Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя
(повне найменування вищого навчального закладу)
Комп'ютерно-інформаційних систем і програмної інженерії
(назва факультету)
Комп'ютерних наук
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту (роботи)

магістр
(освітній ступінь (освітньо-кваліфікаційний рівень))

на тему: **Розробка архітектури графічного рушія на основі об'єктного підходу**

Виконав: студент (ка) 6 курсу, групи СНм-61
спеціальності (напряму підготовки)
122 - комп'ютерні науки
(шифр і назва спеціальності (напряму підготовки))

(підпис) (прізвище та ініціали)

Керівник (підпис) Фриз М. Є. (прізвище та ініціали)

Нормоконтроль (підпис) Мацюк О. В. (прізвище та ініціали)

Рецензент (підпис) Бойко І. В. (прізвище та ініціали)

м. Тернопіль – 2019

АНОТАЦІЯ

Розробка архітектури графічного рушія на основі об'єктного підходу // Дипломна робота ОР «Магістр» // Ячменьов Ігор Сергійович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СНм-61 // Тернопіль, 2019 // С. - 120, рис. – 29, табл. – 8, додат. – 11, бібліогр. – 78.

Ключові слова: ГРАФІЧНИЙ РУШІЙ, ВІЗУАЛІЗАЦІЯ, ДАНІ, АЛГОРИТМИ, ДОСЛІДЖЕННЯ

У дипломній роботі досліджено графічні рушія для візуалізації зображення в реальному часу. Розроблено архітектуру графічного рушія та реалізовано алгоритм для візуалізації даних в реальному часі.

У вступі проведено огляд розробки архітектури графічних рушіїв та охарактеризовано основні завдання, які необхідно вирішити.

В першому розділі описано візуалізацію зображення в реальному часі та яке програмне забезпечення використовується, які етапи виконуються для отримання зображення описано концепції та проектні рішенння, які складають основу для розробки графічного рушія.

В другому розділі розглянуто шаблони проектування які використовуються для розробки архітектури на основі об'єктного підходу, яким чином відбуваєтсья обробка шейдерів та ефектів у графічному рушії, які дані не спряють отриманню кінцевого зображення.

В третьому розділі зображено архітектури графічного рушія на основі об'єктного подходуа та реалізований алгоритму для візуалізації даних в реальному часі на основі об'єктного підходу, та як шаблони проектування були використані в процесі розробки.

ANNOTATION

Graphics engine architecture development based on object approach // Diploma thesis Master degree // Yachmenov Ihor S. // Ternopil’ Ivan Pul’uj National Technical University, Faculty of Computer Information System and Software Engineering, Department of Computer Science, group SNm-61/ / Ternopil', 2019 // P. – 120 , Tables – 8, Fig. – 29, Diagrams – 11, References – 78 .

Keywords: GRAPHICS ENGINE, RENDERING, DATA, ALGORITHMS, RESEARCH

The diploma thesis explores the graphics engine for real-time image rendering. The architecture of the graphical engine was developed and an algorithm for real-time data visualization was implemented.

The introduction provides an overview of the development of the architecture of graphical engines and outlines the main tasks that need to be addressed.

The first section describes real-time image visualization and what software is used, what steps are taken to get the image, and describes the concepts and design solutions that form the basis for the development of the graphical engine.

The second section discusses design patterns used to build an object-based architecture, how shaders are processed, and effects in the graphics engine that do not interfere with the final image.

The third section describes the graphical engine architectures based on the object approach and an algorithm for real-time data visualization based on the object approach and how the design templates were used in the development process.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Program Interface) – програмний інтерфейс;

Cg (англ. C for Graphics) – шейдерна мова високого рівня Nvidia;

CPU (англ. Central processing unit) – центральний процесор;

D3D (англ. Direct3D) – програмний інтерфейс який надає функції для взаємодії операційної системи і програм з драйверами відеокарти, підтримується апаратно відеокартами;

GLSL (англ. OpenGL Shading Language) – шейдерна мова високого рівня OpenGL;

GPU (англ. Graphics processing unit) – графічний процесор;

GUI (англ. Graphical User Interface) – тип інтерфейсу, який дозволяє користувачам взаємодіяти з електронними пристроями через графічні зображення та візуальні вказівки;

HLSL (англ. High Level Shader Language) – шейдерна мова високого рівня DirectX;

Kd – структура даних з поділом простору для упорядкування точок в *k*-мірному просторі;

LGPL (англ. GNU Lesser General Public License) – безкоштовна ліцензія на програмне забезпечення;

LOD (англ. Level of detail) – рівень деталізації;

Mesh (англ. Polygon mesh) – набір вершин, ребер, та граней, що описують форму багатогранного об'єкта в тривимірній графіці;

Octree – дерево октантів, у якому кожна вершина має вісім нащадків;

Open Graphics Library (англ. OpenGL) – специфікація, що визначає незалежний від мови програмування крос-платформовий програмний інтерфейс для написання застосунків, що використовують 2D та 3D комп'ютерну графіку.

Зміст

ВСТУП

Розробка програмного забезпечення для візуалізації даних в режимі реального часу є одна з найскладніших областей інженерії програмного забезпечення через кількість та складність необхідних алгоритмів. Ці програмні проекти мають спільне те, що їм потрібно структурувати дані в основній пам'яті, обробляти їх та надсилати на графічний пристрій для ефективної візуалізації. Ці повторювані та складні алгоритми забезпечуються графічними рушіями, що дозволяють швидше розвивати додатки візуалізації в реальному часі.

Ця дипломна робота описує концепції та проектні рішення, які складають основу для розробки графічного рушія, представленого в цьому документі. В дипломній роботі представлено алгоритм для візуалізації даних в режимі реального часу.

Актуальність тема: візуалізація даних є дуже актуальною проблематикою в будь якій галузі. Оскільки використання візуалізації зображення в реальному часі використовується в багатьох напрямках науки, на основі проаналізованих даних спеціально навчені люди можуть впровадити нові покращення, які вплинуть як і на прибуток так і на інші життєві процеси підприємства чи іншої галузі.

Мета роботи: розробка архітектури графічного рушія на основі об’єктного підходу, та реалізація алгоритму для візуалізації зображення в реальному часі. Даний алгоритм візуалізації може використовуватися в таких завданнях як тривимірна симуляція простору, побудова об’ємних моделей, тривимірному моделюванні простору з меншими затратами ресурсів ніж інші графічні рушії. Дана архітектура може бути удосконалена для надання зображенню реалістичності при необхідності.

Для досягнення поставленої мети було сформульовано такі завдання:

1. Здійснити пошук та аналіз джерел для візуалізації зображення в режимі

реального часу, дослідження методів та засобів опрацювання даних в графічних рушіях, а також способи візуалізації даних.

2. На основі зібраних даних та порівнянь з відомими аналогами запропонувати архітектуру для візуалізації даних в режимі реального часу.

3. Програмно реалізувати алгоритм для візуалізації даних в режимі реального часу.

4. Провести аналіз швидкості роботи алгоритмів для візуалізації даних, оцінити доцільність використання алгоритмів для візуалізації даних відповідно до завдання.

Об’єкт дослідження: процес візуалізації даних в режимі реального часу.

Предмет дослідження: алгоритми для візуалізації даних в реального часу.

Новизна: розроблено архітектуру з використання алгоритму візуалізації даних в реальному часі який використовує менші обсяги ресурсів ніж його аналоги, архітектура може набути подальшого розвитку для візуалізації даних з більшою деталізацією зображення та додаванням нових ефектів.

Практичне значення отриманих результатів: Розроблено архітектуру графічного рушія на основі об’єктного підходу, та реалізовано алгоритм для візуалізації зображення в реальному часі.

Апробація результатів магістерської роботи. Окремі результати роботи представленні на двох наукових конференціях:

1. VIII Міжнародна науково – технічна конференція молодих учених та студентів, «ВИКОРИСТАННЯ ЗАСОБІВ GOOGLE MAPS ДЛЯ ВІЗУАЛІЗАЦІЇ ДАНИХ».

2. VIII Міжнародна науково-технічноа конференція молодих учених та студентів, «СТАНДАРТИ РОЗУМНОГО МІСТА».

1 АНАЛІЗ ЛІТЕРАТУРНИХ ДЖЕРЕЛ ЗА ТЕМАТИКОЮ «ГРАФІЧНІ РУШІЇ»

1.1 Візуалізація в реальному часі

Візуалізація в реальному часі це процес відображення тривимірних даних в вигляді картинки на комп'ютері з інтерактивною швидкістю відображення. Швидкість відображення вимірюється в кадрах в секунду (fps), що для візуалізації в режимі реального часу має бути вище 15 кадрів в секунду [34], щоб користувач визнавав їх гладкими. Приблизно від 72 кадрів в секунду і вище[34], людське око не може виявити будь-яких відмінностей у швидкості відображення.

Для досягнення інтерактивної частоти кадрів необхідно здійснити деякі компроміси. По – перше, згенероване зображення зазвичай не настільки фотореалістичне, як у глобальних методах освітлення, таких як трасування шляху[8] та радіозвучність[14]. На рисунку 1.1 показано порівняння зображення, згенерованого за допомогою методів візуалізації в режимі реального часу, та зображення, створеного за допомогою методів глобального освітлення. Порівняння відображеного в реальному часі зображення (~ 500 кадрів в секунду) зліва та зображення, створеного методом глобального освітлення праворуч (~ 50000 секунд на кадр) [8]. Крім того, необхідно застосовувати складні алгоритми, щоб уникнути роботи, яка не сприяє остаточному зображенню.

Рисунок 1.1 – Порівняння візуалізованих зображень в реальному часі

З моменту впровадження графічної картки 3Dfx Voodoo 1 у 1996 році [7], інтерактивна візуалізація в режимі реального часу стає доступною на настільних ПК. Перед цим повне обчислення кінцевого зображення повинен був виконувати центральних процессор . З того часу графічні прискорювачі (їх також називають графічними процесорами) швидко розвивалися з точки зору продуктивності, функціональності та гнучкості. Тому все більше і більше робіт переноситься з процесора на GPU, щоб збільшити загальну ефективність процесу візуалізації. В даний час ЦП використовується, головним чином, для підготовки даних для графічного процесора шляхом структурування та обходу даних моделі в основній пам'яті. Додатково до цих моделей застосовується фізичне моделювання та штучний інтелект.

Для прикладу взято Geforce 2060 NVIDIA[32], який здатний обчислювати близьке до фотореалістичності зображення у режимі реального часу. Рисунок 1.2 зразок зображення, створеного цим графічним процесором. На рисунку зображено фотореалістичне відображення модної моделі, що відображається з інтерактивною швидкістю кадрів за допомогою графічного прискорювача Geforce 2080 NVIDIA.

Рисунок 1.2 – Відображення моделі з інтерактивною швидкістю

Оскільки більшості програм візуалізації в режимі реального часу потрібно вирішувати майже однакові функції графічного прискорювача, було розроблено декілька стандартних інтерфейсів програмування (їх також називають графічними API), щоб забезпечити портативність програм, що дозволяють надсилати команди різним графічним пристроям. Два найвизначніші графічні API – це Microsoft DirectX та OpenGL.

Перша версія мультимедійного API DirectX була розроблена Microsoft в 1995 році і отримала назву GameSDK. Він надав інтерфейсні класи, які могли використовуватися мовами програмування C і C ++.

OpenGL був спочатку представлений Silicon Graphics у 1992 році і підтримується більшістю операційних систем, доступних до цього часу. Це робить його першим вибором для розробки портативних графічних програм. Постачальники графічного обладнання та інші компанії, пов'язані з графікою, організували себе як BoardGL Review Review Board (ARB), який веде специфікацію інтерфейсу OpenGL. OpenGL використовує так звану концепцію розширення для ранньої інтеграції нових функцій, наданих графічними прискорювачами.

1.2 Графічне програмне забезпечення

Оскільки програмування графічних рушіїв стало дуже складним завданням, і часто той самий код потрібен знову і знову, графічне програмне забезпечення потрібне для підтримки розробників. Це програмне забезпечення (їх також називають Render Engine) забезпечує обгортку навколо графічного API та часто потрібне для швидкої розробки графічних програм. Таким чином, їх основне завдання – підтримка розробників графічних програм.

Загальноприйняті особливості графічних рушіїв:

- завантаження та надання 3D – моделей;

- завантаження та керування різними видами форматів зображень;
- застосування текстур, затінення та більш досконалі ефекти до моделей;
- організація моделей для легкого виконання різних операцій на моделях;
- оптимізація продуктивності для більш швидкого відтворення моделей.

Графічні рушії доступні як комерційні (і часто дуже дорогі) пакети, так і як проекти з відкритим кодом. Деякі з них представлені у розділі 2.6. Це залежить від наступних фактів, який механізм візуалізації відповідає конкретному проекту:

- бюджет проекту;
- мова програмування проекту – графічні рушії часто доступні на C / C ++, C #, Java;
- платформи, на яких планують працювати проекти. Деякі графічні рушії підтримують як ПК, так і ігрові консолі. Деякі графічні рушії можуть працювати лише в Microsoft Windows, а інші також підтримують Linux;
- графічні рушії можуть надавати скриптові мови які не завжди надають графічні бібліотеки.

Графічний ефект – це поєднання алгоритмів, необхідних для візуалізації 3D-моделі на екран, щоб надати бажаний візуальний вигляд. Тому графічні ефекти можуть відрізнятися за такими моментами:

- геометрією, вертексними і фрагментними шейдерами, що вони використовують;
- числом графічних проходів, що потрібна;
- необхідними данними на вхід, це включає число і типи текстур та освітлення, а також вертексні атрибути геометрії.

1.3 Графічний конвеєр. Етапи роботи графічного конвеєра

У комп'ютерній графіці конвеєр візуалізації або просто графічний конвеєр – це концептуальна модель, яка описує, які кроки потрібно виконати графічній системі, щоб зробити 3D – сцену на 2D – екрані. Після створення 3D – моделі, наприклад, у відеоігри чи будь-якій іншій 3D-комп'ютерній анімації, графічний конвеєр – це процес перетворення цієї 3D – моделі в те, що відображається на комп'ютері. Оскільки етапи, необхідні для цієї операції, залежать від використовуваного програмного та апаратного забезпечення та бажаних характеристик дисплея, не існує універсального графічного конвеєра, придатного для всіх випадків. Однак інтерфейси програмування графічних прикладних програм (API), такі як Direct3D і OpenGL, були створені для уніфікації подібних кроків та управління графічним конвеєром певного апаратного прискорювача. Ці інтерфейси API абстрагують базове обладнання та дозволяють програмісту уникати написання коду для управління прискорювачами графічного обладнання (AMD / Intel / NVIDIA тощо).

Модель графічного конвеєра зазвичай використовується при візуалізації в режимі реального часу. Найчастіше більшість етапів конвеєра реалізовані апаратно, що дозволяє здійснити спеціальні оптимізації. Термін "графічний конвеєр" використовується в аналогічному сенсі до конвеєра в процесорах: окремі етапи конвеєра проходять паралельно, але блокуються до тих пір, поки не буде виконаний найповільніший крок.

Графічний конвеєр описує традиційні етапи обробки, що виконуються 3D – програмами рендерингу в реальному часі (додатки RTR). Кожному з цих додатків необхідно виконати такі завдання:

- будь-яка програма – RTR потребує даних, які вона може візуалізувати. Тому вона може генерувати дані або може завантажувати їх безпосередньо з носія інформації;
- Ці дані мають бути організовані та підготовлені для надання програмі;
- Дані повинні бути перетворені, щоб вони вміщувались у вікно перегляду пристрою виводу;
- потім дані перетворюються у пікселі та відображаються на пристроєві виводу.

Деякі з цих завдань виконуються на процесорі, а інші повинні виконуватись спеціалізованим графічним прискорювачем (також званий графічним процесором – GPU). Типовим прикладом завдання, яке потрібно виконати на процесорі, є завантаження даних із жорсткого диска. З іншого боку, відображення згенерованих пікселів може здійснюватися тільки GPU.

Зі збільшенням потужності та гнучкості відеокарт сьогодні деякі завдання можуть бути перенесені з процесора на GPU. Навіть генерування даних тепер може здійснюватися GPU в деяких випадках. Основною причиною такої підвищеної гнучкості GPU є його програмованість. Іншим прикладом завдання, яке тепер може виконуватися на графічному процесорі, є фізичне моделювання, включаючи виявлення зіткнень. Govindaraju представив алгоритм [16], який здатний обчислити набір об’єктів які можуть потенційно перетнутися на графічному процесорі за допомогою запитів оклюзії простору. Іншим прикладом фізичного моделювання, яке добре піддається реалізації GPU, є моделювання тканини. Zeller представив алгоритм моделювання полотна [57], де полотно моделюється як набір 3D частинок, збережених у текстурі з плаваючою комою. Потім рух цих частинок обчислюється лише вершинними та фрагментарними програмами, що працюють на GPU.

Вище зазначені завдання безпосередньо відображаються на етапах графічного конвеєра, які детально описані в наступних розділах. Огляд етапів разом з можливими ресурсами які вони використовують для обробки можна знайти на рисунок 2.1. Всі етапи працюють паралельно. Це означає, що хоча напр. гама – корекція застосовується до фрагменту на етапі растерізації, нові вершини перетворюються на етапі геометрії. Тому графічний конвеєр поводиться як виробнича збірка, і кожен етап додає щось до попереднього етапу.

Основне завдання етапу генерації – створити дані об'єктів, які мають зберігатися в основній пам'яті комп'ютера для подальшої візуалізації. Цього можна досягти, генеруючи дані під час виконання або завантажуючи попередньо обчислені дані з носія інформації. Потім моделі разом зі своїми даними організовуються в основну пам'ять для гнучкої та ефективної подальшої обробки. Це може бути таким же простим, як перелік посилань на моделі або вони можуть бути організовані у графіку, як описано в розділі 2.5.

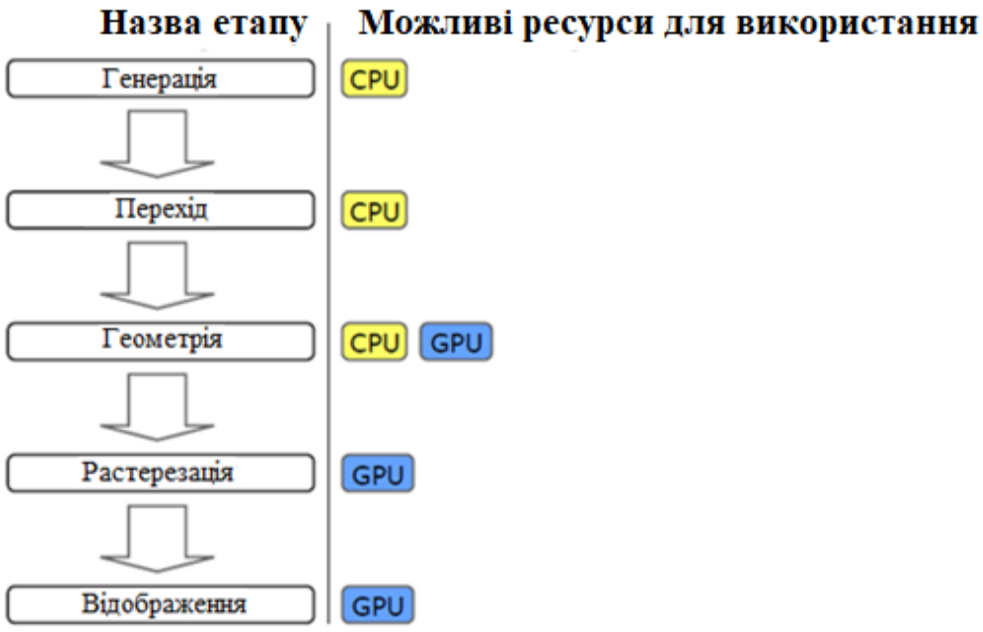


Рисунок 1.3 – Етапи роботи графічного конвеєра

На етапі переходу структуру даних додатка відміняють та змінюють у разі необхідності. Під час проходу відповідні графічні команди надсилаються на графічний прискорювач для створення візуального зображення даних моделі. Як комунікаційний інтерфейс між процесором та графічним процесором, більшість програм використовують або Direct3D [11] Microsoft, або інтерфейс програмування графічної програми OpenGL [1] SGI (графічний API). Поєднання етапу генерації та обходу часто називають етапом застосування. Цей етап також може включати обробку подій додатком, штучний інтелект та фізичне моделювання.

На етапі геометрії обробляються багатокутники і вершини моделі. Задіяні етапи обробки включають:

- перетворення вершинних позиції з об'єктного на перспективне у просторі кліпів;
- обчислення освітлення та затінення за вершиною;
- координати текстури можна генерувати та трансформувати (якщо потрібно);
- збирання вершин в примітиви (наприклад, трикутники);
- виконання відсікання примітивів;
- виконання перспективного поділу;
- перетворення координат багатокутника з простору кліпу у вікно перегляду, залежне від пристрою.

До 1998 року графічні прискорювачі ПК не змогли виконати описані вище кроки в апаратному забезпеченні. Завдання розробника було реалізувати цю функціональність за допомогою процесора. У 1999 році NVIDIA випустила графічний прискорювач під назвою GeForce 256 [32], який здійснив перетворення вершин та обчислення освітлення в апаратному забезпеченні. Ця нова функція отримала назву апаратної трансформації та освітлення (T&L). Оскільки ці обчислення проводяться набагато швидше, ніж за допомогою процесора, полігонна

пропускна здатність графічного конвеєра була збільшена. Розрахунки перетворення, виконані на векторах позицій моделі, проілюстровані на рисунку 1.4.

Однак, зростаюча складність графічних додатків вимагає також більшої гнучкості графічних процесорів з точки зору можливих операцій за вершиною. Моделі трансформації та освітлення вже не були достатніми. Тому графічна індустрія запровадила так звані вершинні програми (також звані як вершинні шейдери). З цього моменту можна змінити атрибути вершин, такі як нормалі та кольори, визначені користувацьким способом.

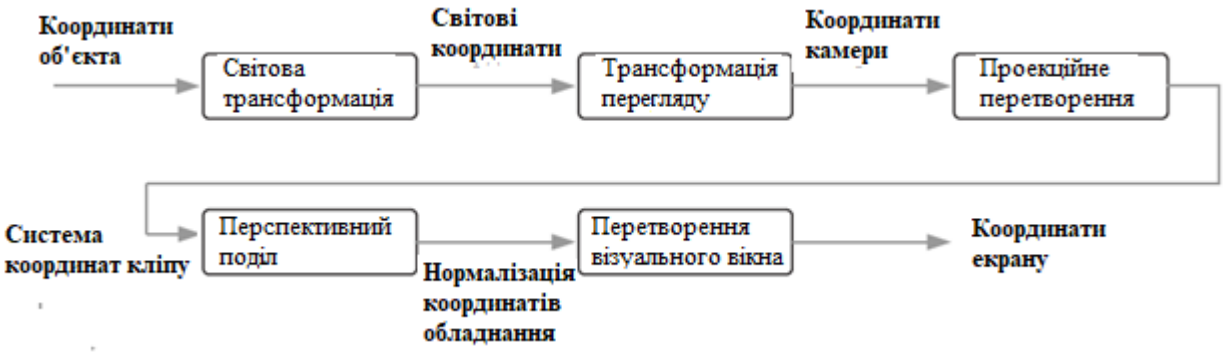


Рисунок 1.4 – Обчислення геометрії

Завдання етапу растеризації полягає в перетворенні двовимірних примітивів (включаючи кольори вершин і подібні атрибути) у кольорові пікселі, що зберігаються у двовимірному масиві під назвою framebuffer. Цей процес також часто називають перетворенням.

Виконані кроки наведені нижче:

- налаштування трикутника;
- перетворення трикутники у фрагменти, інтерполюючи атрибути вершин кутів трикутника. Інтерпольовані атрибути є, наприклад, значення кольору, текстурних координат та глибини;
- перетворення власної прозорості. За допомогою таких тестів фрагменти можна відкинути, перш ніж вони відображатимуться;

- обчислення остаточних кольорів фрагментів, використовуючи освітлення, відображення текстури та змішування альфа;
- визначити видимість між фрагментами, протестувавши їхні значення глибини, що зберігається в Z – буфері [53].

Для підвищення гнучкості цього етапу конвеєру були введені фрагментальні програми. За допомогою цих програм GPU можна запрограмувати для виконання визначеного користувачем набору операцій над фрагментами.

На етапі відображення корекція гамми може бути застосована до кольорів фрагментів згідно рівняння 1.3.1, де V – вхід напруги, α і γ (гама) є константами для кожного монітора, $\in$ встановлює рівень чорного рівня (яскравість) для монітора, *I* - генерована інтенсивність [47].

$$I = \alpha (V + \in)^{\gamma} \qquad (1.3.1)$$

Потім ця інтенсивність перетворюється на вихід електричної напруги, який буде використовуватися пристроями виведення для відображення вмісту фреймбуфера.

Однак, як правило, гама-корекція не застосовується на етапі відображення, але всі кольори (текстури, кольори вершин, вкладки шейдерів) вже зберігають гамма-виправленими. Це дозволяє найкраще використовувати наявну точність з плаваючою точкою в графічному конвеєрі [50].

У сучасних графічних рушіях для візуалізації фотореалістичних і нефотореалістиних зображень використовуються різні алгоритми вузіалізації такі як: високодинамічне відображення діапазону (High Dynamic Range Rendeting), алгоритм «Scanline», алгоритм «Ray Tracing», алгоритм «Ray Casting», об’ємний рендеринг (Volume Rendering), z-буферизація. Кожен з алгоритмів має свої переваги та недоліки при візуалізації, детальнішу інформацію про основні

алгоритми візуалізації відображено у розділі 4. У даній роботі буде продемонстровано реалізації одного із основних алгоритмів візуалізації зображення «Ray Casting».

1.4 Програмування GPU

Розрахунки, виконані на етапі геометрії та на етапі растерізації, як описано в розділі 1.3, використовувались в графічні прискорювачі до виходу графічного пристрою NVIDIA Geforce3 в 2001 році. Цей графічний пристрій зміг запускати невеликі програми, щоб модифікувати вершинні та фрагменті дані на GPU. Такі програми називаються шейдерами вони були представлені з DirectX 8 (вони також були доступні як розширення OpenGL). Завдання шейдерів (їх також називають програмами) – замінити жорстку провідну модель трансформації та модель освітлення GPU (так звану фіксовану функцію конвеєру) на програмовану.

У цьому розділі представлено три типи шейдерів(вершинні, фрагментні, геометричні), які проілюстровані на рисунку 1.5 на якому зображена розширена версія графічного конвеєра що працює на GPU, частини фіксованої функції конвеєра була замінені програмованим ступенем.

Програми вершин застосовуються до кожної вершини моделі та використовуються для зміни їх атрибутів. Сюди входить перетворення векторів або обчислення моделей освітлення. Так звані єдині параметри – це додаткові входи до програми, значення яких постійні для кожного виклику шейдера. Одне обмеження вершинних програм полягає в тому, що вони мають доступ лише до даних однієї вершини. Вилучення даних іншої вершини в моделі неможливо. Крім того, програма вершин не може створювати нові вершини.

Приклади використання включають:

- ефекти лінзи, як лінзи з риб'ячих очей;

- скручення і згинання предметів;
- рух тканини або водних поверхонь;
- генерація координат текстур для вершинних програм.

Рисунок 1.5 – Розширена версія графічного конвеєра

Програми геометрії були введені разом із моделлю Шейдер 4.0, як описано в підрозділі 1.4. Програма геометрії викликається для кожної примітивної групи, наприклад трикутної смужки або списку точок, і здатна генерувати нові примітиви (включаючи нові вершини) і навіть може конвертувати тип примітиву. Приклади

використання включають генерацію шумів тіні на графічному процесорі та процедурну генерацію моделей.

Програми з фрагментами застосовуються до кожного фрагмента, що генерується перетворенням сканування, як описано в підрозділі 1.3 в етапі растеризації. Завдання програми фрагмента – взяти атрибути фрагмента та рівномірні параметри як вхідні дані та обчислити кінцевий колір для цього фрагмента який буде записаний до цілі візуалізації. Атрибути фрагмента є інтерпольованими атрибутами пов'язаних вершин під час сканування перетворення.

Приклади використання включають:

- розрахунок точних моделей освітлення;
- моделювання властивостей багатошарових поверхонь;
- ефекти після обробки, такі як світіння та глибина різкості.

З кожною новою версією графічних прискорювачів та графічних API, таких як DirectX та OpenGL, розширюється діапазон доступних інструкцій для програм вершин та фрагментів. Версія набору інструкцій також називається шейдерною моделлю. Наступний перелік надає чудові нові функції, представлені з новими моделями шейдерів.

- Shader Model 1.0: Базова модель, введена першими вершинними та фрагментними програмами;
- Shader Model 2.0: збільшена максимальна кількість інструкцій у вершино-фрагментарних програмах; введені інструкції з розгалуженням; необмежена кількість інструкцій з текстурою; збільшена кількість тимчасових і постійних реєстрів;
- Shader Model 3.0: Нескінченна довжина програм вершин та фрагментів, повна підтримка підпрограм, циклів та гілок, отримання текстур у вершинних програмах, одночасний вихід на кілька цілей візуалізації (MRT);

- Shader Model 4.0: Впровадження шейдерів з геометрії, що дозволяють генерувати примітиви на GPU, уніфікація шейдерів, що усунула відмінності між піксельними та вершинними шейдерами, збільшена кількість прикладів текстур (128), збільшена кількість тимчасових реєстрів.

Мова програмування, що використовується для реалізації шейдерів – це мова програмування низького рівня, дуже схожа на мову асемблера, що використовується для програмування процесора. Тому головним завданням шейдерних мов високого рівня є надання простішої у користуванні мови програмування для реалізації шейдерів.

Завдяки існуванню різних постачальників графічних пристроїв та графічних API, були розроблені різні шейдерні мови. Найбільш релевантні шейдерні мови високого рівня представлені в наступних розділах.

