МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітньо-професійна програма: «Комп'ютерна графіка і Web-дизайн»

Група: 4КГ-08

Дипломний проект

здобувача освіти денної форми навчання
КГ.08.17.000.ДП

ПОЛТАВСЬКОГО
КИРИЛА АРТЕМОВИЧА

м. Одеса
2025 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: «**Комп'ютерна графіка і Web-дизайн»**

Група: **4КГ-08**

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проекту на тему:

Розробка застосунку для редагування тривимірних графічних об'єктів

Проектний матеріал складається з пояснювальної записки на 89 сторінках та графічного (презентаційного) матеріалу на 16 аркушах (слайдах)

Дипломник _____ (Полтавський К.А.)

Керівник _____ (Закроєв Ю.М.)

Консультанти:

з економічного розділу _____ (Канський М.Ю.)

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Голова циклової комісії _____ (Кривченко Ю.В.)

Завідувач відділення _____ (Краснокутська К.Г.)

Захист «23» червня 2025 р. Протокол ЕК № 3

Оцінка ЕК 4 (добре) / 80 б.

Секретар ЕК _____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *комп'ютерних систем* Комісія *КТ та ПІ*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна графіка і Web-дизайн»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"19" 05 2025 р.

ЗАВДАННЯ

на дипломний проект

Здобувачеві освіти *Полтавському Кирилу Артемовичу*

(прізвище, ім'я, по батькові)

1. Тема проекту *Розробка застосунку для редагування тривимірних графічних об'єктів*

затверджена наказом по коледжу від "14" 11 2024 р. № 246

2. Термін здачі закінченого проекту 16.06.25.

3. Вихідні данні до проекту *Передбачити завантаження тривимірних графічних об'єктів у форматі *.off; реалізувати опції редагування графічних об'єктів, зокрема функції зміни кольору, масштабування, обертання, реалізації віддзеркалення поверхні, побудови моделі Кука-Торренса, ієрархії обмежуючих об'ємів, освітлення та тіні на поверхні, налаштування шорсткості, згладжування та прозорості тривимірного графічного об'єкту; перебачити збереження файлів графічних об'єктів у форматі *.off*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які необхідно розробити)

Аналітичний огляд програмних засобів генерування тривимірної графіки; Аналіз принципів побудови тривимірних моделей; Огляд основних алгоритмів тривимірного моделювання; Вибір і обґрунтування інструментальних засобів розробки та компонентів застосунку; Розробка архітектури застосунку та ОО-моделі; Програмування і опис класів застосунку; Тестування розробленого застосунку; Економічні розрахунки; Заходи ТБ

5. Перелік графічного (презентаційного) матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів)

Порівняння методів рендерингу, згладжування поверхонь, процедурних алгоритмів для різних типів об'єктів, видів сплайнів, Структурна схема основних компонентів застосунку, Діаграма класів застосунку, Блок-схема алгоритму побудови ієрархії обмежуючих об'ємів, Головне вікно застосунку, Вікно застосунку з завантаженою тривимірною моделлю, Режими обертання та віддзеркалення для моделі, Режим побудови ієрархії обмежуючих об'ємів, Режим освітлення та тіні для моделі, Змінення кольору моделі та активація режиму Кука-Торренса

6. Консультанти по проекту, із зазначенням розділів проекту, що їх стосується

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Основний розділ	*Закроєв Ю.М.*		
Економічний розділ	*Канський М.Ю.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 6.05.25

Керівник *Закроєв Ю.М.* _____
(підпис)

Завдання прийняв до виконання *Полтавський К.А.* _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/р	Назва етапів дипломного проекту	Термін виконання етапів дипломного проекту (роботи)	Відмітка про виконання
1	*Вступ. Постановка мети та задач проектування*	14.05.25	виконано
2	*Аналітичний огляд програмних засобів генерування*	14.05.25 –	виконано
	тривимірної графіки	– 15.05.25	виконано
3	*Аналіз принципів побудови тривимірних моделей*	15.05.25	виконано
4	*Вибір та аналіз засобів розробки*	16.05.25	виконано
5	*Огляд основних алгоритмів 3D- моделювання*	17.05.25 –	виконано
6	*Обґрунтування інструментальних засобів розробки*	– 19.05.25	виконано
	та компонентів застосунку		
7	*Розробка архітектури застосунку та ОО-моделі*	23.05.25	виконано
8	*Програмування і опис класів застосунку*	25.05-25	виконано
9	*Аналіз отриманих результатів роботи застосунку*		
	для редагування тривимірних графічних об'єктів	28.05.25	виконано
10	*Економічні розрахунки та питання з охорони праці*	5.06.25	виконано
11	*Підготовка графічної частини проекту*	7.06.25	виконано
12	*Підготовка проекту до захисту та тестування ПЗ*	10.06.25	виконано

Дипломник _____
(підпис)

Керівник _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			КГ 08. 17 000. 00 ДП	Дипломний проект		
А4			КГ 08. 17 000. 00 ДП ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	КГ 08. 17 000. 00 ДП		
Розробив		Полтавський К.А.		16.06	Розробка застосунку для редагування тривимірних графічних об'єктів	Літ.	Аркуш 4 / Аркушів 89
Перевірив		Закроєв Ю.М.		16.06		Н Д П	
Н. контр.		Петрашова В.І.		16.06		ВСП "ОТФК ОНТУ" гр.4КГ-08	
Затверд.		Кривченко Ю.В.		16.06			

ЗМІСТ

ВСТУП

З розвитком сучасних технологій, зокрема комп'ютерних систем і графічного обладнання, тривимірне моделювання стало невід'ємною частиною багатьох галузей. Завдяки швидшій обробці даних та потужним алгоритмам, 3D-графіка отримала широке застосування не лише у спеціалізованих галузях, але й на рівні повсякденного використання, включаючи персональні комп'ютери.

Тривимірне моделювання використовується у найрізноманітніших галузях, таких як медицина, інженерія, архітектура, розважальні медіа та наукові дослідження. Наприклад, у медицині зображення, отримані за допомогою магнітно-резонансної томографії (МРТ) або комп'ютерної томографії (КТ), можуть бути об'єднані в детальні 3D-моделі органів людини. У сфері архітектури тривимірні моделі використовуються для створення віртуальних макетів будівель і ландшафтів, що дає можливість візуалізувати проєкти на різних стадіях їх розробки.

Інженерні галузі також активно використовують 3D-моделі для проєктування нових пристроїв, механізмів і транспортних засобів. Тривимірна графіка дозволяє створювати детальні прототипи, які можна випробувати на різних етапах розробки. Додатково, завдяки технології 3D-друку, ці моделі можуть бути фізично реалізовані, що сприяє подальшому аналізу та вдосконаленню конструкцій. У наукових дослідженнях тривимірні моделі використовуються для візуалізації складних об'єктів, таких як молекули або хімічні сполуки, що дозволяє вченим досліджувати їх з нових точок зору та проводити експерименти.

Проте, незважаючи на широке розповсюдження та значні переваги тривимірного моделювання, цей процес залишається складним для багатьох користувачів, особливо тих, хто тільки починає вивчати 3D-графіку. Багато професійних програм для 3D-моделювання мають складний інтерфейс і безліч функцій, які не завжди потрібні для швидкого редагування простих об'єктів.

Метою цього дипломного проєкту є розробка застосунку, що дозволить користувачам легко редагувати тривимірні об'єкти без потреби в складних і ресурсомістких інструментах.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналітичний огляд програмних засобів генерування тривимірної графіки

Генерування тривимірних графічних об’єктів стало невід'ємною частиною багатьох галузей — від індустрії розваг до науки та промисловості. Сучасне програмне забезпечення для 3D-графіки дозволяє створювати детальні візуалізації, моделі, анімації та прототипи, що використовуються в іграх, кіно, архітектурі, інженерії тощо. Розглянемо найбільш популярні та ефективні програмні засоби, що використовуються для генерування тривимірної графіки, їх основні можливості, переваги та обмеження.

1.1.1 Програмне забезпечення Blender

Blender є одним із найпопулярніших безкоштовних програм для створення тривимірних моделей. Це програмне забезпечення з відкритим кодом, що дозволяє виконувати широкий спектр завдань, включаючи 3D-моделювання, рендеринг, анімацію, скульптуру та навіть відеомонтаж.

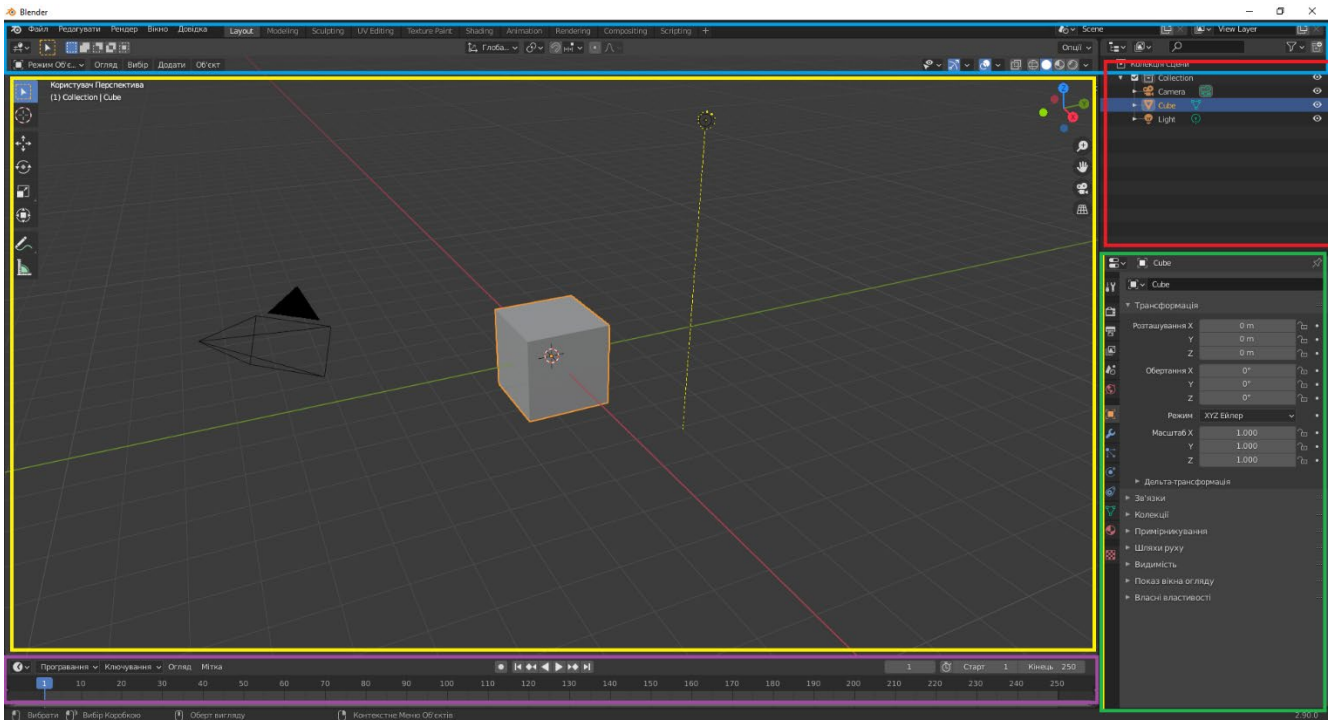


Рисунок 1.1. Інтерфейс програми Blender

Основні можливості Blender:

- Моделювання з використанням полігонів, кривих, текстур і матеріалів;

- Вбудований рушій рендерингу Cycles, який забезпечує високу якість фотореалістичних зображень;
- Підтримка фізичних симуляцій (тканини, рідини, дим, тверді тіла);
- Інструменти для створення анімацій та роботи з кістками (риггінг);
- Підтримка Python для написання скриптів, що дозволяє автоматизувати процеси.

