НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп'ютерних систем

«На правах рукопису»
УДК 004.93

До захисту допущено:
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
«___»_____2022 р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-професійною програмою

«Системне програмування і спеціалізовані комп'ютерні системи»

зі спеціальності 123 «Комп'ютерна інженерія»

на тему: «Способи поліпшення візуалізації методом Forward+ рендерінгу»

Виконав:
студент ІІ курсу, групи КВ-12МП
Денисенко Іван Віталійович _____

Науковий керівник:
Доцент кафедри СПСКС, к.т.н., доцент,
Павловський Володимир Ілліч _____

Рецензент:
_____
_____ _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – другий (магістерський)

Спеціальність – 123 «Комп‘ютерна інженерія»

Освітньо-професійна програма «Системне програмування і спеціалізовані комп‘ютерні системи»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Віталій РОМАНКЕВИЧ
«___»_____2022 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Денисенко Івану Віталійовичу

1. Тема дисертації «Способи поліпшення візуалізації методом Forward+ рендерінгу», науковий керівник дисертації Павловський Володимир Ілліч, к.т.н., доцент, затверджені наказом по університету від «25» 11.2022р. №4317-С.
2. Термін подання студентом дисертації 12.12.2022.
3. Об’єкт дослідження: Алгоритм Forward+ рендерінгу, його модифікації та застосування у програмних засобах.
4. Вихідні дані: покращений алгоритм Forward+ рендерінгу з можливістю відображення напівпрозорих тіней.
5. Перелік завдань, які потрібно зробити:
    1) Проаналізувати наявні алгоритми рендерінгу графічних сцен
    2) Розробити модифікований алгоритм Forward+ рендерінгу, що здатен відображати напівпрозорі тіні.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: презентація.

7. Перелік публікацій:

1) Прикладна математика та комп'ютинг. «XV науково-практична конференція магістрантів та аспірантів ПМК-2022 факультету прикладної математики» (Київ, 16-18 листопада 2022 р.).
2) IX Міжнародній науково-технічній Internet-конференції «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами» (Київ, 25 листопада 2022 р.).

8. Дата видачі завдання: 7.10.2021

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Вивчення літератури за тематикою МД	02.10.2021	
2.	Розроблення та узгодження технічного завдання	07.10.2021	
3.	Аналіз існуючих рішень	19.12.2021	
4.	Підготовка матеріалів першого розділу МД	25.05.2022	
5.	Розроблення програмного забезпечення	07.09.2022	
6.	Тестування розробленого програмного забезпечення	24.10.2022	
7.	Підготовка матеріалів другого та третього розділів МД	26.10.2022	
8	Підготовка матеріалів четвертого розділу МД	15.11.2022	
9.	Оформлення документації дипломного МД	27.11.2022	
10.	Попередній розгляд магістерської дисертації на кафедрі	01.12.2022	

Студент Іван ДЕНИСЕНКО

Науковий керівник Володимир ПАВЛОВСЬКИЙ

РЕФЕРАТ

Головним джерелом в сприйнятті інформації людиною є зір – тобто, графічне подання інформації грає для людини найважливішу роль. Тому для ефективного сприйняття інформації і постає завдання надавати інформацію в візуальному, графічному, вигляді.

Важливим напрямком подання інформації в графічному вигляді є її генерування в реальному часі – рендерінг. Відбувається це на окремому обчислювальному пристрої – графічному процесорі, адже рендерінг графічної сцени важкий процес з точки зору обчислень.

Тому зростає потреба в алгоритмах рендерінгу, що ефективно використовують особливості сучасних графічних процесорів та забезпечують великий рівень візуальної якості результуючого зображення.

Об’єктом дослідження є проблема рендерінгу графічних сцен.

Предметом дослідження є алгоритм обчислення освітленості графічної сцени Forward+.

Мета роботи: покращити швидкодію алгоритму Forward+ рендерінгу при обчисленні освітленості графічної сцени, що здатен відображати напівпрозорі тіні, та розробка застосунку для відображення графічних сцен на основі модифікованого алгоритму.

Наукова новизна полягає в наступному: запропоновано спосіб відображення напівпрозорих тіней, який базується на алгоритмі shadow mapping, що підвищує візуальну якість результуючого зображення. Також запропоновано покращення швидкодії алгоритму Forward+ рендерінгу за рахунок реалізації кластерного освітлення, що дозволяє більш ефективно

фільтрувати джерела освітлення, які підлягають обробці, і, відповідно призводить до зменшення часу виконання алгоритму.

Практична цінність отриманих в роботі результатів полягає в підвищенні якості зображення завдяки можливості відображати напівпрозорі тіні та зменшенні часу виконання алгоритму Forward+ рендерінгу, що дозволяє або збільшити кількість джерел освітлення в графічній сцені, або генерувати більшу кількість кадрів за одиницю часу.

Розроблений застосунок рендерінгу графічних сцен має модульну архітектуру та не залежить від вибору графічного процесору, що дозволяє застосовувати його на будь-якому комп'ютері.

Апробація роботи. Основні положення та результати були розкладені на XV науковій конференції магістрантів та аспірантів «Прикладна математика та комп'ютинг» ПМК-2022 (Київ, 16-18 листопада 2022 р.). Також опублікована стаття на конференції «IX Міжнародна науково-технічна Internet-конференція «Сучасні методи, інформаційне, програмне та технічне забезпечення систем керування організаційно-технічними та технологічними комплексами» (Київ, 25 листопада 2022 р.).

Структура та обсяг роботи. Магістерська дисертація складається з вступу, чотирьох розділів та висновків.

У *першому розділі* проаналізовано предметну область: проблему обчислення освітленості графічних сцен та існуючі алгоритми рендерінгу графічних сцен. Також розглянуто та виявлено недоліки вже існуючих алгоритмів обчислення освітленості графічних сцен.

У *другому розділі* детально описано алгоритм Forward+ рендерінгу, детально описано та обґрунтовано модифікації алгоритму Forward+

рендерінгу, а сам: оптимізація алгоритму та покращення візуальної якості результуючого зображення.

У *третьому розділі* проведено аналіз та обґрунтовано вибір мови програмування та доступних на сьогодні інтегрованих середовищ розробки. Окрім цього, розділ містить детальний опис структури та архітектури розробленого застосунку.

У *четвертому розділі* розглянуто підходи до тестування застосунку та описано як саме зазначені підходи були використані при розробці застосунку та тестуванні фінальних результатів

У *висновках* представлені результати роботи.

Робота представлена на ХХ аркуші, містить посилання на список використаних літературних джерел.

Ключові слова: GPU, Forward, Deferred, Forward+, GPU, C++, Vulkan, рендерінг, шейдер.

ABSTRACT

The main source of human perception of information is sight - that is, the graphic presentation of information plays the most important role for a person. Therefore, for the effective perception of information, the task of providing information in a visual, graphic form arises.

An important direction of presenting information in graphic form is its generation in real time - rendering. This happens on a separate computing device - a graphics processor, because the rendering of a graphic scene is a difficult process from the point of view of calculations.

Therefore, there is a growing need for rendering algorithms that effectively use the features of modern graphics processors and provide a high level of visual quality of the resulting image.

The **object of research** is the problem of rendering graphic scenes.

The **subject of research** is the Forward+ rendering algorithm for lighting calculations.

The **purpose of the work:** to improve the speed of the Forward+ rendering algorithm when calculating the lighting of a graphic scene capable of displaying translucent shadows, and to develop an application for displaying graphic scenes based on the modified algorithm.

The scientific innovation is as follows: optimization improvements of the Forward+ rendering algorithm are proposed, which leads to reduction of the execution time of this algorithm. A method of displaying translucent shadows is also proposed, which increases the visual quality of the resulting image.

The practical value of the results obtained in the work consists in reducing the execution time of the Forward+ rendering algorithm and increasing the quality of the image due to the ability to display translucent shadows.

The developed graphic scene rendering application has a modular architecture and does not depend on the choice of graphics processor, which allows it to be used on any computer.

Approbation of work. The main provisions and results were presented at the XV Scientific Conference of Master's and Postgraduate Students "Applied Mathematics and Computing" PMK-2022 (Kyiv, November 16-18, 2022). An article was also published at the conference "IX International Scientific and Technical Internet Conference "Modern methods, information, software and technical support of management systems of organizational, technical and technological complexes" (Kyiv, November 25, 2022).

Structure and scope of work. The master's thesis consists of an introduction, four chapters and conclusions.

The *first section* analyzes the subject area: the problem of calculating the illumination of graphic scenes and the existing algorithms for rendering graphic scenes. The shortcomings of already existing algorithms for calculating the illumination of graphic scenes are also considered and revealed.

In the *second section*, the Forward+ rendering algorithm is described in detail, the modifications of the Forward+ rendering algorithm are described in detail and justified, and the optimization of the algorithm and improvement of the visual quality of the resulting image.

In the *third chapter*, the analysis and justification of the choice of the programming language and integrated development environments available today

is carried out. In addition, the chapter contains a detailed description of the structure and architecture of the developed application.

The *fourth chapter* examines the approaches to testing the application and describes exactly how these approaches were used in the development of the application and testing of the final results

The results of the work are presented in the conclusions.

The work is presented on XX sheets, contains a link to the list of used literary sources.

Keywords: GPU, Forward, Deferred, Forward+, GPU, C++, Vulkan, rendering, shader.

ЗМІСТ

ДОДАТКИ

Додаток А. Копія графічного матеріалу (презентація)

Додаток Б. Копія тез доповіді на тему «Додаток для відображення графічних сцен за допомогою модифікованого алгоритму Forward+ рендерінгу»

Додаток Г. Фрагменти програмного тексту реалізації алгоритму Forward+ рендерінгу

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

Deferred rendering – спосіб рендерінгу, в якому спочатку обробляється геометрія графічної сцени, а потім відбувається обчислення освітленості графічної сцени.

Forward rendering – спосіб рендерінгу, що базується на покроковому відображенні кожного об’єкта графічної сцени один за одним.

Forward+ (tiled) rendering – спосіб рендерінгу, в якому екран розбивається на рівні за розміром зони – тайли, після чого відбувається фільтрація джерел для кожного тайлу та розрахунок освітленості графічної сцени.

GPU (Graphics Processing Unit) – графічний процесор, на якому відбувається процес рендерінгу зображення.

HLSL – шейдерна мова програмування, що використовується для написання шейдерних програм.

Light culling – фільтрація джерел освітлення. Процес визначення джерел освітлення, що потрапляють на певну частину екрану.

Physically based rendering – рендерінг з використанням фізично-коректних моделей освітлення, що базуються на теорії мікроповерхонь.

Rendering – процес генерації зображення графічної сцени. Зазвичай відбувається на графічному процесорі.

View Frustum – зрізана піраміда огляду, що вміщує в себе об’єкти графічної сцени, що потрапляють на екран.

Vulkan – багатоплатформний графічний API, що розроблений Khronos Group.

Графічний API – прикладний програмний інтерфейс, за допомогою якого відбувається програмування графічного процесору.

Рендерер – програмна підсистема, що відповідає за генерацію зображення. Зазвичай, використовується в значення рендерер реального часу.

Шейдерна програма – програма, що виконується на графічному процесорі.

ВСТУП

Головним джерелом в сприйнятті інформації людиною є зір – тобто, графічне подання інформації грає для людини найважливішу роль. Тому для ефективного сприйняття інформації і постає завдання надавати інформацію в візуальному, графічному, вигляді.

З розвитком комп'ютерних технологій з'явилась можливість подавати графічну інформацію за допомогою спеціальних пристроїв – дисплеїв. Вони здатні показувати будь-який рисунок. При цьому, цей рисунок може як генеруватися в реальному часі, так і змінюватися з часом, створюючи анімовану послідовність зображень.

При цьому, для збільшення якості відображуваного зображення, потрібна велика роздільна здатність дисплеїв, а, отже, для генерації зображень в реальному часі для таких дисплеїв, потрібно більше обчислювальних спроможностей.

Також, потрібно генерувати достатню кількість зображень за одиницю часу, аби послідовність зображень сприймалася людиною як послідовна, без розривів.

Тому для генерації зображень використовуються спеціальні процесори – графічні процесори. Даний тип обчислювальних пристроїв має декілька відмінностей від центральних процесорів. Так, графічні процесори роблять акцент на кількості обчислень за одиницю часу, а не на швидкості кожного обчислення.

Традиційним для графічних процесорів є відрисовування графічних сцен – рендерінг. Даний процес являє собою велику кількість незалежних обчислень. При цьому ці обчислення проходять в декілька послідовних

стадій, що називається графічним пайплайном. Тому початково графічні процесори проектувалися з урахуванням даного факту. Тобто, були вузько спеціалізованими на відображенні графіки.

Проте, з плином часу, графічні процесори набували все більше спроможностей для виконання обчислень загального призначення. З'являлися програмовані стадії графічного пайплайну. З'явилася підтримка різноманітних структур пам'яті, а також доступ (зчитування та запис) до довільної адреси в пам'яті. Також з'явилася можливість виконувати обчислення в обхід графічного пайплайну.

Нові можливості графічних процесорів мали широке використання в тих областях людських знань, де є потреба в проведенні великої кількості незалежних один від одного обчислень. Але актуальним залишалося і традиційне для графічних процесорів використання – рендерінг.

Одночасно з появою нових можливостей графічні процесори ставали все більш потужними та швидкими. Тому з плином часом з'являлися потреби в нових алгоритмах відображення графіки, що використовують як нові можливості графічних процесорів, які б використовували всі наявні обчислювальні спроможності даних пристроїв.

З плином часу змінювалися і графічні сцени. Так, вони ставали більш деталізованими та насиченими об'єктами. А кількість джерел освітлення в графічних сценах безупинно збільшується. Це, в свою чергу, збільшує складність обчислення освітленості графічної сцени.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Освітленість графічної сцени

Кожен елемент графічної сцени має дві властивості: геометричну форму та те, як поверхня даного об’єкту відображається.

Найпоширенішим представленням об’єктів графічної сцени є використання полігонів для формування об’єму об’єктів сцени[1]. В якості полігонів використовуються багатокутники з найменшою кількістю кутів, що можуть мати площу – тобто, трикутники.

Знаючи геометричну форму об’єкта, можна визначити, яку частину екрану займає даний об’єкт, та чи потрапляє він на екран взагалі. У випадку, якщо об’єкт потрапляє на екран, потрібно визначити, як даний об’єкт в даній точці буде відображений.

Зважаючи той факт, що сучасні дисплеї складаються з пікселів, які, в свою чергу, складаються з субпікселів, кожен з яких може відображати червоний, зелений або синій колір, то проблема відображення об’єкту зводиться до визначення кольору, який буде мати екран в певній точці.

Існує багато різних підходів до визначення результуючого кольору об’єкту в певній точці. Найпростішим є задання базового кольору для всього об’єкту і використання його для кожного пікселя, на який потрапляє об’єкт графічної сцени. Такий метод зображено на рисунку 1.1. Модифікацією даного методу визначення кута між нормаллю поверхні та вектором камери. Даний метод зображено на рисунку 1.2.

Більш складним методом визначення кольору об’єкту є запровадження джерел освітлення. Даний метод відображення об’єктів зображено на рисунку 1.3.

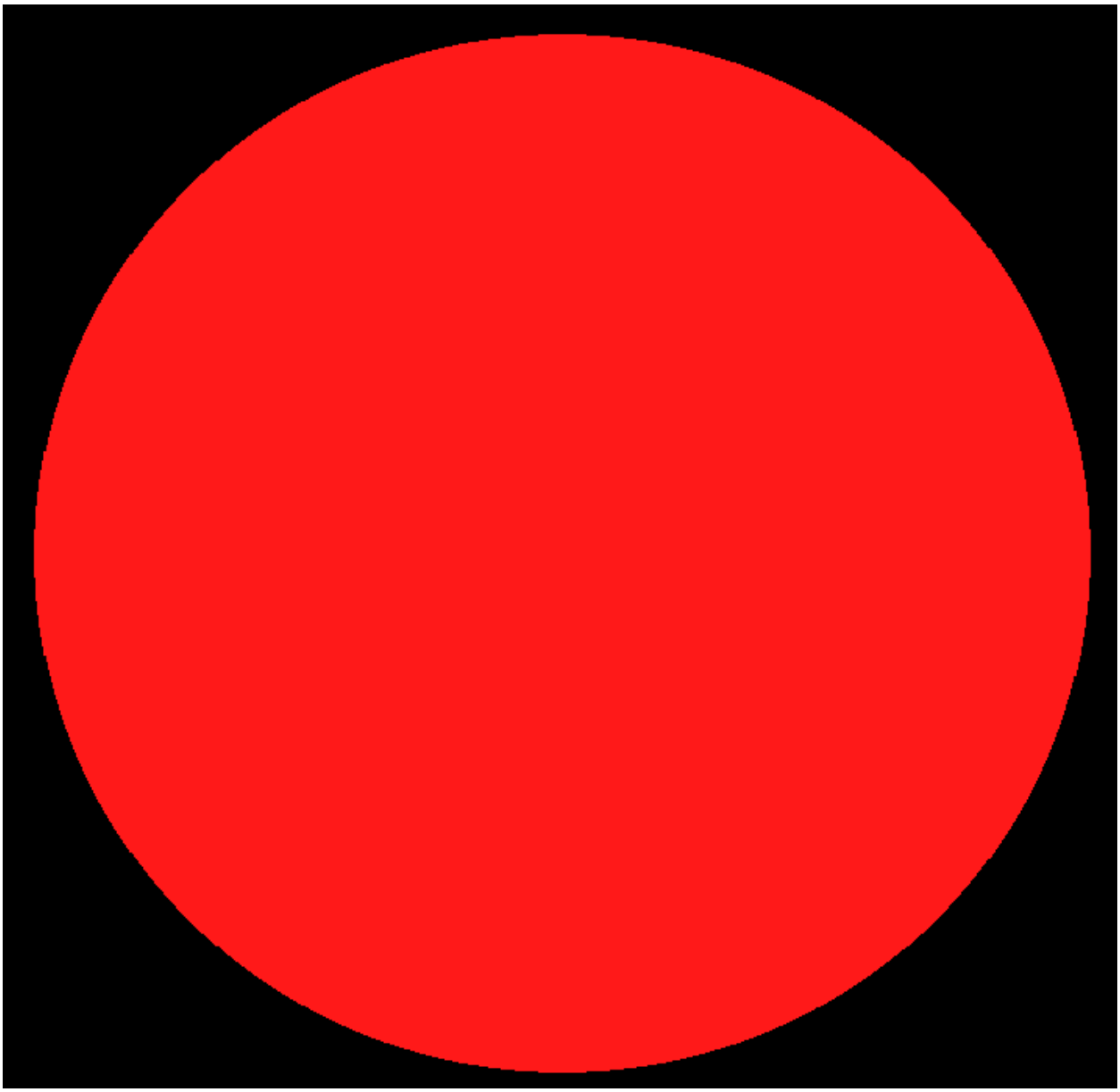

Рисунок 1.1 – Відображення об’єкту графічної сцени з використанням базового кольору

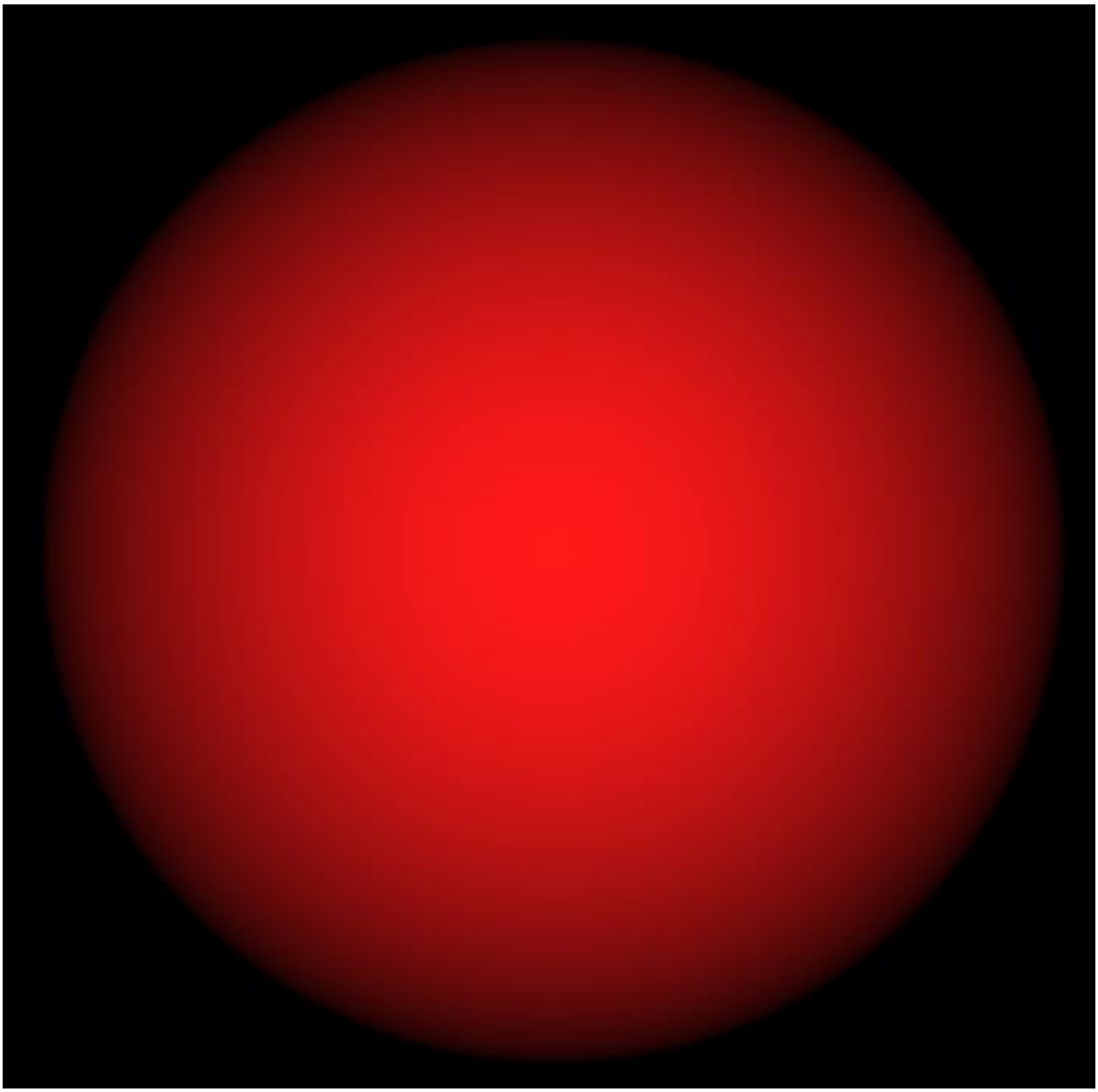

Рисунок 1.2 – Відображення об’єкту графічної сцени з визначенням кута між нормаллю поверхні та вектором камери