У лістингу 1.1 наведено приклад вершинного шейдера DirectX:

Лістинг 1.1 – Вершинний шейдер на DirectX

```
vs.1.1;
dp4 oPos.x, v0, c4;
dp4 oPos.y, v0, c5;
dp4 oPos.z, v0, c6;
dp4 oPos.w, v0, c7;
mov oT0.xy, v7;
mov oD0, v5;
```

Шейдерна мова OpenGL також відома як GLSL [15] або glslang може використовуватися лише графічним OpenGL API. GLSL був визначений комітетом з архітектурного огляду [1] під час специфікації OpenGL 2.0. У лістингу 1.2 нижче показана програма вершин та фрагментів, написана в GLSL для ефекту дифузії:

Лістинг 1.2 – Вершинна та фрагментна программа на GLSL

```
// --------------------------------
// Вершинна програма
// --------------------------------
```

```
varying vec3 normal;
varying vec3 vertex_to_light_vector;
void main(){
// Трансформація вектерної проекції на простір.
gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
// Трансформація нормалів на модель огляду у просторі.
normal = gl_NormalMatrix * gl_Normal;
// Трансформація вертексної позиції на проекцію в просторі.
vec4 vertex_in_modelview_space = gl_ModelViewMatrx * gl_Vertex;
//Отримання вектору з вертексної позиції на позицію освітлення
vertex_to_light_vector = vec3(gl_LightSource[0].position -
vertex_in_modelview_space);
}// ---------------------------------
// Фрагментна програма
// ---------------------------------
varying vec3 normal;
varying vec3 vertex_to_light_vector;
void main(){
// Визначення коефіцієнтів матеріалів.
const vec4 AmbientColor = vec4(0.1, 0.0, 0.0, 1.0);
const vec4 DiffuseColor = vec4(1.0, 0.0, 0.0, 1.0);
vec3 normalized_normal = normalize(normal);
vec3                normalized_vertex_to_light_vector                =
normalize(vertex_to_light_vector);
float  DiffuseTerm  =  clamp(dot(normal,  vertex_to_light_vector),  0.0,
1.0);
gl_FragColor = AmbientColor + DiffuseColor * DiffuseTerm; }
```

Шейдерна мова високого рівня була введена Microsoft [18] з DirectX 9.0 і називається HLSL [17]. HLSL був випущений до GLSL і пізніше розширений, щоб відповідати набору функцій GLSL. Як і у GLSL, HLSL пов'язаний з одним графічним API, а саме Direct3D.

У лістингу 1.3 нижче показана програма вершин та фрагментів, написана HLSL для ефекту дифузії:

Лістинг 1.3 – Вершинна та фрагментна программа для ефекту дифузії

```
float4x4 matWorldViewProj;
float4x4 matWorld;
float4 vecLightDir;
struct VS_OUTPUT {
float4 Pos : POSITION;
```

```
float3 Light : TEXCOORD0;
float3 Norm : TEXCOORD1; };
// Вершинна программа
VS_OUTPUT VS(float4 Pos : POSITION, float3 Normal : NORMAL){
VS_OUTPUT Out = (VS_OUTPUT)0;
Out.Pos = mul(Pos, matWorldViewProj);
Out.Light = vecLightDir;
Out.Norm = normalize(mul(Normal, matWorld));
return Out;}
// Фрагментна програма
float4 PS(float3 Light: TEXCOORD0, float3 Norm : TEXCOORD1) : COLOR
{
float4 diffuse = {1.0f, 0.0f, 0.0f, 1.0f};
float4 ambient = {0.1f, 0.0f, 0.0f, 1.0f};
return ambient + diffuse * saturate(dot(Light, Norm));
}
```

Мова програмування Cg [32] (Cg означає C для графіки) була розроблена NVIDIA і може використовуватися з обома основними графічними API: OpenGL та Direct3D [11]. Cg має той самий синтаксис, що і HLSL. У лістингу 2.4 наведену просту програму вершин та фрагментів, написану на Cg:

Лістинг 1.4 – Вершинна та фрагмента программа на Cg

```
struct appdata
{
float4 Position : POSITION;
float3 Normal : TEXCOORD0;
};
struct vfconn
{
float4 HPOS : POSITION;
float3 Normal : TEXCOORD0;
};
// Вершинна программа
vfconn main(appdata IN,
uniform float4x4 WorldViewMatrixIT,
uniform float4x4 WorldViewProjectionMatrix)
{
vfconn OUT;
OUT.HPOS = mul(WorldViewProjectionMatrix, IN.Position);
OUT.Normal = mul(WorldViewMatrixIT, float4(IN.Normal,1)).xyz;
return OUT;
}
```

```
// Фрагмента программа
fragout main(vfconn IN) {
fragout OUT;
float grey = clamp(dot(IN.Normal, float3(0,0,1)), 0, 1);
OUT.col = float4(grey, grey, grey, 1);
return OUT; }
```

Відмінною особливістю Cg є те, що мова дозволяє використовувати шейдерні інтерфейси як єдині вхідні параметри [28]. У лістингу 1.5 наведено приклад який визначає інтерфейс для всіх вогнів і дає точкову реалізацію цього інтерфейсу:

Лістинг 1.5 – Приклад використання шейдерних інтерфейсів

```
// Інтерфейс для всіх вогнів.
interface ILight
{
float3 illuminate(float3 p, out float3 L);
};
// Реалізація точкового світла інтерфейсу ILight.
struct PointLight : ILight
{
float3 Plight, Clight;
float3 illuminate(float3 P, out float3 L)
{
L = normalize(Plight - P);
return Clight;
}
};
```

Потім ці інтерфейси можуть бути реалізовані структурами, які можна перемикати під час виконання.

1.5 Висновки до першого розділу

В даному розділі було розглянуто візуалізацію в режимі реального часу. Розглянуто графічне програмне забезпечення та які API воно може використовувати. Описано етапи роботи графічного конвеєра які необхідні для

отримання 2D зображення на екрані. Також коротко оглянули яким чином відбувається програмування GPU та які інструменти використовуються для написання вертексних та фрагментних програм на GPU. Описано основні алгоритми візуалізації зображенням.

З розділу стає зрозуміло складність побудови графічних рушіїв та складність реалізації графічних ефектів, так як в кожного з них є свої переваги та недоліки. Перед побудовою графічного рушію головною ціллю стає визначення його призначення та який функціонал інструментів він буде у собі мати. На даний момент є великий вибір графічних рушіїв на ринку від дуже простих до надзвичайно комплексних які мають в своєму арсеналі великий набір інструментів, що дає змогу використовувати їх у різних областях, починаючи від розваг і закінчуючи складними інженерними рішеннями які мають вагомий внесок в суспільство.

2 ДОСЛІДЖЕННЯ ТА АНАЛІЗ РОБОТИ ГРАФІЧНОГО РУШІЯ

2.1 Обробка шейдерів та ефектів

Передача: Сума всіх команд обробки, необхідних для перетворення набору геометрій з однаковими текстурами та програмами GPU, в один конкретний об’єкт рендерингу.

Локальний мультипропуск: процес візуалізації однієї геометрії кілька разів з різними передачами даних.

Глобальний мультипропуск: процес візуалізації геометрій повної сцени кілька разів з різними передачами даних.

Однією з вимог користувачів графічних рушіїв є підтримка завжди найновіших ефектів рендерингу на ринку. Тому час від винаходу ефекту до інтеграції в механізм візуалізації повинен бути коротким. Це дозволяє використовувати рішуй для швидкого прототипування. Ще одна вимога полягає в тому, що графічний рушій повинен бути добре розроблений і мати чистий API, який використовуватимуться іншими розробниками. Це змушує розробників графічного рушія структурувати рушій таким чином, що інтеграція додаткових методів візуалізації може бути виконана чистим способом.

Графічний рушій повинен не тільки дозволяти використовувати нові програми вершин та фрагментів, але також повинен забезпечувати чистий та заздалегідь визначений спосіб впровадження абсолютно нових методів візуалізації, що мають глобальний вплив на обробку всієї сцени.

Один із підходів до організації шейдерів в механізмі візуалізації – це індивідуально – програмний підхід [39], де існує колекція шейдерних файлів. Кожен з цих файлів є вершиною або фрагментною програмою, що використовується для одного пропуску візуалізації або декількох об'єктів.

Перевагою такого підходу є те, що легко додати нову вершинну або фрагментну програму до списку. Однак недоліком цього підходу є те, що надання лише шейдерних файлів зазвичай недостатньо для реалізації нового ефекту візуалізації. Крім того, потрібне управління окремими пропусками разом з відповідними станами візуалізації, що призведе до спеціального вихідного коду для кожного нового ефекту візуалізації, реалізованого відповідно до індивідуально – програмного підходу.

Інший підхід полягає у використанні так званих файлів ефектів, формат яких надається однією з існуючих графічних бібліотек: графічною бібліотекою Майкрософт [30], або графічною бібліотекою CgFX NVIDIA [11]. При такому підході всі необхідні вершинні та фрагментні програми одного ефекту візуалізації розміщуються в єдиний файл і структуруються у проході візуалізації. Крім того, файли ефектів дозволяють уточнити необхідні стани візуалізації для передачі даних. Таким чином, впроваджуючи новий ефект візуалізації, який потребує локального мультипропуску, користувач повинен просто додати новий файл ефекту до колекції.

Використовуючи інструкції з регулювання потоку в коді шейдера, динамічно визначаючи, які елементи шейдера потрібно обчислити.

Використання попередньої обробки коду шейдера разом із блоками #ifndef.

Недоліком цього підходу є те, що він швидко ускладнюється об'єднанням усіх можливих методів візуалізації в єдиний вихідний код шейдера. Також розширення цього шейдера може легко порушити функціональність іншого елемента шейдера.

Зворотний підхід до шейдерів Uber – це мікро шейдери, представлені Шоном Харгрівсом. З Micro Shaders загальна функціональність розбита на невеликі фрагменти коду шейдера [17], які об'єднані додатком для створення потрібного коду шейдера. Проблема такого підходу полягає в тому, що фрагменти вихідного

коду можуть вступати в конфлікт один з одним, якщо вони використовують однакові регістри або те саме ім'я для змінних.

Аналогічний підхід до Micro Shaders – це абстрактно затінені дерева, представлені McGuire [17]. За допомогою цього методу елементи шейдера виражаються у вигляді атомів, які визначаються декларацією, набором структурних / глобальних визначень функцій та тілом. Тіло містить реальний шейдерний код, який можна записати будь-якою доступною мовою затінення високого рівня. Потім користувач системи може створити дерево з цих вузлів, яке потім аналізується алгоритмом для генерування повного коду шейдера. Впроваджена система робить неможливими розбіжності типів у шейдерах. Обмеженнями абстрактно затінених дерев є:

- Створені шейдери можуть перевищувати інструкції та реєструвати обмеження кількості наявних GPU.
- Реалізований компілятор не містить оптимізацій для всієї програми.

Цей, також здатний вирішити глобальний мультипропуск, необхідний для глобальних ефектів, таких як затінення, відображення та заломлення.

2.2 Структури просторових даних

У цьому розділі описані структури просторових даних, які використовуються для запобігання непотрібної роботи GPU. Непотрібне в цьому контексті означає, що робота не сприяє остаточному образу і тому може бути опущена. Представлені методи мають спільне те, що вони використовують ієрархічну структуру для зберігання об’єктів сцени. Потім ця ієрархічна структура тестується на оглядовій пісочниці, щоб знайти список видимих об'єктів сцени. Усі об’єкти, не видимі в пісочниці, що переглядаються, не будуть надані. Це призводить до підвищення продуктивності графічного рушія.

Дерево октантів [48] структурує дані сцени як дерева. Кожен вузол цього дерева може мати до восьми нащадків і являє собою поле з тривимірного простору. Кожен вузол також має перелік об'єктів сцени, які повністю або частково лежать в області вузла. На рисунку 2.1 показаний дерево октантів сцени, що містить два об’єкти. На лівому зображенні зображено поділ простору на правому зображено внутрішню структуру дерева.

Дерево октантів будується, починаючи з кореневого вузла, який являє собою обмежувальне поле всієї сцени. З кожним об'єктом, доданим до дерева октантів, обмежуючий об'єм об'єкта тестується на вікні вузла. Якщо обмежуючий об'єм перетинає цей ящик, нащадки вузла запитуються на тест перетину. Це робиться до тих пір, поки нащадки не стануть занадто маленькими, щоб розколотись далі, або обмежуючий об'єм предмета не перевищить коробку нащадка.

Існує багато різних реалізацій концепції дерева октантів. З деякими з них дозволяється додавати предмети до внутрішніх вузлів дерева, а для інших дозволяється лише додавати об'єкти до вузлів листя.

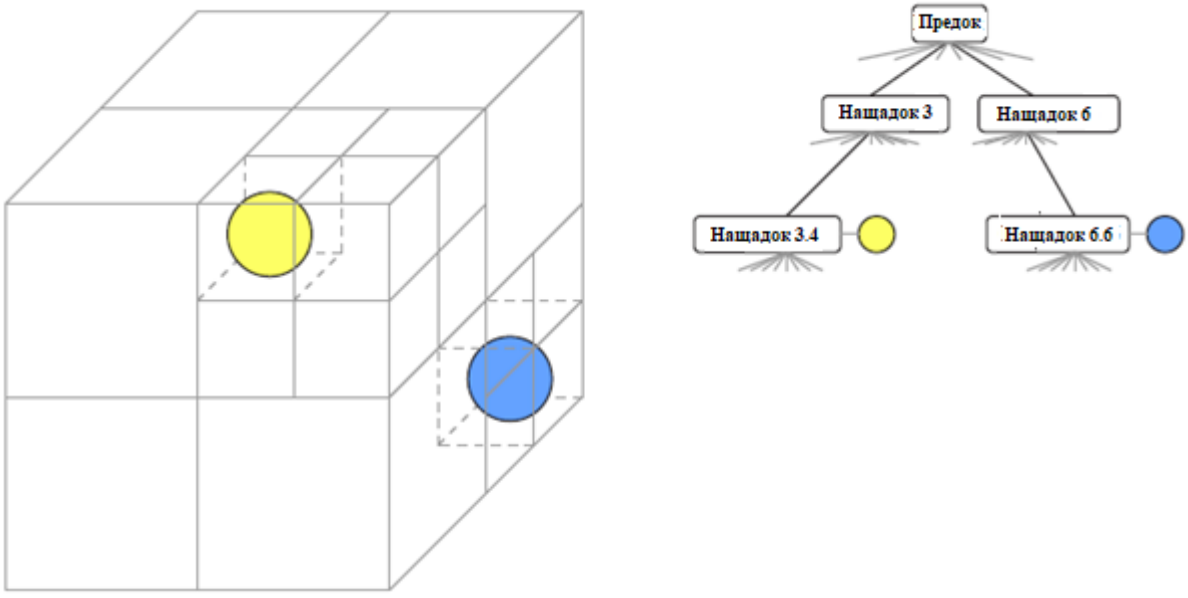


Рисунок 2.1 – Дерево октантів сцени, що містить два об’єкти

Крім того, дерево октантів може розділити додані об’єкти, якщо вони перетинають більше ніж одне поле або дерево зберігає посилання на об'єкт для кожного вузла, який містить цей об'єкт.

K – вимірне дерево [48] схоже на дерево октанів, представлене в попередньому розділі, за винятком того, що воно є двійковим деревом. Тільки площини, перпендикулярні одній із осей системи координат, використовуються як площини розщеплення. Рухаючись вниз по дереву, осі, які використовуються для вибору площин розщеплення, проїжджаються наскрізь. Як на рисунку 2.2, перша площина перпендикулярна осі Y, а друга площина перпендикулярна до осі Z. Зображення зліва показує поділ простору, а праве зображення – внутрішню структуру дерева.

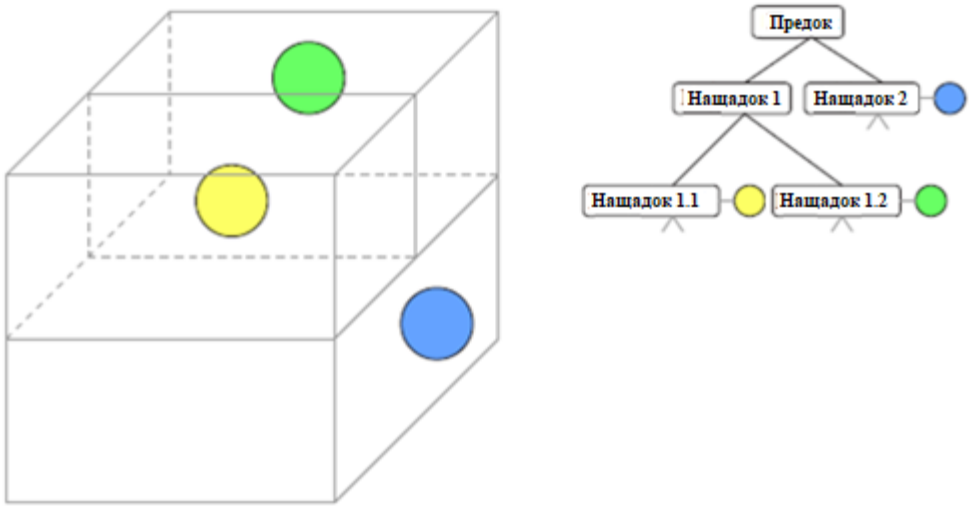


Рисунок 2.2 – k-вимірне дерево сцени, що містить три об’єкти

K – вимірне дерево будується, починаючи з порожнього кореневого вузла, який являє собою обмежувальне поле всієї сцени. Цей ящик розділений двома дітьми на нижній та верхній напівбокс. Наступний розкол робиться по іншій осі. Існує безліч алгоритмів, щоб вирішити, де розмістити площину розщеплення і коли зупинити цей процес. Один з цих алгоритмів - це евристика площі поверхні [27].

Нижче наведено два з критеріїв припинення процесу розщеплення, які використовуються багатьма алгоритмами:

- процес зупиняється, коли глибина графіка коробки вище або дорівнює фіксованій константі;
- процес повторюється до тих пір, поки кожен розділений ящик не буде містити певну кількість об'єктів.

Алгоритм знаходить положення площини розщеплення, мінімізуючи функцію витрат, яка оцінює вартість променя, що проходить через kd-дерево.

2.3 Граф сцени

Граф сцени – це графік, який впорядковує об’єкти сцени в ієрархічному порядку. На відміну від дерева октантів та k – вимірного дерева, графи сцени зберігають не лише геометрію предметів. Оскільки граф сцени повинен зберігати всі відповідні дані сцени, він також зберігає перетворення, параметри матеріалу, текстури, обмежувальні об'єми, світло, камери та ефекти кожного об'єкта на сцені, як показано на рисунку 2.3.

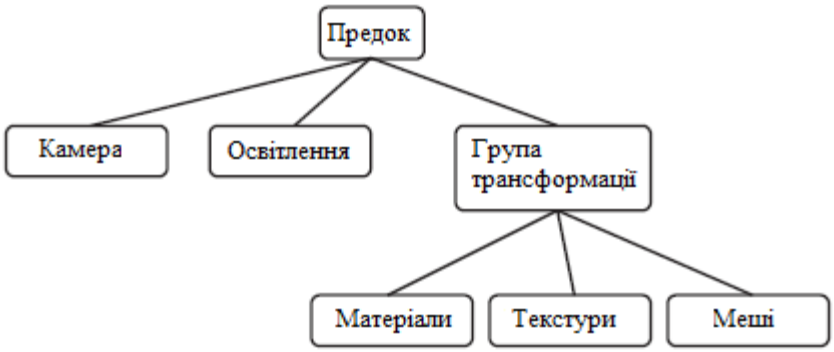


Рисунок 2.3 – Приклад графа сцени, що містить різні типи вузлів

Щоб відобразити сцену, кожен вузол у графіку повинен бути пройдений. Цей обхід зазвичай потребує певного стану, щоб запам'ятати дані візуалізації, який вузол змінив (наприклад, візуалізація сталі та перетворення). Цей об'єкт стану передається кожному вузлу, щоб поточний вузол міг змінити стан. Об'єкт стану також може повернути зміни, які застосував вузол. Кожен відвідуваний вузол може змінити поточний стан графічного пристрою, наприклад вузол матеріалу застосовуватиме його параметри до графічного пристрою за допомогою графічного API, наприклад OpenGL. Якщо відвідувач перетинає вузол геометрії, викликається виклик візуалізації для цієї геометрії.

Важливою відмінністю між реалізаціями сценографів є те, як розповсюджуються зміни стану в вузлах групи. Деякі сценографи поширюють зміни на нащадків вузла, а також на правий сусідній вузла. Це дозволяє легко зібрати сценограф, вузли якого мають однакові атрибути. Ці атрибути можуть бути розміщені в лівій верхній частині графіка і поширюватися на весь графік під час обходу. Інші сцени дозволяють поширювати стан лише дітям вузла. Це полегшує реалізацію паралельних переходів.

Як і в деяких реалізаціях сценографа, не кожна інформація повинна зберігатися як явний вузол у графі. Наприклад така інформація, як матеріали та текстури, може зберігатися як властивості іншого вузла. Ці властивості часто називають вузловими компонентами.

Для зберігання перетворень предметів один до одного та мінімізації їх позицій, сценограф також передбачає особливий вид групового вузла. Цей груповий вузол (часто його називають перетворювальною групою) зберігає матрицю перетворення, яка застосовується до всіх її нащадків під час обходу.

Якщо значення трансформації групи трансформацій змінюється, це значення слід поширювати на своїх дітей. Але оскільки ця зміна визнає недійсним

обмежувальний об'єм трансформаційної групи та її нащадків, обмежуючі обсяги батьків до кореневого вузла також повинні бути оновлені.

Однією з головних причин використання графу сцени є те, що його ієрархічний макет може бути використаний для підвищення продуктивності візуалізації шляхом відсікання. Оскільки кожен вузол геометрії має обмежуючий об'єм, який інкапсулює його вектори позиції, з вузлів графіка може бути побудована ієрархія обмежувального об'єму. На рисунку 2.4 видно ієрархію обмежуючих обсягів (праворуч) невеликого сценографа (зліва). Обмежуючий об'єм групового вузла інкапсулює всі обмежуючі обсяги його дочірніх ділянок. Під час обходу сценографа для візуалізації обмежуючий об'єм кожного відвідуваного вузла тестується на огляд фрустрації камери. Якщо вузол знаходиться за межами цього фрустума, сам вузол та всі його нащадки вважаються бракованими і не проходять далі [20]. Використовуючи цей алгоритм (також його називають ієрархічним видом-фрустуванням) загальна ефективність візуалізації може бути збільшена.

Оскільки сценограф забезпечує лише логічну організацію вузлів, додаткова структура просторових даних могла б забезпечити кращі результати відсікання (щодо продуктивності процесу відсікання), ніж ієрархія обмежувального обсягу сценографу. Приклади таких структур даних наведені в розділі 2.4.

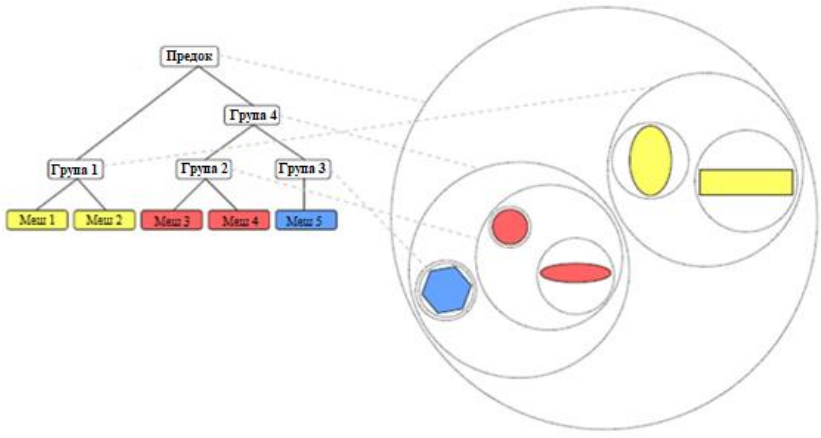


Рисунок 2.4 – Обмежена ієрархії обсягу невеликого сценографу

Важливим аспектом сценографу є те, як можна обмінюватися даними між його вузлами. Це особливо важливо, якщо сценограф містить тисячі сітчастих вузлів, які мають однакову геометрію.

Один популярний підхід – дозволити вузлам мати декількох батьків. Таким чином, до даних спільного підграфа можна посилатися кілька разів, як проілюстровано на рисунку 2.5. Оскільки ця концепція ускладнює обробку сценографу, деякі бібліотеки дозволяють використовувати спільний доступ лише до листкових вузлів. Або вони вводять спеціальний клас вузлів, які обробляють цю функцію. Наприклад Java 3D вводить вузол посилання на загальний груповий вузол [20], де вузол посилання вказує на один екземпляр спільного групового класу. Простий сценарій із використанням цієї функції проілюстрований на рисунку 2.6.

Інші бібліотеки сценографа інкапсулюють дані у так званих компонентах вузла, на які можна посилатися декількома екземплярами вузлів.

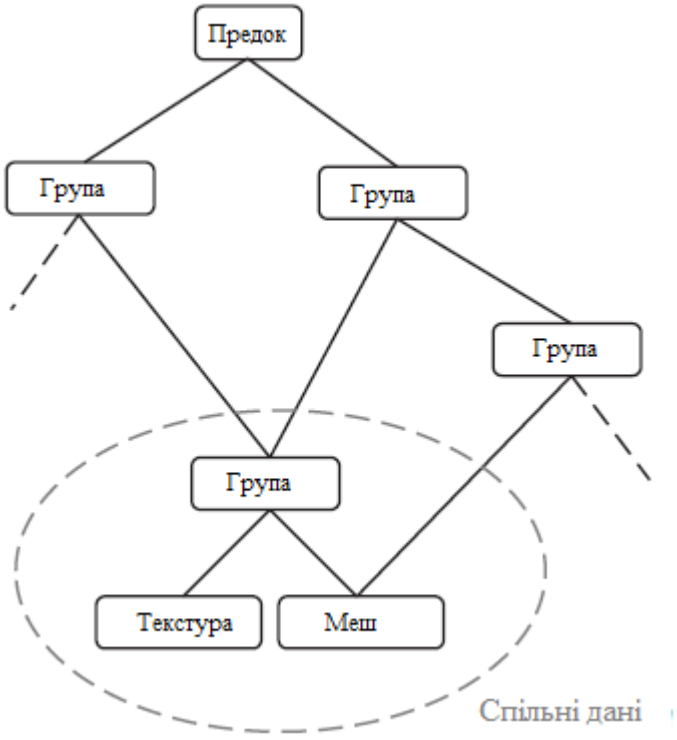


Рисунок 2.5 – Граф сцени, що містить вузли з кількома групами

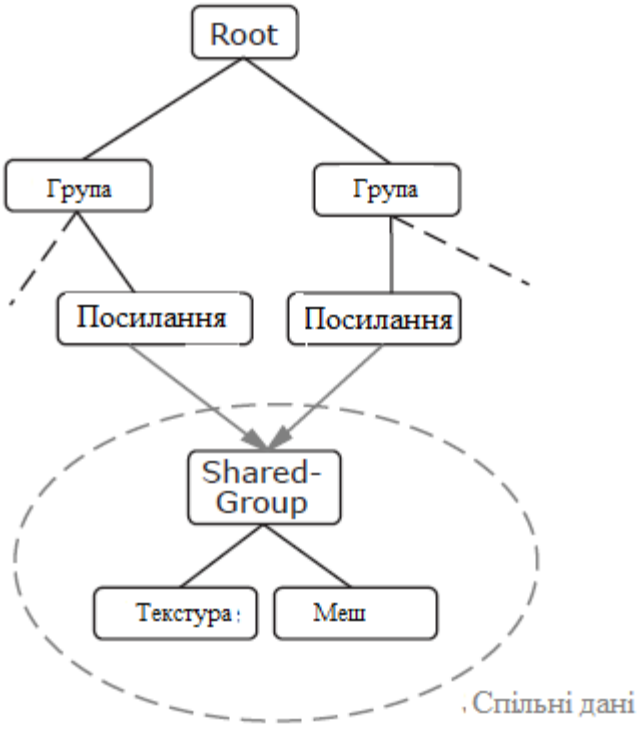


Рисунок 2.6 – Сценограф із використанням вузлів та спільною групою.

Компоненти вузлів не походять від класу вузлів і тому не можуть бути безпосередньо інтегровані в сценограф, а лише внутрішньо посилаються на вузли. Такий підхід спрощує обробку сценографу. Його недолік полягає в тому, що для кожного логічного екземпляра даних повинен бути побудований вузол, який посилається на спільний компонент вузла, викликаючи накладні витрати на обході. На рисунку 2.7 зображено сценограф із використанням компонентів вузла для інкапсуляції своїх даних.

Популярна бібліотека сценографу – Open Inventor – API сценографа з відкритим кодом, написаний на C ++. Він надається від SGI та побудований поверх OpenGL. Тому він підтримується на широкому спектрі платформ. Він також визначає власний формат файлу.