Blender користується великою популярністю серед як професійних 3D-художників, так і початківців, завдяки великій кількості навчальних ресурсів та активній спільноті.

1.1.2 Програмне забезпечення Autodesk Maya

Autodesk Maya — це професійне програмне забезпечення для 3D-моделювання, анімації та візуалізації, що використовується в кіноіндустрії, анімаційних студіях та розробці відеоігор. Ця програма є одним із провідних інструментів для створення високоякісних тривимірних об'єктів і спецефектів.

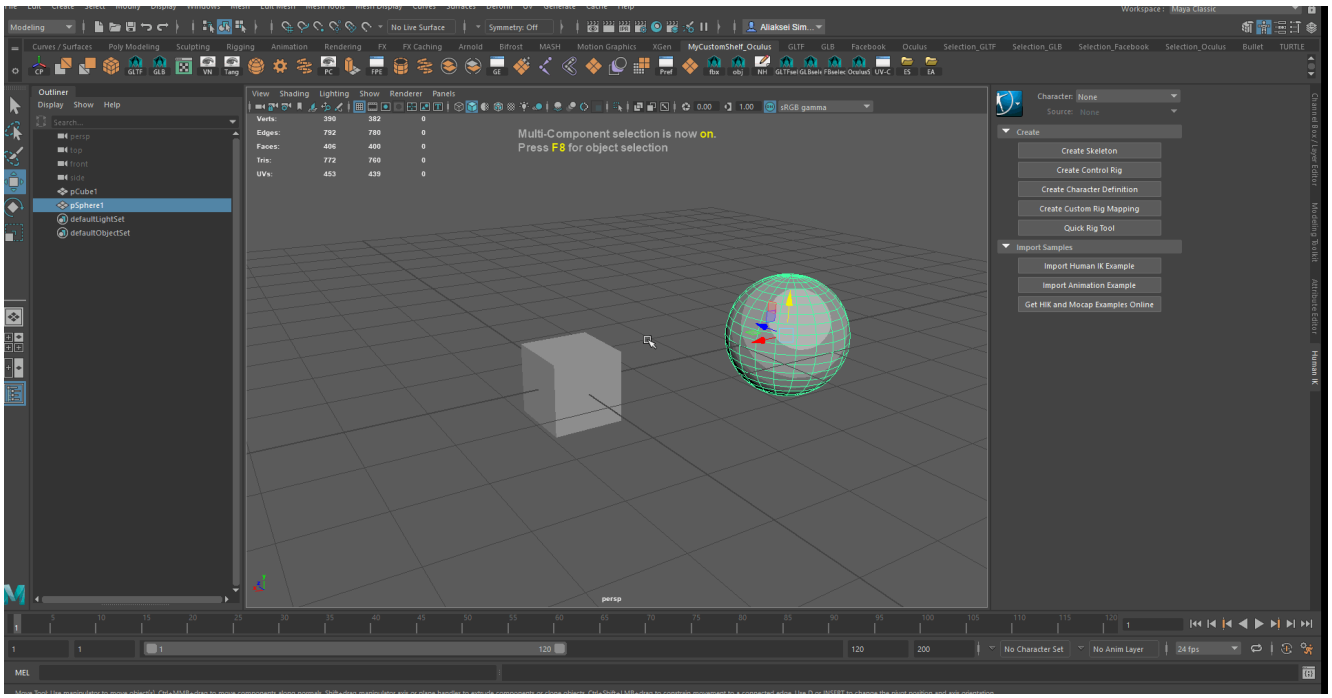


Рисунок 1.2. Інтерфейс програми Autodesk Maya

Основні можливості Autodesk Maya:

- Підтримка полігонального та NURBS-моделювання;
- Розвинені засоби для створення складних анімацій, включаючи інструменти для динаміки та симуляції частинок;

- Високоякісний рендеринг за допомогою рушіїв Arnold та Mental Ray;
- Інтеграція з іншими продуктами Autodesk, такими як 3ds Max та Revit;
- Підтримка великих робочих процесів у великих командах завдяки інструментам управління проєктами.

Autodesk Maya є одним із найпотужніших інструментів на ринку, однак складний інтерфейс і висока вартість ліцензії можуть стати перешкодою для новачків.

1.1.3 Програмне забезпечення 3ds Max

Autodesk 3ds Max є іншим популярним продуктом від Autodesk, орієнтованим на 3D-моделювання, анімацію та візуалізацію. Ця програма широко використовується в архітектурній візуалізації, дизайні інтер'єрів та у відеоіграх.

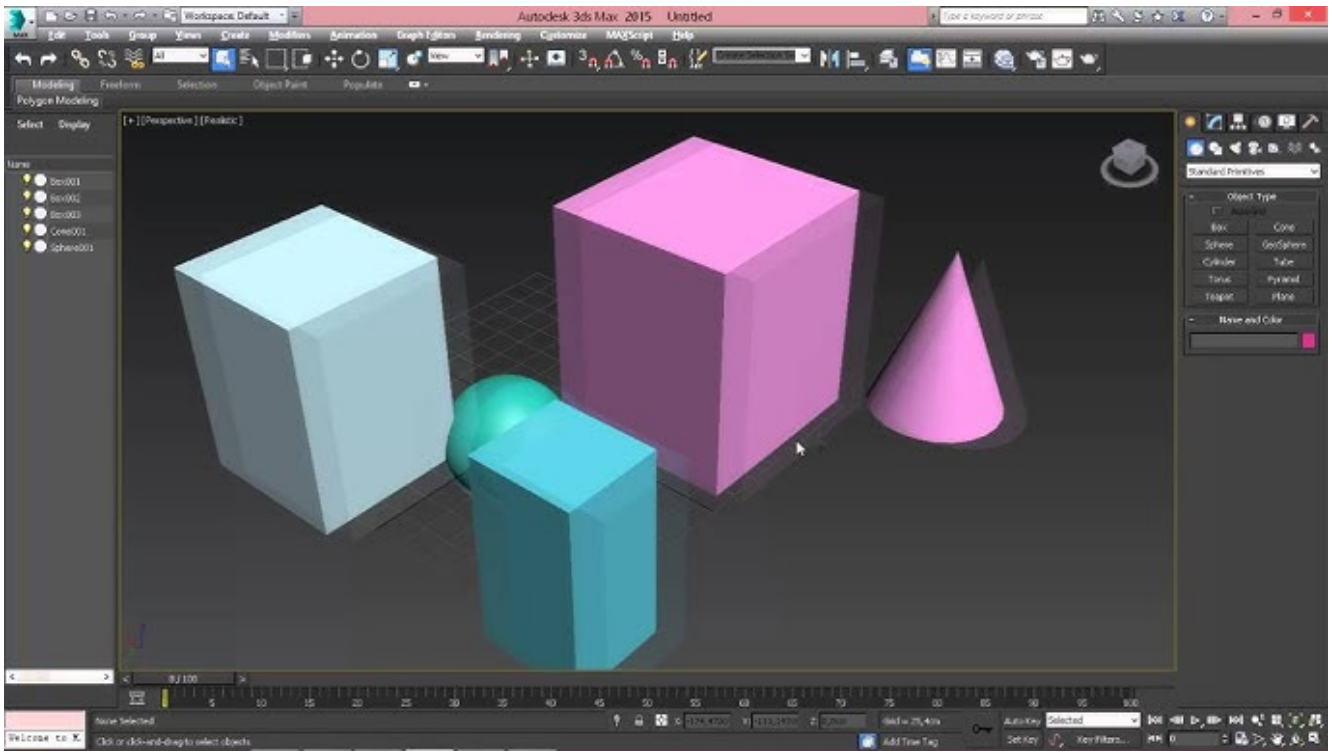


Рисунок 1.3. Інтерфейс програми Autodesk 3ds Max

Основні можливості Autodesk 3ds Max:

- Інструменти для полігонального моделювання та скульптури;
- Підтримка процедурного моделювання за допомогою плагіна MCG (Max Creation Graph);
- Рендеринг з використанням рушія Arnold;

– Засоби для створення анімацій та інтеграція з технологіями VR;
– Підтримка численних плагінів, що розширюють функціональні можливості програми.

3ds Max є потужним інструментом для створення архітектурних візуалізацій, однак в порівнянні з Maya має більш зручний інтерфейс для моделювання та роботи з геометрією.

1.1.4 Програмне забезпечення Houdini

Houdini від SideFX — це інструмент для створення процедурної тривимірної графіки та спецефектів. Програма широко використовується в кінематографі для створення реалістичних симуляцій динамічних об'єктів, таких як вибухи, рідини, дим і інші складні ефекти.

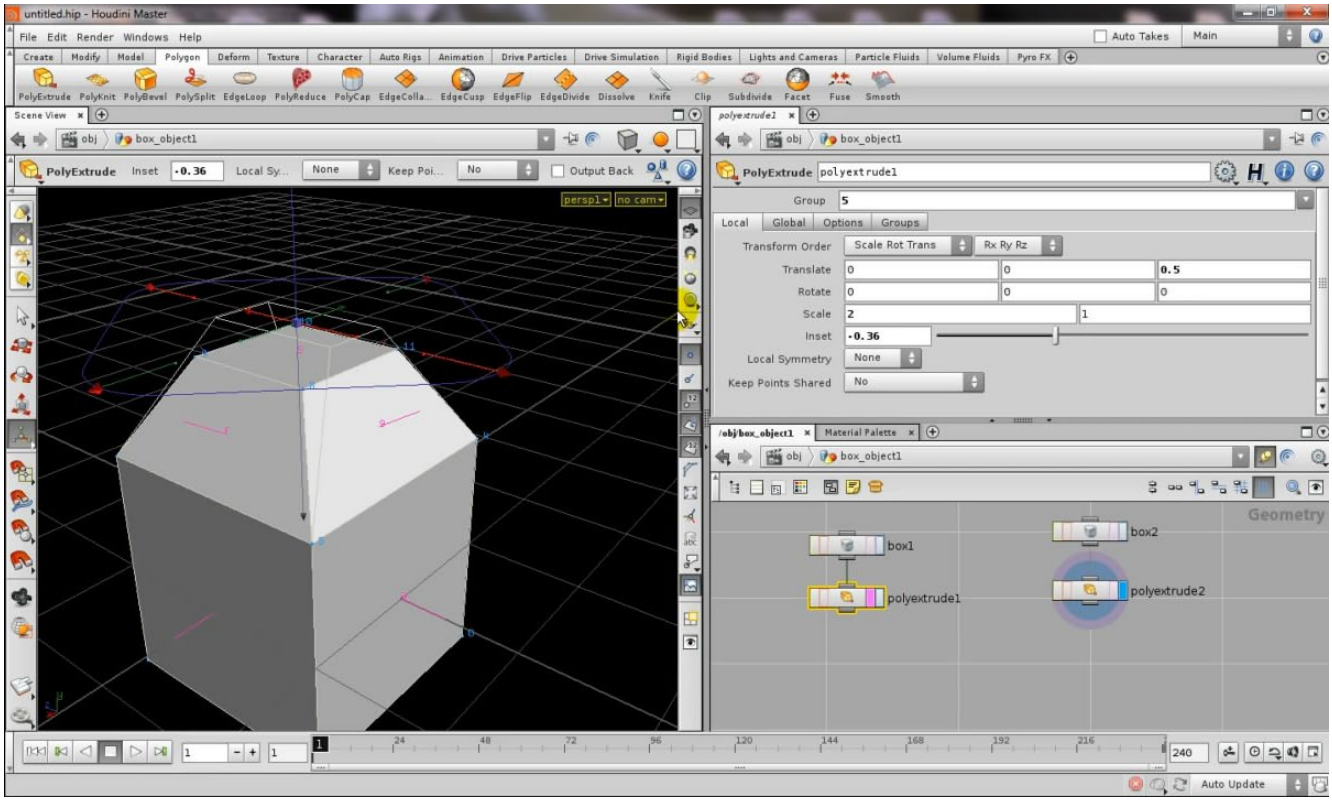


Рисунок 1.4. Інтерфейс програми Houdini

Основні можливості Houdini:

– Процедурне моделювання, яке дозволяє створювати складні сцени на основі алгоритмів;
– Підтримка фізичних симуляцій для створення реалістичних ефектів;

– Інтеграція з іншими інструментами для рендерингу та композитингу;
– Високий рівень налаштовуваності через скрипти та процедурні мережі.

Houdini є потужним інструментом для створення процедурної графіки, але він має високий поріг входу через складність процедурного підходу до моделювання.

1.1.5 Програмне забезпечення ZBrush

ZBrush — спеціалізоване програмне забезпечення для цифрової скульптури, яке дозволяє створювати високодеталізовані 3D-моделі. На відміну від традиційного полігонального моделювання, ZBrush використовує підхід, заснований на скульптурі, що робить його популярним інструментом серед художників і дизайнерів персонажів.

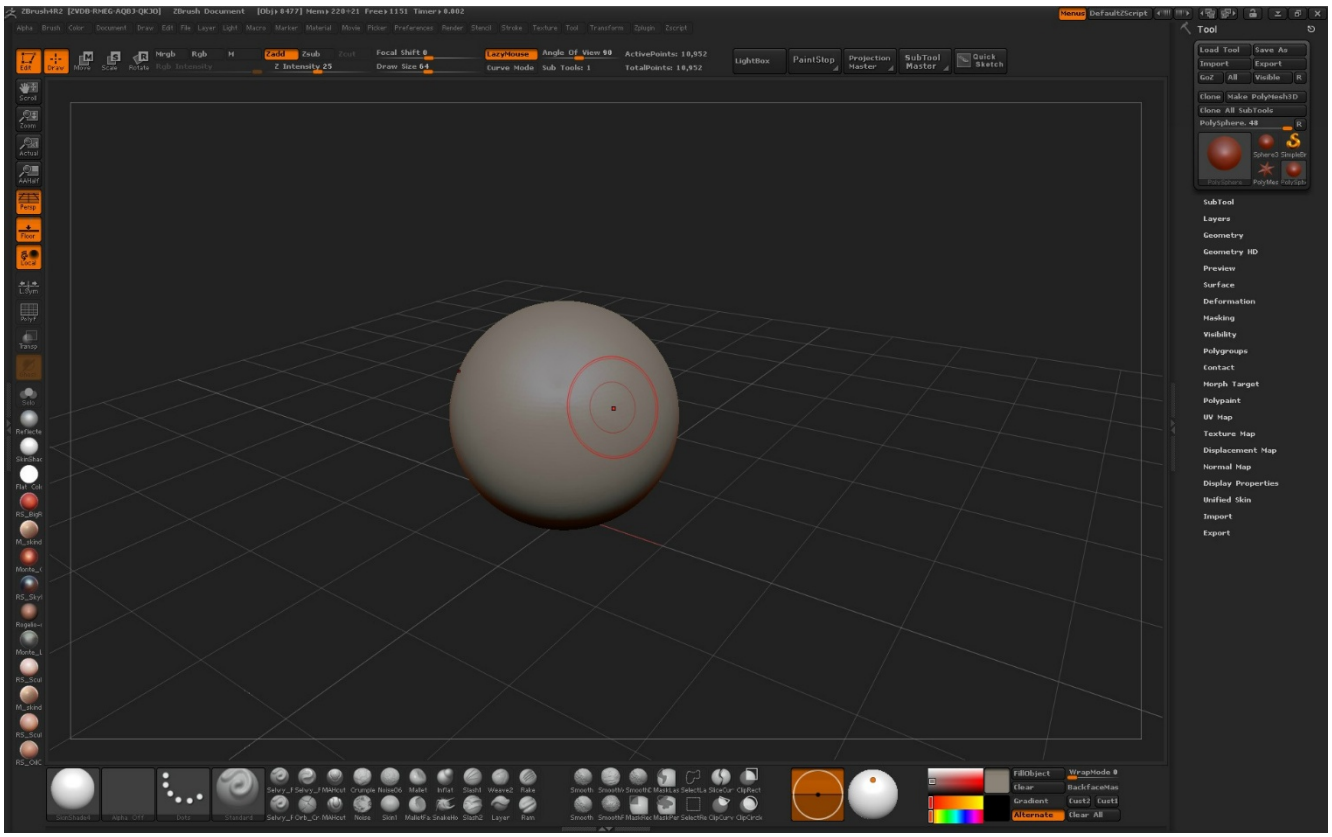


Рисунок 1.5. Інтерфейс програми ZBrush

Основні можливості ZBrush:

– Деталізована цифрова скульптура з можливістю створення моделей із мільйонами полігонів;
– Підтримка динамічних текстур та кольорів у процесі моделювання;

– Інструменти для ретопології моделей, що дозволяє зменшувати кількість полігонів для використання у відеоіграх чи анімаціях;

– Інтеграція з іншими 3D-пакетами для експорту та рендерингу;

– ZBrush є основним інструментом для створення персонажів та органічних моделей, але через свій унікальний підхід до моделювання може бути складним для новачків.

1.1.6 Програмне забезпечення SketchUp

SketchUp — це інструмент для тривимірного моделювання, орієнтований на простоту використання та швидке створення моделей. Програма широко використовується в архітектурі, дизайні інтер'єрів та промисловому дизайні.

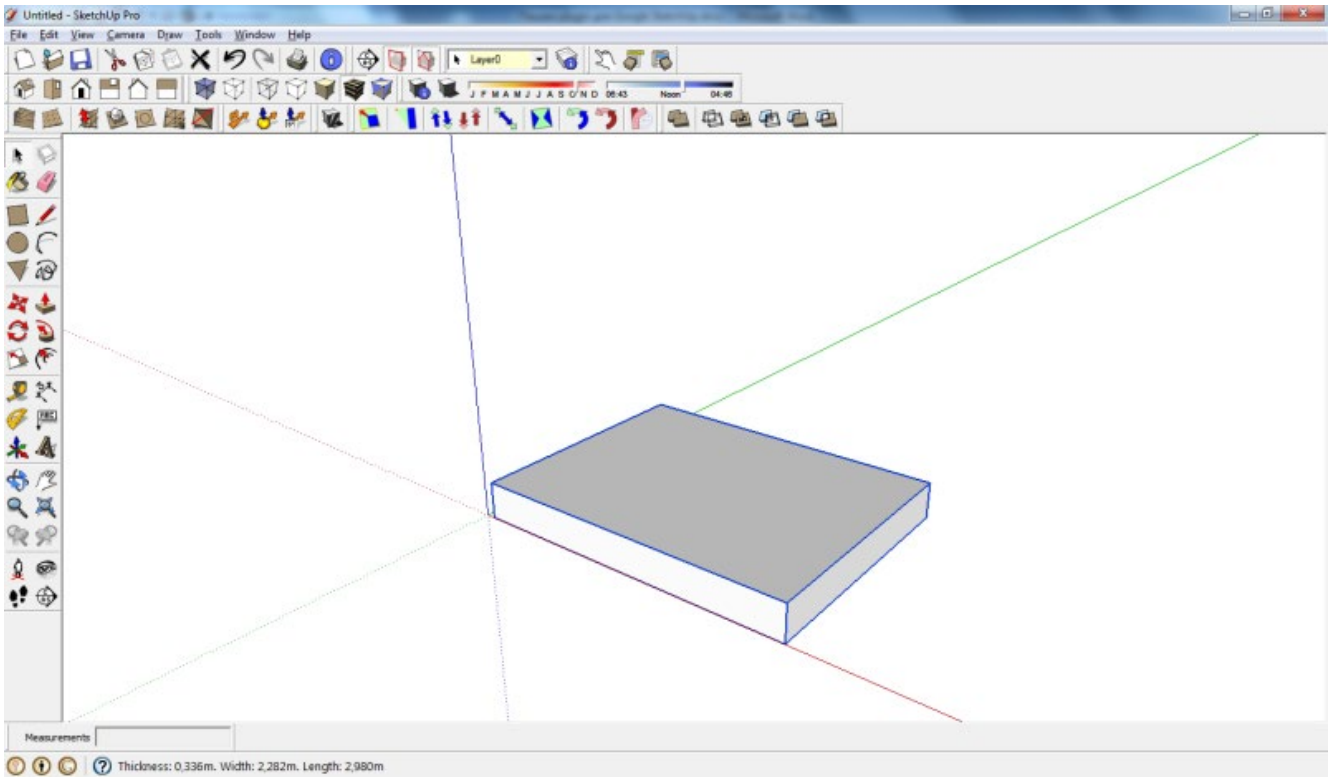


Рисунок 1.6. Інтерфейс програми SketchUp

Основні можливості SketchUp:

- Простий інтерфейс, орієнтований на користувачів з мінімальним досвідом у 3D-моделюванні;
- Інструменти для швидкого створення архітектурних моделей та візуалізацій;
- Велика бібліотека готових моделей, доступних через 3D Warehouse;
- Інтеграція з різними плагінами для рендерингу та розширення функціоналу.

SketchUp є ідеальним вибором для початківців або професіоналів, які хочуть швидко створювати архітектурні моделі без необхідності глибокого освоєння складних програм.

1.1.7 Порівняння характеристик програмного забезпечення генерування тривимірної графіки для 3D-моделювання

У процесі розробки програмного забезпечення для тривимірної графіки важливо враховувати потреби користувачів та особливості різних програмних засобів. Вибір інструменту залежить від конкретних завдань, бюджету, вимог до якості та складності проєкту.