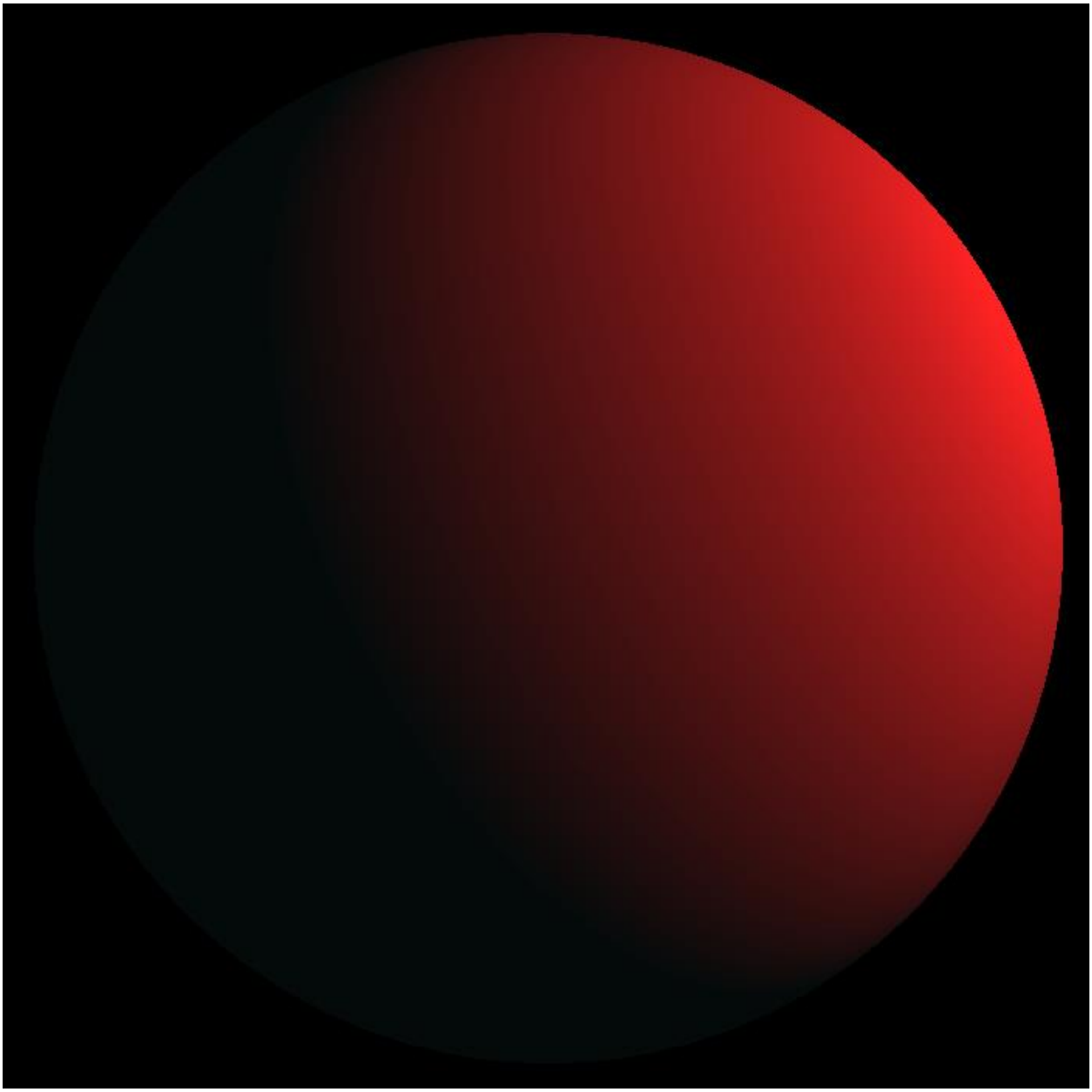

Рисунок 1.3 – Відображення об’єкту графічної сцени з використанням одного джерела освітлення

1.2. Фізично-коректне освітлення

З розвитком комп’ютерних технологій з’явилася потреба рендерити об’єкти з максимальним наближенням до реальності. Даний підхід до генерації зображень отримав назву фізично-коректного рендерінгу.

Суть даного підходу полягає в дотриманні законів фізики, зокрема закону збереження енергії, та використанні моделей відображення об’єктів, що використовують реально існуючі фізичні властивості матеріалів.

Вважається, що фізично коректний рендерінг базується на моделях, що засновані на понятті мікрогеометрії[2]. Кожна видима точка поверхні складається з багатьох нормалей мікроповерхонь, які відбивають світло в різні напрямки. Зважаючи на те, що орієнтація мікроповерхонь є випадковою, то їх розподіл представляється статистично. Для більшості поверхонь розподіл мікроповерхонь є неперервним, з піком, що збігається з нормаллю макроповерхні. «Ширина» даного розподілу визначається шорсткістю поверхні. Чим поверхня більш шорстка, тим «ширше» буде розподіл, що означає, що нормалі мікроповерхонь направлені в різні сторони. Приклад освітлення за допомогою фізично коректної моделі зображено на рисунку 1.4.

Використання фізично-коректних моделей освітлення дозволяє наблизити результуюче зображення до реальності. Але обчислення за допомогою методу фізично-коректного освітлення з точки зору обчислень набагато важче, ніж будь-яким іншим, представленим вище, методом.

Порівняння швидкодії різних методів освітлення об’єктів наведено в таблиці 1. З цих даних добре видно, що використання фізично-коректних методів освітлення накладає певні обмеження. Головним обмеженням є кількість джерел освітлення на кадр. Дане обмеження є неприємним, адже художники по освітленню, що займаються налаштуванням освітлення графічних сцен, мають значно менше можливостей, аніж якби даного обмеження не існувало.

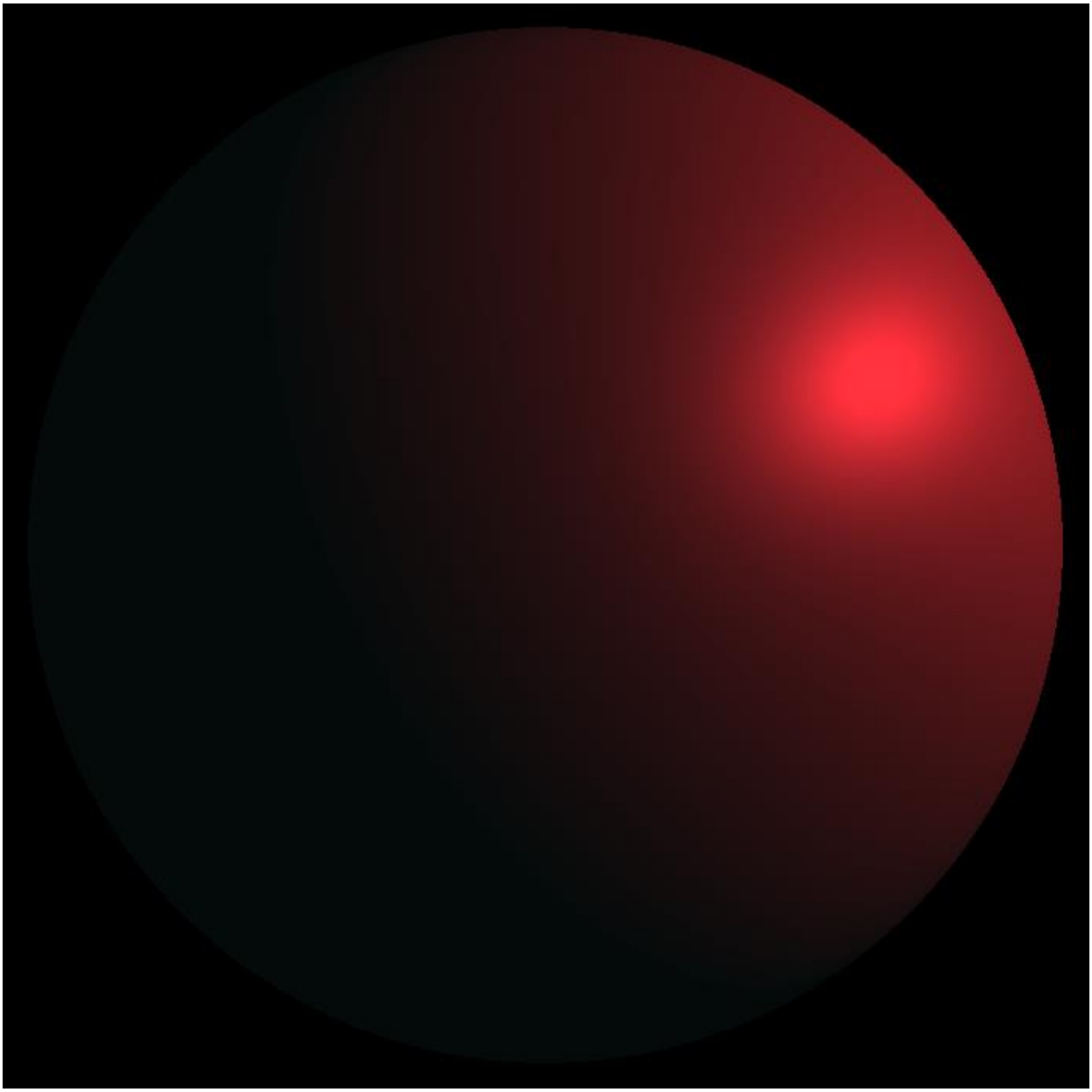

Рисунок 1.4 – Освітлення об’єкту за допомогою фізично-коректної моделі

Таблиця 1.1 – Порівняльна таблиця часу генерацій одного кадру за допомогою різних методів освітлення в утиліті Shadertoy

Метод освітлення	**Час генерації одного кадру, мс**
Використання базового кольору	40

Метод освітлення	Час генерації одного кадру, мс
Визначенням кута між нормаллю поверхні та вектором камери	51.5
З використанням одного джерела освітлення	70.6
Фізично-коректна модель з трьома джерелами освітлення	114.9

Саме тому актуальним постає питання використання ефективних алгоритмів відображення об’єктів. Розглянемо деякі з них.

1.3. Аналіз відомих алгоритмів розрахунку освітленості графічних сцен

Історично найбільш поширеними алгоритмами розрахунку освітленості графічних сцен є наступні: Forward, Deferred, Forward+. Кожен наступний алгоритм має переваги відносно попереднього. Так, найпершим і найпростішим алгоритмом є Forward рендерінг.

1.3.1. Алгоритм Forward рендерінгу

Алгоритм Forward рендерінгу можна описати наступним чином: для кожного об’єкту сцени розрахувати освітленість відносно кожного джерела освітлення.

Окрема шейдерна програма[1] буде відповідати повністю за відрисовку об’єкта, починаючи з вершинного шейдера і до фрагментного шейдера.

Джерела освітлення, в такому випадку, представляються аналітично: місце в просторі та описом обмежуючого простору, в якому дане джерело освітлення знаходиться.

Таким чином, в фрагментному шейдері кожного об’єкту будуть проводитись розрахунки з кожним джерелом освітлення графічної сцени, чи потрапляє дане джерело освітлення на певну точку даного об’єкту.

Результатом фрагментного шейдеру буде результуюче зображення освітленості графічної сцени. Послідовність операцій Forward рендерінгу зображено на рисунку 1.5.

Але зі збільшенням кількості джерел освітлення, збільшується і час виконання Forward рендерінгу, що є його головним недоліком.

Ще одним недоліком алгоритму Forward рендерінгу є накладання різних об’єктів графічної сцени на одні й ті самі пікселі екрану. Це відбувається в тому випадку, коли два об’єкти графічної сцени розташовані на різній відстані від камери, але вздовж умовної прямої. Спочатку відбувається розрахунок обчислення освітленості для одного об’єкту, а потім це значення відкидається, і відбувається розрахунок освітленості для того ж самого пікселя, але іншого об’єкту сцени.

Блок-схема алгоритму Forward рендерінгу зображена на рисунку 1.6.

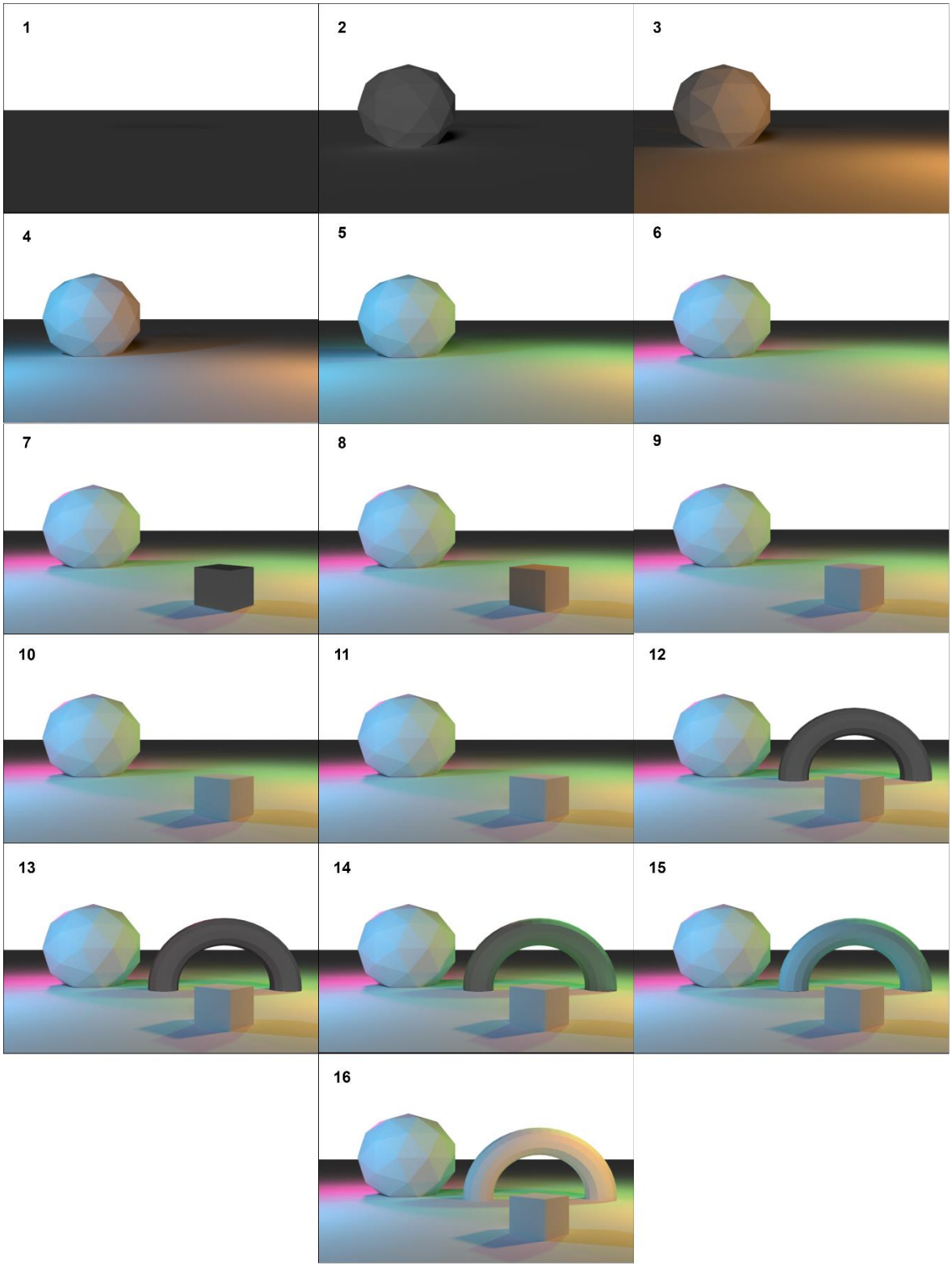

Рисунок 1.5 – Послідовність операцій Forward рендерінгу

Рисунок 1.6 – Блок-схема алгоритму Forward рендерінгу

1.3.2. Алгоритм Deferred рендерінгу

Алгоритм Deferred рендерінгу має дві ключові відмінності від Forward рендерінгу.

Першою відмінністю є те, що результатом фрагментного шейдеру є набір значень, таких як: шорсткість (roughness), металевість (metalness), базовий колір (base color), нормаль поверхні (normal), світимість поверхні (emissive) та глибина об’єкту відносно камери (depth).

Дані значення записуються в окремі текстури. Ці текстури називаються GBuffer. Приклад GBuffer зображено на рисунку 1.7.

Після формування GBuffer графічної сцени, відбувається обчислення освітленості графічної сцени з використанням значень з текстур GBuffer.

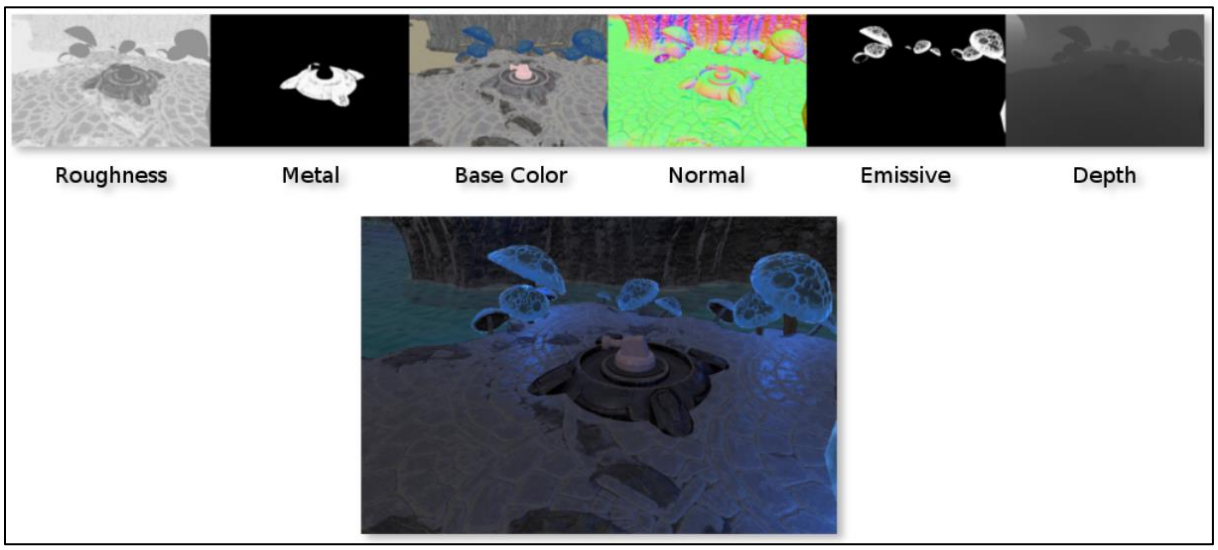

Рисунок 1.7 – Вивід зображення в декілька буферів

Другою відмінністю алгоритму Deferred рендерінгу є те, що джерело освітлення в даному випадку представляється у вигляді меша, який рендериться як звичайний об’єкт графічної сцени. Для кожного пікселя, на який потрапляє даний меш, відбувається розрахунок освітленості.

Дана оптимізація дозволяє прибрати розрахунок освітленості для тих пікселів, на які не потрапляють джерела освітлення. Покрокова робота алгоритму Deferred рендерінгу зображена на рисунку 1.8.

Також, даний алгоритм вирішує проблеми накладання різних об’єктів графічної сцени на один і той самий піксель екрану. Тобто, для кожного пікселя розрахунок освітленості відбувається лише один раз. Блок-схема роботи алгоритму Deferred рендерінгу зображений на рисунку 1.9.

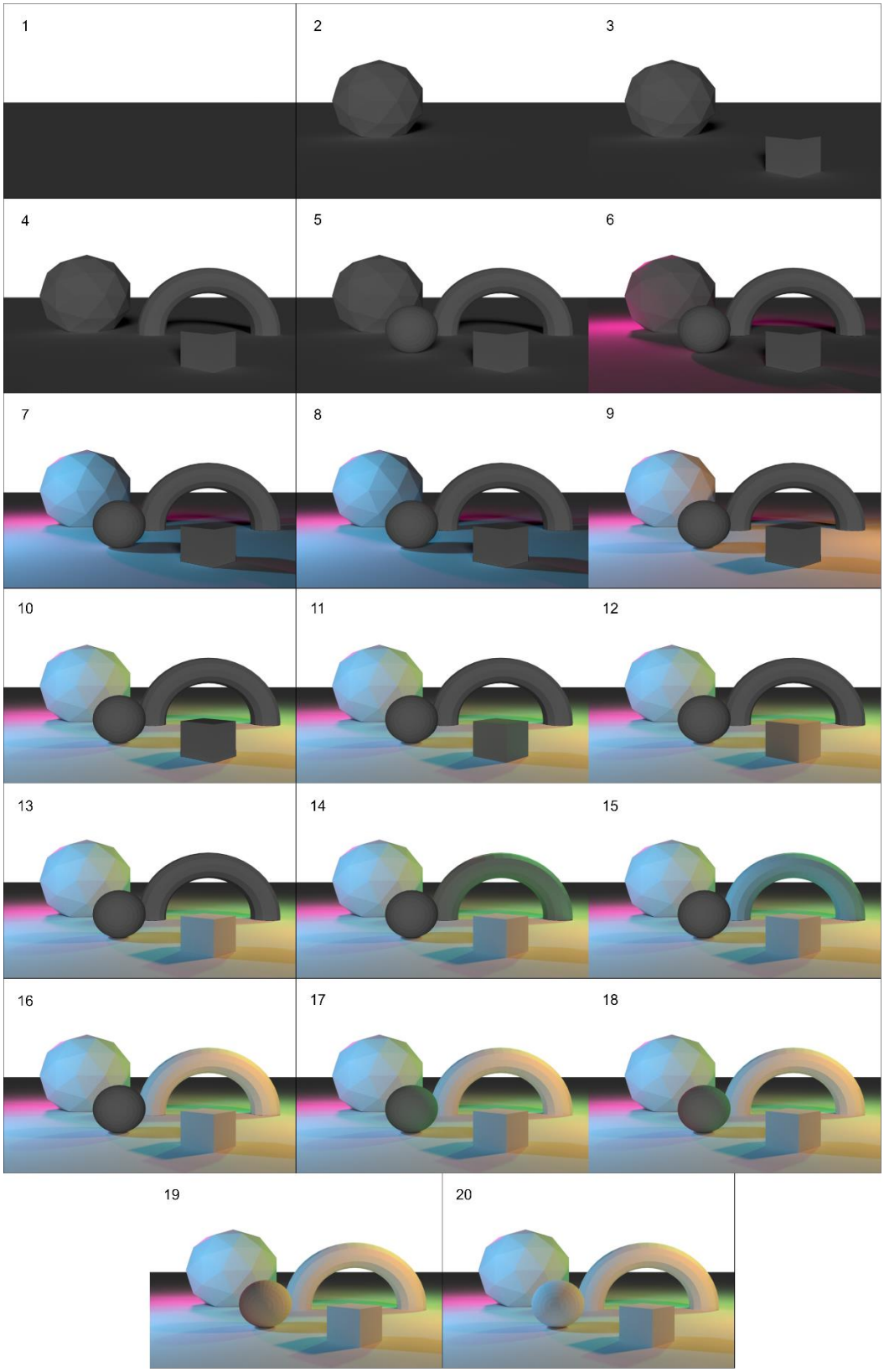

Рисунок 1.8 – Покрокова робота алгоритму Deferred рендерінгу

Рисунок 1.9 – Блок-схема алгоритму Deferred рендерінгу

1.3.3. Алгоритм Forward+ рендерінгу

Алгоритм Forward+ рендерінгу є хронологічно наймолодшим серед трьох. Поява даного алгоритму тісно пов’язана з графічним API DirectX 11, в рамках якого було представлено новий тип шейдерів – compute шейдер. Це дозволило використовувати графічні процесори для обчислень загального призначення. Саме compute шейдери використовуються в алгоритму Forward+ рендерінгу.