Інший доступний сценографічний API – це Java 3D, який складається з бібліотеки класів Java і спочатку був розроблений компанією Sun Microsystems у 1997 році. З 2004 року Java 3D стала проектом з відкритим кодом. З версії 1.4

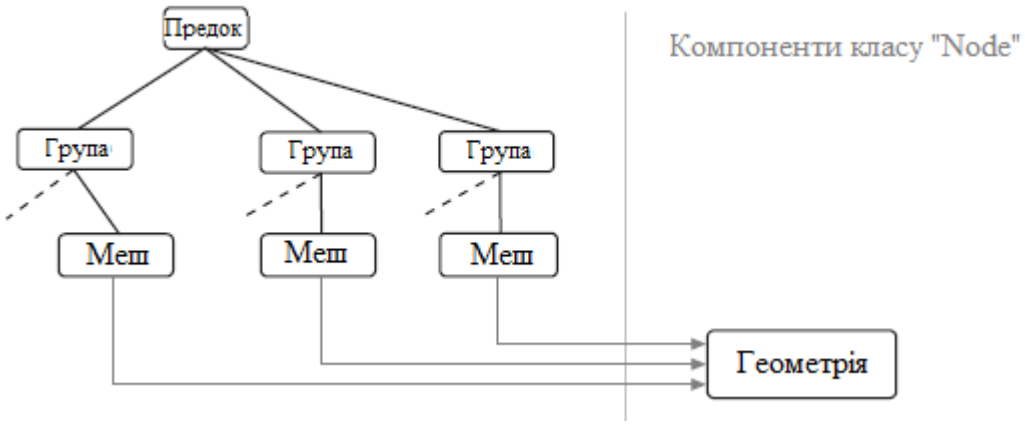


Рисунок 2.7 – Сценограф

інший доступний сценографічний API – це Java 3D, який складається з бібліотеки класів Java і спочатку був розроблений компанією Sun Microsystems у 1997 році. З 2004 року Java 3D стала проектом з відкритим кодом. З версії 1.4 (доступна з 2006 року) розробка шейдерів підтримується і Java 3D. На відміну від Open Inventor, прямий доступ до базового графічного API (OpenGL або Direct3D) не призначений. На даний момент Java 3D доступний для Microsoft Windows, Linux, Mac OS та Solaris.

OpenSG – система візуалізації в режимі реального часу, яка містить графік сцени з підтримкою багатопотокових та кластеризованих підключень. Його автори стверджують, що OpenSG дуже розширюється і працює в різних системах Unix та Microsoft Windows.

OpenSceneGraph – ще один високопродуктивний інструментарій для 3D графіки. У ньому представлений багатосхемовий графік сцен. В основному використовується в сферах візуального моделювання, іграх, наукових візуалізаціях

та віртуальній реальності. Він написаний у стандарті C ++ і використовує OpenGL для роботи в Microsoft Windows, Mac OS, Linux, Solaris, HP-Ux, AIX та FreeBSD.

2.4 Шаблони проектування

Шаблони дизайну в контексті інженерії програмного забезпечення описують прості та елегантні рішення проблем в об'єктно-орієнтованому програмуванні. Використання моделей дизайну, представлених у цьому розділі, призведе до гнучких, модульних, багаторазових та зрозумілих застосувань макету коду далі наведемо приклади найбільш часто застосовуваних шаблонів проектування [42].

Абстрактна фабрика (AbstractFactory) надає інтерфейс, який клієнти можуть використовувати для створення родин пов'язаних об'єктів, не знаючи про конкретний клас, який стає інстальованим. Клієнт бачить лише інтерфейси (AbstractProducts) створених об'єктів. Тільки класи (ConcreteFactory), що реалізують інтерфейс Abstract-Factory, знають, які класи вони мають інстанціювати (ConcreteProducts).

Шаблон прототип(Prototype) допомагає створювати екземпляри конкретного класу, викликаючи спеціальний метод існуючого об'єкта (Prototype). Цей метод (часто його називають Clone ()) створює копію прототипу і повертає цей новий екземпляр клієнту. Клієнту не потрібно знати, який конкретний клас створюється втіленням (ConcretePrototype) інтерфейсу прототипу.

Шаблон проектування одинак (Singleton) повинен використовуватися, якщо в системі є дозволений лише один екземпляр класу. Сам клас несе відповідальність за надання глобальної точки доступу до свого екземпляра. Це досягається шляхом надання методу класу, який створює глобальний екземпляр, якщо цього ще не було зроблено, і повертає його виклику. Крім того, конструктор цього класу

оголошується приватним, щоб переконатися, що ніхто інший не може зробити його примірник.

За допомогою шаблону адаптер (Adapter) система маж можливість реалізувати відомий інтерфейс (Target), який використовується клієнтом, загортаючи інший (невідомий клієнту) інтерфейс. За допомогою цього шаблону дизайну можна працювати разом, з клієнтами які мають несумісні інтерфейси. Адаптер також відомий як обгортка.

Шаблон проектування міст (Bridge) відокремлює абстракцію від його реалізації, розміщуючи їх в окремі ієрархії класів. Це має ту перевагу, що обидва можуть змінюватись незалежно. Клас абстракції містить посилання на екземпляр (ConcreteImplementor), що реалізує інтерфейс Implementor. Класи, які розширюють клас абстракції (Refined – Abstractions), також використовують це посилання для досягнення своєї функціональності.

За допомогою шаблону замісник (Proxy) клієнт використовує заповнювач місця для іншого об'єкта. Цей сурогат контролює доступ до іншого об'єкта. Популярним прикладом проксі – схеми є опорні підрахунки покажчиків (їх також називають смарт-покажчиками). Тут клієнт має лише доступ до екземпляра інтелектуального вказівника (Proxy), який контролює знищення реального вказівника шляхом підрахунку посилань на нього.

Шаблон команди (Command) інкапсулює клієнтські запити як об'єкти, які можуть бути, наприклад, зберігатися, стояти у черзі та реєструватися. Ці об'єкти (ConcreteCommand) підтримують інтерфейс Command та визначають прив'язку між об'єктом Receiver та дією. Завдання клієнта – створити цей об'єкт команди та встановити його приймач.

З допомогою шаблону спостерігач (Observer) список спостережуваних об'єктів може зареєструвати себе так званим предметним об'єктом. Щоразу, коли

суб'єкт змінює свій стан, він повідомляє спостерігачів, які, в свою чергу, синхронізують свій внутрішній стан із станом суб'єкта.

За допомогою шаблонного методу (Template method) визначається глобальна структура алгоритму в методі об'єкта. Цей метод називає інші абстрактні методи, підкласи яких об'єкт може переосмислити, щоб забезпечити конкретну поведінку реалізованого алгоритму.

Шаблон відвідувач (Visiter) дозволяє відокремлювати операції від самих вузлів, які виконуються на елементах об'єктної структури. Тому введення нової операції не примушує змін до класів вузлів, над якими вона працює. Операції надаються конкретними класами (ConcreteVisitor) абстрактного батьківського класу (Visitor). Клас відвідувачів оголошує операцію для кожного класу вузлів. Клієнт передає відвідувача об’єктній структурі. Якщо елемент структури приймає відвідувача, він викликає відповідний метод для свого класу і передає себе як параметр цього методу. Бетонний відвідувач виконує свою конкретну операцію над елементом і після цього відвідувач направляється на наступний елемент конструкції. Нова функціональність вводиться шляхом визначення нових підкласів відвідувачів.

2.5 Порівняння доступних графічних рушіїв

У цьому розділі подано порівняння між доступними рушіями візуалізації. У цьому списку містяться лише вільні доступні системи візуалізації, оскільки загалом не можна давати внутрішню інформацію про комерційні двигуни. Двигуни візуалізації порівнюються, виходячи з тем, розглянутих у цій главі.

Об'єктно-орієнтований графічний візуалізатор (OGRE) – орієнтований на сцену, в реальному часі, рушій 3D – візуалізації. Програмне забезпечення є

відкритим кодом. Перша версія програмного забезпечення була випущена в лютому 2005 року[35]. Має наступні характеристики:

- підтримує такі платформи як: Windows, Linux, Mac OS X;
- підтрика графічних API: OpenGL, Direct3D;
- підтримка шейдерів: Assembler, Cg, GLSL, HLSL;
- підтрика обробки ефектів: надає власний формат файлу матеріалів та ефектів, подібний до CgFX та HLSL, який підтримує визначення декількох проходів, рендерингу та використаних текстур;
- підтримка структури просторових даних: Немає вбудованих структур, але забезпечує плагін, що дозволяє користувачеві реєструвати власні реалізації в двигуні. Приклад плагінів забезпечує функціональність дерева Octree та BSP;
- підтримку сценографу: надає лише один клас вузлів для зберігання будь-яких даних, які користувач хоче структурувати як графік. Вузлами обробляється менеджер сцени, який може бути реалізований за допомогою графіків, таких як octrees або BSP – дерева.

NVIDIA Scene Graph кросс – платформенний граф сцени з повної підтримкою затінення через найновішу технологію візуалізації для отримання максимально реалістичної картинки [5]. Має наступні характеристики:

- підтриму є такі платформи як: Windows, Win64, Linux;
- підтримує графічні API: OpenGL;
- підтримка шейдера: Cg, CgFX;
- поводження з ефектами: ефекти та матеріали безпосередньо призначаються об’єктам на сцені. Ніякої спеціальної підтримки глобальної мультидоступності не надається;
- граф сцени: Сценограф підтримує кілька типів вузлів, таких як групи, світильники, камери та сітки. Обмін даними між вузлами може бути здійснено шляхом дозволу декількох вузлів безпосередньо посилатися на спільні дані або

повторно використовувати один і той же вузол кілька разів. Це можливо, оскільки всі вузли підтримують декількох батьків. NVSG використовує концепцію проходу відвідувачів для виконання таких операцій, як відсікання та візуалізація на вузлах.

Irrlicht – це графічний рушій із відкритим кодом, написаний на C ++. Irrlicht відомий своїми невеликими розмірами та сумісністю з новими та старими апаратними засобами, простотою навчання. Має наступні характеристики:

- підтримує такі платформи: Windows, Linux, Mac OS;
- підтримує графічні API: OpenGL, Direct3D та два програмних рендери;
- підтримка шейдера: Асемблер, HLSL, GLSL;
- поводження з ефектами: ефекти та матеріали безпосередньо призначаються об’єктам на сцені. Ніякої спеціальної підтримки глобальної мультидоступності не надається. Ефект тіньової гучності, а також ефекти прозорості чітко кодуються в процесі візуалізації;
- підтримка структури просторових даних: реалізовано впровадження указу. Але це лише особливий тип сценографічного вузла, а не інший погляд на цілі дані сцени;
- граф сцени: Сценограф підтримує освітлення, сітки, камери, рекламні щити, горизонти, місцевості та октриси, використовуючи спеціальні класи вузлів. Тіньові томи інтегруються в сценограф, використовуючи і власний клас вузлів.

OpenSG є системою графічних сцен для створення графічних програм у режимі реального часу, наприклад. для додатків віртуальної реальності. Він розроблений на основі принципів відкритого коду, з ліцензією LGPL та може вільно користуватися. Основними його особливостями є розширена підтримка багатопотокової та кластеризованої роботи [19]. Має наступні характеристики:

- підтримує платформи: Windows, Linux;
- підтримує графічні API: OpenGL;
- підтримка шейдера: Асемблер, GLSL;

- обробка ефектів: до сценографу додаються матеріальні вузли, які вказують необхідні стани OpenGL для надання геометрії. Мультипропуск реалізується шляхом додавання вузла в дерево малювання;
- граф сцена: забезпечена безпечна реалізація сценографу.

Дані можна передавати через компонент вузла, як концепцію. OpenSG використовує шаблон відвідувача для виконання операцій над вузлами. Для візуалізації сценографу спочатку викреслюються невидимі предмети. Після цього створюється спеціальний вид дерева (креслярське дерево) для сортування та подальшого надання.

OpenSceneGraph – це інтерфейс програмування 3D-програм із відкритим кодом, який використовується розробниками додатків у таких сферах, як візуальне моделювання, комп'ютерні ігри, віртуальна реальність, наукова візуалізація та моделювання. Інструментарій пишеться на стандартному C ++ за допомогою OpenGL [41]. Має наступні характеристики:

- підтримує платформи: Win32, Linux, Mac OS X;
- підтримує графічні API: OpenGL;
- підтримка шейдера: Асемблер, GLSL, Cg;
- обробка ефектів: Ефекти інтегровані в сценограф як спеціальні класи вузлів, де один ефект може працювати лише на одному вузлі;
- граф сцени: сценарій забезпечує багато різних класів вузлів. Класичний шаблон відвідувачів використовується для роботи над вузлами. Кожен клас вузлів має прямий доступ до базового API OpenGL.

2.6 Висновки до другого розділу

В даному розділ було розглянуто приклад обробки шейдерів, наведено приклад як граф сцени впорядковує елементи для візуалізації в ієрархічному порядку та як структури просторових даних використовуються для запобігання непотрібної роботи GPU.

Оскільки графічний рушій це складна і комплексна програма яка потребує правильного проектування було наведено шаблони проектування які дозволяють спростити масштабування програмного забезпечення за допомогою правильного проектування архітектури використовуючи шаблони об’єктно орієнтованого програмування.

Було наведено приклади графічних рушіїв з відкритим вихідним кодом які продемонстрували слабкі та сильні сторони свого застосування в різних сферах застосування.

3 АРХІТЕКТУРА ГРАФІЧНОГО РУШІЯ НА ОСНОВІ ОБ'ЄКТНОГО ПІДХОДУ

3.1 Рівні графічного рушія

Щоб згрупувати функціональність механізму візуалізації за тематичним та програмним рівнем, введено кілька модулів, організованих у вигляді шарів. Ці модулі називаються ядро та графіка. Модуль ядро працює на найнижчому рівні програмного забезпечення рушія і надає під модулі, такі як системні функції та загальні функції утиліти. Модуль графіка побудований поверх основного модуля і забезпечує інтерфейс до 2D та 3D графіки. Серед інших під модулів модуль Графіка містить інтерфейс візуалізації, незалежні від графічного інтерфейсу API, бібліотеки ефектів, графа сцени та імпортера декількох форматів файлів 3D – моделі. Ця реалізація будує міст між інтерфейсом візуалізації OpenGL або Direct3D. Повний перелік під модулів ядра та графіки з коротким описом у таблиці 3.1 та таблиці 3.2

Щоб мінімізувати залежності між під модулями та привести їх у порядок було введено правило яке забороняє включати кожен заголовок у кожен довільний під модуль. Існує визначення під назвою "Включити керівництво", яке дає інформацію про те, до якого під модулю дозволено включати, інший під модуль. На рисунку 3.1 показано графічне зображення з включенням, де під модулі рівня дозволяють включати під модулі того ж рівня або нижнього рівня.

Таблиця 3.1 – Підмодулі модуля Ядро

Назва під модуля	**Короткий опис**
Модуль платформи	Набір файлів заголовків, що визначають типи даних, залежні від платформи та ОС.
Модуль налагодження	Забезпечує функції налагодження, такі як ведення журналу, винятки, твердження та пошук виклику.
Модуль відбиття	Надає функції для запитів інтерфейсів, описів типів та перетворень, властивостей класів методами getter та setter. Цей модуль також забезпечує базовий інтерфейс, підрахунку посиланням.
Модуль вводу/виводу	Забезпечує класи для підтримки будь-якого типу введення та виводу програми, таких як файли, миша та клавіатура.
Модуль даних	Забезпечує керовану систему управління ресурсами.
Математичний модуль	Класи та структури для геометричних математичних операцій.
Модуль рушія	Забезпечує програмний каркас та реалізацію концепції актора.
Скриптовий модуль	Експортує всі властивості акторів та функції утиліти до мови сценарію, наприклад Lua.
Модуль малювання	Забезпечує класи для завантаження, обробки та запису зображень різних видів.
Модуль утиліт	Містить часто потрібні класи, такі як кеш, потік, події, критичні розділи та стрічкові утиліти.

Таблиця 3.2 – Підмодулі модуля Графіка

Назва під модуля	**Короткий опис**
Модуль змінних	Поняття абстрактного контейнера даних та його маніпулювання
Модуль візуалізації	Забезпечує графічний інтерфейс, незалежний від інтерфейсів API для загальних операцій візуалізації
Модуль графічного інтерфейсу користувача	Забезпечує на основі стилю 2D графічний інтерфейс користувальницького інтерфейсу, наприклад форми, кнопки та повзунки
Модуль ассетів	Надає імпортерів для файлів 3D-моделей
Модуль ефектів	Забезпечує рамки ефектів для об'єктів, що передаються
Модуль відтворення	Обробляє візуалізацію об'єктів, за допомогою інтерфейсу візуалізації
Модуль створення графіки	Структури даних для прискорення візуалізації шляхом відсікання
Модуль утиліт	Надає допоміжні класи, такі як кешування текстур

На рисунку 3.1 можна побачити графічне зображення модулів який демонструє модуль ядра та графіки та підмодулі які входять в них. Модулям дозволяється включати сабмодулі того ж рівня або рівня нижче.

Тепер оглянемо роботу найважливіших під модулів у модулі ядра в графічному рушії. Математична бібліотека містить усі класи математики, необхідні для програмного забезпечення, пов'язаного з графікою. Він містить усі визначення та операції, необхідні для векторної математики та інтерполяції.

Рисунок 3.1 – Графічне зображення модулів графічного рушія

Наступні типи операцій підтримуються з однією і подвійною точністю:

- операції над векторами в 2D просторі;
- операції над векторами в 3D просторі;
- операції над векторами в 4D просторі;
- операції над матрицями 4x4;
- операції з кфатерніонами;
- операції над лініями;
- операції над променями.

Математична бібліотека також може обробляти анімацію ключових кадрів усіх типів інтерполяції.

Підмодуль вводу / виводу надає класи для підтримки будь-якого типу введення та виводу програми. Він забезпечує кодек-менеджер для обробки рівномірних кодеків ресурсів. Кодек – алгоритм, інкапсульований у класі, який може читати та / або записувати дані в одному конкретному форматі. Прикладом кодека є XML-кодек, який здатний читати та записувати XML-файли. Він підтримує файлові потоки та потоки пам’яті, а також обробляє введення користувача від загальних пристроїв введення, таких як клавіатура та миша.

У підмодулі даних передбачена система управління ресурсами на основі блоку. Менеджер ресурсів дозволяє завантажувати будь-який ресурс з диска. Спочатку менеджер ресурсів намагається знайти імпортера для даного типу ресурсу, а потім передає роботу цьому імпортеру. Усі ресурси кешовані і не завантажуватимуться двічі.

Данні мають такі атрибути:

- унікальна назва;
- значення як рядок;
- карта атрибутів із рядом ключ та рядом значення.

Модуль даних сильно залежить від під модуля вводу – виводу для завантаження або запису даних на жорсткий диск.

Завданнями системи відображення є:

- дозволяти отримувати дескриптор типу для будь-якого об’єкта в рушії;
- зберігати ідентифікатор такий як рядок і надавати методи для створення іншого екземпляра цього типу;
- система відбиття зберігає посилання на батьківський тип;
- система відбиття також пропонує методи запиту на інтерфейси це дозволяє розробнику конвертувати ідентифікатор типу до класу типу і навпаки;
- задавати властивості типів, використовуючи макроси для визначення методів get- та set для змінних членів класу.

Ядром системи відображення є IType – інтерфейс, який забезпечує більшість функцій, перелічених вище. Використання макросів для оголошення визначених користувачем типів та властивостей нащадків IType – інтерфейсу створюються неявно. У лістингу 3.1 наведено приклад того, як використовувати макроси системи відображення для визначення типу.

Лістинг 3.1 – використання макросів відображення

```
// У файлі заголовка:
class Light : public Leaf
{
DECLARE_ACTOR( Light ) // Цей макрос визначає новий тип.
...
// У файлі реалізації:
// Реалізуйте цей тип для успадкування всіх властивості батьківського
типу
IMPLEMENT_OBJECT_PARENT_BEGIN(Light,Leaf) 'Leaf'.
PROPERTY_DECLARE("Intensity") // Визначте деякі властивості.
PROPERTY_DECLARE("Attenuation")
IMPLEMENT_OBJECT_PARENT_END(Light,Leaf)

// Визначення змінних членів та методів отримання для властивостей.
PROPERTIES_BEGIN_GET( Light )
// Прямий доступ до змінної var:
PROPERTIES_ADD_GET( "Intensity", mIntensity )
// Спершу викликаємо методу getter, а потім отримуємо значення з
елемента var:
PROPERTIES_ADD_GET_CALL( "Attenuation", mAttenuation,
RecalcAttenuation() )
PROPERTIES_END_GET()

// Визначення змінних членів та методів встановлення властивостей.
PROPERTIES_BEGIN_SET( Light )
// Прямий доступ до змінної var:
PROPERTIES_ADD_SET( "Intensity", mIntensity, Vec3f )
// Спочатку призначаємо значення для змінної, а потім викликаємо
метод встановлення:
PROPERTIES_ADD_SET_CALL( "Attenuation", mAttenuation, Vec3f,
UpdateProps() )
PROPERTIES_END_SET()
```

Під час виконання розробники можуть створювати об'єкти, передаючи ідентифікатор типу та конфігуруючи новий об'єкт, передаючи дерево даних на основі фрагмента.

3.2 Графічний конвеєр

У цьому розділі подано огляд того, як і де визначено графічні дані в графічному рушії. Також описано, як ці дані обробляються для створення зображення сцени на пристрої виводу.

Дані сцени в цьому контексті складаються з:

- мешів, що містять атрибути вершин і трикутників (які визначають форму тривимірного об’єкта), такі як положення векторів;
- текстурних карт;
- параметрів світла, наприклад напрямок, положення та інтенсивність;
- параметрів камери, наприклад положення, напрямок і поле зору;
- параметрів ефекту, наприклад коефіцієнти дифузного матеріалу, масштабних показників та шкали координат текстури.

Алгоритм, що виконується на даних сцени, включає:

1. Перетворення мешів на об'єкти відображення.
2. Призначення ефектів сцени на об’єкти відображення.
3. Знаходження видимих предмети по відношенню до предмета вибивання, таких як камера.
4. Візуалізацію розташованих об'єкти у правильному порядку.
5. Призначте значення світла та параметрів камери для відтворюваних об'єктів.
6. Надсилання графічних команд на графічний пристрій для відображення об'єктів.

Етапи роботи графічного конвеєра у графічному рушії представлені нижче. Ілюстрацію етапів конвеєра можна знайти на рисунку 3.2.

Граф сцени в графічному рушії, використовується для організації об’єктів сцени в ієрархічному порядку. Користувач може створювати різні види нодів, щоб упорядкувати полігони, освітлення та камеру сцени. Ефекти візуалізації не

застосовуються безпосередньо до мешів, але поширюються на сцену, створюючи екземпляри спеціальних класів вузлів. З кожним оновленням графу сцени його відтворювані об'єкти синхронізуються з базовим малюнком, як це представлено у наступному розділі. Об'єкти, що передаються в цьому контексті, складаються з об'єкта геометрії разом з ефектом, що визначає спосіб відображення цієї геометрії.

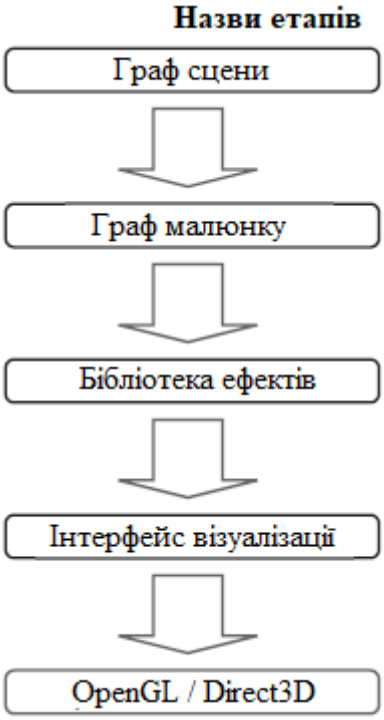


Рисунок 3.2 – Графічний конвеєр графічного рушія

Завдання графу малюнка полягає в тому, щоб забезпечити ефективний спосіб пошуку видимих предметів стосовно предметів, що відбивають, наприклад камера. Для кожного кадру візуалізації малюється запит на всі видимі об'єкти. Елемент управління передається до рамки ефектів, командуючи ефектом об'єкта для надання заданої геометрії.

Завданням бібліотека ефектів приймати дані візуалізації і оновлювати їх. Для візуалізації геометрії програми вершин та фрагментів під час проходу

активують дані геометрії які надсилаються на графічний пристрій. Усі команди, що надсилаються на графічний пристрій, виконуються через інтерфейс візуалізації.

Завдання інтерфейсу візуалізації полягає в перетворенні графічних команд, викликаних етапом вище, у графічні API команди, які можна надіслати на графічний пристрій.

3.3 Інтерфейс візуалізації

Основна мета інтерфейсу візуалізації - забезпечити простий у користуванні інтерфейс для кодування 2D та 3D графіки. Навіть із більш досконалими методами візуалізації, такими як шейдери високого рівня, розробник не повинен перейматися конкретними деталями графічного API. Тому цей інтерфейс абстрагує функції графічного пристрою незалежним від графічного інтерфейсу API.

Сам інтерфейс візуалізації забезпечує класи інтерфейсів для всіх загальних операцій візуалізації, таких як створення геометрій та текстурування, встановлення станів візуалізації та створення шейдерів. Ці інтерфейси потім реалізуються драйвером візуалізації, який можна завантажувати під час виконання. Ця концепція робить модуль Graphics повністю незалежним від базового обладнання та будь-якого графічного API (наприклад, OpenGL та Direct3D). Основним класом C ++ є інтерфейс пристрою IDevice, який є вхідною точкою до функцій візуалізації графічного рушія. Цей інтерфейс також є заводським класом важливих об’єктів візуалізації, таких як текстури та геометрія.

Геометрія в графічного рушія складається зі списку змінних, де кожна змінна відображається на вертексний атрибут (наприклад, положення, нормалі чи колір). Геометрія також може містити змінну буфера індексу в змінних атрибутів. Підтримується лише один буфер індексів на геометрію, оскільки в даний час графічні пристрої не підтримують більше буферів індексів. Геометрія також

повинна містити змінні із зазначенням примітивних типів та кількості елементів, що використовуються кожним примітивним типом. На рисунку 3.3 показаний перелік змінних, що складають геометрію. Даний графічний рушій може виконувати обробку декількох примітивних типів на кожну геометрію. У додатку Л наведено лістинг інтерфейсу візуалізації.

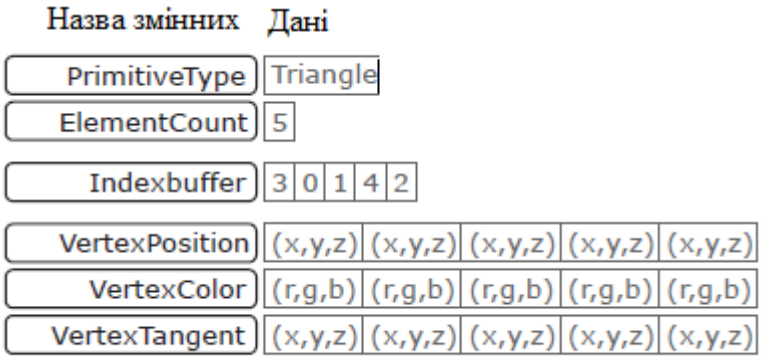


Рисунок 3.3 – Змінні, що формують геометрію

3.4 Бібліотека ефектів

Використовуючи бібліотеку ефектів, користувач системи може використовувати попередньо визначені невеликі основні ефекти (наприклад, розсіяне освітлення, тіньове відображення та відображення текстур), щоб легко створювати більші та складні ефекти візуалізації. Для цього користувачеві не потрібно мати знання про базовий шейдерний вихідний код, багатопрохідність або візуалізації. Ці питання автоматично обробляються бібліотекою ефектів.

Кожен базовий ефект відтворення складається з однієї або декількох технік, що реалізують бажаний ефект. Одну техніку вибирають під час виконання залежно від наявного обладнання та рівня деталізації. Така техніка візуалізації складається з однієї або декількох частин техніки, які являють собою найменший блок завдання для досягнення ефекту візуалізації. Техніки мають необхідні вхідні дані та

результати, які вони виробляють. Відповідна техніка може виконувати свій код на різних ресурсах, таких як CPU, блок обробки вершин GPU або блок обробки GPU фрагментів.

Кожен ефект структурується таким чином: Ефект (IEffect) складається з однієї або декількох технік (ITechnique). Кожна окрема техніка повністю реалізує бажаний ефект для себе особливим чином, однак, лише один із них вибирається під час виконання для використання для візуалізації. Це рішення базується на тому, яка техніка підтримується на поточному графічному обладнанні. Крім того, для кожної техніки може бути визначений діапазон рівня деталізації (LOD), який потім використовується для вибору належної техніки для поточної відстані між камерою та об’єктом.

Оскільки методи можуть бути складними алгоритмами та конструкціями, вони розбиваються на одну або кілька частин (ITechniquePart). Кожна частина техніки обробляє невеликий набір завдань для досягнення бажаного ефекту. Прикладами таких завдань є:

- обчислення об'єму тіні для заданої геометрії та світла;
- перетворення напрямку камери в дотичний простір вершини;
- обчислення зміщення координати текстури для рельєфного чи паралаксального відображення.

Деякі з цих завдань вирішуються за допомогою центрального процесора, а інші реалізуються за допомогою програми вершин або фрагментів, що працює на GPU. Оскільки необхідно обробляти вищезазначені завдання в межах одного дзвінка виклику, технічні частини, які потребують GPU, повинні бути з'єднані разом, щоб розширити лише одну програму вершин та фрагментів. Такі частини є похідними від класу C ++ PipelinePartTechniquePart. Назва випливає з того, що кожна така техніка частково реалізує частину з графічного конвеєра. Ідея бібліотеки ефектів полягає в тому, що такі деталі графічного конвеєра повинні бути

комбіновані автоматично. Користувач вибирає, які ефекти використовувати, а бібліотека ефектів перетворює вміст PipelinePartTechniqueParts в програму вершин та фрагментів, яку можна передати.

Частини техніки мають входи, які змінюють поведінку деталей і можуть мати результати, обчислені частинами. Відповідні інтерфейси C ++ – ц е IInput та IOutput. Введення можна класифікувати за їх варіацією: вони можуть бути на основі вершин (атрибутами вершин), тобто вони можуть змінюватися в залежності від вершини, або вони можуть бути рівними для всієї виведеної геометрії. Вихідні дані завжди розраховуються на основі вершин.