Таблиця 1.1. Порівняння характеристик програмних засобів для 3D-моделювання

Програма	*Цільове використання*	*Тип моделювання*	*Ціна*	*Складність освоєння*	*Ключові особливості*
Blender	Мультизадачність, створення 3D-моделей, анімацій, рендеринг	Полігональне, скульптура, текстурування	Безкоштовна	Середня	Відкритий код, широкий набір інструментів, рендер Cycles
Autodesk Maya	Професійне 3D-моделювання, спецефекти, анімація	Полігональне, NURBS	Платна (підписка)	Висока	Підтримка складних анімацій, інтеграція з іншими продуктами
Autodesk 3ds Max	Архітектурне моделювання, дизайн інтер'єрів, відеоігри	Полігональне	Платна (підписка)	Середня	Процедурне моделювання, підтримка плагінів, рендер Arnold
Houdini	Спецефекти, процедурне моделювання, кінематографія	Процедурне	Платна (є безкоштовна версія Apprentice)	Висока	Процедурні мережі, динамічні симуляції, спецефекти
ZBrush	Скульптура, створення персонажів та високодеталізованих моделей	Цифрова скульптура	Платна (одноразовий платіж)	Висока	Деталізована скульптура, ретопологія, інтеграція з іншими 3D-програмами
SketchUp	Архітектура, дизайн інтер'єрів, промисловий дизайн	Полігональне	Безкоштовна (є платна версія)	Низька	Простий інтерфейс, швидке створення архітектурних моделей

1.2 Аналіз принципів побудови тривимірних моделей

Тривимірне моделювання полягає у створенні математичних описів об'єктів у тривимірному просторі. Ці моделі можуть бути створені вручну або автоматично за допомогою спеціальних алгоритмів і технік. Тривимірні моделі складаються з точок, ліній, багатокутників і поверхонь, що утворюють форму об'єкта. Основними компонентами будь-якої 3D-моделі є вершини (точки у просторі) та багатокутники (полігони), які об'єднуються для створення поверхонь.

1.2.1 Основні принципи програмної реалізації 3D-моделей

1. Вершини (точки у просторі). Вершина — це основний структурний елемент 3D-моделі, який представляє координати точки в тривимірному просторі. Кожна вершина має три координати (x, y, z), які визначають її положення. Визначення вершини у просторі:

$$V_i=(x_i,y_i,z_i) \qquad (1.1)$$

де V_i — це вершина, x_i, y_i, z_i — координати вершини у просторі.

2. Ребра та полігони. Полігон — це сукупність вершин, об'єднаних у замкнену форму (наприклад, трикутник чи чотирикутник). Полігони формують поверхню об'єкта (рис.1.7). Трикутники найчастіше використовуються через їх простоту та універсальність. Площа трикутника, визначеного трьома вершинами:

$$A = \frac{1}{2} \cdot |V_1 \cdot (V_2 \times V_3)| \qquad (1.2)$$

де A — це площа трикутника, V_1,V_2,V_3 — це вершини трикутника.

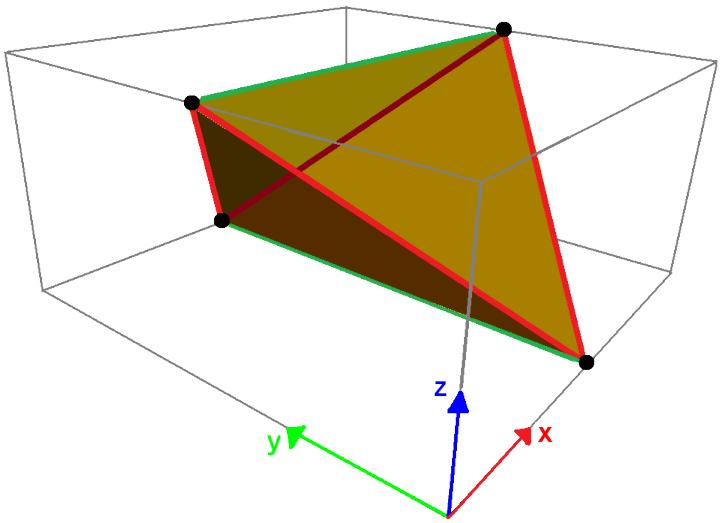


Рисунок 1.7. Багатокутник у тривимірному просторі

3. Сітка. Сітка — це структура, яка складається з вершин і полігонів. Вона визначає форму об'єкта у 3D-просторі. Кожна вершина в сітці має свої координати, і через комбінацію вершин формується поверхня об'єкта. Маніпулюючи вершинами, користувач може змінювати форму моделі (рис.1.8).

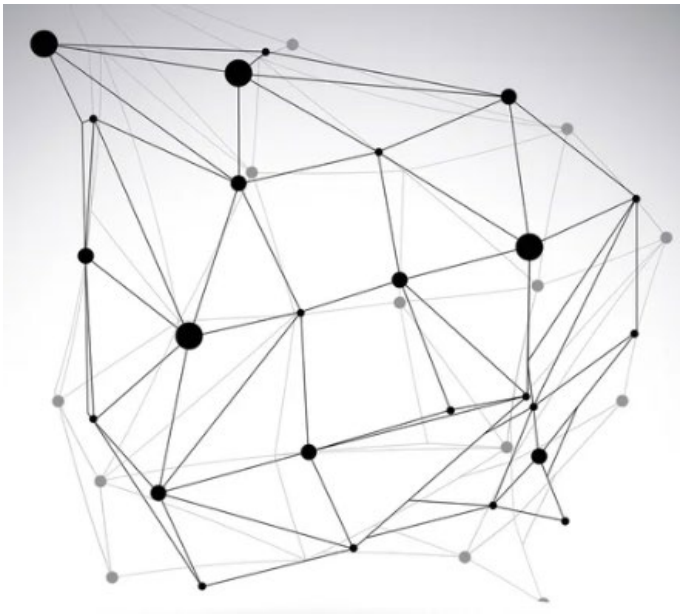


Рисунок 1.8. Тривимірна сітка об'єкта

1.2.2 Алгоритми створення тривимірних об'єктів

Існує кілька підходів для генерування 3D-моделей:

1. Автоматичне генерування. Автоматичне створення моделей може здійснюватися за допомогою алгоритмів, таких як генерація за допомогою шуму Перліна або фрактальних алгоритмів. Ці алгоритми часто використовуються для створення ландшафтів або органічних структур, таких як гори, дерева тощо.

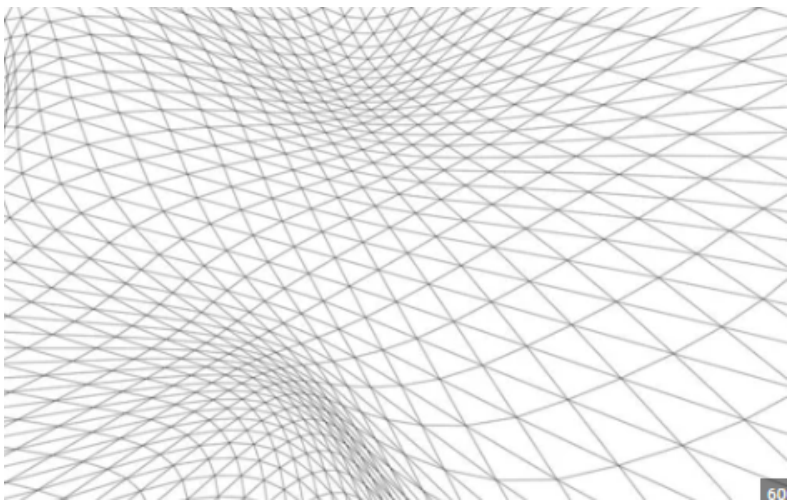


Рисунок 1.9. Деформація об'єкта шляхом маніпулювання вершинами

2. Ручне створення та деформація. Ручне моделювання передбачає зміну форми об'єкта шляхом маніпуляції його вершинами, ребрами і полігонами. Користувач може витягувати, масштабувати або обертати частини об'єкта для створення бажаної форми (рис.1.9).

1.2.3 Процес створення анімованого цифрового об'єкта

Для того, щоб тривимірний об'єкт був здатний до анімації, він зазвичай створюється з використанням технології риггінгу. Це процес додавання «скелета» або системи кісток до моделі. Кожна кістка відповідає за частину сітки, дозволяючи створювати плавні рухи при анімації.

Створення кісток дозволяє анімувати персонажів або об'єкти. Сітка об'єкта «прив'язується» до цих кісток, що дозволяє їх гнучке переміщення та деформацію без втрати структури (рис.1.10).

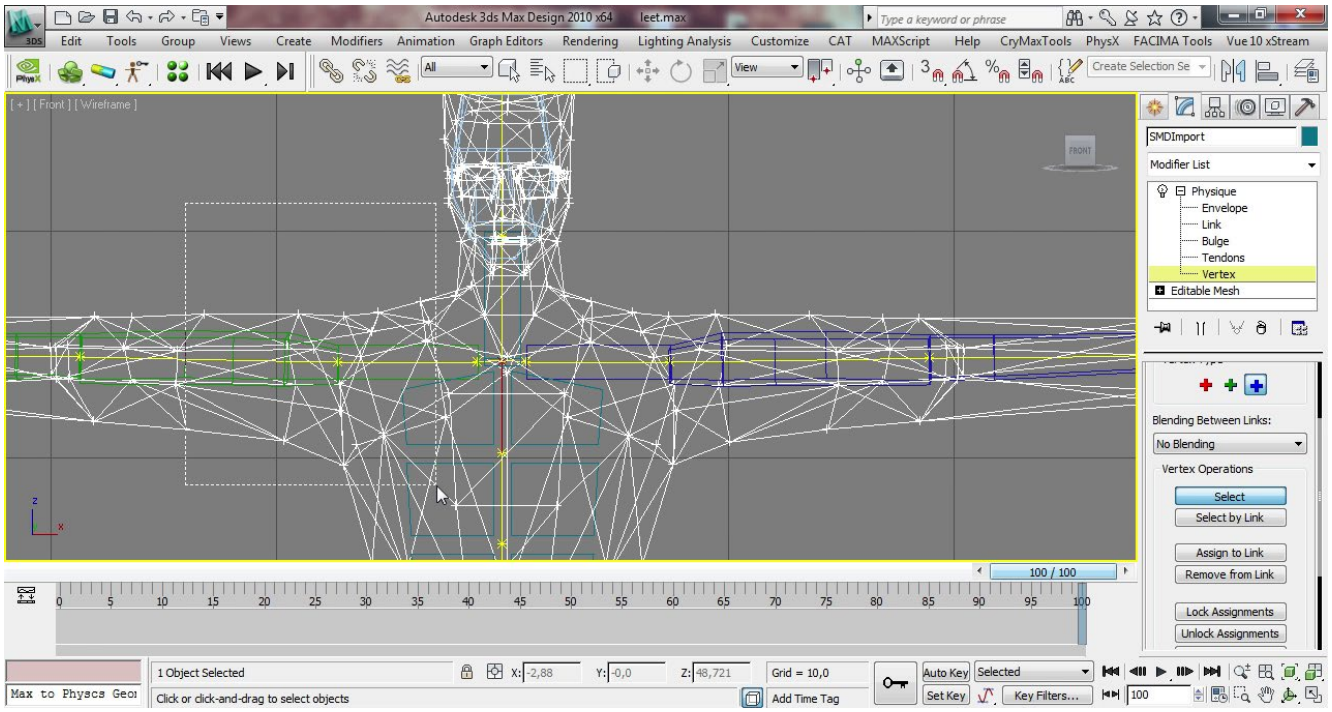


Рисунок 1.10. Риггінг для анімації персонажа

1.2.4 Алгоритми опису тривимірної моделі

1. Алгоритм Bounding Volume Hierarchy (BVH). Це алгоритм, що використовує ієрархію обмежуючих об'ємів для швидкої перевірки перетинів об'єктів у 3D-просторі. Він часто використовується у відеоіграх і рендерингу для прискорення процесу зіткнення об'єктів.