Ідея Forward+ рендерінгу наступна. Екран розбивається на множину тайлів та створюється спеціальний буфер, в якому зберігаються, які з джерел освітлення потрапляють на певний тайл екрану. Далі під час обчислень освітленості графічної сцени до уваги беруться лише ті джерела освітлення, що знаходяться у відповідному буфері того тайлу, до якого належить відповідний піксель. Послідовність фільтрації джерел освітлення зображена на рисунку 1.10.

Алгоритм Forward+ рендерінгу також схожий на алгоритм Deferred рендерінгу тим, що спочатку формується GBuffer. Потім відбувається фільтрація джерел освітлення для кожного тайлу екрану. Останнім етапом алгоритму є обчислення освітленості для кожного пікселя тайлу, де до уваги беруться лише ті джерела освітлення, що відносяться до даного тайлу.

Блок-схема алгоритму Forward+ рендерінгу зображена на рисунку 1.11.

Більш детально алгоритм Forward+ рендерінгу буде розглянуто нижче.

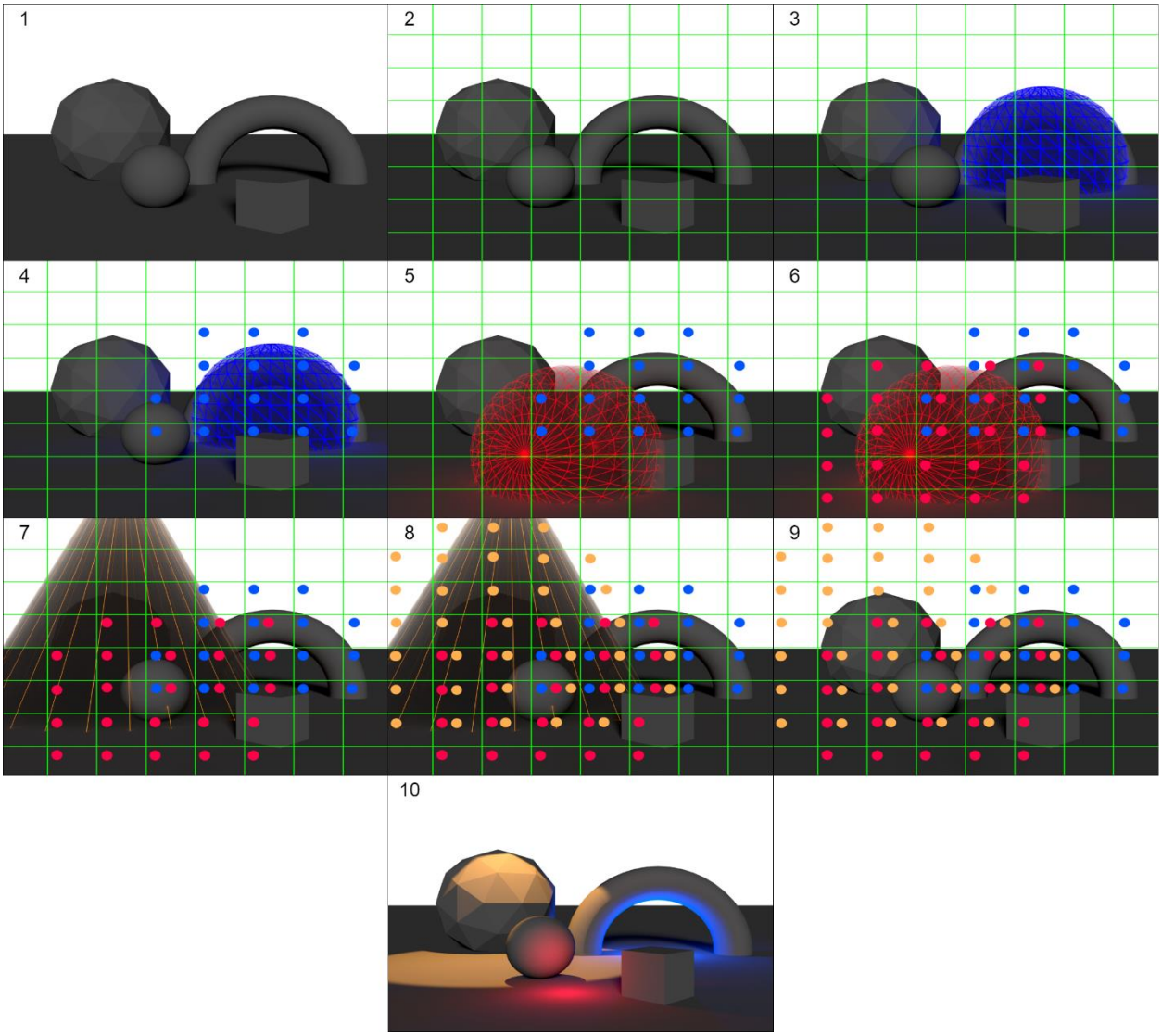

Рисунок 1.10 – Покрокова робота алгоритму фільтрації джерел освітлення

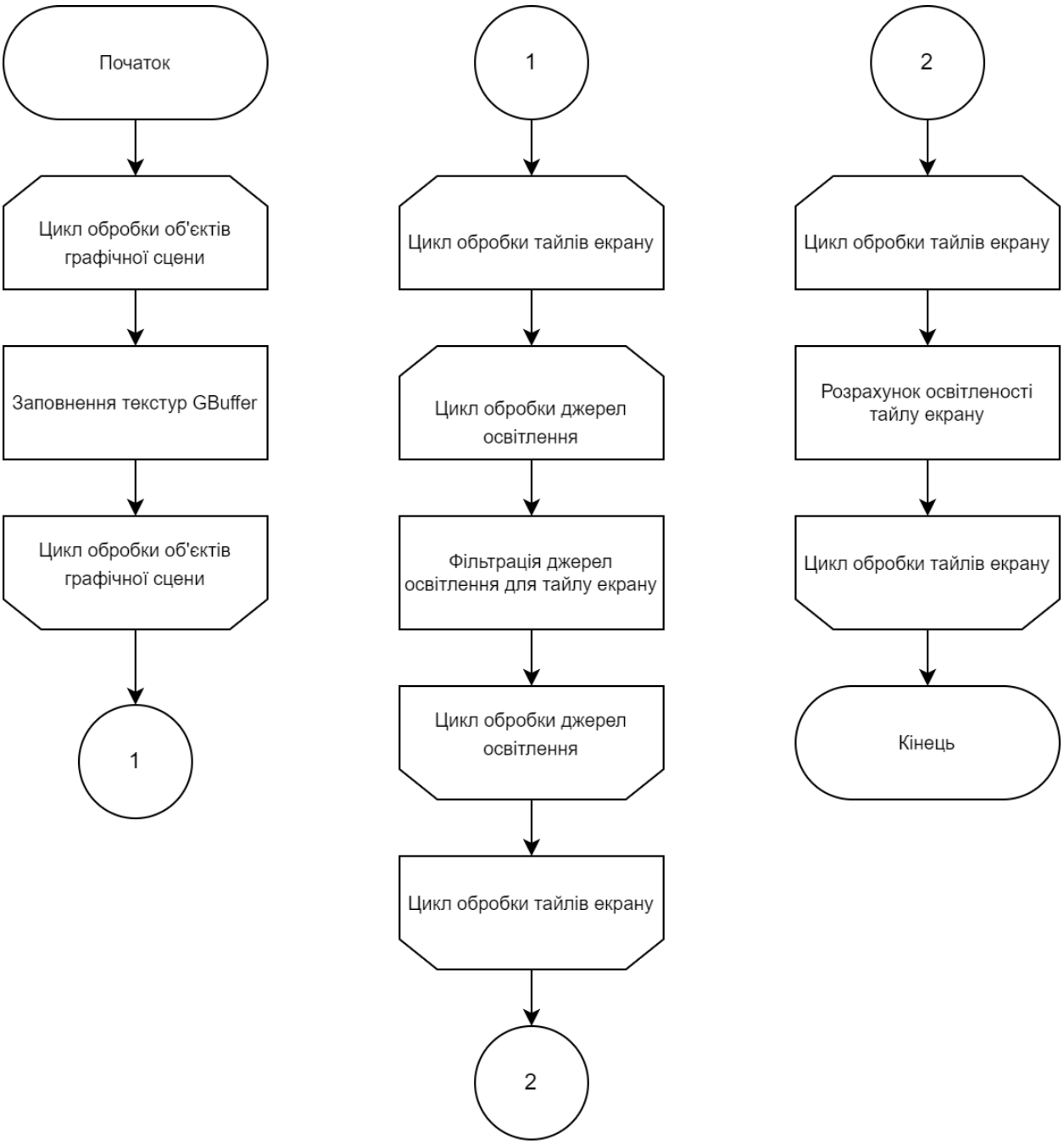

Рисунок 1.11 – Блок-схема алгоритму Forward+ рендерінгу

Висновки до розділу

В даному розділі було розглянуто проблему відображення графічної сцени. А саме, процес обчислення освітленості для кожного об’єкту графічної сцени.

Було зазначено важливість коректного відображення графічної сцени з використання фізично-коректних моделей освітлення, адже від цього напряму залежить сприйняття згенерованого зображення користувачем.

Розглянуто найбільш поширені алгоритми обчислення освітленості графічної сцени – алгоритми Forward, Deferred та Forward+ рендерінгу. Було наведено особливості кожного з них. Також було проведено аналіз переваг та недоліків наведених алгоритмів та порівняно алгоритми між собою.

2. ПОЛІПШЕННЯ АЛГОРИТМУ FORWARD+ РЕНДЕРІНГУ

2.1. Детальний опис роботи алгоритму

Алгоритм Forward+ рендерінгу можна розділити на декілька етапів. Розглянемо кожен з них окремо.

2.1.1. Формування карти глибини

Першим етапом роботи алгоритму Forward+ рендерінгу є формування карти глибини.

Даний етап полягає в обробці геометрії кожного об’єкту графічної сцени без обчислень освітленості даного об’єкту. Таким чином, запис результатів відбувається лише в карту глибини. Формування даної карти дозволяє зрозуміти, в яких пікселях екрану на якій відстані від камери знаходиться об’єкт. Приклад карти глибини зображено на рисунку 2.1.

Карта глибини представляє собою нормалізовані значення від 0 до 1, де 0 відповідає near clip plane, а 1 – far clip plane обрізаної піраміди огляду.

Дана карта дозволяє в наступних етапах більш точно відфільтровувати джерела освітлення.

Рисунок 2.1 – Приклад карти глибини

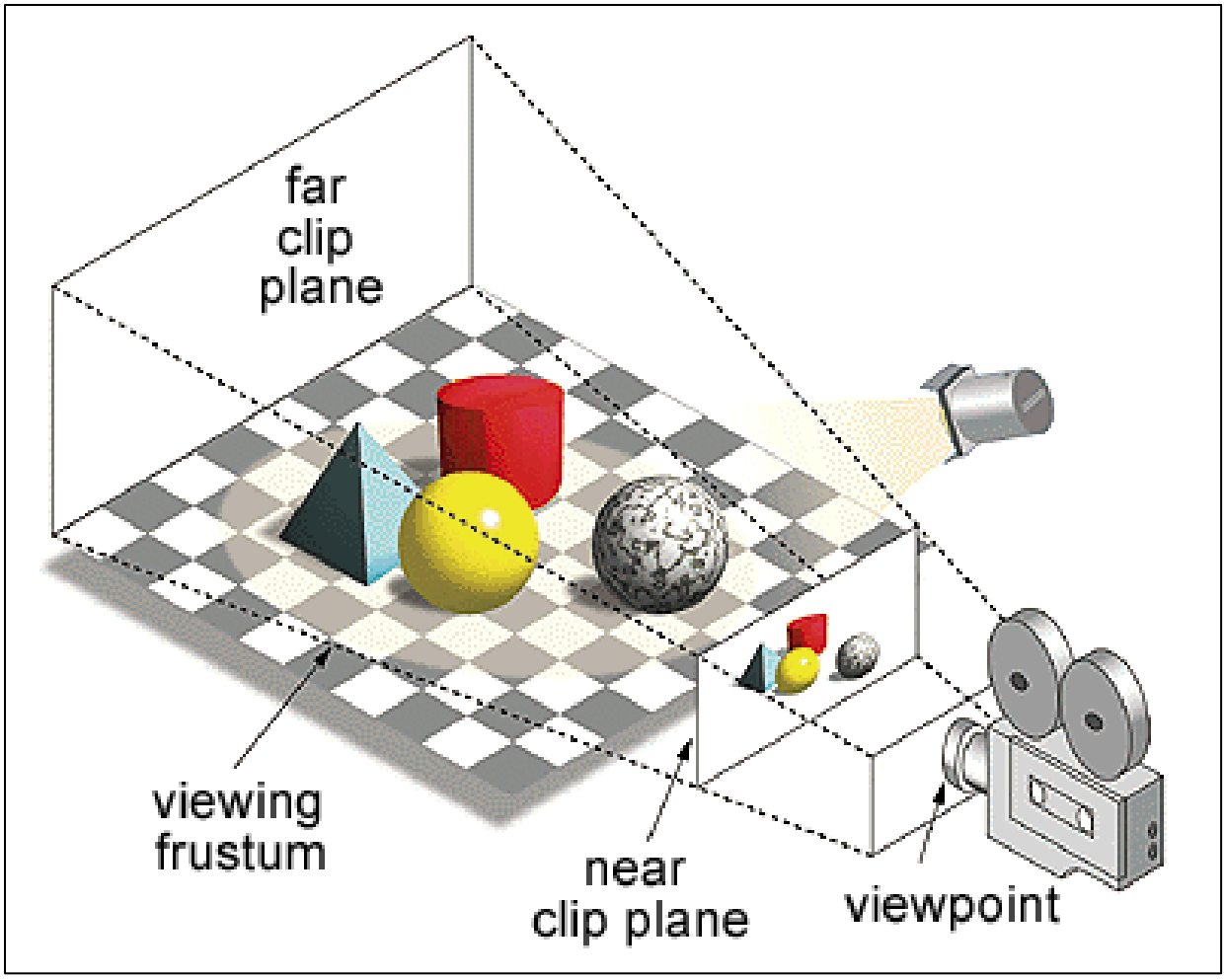

Рисунок 2.2 – Обрізана піраміда огляду

2.1.2. Розбиття екрану на тайли

Наступним етапом роботи є розбиття екрану на тайли. Тайл – сегмент екрану з шириною в декілька пікселів та висотою в декілька пікселів. Розмір тайлів потрібно підбирати таким чином, щоб їх кількість була цілим числом. В такий спосіб досягається максимальна ефективність алгоритму. Приклад розбиття екрану на тайли зображено на рисунку 2.3. При цьому, тайл має не лише ширину та висоту, а й глибину, що визначається за допомогою карти глибини.

2.1.3. Фільтрація джерел освітлення

На етапі фільтрації джерел освітлення використовується вся та інформація, що була розрахована на попередніх стадіях.

Наступним кроком є визначення мінімальної та максимальної глибини для кожного тайлу. Дана інформація може бути розрахована викликом окремої шейдерної програми та записана у відповідну карту – карту мінімальної та максимальної глибини тайлу. Приклад такої карти зображено на рисунку 2.4.

Зазвичай, фільтрація джерел освітлення відбувається за допомогою AABB.

AABB (англ. Axis Aligned Bounding Box) – обмежуючий паралелепіпед, сторони якого паралельні до осей координат. Дана структура даних зберігає мінімальне та максимальне значення глобальних координат об’єкту. Приклад AABB зображено на рисунку 2.5.

AABB є спрощенням реальної геометрії об’єкту, але її використання є досить ефективним під час фільтрації джерел освітлення. Адже, якщо AABB не потрапляє в тайл екрану, то і сам об’єкт не може потрапити в даний тайл, адже об’єкт повністю міститься в даному паралелепіпеді. А розрахунок перетину між тайлом екрану та AABB є набагато менш витратним з точки зору обчислень.