Важливою особливістю бібліотеки ефектів є те, що вони можуть змінювати об'єм вилучених об'єктів, які надаються з цим ефектом. Це потрібно, якщо об’єкт A не знаходиться у площині перегляду сценічної камери, але видно завдяки ефекту від іншого об'єкта B, який змушує об'єкт A з'являтися на остаточному зображенні. Прикладом такого ефекту є сцена, що містить дзеркало, як показано на рисунку 3.4. У таких умовах об'єкти поза оглядовим фруструмом, але відображення у дзеркало і все ще потрібно намалювати.

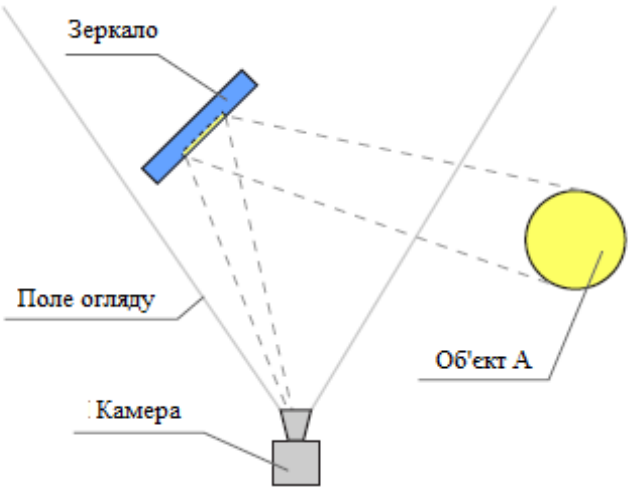


Рисунок 3.4 – Приклад використання техніки відсікання

Тепер поговоримо про класи які приймають участь у процесі візуалізації зображення найважливішим інтерфейсом у цьому контексті є IPass, який є інтерфейсом для всіх проходів візуалізації та реалізований наступними класами:

- DefaultPass: Цей пропуск відображається до фреймбуфера і не має особливої поведінки;
- AdditionalPass: Цей клас пропускання також реалізує інтерфейс ITechniquePart і тому може використовуватися як частина техніки. Зазвичай у цього екземпляра є власний об'єкт візуалізації;
- PipelinePass: Внутрішньо цей пропуск має власне управління візуалізацією – як це має пропуск ефекту CgFX. Таким чином, вони не встановлюються безпосередньо користувачем, а надаються CgFXEffects. Цей клас також походить від AdditionalPass;
- PostProcessingPass: Цей пропуск вимагає заповненого буфера кадрів та глибинного буфера як вхідного сигналу та відображається до буфера кадрів або окремого об'єкта візуалізації. Цей клас також походить від AdditionalPass.

Ще один важливий клас – стек об'єктів відображення. Об'єкти відображення можна помістити в стек і активувати цей клас. Коли з'являються об'єкти візуалізації з верхньої частини стека, основні візуалізатори знову активуються. Цей клас (RenderTargetStack) потрібен, коли мова йде про багатопроменеву візуалізацію з різними об'єктами візуалізації для кожного пропуск.

У наступному абзаці подано повне покрокове введення внутрішньо виконаних кроків для отримання остаточного зображення у буфері кадрів із списку вибраних користувачем ефектів.

1. Залежно від поточного графічного обладнання та поточного рівня деталізації об'єкта вибирається перелік методик реалізації бажаних ефектів.
2. Перелік деталей техніки вилучається зі списку техніки.

3. Цей список розділений на дві групи: AdditionalPasses і PipelinePartTechniqueParts.

4. Список PipelinePartTechniqueParts передається PartConnector, який будує групи технічних частин, де кожна група буде з'єднана разом в одну вершинну або фрагментну програму. Якщо користувач бажає розмістити деталі техніки в різні групи, він повинен призначити різні групові індекси.

5. Кожна група частин техніки присвоюється екземпляру AdditionalPasses та екземпляру DefaultPass. Присвоюючи деталі техніки для передачі візуалізації, пропуск приймає всі входи, виходи та залежність стану деталей техніки.

На цьому етапі сформовано список пропусків, що передаються, і очікують виконання за допомогою виклику методу Render ().

Порядок виконання пропусків має вирішальне значення.

1. При виклику Render () – методу IPass, призначені технічні частини перетворюються на вершинні – та фрагментні програму.

2. Якщо будь – які атрибути вершин геометрій, переданих методу Render (), відсутні (але їх вимагає програма вершини), вони генеруються автоматично, якщо це можливо.

3. Якщо у пропуску призначений власний об'єкт візуалізації, його переміщують у RenderTargetStack.

4. Відображення стану змінних, що передаються методу Render (), виконуються. Це означає, що їх інкапсульована візуалізація застосовується до графічного пристрою.

5. Значення змінних, переданих методу Render (), копіюються на входи проходу.

6. Усі вхідні передачі присвоюються створеній програмі вершин та фрагментів.

7. Геометрія передана методу Render (), і відображається.

8. Призначені стани відображення деактивовані.

9. Якщо пропуску призначено спеціальні об'єкти відображення, вони видаляються з RenderTargetStack.

10. StateChanges виконується.

Бібліотека ефектів все ще підтримує візуалізацію за допомогою фіксованого функціонального конвеєра графічного пристрою. Це досягається присвоєнням PipelinePartTechniqueParts для реалізації IPass. Отже, жодна вершинна або фрагментна програма не буде генерована, але інші кроки залишаються такими ж. Діаграма UML класів ефектів міститься в додатку Е.

Якщо об’єкти знаходяться далеко від камери, не потрібно відображати їх з усіма дрібницями, оскільки ці деталі взагалі неможливо побачити. Спростити їх зовнішній вигляд можна, наприклад, досягти за рахунок зменшення граней геометрії. Іншим варіантом досягнення кращих показників є спрощення алгоритму, що використовується для візуалізації геометрії.

Бібліотека ефектів графічного рушія дозволяє вказати мінімальну та максимальну відстань для техніки, що реалізує ефект. Здійснюючи поточну відстань між положенням камери та світовим положенням наданого об'єкта, бібліотека ефектів автоматично вибирає техніку. На прикладі, представленому на рисунку 3.5, показаний чайник з ефектом, що складається з трьох застосованих методик, зліва направо: рельєфне картування, паралаксне відображення та звичайне відображення. Техніка, що використовується для найближчого візуалізації, реалізує рельєфне відображення. Наступні два рівні деталізації реалізують відображення паралакса та нормальне відображення.

Рисунок 3.5 – Приклад показує ефект LOD у використанні

Якщо на даній відстані доступно більше однієї техніки, алгоритм вибирає техніку, яка також буде використовуватися для подальшого надання. Для такої бажаної поведінки алгоритм спостерігає за варіантами відстані, щоб мати змогу робити припущення щодо того, яку техніку вибиратимуть у майбутньому. Це досягається, якщо припустити, що зміна відстані залишається постійною, а потім мінімізуючи перемикачі техніки відтворення при ходьбі по осі відстані. У додатку Б можна побачити UML діаграму бібліотеки ефектів.

3.5 Граф малюнку

Граф малюнку у графічному рушії це структура даних, яка використовується для прискорення візуалізації. Як введення, граф малюнку потребує повного перегляду даних про всі об'єкти, які потенційно можуть бути відображені, а як вихідний графік структура даних надає перелік об'єктів, які видно відносно списку об'єктів вилучення, і тому надсилаються на графічний пристрій для візуалізації. Завданням графу малюнку є знаходження набору видимих об’єктів.

Найпростішим підходом до вирішення завдання графу малюнку, було б просто перерахувати кожен окремий об'єкт вхідного списку і протестувати його. Незважаючи на те, що цей алгоритм є математично правильним, його не

рекомендується до використання, оскільки тестування кожного об'єкта може призвести до ще нижчої продуктивності, ніж без використання графу сцени. Це в тому випадку, якщо надання об’єкта відбувається швидше, ніж тестування його на всіх об'єктах вилучення.

Як випливає з назви граф малюнку, загальні рішення цієї проблеми використовують графік як внутрішню структуру даних. Можливими підходами є використання kd-дерева, octree або подібних структур даних.

Графік малюнку у графічному рішуї зберігається абстрактно за допомогою визначеного інтерфейсу. У додатку М можна побачити реалізацію графу малюнку.

3.6 Граф сцени

Граф сцени – це загальна структура даних, яку зазвичай використовують векторні програми редагування графіки та сучасні комп’ютерні ігри, яка упорядковує логічне та часто просторове зображення графічної сцени.

Завдання графу сцени в графічному рушії:

- забезпечте структурований вигляд усіх даних сцени;
- дозволити користувачеві графа сцени отримати доступ до даних сцени (включаючи меші, світло, ефекти, текстури та подібні елементи);
- синхронізувати інформацію про зображувані об'єкти;
- зберігати інформацію про ієрархічну трансформацію об'єктів сцени;
- дозволяти користувачеві групувати логічно пов'язані об'єкти разом;
- створювати представлення на пам'ять загальних форматів файлів графу сцени.

До завдань графу сцени не відноситься:

- надсилати графічні команди графічному пристрою;

- викликати будь – які графічних функцій API, такі як OpenGL та Direct3D.

Отже, головне завдання графу сцени в графічному рушії – організувати об’єкти структурованим, ієрархічним способом. Алгоритми, які виконуються графом сцени, безпосередньо не відображають будь – які об'єкти. Причиною такого підходу є повне розчленування даних та операцій, що виконуються на них. Поділ на ці дві частини проілюстровано на рисунку 3.7. Таким чином полегшується заміна існуючих алгоритмів. Навіть введення нових операцій не потребує зміни будь – якого коду в класах, які надають лише дані для алгоритмів. Це призводить до чистої внутрішньої поведінки та добре структурованого API графу сцени.

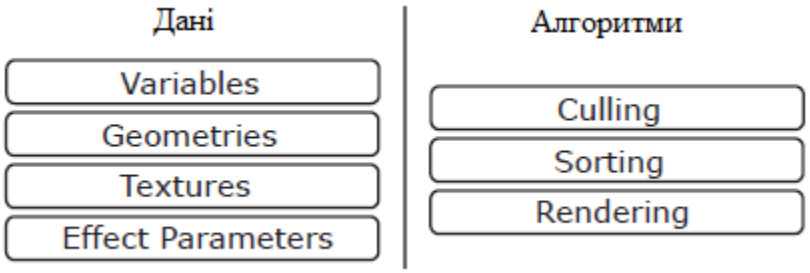


Рисунок 3.7 – Повне роз'єднання даних та алгоритмів у графічному рушії

В графі сцени оновлення даних відбувається наступним чином:

1. Змінилися дані вузла в сценограмі.
2. Вузол посилає подію в глобальний контекст сценограму, що містить ідентифікатор події та сам вузол як об'єкт відправника. Ідентифікатор події вказує, який тип даних було змінено.
3. На початку кадру контекст глобального сценограму обробляє всі підняті події.
4. Для кожної події складається список завдань, які необхідно виконати в базі даних сцени. Генеровані завдання збираються в список.

5. У списку всіх завдань шукаються дублюючі завдання та сортується за заданим замовленням. Наприклад матриці перетворень світових вузлів потребують оновлення, перш ніж їх обмежуючі обсяги у світовому просторі можуть бути перераховані.

6. Нарешті, виконується перелік завдань.

Базовий клас для всіх об’єктів графа сцени це класс «Node». Основною властивістю Node класу є те що клас «Node» також реалізує інтерфейс IPositioned, який дозволяє користувачеві запитати про позицію «Node» об’єкта у світовому просторі.

Group клас реалізує методи для організації двох або більше вузлів разом, як нащадків цього вузла.

RootGroup клас не може мати предків, а лише нащадків. Зазначене застосування цього класу полягає в тому, що він використовується як кореневий вузол сценографа.

Separator клас не дозволяє транспортувати структурні дані, такі як змінні, та надавати ефекти за межами цієї групи.

TransformGroup клас додатково зберігає інформацію про перетворення, яка застосовується до всіх нащадків цієї групи.

Switch клас дозволяє динамічно включати та виключати своїх предків із сцени. Інтерфейс дозволяє вибирати всіх нащадків на основі масок. При виборі на основі маски використовується булевий прапор для кожного нащадка, який включає або виключає цей вузол.

Протилежним до класу Node є клас Leaf, який не має дочірніх вузлів і тому не може бути батьківським жодному інших вузлів класу Node.

Camera даний клас представляє камеру за її положенням, напрямком, вектором вгору, полем зору, біля площини та далеко від площини. Він також реалізує інтерфейс IVariableEmitter використовується для збереження таких даних

як: глобальні змінні, дані про об’єкти які будуть візуалізовані та дані про об’єкт коли він вже візуалізований.

Effect даний клас є лише обгорткою для бібліотеки ефектів. Завдання цього класу є візуалізовувати ефекти, що візуалізовані на сцені.

FileTexture використовується для накладання текстур на об’єкти, потім текстура перетворюється в змінну, яка буде використана в графі сцени.

Light цей клас реалізує змінні, що містить інтенсивність світла, та інші змінні, що містить коефіцієнти ослаблення. Це також базовий клас всіх інших класів освітлення.

DirectionalLight клас є похідним від Light – класу вище та додатково реалізує змінні, що містить напрямок освітлення.

PointLight клас клас є похідним від Light – класу вище та додатково реалізує змінні, що містить положення освітлення.

SpotLight клас є похідним від Light – класу вище та додатково реалізує змінні, що містять такі дані: положення та напрямок точкового освітлення, кут світлового конуса, матрицю огляду та матрицю проекції точкового світла.

Mesh даний клас містить один або кілька об'єктів IGeometry. Він реалізує змінні, що містять текстури, стадії візуалізації та дані геометрії. Цей клас використовується для побудови зображуваних об'єктів з їхньою геометрією.

DefaultVariableEmitter клас використовує змінні зображення графу сцени. Змінна може бути з базових типів float, double, bool та integer та будь – якого з них.

RenderStateEmitter реалізує змінні візуалізації, які можна використовувати для візуалізації ефектів.

Global клас, який поєднує все: має посилання на кореневий вузол графу сцени та посилання на граф малюнку. Він обробляє події, що надсилаються класами сцени, і містить метод, який програма повинна викликати кожен раз для

оновлення кадру графу сцени. Оскільки цей клас повинен бути встановлений лише один раз у програмі, він реалізовується відповідно до однотонної схеми дизайну.

TransformStack клас реалізовує стек з матрицями як елементами стеку. Клас може виконувати комбіноване перетворення всіх елементів на стеку.

Database клас тримає в собі копії всіх змінних які використовуються. В додатку К зображено UML діаграму класів графу сцени.

3.7 Застосування шаблонів проектування

У цьому розділі дано опис того як шаблони проектування були використані в проектування архітектури графічного рушія на основі об'єктного підходу.

- шаблон абстрактна фабрика: використовується для створення змінних;
- шаблон прототип: клонування змінних;
- шаблон одинка: застосовується до всіх глобальних класів, дозволений лише один екземпляр;
- шаблон адаптера: загортання різних реалізацій OpenGL одного і того ж функціоналу за допомогою одного загального інтерфейсу C ++ для кожної методики OpenGL;
- шаблон міст: Інтерфейс візуалізації, незалежний від графічного API. У цьому контексті інтерфейс візуалізації є реалізатором, драйвери візуалізації OpenGL та Direct3D – це конкретні реалізатори, IPass – інтерфейсу;
- шаблон проксі: використовується для підключення сторонніх бібліотек;
- шаблон спостерігач: використовується з генераторами завдань графу сцени, генератори завдань реєструються в контексті об'єкта сценографу, який чекає, коли подія буде звільнена. Потім контекстний об'єкт запитує у зареєстрованих генераторів завдань, чи можуть вони обробляти запущену подію;

- шаблон шаблонного методу: використовується з великою кількістю базових класів. Базові класи реалізують інтерфейси, а потім отримують конкретні класи. Прикладами базових класів в графічному рушії є BaseEffect, BaseTechnique, BaseTechniquePart і BasePass;
- шаблон відвідувача: шаблон відвідувача використовується для оновлення операцій на сцені.

Проектуючи архітектуру програмного забезпечення за допомогою об’єктно – орієнтованої парадигми ефективність архітектури залежить від правильно вибраних шаблонів проектування.

3.8 Реалізація алгоритму «Ray casting»

В даному розділі продемонстровано 2D та 3D реалізацію алгоритму «Ray Casting» за допомогою графічного рушія, але спочатку поговоримо про сам алгоритм.

Ray casting це алгоритм, який відображає 3D – світ на основі 2D – карти. Ray casting, як правило це кадри з низькою роздільною здатністю, це зроблено для того щоб запобігти відставанню картинки так як цей алгоритм в основому використовується на машинах з слабкою обчислювальною потужністю.

Ray casting не слід путати з ray tracing, який надає променям більшу фізичну точність, забезпечуючи відбиття та заломлення світлових променів, і відстежує промені у двох вимірах, а не в одному вимірі, як ray casting.

Raycasting працює, викидаючи "промені", щоб виміряти відстань до найближчої стіни, звідси і термін "raycaster". Програма висилає промені, починаючи від початку камери, рухаючись вперед, поки камера не вдариться об стіну, і в цей момент камера проходить відстань, і намалює стовпчик на основі відстані. Чим ближче стіна, тим більша колона. Промені надсилаються в різні боки,

при цьому кут надсилання визначає, де колона буде намальована. Промінь, надісланий праворуч від точки зору гравця, намалює стовпчик у правій частині екрана, а промінь, надісланий ліворуч, намалює колонку зліва. Після відправлення всіх променів буде видно повну картину.

Є два типи алгоритму ray casting, які можна запрограмувати: метод на основі спрайту та метод на основі списку. У кожного є свої переваги та недоліки таблиця 3.3 [27].

В даній роботі реалізовано алгоритм на основі списку, у лістингу 3.2 наведено мапу сцени яка буде візуалізована.

Таблиця 3.3 – Методи алгоритму «Ray casting»

	На основі спрайту	**На основі списку**
Рівень складності	Легко	Складно
Частота кадрів	Низька	Висока
Знання необхідні	Базові знання програмування	Базові знання програмування, тригонометрії, масивів
Як зберігається карта	Карта зберігається у вигляді спрайтів, окрема сцена, окремі спрайти.	Як масив (сітка чисел), кожна сцена - це окремий масив.
К-сть спрайтів яка потрібна	Один або більше, залежить від сцени	Хоча б один
Переваги	Можна використовувати для виготовлення сцен із вигнутими стінами	Легко генерувати випадкові сцени

Лістинг 3.2 – Характеристики мапи яка буде візуалізована.

```
// Карта створена з # блоків, # = стіни, . = вільний простір.
wstring map;
map += L"#########.......";
map += L"#...............";
map += L"#.......########";
map += L"#..............#";
map += L"#......##......#";
map += L"#......##......#";
map += L"#..............#";
map += L"###............#";
map += L"##.............#";
map += L"#......####..###";
map += L"#......#.......#";
map += L"#......#.......#";
map += L"#..............#";
map += L"#......#########";
map += L"#..............#";
map += L"################";
```

На рисунку 3.8 можна побачити як карта та позиція камери відображаються у вікні візуалізації.

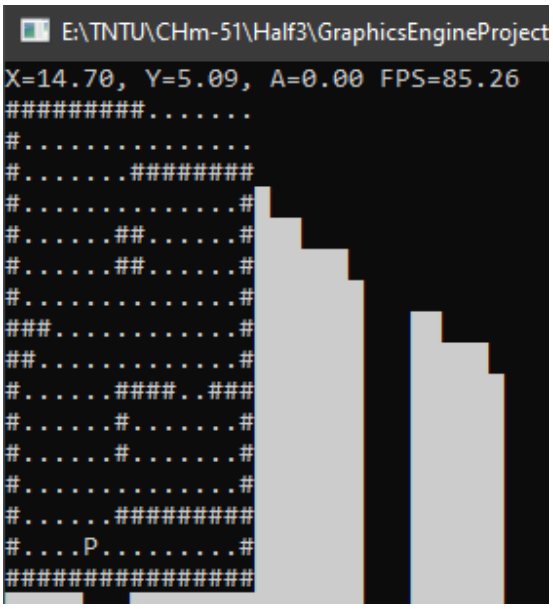


Рисунок 3.8 – Візуалізація мапи

На рисунку 3.9 зображено як відображаються об’єкти які візуалізовано на відстані.

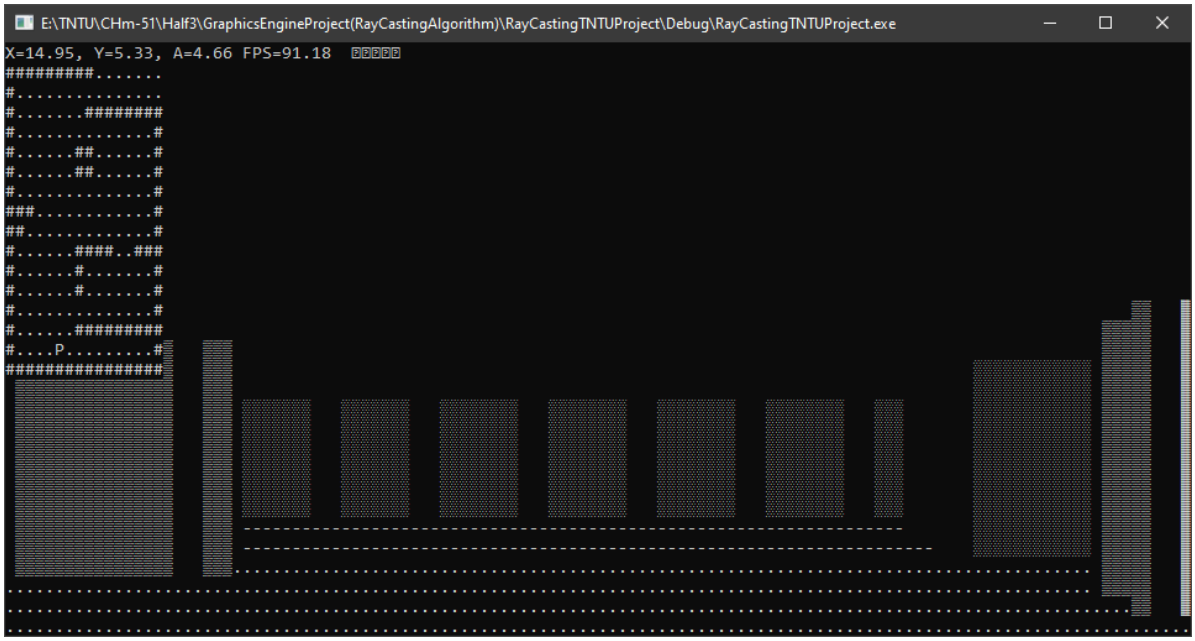


Рисунок 3.9 – Візуалізація об’єктів на відстані

На рисунку 3.10 можна побачити як буде візуалізовуватися об’єкт праворуч який знаходиться на ближній дистанції до камери та як візуалізовується об’єкт що знаходиться у кінці мапи.

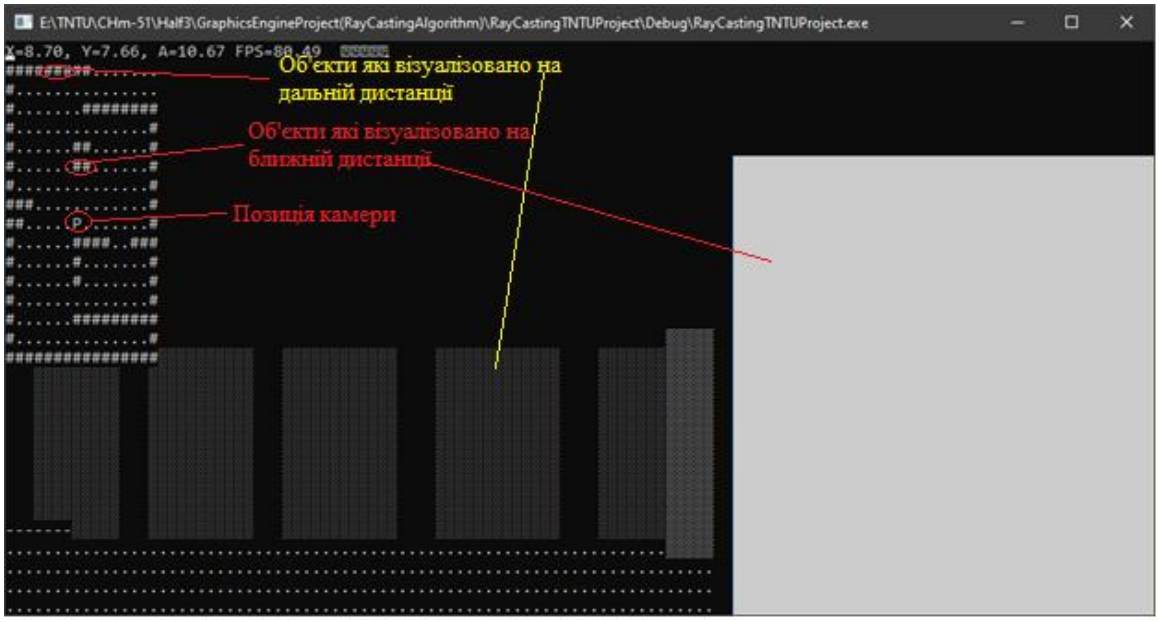


Рисунок 3.10 – Візуалізація об’єктів

На рисунку 3.11 показано як алгоритм візуалізація об’єктів на ближній, середній на дальній дистанцій відповідно до напрямку погляду камери. У додатку И можна наведено повний код програми.

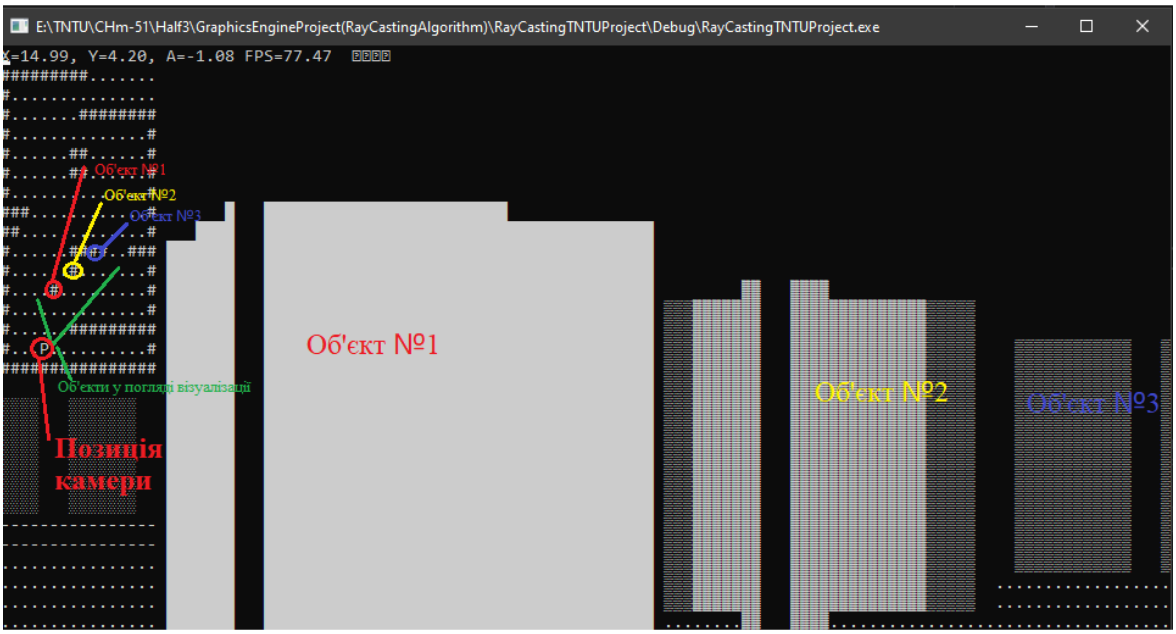


Рисунок 3.11 – Візуалізація об’єктів на різній дистанції

На рисунку 3.12 та 3.13 приведено 2D візуалізація алгоритму Ray Casting. Промені кидаються у всі сторони і відбиваються від об’єктві на їхньому шляху, за рахунок чого відбувається прорахунок відстані до об’єкта та його рівень візуалазації, на даному рисунку промені кидаються у всі сторони так як в ньому продемонсторано погляд на об’єкти під кутом 365 градусів, кожен об’єкт візуалізовується відповідно до відстані. Повний код програми наведено у додатку Ж.

Кожен з алгоритмів має свої переваги та недоліки відповідно до завдання яке ставиться перед ним. В даному підрозділі відображено швидкодію алгоритмів візуалізації та їх переваги.

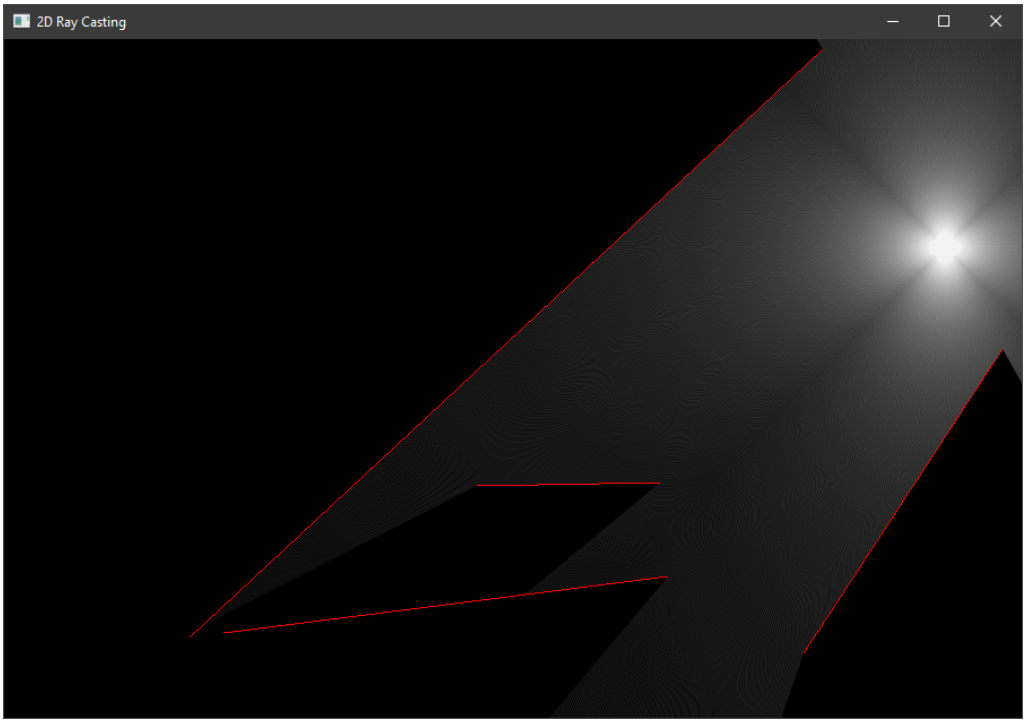


Рисунок 3.12 – 2D візуалізація алгоритму «Ray Casting»

Алгоритм «Scanline»

Переваги:

- користується когерентністю, що призводить до збільшення швидкості;
- візуалізовує лише видимі пікселі.