2. Алгоритм Marching Cubes. Використовується для генерації тривимірних поверхонь із воксельних даних. Алгоритм знаходить межі поверхонь, що проходять через вокселі, і створює полігони на їх основі.

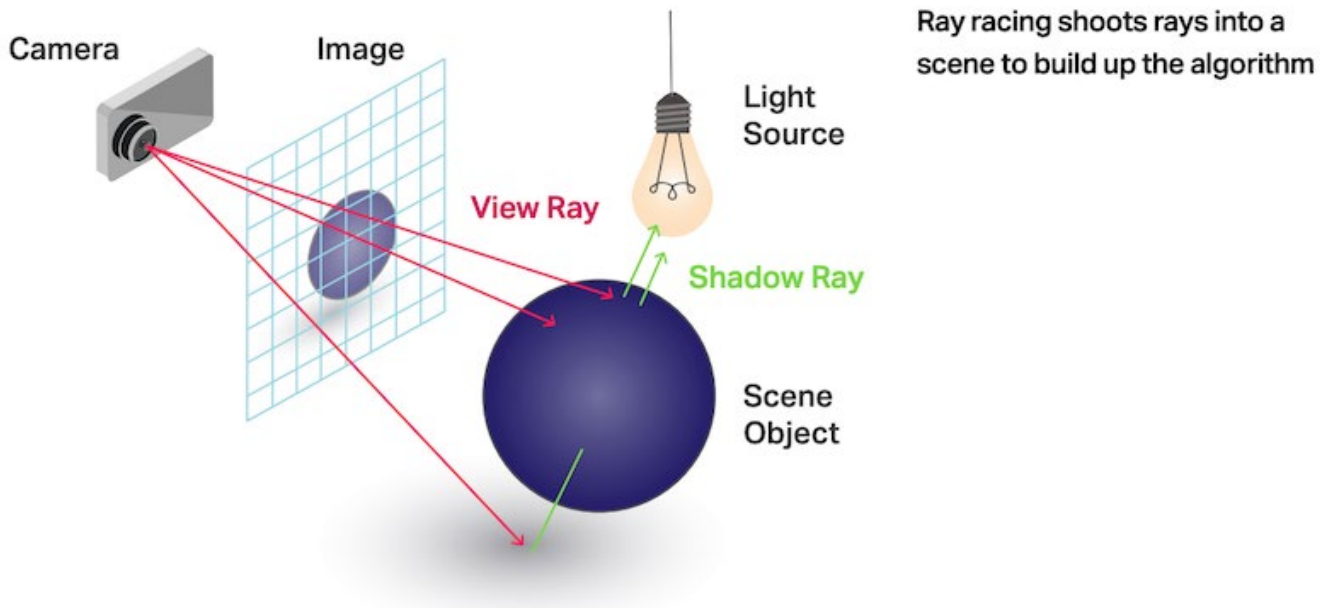


Рисунок 1.11. Ілюстрація алгоритму Ray tracing

3. Алгоритми освітлення:
   - Phong shading — використовує нормалі поверхонь для розрахунку освітлення на кожній точці полігону.
   - Ray tracing — складніший алгоритм, який моделює поведінку променів світла, щоб створювати реалістичні тіні, відображення та рефракції (рис.1.11).

1.2.5 Експорт 3D-моделей

Після створення тривимірної моделі, її можна експортувати у формати, які використовуються іншими програмами для анімації, ігор або фільмів. Найбільш поширені формати для експорту — це FBX, OBJ та STL. Ці формати підтримують передачу даних про геометрію, матеріали та анімацію.

- FBX — формат для передачі складних сцен із текстурами та анімаціями між різними програмами;
- OBJ — простий формат для збереження геометрії та текстур;
- STL — популярний формат для 3D-друку.

1.2.6 Технології 3D-друку та стереолітографія (SLA)

3D-друк використовує тривимірні моделі для фізичного створення об'єктів. Одним із найбільш поширених методів є стереолітографія (SLA), яка використовує лазер для затвердіння фотополімерів у рідкому стані шар за шаром, створюючи фізичну модель.

Формат STL (стереолітографія) є стандартом для передачі 3D-моделей у 3D-принтери. Цей формат зберігає дані про геометрію об'єкта у вигляді трикутників (рис.1.12).

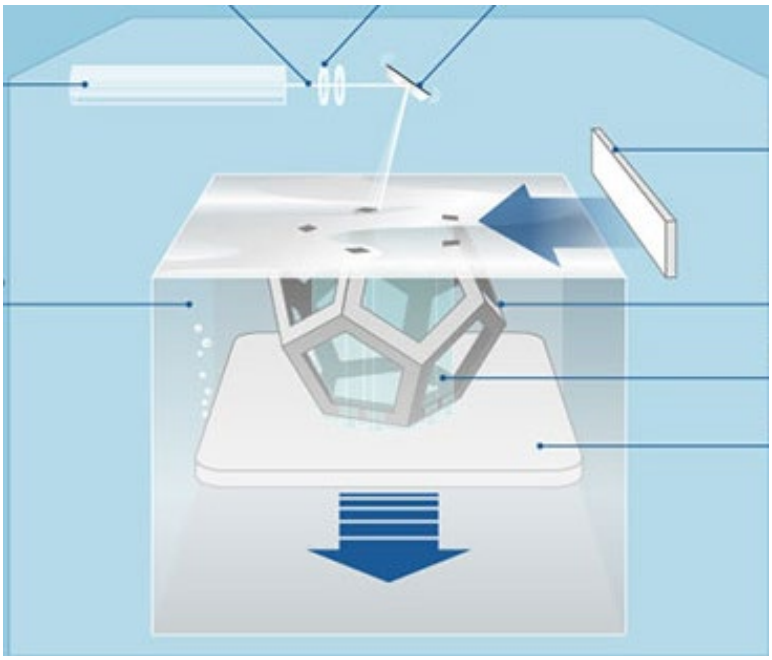


Рисунок 1.12. Принцип стереолітографії

1.2.7 3D-рендеринг

Процес 3D-рендерингу полягає в тому, щоб перетворити тривимірну модель на двовимірне зображення або відео з використанням алгоритмів освітлення, текстур і тіней. Одним із найбільш використовуваних методів рендерингу є Ray tracing, який обчислює траєкторію світла і його взаємодію з поверхнями.

Інші методи включають:

- Rasterization — швидший, але менш точний метод, який використовується в реальному часі у відеоіграх;
- Global illumination — розширений метод освітлення, що враховує не тільки пряме, але й непряме світло.

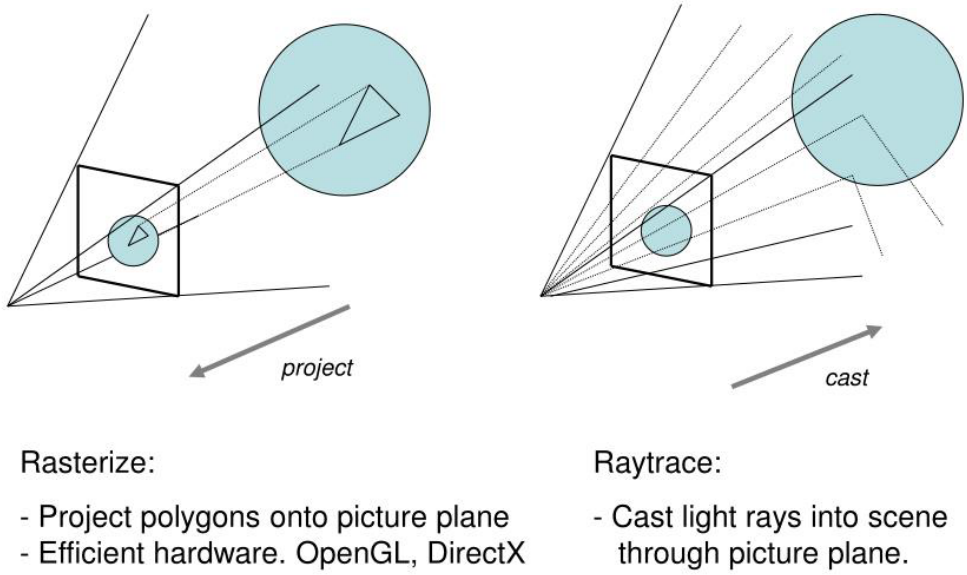


Рисунок 1.13. Порівняння методів рендерингу: Ray tracing та Rasterization

Створення тривимірних моделей — це складний процес, який включає використання різних алгоритмів і технологій для отримання реалістичних об'єктів. Вибір методу залежить від конкретного завдання, бюджету та вимог до точності. Тривимірні моделі активно використовуються в багатьох галузях, таких як кіно, ігри, анімація та інженерія. Окрім того, завдяки технологіям 3D-друку, створені моделі можуть бути перетворені на фізичні об'єкти.

1.3 Огляд основних алгоритмів тривимірного моделювання

1.3.1 Алгоритм полігонального моделювання та його реалізація

Полігональне моделювання є основним методом створення тривимірних графічних об’єктів у сучасній комп’ютерній графіці. Цей підхід полягає в розбитті поверхні об'єкта на багатокутники, зазвичай трикутники або чотирикутники, що забезпечує ефективну візуалізацію і подальше опрацювання моделі. Даний метод дозволяє досягти високої деталізації об’єктів при збереженні прийнятної продуктивності обчислень.

Полігональне моделювання складається з кількох основних етапів, кожен із яких забезпечує поступове створення тривимірної моделі. Нижче наведено ключові етапи разом із відповідними формулами:

1. Створення початкової форми

Створення початкової форми зазвичай відбувається на основі геометричних примітивів, таких як куб, сфера або циліндр. Ці форми задаються математичними рівняннями. Наприклад, параметричне рівняння сфери радіуса r в тривимірному просторі може бути представлене так:

$$x(\theta,\phi) = r \cdot sin(\theta) \cdot cos(\phi)$$
$$y(\theta,\phi) = r \cdot sin(\theta) \cdot sin(\phi) \quad (1.3)$$
$$z(\theta,\phi) = r \cdot cos(\theta)$$

де θ і ϕ — кутові координати в сферичній системі координат.

2. Розбиття на полігони

Для полігонального моделювання важливо, щоб поверхня об'єкта була розбита на окремі трикутники або чотирикутники. Один із поширених алгоритмів для цього — алгоритм Делоне. Він полягає в побудові тріангуляції множини точок таким чином, щоб не було гострих кутів між трикутниками. Математична задача для побудови тріангуляції може бути сформульована як:

$$\mathrm{T} = \arg\max_{T_i \in S} \min \angle(T_i) \quad (1.4)$$

де T — сукупність трикутників, а $\angle(T_i)$ — кути трикутника T_i.

