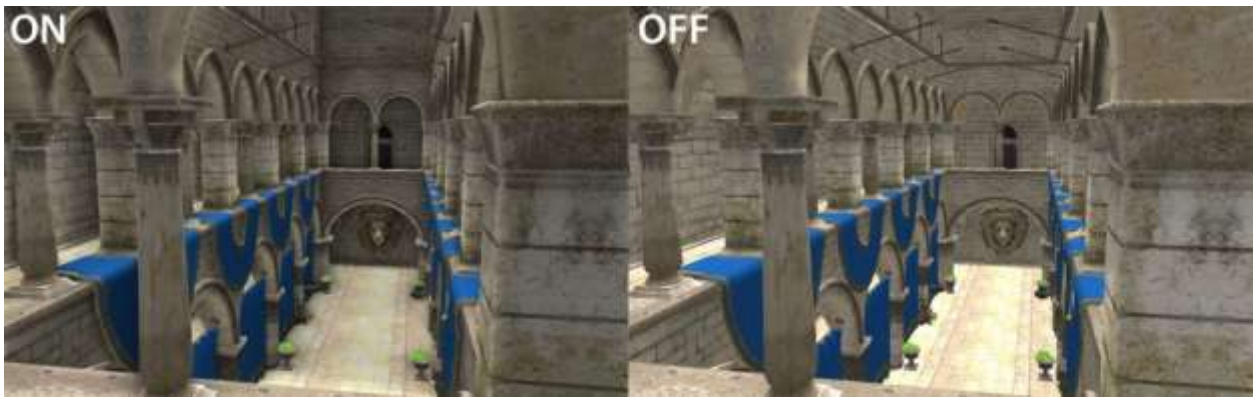

Рисунок 5.1 – Ефект bloom

Ще одним можливим покращенням зображення є система частинок (particle system). Це спосіб представлення об’єктів без чітких меж, таких як: дим, вогонь, хмари, туман, пара тощо. За допомогою цієї можливе моделювання багатьох явищ природи. Можливості системи частинок зображено на рисунку 5.2 [9].

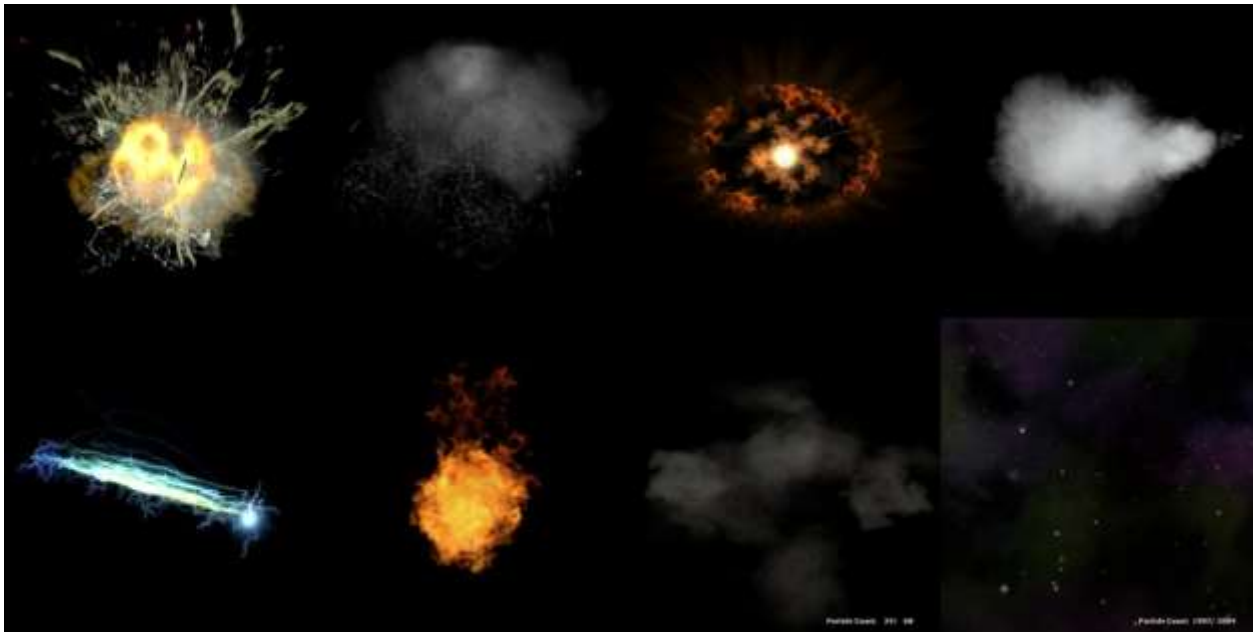

Рисунок 5.2 – Система частинок

Ще одним методом покращення графічного рендереру без великих зусиль є реалізація анімації об’єктів. Для вирішення цієї проблеми потрібно змінити вершинні шейдери деяких графічних пайплайнів, але ці зміни не є дуже суттєвими.

Ще одним способом покращити зображення є реалізація пост-ефекту SSAO (screen-space ambient occlusion). Ця технологія дозволяє наближено розраховувати навколишнє світло. Суть полягає в тому, що на об’єкти, які поруч мають багато сусідів, потрапляє менше світла, що робить їх менш яскравими. Наприклад, стики між цеглинами у стіні будуть освітлені менше, ніж самі цеглини, адже між цеглин потрапляє менше світла. Ефект від використання цієї технології зображено на рисунку 5.3 [10], де ліва частина зображення згенерована без використання SSAO, а права – з використанням SSAO.