Недоліки:

- складний до реалізації;
- потребує отримання всіх полігонів перед візуалізацією.

Алгоритм «Ray casting»

Переваги:

- легка реалізація;
- не потребує великої обчислювальної потужності.

Недоліки:

- невелика роздільна здатність картинки;
- повільний порівняно з іншими методами;
- багато артефактів у складних ситуаціях.

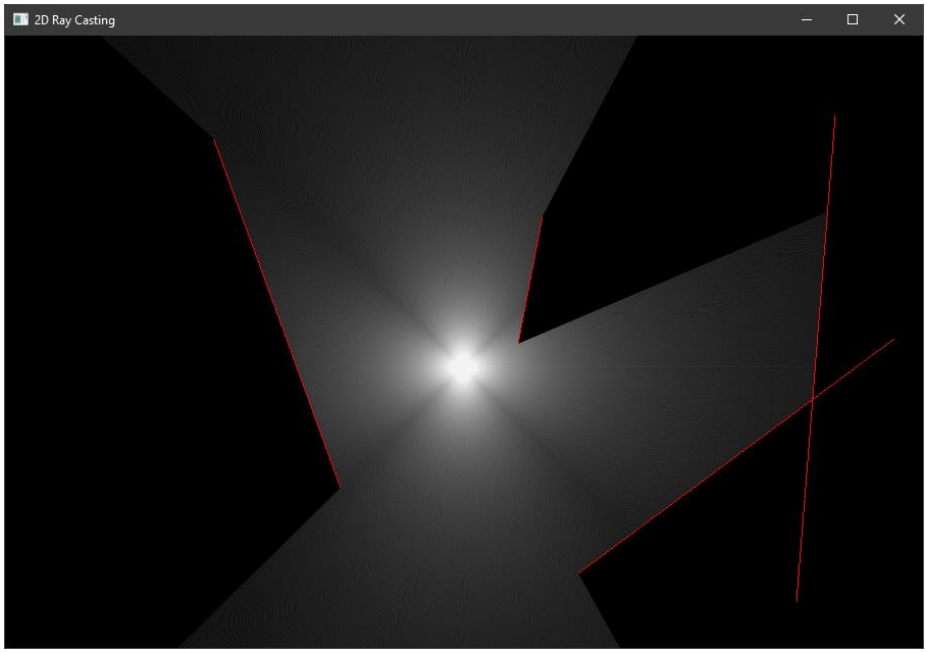


Рисунок 3.13 – 2D візуалізація алгоритму «Ray Casting»

Алгоритм «Ray tracing»

Переваги:

- реалістичне відображення відбиття, заломлення, тіней;
- легко досягається згладжування ефектів.

Недоліки:

- потрібна велика обчислювальна потужність;
- важка реалізація;
- передові освітлювальні ефекти віжко відстежувати.

На рисунку 3.14 зображено графік який відображає швидкість візуалізації 60 кадрів кожним із алгоритмів.

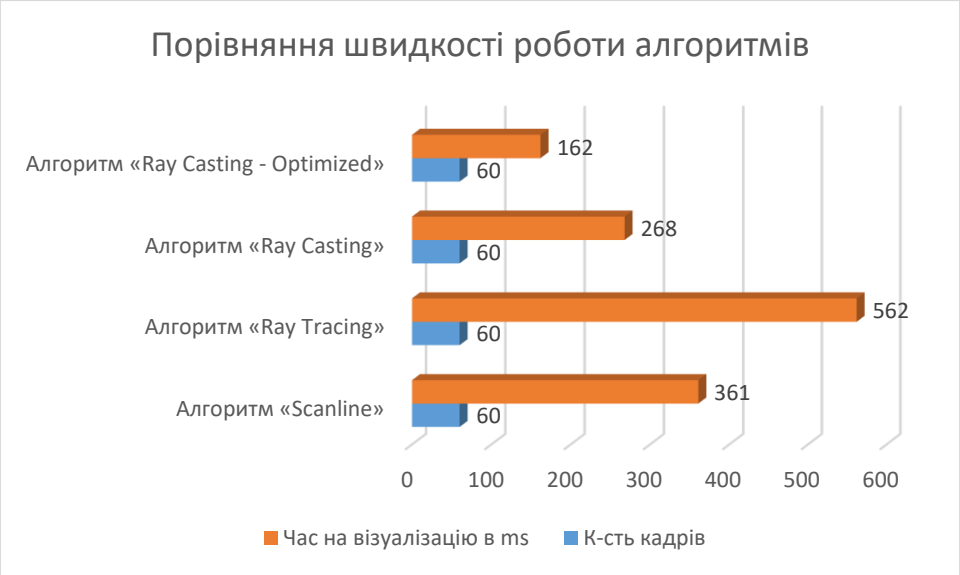


Рисунок 3.14 – Порівняння швидкодії алгоритмів.

З графіка видно, що для візуалізації 60 кадрів алгоритму «Ray Casting» потрібно 268 мілісекунд, алгоритму «Ray Tracing» потрібно 562 мілісекунд, алгоритму «Scanline» потрібно 361 мілісекунди, алгоритму «Ray Casting – Oprimized» потрібно 162 мілісекунд. Найкращу якість візуалізації дає алгоритм

«Ray Tracing», але за рахунок складність реалізації та потреб до високої обчислювальної потужності, його потрібно реалізовувати відповідно до завдань які ставляться перед розробником програмного забезпечення. У додатку Н наведено виміряні дані за допомогою програми «Intel Graphics Monitor».

3.9 Висновки до третього розділу

У даному розділі було розглянуто реалізацію архітектури елементів графічного конвеєра у графічному рушію, продемонстровано застосовування шаблонів об’єктно орієнтованого програмування у розробці архітектури графічного рушія, також наведено 3D та 2D реалізацію алгоритму «Ray Casting».

Даний алгоритм має не важку реалізацію та не потребує великої обчислювальної потужності, його використання є доцільним для демонстрації роботи графічного рушія.

4 СПЕЦІАЛЬНИЙ РОЗДІЛ

Візуалізація або синтез зображень – це автоматичний процес генерації фотореалістичного або нефотореалістичного зображення з 2D або 3D моделі за допомогою комп'ютерних програм. Також результати показу такої моделі можна назвати візуалізацією. Файл сцени містить об'єкти в строго визначеній мові або структурі даних, вона містить інформацію про геометрію, точку зору, текстуру, освітлення та затінення як опис віртуальної сцени. Дані, що містяться у файлі сцени, потім передаються програмі візуалізації для оброблення та виводяться в цифрове зображення або растровий графічний файл зображення.

Хоча технічні деталі методів візуалізації відрізняються, загальні завдання для подолання створення 2D-зображення із 3D-зображення, що зберігається у файлі сцени, окреслені як графічний конвеєр уздовж пристрою візуалізації, такого як GPU. Графічний процесор – це цільовий пристрій, здатний допомогти CPU у виконанні складних обчислень візуалізації. Якщо сцена повинна виглядати відносно реалістично та передбачувано при віртуальному освітленні, програмне забезпечення для візуалізації повинно вирішити рівняння візуалізації. Рівняння візуалізації не враховує всіх явищ освітлення, але є загальною моделлю освітлення для комп'ютерних зображень. "Візуалізація" також використовується для опису процесу обчислення ефектів у програмі редагування відео для отримання остаточного результату [15].

Візуалізація – одна з головних підтем 3D-комп’ютерної графіки, і на практиці завжди пов'язана з іншими. У графічному конвеєрі це останній головний крок, який надає остаточний вигляд моделям та анімації.

Візуалізація використовує в архітектурі, відеоіграх, тренажерах, візуальних ефектах для кіно чи телевізійних програм та візуалізації дизайну, кожен з яких використовує різний баланс функцій та прийомів. В якості продукту доступна

широка кількість рендерів. Деякі інтегруються у більші пакети моделювання та анімації, деякі – автономні, деякі – безкоштовні проекти з відкритим кодом. Зсередини рендерінг – це ретельно розроблена програма, заснована на вибірковій суміші дисциплін, пов'язаних із: легкою фізикою, зоровим сприйняттям, математикою та розробкою програмного забезпечення.

У випадку 3D – графіки візуалізація може здійснюватися повільно, як при попередньому візуалізації, так і в режимі реального часу. Попереднє візуалізація – це обчислювально інтенсивний процес, який зазвичай використовується для створення фільмів, тоді як візуалізація в режимі реального часу часто проводиться для 3D-відеоігор, які покладаються на використання графічних карт з апаратними прискорювачами 3D.

Програмне забезпечення для візуалізації може використовувати декілька різних технік для отримання остаточного зображення [13].

Рисунок 4.1 – Остаточне візуалізоване зображення

Далі ми розглянемо основні методи 3D – візуалізації.

4.1 Scanline Rendering

Scanline rendering – це алгоритм визначення видимої поверхні в 3D – комп'ютерній графіці, який працює на основі рядків за рядком, а не на багатокутнику за полігоном або пікселем за пікселем. Усі багатокутники, які потрібно винести, спочатку сортуються за верхньою координатою Y, на якій вони спочатку з'являються, потім кожен рядок або лінія сканування зображення обчислюється за допомогою перетину сканування з полігонами на передній частині відсортованого списку, при цьому відсортований список оновлюється, щоб відкинути вже не видимі багатокутники, оскільки активна лінія сканування просувається вниз по малюнку [19].

Основна перевага цього методу полягає в тому, що сортування вершин вздовж нормальної площини сканування зменшує кількість порівнянь між ребрами. Ще одна перевага полягає в тому, що не потрібно переводити координати всіх вершин з основної пам'яті в робочу пам'ять – лише вершини, що визначають ребра, які перетинають поточну лінію сканування, повинні бути в активній пам'яті, і кожна вершина читається лише один раз. Основна пам'ять часто дуже повільна в порівнянні з зв'язком між центральним процесорним блоком і кеш-пам'яттю, і, таким чином, уникнення повторного доступу до вершин в основній пам'яті може забезпечити значне прискорення.

Цей алгоритм може бути легко інтегрований з багатьма іншими графічними методами.

4.2 Ray Casting Rendering

Ray casting – це використання тестів перетину променів на поверхню для вирішення найрізноманітніших завдань у 3D – комп'ютерній графіці та обчислювальній геометрії. Термін вперше був використаний у комп'ютерній

графіці в статті 1982 року Скоттом Ротом для опису способу надання конструктивних суцільних геометричних моделей.

Ray casting може стосуватися різноманітних проблем і прийомів:

- загальна проблема визначення першого об'єкта, що перетинається променем, як при виявленні зіткнень.
- методика прихованого видалення поверхні, заснована на знаходженні першого перетину променя, відкинутого від очей, через кожен піксель зображення.
- алгоритм відтворення нерекурсивного трасування променів, який відкидає лише первинні промені.
- об'ємне випромінювання променя, метод прямого об єму, в якому промінь "проштовхується" через об'єкт, а 3D-скалярне поле, що цікавить, відбирається вздовж променя всередині об'єкта.

Хоча "Ray casting" та "Ray tracing" часто використовувались взаємозамінно в ранній літературі 3D-комп'ютерної графіки, більш недавнє використання намагається розрізнити ці два, хоча і не дуже успішно. Основна відмінність полягає в тому, що термін «Ray casting» ніколи не застосовується, коли промені простежуються рекурсивно [21].

Ray casting – най основніший з багатьох алгоритмів візуалізації комп'ютерної графіки, які використовують геометричний алгоритм трасування променів. Алгоритми візуалізації на основі трасування функціонують у порядку зображення для відображення тривимірних сцен на двовимірних зображеннях. Геометричні промені простежуються від очей спостерігача для зразка світла (сяйва), що рухається до спостерігача з напрямку променя. Швидкість і простота випромінювання «Ray casting» пояснюється обчисленням кольору світла, не рекурсивно відстежуючи додаткові промені, які відбирають випромінювання, що падає в точку, в яку потрапив промінь. Це виключає можливість точного відображення відбитків, заломлення або природного випадання тіней; однак усі ці

елементи можуть бути підроблені певною мірою за допомогою творчого використання текстурних карт або інших методів. Висока швидкість обчислення зробила випромінювання зручним методом візуалізації у ранніх 3D-іграх у реальному часі.

У природі джерело світла випромінює промінь світла, який рухається, врешті-решт, до поверхні, яка перериває її прогрес. Про цей «промінь» можна думати як про потік фотонів, що подорожує тією ж доріжкою. У цей момент з цим світловим променем може статися будь – яке поєднання трьох речей: поглинання, відбиття та заломлення. Поверхня може відображати весь або частину променя світла в одному або декількох напрямках. Він також може поглинати частину світлового променя, що призводить до втрати інтенсивності відбитого та / або заломленого світла. Якщо поверхня має якісь прозорі або напівпрозорі властивості, вона заломлює частину світлового променя в себе в іншому напрямку, поглинаючи деякий (або весь) спектр (і, можливо, змінюючи колір). Між поглинанням, відбиттям і заломленням все світло що повинно враховуватися, і не більше. Наприклад, поверхня не може відображати 66% променів світла і заломлювати 50%, оскільки два склали б 116%. Звідси відбиті та / або заломлені промені можуть вражати інші поверхні, де їх поглинаючі, заломлюючі та відбиваючі властивості знову обчислюються на основі вхідних променів. Деякі з цих променів подорожують таким чином, що вони потрапляють у наше око, змушуючи нас бачити сцену і так сприяють остаточному виведеному зображенню. Спроба моделювати цей реальний процес відстеження світлових променів за допомогою комп’ютера може вважатися надзвичайно марнотратною, оскільки лише незначна частка променів у сцені насправді досягає очей [25].

Перший алгоритм відливання променів, використаний для візуалізації, був представлений Артуром Апелем у 1968 році. Ідея випромінювання променів полягає в тому, щоб простежити промені від очей, по одному на піксель, і знайти

найближчий об’єкт, що перекриває шлях цього променя подумайте про зображення як заслінку, при цьому кожен квадрат на екрані є пікселем. Це об'єкт, який око бачить через цей піксель. Використовуючи властивості матеріалу та ефект вогнів на сцені, цей алгоритм може визначити затінення цього об'єкта. Спрощується припущення, що якщо поверхня стикається зі світлом, світло дістанеться до цієї поверхні і не буде заблокованим або в тіні. Затінення поверхні обчислюється за допомогою традиційних моделей затінення комп'ютерної графіки 3D. Однією з важливих переваг рентгенівського лиття, запропонованих у порівнянні зі старими алгоритмами сканування, була його здатність легко працювати з непланарними поверхнями і твердими тілами, такими як конуси та сфери. Якщо математичну поверхню можна перетинати промінням, її можна відтворити за допомогою випромінювання. Опрацьовані об'єкти можуть бути створені за допомогою методів твердого моделювання та їх легко відобразити.

4.3 Ray Tracing Rendering

У комп’ютерній графіці трасування променів – це техніка візуалізації для генерування зображення шляхом відстеження шляху світла як пікселів у площині зображення та моделювання наслідків його зустрічей з віртуальними об’єктами. Метод здатний виробляти дуже високий ступінь візуального реалізму, більш високий ніж метод «Scanline Rendering», але з більшими обчислювальними витратами. Це робить трасування променів найкраще підходящим для програм, де відносно тривалий час візуалізації кадру може бути допущено, наприклад, для нерухомих зображень та відео та телевізійних візуальних ефектів, і більш погано підходить для програм у режимі реального часу, таких як відеоігри, де швидкість критичний [26].

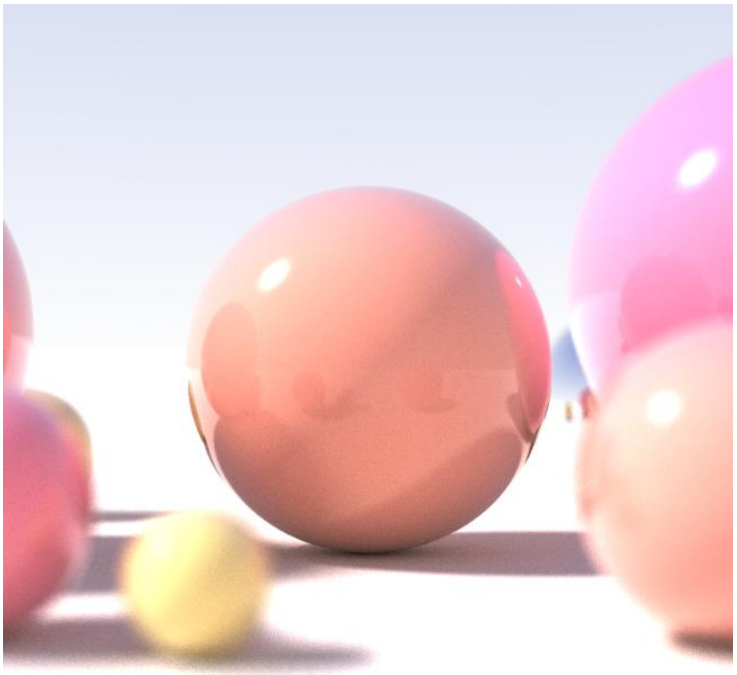


Рисунок 4.2 – візуалізація зображення за допомогою алгоритму «Ray Tracing»

Трасування променів здатне імітувати широкий спектр оптичних ефектів, таких як явища відбиття та заломлення, розсіювання та дисперсії (наприклад, хроматична аберація). На рисунку 4.2 можна побачити як сфера демонструє ефект малої глибини поля, джерела світла та дифузійного відбиття за допомогою рекурсивного трасування променів.

Оптичне трасування променів описує спосіб отримання візуальних зображень, побудованих у середовищі 3D – комп'ютерної графіки, з більшою кількістю фотореалізму, ніж методики "Ray casting" або методів "Scanline rendering". Він працює, відстежуючи шлях від уявного ока через кожен піксель на віртуальному екрані та обчислює колір об'єкта, видимого через нього.

Сцени в трасування променів описуються математично програмістом або візуальним художником (як правило, використовуючи посередницькі інструменти). Сцени також можуть містити дані із зображень та моделей, знятих такими засобами, як цифрова фотографія.

Зазвичай кожен промінь повинен бути перевірений на перетин з деякою підмножиною всіх об'єктів на сцені. Після виявлення найближчого об'єкта алгоритм оцінить вхідне світло в точці перетину, вивчить матеріальні властивості об'єкта та об'єднає цю інформацію для обчислення кінцевого кольору пікселя. Певні алгоритми освітлення, світловідбиваючі або напівпрозорі матеріали можуть вимагати повторного викидання на сцену більше променів.

Спочатку може здатися проти-інтуїтивним або "зворотним" відправляти промені подалі від камери, а не в неї (як реально світло робить насправді), але це робиться на багато порядків ефективніше. Оскільки переважна більшість променів світла від даного джерела світла не потрапляє прямо в очі глядача, моделювання "вперед" потенційно може витратити величезну кількість обчислень на світлових шляхах, які ніколи не записуються.

Отже, ярлик, який застосовується при відстеженні променів, передбачає, що даний промінь перетинає кадр огляду. Після максимальної кількості відбитків або променя, який проходить певну відстань без перетину, промінь перестає подорожувати, а значення пікселя оновлюється на рисунку 4.3 зображено як алгоритм «Ray tracing» будує зображення поширюючи промені на сцену [29].

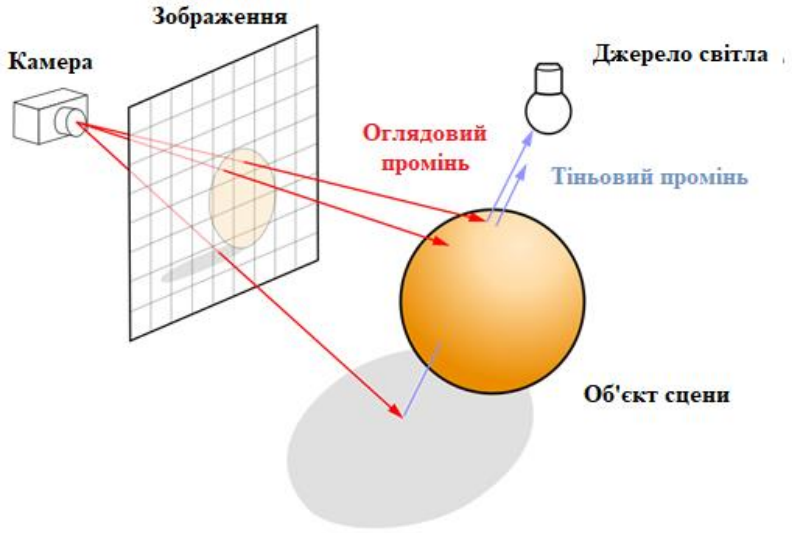


Рисунок 4.3 – Подубова зображення алгоритмом «Ray Tracing»

4.4 Висновки до четвертого розділу

В даному розділі було розглянути техніки візуалізації зображення в комп’ютерній графіці для фотореалістичної та нефотореалістичної графіки. Наведено основні методи візуалізації зображення, їх переваги та недоліки.

Зображення це результат рендерингу який описаний як набір певних візуальних особливостей, що відповідають справжнім фізичним явищам, властивостям об'єкту. Дослідження та розробки у області рендерингу продовжують шукати найкращі шляхи для більш кращої та ефективної їх симуляції. Деякі з цих особливостей можуть бути прив'язані до конкретного алгоритму або методу, інші ж являти їх сукупність.

5 ОБҐРУНТУВАННЯ ЕКОНОМІЧНОЇ ЕФЕКТИВНОСТІ

Метою розділу є обґрунтування економічної ефективності розробки архітектури графічного рушія на основі об’єктного підходу та реалізації даної розробки.

Щоб виконати оцінку економічної ефективності необхідно розрахувати трудомісткість реалізації проекту, витрати на оплату праці найманим працівникам, витрати апаратного і програмного забезпечення, амортизаційні відрахування, витрати енергоресурсів та інші витрати які є основними пунктами виконання обчислень, а також показники економічної ефективності розробки проекту.

5.1 Розрахунок норм часу на виконання науково-дослідної роботи

Ефективне використання часу має велике значення тому, що коефіцієнт корисної дії залежить від оптимального використання часу.

Кожен із етапів реалізації проекту характеризується метою та змістом, оцінкою часу виконання, кількістю та спеціалізацією виконавців, а також приблизною оцінкою вартості.

Розробку архітектури графічного рушія поділено на декілька етапів, що дозволяє полегшити і структурувати виконання роботи.

Основні етапи при виконанні

1. Формування списку програмного забезпечення яке потрібно встановити.
2. Встановлення програмного забезпечення.
3. Налаштування операційної системи
4. Розробка модуля ядра.
5. Розробка модуля графіки.

6. Розробка графічного конвеєра.

7. Тестування.

Для оцінки тривалості виконання окремих робіт використовують нормативи часу.

Виконавцем усіх операцій по розробці архітектури графічного рушія виступає інженер – програміст.

Витрати часу по окремих операціях технологічного процесу відображені в таблиці 5.1.

Таблиця 5.1 – Операції технологічного процесу та їх час виконання

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1	Формування списку програмного забезпечення яке потрібно встановити.	Інженер-програміст	9
2	Встановлення програмного забезпечення	Інженер-програміст	8
3	Налаштування операційної системи	Інженер-програміст	10
4	Розробка модуля ядра	Інженер-програміст	65
5	Розробка модуля графіки	Інженер-програміст	65
6	Розробка графічного конвеєра	Інженер-програміст	90
7	Тестування	Інженер-програміст	20
Разом			267

Загальні затрати часу на реалізацію архітектури графічного рушія на основі об’єктного підходу становить 267 години, найбільше часу витрачено на створення графічного конвеєра – 90 годин.

5.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

Відповідно до Закону України «Про оплату праці» заробітна плата – це «винагорода, обчислена, як правило, у грошовому виразі, яку власник або уповноважений ним орган виплачує працівникові за виконану ним роботу».

Розмір заробітної плати залежить від складності та умов виконуваної роботи, професійно-ділових якостей працівника, результатів його діяльності. Заробітна плата складається з основної та додаткової оплати праці.

Основна заробітна плата нараховується за виконану роботу за тарифними ставками, відрядними розцінками чи посадовими окладами.

Додаткова заробітна плата – це складова заробітної плати працівників, до якої включають витрати на оплату праці, не пов’язані з виплатами за фактично відпрацьований час. Нараховують додаткову заробітну плату залежно від досягнутих і запланованих показників, кваліфікації виконавців. Джерелом додаткової оплати праці є фонд матеріального стимулювання, який створюється за рахунок прибутку.

При розрахунку заробітної плати кількість робочих днів у місяці слід в середньому приймати – 24,5 дні/міс., або ж 196 год./міс. (тривалість робочого дня – 8 год.).

Наймані працівники для розробки архітектури графічного рушія працюють згідно контракту, в якому вказано їхню погодинну ставку. Тобто розрахунок заробітної плати працівників відбуватиметься на базі тарифної ставки.

Тарифна ставка розробника архітектури графічного рушія на основі об’єктного підходу:

- Інженер – програміст – 107 грн./год

Основна заробітна плата розраховується за формулою:

$$Зосн. = Тс \cdot Кг, \quad (5.1)$$

де Тс – тарифна ставка, грн.; Кг – кількість відпрацьованих годин.

Оскільки всі види робіт виконує інженер – програміст, то основна заробітна плата буде розраховуватись тільки за однією формулою.

$$Зосн.= 107 \cdot 267 = 25{,}569 \text{ грн.}$$

Додаткова заробітна плата становить 10–15 % від суми основної заробітної плати.

$$Здод. = Зосн. \cdot Кдопл. \quad (5.2)$$

де Кдопл – коефіцієнт додаткових виплат працівникам, 0,1–0,15 (візьмемо його рівним 0,15).

$$Здод = 25{,}569 \cdot 0{,}15 = 3{,}835.0 \text{ грн.}$$

Звідси загальні витрати на оплату праці (Во.п.) визначаються за формулою:

$$Во.п. = Зосн. + Здод. \quad (5.3)$$

$$Во.п. = 25{,}569 + 3{,}835= 29{,}404 \text{ грн.}$$

Крім того, слід визначити відрахування на соціальні заходи:

1) єдиний соціальний внесок ЄСВ(прибутковий податок) – 22%;

2) військовий збір – 1,5%.

у сумі зазначені відрахування становлять 23,5 %.

Отже, сума відрахувань на соціальні заходи буде становити:

$$Вс.з. = Фоп \cdot 0,235 \quad (5.4)$$

де Фоп – фонд оплати праці, грн.

$$Вс.з. = 29,404 \cdot 0,235 = 6,909 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці наведено у таблицю 5.2.

Таблиця 5.2 – Розрахунки витрат на оплату праці

№ з/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	Нарахув. на ФОП, грн.	Всього витрати на плату праці, грн. (6=3+4+5)
		Тарифна ставка, грн.	Кількість відпрацьованих год.	Фактично нарах. з/пл., грн.			
А	Б	1	2	3	4	5	6
1.	Інженер-програміст	107	267	25,569	3,835	6,909	36,687

З таблиці розрахунки витрат на оплату праці видно, що всього витрати на плату праці становить 36,687 грн.

5.3 Розрахунок матеріальних витрат

Матеріальні витрати визначаються як добуток кількості витрачених матеріалів та їх ціни:

$$Мві = qi \cdot pi \quad (5.5)$$

де: qi– кількість витраченого матеріалу і–го виду; pi – ціна матеріалу і –го виду.

Звідси, загальні матеріальні витрати можна визначити:

$$Зм.в. = \sum Мві. \quad (5.6)$$

Розрахунки занесемо у таблицю 5.3.

Таблиця 5.3 – Розрахунки матеріальних витрат

Найменування матеріальних ресурсів	Один. виміру	Норма витрат	Ціна за один., грн.	Затрати матер., грн.	Транспортно–заготівельні витрати, грн.	Загальна сума витрат на матер., грн.
1. Основні матеріали						
Глобальна мережа Internet	Мбайти	–	–	200	–	400

Загальні матеріальні витрати на Internet становлять 400 грн.

5.4 Розрахунок витрат на електроенергію

Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$Зв = W \cdot T \cdot S \quad (5.7)$$

де W – необхідна потужність, кВт; T – кількість годин на реалізацію розробки; S – вартість кіловат-години електроенергії.

Вартість кіловат-години електроенергії слід приймати згідно існуючих на даний час тарифів. Отже, 1 кВт з ПДВ коштує 2,42 грн.

Потужність комп'ютера для розробки архітектури графічного рушія на основі об'єктного підходу – 400 Вт, кількість годин роботи обладнання згідно таблиці 5.1 –267 годин.

Тоді, Зв = 0,4 · 267 · 2,42 = 258,45 грн.

Згідно формули затрати на електроенергію де необхідна потужність множиться на кількість годин на розробку архітектури графічного рушія на основі об'єктного підходу і множиться на вартість кіловат-години електроенергії що в висновку дорівнює 25,168 грн.

5.5 Розрахунок суми амортизаційних відрахувань

Характерною особливістю застосування основних фондів у процесі розробки архітектури графічного рушія на основі об'єктного підходу є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їхнього повного відновлення.