3. Деформація полігональної сітки

На етапі деформації здійснюється перетворення геометрії об’єкта шляхом маніпуляції вершинами сітки. Для цього часто використовуються інструменти, такі як екструзія (витягування поверхні). Формально екструзію можна описати за допомогою формули:

$$V_{\text{new}} = V_{\text{old}} + d \cdot n \quad (1.5)$$

де:

- V_{new} — нова позиція вершини;
- V_{old} — початкова позиція вершини;
- d — величина екструзії;
- n — нормаль до поверхні.

4. Згладжування поверхонь

Для досягнення більш реалістичної форми об’єкта застосовується метод субдивізії поверхні. Алгоритм Catmull-Clark є одним із популярних методів

згладжування полігональних сіток (табл.1.2). Він полягає у повторному розбитті граней на менші частини, поступово згладжуючи форму об'єкта. Розрахунок нових координат вершини для кожної ітерації можна описати як:

$$P_{\text{new}} = \frac{P_{\text{old}} + \sum_{i=1}^{n} P_{\text{neigh}_i}}{n+1} \quad (1.6)$$

де:

- P_{new} — нова позиція вершини;
- P_{old} — стара позиція вершини;
- P_{neigh_i} — позиція сусідньої вершини;
- n — кількість сусідніх вершин.

5. Текстурування та накладання матеріалів

На цьому етапі модель отримує текстури та матеріали для надання реалістичного вигляду. Координати текстур залежать від UV-розгортки поверхні. Для цього кожна вершина моделі має бути співвіднесена з координатами текстури (u, v).

Таблиця 1.2. Порівняння методів згладжування поверхонь

Алгоритм	*Кількість операцій*	*Реалістичність рез-ту*	*Продуктивність*
Catmull-Clark	$O(n^2)$	Висока	Помірна
Loop Subdivision	$O(n^2)$	Середня	Висока
Butterfly	$O(n \log n)$	Низька	Висока

Реалізація алгоритму полігонального моделювання у застосунку за допомогою OpenGL дозволяє працювати з тривимірними об'єктами у реальному часі, забезпечуючи маніпуляцію вершинами та текстурами. Приклад коду для створення та рендерингу полігональної моделі наведено нижче:

```
// Ініціалізація полігональної сітки
GLuint vao, vbo;
glGenVertexArrays(1, &vao);
glGenBuffers(1, &vbo);
// Прив'язка та налаштування буфера
glBindVertexArray(vao);
glBindBuffer(GL_ARRAY_BUFFER, vbo);
```

```
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);
// Визначення атрибутів вершини (позиція, нормаль, UV)
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * sizeof(float),
(void*)(3 * sizeof(float)));
glEnableVertexAttribArray(1);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float),
(void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2);
// Рендеринг моделі
glBindVertexArray(vao);
glDrawArrays(GL_TRIANGLES, 0, num_vertices);
```

1.3.2 Алгоритм моделювання кривих NURBS та його реалізація

NURBS (Non-Uniform Rational B-Splines) — це потужний інструмент для моделювання кривих і поверхонь у тривимірному просторі. NURBS широко використовуються в CAD-системах (системах автоматизованого проектування), анімації, моделюванні поверхонь об'єктів у машинобудуванні та інших сферах. Основними перевагами NURBS є можливість точно описувати як прості, так і складні форми, використовуючи мінімальну кількість параметрів. Математична основа цього підходу забезпечує гнучкість і контроль над кривими та поверхнями.

Крива NURBS визначається такими елементами:

- Контрольні точки — визначають форму кривої;
- Вагові коефіцієнти — дозволяють контролювати вплив кожної контрольної точки на загальну форму;
- Вузловий вектор — визначає параметризацію кривої;
- Ступінь — показує, наскільки гладкою буде крива (лінійна, квадратична, кубічна тощо).

Математичне представлення NURBS-кривої виглядає так:

$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) \cdot P_i \cdot w_i}{\sum_{i=0}^{n} N_{i,p}(u) \cdot w_i} \quad (1.7)$$

де:

- C(u) — положення точки на кривій при параметрі u;
- $N_{i,p}(u)$ — функція базисних сплайнів ступеня p;
- P_i — контрольні точки;
- w_i — вагові коефіцієнти контрольних точок;
- u — параметр, який змінюється в межах вузлового вектора.

Базисні функції B-сплайнів $N_{i,p}(u)$ обчислюються рекурсивно за допомогою наступних формул:

для нульового ступеня:

$$N_{i,0}(u) = \begin{cases} 1, & \text{якщо } u_i \leq u < u_{i+1}, \\ 0, & \text{інакше.} \end{cases} \quad (1.8)$$

для ступеня p:

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \quad (1.9)$$

Алгоритм моделювання кривих NURBS складається з кількох основних етапів:

1. Визначення контрольних точок

Контрольні точки формують основу для побудови кривої. Наприклад, набір контрольних точок $P = \{P_0, P_1, P_2, P_3\}$ може бути представлений у вигляді матриці:

$$P = \begin{bmatrix} x_0 & y_0 & z_0 \\ x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{bmatrix} \quad (1.10)$$

2. Формування вузлового вектора

Вузловий вектор визначає параметризацію кривої. Для кубічної кривої із 4 контрольними точками можливий вузловий вектор може бути таким:

$$U = \{0, 0, 0, 1, 2, 2, 2\} \quad (1.11)$$

3. Розрахунок функцій базисних сплайнів

Базисні функції обчислюються за допомогою наведених вище рекурсивних формул. Для кожного параметра u із вузлового вектора, функція визначає, які контрольні точки матимуть найбільший вплив на криву в цьому інтервалі.

4. Обчислення координат точок кривої

Координати кожної точки кривої обчислюються за формулою NURBS-кривої. Наприклад, для параметра u = 1.5 обчислюється положення точки на кривій, враховуючи вагові коефіцієнти та контрольні точки.

Таблиця 1.3. Приклад вузлового вектора та контрольних точок для NURBS-кривої

Контрольна точка	*Координати (x, y, z)*	*Ваговий коефіцієнт*
P_0	(0, 0, 0)	1.0
P_1	(1, 2, 0)	0.5
P_2	(3, 5, 0)	1.0
P_3	(4, 0, 0)	0.8

Для реалізації алгоритму NURBS у додатку використовується бібліотека OpenGL та її розширення для роботи з кривими і поверхнями. Приклад фрагмента коду для обчислення NURBS-кривої виглядає наступним чином (рис.1.14):

```
// Ініціалізація контрольних точок
GLfloat controlPoints[4][3] = {
    {0.0, 0.0, 0.0},
    {1.0, 2.0, 0.0},
    {3.0, 5.0, 0.0},
    {4.0, 0.0, 0.0}
};
// Вузловий вектор
GLfloat knots[8] = {0.0, 0.0, 0.0, 1.0, 2.0, 2.0, 2.0};
// Ініціалізація NURBS об'єкта
GLUnurbs* nurbsObject = gluNewNurbsRenderer();
gluBeginCurve(nurbsObject);
```

gluNurbsCurve(nurbsObject, 8, knots, 3, &controlPoints[0][0], 4,
GL_MAP1_VERTEX_3);
gluEndCurve(nurbsObject);

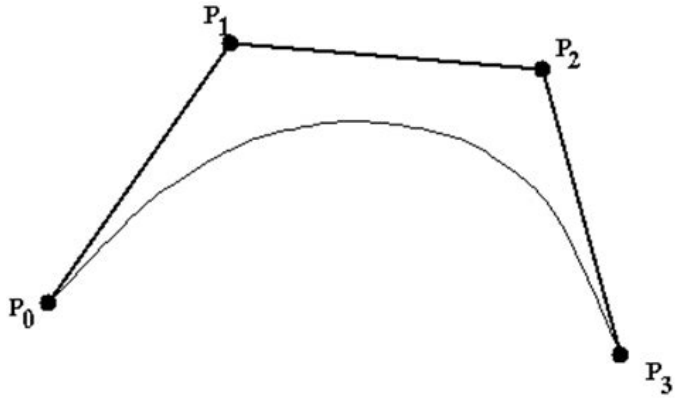


Рисунок 1.14. NURBS-крива, побудована за 4 контрольними точками

NURBS-криві є універсальним і потужним інструментом для моделювання складних геометричних форм із високою точністю. Їх використання у проекті забезпечує ефективне створення плавних, гладких кривих і поверхонь, що дозволяє генерувати реалістичні тривимірні об’єкти в застосунку.

1.3.3 Алгоритм процедурного моделювання та його реалізація

Процедурне моделювання — це метод створення тривимірних об'єктів та середовищ за допомогою алгоритмів і математичних процедур. Замість ручного моделювання кожного елемента сцени, процедурне моделювання дозволяє автоматично генерувати складні форми та текстури, використовуючи правила або математичні функції. Цей підхід широко використовується в ігровій індустрії, анімації, архітектурному моделюванні для генерації природних ландшафтів, міських структур, рослинності тощо.

Процедурне моделювання базується на використанні алгоритмів для генерації геометрії та текстур на основі параметрів, які можуть бути випадковими або детермінованими. Основними компонентами процедурного моделювання є:

- Формальні правила — визначають структуру та взаємодію між елементами;
- Фрактали — використовуються для створення нерегулярних і складних форм, таких як гірські ландшафти або дерева;
- Граматики Ліндемаєра (L-системи) — підхід, що використовується для

моделювання рослин та дерев;

- Шум Перліна та інші алгоритми шуму — широко використовуються для генерації текстур і висотних карт.

Процедурне моделювання може бути реалізоване за допомогою різних підходів в залежності від типу об’єкта. Ось кілька ключових етапів типового алгоритму:

1. Вибір математичної моделі

На початковому етапі необхідно вибрати математичну модель для генерації об'єкта. Наприклад, для генерації гірського ландшафту часто використовуються фрактальні алгоритми або шум Перліна. Математичне рівняння для генерації значень шуму Перліна можна описати так:

$$P(x,y) = \sum_{i=0}^{n} \frac{1}{2^i} \cdot \text{noise}\left(2^i x, 2^i y\right) \qquad (1.12)$$

де:

- P(x, y) — значення функції шуму для координат (x,y)(x, y),
- $\text{noise}(x,y)$ — значення базової функції шуму,
- n — кількість рівнів деталізації (октав).

2. Використання L-систем для моделювання рослин

L-системи використовуються для моделювання органічних структур, таких як дерева або кущі. L-система задається набором правил переписування символів і початковою аксіомою. Наприклад, правила для створення дерева можуть виглядати так:

- $A \rightarrow AB$
- $B \rightarrow A$

Кожен символ може інтерпретуватися як команда для графічної побудови (наприклад, A — це гілка, B — лист).

3. Генерація текстур за допомогою шуму Перліна

Шум Перліна є ключовим елементом для створення природних текстур, таких як хмари, земля, вода та інші нерегулярні поверхні. Він дозволяє отримувати плавні переходи між кольорами та відтінками. Формула для генерації

текстури на основі шуму Перліна може бути такою:

$$T(x, y) = \frac{\text{Perlin}(x, y) + 1}{2} \quad (1.13)$$

де T(x, y) — значення текстури в точці (x, y), а $\text{Perlin}(x, y)$ — результат обчислення функції шуму.