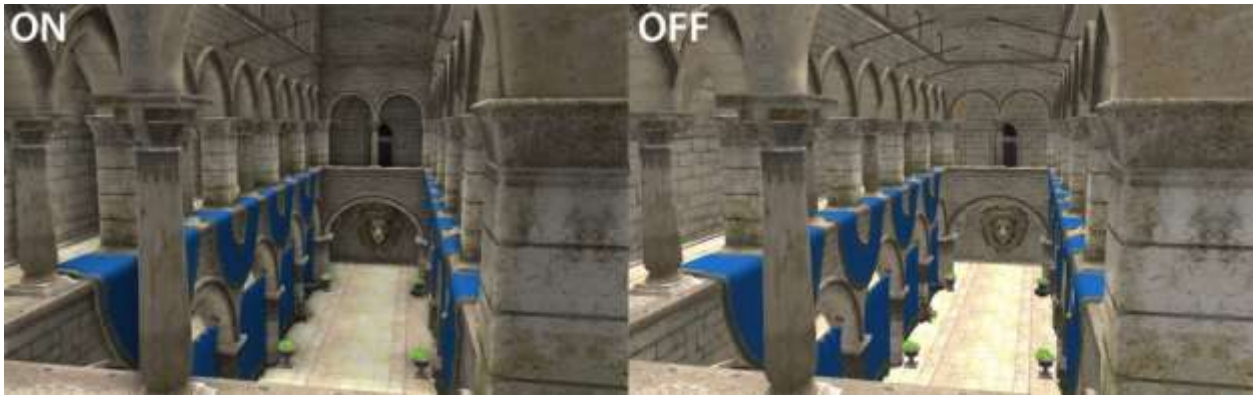

Рисунок 5.3 – Ефект від використання SSAO

Можливих шляхів покращення графічного рендерера дуже багато. Існують і більш складні методи, аніж ті, що описані вище. Наприклад, це можу бути реалізація системи погоди, де представлена можливість симулювати погодні умови, такі як: дощ, сніг, метелиця, пилова буря, гроза, смерч, удари блискавок тощо. Також можливим шляхом покращення графічного рендерера є використання технології трасування променів. Станом на зараз ця технологія є «в тренді». Але реалізація її вимагає внесення суттєвих змін до 3Д-рендерера та наявність графічного процесору, що підтримує апаратне прискорення трасування променів, адже дана технологія потребує багато обчислювальних ресурсів.

5.2. Шляхи поліпшення програмного додатку

Можливих шляхів поліпшення програмного додатку також багато. В першу чергу це ще збільшення ступеня контролю над налаштуваннями графічного рендереру.

Наступним кроком може бути творення системи фізики. Адже рух об’єктів за фізичними законами значно підвищує реалістичність зображення та сприйняття такого зображення людиною.

Ще одним шляхом покращення програмного додатку є створення інструментарію для створення демонстраційних анімаційних фільмів. Така демонстрація можливостей графічного рендерера має найвищий ступінь сприйняття користувачем. Також можливо реалізувати аудіо систему, що вкупі дає більший ступінь занурення користувача.

ВИСНОВКИ

Під час виконання дипломного проекту був розроблений графічний рендерер та програмний додаток для демонстрації можливостей рендерера.

В ході виконання дипломного проекту було був проведений аналіз предметної області та вже існуючих рішень.

Під час проектування було розглянуто різні підходи до реалізації графічного рендерера та методи відображення об’єктів. Детально описані технології, що використовуються, їх плюси та мінуси та обґрунтування доцільності використання обраних технологій.

Під час реалізації системи детально описано структуру та алгоритми функціонування основних компонентів розроблюваного продукту. Продемонстровано результати роботи графічного рендерера та проведено тестування графічного рендерера в різних умовах.

Розроблений графічний рендерер має місце використання в додатках, що відображають комп’ютерну графіку в реальному часі, а потреба в таких додатках є майже у всіх сферах людської діяльності. Також графічний рендерер може бути використаний для навчання людей комп’ютерній графіці.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Physically Based Rendering [Електронний ресурс] – Режим доступу до ресурсу: http://www.lived3d.com/blog/TAB_TiledShading.html.
2. Image synthesis and OpenGL: graphics pipeline [Електронний ресурс] – Режим доступу до ресурсу: http://romain.vergne.free.fr/teaching/IS/SI03-pipeline.html.
3. Rasterization: a Practical Implementation [Електронний ресурс] – Режим доступу до ресурсу: https://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/rasterization-stage.
4. Real-Time Rendering / [T. Akenine-Moller, E. Haines, N. Hoffman та ін.]. – 6000 Broken Sound Parkway NW, Suite 300 Boca Raton: CRC Press, 2018. – 1199 с. – (Taylor & Francis Group).
5. PBR Theory [Електронний ресурс] – Режим доступу до ресурсу: https://learnopengl.com/PBR/Theory.
6. О PBR на пальцах [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/funcorp/blog/465457/.
7. Типы сглаживания в играх [Електронний ресурс] – Режим доступу до ресурсу: https://www.iguides.ru/main/gadgets/other_vendors/tipy_sglazhivaniya_v_igrakh/.
8. Bloom (shader effect) [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Bloom_(shader_effect).
9. Particle System Overview [Електронний ресурс] – Режим доступу до ресурсу: https://developer.valvesoftware.com/wiki/Particle_System_Overview:ru.
10. Как работает затенение в компьютерных играх [Електронний ресурс] – Режим доступу до ресурсу: https://www.iguides.ru/main/other/kak_rabotaet_zatenenie_v_kompyuternykh_igrakh/.