Для визначення амортизаційних використовується формула:

$$A = \frac{Б_В \cdot Н_А}{100} \quad (5.8)$$

де А – амортизаційні відрахування за звітний період, грн.; БВ – балансова вартість групи основних фондів на початок звітного періоду, грн.; НА – норма амортизації.

Комп'ютери та оргтехніка належать до четвертої групи основних фондів. Для цієї групи річна норма амортизації дорівнює 60 % (квартальна – 15 %).

Для розробки архітектури на основі об'єктного підходу засобом розробки є комп'ютер. Його сума становить 35000 грн. Отже, амортизаційні відрахування будуть рівні:

$$А = 35000 \cdot 5\% / 100\% = 1750, \text{ грн.}$$

5.6 Обчислення накладних витрат

Накладні витрати пов'язані з обслуговуванням апаратури та створенням необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб'єкта, накладні витрати можуть становити 20 – 60 % від суми основної та додаткової заробітної плати працівників.

$$Нв = Во.п. \cdot 0,2…0,6 \quad (5.9)$$

де Нв – накладні витрати.

Отже, накладні витрати:

$$Нв = 29,404 \cdot 0,2 = 5,880 \text{ грн.}$$

Накладні витрати згідно розрахунку формули на підтримку апаратури для розробки архітектури графчного рушія на основі об'єктног підходу, становить 6,570 грн.

5.7 Складання кошторису витрат та визначення собівартості науково-дослідницької роботи

Результати проведених вище розрахунків зведемо у таблицю 5.4.

Таблиця 5.4 – Кошторис витрат на НДР

Зміст витрат	Сума, грн.	В % до загальної суми
Витрати на оплату праці	25,569	60,39
Відрахування на соціальні заходи	6,909	16,32
Витрати на електроенергію	150,4	1,94
Амортизаційні відрахування	1,750	5,49
Накладні витрати	5,880	13,89
Собівартість	56,009	100,00

Собівартість (Св) розробки архітектури графічного рушія на основі об’єктного підходу розрахуємо за формулою:

$$Св = Во.п. + Вс.з. + Зм.в. + Зв + А + Нв \quad (5.10)$$

Отже, собівартість розробки архітектури графічного рушія на основі об’єктного підходу дорівнює:

$$Св = 25,569 + 6,909 + 150,4 + 1750 + 5,880 = 40,658 \text{ грн.}$$

Загальний кошторис витрат та визначення собівартості науково-дослідницької роботи становить 40,658 грн.

5.8 Розрахунок ціни програмного продукту

Ціну розробки архітектури графічного рушія на основі об’єктного підходу можна визначити за формулою:

$$Ц = \frac{C_B \cdot (1 + Р_{рен}) + К \cdot В_{н.і.}}{К} \cdot (1 + ПДВ) \quad (5.11)$$

де Ррен. – рівень рентабельності, 30 %; К – кількість замовлень, од. (встановлюється лише при розробці програмного продукту та мікропроцесорних систем); Вн.і. – вартість носія інформації, грн. (встановлюється лише при розробці програмного продукту); ПДВ – ставка податку на додану вартість, (20 %).

Оскільки розробка є прикладною, і використовуватиметься тільки для одного підприємства, то для розрахунку ціни не потрібно вказувати коефіцієнти К та Ві.н , оскільки їх в даному випадку не потрібно.

Тоді, формула для обчислення ціни розробки буде мати вигляд:

$$Ц = С_В \cdot (1 + Р_{рен}) \cdot (1 + ПДВ) \quad (5.12)$$

Звідси ціна на роботу складе:

$$Ц = 40{,}658 \cdot (1 + 0{,}3) \cdot (1 + 0{,}2) = 63{,}426 \text{ грн.}$$

Загальний розрахунок ціни архітектури графічного рушія на основі об’єктного підходу становить 63,426 грн.

5.9 Визначення економічної ефективності і терміну окупності капітальних вкладень

Ефективність виробництва – це узагальнене і повне відображення кінцевих результатів використання робочої сили, засобів та предметів праці на підприємстві за певний проміжок часу. Економічна ефективність (Ер) полягає у відношенні результату виробництва до затрачених ресурсів:

$$E_р = \frac{П}{С_В} \quad (5.13)$$

де П – прибуток; СВ – собівартість. Плановий прибуток (Ппл) знаходимо за формулою:

$$Ппл = Ц – Св . \quad (5.14)$$

Розраховуємо плановий прибуток:

Ппл = 63,426 – 40,658 = 22,678 грн.

Отже, формула для визначення економічної ефективності набуде вигляду:

$$E_р = \frac{П_{пл}}{Св} \quad (5.15)$$

Тоді, Ер = 22,678 / 40,658 = 0,56.

Поряд із економічною ефективністю розраховують термін окупності капітальних вкладень (Тр):

$$T_р = \frac{1}{E_р} \quad (5.16)$$

Термін окупності дорівнює: Тр = 1 / 0,56 = 1,7 р.

Згідно формул плановий прибуток від розробки архітектури графічного рушія на основі об’єктного підходу становить 22,678 грн., економічна ефективність дорівнює 0,56 а термін окупності становить 1,7 роки що вважається доцільним та економічно вигідним.

5.10 Висновок до п’ятого розділу

В обґрунтуванні економічної ефективності магістерської роботи освітнього рівня «магістр» було розраховано основні техніко-економічні показники розробки архітектури на основі об’єктного підходу (див. таблиця 5.5). Орієнтоване значення економічної ефективності становить 0,56 що є достатньо високим значенням.

Період окупності повинен варіюватися від 1 до 3 років, тоді розвиток вважається доцільним та економічно вигідним. Термін окупності архітектури графічного рушія на основі об’єктного підходу становить 1,7 років.

Таблиця 5.5 – Техніко-економічні показники розробки архітектури графічного рушія но основі об’єктного підходу

№	Показник	Значення
1	Собівартість, грн	40,658
2	Плановий прибуток, грн.	22,678
3	Ціна, грн.	63,426
4	Економічна ефективність	0,56
5	Термін окупності, рік	1,7

Отже, розробка архітектури графічного рушія на основі об’єктного підхожу може бути реалізована та розвинена, оскільки вона є економічно вигідною для всіх основних технічних та економічних показників.

6 ЕКОЛОГІЯ

6.1 Застосування екологічних знань у різних галузях соціально-політичного життя

В найважливіших міжнародних документах останнього десятиріччя, присвячених проблемам навколишнього середовища і гармонійного розвитку людства велика увага приділяється екологічній культурі і свідомості, інформованості людей про екологічну ситуацію в світі, регіоні, на місці проживання, їх обізнаності з можливими шляхами вирішення різних екологічних проблем, з концептуальними підходами до збереження біосфери і цивілізації [59]. Шлях до високої екологічної культури лежить через ефективну екологічну освіту.

Екологічна освіта на порозі 3-го тисячоліття стала необхідною складовою гармонійного, екологічно безпечного розвитку. Екологічне виховання і інформування населення, підготовка висококваліфікованих фахівців названі в програмних документах найвизначнішого міжнародного форуму 20-го сторіччя в Ріо-де-Жанейро (1992), присвяченого навколишньому середовищу і сталому розвитку, одним з найважливіших і необхідних засобів здійснення переходу до гармонійного розвитку всіх країн світу. Це положення підкреслюється і в останніх міжнародних документах (міжнародний звіт "Ріо+5", "Керівництво з підготовки національних доповідей про виконання країнами "Порядку денного на 21 сторіччя" та ін.).

Концепція екологічної освіти України, як елемент концепції гармонійного розвитку держави, набуває сьогодні ваги актуального і важливого державного документа.

Підготовка громадян з високим рівнем екологічних знань, екологічної свідомості і культури на основі нових критеріїв оцінки взаємовідносин людського суспільства й природи (не насильство, а гармонійне співіснування з нею!), повинна

стати одним з головних важелів у вирішенні надзвичайно гострих екологічних і соціально-економічних проблем сучасної України.

Екологічна освіта, як цілісне культурологічне явище, що включає процеси навчання, виховання, розвитку особистості, повинна спрямовуватися на формування екологічної культури, як складової системи національного і громадського виховання всіх верств населення України (у тому числі через екологічне просвітництво за допомогою громадських екологічних організацій), екологізацію навчальних дисциплін та програм підготовки, а також на професійну екологічну підготовку через базову екологічну освіту.

Вирішення цих питань має забезпечити формування цілісного екологічного знання й мислення, необхідних для прийняття екологічно-обгрунтованих народногосподарських рішень на рівні підприємств, галузей, регіонів, країни загалом [64].

Реформування екологічної освіти та виховання має здійснюватися з обов'язковим врахуванням екологічних законів, закономірностей, наукових принципів, що діють комплексно в біологічній, технологічній, економічній, соціальній і військовій сферах.

Глибоким опануванням екологічними знаннями, формуванням екологічного мислення, свідомості і культури мають бути охоплені громадяни всіх категорій, вікових груп і сфер діяльності.

Збалансований, екологічно безпечний (гармонійний) розвиток повинен бути базисною, вихідною ідеєю, методологічною основою екологічної освіти згідно з міжнародними вимогами.

Головними складовими системи екологічної освіти та виховання мають бути її формальна й неформальна частини, форми й методи яких різні, а мета одна: різнобічна підготовка громадян, здатних визначати, розуміти й оптимально вирішувати екологічні та соціально-економічні проблеми регіонів проживання на

основі наукових знань процесів розвитку біосфери, здорового глузду, загальнолюдських досвіду й цінностей.

Базою для здійснення заходів по вирішенню цієї важливої і складної державної проблеми повинна стати Концепція екологічної освіти в Україні.

Концепція складена з урахуванням сучасного стану і перспектив розвитку суспільного знання, спрямована на перебудову змісту освіти й виховання відповідно до вимог часу та основних положень Національної доктрини розвитку освіти у XXI столітті та базується на сформульованій у Посланні Президента України до Верховної Ради України "Україна: поступ у XXI століття. Стратегія економічної та соціальної політики на 2000-2004 рр." (276а/2000) стратегії сталого розвитку України. При підготовці концепції були проаналізовані і враховані всі попередні (1991-2001 рр.) матеріали щодо реформування освітнього процесу в Україні, а також матеріали, наведені в урядових документах.

6.2 Статистичні показники екологічних явищ

Річні сторони екологічних явищ характеризуються за допомогою статистичних показників. З їх допомогою можна одержати різноманітні кількісні і якісні характеристики різних сторін соціально-екологічних явищ: об’єми та інтенсивність забруднень середовищам, склад забрудників [65].

Статистичний показник – узагальнююча характеристика для кількісного виміру екологічних явищ.

Кожен з статистичних показників мас три характеристики:

- визначеність, кількість і якість;
- модель розрахунку, екологічний зміст і числове значення змісту;
- адекватність відображення, точність вимірювання і достовірність

інформації зображена на рисунку 6.1.

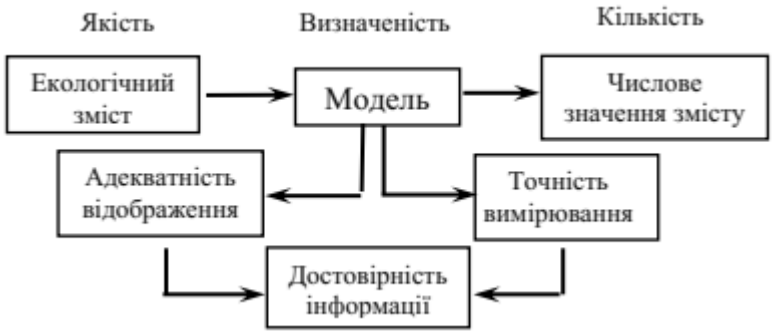


Рисунок 6.1 – Статистична модель показника

За допомогою статистичних показників вирішується одна з головних задач статистики: визначається кількісна сторона явища чи процесу у поєднанні з якісною стороною. Показник — кількісно-якісна характеристика соціально-екологічних явищ і процесів.

Якісна сторона показника відображає сутність явища або процесу в конкретних умовах місця й часу, а кількісна – розмір, абсолютну або відносну його величину, тобто числове значення та відповідну одиницю виміру, та іншими категоріями працівників магазину).

Якісний зміст показника залежить від суті досліджуваного явища (процесу) і відображається у назві показника (викиди, скиди, відходи) та поєднується з числовим вираженням за допомогою моделі показника, що містить ного назву, алгоритм розрахунку, одиницю, застосовувану під час вимірювання, а також місце та час. на який припадає явище чи процес.

Різноманіття явищ громадського життя. їхніх властивостей, руху, відносин обумовило й різноманіття статистичних показників. Показники поділяються на види залежно від їх аналітичної функції, способу обчислення та ознаки часу, виконання своїх функцій. Існує багато видів показників, що потребує їх упорядкування і класифікації.

Класифікація – це розчленовування показників за суттєвими познаками на

групи.

Ознаками класифікації виступають: спосіб вирахування, час. до якого відносяться показники, характер взаємозв'язку показників, зміст показників.

Первинні – визначаються шляхом зведення та групування даних і подаються у формі абсолютних явищ (наприклад, кількість джерел забруднення, об’єм видів та скидів). Похідні – обчислюються на базі первинних і мають форму середніх або відносних величин (наприклад, викиди в розрахунку на одну особу або одиницю площі). У процесі їх порівняння отримують похідні показники другого порядку (індекси).

Інтервальні – характеризують явище та певний період часу (місяць, квартал, рік): наприклад, середньомісячні викиди забруднюючих речовин в атмосферне повітря. Моменти – дають кількісну характеристик)' явищ на певний момент часу: на початок або кінець року): наприклад, залишок обігових коштів на початок місяця.. Інтервальні та моментні показники можуть бути як первинними, так і похідними.

Адитивні (підсумовуючі) показники – це всі абсолютні показники, що здатні підсумовуватися; неадитивні (моменті, відносні, середні) показники не можна підсумовувати.

Індивідуальні показники характеризують окрему одиницю статистичної сукупності – окреме явище, об’єкт і групові (частки) – групу одиниць того самого виду; загальні (зведені) – всю їхню сукупність.

За виконанням своїх функцій розглядають показники, що відбивають обсяг явища його середній рівень, інтенсивність прояву, структуру, аміну в часі або порівняння у просторі.

Розрізняють також показник якісний і показник об'ємний. Кожний конкретний статистичний показник має якісну визначеність, визначеність простору, визначеність часу й кількісну визначеність. Число, позбавлене хоча б

однієї із цих визначеностей. не є статистичним показником.

Сукупність статистичних показників називається системою показників.

Статистичний показник – найважливіша категорія статистичної науки. Він служить узагальнюючою кількісною характеристикою властивостей сукупності загалом та її частин зокрема. Статистичні показники поділяються на абслютні, відносні, середні велечини.

6.3 Висновок до шостого розділу

В даному розділі було розглянуто застосування екологічних знань у галузях соціально-політичного життя та розкрито питання екологічних явищ.

Статистика – це наука, яка вивчає і обробляє інформацію про всі суспільні явища. Вся статистична інформація передається за допомогою статистичних показників. Статистичний показник – це кількісно-якісна характеристика будь – якого соціально-економічного явища.

7 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

7.1 Охорона праці

7.1.1 Навчання з питань охорони праці

Навчання працівників з питань охорони праці повинно відбуватися відповідно до вимог Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці. У ст. 18 Закону України «Про охорону праці» [1] сказано, що працівники під час прийняття на роботу і в процесі роботи повинні проходити за рахунок роботодавця інструктаж, навчання з питань охорони праці, з надання домедичної допомоги потерпілим від нещасних випадків і правил поведінки у разі виникнення аварії.

Періодичність навчання з питань охорони праці на підприємстві

Якщо працівники, зайняті на роботах з підвищеною небезпекою або там, де є потреба у професійному доборі, вони мають щороку проходити спеціальне навчання і перевірку знань відповідних нормативно-правових актів з охорони праці. Таке навчання відбувається за рахунок роботодавця.

Один раз на 3 роки проходять навчання, а також перевірку знань з питань охорони праці за участю профспілок посадові особи, діяльність яких пов’язана з організацією безпечного ведення робіт, під час прийняття на роботу [2].

Працівники та посадові особи, які не пройшли навчання, інструктаж і перевірку знань з охорони праці до виконання роботи не допускаються. У тих, хто продемонстрував незадовільний рівень знань з питань охорони праці, є один місяць для того, щоб пройти повторне навчання і перевірку знань.

Організація навчання з охорони праці

Порядок навчання з питань охорони праці та перевірки знань посадових осіб визначається типовим положенням, що затверджується центральним органом

виконавчої влади, який забезпечує формування державної політики у сфері охорони праці.

Нова редакція наказу Держнаглядохоронпраці про затвердження Типового положення про порядок проведення навчання і перевірки знань з питань охорони праці та Переліку робіт з підвищеною небезпекою від 26.01.2005 № 15 набула чинності з 14 квітня 2017 року (далі – Типове положення). Це положення про навчання з охорони праці є чинним і у 2019 році [4]. Головні зміни, яких зазнав цей документ після останнього оновлення:

- усунули обмеження в організації навчання;
- скасували обов'язкову видачу посвідчень;

Типовим положенням передбачені такі види навчання з питань охорони праці:

- навчання з питань охорони праці на підприємстві;
- навчання з питань охорони праці у навчальних центрах;
- інструктажі з питань охорони праці;
- навчання з питань охорони праці посадових осіб;
- спеціальне навчання;
- стажування;
- дублювання.

Пункт 3 положення про навчання з охорони праці зобов'язує розробляти і затверджувати власні положення про навчання з питань охорони праці з урахуванням специфіки свого виробництва. При їх розробці потрібно спиратися на Типове положення та вимоги нормативно-правових актів з охорони праці. Також необхідно розробити план – графік навчання працівників з питань охорони праці та перевірки знань з питань охорони праці, які мають бути оприлюднені роботодавцем.

Навчання з охорони праці організовує і проводить (зокрема під час професійної підготовки, перепідготовки та підвищення кваліфікації на підприємстві) служба кадрів або інші спеціалісти, яким роботодавцем доручена організація цієї роботи [5].

Рівень знань з питань охорони праці на підприємстві перевіряє спеціальна комісія, склад якої затверджується наказом/розпорядженням роботодавця. Головою комісії з перевірки призначається керівник підприємства або його заступник, до службових обов'язків яких входить організація роботи з охорони праці.

До складу комісії підприємства входять спеціалісти служби охорони праці, представники юридичної, виробничих, технічних служб, представник профспілки або уповноважена найманими працівниками особа з питань охорони праці. Комісія вважається правочинною, якщо до її складу входять не менше трьох осіб.

Перелік питань для перевірки знань з охорони праці працівників, з урахуванням специфіки виробництва, складається членами комісії та затверджується роботодавцем.

Така перевірка може здійснюватися у вигляді:

- тестування;
- заліку;
- іспиту.

Результат перевірки знань з питань охорони праці оформлюється протоколом засідання комісії з перевірки знань з питань охорони праці.

Особам, які під час перевірки знань з охорони праці виявили задовільні результати, видається посвідчення про перевірку знань з питань охорони праці. При цьому в протоколі та посвідченні у стислій формі зазначається перелік основних нормативно-правових актів з охорони праці та з безпечного виконання конкретних видів робіт, в обсязі яких працівник пройшов перевірку знань.

Відповідальність за організацію і здійснення інструктажів, навчання та перевірки знань працівників з питань охорони праці покладається на роботодавця.

7.1.2 Вплив шуму на здоров′я та працездатність людини, гранично допустимий рівень шуму

Шум – це сукупність звуків різноманітної частоти та інтенсивності, що виникають у результаті коливального руху частинок у пружних середовищах (твердих, рідких, газоподібних). Шумом також вважають будь-який небажаний для людини звук.

Важливою характеристикою шуму є його частотний склад. Якщо в складі шуму переважають звуки з частотою коливань до 400 Гц, такий шум називається низькочастотним, якщо переважають звуки з частотою 400 – 1000 Гц – середньочастотним, якщо понад 1000 Гц – високочастотним.

Низькочастотний шум інтенсивністю до 100 дБ не викликає відчутної несприятливої дії на орган слуху, для середньочастотного шуму ця норма складає 85 – 90 дБ, для високочастотного – 75 – 85 дБ. Несприятливі суб’єктивні відчуття і вплив на організм людини зумовлює високочастотний шум [6].

Шум підступний, його шкідливий вплив на організм відбувається незримо, непомітно. Організм людини проти шуму практично беззахисний.

Вплив шуму на організм умовно поділяють на:

- специфічний, що спричиняє зміни в органі слуху;
- неспецифічний – з боку інших органів і систем.

Основну увагу приділяють стану органа слуху, тому що слуховий аналізатор першим сприймає звукові коливання і потерпає від впливу шуму на організм.

Дія шуму на організм людини пов’язана головним чином із застосуванням нового, високопродуктивного устаткування, з механізацією або автоматизацією трудових процесів.

Джерелами шуму можуть бути двигуни, насоси, компресори, пневматичні та електричні інструменти, молоти, дробарки, верстати, центрифуги та інше обладнання, що має рухомі деталі.

Крім того, за останні роки, у зв’язку із значним розвитком міського транспорту, зросла інтенсивність шуму і в побуті.

Короткочасний, навіть одноразовий вплив шуму високої інтенсивності може спричинити повну загибель спірального органа або розрив барабанної перетинки, що супроводжується почуттям закладеності та різким болем у вухах. Наслідком баротравми нерідко буває повна втрата слуху.

Основною ознакою впливу шуму є зниження слуху по типу кохлеарного невриту. Професійне зниження слуху зазвичай буває двостороннім. Стійкі зміни слуху, як правило, розвиваються повільно, нерідко їм передує адаптація до шуму, яка характеризується нестійким зниженням слуху, що виникає безпосередньо після його впливу і зникає після припинення його дії.

Гігієнічна регламентація шумів ґрунтується на критерії збереження здоров'я та працездатності людини. Гранично допустимі рівні шуму на виробництві мають забезпечувати функціонування фізіологічних систем організму в межах адаптаційних можливостей на весь час трудової діяльності. Чинні на цей час гігієнічні нормативи, які регламентують допустимі рівні шуму, інфразвуку та ультразвуку, побудовані на єдиному енергетичному принципі і практично включають увесь частотний діапазон акустичних коливань, що впливають на людину [7].

Нормування шуму здійснюється згідно з ГОСТом 12.1.003 – 83* ССБТ.

Шум на робочих місцях не повинен перевищувати допустимих рівнів, значення яких наведені у ГОСТІ 12.1.003.83*.

Як нормативний рівень шуму на постійних робочих місцях та на території підприємств запроваджено гранично допустимий рівень звуку 80 дБА, який

забезпечує відсутність ризику втрати слуху і практично не впливає на працездатність та стан здоров'я. Для зменшення шуму в житлових будинках у державних санітарних нормах на інженерне обладнання та електропобутову техніку запроваджуються вимоги з обмеження шумності.

Для забезпечення нормальних умов праці та відпочинку людей ГОСТ 27436 та ОСТ 27.004.022-86 нормують основний для міської забудови шум транспорту, який не має перевищувати: для легкових автомобілів 77 дБА, вантажних автомобілів – 79-84 дБА; автобусів – 83 дБА; мотоциклів, моторолерів та мопедів - 85 дБА.

Унормовано шумові характеристики місць перебування людей. Рекомендуються такі діапазони звукового тиску всередині приміщень: для сну, відпочинку – 30 – 45 дБ; для розумової праці – 45 – 55; для лабораторних досліджень, роботи з персональним комп’ютером – 50 65; для виробничих цехів, магазинів, гаражів – 56 – 70 дБ.

Нормальний шум навколишнього середовища варіює в межах 35 – 60 дБ. Але до цього фону додаються все нові децибели внаслідок чого рівень шуму часто перевищує 100 дБ.

7.2 Безпека в надзвичайних ситуаціях

7.2.1 Проведення рятувальних та інших невідкладних робіт на об’єкті господарської діяльності в осередку ураження (зараження)

Унаслідок надзвичайних ситуацій у населених пунктах країни і на підприємствах можуть виникнути руйнування, зараження місцевості радіоактивними та хімічними речовинами. Люди можуть опинитися у завалах, пошкоджених та палаючих будинках, інших непередбачуваних ситуаціях. У зв'язку з цими обставинами буде потрібне проведення заходів із рятування людей, надання

їм допомоги, локалізації аварій та усунення пошкоджень. При вирішенні цих проблем виходять з того, що в осередках ураження і районах лиха будуть проводитися не тільки суто рятувальні роботи, а й деякі невідкладні, що не пов'язані з рятуванням людей. Рятувальні та інші невідкладні роботи (РіНР) проводяться з метою порятунку людей та надання допомоги ураженим, локалізації аварій та усунення пошкоджень, створення умов для наступного проведення відновлювальних робіт. При проведенні РіНР великого значення має дотримання певних умов. Такими умовами є: своєчасне створення угруповань, сил, що залучаються для проведення РіНР, своєчасне ведення розвідки, швидкий рух і введення сил у осередок ураження, безперервне проведення РіНР до їх повного завершення, тверде й оперативне управління силами, що залучаються до проведення РіНР, і всебічне забезпечення їх діяльності [75].

Заходи, що відносяться до рятувальних робіт:

- локалізація і гасіння пожеж;
- пошук і рятування людей з-під завалів;
- подача повітря у завалені захисні споруди;
- надання ураженим першої медичної допомоги та їх евакуація;
- санобробка людей та знезараження їх одягу;
- знезараження місцевості, споруд, техніки.

Крім рятувальних робіт, в осередках ураження проводяться невідкладні роботи, до яких відносяться:

- прокладання маршрутних шляхів на заражених територіях і будування проїздів у завалах;
- локалізація аварій на комунально-енергетичних мережах, лініях зв'язку та їх відновлення;
- закріплення або ліквідація конструкцій споруд, які загрожують падінням та перешкоджають проведенню рятувальних робіт;

- ліквідація боєприпасів та інших вибухонебезпечних предметів (балони з газом, бочки з бензином тощо).

Керівництво проведенням усіх цих робіт у надзвичайних ситуаціях проводяться надзвичайними комісіями держави, області, міста тощо.

При аваріях на об'єктах народного господарства, установах, якщо їх наслідки не виходять за межі об'єктів захисних зон, керівництво роботами проводиться адміністрацією підприємств.

Виникнення стихійних лих, а також аварій та катастроф можна в деяких випадках прогнозувати. Ці прогнози, як правило, закладаються в плани ЦО підприємств, установ, що передбачають попереджувальні заходи, які повинні зменшити наслідки аварій і катастроф.

Характер та обсяг таких заходів залежать від виду та рівня аварії або стихійного лиха, масштабів і часу їх виникнення [76].

Загалом до таких заходів відносяться:

- приведення в готовність засобів захисту;
- перевірка готовності систем оповіщення;
- підготовка і видача населенню засобів індивідуального захисту та особистої профілактики;
- підготовка до евакуації або відселення та їх проведення;
- вивезення матеріальних цінностей;
- захист продуктів харчування, джерел води тощо;

Способи і послідовність проведення цих робіт залежать від обставин, що склались у районі аварії чи катастрофи, та наявності сил і засобів для проведення таких рабіт.

Ліквідація наслідків надзвичайної ситуації проводиться для відновлення роботи підприємств, організацій, навчальних закладів тощо.

При ліквідації наслідків надзвичайної ситуації здійснюються такі заходи:

- розвідка осередків надзвичайних ситуацій;
- локалізація і гасіння пожеж;
- відбудівля споруд і шляхів сполучення;
- проведення ізоляційне обмежених заходів в осередках інфекційного зараження;
- проведення спецобробки населення;
- дезактивація, дегазація техніки, майна, доріг, місцевості тощо.

Розвідку осередків надзвичайних ситуацій проводять сили Збройних сил, ЦО і невоєнізовані формування підприємств, організацій тощо.

Воєнізовані сили розвідки ЗС і ЦО включають підрозділи радіаційної, хімічної, біологічної та інженерної розвідок. У завдання цих підрозділів входить виявлення загального стану в осередках та визначення меж ураження, руйнування, повені і пожеж, а також виставлення спостереження на особливо важливих напрямках (станціях, переправах, перехресті доріг тощо) [80].

Сили і засоби для проведення рятувальних робіт.

Для проведення рятувальних робіт залучаються невоєнізовані формування ЦО, військові частини і підрозділи, медорганізації тощо Невоєнізовані формування мають у першу чергу проводити рятувальні роботи на об'єктах народного господарства.

До них входять формування загального призначення, які мають у своєму складі аварійно-технічні формування, формування механізації робіт тощо. Вони можуть бути посилені протипожежними, дорожніми, автомобільними та іншими підрозділами.

Від швидкості та рішучості дій формувань залежить життя багатьох людей та збереження матеріальних цінностей.

Під стійкістю роботи об'єкта народного господарства розуміють здатність підприємства, установи попереджувати виникнення виробничих аварій, катастроф,

протистояти впливу уражаючих факторів, аби запобігти або зменшити загрозу життю і здоров'ю робітників і службовців, матеріальних втрат, а також забезпечити відновлення порушеного виробництва в мінімально короткий термін.

7.2.2 Вплив електромагнітного імпульсу (EMI) ядерного вибуху на елементи виробництва та заходи захисту

EMI здатний викликати потужні імпульси струмів і напруг у проводах і кабелях повітряних і підземних ліній зв'язку, сигналізацій, управління, електропередачі, в антенах радіостанцій.

Вплив EMI може призвести до згорання чутливих електронних і електричних елементів, пов'язаних з великими антенами чи відкритими проводами, а також до серйозних порушень в цифрових і контрольних пристроях, зазвичай без необоротних змін. Для найбільш важливих пристроїв треба застосовувати заходи захисту та підвищувати їх стійкість до EMI.

Ступінь ушкодження залежить в основному від амплітуди наведеного імпульсу напруги чи струму і електричної міцності обладнання.