Таблиця 1.3. Порівняння процедурних алгоритмів для різних типів об'єктів

Алгоритм	*Тип об'єкта*	*Рівень складності*	*Реалістичність результату*
Шум Перліна	Ландшафти, текстури	Середній	Висока
L-системи	Дерева, рослинність	Низький	Висока
Фрактали	Гори, гірські масиви	Високий	Висока
Шум Вороново	Камені, поверхні	Середній	Середня

Процедурне моделювання було реалізовано в цьому проекті за допомогою використання шуму Перліна для генерації ландшафтів і L-систем для моделювання дерев. Ці алгоритми інтегровані з OpenGL для візуалізації тривимірних об'єктів у реальному часі.

Приклад коду для генерації ландшафту за допомогою шуму Перліна:

```
float perlinNoise(float x, float y) {
    // Функція для генерації значення шуму Перліна для точки (x, y)
    return glm::perlin(glm::vec2(x, y));
}
void generateTerrain() {
    for (int i = 0; i < terrainWidth; ++i) {
        for (int j = 0; j < terrainHeight; ++j) {
            float x = (float)i / terrainWidth;
            float y = (float)j / terrainHeight;
            heightMap[i][j] = perlinNoise(x * scale, y * scale);
        }
    }
}
```

Приклад коду для моделювання дерева з використанням L-системи:

```
std::string axiom = "A";
std::unordered_map<char, std::string> rules;
rules['A'] = "AB";
rules['B'] = "A";
std::string generateTree(int iterations) {
```

```
    std::string result = axiom;
    for (int i = 0; i < iterations; ++i) {
        std::string newResult = "";
        for (char c : result) {
            newResult += rules[c];
        }
        result = newResult;
    }
    return result;
}
void drawTree(const std::string& tree) {
    for (char c : tree) {
        if (c == 'A') {
            // Намалювати гілку
        } else if (c == 'B') {
            // Намалювати листя
        }
    }
}
```

Процедурне моделювання є ефективним підходом для автоматизованого створення складних тривимірних об'єктів та середовищ із мінімальними витратами часу. Використання алгоритмів шуму Перліна для ландшафтів і L-систем для рослин дозволяє створювати реалістичні сцени з високою деталізацією. Реалізація цих алгоритмів у рамках даного проекту значно покращила можливості генерації тривимірних об'єктів у застосунку.

1.3.4 Алгоритм сплайнового моделювання та його реалізація

Сплайнове моделювання є одним із найпоширеніших методів створення гладких, неперервних кривих та поверхонь у комп'ютерній графіці та тривимірному моделюванні. Сплайни використовуються для представлення кривих з контрольними точками, що дозволяє будувати складні форми з плавними вигинами. Цей метод часто застосовується в анімації, моделюванні поверхонь автомобілів, літаків, архітектурних конструкцій та в багатьох інших галузях, де необхідно створювати геометрично точні та гладкі об’єкти.

Сплайни являють собою функції, що описують сегменти кривих, які з'єднуються в заданих контрольних точках. Вони забезпечують гладкість і неперервність кривих, водночас зберігаючи гнучкість у налаштуванні форми

через зміщення контрольних точок.

Найбільш поширені види сплайнів:

- Поліноміальні сплайни — криві, задані поліноміальними рівняннями, такими як лінійні або кубічні сплайни;
- Кубічні B-сплайни — забезпечують більш гладкі та гнучкі криві завдяки використанню кубічних поліномів;
- NURBS (раціональні B-сплайни) — дозволяють описувати як звичайні криві, так і кола, еліпси, поверхні другого порядку та інші складні форми.

Кубічні сплайни — це один із найпоширеніших видів сплайнів, які використовують поліноми третього степеня для опису кривої між двома точками. Формула кубічного сплайну для одного сегмента кривої має вигляд:

$$S(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \qquad (1.14)$$

де:

- S(x) — значення сплайну в точці xx,
- x_i — координата початкової точки сегмента,
- a_i, b_i, c_i, d_i — коефіцієнти, які визначають форму кривої між двома точками.

Для побудови кубічного сплайну на проміжку від x_0 до x_n, потрібно вирішити систему рівнянь для всіх сегментів. Для цього використовують такі умови:

1. Неперервність сплайна на кожному проміжку;
2. Гладкість сплайна, тобто неперервність першої та другої похідних;
3. Граничні умови, які залежать від типу сплайну (натуральний, напівнатуральний тощо).

B-сплайни є узагальненням кубічних сплайнів і забезпечують більшу гнучкість в описі кривих. Вони дозволяють змінювати форму кривої шляхом переміщення контрольних точок без зміни загальної форми кривої. Загальне рівняння B-сплайну виглядає так:

$$C(u) = \sum_{i=0}^{n} N_{i,k}(u) P_i \qquad (1.15)$$

де:

- C(u) — точка на кривій для параметра u;
- $N_{i,k}(u)$ — базисна функція B-сплайну, яка залежить від контрольних точок і ступеня сплайну k;
- P_i — контрольні точки, що визначають форму кривої.

Для побудови сплайнів на мові програмування C++ можна використовувати бібліотеку Eigen для роботи з матрицями, яка дозволить ефективно розраховувати параметри сплайнів:

```
#include <iostream>
#include <vector>
#include <cmath>
#include <eigen3/Eigen/Dense>
using namespace std;
using namespace Eigen;
class CubicSpline {
private:
    vector<double> x, y;
    VectorXd a, b, c, d;
    VectorXd h;
public:
    CubicSpline(const vector<double>& x_points,
    const vector<double>& y_points) {
        x = x_points;
        y = y_points;
        int n = x.size() - 1;
        a = VectorXd::Zero(n + 1);
        b = VectorXd::Zero(n);
        c = VectorXd::Zero(n + 1);
        d = VectorXd::Zero(n);
        h = VectorXd::Zero(n);
        // Кроки між точками
        for (int i = 0; i < n; ++i) {
            h[i] = x[i + 1] - x[i];
        }
        // Визначення коефіцієнтів системи рівнянь
        MatrixXd A = MatrixXd::Zero(n + 1, n + 1);
        VectorXd B = VectorXd::Zero(n + 1);
        A(0, 0) = 1.0;
        A(n, n) = 1.0;
        for (int i = 1; i < n; ++i) {
            A(i, i - 1) = h[i - 1];
            A(i, i) = 2 * (h[i - 1] + h[i]);
            A(i, i + 1) = h[i];
```

```
            B[i] = 3 * ((y[i + 1] - y[i]) / h[i] - (y[i] - y[i - 1]) / h[i - 1]);
        }
        c = A.colPivHouseholderQr().solve(B);
        // Обчислення коефіцієнтів b та d
        for (int i = 0; i < n; ++i) {
            b[i] = (y[i + 1] - y[i]) / h[i] - h[i] * (2 * c[i] + c[i + 1]) / 3;
            d[i] = (c[i + 1] - c[i]) / (3 * h[i]);
        }
    }
    double evaluate(double x_value) {
        int i = x.size() - 2;
        while (i > 0 && x_value < x[i]) {
            --i;
        }
        double dx = x_value - x[i];
        return y[i] + b[i] * dx + c[i] * dx * dx + d[i] * dx * dx * dx;
    }
};
int main() {
    vector<double> x = {0, 1, 2, 3, 4, 5};
    vector<double> y = {0, 1, 0, 1, 0, 1};
    CubicSpline spline(x, y);
    for (double x_value = 0; x_value <= 5; x_value += 0.1) {
        cout << "Spline at x = " << x_value << " is " << spline.evaluate(x_value)
<< endl;
    }
    return 0;
}
```

Цей код демонструє реалізацію кубічного сплайну за допомогою бібліотеки Eigen для розв'язку систем лінійних рівнянь.

Таблиця 1.4. Порівняння видів сплайнів

Вид сплайна	*Використання*	*Гладкість кривої*	*Гнучкість в керуванні*
Кубічний сплайн	Просте моделювання	Висока	Середня
B-сплайн	Складні криві	Висока	Висока
NURBS	Інженерне моделювання, CAD	Дуже висока	Дуже висока

Приклад коду для генерації B-сплайну:

```
#include <iostream>
#include <vector>
#include <cmath>
```

```
using namespace std;
class BSpline {
private:
    vector<double> knots;
    vector<double> control_points;
    int degree;
public:
    BSpline(const vector<double>& knots_, const vector<double>&
control_points_, int degree)
        : knots(knots), control_points(control_points), degree(degree) {}
    double basis(int i, int k, double t) {
        if (k == 0) {
            return (knots[i] <= t && t < knots[i + 1]) ? 1.0 : 0.0;
        } else {
            double w1 = (t - knots[i]) / (knots[i + k] - knots[i]);
            double w2 = (knots[i + k + 1] - t) / (knots[i + k + 1] - knots[i + 1]);
            return w1 * basis(i, k - 1, t) + w2 * basis(i + 1, k - 1, t);
        }
    }
    double evaluate(double t) {
        double sum = 0.0;
        for (size_t i = 0; i < control_points.size(); ++i) {
            sum += basis(i, degree, t) * control_points[i];
        }
        return sum;
    }
};
int main() {
    vector<double> knots = {0, 0, 0, 1, 2, 3, 3, 3};
    vector<double> control_points = {0, 1, 0, 1, 0, 1};
    BSpline b_spline(knots, control_points, 2);
    for (double t = 0; t <= 3; t += 0.1) {
        cout << "B-Spline at t = " << t << " is " << b_spline.evaluate(t) << endl;
    }
    return 0;
}
```

Сплайнове моделювання є потужним інструментом для створення плавних кривих і поверхонь. Кубічні сплайни дозволяють моделювати складні форми з високим рівнем гладкості, а B-сплайни та NURBS розширюють можливості для керування формою за допомогою контрольних точок і параметричних функцій. У рамках цього проекту використання сплайнів забезпечує точне та гнучке моделювання тривимірних об'єктів з можливістю плавного редагування форм.

Таким чином можна зазначити, що полігональне моделювання є універсальним методом для створення тривимірних моделей з високою деталізацією. Реалізація цього підходу у дипломному проекті забезпечує ефективну роботу з тривимірними об'єктами, що є основою для розробки застосунку для редагування графічних об'єктів.

1.4 Вибір і обґрунтування інструментальних засобів розробки та компонентів застосунку

Для реалізації проекту застосунку для редагування тривимірних графічних об’єктів обрано специфічні інструменти з урахуванням їхніх можливостей, ефективності та підтримки необхідної функціональності. Обґрунтування вибору кожного з інструментів:

У якості інструментів для зчитування, запису та збереження тривимірних графічних моделей обрано мову програмування C++ через високу продуктивність та можливість прямого доступу до апаратних ресурсів, що важливо для роботи з об'ємними моделями. C++ дозволяє максимально оптимізувати код, забезпечуючи високу швидкість виконання, що критично для задач з рендерингом та обробкою тривимірних об’єктів.

Клас Mesh є ключовим для зберігання та обробки даних про сітки тривимірних моделей, оскільки кожен тривимірний об'єкт у нашому застосунку буде представлений через сітку, що складається з набору вершин і граней. Використання цієї структури дає можливість легко здійснювати маніпуляції з геометрією об'єктів, наприклад, перетворення, масштабування або обертання.