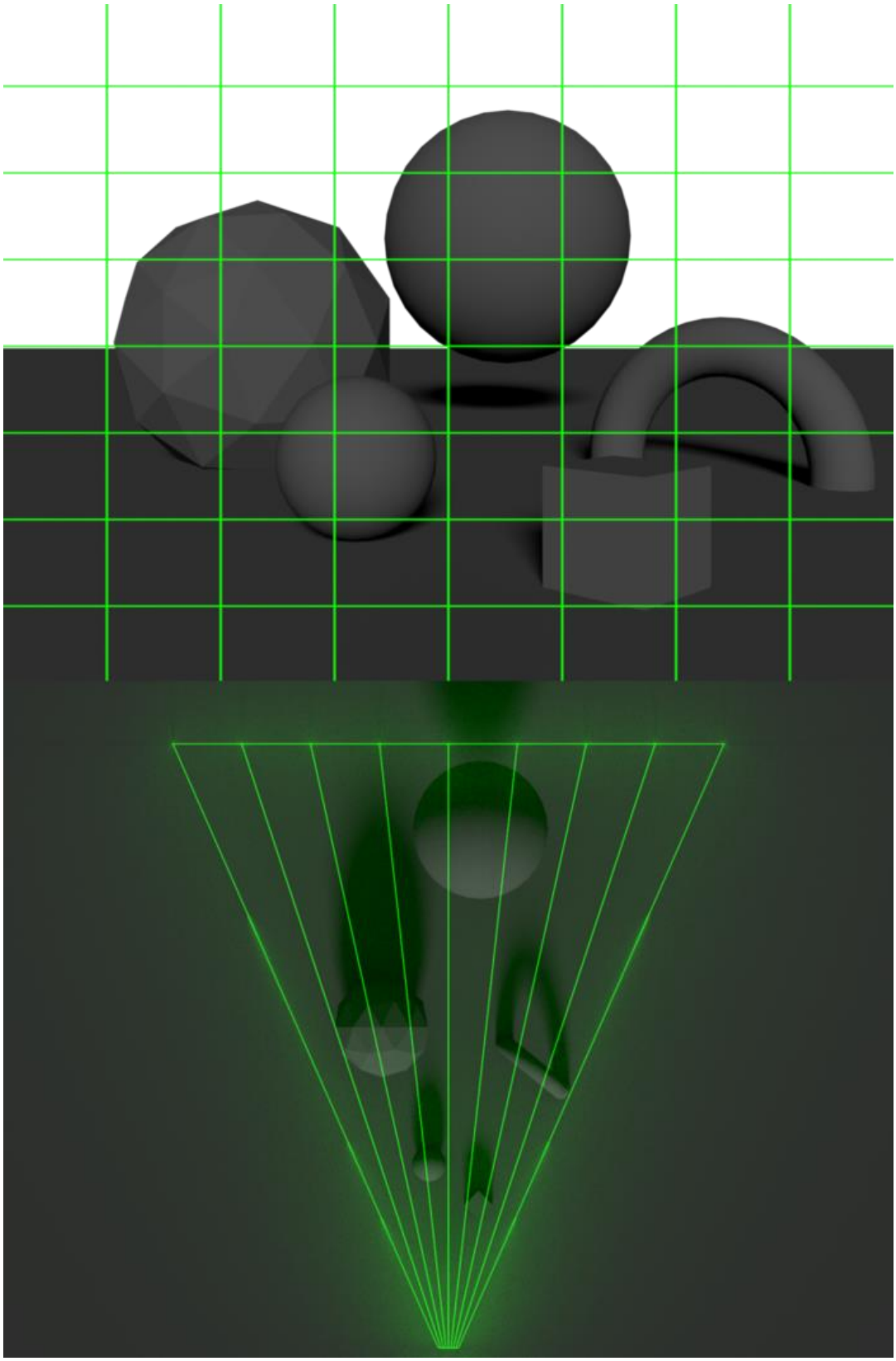

Рисунок 2.3 – Приклад розбиття екрану на тайли

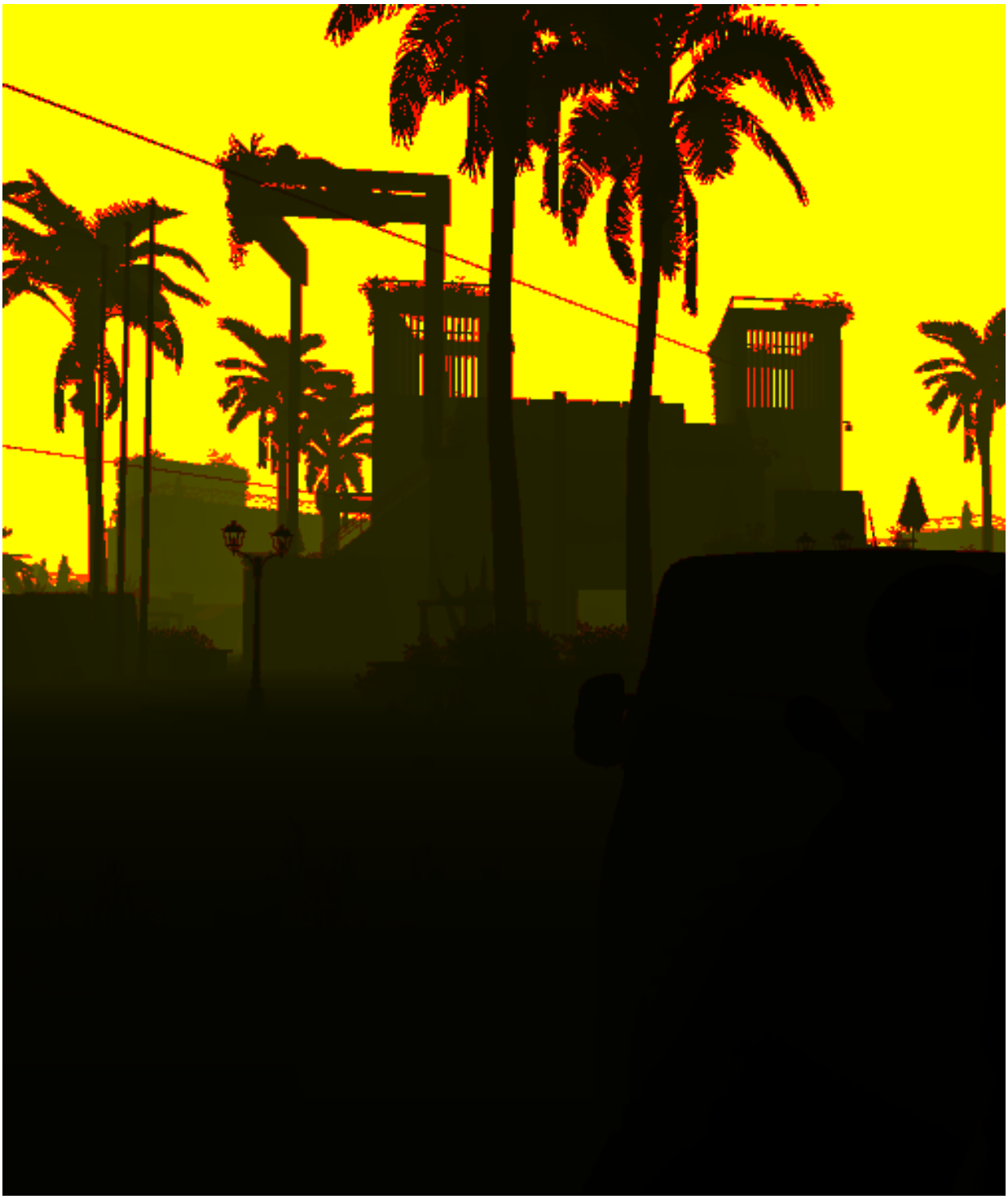

Рисунок 2.4 – Приклад карти глибини з мінімальним та максимальним значеннями

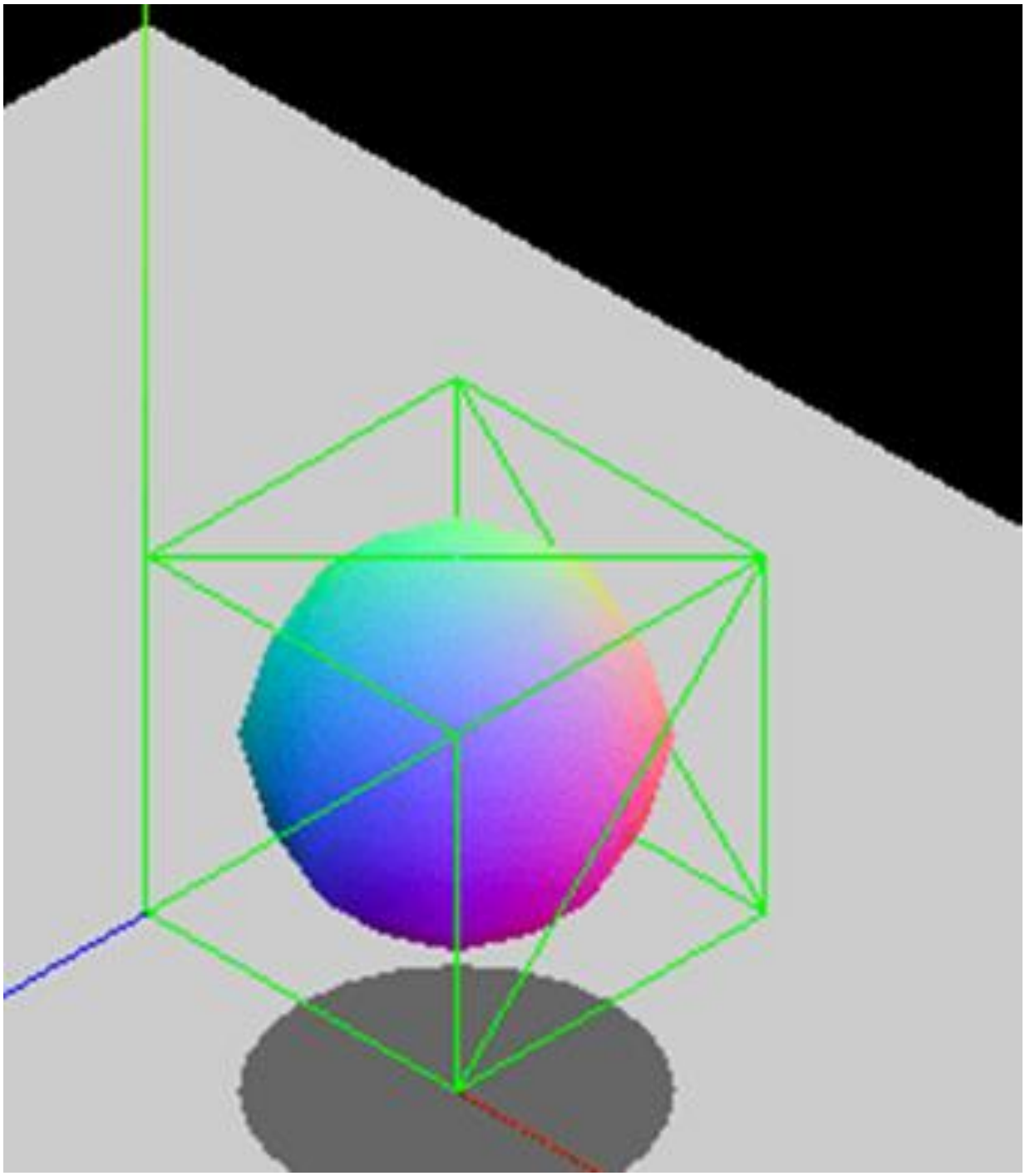

Рисунок 2.5 – Приклад AABB для сфери

AABB може бути застосовано і для джерел освітлення, адже кожне джерело освітлення в рендерерах реального часу має межі [1]. Таким чином,

фільтрація джерел освітлення спрощується до визначення, чи перетинається AABB джерела освітлення з тайлом екрану.

Але даний алгоритм можна зробити більш ефективним, якщо прийняти до уваги те, що ми знаємо мінімальну глибину та максимальну глибину графічної сцени в кожному конкретному пікселі, а, значить, достатньо обчислювати перетинання AABB джерела освітлення лише з сегментом даного тайлу.

Якщо тайл екрану та AABB джерела освітлення не перетинаються, то для даного тайлу це джерело відкидається. Якщо перетинаються, то індекс даного джерела освітлення записується у відповідний буфер.

2.1.4. Розрахунок освітленості графічної сцени

Завершаючим етапом роботи алгоритму Forward+ є розрахунок освітленості графічної сцени.

Для кожного об’єкту графічної сцени відбувається виклик на відрисовку. В загальному випадку рендерінг відбувається в одну текстуру, в яку записується фінальний колір відповідного пікселя екрану.

При цьому, розрахунок освітленості графічної сцени займає мінімально можливий час, адже завдяки вже сформованій карті глибини розрахунок кожного пікселя відбувається лише один раз. Бо, наприклад, об’єкти графічної сцени, які могли б попадати на піксель екрану, відкидаються, адже карта глибини містить значення глибини, яке ближче до камери, аніж те, що має даний об’єкт. Тому розрахунок освітленості для кожного пікселя екрану відбувається лише для тієї геометрії, що згенерувала відповідне значення в карті глибини.

2.2. Покращення точності алгоритму фільтрації джерел освітлення

Вище описаний алгоритм Forward+ може бути покращений з точку зору ефективності виконання.

Таким покращенням є збільшення кількості джерел освітлення, що не потрапляють на тайл екрану, але не можуть бути відфільтровані вище описаним способом[5].

Цьому є дві причини. Першою причиною такого явище є непостійність є той факт, що AABB джерела освітлення надає лише приблизні розміри даного джерела. На рисунку 2.6 зображено джерело освітлення, що має форму сфери, при цьому, AABB має форму рівностороннього паралелепіпеду. То ж ближче до вершин даного AABB джерело освітлення не потрапляє, вище описаний алгоритм фільтрації джерел освітлення буде вважати, що дане джерело освітлення потрапляє на тайл екрану.

Другою причиною є те, що джерело освітлення має потрапляти саме на геометрію об’єкту графічної сцени, тобто, перетинатися з самим об’єктом. В свою чергу, вище описаний алгоритм ігнорує даний факт.

Поглянемо на тайл на рисунку 2.7. Враховуючи мінімальне та максимальне значення глибини для кожного тайлу, їх потрібно розбити на сегменти. Сегментом є саб-тайл, що не змінює ширину та висоту тайлу, але має зменшену глибину. Приклад сегментів зображено на рисунку 2.8.

Наступним кроком є визначення маски глиби тайлу. Маска глибини тайлу представляє собою число, де кожному біту даного числа відповідає сегмент тайлу. Якщо в будь-якому з пікселів значення глибини припадає на даний сегмент, то відповідний біт маски глибини тайлу встановлюється в 1. Якщо жоден піксель не має значення, глибини, що потрапляє на певний тайл,

то відповідний біт маски глибини встановлюється в 0. Перший та останній біти маски глибини встановлюються за замовчуванням в 1.

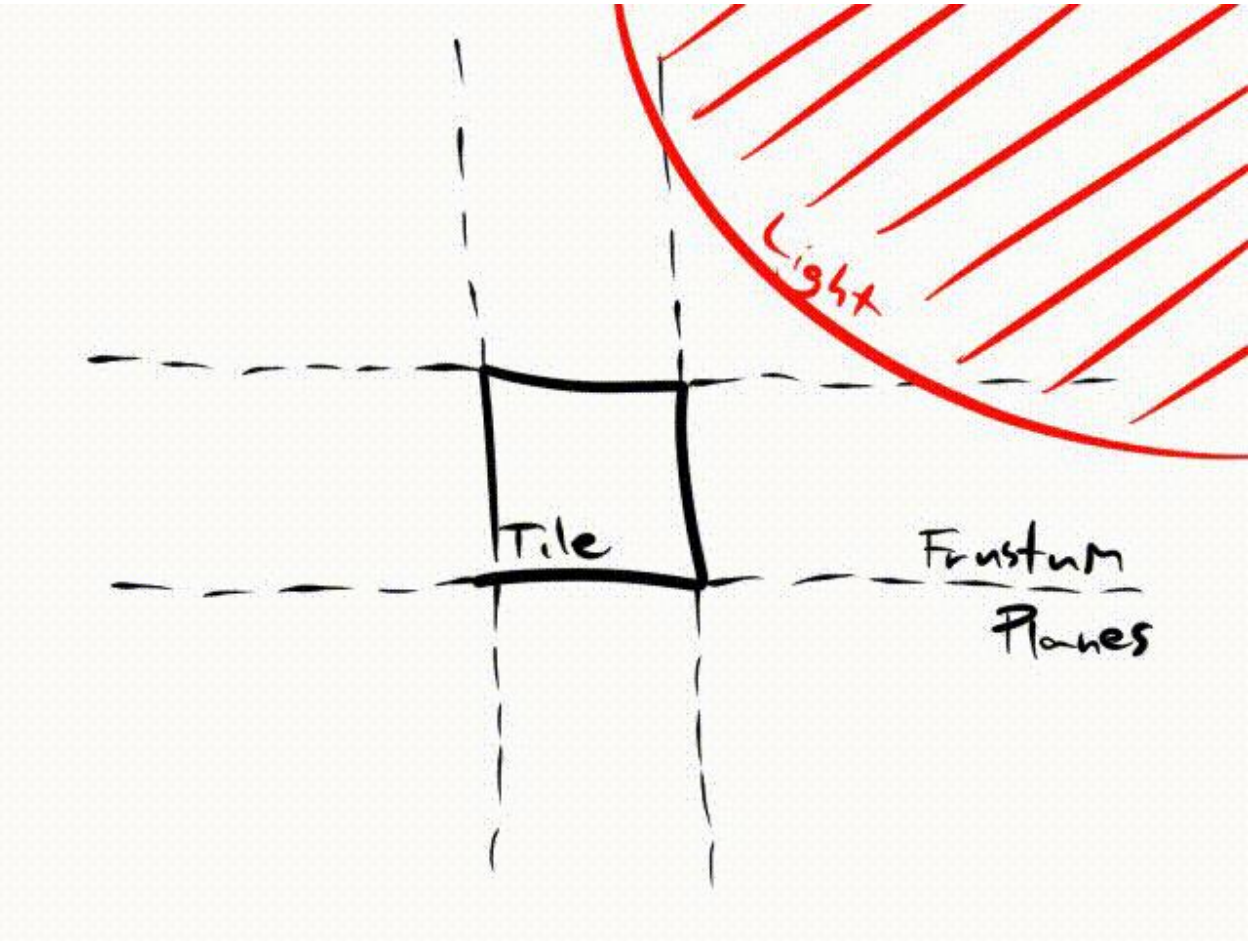

Рисунок 2.6 – Приклад джерела освітлення, що не потрапляє на тайл екрану

Тобто, Маска глибини представляє собою число, де біт з індексом *i* відповідає сегменту з індексом *i*. Якщо даний біт має значення 1, то принаймні один піксель тайлу має глибину, що потрапляє в даний сегмент.

Наступним кроком в роботі алгоритму є визначення маски глибини джерела освітлення.

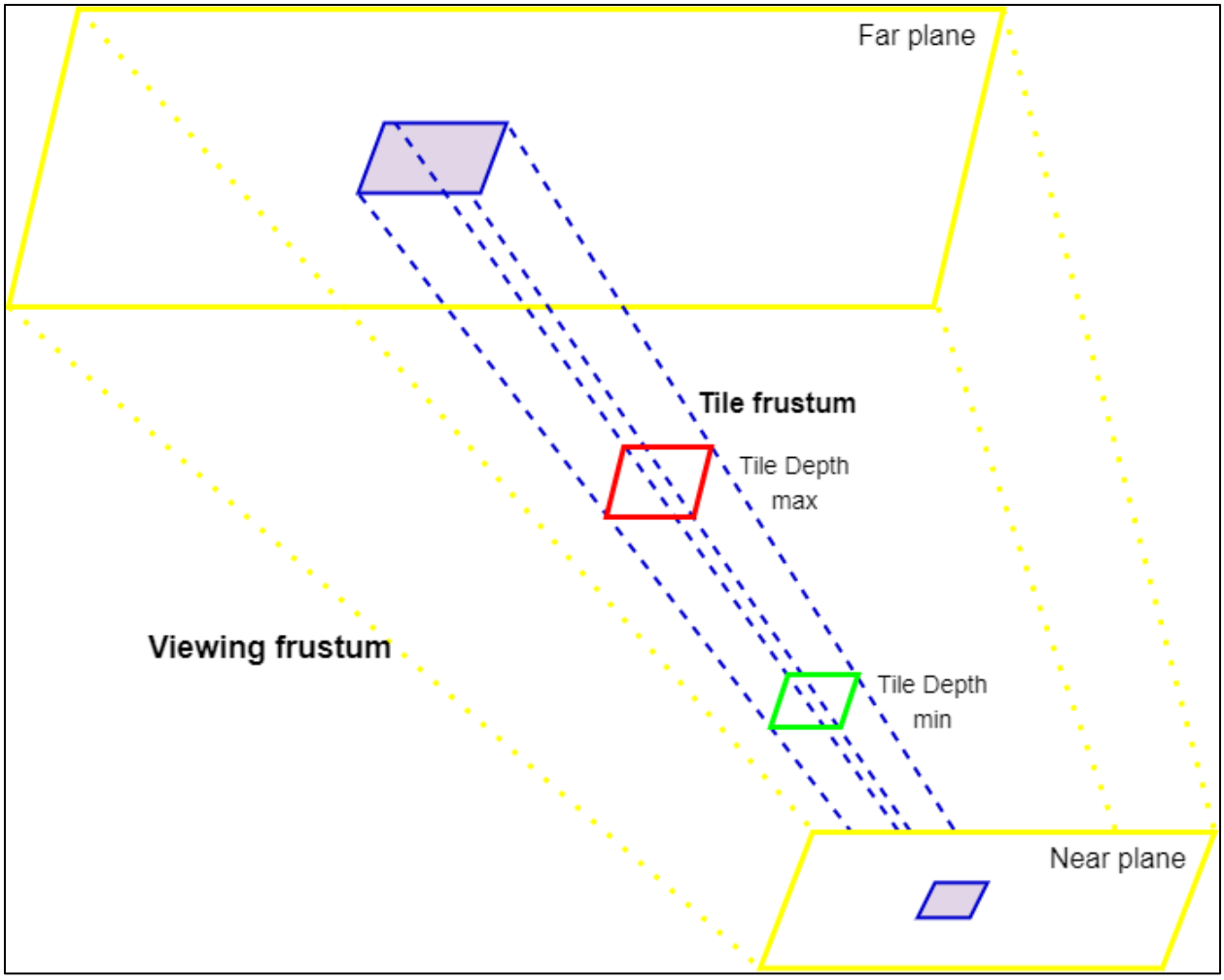

Рисунок 2.7 – Тайл екрану, мінімальна та максимальна глибина тайлу екрану

Маска глибини джерела освітлення визначається схожим чином. Для кожного пікселя даного тайлу визначається, чи потрапляє джерело освітлення в даний тайл, і, якщо потрапляє, то в які сегменти. Відповідно, для кожного пікселя екрану формується маска глибини джерела освітлення. Потім за допомогою операції Or масок глибини джерела освітлення всіх пікселів екрану формується результуюча маска глибини джерела освітлення для всього тайлу.

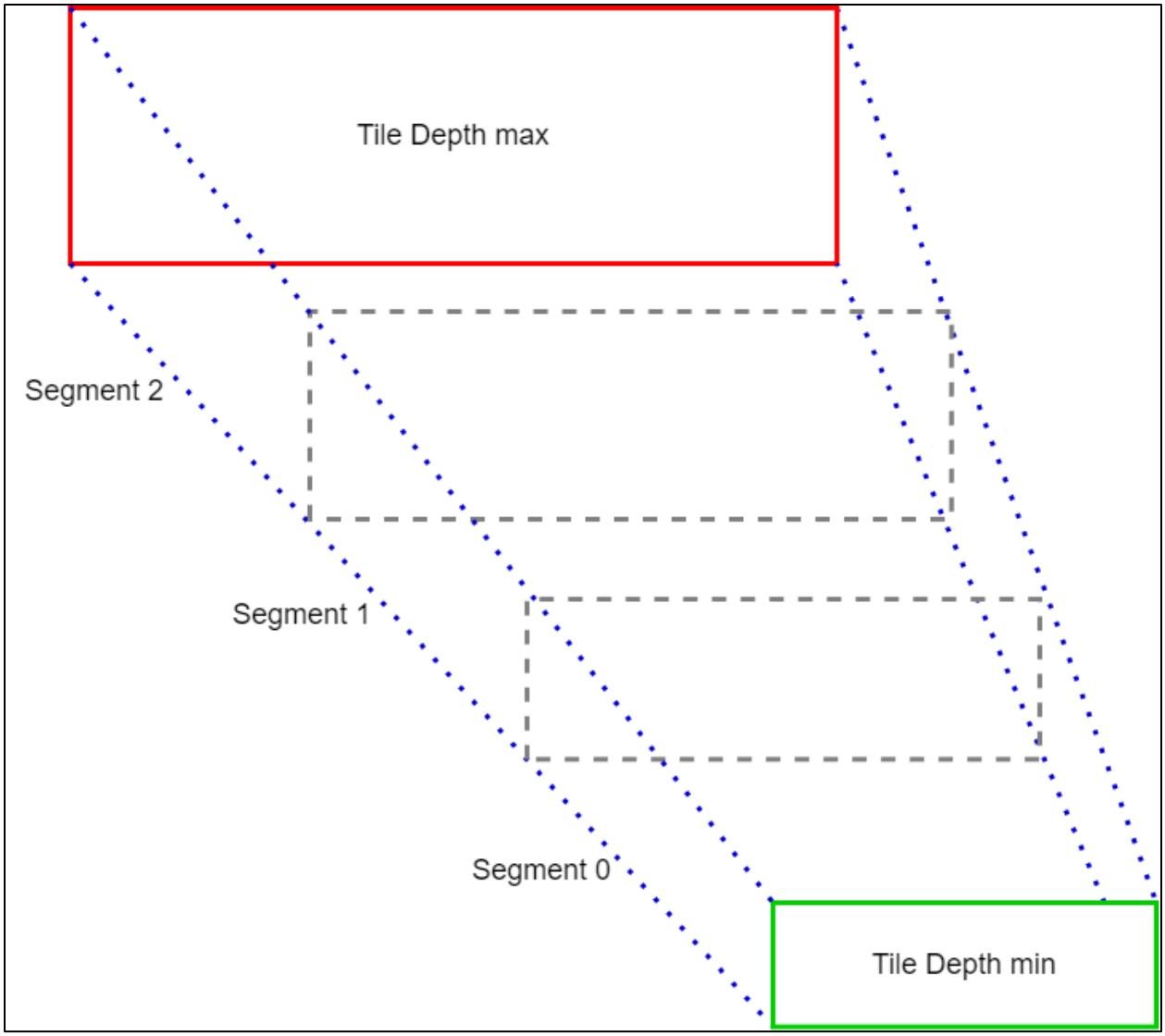

Рисунок 2.8 – Розбиття тайлу на сегменти

Після цього відбувається порівняння масок глибини тайлу та джерела освітлення за допомогою операції And. Якщо результат ненульовий, то дане джерело перетинається з тайлом екрану, і записується у відповідний буфер. Якщо результат операції And нуль, то дане джерело не потрапляє в даний тайл екрану.