Особливо схильна до впливу EMI радіоелектронна апаратура, виконана на напівпровідникових та інтегральних системах, працюючих на малих струмах і напругах і, чутливих до впливу зовнішніх електричних і магнітних полів. EMI пробиває ізоляцію, випалює елементи електросхем радіоапаратури, викликає коротке замикання в радіопристроях, іонізацію діелектриків, спотворює або повністю стирає магнітний запис, позбавляє пам'яті ЕОМ.

Інженерно-технічні заходи мають забезпечити підвищену стійкість виробничих споруд, технологічних ліній, устаткування, комунікацій об'єкта до впливу уражаючих факторів під час надзвичайних ситуацій.

При проведенні цих заходів необхідно враховувати конкретні умови підприємства. Проте є загальні організаційні інженерно-технічні заходи, які мають проводитись на всіх об'єктах.

Захист цінного й унікального устаткування. Захистити цінне і унікальне устаткування можна завдяки проведенню інженерно-технічних заходів, щоб зменшити небезпеку пошкодження і руйнування цінного й унікального устаткування, комп’ютерної техніки, станків з програмним керуванням, шліфувальних, токарних, розточних, зубофрезерних, пресових станків, автоматичних конвеєрних ліній та іншого устаткування. Варіантами такого захисту є розміщення зазначеного устаткування в заглиблених приміщеннях, а також використання спеціальних захисних пристосувань, закріплення станків на фундаментах, застосування контрфорсів для підвищення стійкості проти перекидання обладнання

Забезпечення стійкості роботи паливно – енергетичного комплексу. Створення резерву енергетичних потужностей за рахунок автономних пересувних електростанцій, а також місцевих джерел електроенергії. Підготовка автономних електростанцій до роботи за спеціальним режимом (графіком) для забезпечення технологічних процесів виробництва, для яких неможливі тривалі перерви в електропостачанні. З метою попередження аварій на електричних мережах необхідно установити автоматичну систему відключення при виникненні перенапруги. Повітряні лінії електропостачання замінити на підземно-кабельні. Створення необхідних запасів (резервів) паливно-мастильних матеріалів та інших видів палива й організація їх безпечного зберігання. Щоб не допустити зупинки підприємства через дефіцит палива, необхідно підготуватись для роботи на різних видах палива: нафта, вугілля, газ.

Забезпечення стійкого постачання підприємства. Для забезпечення виробництва продукції необхідні електроенергія, паливо, мастила, запасні частини,

сировина та інші матеріально-технічні засоби. Забезпечення об'єктів цими ресурсами дасть можливість випускати необхідну продукцію в надзвичайних умовах мирного і воєнного часу. Тому повинні проводитись такі заходи, які б забезпечили стійкість постачання і сприяли підвищенню захисту мережі електро, водо, газопостачання, транспортних комунікацій і джерел постачання всім необхідним для забезпечення функціонування підприємства в надзвичайних умовах. З метою попередження аварій на електричних мережах необхідно встановити автоматичну систему відключення перенапруги. Повітряні лінії електропостачання слід замінити на підземно-кабельні. Запас резервних матеріалів необхідно розраховувати на такі строки роботи підприємства, за які можливе відновлення регулярного постачання. Передбачити, на випадок перебоїв в постачанні підприємствами-суміжниками, створення місцевих матеріалів, сировини для виготовлення комплектуючих виробів і інструментів силами свого підприємства.

Як захиститись від електромагнітного випромінювання?

В загальному, методи і засоби захисту від електромагнітних полів можна умовно розділити на інженерно-технічні, організаційні та лікувально-профілактичні.

Згідно зі встановленою процедурою, захист людини від такого небезпечного впливу повинен здійснюється такими способами:

- зменшення випромінювання від джерела;
- екранування джерела випромінювання та робочого місця;
- встановлення санітарно-захисної зони;

Отже, перший крок у цьому напрямку можна зробити не вдаючись до застосування спеціальних засобів захисту від електромагнітних полів та екрануючих матеріалів [79]. Оптимально розмістіть електроприлади вдома, збільшуйте відстань від електроприладу до спального місця, видаліть штучні

матеріали, зменшити навантаження, виконуйте елементарні правила техніки безпеки користування електроприладами, мобільними телефонами, засобами зв'язку. Однак цього не завжди достатньо. Якщо ж ви в групі ризику, якщо відчуваєте дискомфорт, пов'язаний з надмірним впливом ЕМВ на робочому місці чи вдома – застосування екрануючих матеріалів є життєво необхідним.

Вимірювання електромагнітного випромінювання необхідно провести перед тим, як розпочати роботи із захисту від ЕМП, оскільки саме вимірювання електромагнітного випромінювання покаже, що є основними джерелами випромінювання, відповідно, які об'єкти необхідно екранувати першочергово. Залежно від джерел випромінювання та того, де саме вони знаходяться – ззовні чи в середині приміщення, можна оптимально підібрати захист від ЕМП.

Якщо джерело забруднення електромагнітним випромінюванням знаходиться ззовні приміщення (трансформатора підстанція чи електрощитова, вишка стільникового зв'язку, тощо), надійним засобом захисту від електромагнітного випромінювання у цьому випадку є спеціально розроблена для захисту від ЕМП екрануюча сітка HEG10 фірми «Gigahertz Solutions». Цю сітку кріплять до стіни за допомогою спеціального клею, що також має екрануючі властивості. Сітка HEG10 забезпечує захист від ЕМП високих частот та від малих електричних полів, сітка потребує заземлення.

Якщо ж джерело ЕМП знаходиться в середині приміщення, оптимальним засобом захисту від електромагнітного випромінювання є екрануюча фарба. Надійний захист від електромагнітних полів забезпечує фарба для екранування ЕМП виробництва «Gigahertz Solutions». Вона не токсична, відтак, придатна до використання як в середині приміщення, так і ззовні.

Окрім функції захисту від електромагнітного випромінювання, ці засоби можна також використовувати і для захисту даних з комп'ютерних мереж та комп'ютерів. Окрім того, захист офісу від ЕМП ззовні дасть можливість

заблокувати зовнішні мережі WIFI, та захистити від проникнення ззовні власну мережу WIFI, що актуально для офісів, розміщених у великих офісних центрах.

7.3 Висновок до сьомого розділу

В даному розділі розглянуто навчання з охорони праці та безпеки життєдіяльності та вплив шуму на здоров'я та працездатність людини. Працівники при прийнятті на роботу та у процесі її виконання повинні проходити за рахунок роботодавця інструктажі та навчання з питань ОП. Працівники, зайняті на роботах підвищеної небезпеки, і посадові особи, діяльність яких пов'язана з організацією безпечного ведення робіт, повинні проходити спец-навчання і перевірку знань з питань ОП. При організації та проведенні інструктажів, навчання, а також перевірки знань з питань ОП працівників необхідно керуватися Типовим положенням № 15.

У бухгалтерському обліку витрати на проведення «праце – охоронного» навчання обліковують у складі загальновиробничих витрат або у складі поточних витрат діяльності залежно від місця роботи співробітника, який навчається.

Останнім часом проблемі шуму надають великої ваги. Є багато способів боротьби з ним: використання шумопоглинальних екранів, фільтрів, матеріалів, зміна технології виробництва, запровадження безшумних механізмів і деталей, зміна режиму, деталей та особливостей транспортних потоків у містах.

За результатами аналізу інформації про стан техногенної та природної безпеки у 2016 році населення, територія та навколишнє середовище в Україні знаходились під дією чинників, які комплексно та негативно впливають на життєдіяльність країни через виникнення надзвичайних ситуацій (НС) техногенного та природного характеру, а їх наслідки призводять до загибелі людей, економічних втрат та погіршення стану навколишнього природного середовища.

Наприкінці минулого і початку XXI століття проблема гарантування і підвищення безпеки у разі виникнення надзвичайних ситуацій стає однією з важливих соціально-політичних, економічних, соціально-демографічних та екологічних проблем. Ризик виникнення різного роду небезпек на території України залишається високим. Зростає масштабність наслідків аварій, катастроф і стихійних лих, що створює проблему запобігання виникненню надзвичайних ситуацій, ліквідації або мінімізації їх наслідків. Останніми роками зусиллями органів виконавчої влади, наукових установ розроблено і прийнято низку законодавчих і нормативно-правових актів, які регулюють діяльність у сфері запобігання і ліквідації надзвичайних ситуацій, нагромаджено значний Досвід у проведенні заходів з попередження аварій, катастроф і стихійного лиха та ліквідації їх наслідків, створена наукова база протидії цим негативним явищам. Проблема збереження життєдіяльності людини має глобальний характер, тому кожна людина повинна зробити свій внесок у її вирішення як заради сьогоднішнього суспільства, так і для добробуту прийдешніх поколінь.

ВИСНОВОК

У роботі розроблено архітектуру графічного рушія на основі алгоритму «Ray casting», яка дозволяє зменшити затрати на ресурси та покращити швидкість візуалізації зображення. Отримано наступні результати, реалізованому алгоритму для візуалізації 60 кадрів зображення потрібно 162 мілісекунди.

Провівши аналіз огляду літературних джерел по візуалізації зображення в реальному часы та дослідивши методи обробки зображення у графічних рушіях було модифіковано етапи роботи графічного конвеєра у графічному рушії, що дозволило зменшити затрати часу на візуалізацію зображення та вимоги до обсягу необхідних ресурсів, з використання мови програмування C++ було розроблено архітектуру на основі алгоритм для візуалізації даних в реальному часі «Ray casting».

Провівши аналіз швидкості роботи алгоритмів для візуалізації даних в реальному часі в графічних рушіях, було визначено доцільним використання архітектури на основі алгоритму «Ray casting» так як швидкість його роботи на 100 мілісекунд меньша ніж у аналогів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Архітектура OpenGL [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.opengl.org/about/arb/ – Дата доступу: 01.12.19
2. Майкл Віммер, Харальд Пірінгер та Вернер Пургатофер, «Когерентний ієрархічний кулінг: Застосування апаратних запитів», 2004 – 624 с.
3. І. Бак, Т. Фолі, Д. Хорн, «Потокові обчислення на графічному обладнанні, подані до транзакцій ACM з графіки», 2004 – 786 с.
4. NVIDIA Corporation, Інструментарій Cg [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://developer.nvidia.com/object/cg_toolkit.html – Дата доступу: 01.12.19
5. NVIDIA Corporation, Огляд CgFX [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://developer.nvidia.com/object/cg_users_manual.html – Дата доступу: 25.11.19
6. Джеймс Х. Кларк, «Ієрархічні геометричні моделі для алгоритмів видимих поверхонь, комунікації» 1976 – 554 с.
7. Khronos Group, Collada – 3D схема обміну ассетами [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.khronos.org/collada/ – Дата доступу: 20.11.19
8. Майкл Ф. Коен, Кріс Чху, Джон Р. Уоллес, «Радіознавство та реалістичний синтез зображень» 1993 – 889 с.
9. Майкл Дірінг, Стефані Віннер, Бік Схедіві, Кріс Даффі та Ніл, «Векторні шейдери: система VLSI для високоефективної графіки», 1988 – 979 с.
10. Microsoft Direct3D [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.microsoft.com/directx/ – Дата доступу: 05.11.19

11. Девід Еллсворт, «Паралельні архітектури та алгоритми для синтезу в режимі реального часу високоякісних зображень з використанням відкладеного затінення», 1990 – 950 с.

12. Джеймс Кларк, XML-аналізатор Expat [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://expat.sourceforge.net/ – Дата доступу: 15.11.19

13. Маркус Гігль та Майкл Віммер, «Проблема поєднання зображень та простору для плавного дискретного переходу LOD», 2007 – 678 с.

14. Шедерна мова OpenGL [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.opengl.org/documentation/glsl/ – Дата доступу: 19.11.19

15. Стефан Редон, «Інтерактивне виявлення зіткнень між складними моделями у великих середовищах з використанням графічного обладнання», 2015 – 695 с.

16. Шон Харгрівз, «Генерування шейдерів з фрагментів HLSL розширена візуалізація за допомогою DirectX та OpenGL», 2004 – 785 с.

17. Мережа розробників Microsoft, вступ до мови шейдерів DirectX 9 високого рівня [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://msdn2.microsoft.com/en-us/library/ms810449.aspx – Дата доступу: 15.10.19

18. Ніколас Гебхардт Irrlicht Engine - безкоштовний 3d-двигун з відкритим кодом [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://irrlicht.sourceforge.net/ – Дата доступу: 21.10.19

19. Sun Developer Network, Мова програмування Java [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://java.sun.com/ – Дата доступу: 15.11.19

20. Незалежна група JPEG, бібліотека для стиснення зображень JPEG [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.ijg.org/ – Дата доступу: 23.11.19

21. Джеймс Т. Каджія, «Рівняння візуалізації», 1986 – 335 с.

22. Канеко Т., Такахей Т., Інамі М., Кавакамі Н., Янагіда Я., Маеда Т.,Tachi S., «Детальне представлення фігури з паралелефоюванням», 201 – 485 с.

23. lib3ds ANSI-C бібліотека для 3ds моделей [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://lib3ds.sourceforge.net/ – Дата доступу: 10.10.19

24. Програмне забезпечення Rasterbar, бібліотека Luabind [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.rasterbar.com/products/luabind.html – Дата доступу: 28.10.19

25. Макдональд Дж. та Бут К. «Евристика для відстеження променів за допомогою ділення простору», 1990 – 335 с.

26. У. Марк, С. Гланвілл і К. Еклі «Cg: Система програмування графічного обладнання на С-подібній мові», 2003 – 555 с.

27. Microsoft Corporation, Microsoft Website [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.microsoft.com – Дата доступу: 23.11.19

28. Microsoft Corporation – MSDN Online, Довідки про ефекти [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://msdn.microsoft.com/archive/en-us/ – Дата доступу: 23.11.19

29. Microsoft Visual Studio Developer Center [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://msdn.microsoft.com/vstudio/ – Дата доступу: 23.11.19

30. NVIDIA Website [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.nvidia.com/ – Дата доступу: 24.11.19

31. Гарі Бішоп, Девід Макаллістер, «Картографування текстури рельєфу», 2000 – 369 с.

32. Веб-сайт SGI OpenGL, реєстр розширень OpenGL [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.opengl.org/registry/ - Дата доступу: 26.11.19

33. Веб-сайт OGRE 3D, OGRE 3D - графічний рушій із відкритим кодом [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.ogre3d.org/ – Дата доступу: 10.10.19

34. Формат файлів зображень високого динамічного діапазону OpenEXR [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.openexr.com/ – Дата доступу: 29.10.19

35. Веб-сайт відкритої графічної мови OpenGL [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL Візуалізація високоякісної графіки, http://www.opengl.org/ – Дата доступу: 25.10.19

36. SGI, Open Inventor – об'єктно-орієнтований 3D інструментарій [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://oss.sgi.com/projects/inventor/ – Дата доступу: 25.10.19

37. Джон О'Рорке, «Інтеграція шейдерів у прикладних програмах», 2004 – 587 с.

38. OpenSG – Основний веб-сайт [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://opensg.vrsource.org/ – Дата доступу: 23.11.19

39. OpenSceneGraph – Основний веб-сайт [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.openscenegraph.org/ – Дата доступу: 24.11.19

40. Еріх Гамма, Річард Хелм, Ральф Джонсон та Джон Вліссайдс, «Шаблони дизайну – елементи багаторазового об'єктно-орієнтованого програмного забезпечення», 1995 – 869 с.

41. Метт Фарр, «Вступ до шейдерних інтерфейсів», 2004 – 550 с.

42. Фонд програмного забезпечення Python, Мова програмування Python [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.python.org/ – Дата доступу: 25.11.19

43. Грег Уорд, Сяйво - Синтетична система візуалізації [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://radsite.lbl.gov/radiance / – Дата доступу: 23.11.19

44. Дієго Нехаб, RPly – «Бібліотека ANSI C для введення та виведення формату файлів», 2008 – 856 с.

45. Томаш Акенін-Моллер та Ерік Хейнс, «Візуалізація в режимі реального часу», 2002 – 458 с.

46. Ханан Самет «Проектування та аналіз структур просторових даних», 1989 – 745 с.

47. Крістоф Шлік «Пікселі високого динамічного діапазону», 1994 – 430 с.

48. Дональд П. Грінберг «Фізичні ефекти відблисків для цифрових зображень», 1995 – 589 с.

49. Джек Тумблін та Холлі Е. Рашмейер, «Відтворення тонів для реалістичних зображень», 1993 – 758 с.

50. Т. Уїтт і Д. М. Ваймер «Тестування програмного забезпечення для розробки тривимірної растрової графічної системи», 1981 – 852 с.

51. Ленс Вільямс, «Кидання вигнутих тіней на вигнуті поверхні», 1978 – 298 с.

52. Операційна система Microsoft Windows [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.microsoft.com/windows/default.mspx - Дата доступу: 25.11.19

53. Майкл Віммер, Маркус Гігл та Дітер Шмальстиег, «Швидкі покрокові інструкції з кешованими зображенням та рейєром», 2005 – 875 с.

54. wxWidgets – Кросплатформна бібліотека графічного інтерфейсу [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://www.wxwidgets.org/ – Дата доступу: 27.11.19

55. Кирило Зеллер, «Практичне моделювання тканини на сучасних графічних процесорах», 2006 – 425 с.

56. Марк Адлер, zlib – Бібліотека стиснення зображення без втрат якості [Електронний ресурс] – 2019 – Режим доступу до ресурсу: URL http://zlib.net/ – Дата доступу: 29.11.19

57. Передельский Л. «Экология», 2014 – 608 с.

58. Фюкс Р. «Зеленая революция», 2016 – 336 с.

59. Вернадский В.И. «Биосфера и неосфера», 1989 – 687 с.

60. Голубець М.А. «Від біосфери до соціосфери», 1987 – 587 с.

61. Доценко И. «Химическая промышленность и охрана окружающей среды», 1986 – 875.

62. Хижняк М.І., Нагорна А.М. «Здоров’я людини та екологія», 1995 – 987 с.

63. МНЭПУ «Экология, охрана природы и экологическая безопасность», 1997 – 858 с.

64. Антоново В. П. «Уроки Чернобыля: радиация, жизнь, здоровье», 1989 – 458 с.

65. Білявський Г.О. «Основи загальної екології», 1995. – 368 с.

66. К.: Юрінком «Кодекс законів про працю України», 1998 – 1040 с.

67. Атаманчук П. «Охорона праці в галузі», 2017 – 322 с.

68. Антонюк А.«Гігієнічна класифікація праці за показниками шкідливості і небезпечності факторів виробничого середовища, важкості та напруженості трудового процесу», 1998 – 875 с..

69. Реєстр ДНАОП «Державний реєстр міжгалузевих і галузевих нормативних актів про охорону праці», 1995 – 223 с.

70. ДСанПІН3.3.2.007-98, «Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно – обчислювальних машин», 2015 – 789 с.

71. Свод стандартов «Система стандартов безопасности труда»,1989 – 874 с.

72. Вігренко І. С «Загальна та медична психологія», 1994 – 458 с.

73. Адміністрація Президента України «Про концепцію захисту населення і територій у разі загрози та виникнення надзвичайних ситуацій: Указ Президента України № 284/99 від 26.03.1999 р.», 1999 – 159 с..

74. Секретаріат Верховної Ради України «Про захист населення і територій від надзвичайних ситуацій техногенного та природного характеру: Закон України № 1809-ПІ від 08.06.2000 р.», 2000 – 759 с.

75. Шубин Е.П. «Гражданская оборона», 1991 – 458 с.

76. Гетьман В.М. «Перша долікарська допомога в екстремальних ситуаціях», 1995 – 458 с.

77. Желібо Е.П., Заверуха Н.М., Зацарний В.В. «Безпека життєдіяльності», 2001 – 199 с.

78. НПАОП 40.1-1.21-98. «Правила безпечної експлуатації електроустановок споживачів», 1998 – 186 с.

ДОДАТКИ

Додаток А

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Національна академія наук України
Університет імені П'єра і Марії Кюрі (Франція)
Маріборский університет (Словения)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Шяуляйська державна колегія (Литва)
Жешувський політехнічний університет ім. Лукасевича (Польща)
Білоруський національний технічний університет (Республіка Білорусь)
Міжнародний університет цивільної авіації (Марокко)
Національний університет біоресурсів і природокористування України (Україна)
Наукове товариство ім. Шевченка
ГО «Асоціація випускників Тернопільського національного технічного університету імені Івана Пулюя»

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

Том ІІ

VIII Міжнародної науково-технічної конференції молодих учених та студентів

27-28 листопада 2019 року

УКРАЇНА
ТЕРНОПІЛЬ – 2019

УДК:004.9
І.Г. П'ятківський, І.С. Ячменьов
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ВИКОРИСТАННЯ ЗАСОБІВ GOOGLE MAPS ДЛЯ ВІЗУАЛІЗАЦІЇ ДАНИХ

I.H.Piatkivskyi, I.S. Yachmenov

USING GOOGLE MAPS TO VISUALIZE DATA

При візуалізації даних перед користувачами постають наступні завдання:

- скорочення часу візуалізації даних;
- спрощення процесу візуалізації даних;
- мінімізація використання системних ресурсів.

При візуалізації даних за допомогою Google Maps, використовуючи безплатну підписку, у студентів є ряд певних обмежень:

- карта може мати лише максимум 10 шарів;
- при малюванні ліній або форм можливо максимум 10 000 ліній, форм або місць;
- максимум 50 000 розташувань на карті(включаючи лінії та форми);
- максимум 20 000 комірок даних у таблиці.

Візуалізацію даних на Google maps можна зробити двома шляхами:

Перший це вносити дані вручну вказуючи координати місця та опис. Для цього потрібно зареєструвати gmail аккаунт, після чого відкрити google maps. Наступним кроком потрібно відкрити меню карт яке знаходиться в лівому верхнього куті екрану зображене 3 горизонтальними лініями і вибрати пункт «Мої місця» після чого на потрібно буде пройти авторизацію. Далі ми вже буде перенаправлені в розділ «Мох місця» та можем вибрати підрозділ «Карти» та натиснути «Створити карту» [1]. Тепер нам доступний весь фукціонал для візуалізації наших даних.

Другий це імпорт даних формату csv, xlsx, kml, gpx, при імпорті потрібно зарання в програмі Excel або її аналогах вказати широту та довготу місця для якого робиться візуалізації, та інші додаткові дані які вважаються за потрібне.

При візуалізації даних з файлу, наші дію аналогічні тільки замість того нам потрібно заповнити таблицю даними, але враховуючи вище описані обмеження. При візуалізації даних з файлу потрібно задати широту та довготу в окремих колонках.

Дана візуалізація даних буде мати наступні властивості: швидке впровадження змін, динамічна інфраструктура [2].

Варто зазначити, що візуалізацію даних можна робити в незалежності від операційної системи ПК, головне це наявність інтернету та веб браузер, при необхідності можливо використовувати інший електронний пристрій.

Література

1. Створення власної карти. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://www.maptive.com/create-a-custom-google-map/ [Дата доступу: 09.04.19]

2. Консолідація інформаційних ресурсів бібліотек, архівів, музеїв: інформаційний соціокомунікаційних проект класу «Розумне місто» / Кунанець Н.Е., Кунанець О.О., Мацюк О.В., Липак Г.І. // Управління проектами: стан та перспективи: метеріали XII міжнародної науково-практичної конференції, 13-16 вересня 2016, Миколаїв.- Миколаїв, 2016.-С.82-84.

Додаток Б

УДК:004.9
О.М. Яковенко, О.І. Забігайло, І.С. Ячменьов
Тернопільський національний технічний університет імені Івана Пулюя, Україна

СТАНДАРТИ РОЗУМНОГО МІСТА

O.M. Yakovenko, O.I. Zabihailo, I.S. Yachmenov
STANDARTS OF SMART CITY

«Розумне місто» – концепція інтеграції декількох інформаційних і комунікаційних технологій (ІКТ) та інтернет речей (IoT рішення) для управління міським майном.

Концептуалізація «розумного міста» варіюється від міста до міста та країни до країни, залежно від рівня розвитку, готовності до змін і реформ, ресурсів та прагнень жителів міста. Для забезпечення прагнень і потреб громадян, містобудівники в ідеалі прагнуть розвивати всю міську екосистему, яка представлена чотирма стовпами всеохоплюючої інституційної, фізичної, соціальної та економічної інфраструктури. Це може бути довгостроковою метою, і міста можуть поступово розвивати таку всеосяжну інфраструктуру, додаючи шари «розумності». Основна увага в «розумному місті» приділяється сталому та інклюзивному розвитку, і ідея полягає в тому, щоб поглянути на компактні зони, створити модель, що буде відтворюватися.

Основні елементи інфраструктури в розумному місті включають:

- достатнє водопостачання;
- забезпечене постачання електроенергії;
- санітарії, включаючи поводження з твердими відходами;
- ефективна міська мобільність і громадський транспорт;
- доступне житло, особливо для бідних;
- надійне підключення до ІТ та його цифро вість;
- належне управління, особливо електронне урядування та участь громадян;
- стале середовище, здоров'я та освіта;
- безпеки та безпеки громадян, зокрема жінок, дітей та людей похилого віку;

Відповідно, мета «розумного міста» полягає в тому, щоб підвищити та покращити якість життя людей, сприяючи розвитку місцевих територій та використанню технологій, особливо технологій, які призводять до розумних результатів. Розвинені райони перетворять існуючі райони (модернізацію та реконструкцію), включаючи трущобу, на кращі планові, тим самим покращуючи життєздатність всього міста. Нові райони (greenfield) будуть розроблені навколо міст з метою пристосування до зростаючого населення у містах. Застосування розумних рішень дозволить містам використовувати технології, інформацію та дані для покращення інфраструктури та послуг. Комплексний розвиток у такий спосіб поліпшить якість життя, створить зайнятість і підвищить доходи для всіх, особливо бідних і знедолених, що призведе до інклюзивних міст.