Приклад джерела освітлення, що не потрапляє на тайл екрану, зображено на рисунку 2.9. На цьому рисунку представлений горизонтальний

зріз тайлу від мінімального (знизу) до максимального (зверху) значення глибини. Справжні значення глибини для кожного пікселя позначені синім кольором. Джерело освітлення має жовтий колір.

Визначимо, чи потрібно враховувати дане джерело під час розрахунку освітлення для даного тайлу.

Розрахунок маски глибини даного тайлу дає результат 10001011, адже пікселі мають значення глибини, що потрапляє в сегменти з індексами 0, 1, 3 та 7. Маска глибини джерела освітлення, що позначений жовтим кольором, буде 01110000, бо джерело освітлення потрапляє в сегменти з індексами 4, 5 та 6. Наступним кроком є побітова операція І між цими масками: 10001011 And 01110000 = 00000000. Результатом є нуль, отже, дане джерело освітлення хоч і потрапляє в даний тайл, проте, не потрапляє в жоден з пікселів тайлу, а, значить, має бути відкинуте під час розрахунку освітлення для цього тайлу.

Головною перевагою даного алгоритму є можливість його паралельного виконання для кожного пікселя окремо, виконуючи операції AND та OR атомарно, що може суттєво зменшити час виконання даного алгоритму.

Також даний алгоритм дозволяє з більшою точністю відфільтровувати джерела освітлення, що потрапляють в тайл, проте, не потрапляють на жоден з пікселів. Це дозволяє зменшити об’єм обчислень під час розрахунку освітленості сцени.

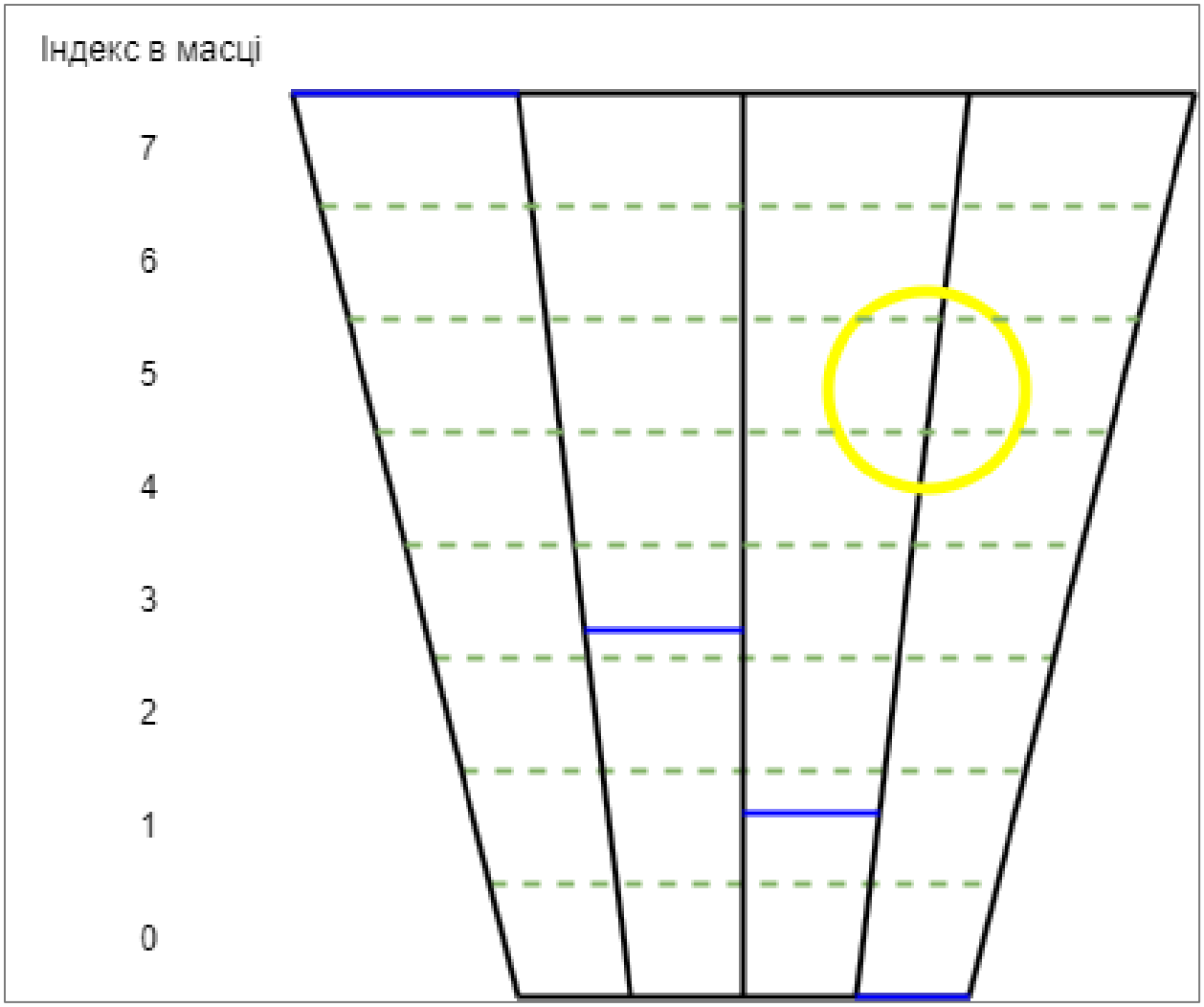

Рисунок 2.9 – Приклад джерела освітлення, що не потрапляє на тайл екрану

2.3. Відображення напівпрозорих тіней

Відображення напівпрозорих тіней є ще одним покращенням алгоритму Forward+ рендерінгу.

Напівпрозорі тіні формуються напівпрозорими об’єктами. Світло, що проходить через такі, не повністю поглинається такими об’єктами, тому за такими об’єктами можна спостерігати кольорові напівпрозорі тіні. Приклад напівпрозорих тіней зображено на рисунку 2.10.

Алгоритм відображення напівпрозорих тіней базується на алгоритмі shadow mapping.

Першим кроком роботи алгоритму є створення карти кольору. Дана карта в кожному пікселі представляє собою колір непрозорого об’єкту. Алгоритм наведено на рисунку 2.11.

Першим чином, потрібно сформувати карту глибини з точки зору джерела освітлення. Дана карта глибини формується шляхом рендерінгу непрозорої геометрії. При цьому, обчислення освітленості даної геометрії не проводиться, лише формується карта глибини. Дана карта потрібна для того, аби визначити, на які об’єкти графічної сцени потрапляє пряме світло з даного джерела освітлення.

Наступним кроком є відрисовка напівпрозорих об’єктів графічної сцени з використанням даної карти. При цьому, карта глибини не оновлюється під час даного кроку, вона використовується лише в режимі читання, без запису в неї. Завдяки ній відрисовуються лише ті пікселі графічних об’єктів, що не «затулені» іншими об’єктами. При цьому, відрисовка відбувається в карту кольору напіврозорих об’єктів. Ключовим моментом є встановлення функції змішування (blending) в мультиплікативний режим. Тобто, значення, записані попередньо, будуть помножені на нові. Таким чином, результуюче значення кожного пікселя карти кольору напівпрозорих об’єктів містить результат множення всіх значень, що були записані в дану карту.

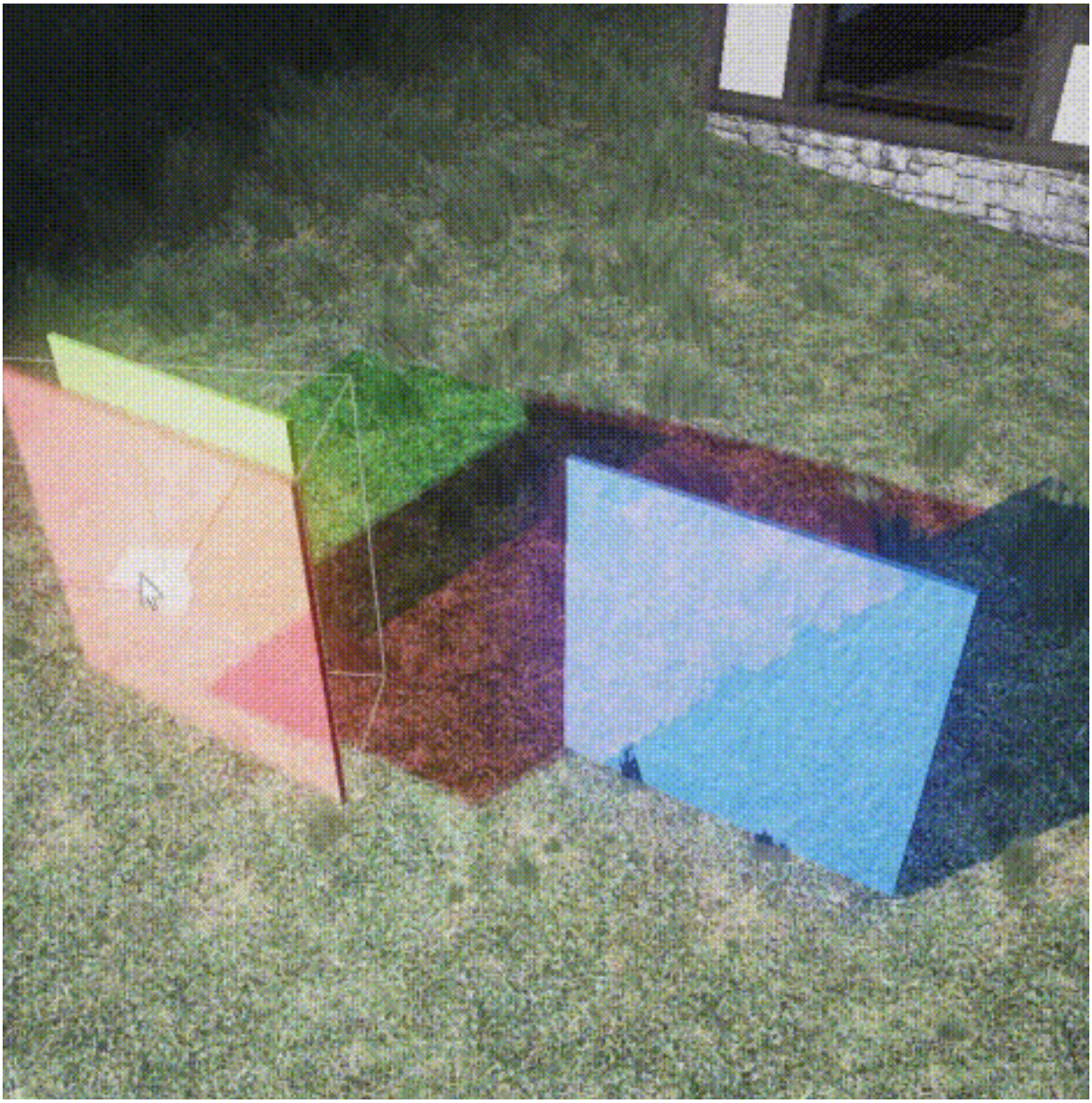

Рисунок 2.10 – Приклад напівпрозорих кольорових тіней

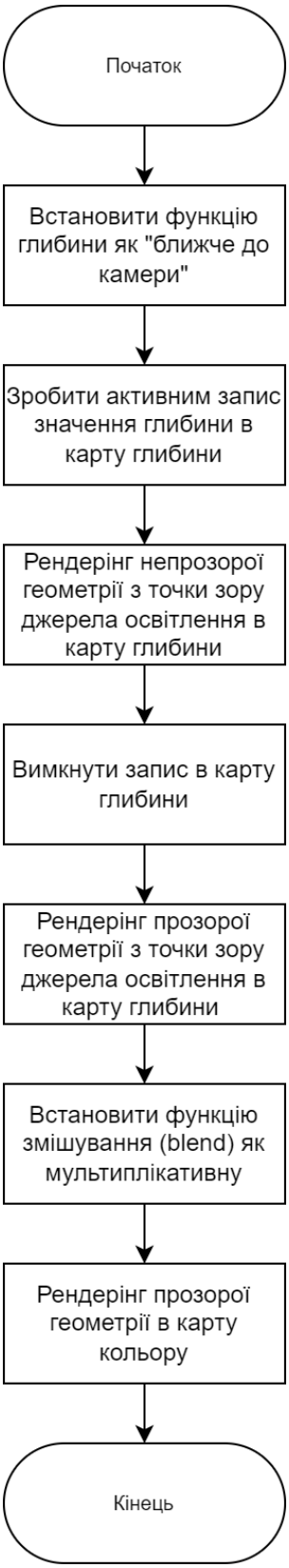

Рисунок 2.11 – Алгоритм формування карти кольору непрозорих об’єктів

Дана карта кольору буде використовуватися під час обчислення освітленості графічної сцени і передбачає модифікацію піксельного шейдеру. Алгоритм обчислення освітленості окремого пікселя об’єкту графічної сцени наведено на рисунку 2.12.

Наступним кроком є модифікація піксельного шейдеру, що обчислює освітленість графічної сцени.

Так, для кожного фрагменту, що потрапляє на екран, і не «загороджений» іншою геометрією з точки зору камери графічної сцени, визначається наступне. Визначається, чи загороджений даний фрагмент іншою геометрією з точки зору джерела освітлення. Очевидно, що у випадку, коли даний фрагмент загороджений геометрією іншого непрозорого об’єкту, то обчислення освітлення для такого фрагменту проводити не треба. У випадку, коли даний об’єкт не загороджений геометрією іншого об’єкту, то обчислюється освітленість даного фрагменту за допомогою фізично-коректної моделі освітлення.

Додатково після проведених обчислень до результуючого значення освітленості фрагменту об’єкту графічної сцени відбувається зчитування значення у відповідному пікселі з карти кольору напівпрозорих об’єктів, що була сформована раніше.

Після зчитування значення кольору, відбувається множення даного значення на значення освітленості фрагменту.

Таким чином, відбувається

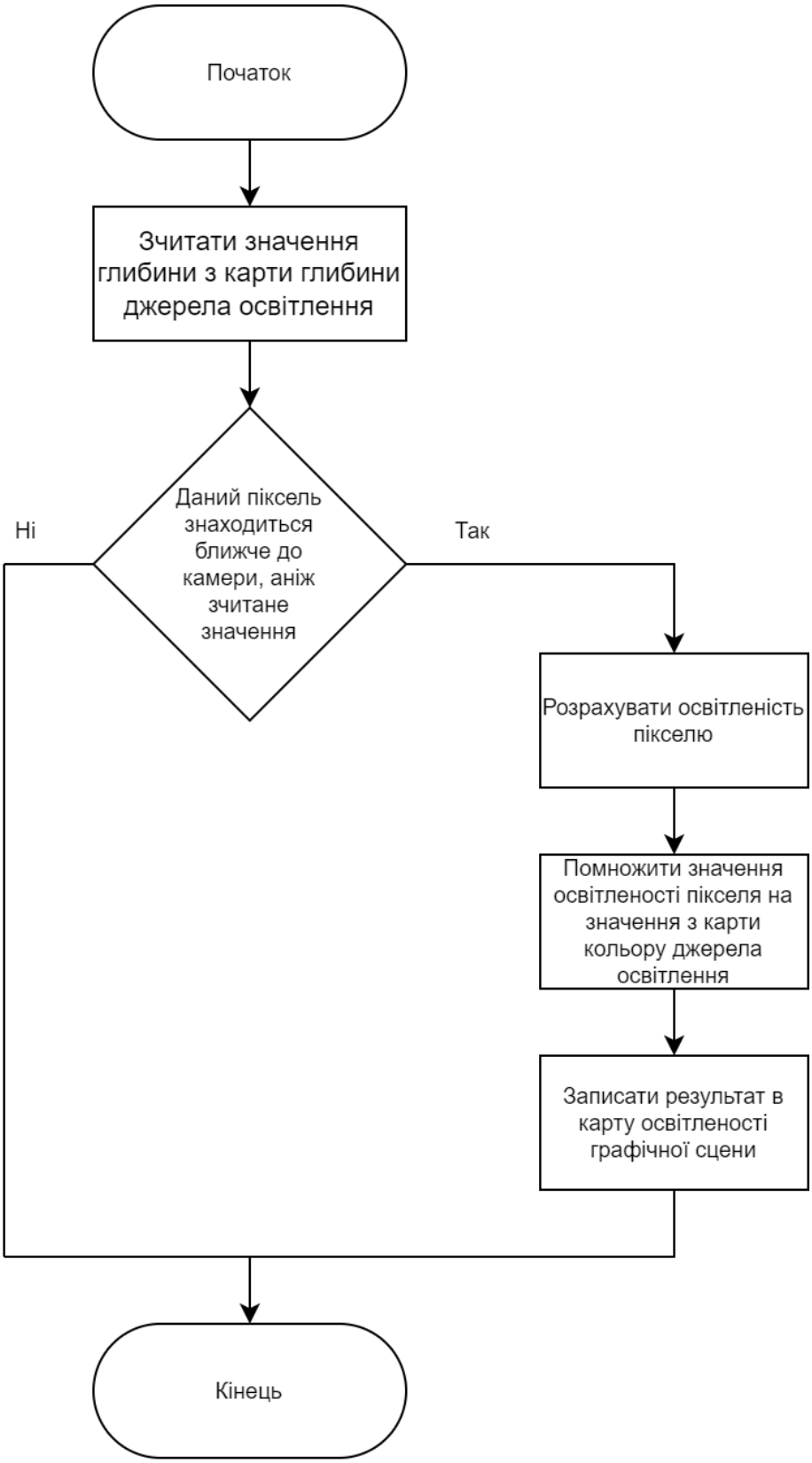

Рисунок 2.12 – Алгоритм обчислень освітленості пікселю екрану

2.4. Алгоритм кластерного освітлення

Ідея кластерного освітлення досить проста: замість того, аби проводити фільтрацію джерел освітлення беручи до уваги 2D тайли екрану, має сенс робити це в 3D, де замість тайлів будуть кластери. Кластери будуються шляхом поділу піраміди огляду по глибині[3].

Покрокова робота алгоритму кластерного освітлення зображена на рисунку 2.13.

Блок-схема алгоритму кластерного освітлення зображена на рисунку 2.14.

Якщо порівнювати даний алгоритм з рендерінгу з Forward+, то відмінностей досить мало. Головною відмінністю є те, що фільтрацію джерел освітлення відбувається не для тайлу екрану, а для кластеру.

Реалізація цієї зміни реалізується шляхом модифікації шейдеру, в якому відбувається фільтрація джерел освітлення.

Якщо порівнювати алгоритми Forward+ та кластерного освітлення, то потрібно поглянути на рисунок 2.13, а саме на жовте джерело освітлення. Як видно з даного рисунку, дане джерело освітлення впливає лише на сферу, що стоїть позаду графічної сцени. При цьому, алгоритм, що розбиває екран на тайли, дав би інший результат: дана сфера приймалась до уваги під час розрахунку освітленості, наприклад, двох інших сфер, що стоять попереду.

Також варто поглянути на червоне джерело освітлення. Як видно з рисунку, фізичні розміри даного джерела досить великі, проте він впливає лише на невелику кількість кластерів, що позитивно впливає на час виконання розрахунків освітленості графічної сцени.

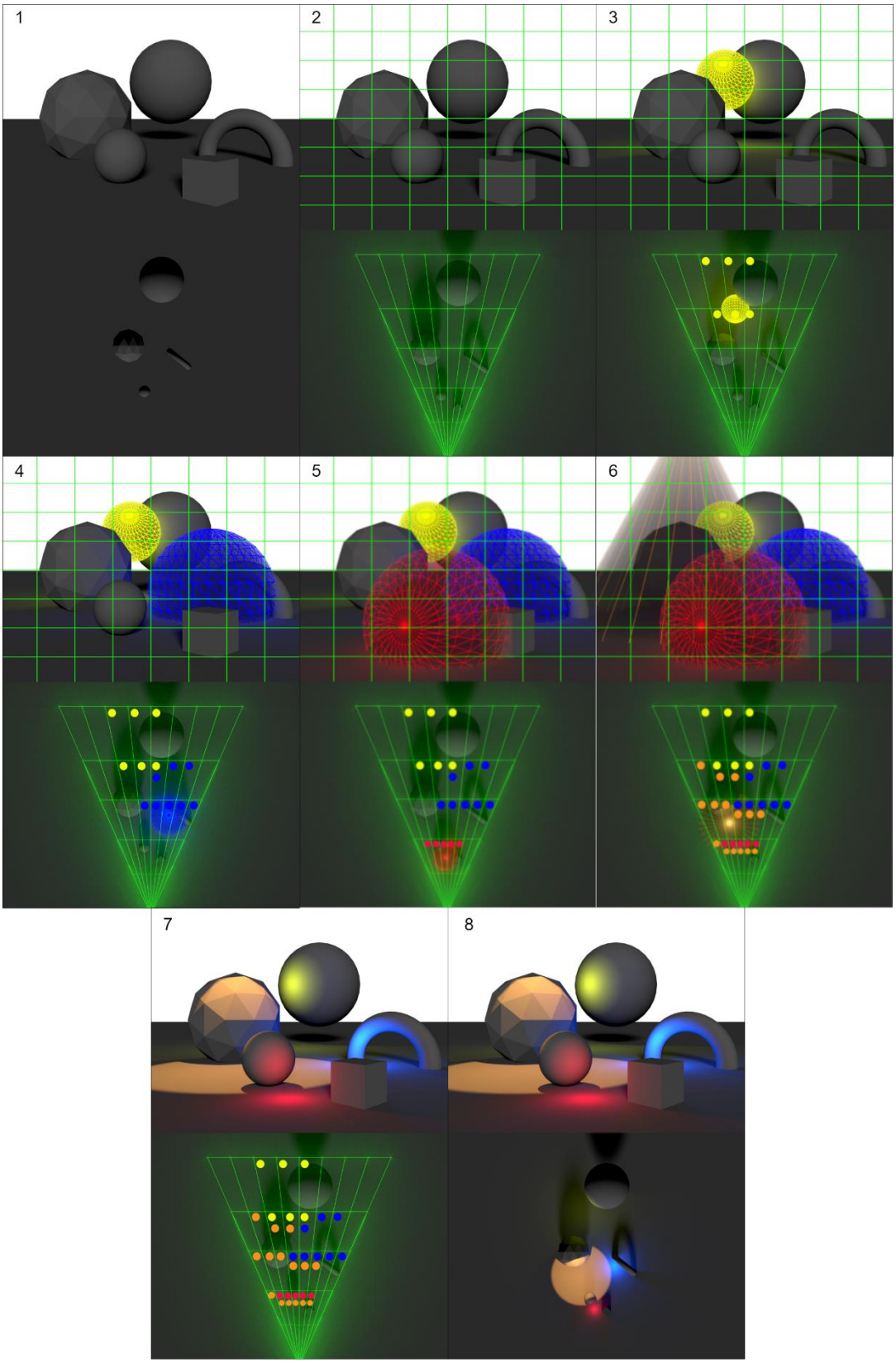

Рисунок 2.13 – Покрокова робота алгоритму кластерного освітлення

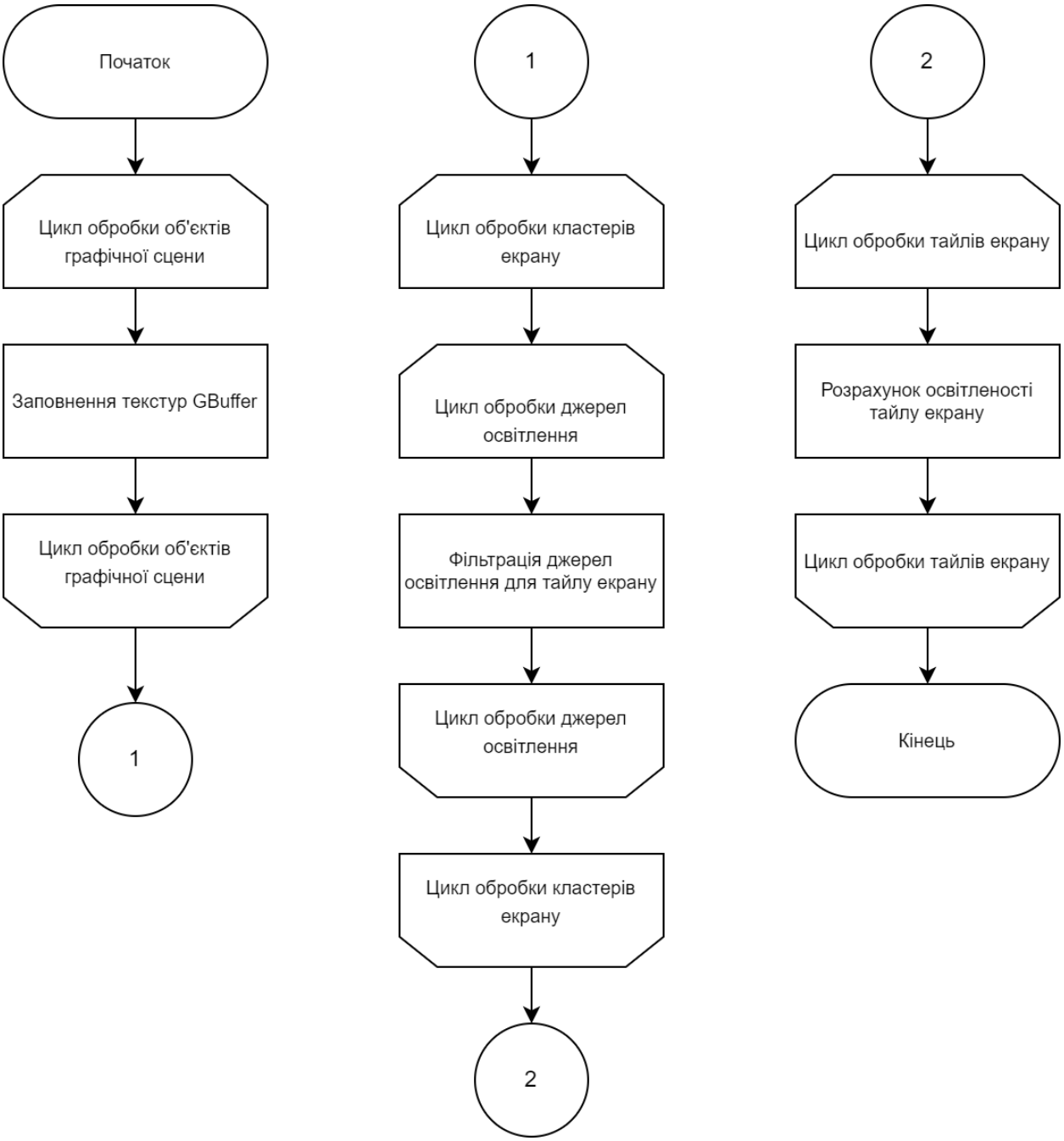

Рисунок 2.14 – Блок-схема алгоритму кластерного освітлення

Додатковою перевагою кластерного освітлення є його гнучкість. Дана особливість виражається в тому, що для даного алгоритму не обов’язково використовувати карту глибини графічної сцени під час фільтрації джерел освітлення. Це досягається за допомогою того, що для розбиття сцени на

кластери достатньо знати властивості камери (розмір, розташування та інші). Знаючи властивості камери графічної сцени, є можливість розбити сцену на кластери. Тобто, в даному випадку досягається повна незалежність джерел освітлення від геометрії графічної сцени – вони можуть оброблятися незалежно один від одного.

Це означає, що складність алгоритму для тайлового освітлення складає O(mesh * lights), тобто складність алгоритму залежить від добутку кількості об’єктів графічної сцени на кількість джерел освітлення. Алгоритм кластерного рендерінгу дозволяє зменшити складність алгоритму до O(mesh + lights), тобто, складність алгоритму залежить від суми кількості об’єктів графічної сцени та кількості джерел освітлення.

Також алгоритм кластерного освітлення має додаткові переваги. Виходячи з того, розбиття піраміди огляду на кластери не залежить від геометрії графічної сцени, це означає, що створення кластерів та фільтрація джерел освітлення може відбуватися на центральному процесорі, а не на графічному процесорі, що вивільнить ресурс графічного процесору на інші операції.

Також немає проблем з відображення напівпрозорих об’єктів, так як для обчислення освітленості напівпрозорих об’єктів використовуються ті ж самі джерела освітлення в тих самих кластерах.

Розглянемо алгоритм більш детально.

2.4.1. Розбиття піраміди огляду на кластери

Першим кроком роботи алгоритму є розбиття піраміди огляду на кластери.

Існує багато різних способів розбиття піраміди огляду на кластери. Приклади такого розбиття наведено на рисунку 2.15.

Розбиття піраміди огляду на кластери буде відбуватися з максимально тісним групуванням джерел освітлення в просторі піраміди огляду. Спочатку екран розбивається на тайли, як це відбувається в алгоритмі тайлового рендерінгу, а потім розбиття тайлів по глибині на проміжки певної довжини. Варто також зауважити, що не існує універсального алгоритму розбиття піраміди огляду на кластери, яка була б оптимальною для всіх можливих сценаріїв та конфігурацій графічної сцени.

Наприклад, якщо більшість джерел освітлення розміщені ближче до камери, а розбиття піраміди огляду реалізовано лінійно по глибині, то щільність джерел освітлення по кластерам буде нерівномірною[4].

Розглянемо розбиття піраміди огляду на кластери на рисунку 2.15. Перший метод розбиття піраміди огляду на кластери відбувається в просторі NDC (Normilized Device Coordinates) – нормалізованих координатах, що є результатом обчислень в вершинному шейдері. Як можна побачити, ті кластери, що знаходяться ближче до far plane, занадто великі, а, отже, неефективні. При цьому, ті кластери, що знаходяться ближче до near plane, надзвичайно великі. Це викликано властивістю нелінійності значення глибини фрагменту в просторі NDC.

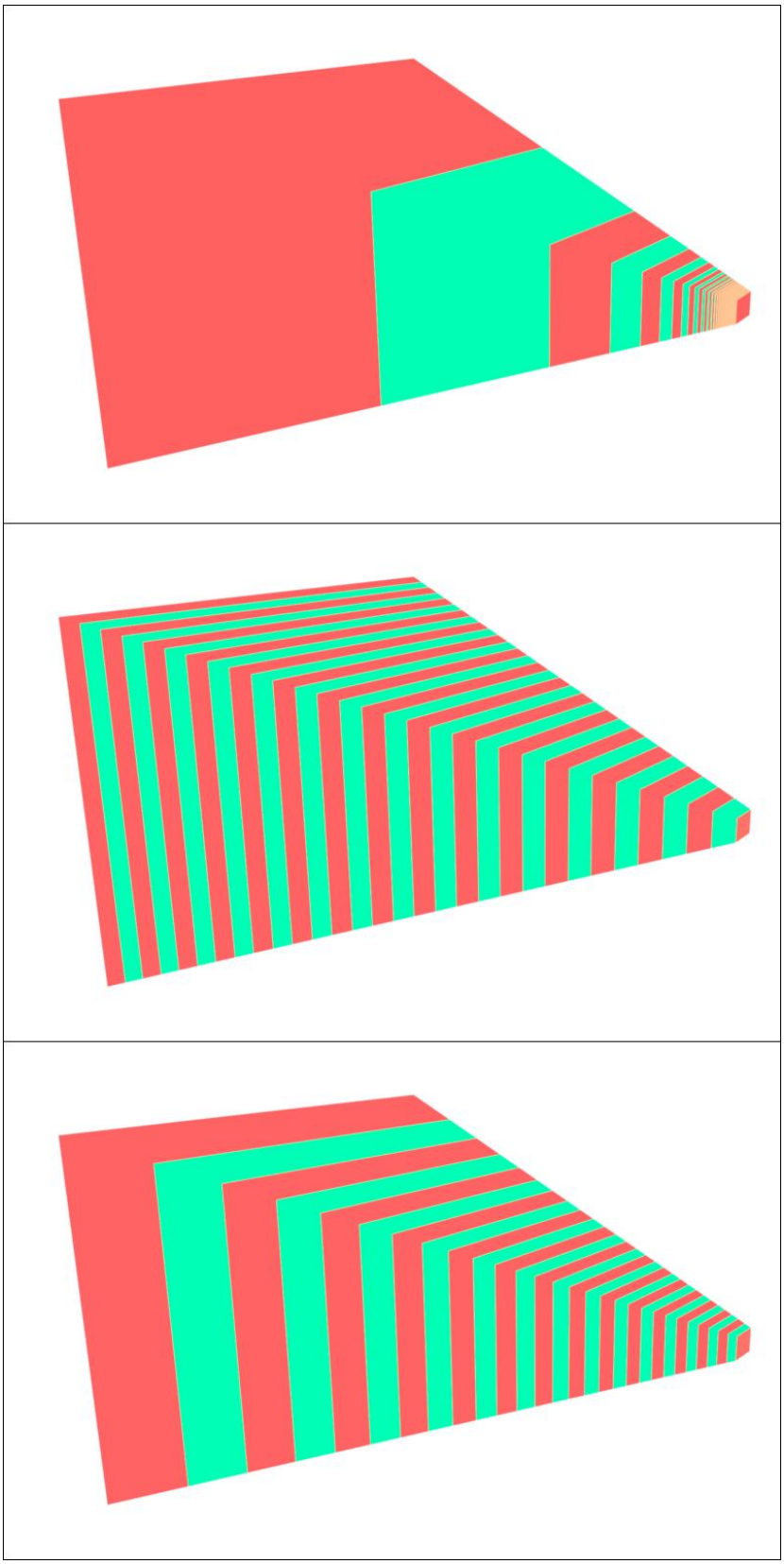

Рисунок 2.15 – Приклад розбиття піраміди огляду на кластери

Даний метод розбиття піраміди огляду не можна вважати гарним, адже розмір кластерів має бути відносно невеликим, щоб він вміщував небагато джерел освітлення, а також рівномірно розподіленим по координаті глибини.

Тому логічним наступним кроком є розбиття піраміди огляду на кластери в просторі view space. Саме такий метод розбиття піраміди огляду зобрежено другим.

Але даний метод має проблеми, що протилежні попередньому методу – кластери, що ближче до near plane занадто великі, тоді як кластери, що ближче до far plane, занадто малі.

Наступним кроком є розбиття піраміди огляду на кластери, використовуючи логарифмічну криву. Формула такого розбиття наведена нижче.

$$Z = Near_z(\frac{Far_z}{Near_z})^{slice/numSlices}$$

де Z – значення глибини, Far_z – значення глибини far plane, $Near_z$ – значення глибини near plane, slice – індекс кластеру, numSlices – кількість кластерів.

Для визначення індексу кластеру використовується наступна формула.

$$slice = \left\lfloor \log(Z)\frac{numSlices}{\log(\frac{Far_Z}{Near_Z})} - \frac{numSlices * \log(Near_Z)}{\log(\frac{Far_Z}{Near_Z})} \right\rfloor$$

На рисунку 2.16 зображено графік залежності розподілення кластерів від значення глибини. Кількість кластерів – 32, $Far_z = 100$, $Near_z = 0.3$.

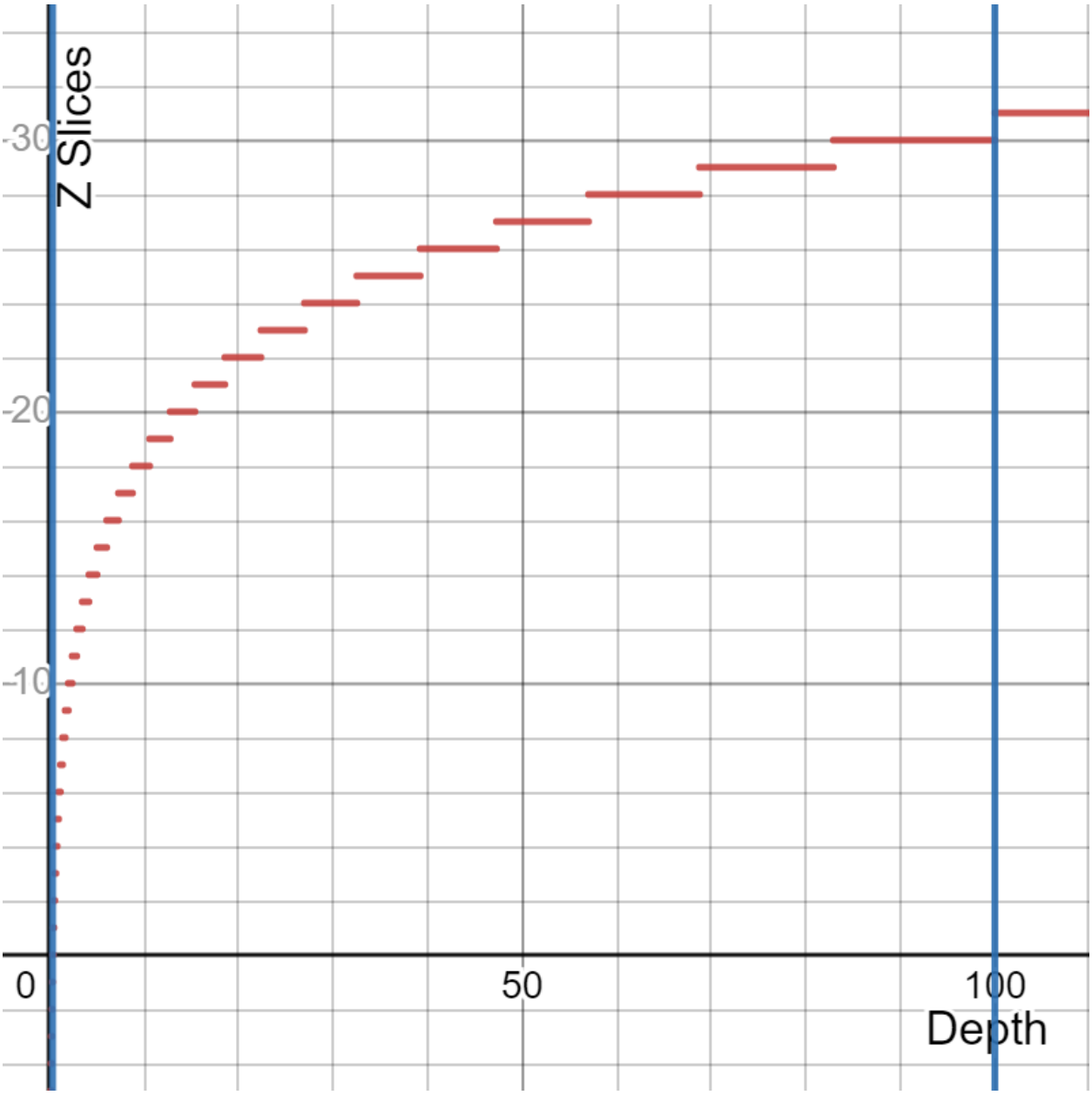

Рисунок 2.16 – Розподілення кластерів в залежності від значення глибини

Варто звернути увагу, що в формулі, що наведена вище, лише одна змінна – значення глибини. Всі інші значення є константою в рамках одного кадру, що спрощує обчислення, адже дані коефіцієнти можуть бути пораховані один раз на кадр.

Приклад розбиття екрану на кластери наведено на рисунку 2.17.

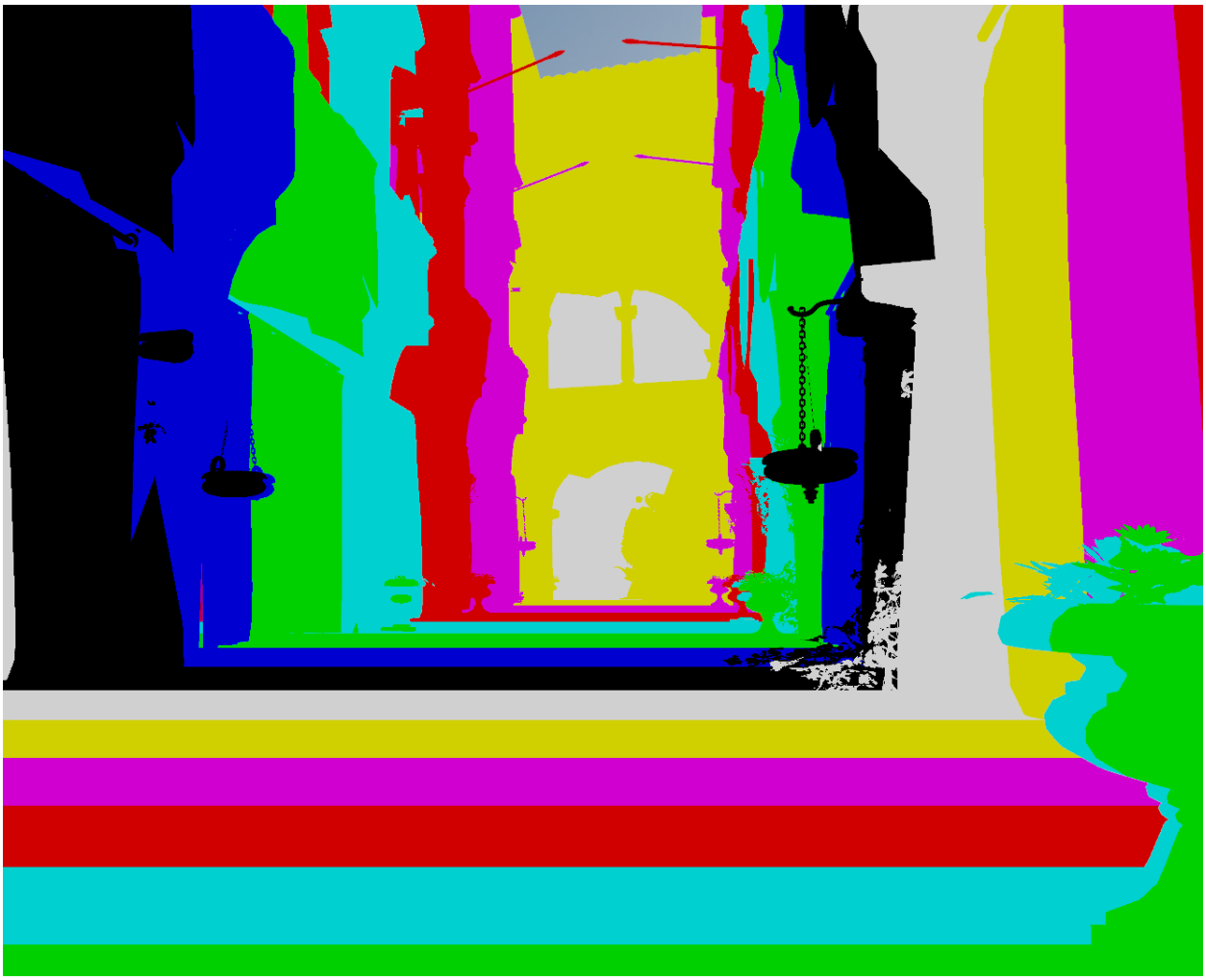

Рисунок 2.17 – Приклад розбиття екрану на кластери

2.4.2. Визначення активних кластерів

Даний етап алгоритму кластерного освітлення є опціональним. Проте, даний етап може досить суттєво прискорити час виконання даного алгоритму. Єдиним недоліком даного етапу є те, що він потребує сформованої карти глибини, тобто, Z-prepass[9].

Так як в загальному випадку не всі кластери можуть бути видимі в кадрі, є сенс визначити, які з них активні, і, відповідно, проводити фільтрацію джерел освітлення лише для цих кластерів.

Перевірка кожного кластеру відбувається незалежно один від одного, що дозволяє проводити дані обчислення паралельно.

Відбувається наступним чином. Для кожного кластеру екрану визначається, яку область екрану він займає. Потім для кожного пікселя екрану з цієї області визначається, чи не «затулений» він геометрією об’єктів графічної сцени. Якщо ні, то даний кластер вважається активним.

Процес визначення активних кластерів може використовувати алгоритм, що наведений в розділі 2.2.

2.4.3. Фільтрація джерел освітлення

Наступним кроком є фільтрація джерел освітлення. Даний процес являє собою визначення, які з джерел освітлення потрапляють в який з кластерів. Якщо джерело освітлення потрапляє в певний кластер, то індекс даного джерела освітлення записується в масив індексів джерел освітлення, що відносяться до даного кластеру[10].