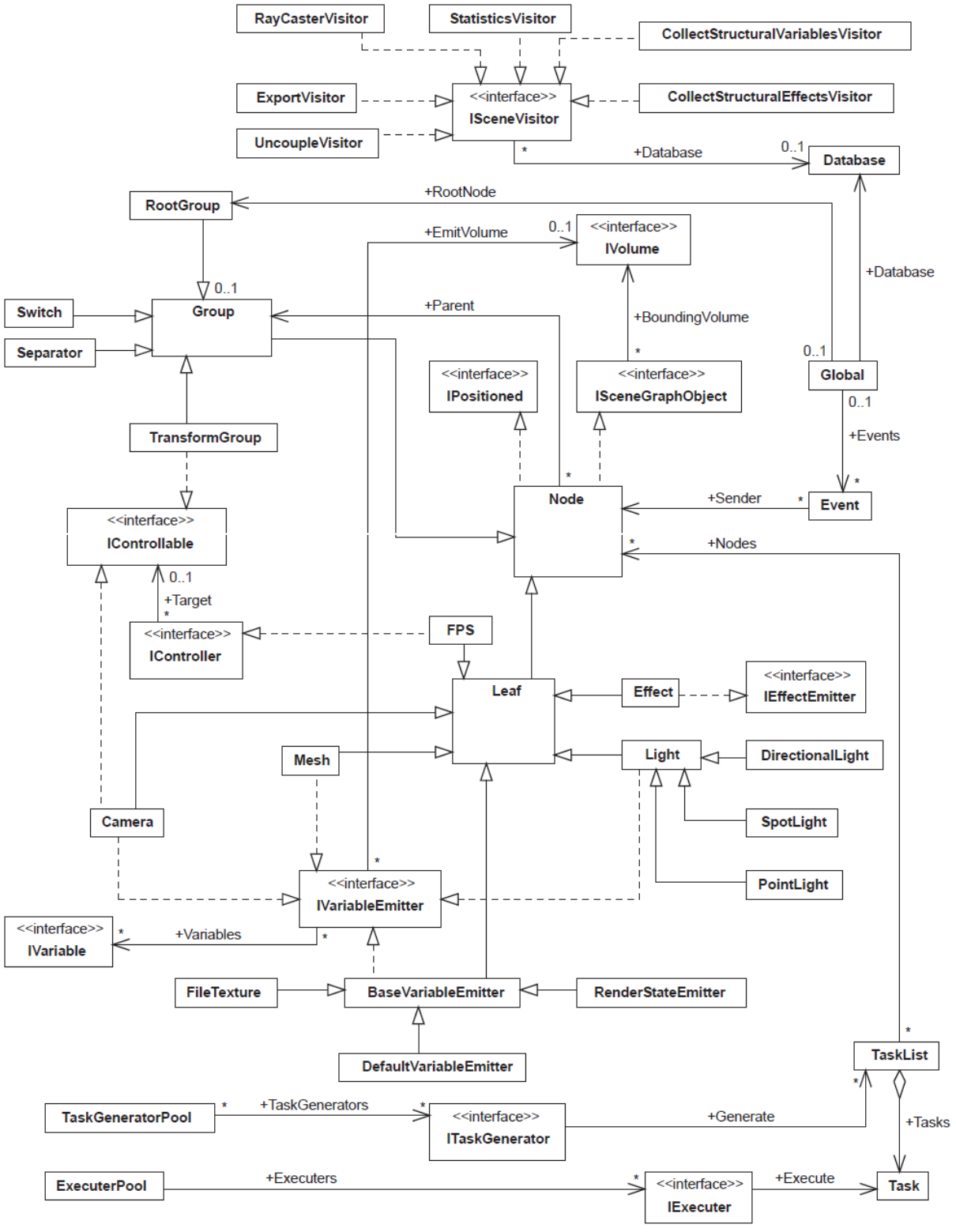
Література

1 Європейська комісія по стандартизації [Електронний ресурс] – Режим доступу: https://ec.europa.eu/eip/ageing/standards/city/smart-cities_en [Дата доступу: 21.05.19]

2. Що таке «розумне місто» [Електронний ресурс] – Режим доступу: https://internetofthingsagenda.techtarget.com/definition/smart-city [Дата доступу: 25.05.19]

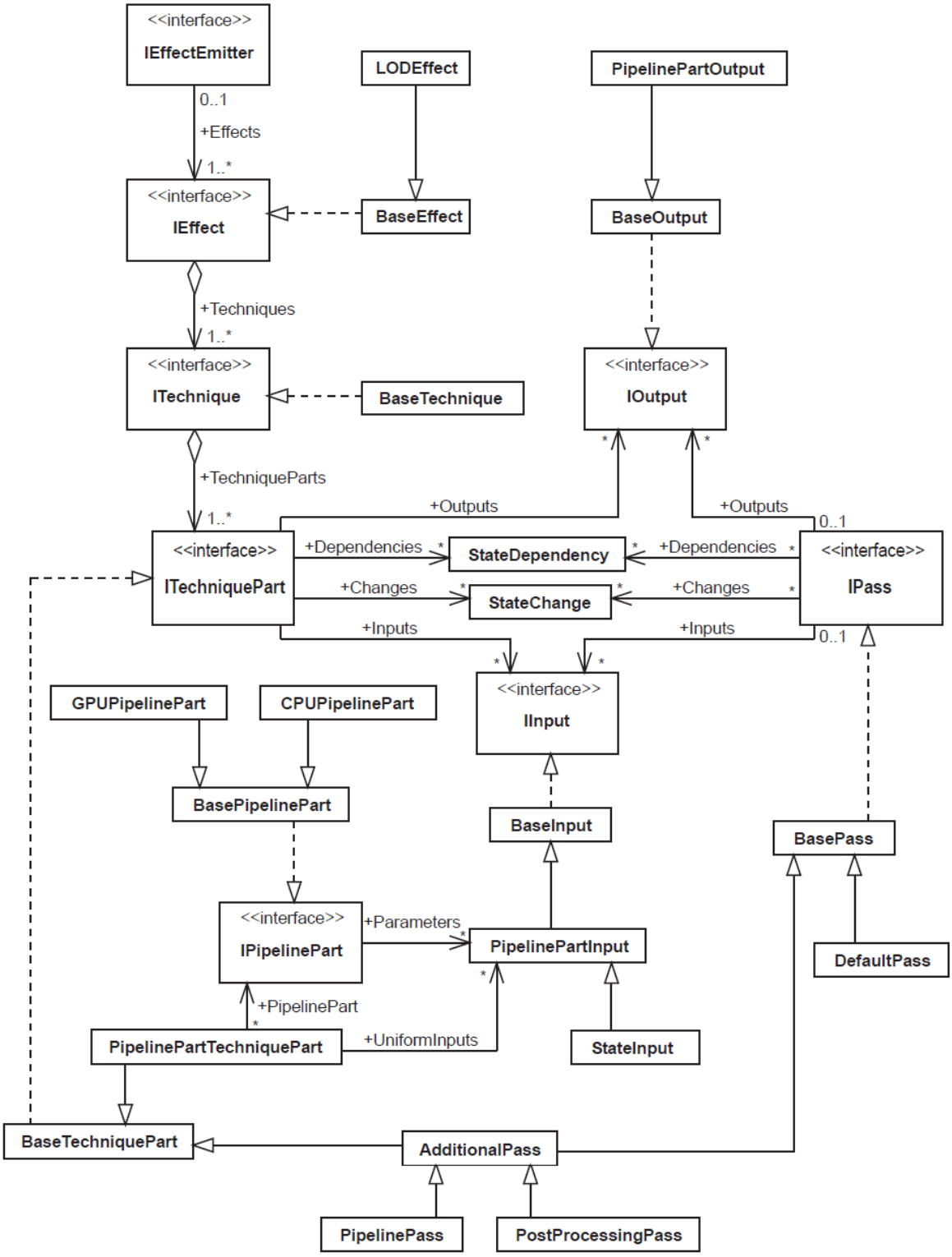
Додаток Д

UML діаграма графа сцени



Додаток Е

UML діаграма бібліотеки ефектів



Додаток Ж

Лістинг коду 2D алгоритму «Ray Casting»

```
#include "globals.h"
using namespace sf;
Vector2f g_mouse_pos = Vector2f(0, 0);
int main()
{
// Налаштування
const float    ray_density = 1000;
const Color    ray_color = Color(255, 255, 255, 10);
const Color    wall_color = Color::Red;
const Vector2f window_size(900, 600);
// Ініціалізація рандомізатора
srand(time(NULL));
// Створення вікна
RenderWindow window(VideoMode(window_size.x, window_size.y), "2D Ray
Casting");
// Створення стін
std::vector<wall> walls;
for (int i = 0; i < 4; i++)
{
// Рандомізація стартової і фінішної позиції
const Vector2f start(random(window_size.x), random(window_size.y));
const Vector2f end(random(window_size.x), random(window_size.y));
walls.push_back(wall(start, end));
}
// Створення променів
std::vector<ray> rays;
const float step = 1.f / ray_density;
for (float a = 0; a < TWO_PI; a += step)
{
const float x = cos(a);
const float y = sin(a);
rays.push_back(ray(x, y));
}
// Створення ліній які використовуються для візуалізації променів
VertexArray ray_line(Lines, 2);
ray_line[0].color = ray_color;
ray_line[1].color = ray_color;
// Створення ліній які використовуються для візуалізації стін
VertexArray wall_line(Lines, 2);
wall_line[0].color = wall_color;
wall_line[1].color = wall_color;
```

```
// Головний цикл
Vector2f mouse_snapshot;
while (window.isOpen())
{
Event event;
while (window.pollEvent(event))
{
if (event.type == Event::Closed)
{
window.close();
}
}
// Встановлення нових позицій для стін якщо клавіша натиснута
if (Keyboard::isKeyPressed(Keyboard::R) ||
Keyboard::isKeyPressed(Keyboard::Space))
{
for (int i = 0; i < walls.size(); i++)
{
walls[i].start = Vector2f(random(window_size.x),
random(window_size.y));
walls[i].end = Vector2f(random(window_size.x),
random(window_size.y));
}
sleep(seconds(0.2));
}
// Оновлення позиції миші
g_mouse_pos = Vector2f(Mouse::getPosition(window));
// Якщо миша на тому ж місці як і в попередньому кадрі переходимо до
наступного
if (mouse_snapshot == g_mouse_pos)
continue;
// Стартова позиція для пере-візуалізації ліній відповідно до
позиції миші
ray_line[0].position = g_mouse_pos;
window.clear();
// Обчислення перетину та візуалація ліній
for (int i = 0; i < rays.size(); i++)
{
// Встановлення кінцевої точки променя за замовчування.
rays[i].reset();
// Проходимося по кожній стіні і встановлюємо кінцеву точку перетину
```

```
// Коли перетин знайдено, кінцева точка встановлюється для цього
перетину, тобто наступна перевірка буде виконуватися для стіни
// між мишею та новою кінцевою точкою. Це означає, що промінь завжди
буде йти до найближчої стіни
for (int j = 0; j < walls.size(); j++)
{
// Обчислення кінцевої точки променя
rays[i].calc_hit(walls[j].start, walls[j].end);
}
// Встановлення кінцевої лінії візуалізації для кінцевого перетину
ray_line[1].position = rays[i].m_end;
// Візуалізація променя
window.draw(ray_line);
}
// Візуалізація стін
for (int i = 0; i < walls.size(); i++)
{
wall_line[0].position = walls[i].start;
wall_line[1].position = walls[i].end;
window.draw(wall_line);
}
// Оновлення вікна
window.display();
// Збереження поточної позиції миші
mouse_snapshot = g_mouse_pos;
}
return 0;
}
#pragma once
#include "globals.h"
using namespace sf;
class ray
{
public:
Vector2f m_end;
ray() {};
ray(float x, float y);
void calc_hit(Vector2f wall1, Vector2f wall2);
void reset();
private:
Vector2f m_relative_end;
};
#include "globals.h"
#include "ray.h"
ray::ray(float x, float y)
```

```
{// Встановлення кінцевої точки залежної від позиції міші
// Додавання довільної довжини
m_relative_end = Vector2f(x, y) * 3000.f;
}
// Скидання кінцевої точки променя
void ray::reset()
{
// Встановлення кінцевої точки на (довжину за замовчуванням)
відстань від миші в заданому напрямку
m_end = g_mouse_pos + m_relative_end;
}
// Обчислення точки перетину двої прямих
// Використовується для отримання перетину між променем і стіною
void ray::calc_hit(Vector2f p3, Vector2f p4)
{
const Vector2f p1 = g_mouse_pos;
const Vector2f p2 = m_end;
// Обчислює знаменник рівнянь
const float den = (p1.x - p2.x) * (p3.y - p4.y) - (p1.y - p2.y) *
(p3.x - p4.x);
if (den == 0)
return;
const float t = ((p1.x - p3.x) * (p3.y - p4.y) - (p1.y - p3.y) *
(p3.x - p4.x)) / den;
const float u = -((p1.x - p2.x) * (p1.y - p3.y) - (p1.y - p2.y) *
(p1.x - p3.x)) / den;
// Якщо є перетин
if (t > 0 && t < 1 && u > 0 && u < 1)
{ // Отримуємо точку перетину
m_end.x = p1.x + t * (p2.x - p1.x);
m_end.y = p1.y + t * (p2.y - p1.y); }}
#pragma once
#include "globals.h"
using namespace sf;
class wall
{ public:
Vector2f start;
Vector2f end;
wall() {};
wall(Vector2f pos1, Vector2f pos2); };
#include "wall.h"
wall::wall(Vector2f pos1, Vector2f pos2)
{
start = pos1;
end = pos2;
}
```

Додаток И

Лістинг коду 3D алгоритму «Ray Casting»

```
#include <iostream>
#include <vector>
#include <utility>
#include <algorithm>
#include <chrono>
using namespace std;
#include <stdio.h>
#include <Windows.h>
int nScreenWidth = 120;// Розміри екрану по позиції X (колонки)
int nScreenHeight = 40;// Розміри екрану по позиції Y (рядки)
int nMapWidth = 16;// Розміри мапи
int nMapHeight = 16;
float fPlayerX = 14.7f;// Стартова позиція
float fPlayerY = 5.09f;
float fPlayerA = 0.0f;// Поворот камери на стартовій позиція
float fFOV = 3.14159f / 4.0f;// Встановлення поля зору
float fDepth = 16.0f;// Maximum rendering distance
float fSpeed = 5.0f;// Швидкість руху камери
int main() {
// СТворення буфера сцени
wchar_t *screen = new wchar_t[nScreenWidth*nScreenHeight];
HANDLE hConsole = CreateConsoleScreenBuffer(GENERIC_READ |
GENERIC_WRITE, 0, NULL, CONSOLE_TEXTMODE_BUFFER, NULL);
SetConsoleActiveScreenBuffer(hConsole);
DWORD dwBytesWritten = 0;
// Створення мати # = блоки сніти, . = простір
wstring map;
map += L"#########.......";
map += L"#...............";
map += L"#.......########";
map += L"#..............#";
map += L"#......##......#";
map += L"#......##......#";
map += L"#..............#";
map += L"###............#";
map += L"##.............#";
map += L"#......####..###";
map += L"#......#.......#";
map += L"#....#.........#";
map += L"#..............#";
map += L"#......#########";
map += L"#..............#";
map += L"################";
```

```
auto tp1 = chrono::system_clock::now();
auto tp2 = chrono::system_clock::now();
while (1) {
// We'll need time differential per frame to calculate modification
// to movement speeds, to ensure consistant movement, as ray-tracing
// is non-deterministic
tp2 = chrono::system_clock::now();
chrono::duration<float> elapsedTime = tp2 - tp1;
tp1 = tp2;
float fElapsedTime = elapsedTime.count();
// Обробка повороту
if (GetAsyncKeyState((unsigned short)'A') & 0x8000)
fPlayerA -= (fSpeed * 0.75f) * fElapsedTime;
// Обробка поворону
if (GetAsyncKeyState((unsigned short)'D') & 0x8000)
fPlayerA += (fSpeed * 0.75f) * fElapsedTime;
// Обробка руху вперед та колізії
if (GetAsyncKeyState((unsigned short)'W') & 0x8000){
fPlayerX += sinf(fPlayerA) * fSpeed * fElapsedTime;;
fPlayerY += cosf(fPlayerA) * fSpeed * fElapsedTime;;
if (map.c_str()[(int)fPlayerX * nMapWidth + (int)fPlayerY] == '#'){
fPlayerX -= sinf(fPlayerA) * fSpeed * fElapsedTime;;
fPlayerY -= cosf(fPlayerA) * fSpeed * fElapsedTime;; }}
// Обробка руху назад та колізії
if (GetAsyncKeyState((unsigned short)'S') & 0x8000) {
fPlayerX -= sinf(fPlayerA) * fSpeed * fElapsedTime;;
fPlayerY -= cosf(fPlayerA) * fSpeed * fElapsedTime;;
if (map.c_str()[(int)fPlayerX * nMapWidth + (int)fPlayerY] == '#'){
fPlayerX += sinf(fPlayerA) * fSpeed * fElapsedTime;;
fPlayerY += cosf(fPlayerA) * fSpeed * fElapsedTime;; }}
for (int x = 0; x < nScreenWidth; x++){
// Для кожного стовпчика обчислення прогнозованого куту кидання
променя у світовий простір
float fRayAngle = (fPlayerA - fFOV / 2.0f) + ((float)x /
(float)nScreenWidth) * fFOV;
// Пошук дистанції до стіни
float fStepSize = 0.1f;
float fDistanceToWall = 0.0f;
bool bHitWall = false;
bool bBoundary = false;
float fEyeX = sinf(fRayAngle);
float fEyeY = cosf(fRayAngle);
// Поступове кидання променя від позиції камери, по куту променя,
// тестуючи на перетин з блоком
while (!bHitWall && fDistanceToWall < fDepth){
```

```
fDistanceToWall += fStepSize;
int nTestX = (int)(fPlayerX + fEyeX * fDistanceToWall);
int nTestY = (int)(fPlayerY + fEyeY * fDistanceToWall);

// Перевірка, на вихід променя за дистанцію візуалізації
if (nTestX < 0 || nTestX >= nMapWidth || nTestY < 0 || nTestY >=
nMapHeight){
bHitWall = true;
fDistanceToWall = fDepth; }
else {
if (map.c_str()[nTestX * nMapWidth + nTestY] == '#') {
bHitWall = true;
// Щоб виділити межі плитки, від кожної позиції камери до кожної
перешкоди.
// Чим меньша дистанція тим ближче ми будем до об'єкта.
vector<pair<float, float>> p;
for (int tx = 0; tx < 2; tx++)
for (int ty = 0; ty < 2; ty++) {
float vy = (float)nTestY + ty - fPlayerY;
float vx = (float)nTestX + tx - fPlayerX;
float d = sqrt(vx*vx + vy * vy);
float dot = (fEyeX * vx / d) + (fEyeY * vy / d);
p.push_back(make_pair(d, dot)); }
sort(p.begin(), p.end(), [](const pair<float, float> &left, const
pair<float, float> &right) {return left.first < right.first; });
float fBound = 0.01;
if (acos(p.at(0).second) < fBound) bBoundary = true;
if (acos(p.at(1).second) < fBound) bBoundary = true;
if (acos(p.at(2).second) < fBound) bBoundary = true; }}}
// Обчислення позиції від стелі до підлоги
int nCeiling = (float)(nScreenHeight / 2.0) - nScreenHeight /
((float)fDistanceToWall);
int nFloor = nScreenHeight - nCeiling;
// Шейдерні стіни на основі відстані
short nShade = ' ';
if (fDistanceToWall <= fDepth / 4.0f)nShade = 0x2588;// Дуже близько
else if (fDistanceToWall < fDepth / 3.0f)nShade = 0x2593;
else if (fDistanceToWall < fDepth / 2.0f)nShade = 0x2592;
else if (fDistanceToWall < fDepth)nShade = 0x2591;
elsenShade = ' ';// Дуже далеко
if (bBoundary)nShade = ' ';
for (int y = 0; y < nScreenHeight; y++) {
if (y <= nCeiling)
screen[y*nScreenWidth + x] = ' ';
else if (y > nCeiling && y <= nFloor)
screen[y*nScreenWidth + x] = nShade;
```

```
else {
float b = 1.0f - (((float)y - nScreenHeight / 2.0f) /
((float)nScreenHeight / 2.0f));
if (b < 0.25)nShade = '#';
else if (b < 0.5)nShade = 'x';
else if (b < 0.75)nShade = '.';
else if (b < 0.9)nShade = '-';
elsenShade = ' ';
screen[y*nScreenWidth + x] = nShade; } } }
// Статистика яка буде відображатися
swprintf_s(screen, 40, L"X=%3.2f, Y=%3.2f, A=%3.2f FPS=%3.2f ",
fPlayerX, fPlayerY, fPlayerA, 1.0f / fElapsedTime);
// Візуалізація мапи
for (int nx = 0; nx < nMapWidth; nx++)
for (int ny = 0; ny < nMapWidth; ny++) {
screen[(ny + 1)*nScreenWidth + nx] = map[ny * nMapWidth + nx]; }
screen[((int)fPlayerX + 1) * nScreenWidth + (int)fPlayerY] = 'P';
// Візуалізація к-сть візуалізованих кадрів у секунду
screen[nScreenWidth * nScreenHeight - 1] = '\0';
WriteConsoleOutputCharacter(hConsole, screen, nScreenWidth *
nScreenHeight, { 0,0 }, &dwBytesWritten); }
return 0; }
```

Додаток К

Лістинг коду «Інтерфейсу візуалізації»

```
class IDevice : public G_ENGINE::Core::Reflection {

public:
// Буфер кадрів
virtual ITargetPtr GetFramebuffer() = 0;
// @param initstruct Структура яка тримає в собі інформацію ро
ініціалізовані змінні
virtual void Initialize( G_ENGINE::Core::Engine::WindowPtr window,
DeviceInit initstruct) = 0;
// Демонстрація к-сті кадрів на екран
virtual void Present() = 0;
// Створення буфера геометрії
virtual IGeometryPtr CreateGeometry( bool target ) = 0;
// Створення прикладу текстур
virtual ISamplerPtr CreateSampler() = 0;
class ITarget : public G_ENGINE::Core::Reflection {
public:
// Розмір об'єкта відображення
virtual Vec2i GetSize() = 0;
// Надсилає точку, об'єкта відображення
virtual IViewportPtr GetViewport() = 0;
virtual const G_ENGINE::Core::Engine::Identifier &GetId() = 0; };
// Інтерфейс управління об'єктами відображення
class ITargetManager : public G_ENGINE::Core::Reflection{
public:
// Встановлює активну ціль відтворення для заданого індексу буфера.
// @param bufferIndex Індекс буфера для прив’язування цілі
відображення.
// @param target Новий об'єкт відображення
virtual void SetTarget(const uint8 &bufferIndex, ITargetPtr target)
= 0;
// Отримуємо активні координати об'єкта відображення
// @param bufferIndex Індекс буфера для прив’язування цілі
відображення.
virtual ITargetPtr GetTarget(const uint8 &bufferIndex) = 0;
/** Інтерфейс для геометричних буферів візуалізації. */
class IGeometry : public G_ENGINE::Core::Engine::Actor {
public:
// Додає задану змінну в буфер геометрії.
virtual void Add( const Variables::IVariablePtr &variable ) = 0;
// Додає всі змінні заданого набору змінних до буфера геометрії.
virtual void Add( const Variables::VariableSetPtr &variables ) = 0;
/// Додає всі змінні даного списку змінних до буфера геометрії.
virtual void Add( const Variables::IVariableList &variables ) = 0;
```

```
// Повертає список змінних.
virtual const Variables::IVariableList &GetVariables() const = 0;
// Малює всі примітиви з цього буфера геометрії.
virtual void Draw() = 0;
// Сортування полігонів за заданими параметрами.
virtual void Sort(const SortOrder &order, const Vec3f &location ) =
0;
/// Оптимізує буфер геометрії відповідно до заданого режиму.
/// This method can be called once for every mode.
/// @param flags Мітка яка вказує як оптимізувати буфер геометрії.
virtual void Optimize(const OptimizeFlag &flag) = 0; };
```

Додаток Л

Лістинг коду «Бібліотеки ефектів»

```
class G_ENGINE_GRAPHICS_API IEffect: public ENGINE::Core::Engine {
public:
// Повертає ідентифікатор ефекта
virtual const Engine::Core::Engine::Identifier &GetIdentifier() = 0;
// Повертає список всіх технік для цього об'єкта
virtual const ITechniqueList &GetTechniques() = 0;
virtual ITechniquePtr GetTechnique(const float effectLOD) = 0;
// Встановлює функцію, яка викликається щоразу, коли ефект
змінюється
virtual void SetOnChange(const OnChangeCallback &callback) = 0;
// Вказує, чи змінює цей ефект обмежуючий об'єм об'єкта
virtual bool ChangesBoundingVolume() const = 0;
/// Повертає змінений  об'єм об'єкта.
virtual Engine::Core::Math::IVolumePtr ChangeBoundingVolume(
const Engine::Core::Math::IVolumePtr &volume) = 0; };
class Engine_GRAPHICS_API ITechnique: public
Engine::Core::Reflection {
public:
// Повертає ідентифікатор техніки
virtual const Engine::Core::Engine::Identifier &GetIdentifier() = 0;
// Повертає список частин з яких складається техніка
virtual const ITechniquePartList &GetParts() = 0;
// Повертає діапазон LOD для техніки
virtual void GetLODRange(float &minLOD, float &maxLOD) const = 0;
// Встановлює діапазон LOD для техніки
virtual void SetLOD(const float &minLOD, const float &maxLOD) = 0;
// Перевіряє, чи підтримується ця техніка на поточному обладнання.
virtual bool IsSupported() = 0;
virtual const IInputList &GetInputs() = 0;
virtual IInputPtr GetInput(const std::string &name) = 0;
virtual bool SetInputValue(const std::string &name,
const boost::any &value) = 0; };

class Engine_GRAPHICS_API ITechniquePart:
public Engine::Core::Reflection {
public:
// нумерація комбінованих ефектів
enum Filter{
TPF_DEPTH_CHANGING = 1
};
// Повертає ідентифікатор технік
virtual const Engine::Core::Engine::Identifier &GetIdentifier() = 0;
// Повертає список вхідних даних для цієї частини техніки.
virtual const IInputList &GetInputs() = 0;
```

```
// Повертає список вихідних даних для цієї частини техніки.
virtual const IOutputList &GetOutputs() = 0;
// Повертає індекс групи цієї частини техніки.
// Групові індекси можна використовувати для розміщення різних
деталей техніки
// в різні проходи візуалізації.
virtual const uint32 &GetGroupIndex() const = 0;
virtual const StateDependencyList &GetStateDependencies() = 0;
virtual const StateChangeList &GetStateChanges() = 0;
//Перевірка на іквевалентність техніки
virtual bool IsEqual(const ITechniquePartPtr &part) = 0;
virtual ITechniquePartPtr GetFilteredVersion(const Filter &filter) =
0;
virtual const ITechniquePtr &GetTechnique() const = 0; };

class Engine_GRAPHICS_API IPass: public Engine::Core::Reflection {
public:
// Повертає ідентифікатор проходу
virtual const Engine::Core::Engine::Identifier &GetIdentifier() = 0;
// Повертає список входів для цього проходу.
// Сюди також входитимуть вкладені деталі техніки.
virtual const IInputList &GetInputs() = 0;
// Повертає список результатів цього проходу.
virtual const IOutputList &GetOutputs() = 0;
// Повертає список залежностей від стану цього проходу.
virtual const StateDependencyList &GetStateDependencies() = 0;
// Повертає список змін, які буде виконано цим пропуском
візуалізації.
virtual const StateChangeList &GetStateChanges() = 0;
virtual void Render( const std::vector<
std::vector<Engine::Graphics::Variables::IVariable*> > &variables,
const std::vector<Engine::Graphics::Rendering::IGeometry*>
&geometry) = 0;
// Не виконує прохід якщо цей прохід вже виконувався.
virtual bool IsEqual(const IPassPtr &pass) = 0;
// Створює копію цього проходу.
virtual IPassPtr Clone() = 0; };
```

Додаток М

Лістинг коду «Графу малюнку»

```
class G_ENGINE_GRAPHICS_API IDrawGraph: public
Engine::Core::Reflection
{
public:
// Додає об'єкт у граф малюнок. Якщо об'єкет вже містить, він не
буде доданий двічі.
virtual void AddRenderable(const RenderablePtr &renderable) = 0;
// Видаляє об'єект із  графу малюнка.
virtual void RemoveRenderable(const RenderablePtr &renderable) = 0;
// Передає графу малюнку, що об'єкт уже у нььому
virtual void RenderableChanged(const RenderablePtr &renderable) = 0;
// Передає список видимих об'єктів до вилучення
virtual void GetRenderables(const std::vector<ICullerPtr> &cullers,
std::vector<RenderablePtr> &renderables) = 0;
// Повертає геометрію про об'єкт у графу малюнку
virtual Engine::Graphics::Rendering::IGeometryPtr GetDebugGeometry()
= 0;
// Повертає дані які використовуються для обчислення геометрії
об'єкта
virtual Engine::Graphics::Effect::IPassPtr GetDebugPass() = 0;
// Повертає список змінних які використовуються для візуалізації
об'єкта
virtual Engine::Graphics::Variables::IVariableList
GetDebugVariables() = 0;
virtual void Optimize() = 0;
virtual const Box3f &GetSceneBoundingBox() = 0;
};
```

Додаток Н

Порівняння швидкості алгоритмів за допомогою «Intel Graphics Monitor»

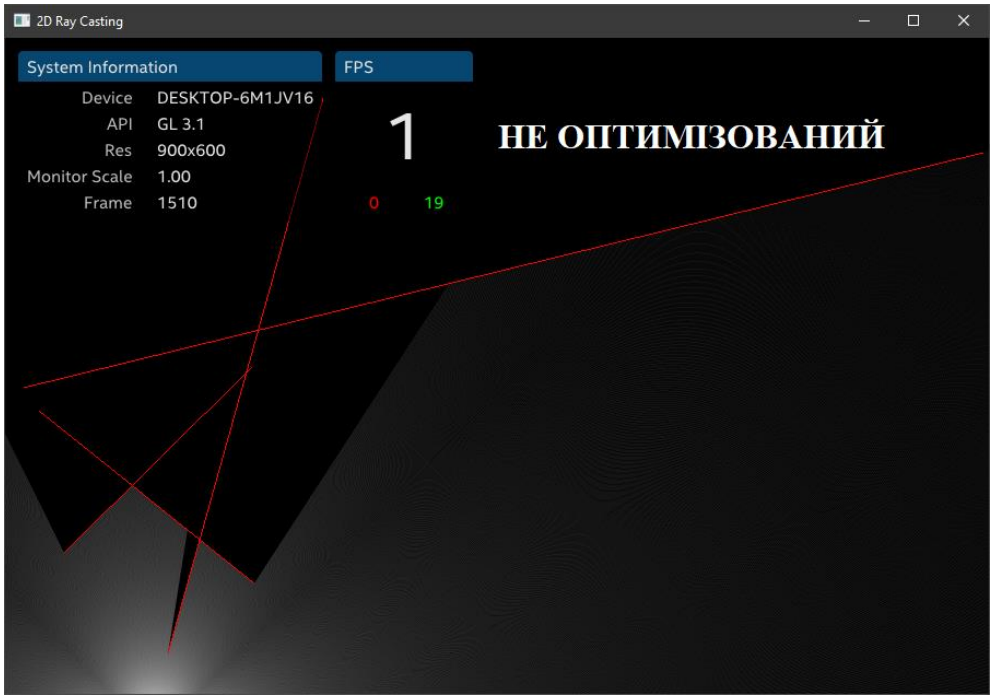


Рисунок Н.1 – Робота не оптимізованого алгоритму

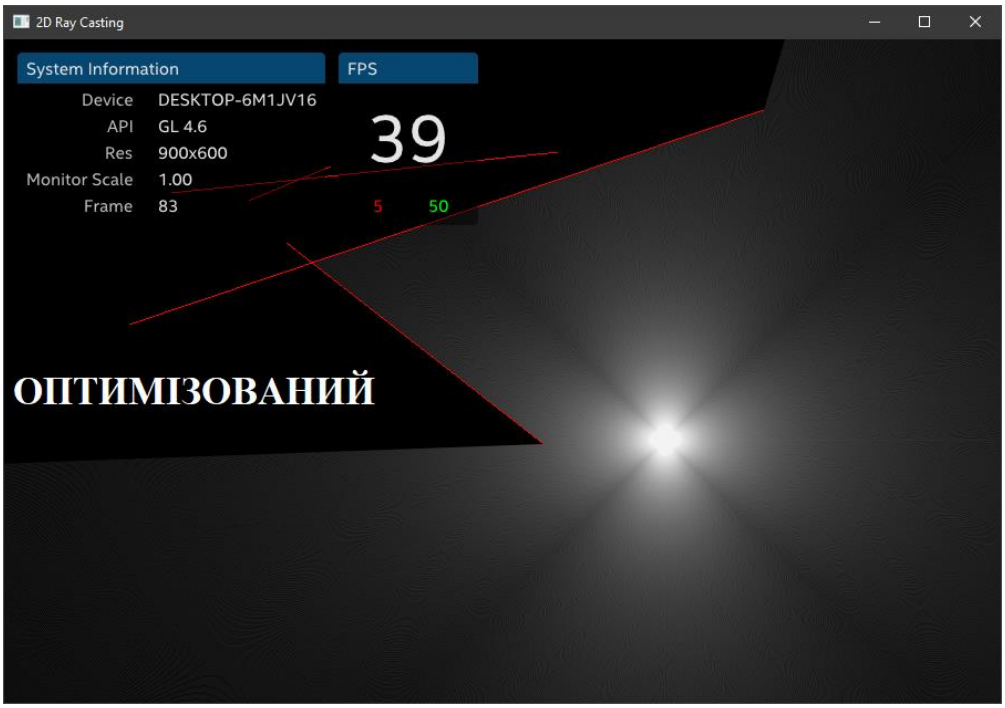


Рисунок Н.2 – Робота оптимізованого алгоритму