Візуалізація даного процесу наведена на рисунку 2.18.

Фільтрація джерел освітлення базується на наступному принципі: кожне джерело освітлення з дистанцією все менше і менше впливає на освітленість об’єктів графічної сцени. Тобто, чим далі знаходиться джерело від об’єкту графічної сцени, тим менше воно впливає на освітленість даного об’єкту.

Для людського сприйняття є певна межа, коли джерело освітлення знаходиться настільки далеко від об’єкту графічної сцени, що вплив даного джерела на освітленість об’єкту графічної сцени стає непомітною для людського ока.

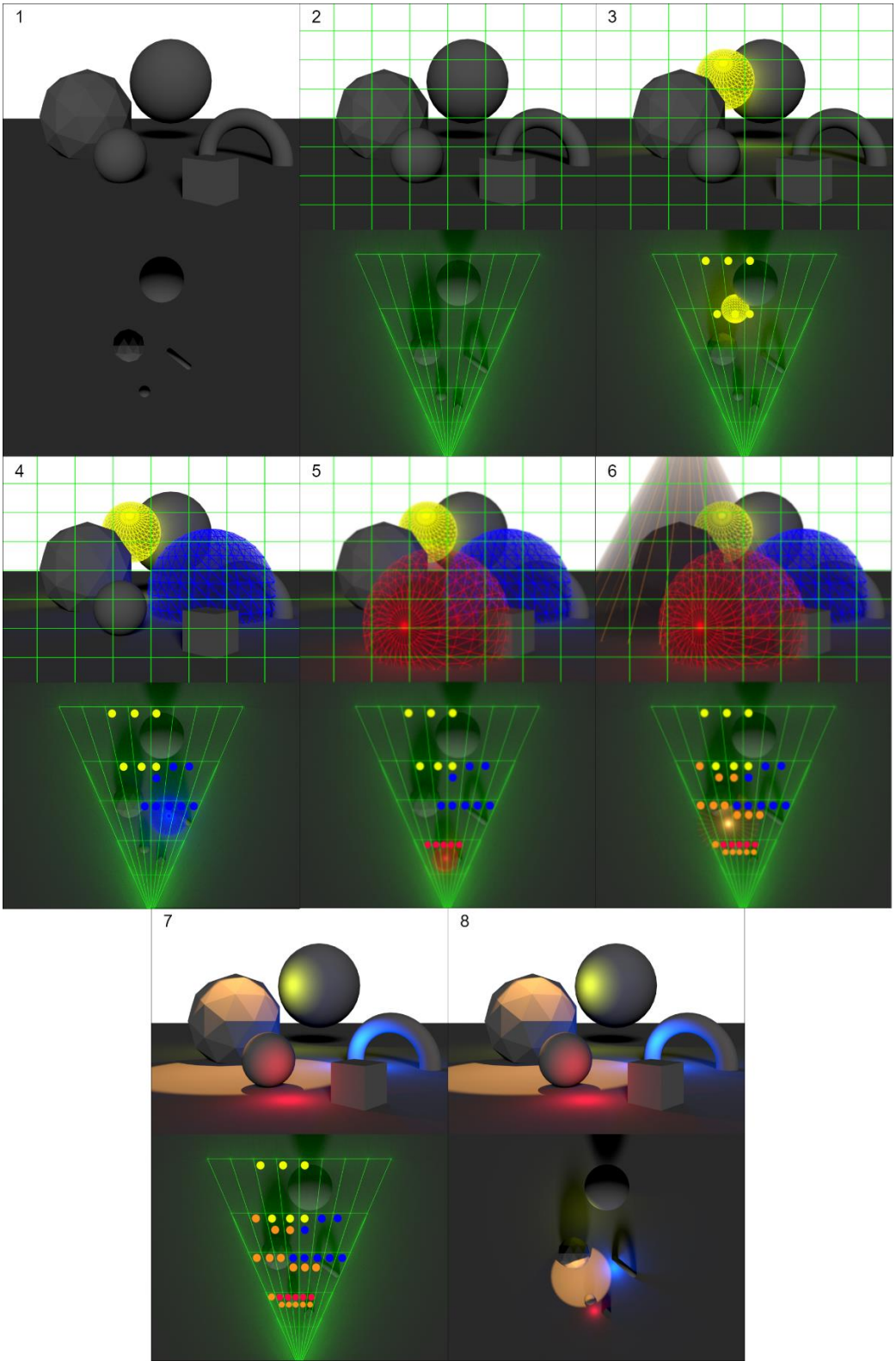

Рисунок 2.18 – Візуалізація процесу фільтрації джерел освітлення алгоритму кластерного освітлення

Саме ця відстань і вважається межею джерела освітлення.

Наступним кроком є визначення обмежуючого об'єму кластеру. В загальному випадку кластер являє собою чотирикутну призму, протилежні сторони якої не є паралельні один до одного. Це створює певні труднощі під час розрахунку перетину обмежуючої сфери джерела освітлення та кластеру. Тому кластер можна представляти у вигляді AABB – точність фільтрації буде трохи нижчою, але зросте швидкодія.

Останнім кроком є формування структури, що зберігає індекси джерел освітлення, що потрапляють в кластер. Дана структура наведена на рисунку 2.19.

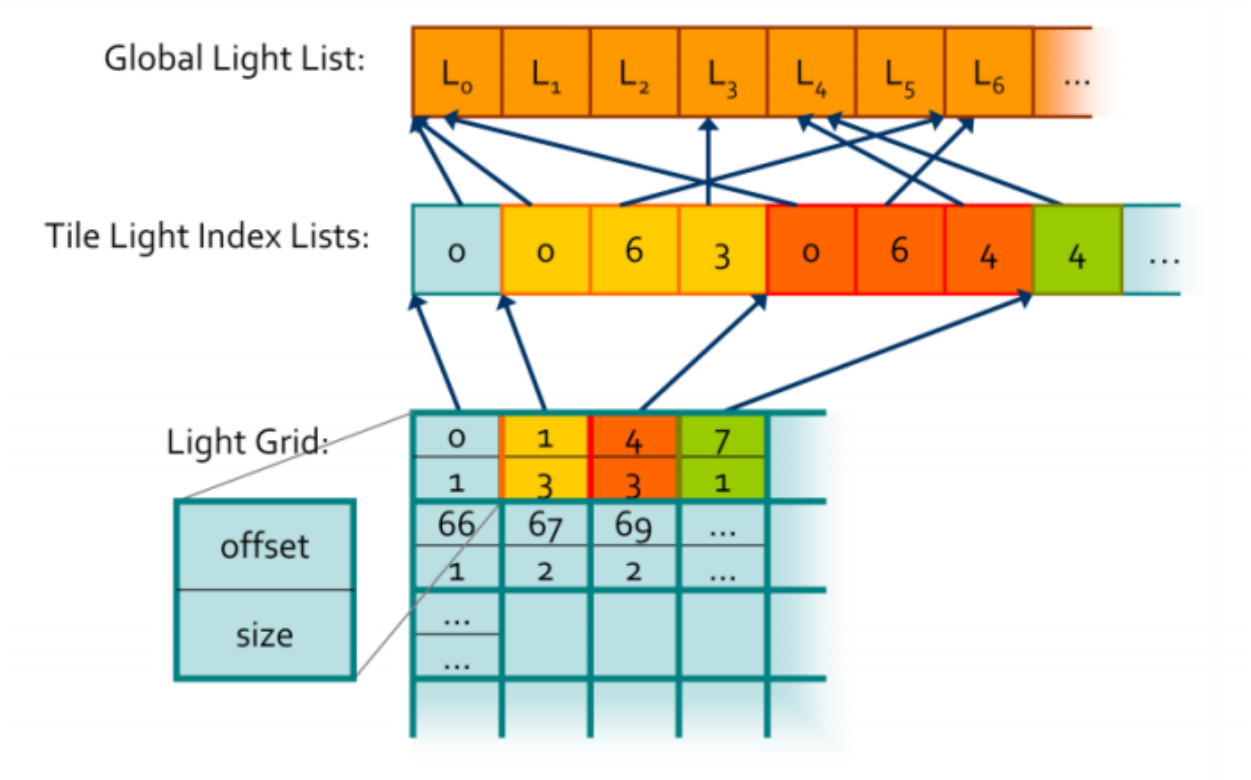

Рисунок 2.19 – Структура даних, що зберігає індекси джерел освітлення для кожного кластеру

Початково всі наявні в графічній сцені джерела освітлення зберігаються в глобальному буфері. Під час одного кадру ця структура даних не змінюється. Вона може змінюватися лише між кадрами. Кожне джерело має певну кількість властивостей, що записані в даному масиві, як от: позиція, колір, яскравість, розмір обмежуючої сфери. Індекс джерела освітлення в даному масиві за час одного кадру не змінюється.

Також потрібен окремий масив, в якому будуть записані індекси всіх джерел освітлення, що потрапляють в кластер. На рисунку 2.19 ним є The Light Index Lists. Але цей масив містить індекси джерел освітлення для всіх кластерів, тож потрібен ще один масив, в якому буде початкова та кінцева позиція в масиві індексів. Саме в даному проміжку і будуть всі джерела освітлення, що потрапляють саме в потрібний кластер. На рисунку 2.19 таким масивом є Light Grid. Цей масив містить стільки елементів, скільки є кластерів.

Висновки до розділу

В даному розділі було детально розглянуто алгоритм Forward+ рендерінгу, де описано основні етапи роботи алгоритму: розбиття екрану на тайли, фільтрація джерел освітлення та обчислення освітленості графічної сцени.

Так, фільтрація джерел освітлення дозволяє суттєво пришвидшити алгоритм Forward+ рендерінгу шляхом відкидання джерел освітлення для тих тайлів, з якими дане джерело освітлення не перетинається.

Також було запропоновано алгоритм більш ефективної фільтрації джерел освітлення, що зменшує кількість хибно позитивних результатів фільтрації.

Разом з цим, було запропоновано також модифікацію алгоритму Forward+ рендерінгу, що дозволяє відображати напівпрозорі тіні.

Наостанок було запропоновано алгоритм кластерного рендерінгу. Його перевагою є те, що фільтрація джерел освітлення може відбуватися ефективно як на графічному процесорі, так і на центральному. Таким чином, навантаження на графічний процесор знизиться, що дозволить проводити більшу кількість інших корисних розрахунків.

3. РЕАЛІЗАЦІЯ ПОЛІПШЕНОГО АЛГОРИТМУ FORWARD+ РЕНДЕРІНГУ

3.1. Вибір мови програмування

Реалізація вище зазначених поліпшень алгоритму Forward+ рендерінгу передбачає створення програмного додатку, що здатен рендерити графічну сцену за допомогою різних методів обчислення освітленості графічної сцени.

Першим кроком в створенні такого додатку є вибір мови програмування. Очевидно, що різні мови програмування мають свої переваги та недоліки, і одні мови програмування краще підходять під одні задачі, і інші мови програмування краще підходять під інші задачі.

Тому вибір мови програмування є важливим етапом. Так як одним з напрямків поліпшення алгоритму Forward+ рендерінгу є покращення його швидкодії, то логічним є вибір мови програмування, що дозволив би використовувати максимум обчислювального потенціалу комп’ютера. Також мова програмування має бути високого рівня для спрощення написання програмного додатку.

Тому найбільш підходящою під такі обмеження є мова програмування C++. Дана мова програмування підтримує багато парадигм програмування, а бере свої витоки з мови програмування C – де-факто стандарт в галузі розробки високопродуктивних систем.

Також перевагою мови програмування C++ є велика стандартна бібліотека, що включає в себе і стандартну бібліотеку мови програмування C. Також мова програмування C++ має велику кількість open-source бібліотек,

що спрощує розробку програмного забезпечення, адже дозволяє використовувати код, перевірений спільнотою open-source розробки[7].

Саме ці переваги мови програмування і стали ключовими під час вибору мови програмування для створення додатку.

3.2. Вибір середовища розробки додатку

Наступним кроком створення програмного додатку є середовище для його розробки. Станом на зараз існує багато різних середовищ, що називаються Integrated Development Environment (IDE). Кожна з них має свої переваги та недоліки. Але було обрано середовище Microsoft Visual Studio.

Дана середа розробки відрізняється широким набором інструментів для тестування та налагодження коду.

3.3. Вибір графічного API

Дуже важливим етапом створення додатку для відображення графічних сцен є вибір графічного API. Графічний API – набір уніфікованих та стандартизованих інтерфейсів взаємодії з драйвером графічного процесору. Уніфікація та стандартизація інтерфейсів дозволяє написати код один раз, а запустити його на великій кількості графічний процесорів. При цьому, не є важливим виробник графічного процесору або його архітектура. Головне, аби драйвер графічного процесору був адаптований під певний графічний API.

Таких API станом на зараз є відносно небагато, найпопулярніші з яких наступні: DirectX, Metal та Vulkan. Перші два є проприєтарними. Тобто, можуть бути використані лише на певній операційній системі. У випадку DirectX такою операційною системою є Windows. Для Metal такою операційною системою виступає MacOS або IOS.

Лише Vulkan має можливість працювати як на Windows, так і на Linux. Це відрізняє його від інших графічних API. Ще однією перевагою Vulkan над іншими є його сучасність, адже даний графічний API досить молодий, і саме тому ввібрав в себе останні тренди розвитку графічних процесорів[6][8].

Тому оптимальним вибором графічного API для створення додатку рендерінгу графічних сцен є Vulkan.

3.4. Вибір шейдерної мови програмування

Дуже важливим аспектом створення програмного додатку є вибір шейдерної мови програмування. Шейдерна мова програмування використовується для написання шейдерних програм – програм, що виконуються на графічному процесорі.

Шейдерні мови програмування мають багато відмінностей від звичайних, адже платформа, на якій цей код буде виконуватися – графічні процесори – накладають певні обмеження.

Найбільш популярною шейдерною мовою програмування є HLSL – High Level Shading Language. Дана мова програмування створена та розвивається компанією Microsoft.

Дана мова програмування має дві ключові особливості. Першою особливістю даної мови програмування є те, що вона є високорівневою та С-подібною. Тобто, шейдери, що написані на даній мові програмування дуже схожа на звичні програми, написані на С. Це суттєво спрощує процес написання коду.

Іншою перевагою даної мови програмування є розвинена інфраструктура. Це виражається великою кількістю компіляторів для цієї

мови, утиліт та інструментів, що спрощуються написання та тестування шейдерів.

Саме тому шейдерною мовою програмування обрано мову HLSL.

3.5. Архітектура та основні компоненти додатку

3.5.1. Модуль роботи з віконною системою Windows

Програмний додаток, що розробляється, має відображати результати роботи в реальному часі. Відповідно, для досягнення цієї мети потрібно результуюче зображення виводити на екрану.

Найпростішим способом це зробити є використання вбудованих механізмів створення вікон Windows, що дозволяє відображати будь-яке зображення в даних вікнах.

Тобто, потрібен компонент додатку, що відповідає за створення вікна, знищення вікна, створення контексту для відображення зображення та обробку подій, що надсилає операційна система даному вікну.

Контекст для відображення зображення має працювати з прикладним графічним програмним інтерфейсом і бути свого роду проміжним етапом між графічним процесором та екраном, на якому дане зображення має відображатися.

Також важливим аспектом даного компоненту є обробка подій, які надсилаються даному вікну від операційної системи. Таким подіями можуть бути наступні: переміщення курсору мишки, натискання клавіши на клавіатурі, відпускання клавіші на клавіатурі, зміна розміру вікна, зміна координат вікна і так далі. Є події, обробка яких включає в себе переналаштування рендерерeу, адже при зміні розміру вікна потрібно

змінити і розмір текстури, в яку відбувається генерація фінального зображення.

3.5.2. Модуль рендереру

Наступним модулем є модуль рендереру. Саме цей модуль відповідає за створення графічного контексту, знищення графічного контексту, надає інтерфейси для створення, модифікації та знищення графічних примітивів.

Цей модуль взаємодіє з модулем віконної системи, адже рендерер відповідає за створення текстури, в яку записується результуюче зображення. Потім це зображення передається віконній системі, які відображає його в створеному для цього вікні.

Також відповідальністю даного модуля є синхронізація між центральним процесором та графічним. Центральний та графічний процесори в загальному випадку працюють незалежно один від одного, проте, час від часу їх робота має бути синхронізована. Наприклад, після завершення генерації нового кадру на центральному процесорі, необхідно дочекатися, коли графічний процесор завершить генерацію попереднього кадру, і лише потім центральний процесор має відправити команди для генерації наступного на графічний процесор.

3.5.3. Модуль алгоритму Forward рендерінгу

Даний модуль відповідає за відображення графічної сцени за допомогою алгоритму Forward рендерінгу.

Даний модуль використовується для порівняння з іншими алгоритмами рендерінгу.

3.5.4. Модуль алгоритму Deferred рендерінгу

Даний модуль відповідає за відображення графічної сцени за допомогою алгоритму Deferred рендерінгу.

Даний модуль використовується для порівняння з іншими алгоритмами рендерінгу.

3.5.5. Модуль алгоритму Forward+ рендерінгу

Даний модуль відповідає за відображення графічної сцени за допомогою алгоритму Forward+ рендерінгу.

Для цього даний модуль створює всі необхідні текстури, буфери, завантажує шейдери та компілює їх в реальному часі.

Цей модуль відповідає за всі етапи алгоритму Forward+ рендерінгу: розбиття екрану на кластери, фільтрація джерел освітлення та обчислення освітленості графічної сцени.

Висновки до розділу

В даному розділі було розглянуто реалізацію програмного додатку, що реалізує алгоритм Forward+ рендерінгу з покращеннями, що наведені в розділі 2.

Мовою програмування було обрано C++, бо вона має ряд переваг: це мова високого рівня, яка компілюється в асемблерний код, що дає максимальну продуктивність програмного додатку.

Середовищем розробки було обрано Microsoft Visual Studio, адже вона відрізняється широким набором інструментів для тестування та налагодження коду.

Графічним API було обрано Vulkan. Даний графічний API є досить молодим, і уособлює в собі всі досягнення в комп’ютерній графіці за останній час. Також даний графічний API є зручним у використанні та дуже гнучким, що дозволяє реалізовувати будь-які алгоритми.

Шейдерною мовою програмування було обрано HLSL. Дана шейдерна мова програмування є найбільш популярною, адже бурхливо розвивається. Вона підтримує всі сучасні підходи до написання коду шейдерних програм.

Також було описано основні компоненти додатку: модуль роботи з віконною системою, модуль рендереру та модуль, що реалізує алгоритм Forward+ рендерінгу.

4. ТЕСТУВАННЯ РОЗРОБЛЕНОГО ДОДАТКУ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

4.1. Тестування додатку

Проведемо тестування розробленого додатку. Розглянемо зображення, згенеровані за допомогою різних алгоритмів рендерінгу.

На рисунку 4.1 зображення, згенеровані за допомогою алгоритму Forward рендерінгу.

Рисунок 4.1 – Зображення, згенеровані за допомогою алгоритму Forward рендерінгу

На рисунку 4.2 зображення, згенеровані за допомогою алгоритму Deferred рендерінгу.

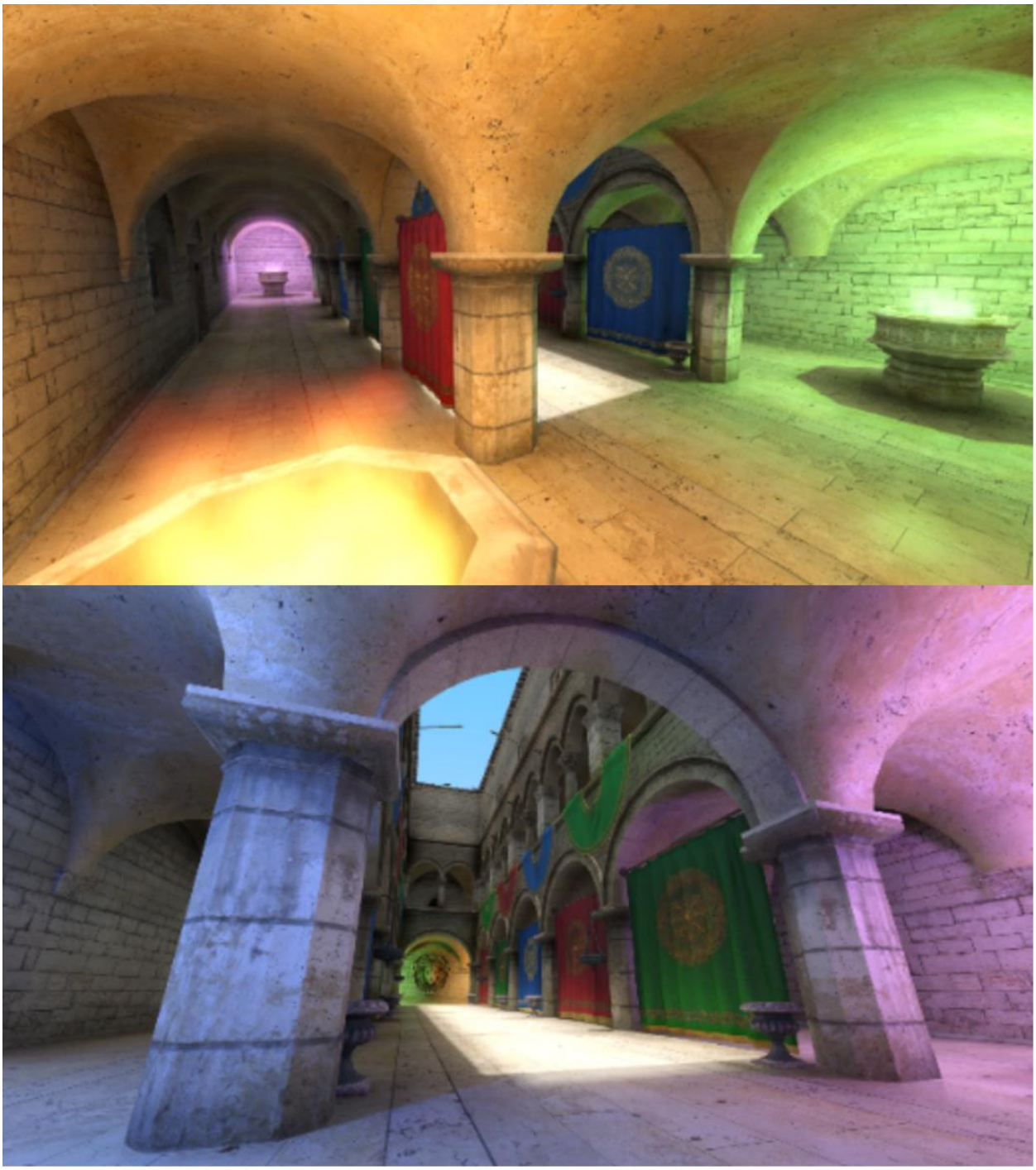

Рисунок 4.2 – Зображення, згенеровані за допомогою алгоритму Deferred рендерінгу

На рисунку 4.3 зображення, згенеровані за допомогою алгоритму Forward+ рендерінгу.

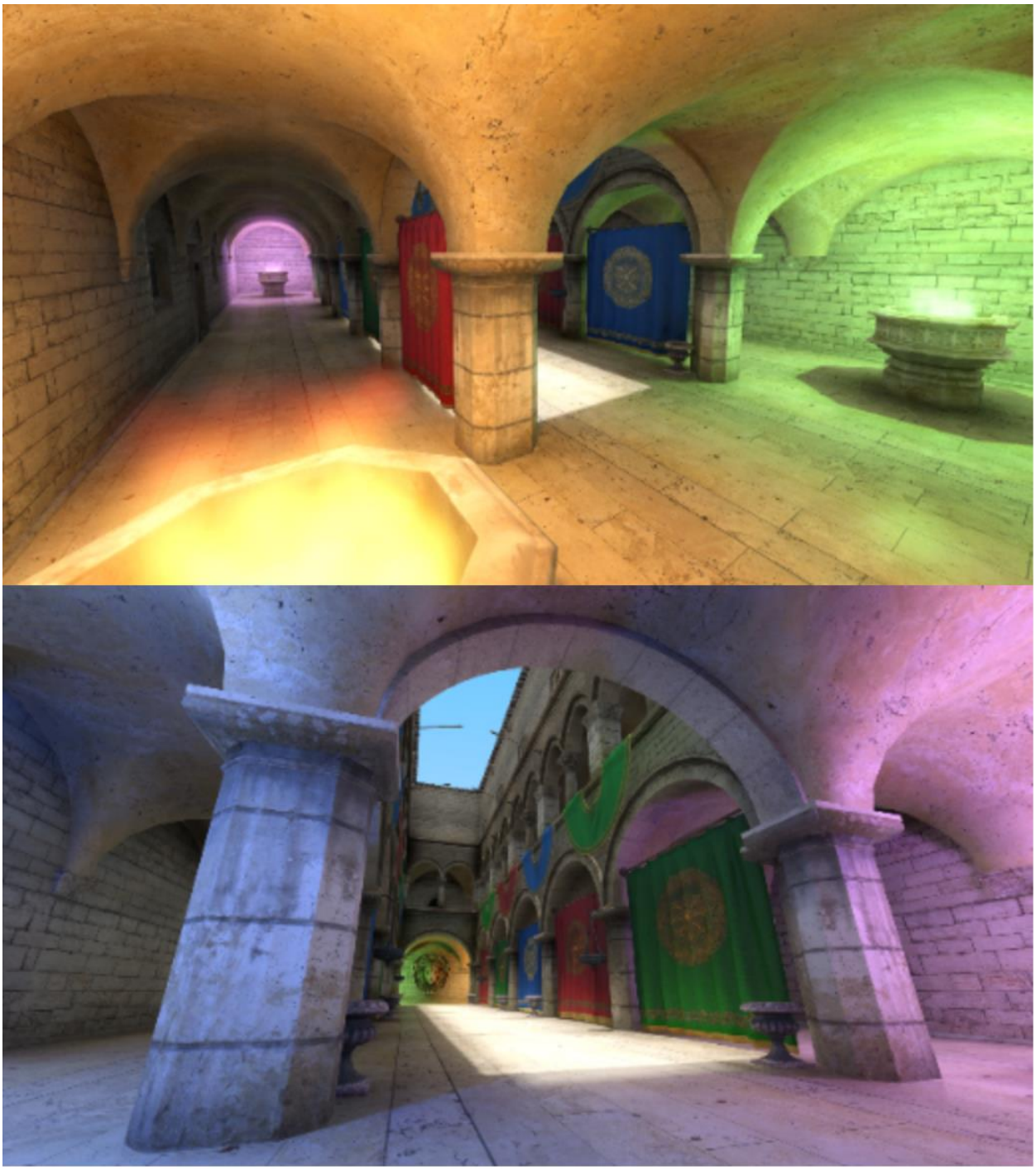

Рисунок 4.3 – Зображення, згенеровані за допомогою модифікованого алгоритму Forward+ рендерінгу

Також на рисунку 4.4 зображено напівпрозорі тіні. Дане зображення було згенеровано за допомогою модифікованого алгоритму Forward+ рендерінгу.

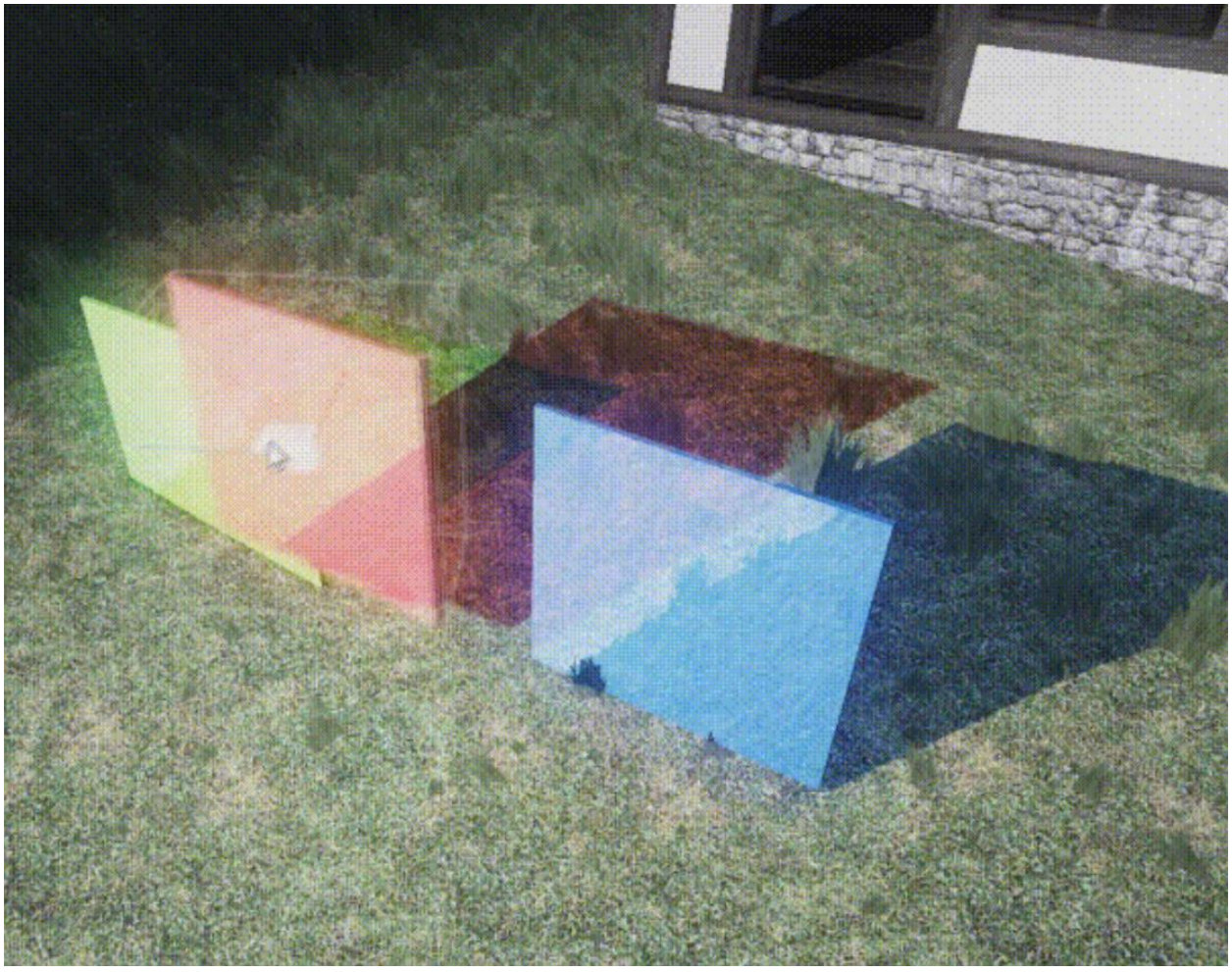

Рисунок 4.4 – Напівпрозорі тіні, що відображаються за допомогою модифікованого алгоритму Forward+ рендерінгу

Проведемо аналіз швидкодії цих трьох алгоритмів, використовуючи однакову конфігурацію графічної сцени. Отримані результати зображені на рисунках 4.5-4.7.

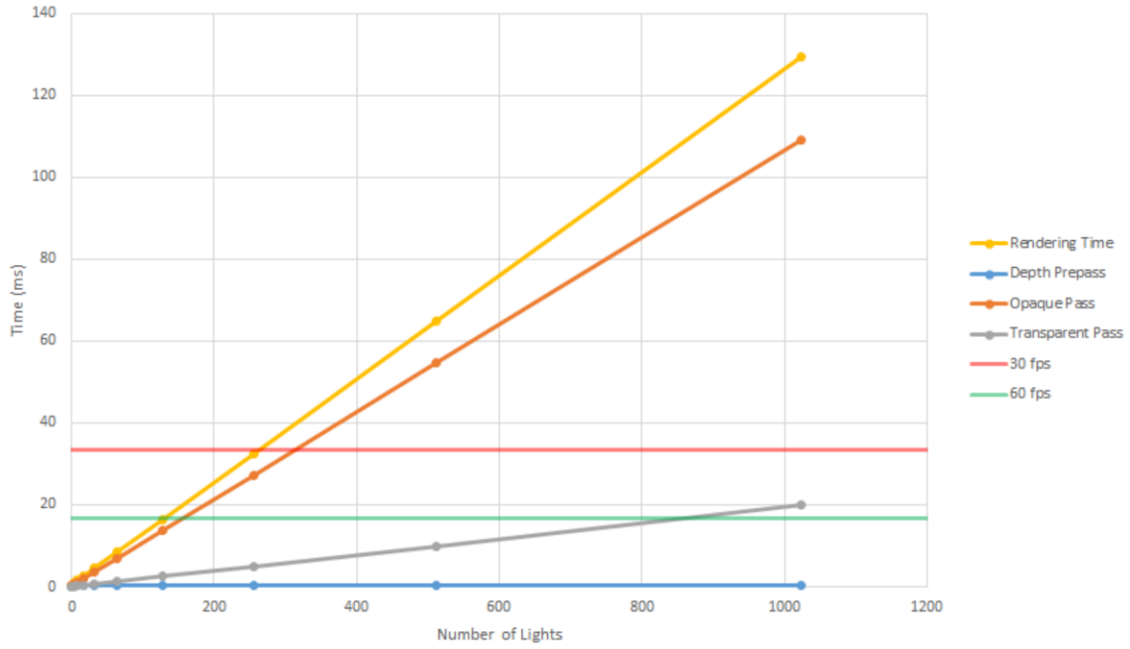

Рисунок 4.5 – Графік залежності часу виконання алгоритму Forward рендерінгу в залежності від кількості джерел освітлення

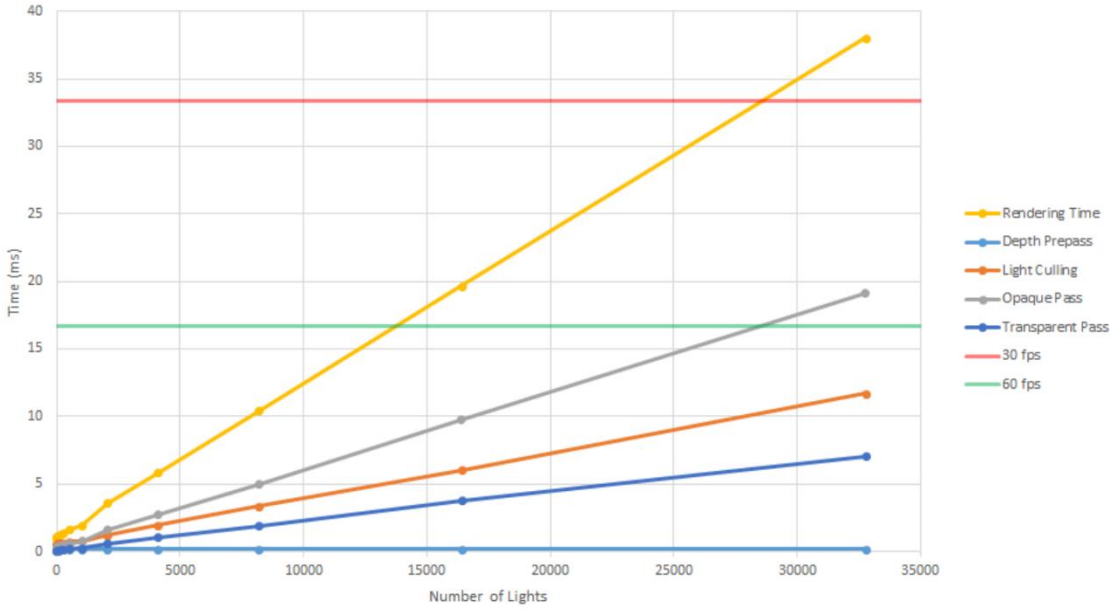

Рисунок 4.6 – Графік залежності часу виконання алгоритму Deferred рендерінгу в залежності від кількості джерел освітлення

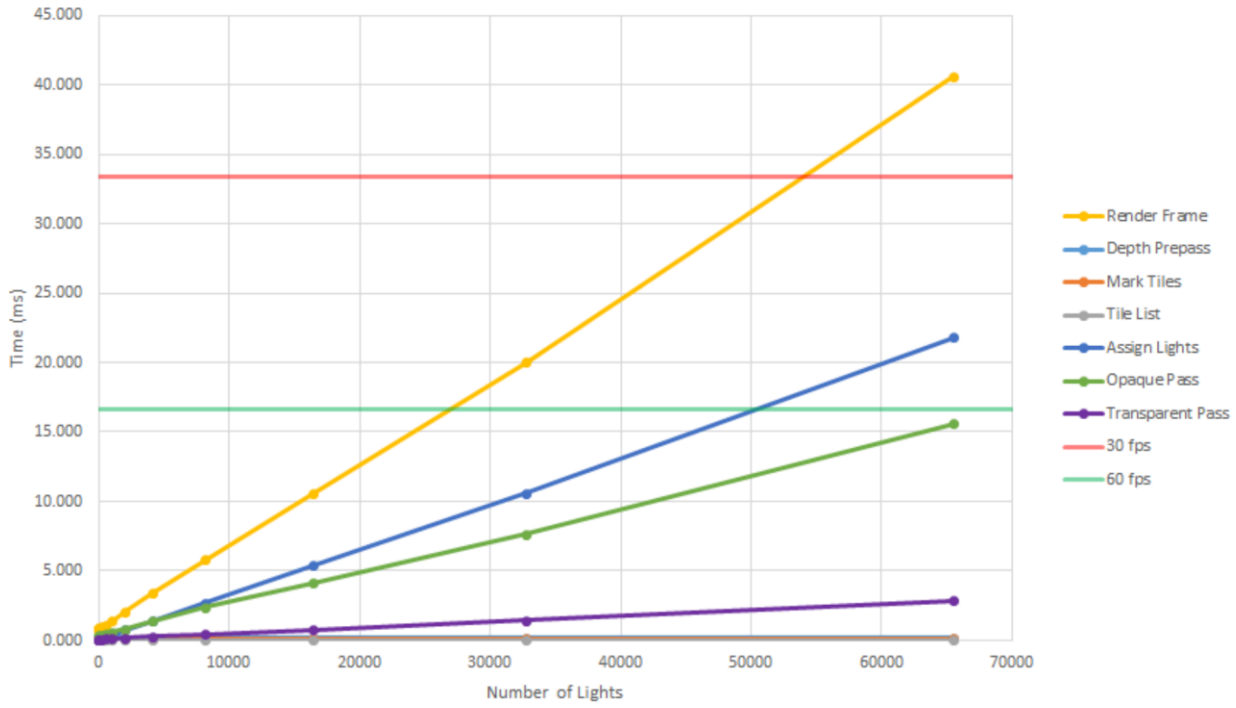

Рисунок 4.7 – Графік залежності часу виконання модифікованого алгоритму Forward+ рендерінгу в залежності від кількості джерел освітлення

4.2. Аналіз результатів

Алгоритми Forward, Deferred та Forward+ рендерінгу генерують однакові зображення, як можна бачити з рисунків 4.1-4.3. Таким чином, в даному аспекті вони є взаємозамінні.

Якщо поглянути на результати швидкодії даних алгоритмів, то в першу чергу увагу привертає швидкодія алгоритму Forward рендерінгу. Для 1000 джерел освітлення, як можна бачити з рисунку 4.5, час виконання алгоритму сягає 130 мс. Тобто, 7.7 кадрів на секунду. Отже, ні про яку плавність зображення в даному мова не йде.

Якщо поглянути на швидкодію алгоритму Deferred рендерінгу на рисунку 4.6, то генерація зображення графічної сцени з 30000 джерел

освітлення, займає 35мс. Це приблизно 29 кадрів на секунду. Враховуючи, що, в порівнянні з минулим випадком, графічна сцена містить в 30 разів більше джерел освітлення, а час виконання алгоритму зменшився в майже 4 рази, то це суттєве покращення швидкодії.

Поглянемо на результати тестування алгоритму Forward+ рендерінгу. Для генерації зображення з 30000 джерел освітлення, потрібно 19мс. Це набагато менше, ніж час виконання алгоритму Deferred рендерінгу. Тобто, алгоритм Forward+ рендерінгу дозволяє генерувати приблизно 53 зображення на секунду.

Вище наведено час роботи трьох алгоритмів рендерінгу. З результатів добре видно, що модифікований алгоритм Forward+ рендерінгу займає найменше часу на виконання, що робить його найбільш прийнятним для реалізації в рендерерах реального часу.

Висновки до розділу

В даному розділі було проведено тестування трьох алгоритмів рендерінгу: Forward, Deferred та модифікований алгоритм Forward+.

Було проведено тестування згенерованих зображень, використовуючи дані алгоритму рендерінгу. Відмінностей в згенерованих зображень виявлено не було, отже, вони є взаємозамінні.

Також було проведено тестування швидкодії даних алгоритмів рендерінгу з різною кількістю джерел освітлення в графічній сцені. Найбільше часу для генерації одного зображення потрібно для алгоритму Forward рендерінгу, найменше – для модифікованого алгоритму Forward+.

ВИСНОВКИ

В магістерській дисертації розглянуто проблему та актуальність розрахунку освітленості графічних сцен під час рендерінгу.

Результати роботи полягають в наступному:

1. Проведено аналіз сфер людської діяльності, де відображення інформації в графічному вигляді грає важливу роль.
2. Проведено аналіз проблеми обчислення освітленості графічних сцен..
3. Проведено аналіз мов програмування, що дозволяють ефективно реалізувати дані алгоритми.
4. Проведено аналіз середовищ розробки, що дозволяють ефективно використовувати обрану мову програмування.
5. Проведено аналіз наступних алгоритмів рендерінгу: Forward, Deferred та Forward+.
6. Проведено порівняння алгоритмів між собою. Описано переваги та недоліки кожного з них.
7. Запропоновано модифікацію до алгоритму Forward+ рендерінгу, що дозволяє збільшити ефективність фільтрації джерел освітлення, що позитивно впливає на час виконання алгоритму.
8. Запропоновано модифікацію до алгоритму Forward+ рендерінгу, що полягає в розбитті піраміди огляду на кластери, що дозволяє більш ефективно проводити фільтрацію джерел освітлення, що позитивно впливає на час виконання алгоритму.

9. Запропоновано модифікацію до алгоритму Forward+ рендерінгу, що дозволяє відображати напівпрозорі тіні.

За результатами досліджень було встановлено, що модифікації до алгоритму Forward+ рендерінгу значно зменшують час виконання алгоритму в порівнянні з алгоритмами Forward та Deferred рендерінгу, що дозволяє або відображати більшу кількість джерел освітлення, або суттєво зменшити час генерації кадру.

Також можливість відображення напівпрозорих тіней позитивно впливає на сприйняття результуючого зображення, адже робить його більш правдоподібним.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. T. Akenine-Möller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki, and S. Hillaire, Real-Time Rendering, Fourth Edition.
2. M.Pharr, W. Jakob, G. Humphreys, Physically Based Rendering, Third Edition.
3. Ángel Ortiz. A Primer On Efficient Rendering Algorithms & Clustered Shading. [Електронний ресурс]. – Режим Доступу: http://www.aortiz.me/2018/12/21/CG.html.
4. Nvidia. Depth precision visualized. [Електронний ресурс]. – Режим доступу: https://developer.nvidia.com/content/depth-precision-visualized.
5. І. Денисенко. «Алгоритм фільтрації джерел освітлення, що не потрапляють на тайл екрану», Прикладна математика та комп'ютинг. «XV науково-практична конференція магістрантів та аспірантів ПМК-2022 факультету прикладної математики» (Київ, 16-18 листопада 2022 р.).
6. Alexander Overvoorde. Vulkan Tutorial. [Електронний ресурс]. – Режим доступу: https://vulkan-tutorial.com.
7. ISO International Standard ISO/IEC 14882:2020(E) – Programming Language C++.
8. Khronos Group. Vulkan 1.3.237 – A specification. [Електронний ресурс]. – Режим доступу: https://registry.khronos.org/vulkan/specs/1.3-extensions/html/vkspec.html.
9. D. Zhdan, Nvidia. Tiled shading: light-culling – reaching the speed of light, GDC 2018. [Електронний ресурс]. – Режим доступу: https://ubm-twvideo01.s3.amazonaws.com/o1/vault/gdc2016/Presentations/Zhdan_Sjoholm_Light_culling_MGPU.pdf.

10. J. Turánszki, Optimizing tile-based light culling. [Електронний ресурс]. Режим доступу: https://wickedengine.net/2018/01/10/optimizing-tile-based-light-culling/.