Управління камерою (Camera) є необхідним для рендерингу тривимірних об’єктів у відповідності до позиції користувача або заданих параметрів сцени. Камера контролює вид з об'єктів та їхнє відображення на екрані, що є важливим аспектом для створення реалістичних візуалізацій. Використання цього класу дає змогу гнучко налаштовувати перспективу та орієнтацію камери для досягнення бажаного ефекту. Промені (Ray) необхідні для застосування рейтрейсингу або інших складних візуальних ефектів, таких як освітлення, тіні, відображення та заломлення. Це основний метод для досягнення фотореалістичної графіки в

рендерингу, тому вибір цього інструменту є виправданим для забезпечення високоякісної візуалізації.

Світло (Light Controller) є основним елементом у візуалізації тривимірних сцен, тому необхідний компонент для контролю джерел світла. Цей інструмент дозволяє моделювати різні типи освітлення (точкове, направлене, розсіяне), що є важливим для досягнення реалістичних ефектів візуалізації. Завдяки цьому компоненту можна ефективно контролювати всі аспекти освітлення сцени.

Для прискорення процесу рендерингу використовується структура BVH (Bounding Volume Hierarchy), яка дозволяє організувати тривимірні об'єкти так, щоб перевірка перетину променів з об'єктами була ефективною навіть для складних сцен. Це значно покращує продуктивність, особливо на великих сценах з великою кількістю об'єктів, що є критичним для забезпечення швидкої реакції системи.

Використання матриць і векторів (Mat4, Vec3 та Vec4) для трансформацій є стандартним підходом у тривимірній графіці, оскільки це дозволяє ефективно обробляти перетворення об'єктів у просторі (масштабування, обертання, трансляція). Вектори та матриці дозволяють безпосередньо здійснювати всі необхідні математичні операції для маніпулювання тривимірними об'єктами.

У якості модулів для візуалізації обрано OpenGL через свою відкритість, портативність та можливість працювати з різними типами апаратного забезпечення. Це потужний і широко використовуваний API для рендерингу двовимірної та тривимірної графіки, що дозволяє ефективно реалізовувати різні візуальні ефекти. OpenGL дає можливість працювати на низькому рівні з графічним процесором, що дозволяє досягти максимальної продуктивності при обробці складних тривимірних сцен. Для програмування шейдерів та рендерингу (GL Program та GL Shader) OpenGL надає потужні можливості, дозволяючи створювати складні ефекти освітлення, текстурування, відображення та багато іншого. Використання шейдерів дозволяє програмістам створювати високопродуктивний та гнучкий код для візуалізації 3D моделей. Ці компоненти дають максимальну гнучкість у налаштуванні вигляду 3D сцени та об'єктів.

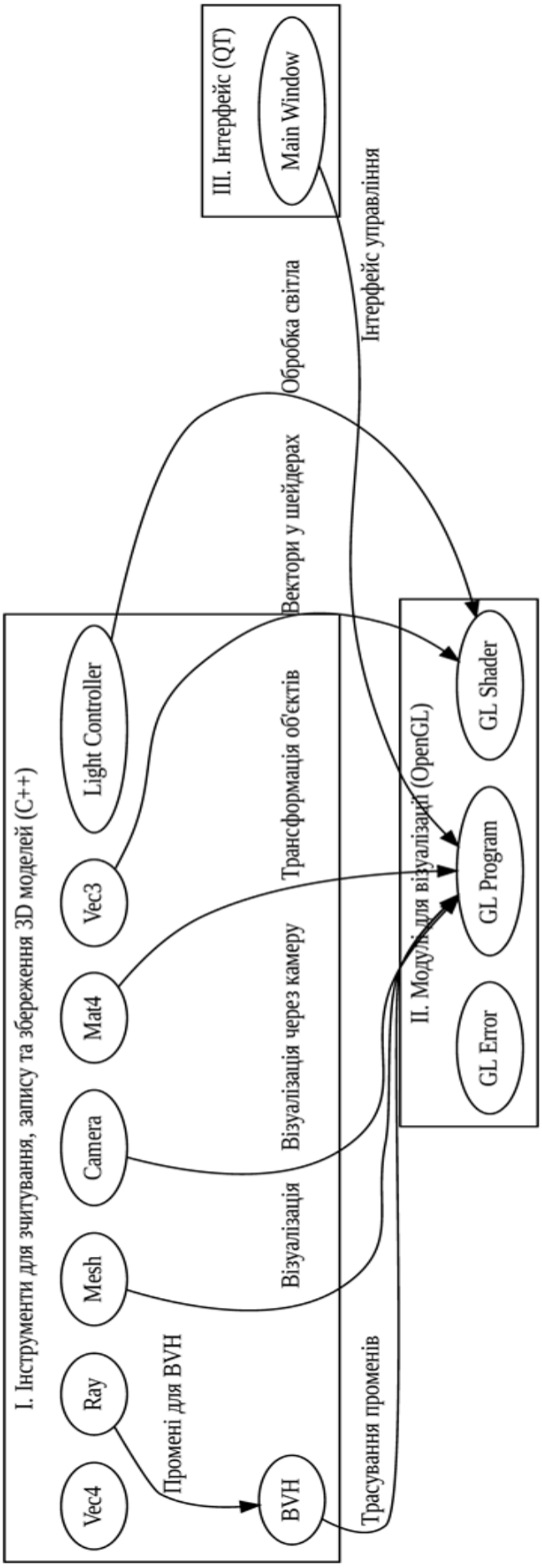


Рисунок 1.15. Структурна схема основних компонентів застосунку

Обробка помилок є важливою частиною будь-якої графічної програми. Модуль для виявлення та обробки помилок у OpenGL дає можливість швидко знаходити й усувати проблеми в процесі рендерингу, що забезпечує стабільність і коректність роботи програми.

Для створення графічного інтерфейсу обрано Qt через його потужні можливості створення кросплатформних додатків з інтерактивними елементами. Qt дозволяє швидко та ефективно розробляти інтерфейси, включаючи вікна, панелі інструментів та інші елементи управління, що забезпечує зручну взаємодію з користувачем. Зважаючи на потребу в інтерактивному управлінні тривимірними моделями, Qt надає необхідний функціонал для побудови зручного та інтуїтивно зрозумілого інтерфейсу.

Обрані інструменти є оптимальними для реалізації проекту, оскільки вони забезпечують високу продуктивність, гнучкість та ефективність у роботі з тривимірними моделями, надають потужні засоби для візуалізації та забезпечують зручний інтерфейс для користувачів. C++ і OpenGL забезпечують необхідну продуктивність для рендерингу та обробки складних тривимірних сцен, а Qt надає простий та зручний спосіб створення інтерфейсу для взаємодії з користувачем.

На основі вказаних у технічному завданні до дипломного проекту вимог та розглянутих у п.1.3 алгоритмів тривимірного моделювання, можна побудувати схему проекту для застосунку редагування тривимірних графічних об'єктів. Схема на рис.1.15 демонструє основні компоненти програми, що включають модулі для зчитування, візуалізації та інтерфейсу.

Секція інструментів для зчитування, запису та збереження тривимірних моделей містить компоненти, які відповідають за обробку тривимірних моделей. Вона написана на C++ і включає наступні класи та структури даних:

- Mesh – відповідає за опис та управління сітками об'єктів;
- Camera – контролює положення та параметри камери для рендерингу;
- Ray – реалізує промені для рейтрейсингу або інших операцій;
- Light Controller – відповідає за керування джерелами світла;
- BVH (Bounding Volume Hierarchy) – структура для прискорення трасування

променів;

- Mat4 – 4x4 матриці для перетворення об'єктів у просторі;
- Vec3 та Vec4 – вектори для обчислень в тривимірному та чотиривимірному просторах.

Для візуалізації використовується OpenGL. Основні компоненти:

- GL Program – забезпечує програмування шейдерів та рендеринг;
- GL Shader – відповідає за обробку шейдерів, що використовуються для візуалізації об'єктів;
- GL Error – модуль для обробки та виявлення помилок в OpenGL.

Ці компоненти відповідають за графічну частину проекту, обробляючи зображення та забезпечуючи якісне відображення тривимірних моделей.

Для взаємодії з користувачем використовується бібліотека Qt. Main Window – головне вікно програми, яке забезпечує графічний інтерфейс для управління та взаємодії з користувачем.

Логіка роботи застосунку для редагування тривимірних графічних об’єктів є такою:

1. Завантаження 3D моделі: За допомогою класу Mesh програма завантажує тривимірні об'єкти. Камера, яку контролює Camera, дозволяє здійснювати навігацію по сцені;

2. Світло та тіні: Використання контролерів світла дозволяє змінювати параметри освітлення сцени, а структура BVH оптимізує процеси трасування променів для реалістичних візуальних ефектів;

3. Візуалізація: OpenGL обробляє графічний вивід через шейдери та програми, що дозволяють отримати високоякісне зображення. Важливі кроки включають рендеринг сітки об'єктів та обробку ефектів освітлення;

4. Інтерфейс користувача: Через головне вікно Qt користувач може завантажувати моделі, налаштовувати камеру, змінювати параметри освітлення та інші графічні параметри.

Сценарій використання застосунку для редагування тривимірних графічних об’єктів є таким:

1. Користувач запускає програму та взаємодіє через головне вікно Qt;
2. Вибирається або завантажується 3D модель, яку відображає OpenGL;
3. Налаштовуються камера та світло для покращення сцени;
4. Виведене зображення відображається з урахуванням всіх графічних параметрів.

1.5 Розробка архітектури застосунку та ОО-моделі

Архітектура застосунку для редагування тривимірних графічних об’єктів є багаторівневою та модульною, де кожен компонент відповідає за певну функціональність. Основна мета архітектури – забезпечити гнучкість, незалежність компонентів та ефективність при обробці й візуалізації тривимірних моделей. Як показано на діаграмі класів (рис.1.16), програма складається з кількох ключових модулів:

1. Зчитування та обробка даних (Mesh, Vec3, Vec4). Основним класом для обробки тривимірних об'єктів є Mesh. Він відповідає за зчитування моделей з файлів формату .off, які містять вершини та грані полігональних об'єктів. Формат файлів *.off, використаний у проекті, дозволяє описувати об'єкти у тривимірному просторі за допомогою вершин та граней. Клас Mesh дозволяє:

- Завантажувати модель з файлу;
- Виконувати трансформації над об'єктами (масштабування, обертання, зсув);
- Оновлювати та зберігати стан об'єкта.

Клас використовує структури даних Vec3 та Vec4 для роботи з векторами у тривимірному та чотиривимірному просторах відповідно. Це дозволяє ефективно маніпулювати координатами вершин та здійснювати операції з ними.

2. Візуалізація об'єктів (GLProgram, GLShader). Для візуалізації тривимірних моделей використовується OpenGL. Головними компонентами є:

- GLProgram – клас, який відповідає за програмування шейдерів і рендеринг об'єктів. Він дозволяє завантажувати шейдери, налаштовувати параметри рендерингу та запускати процес відтворення графіки;
- GLShader – клас для роботи з окремими шейдерами. Шейдери дозволяють обчислювати візуальні ефекти, як-от освітлення, текстурування та інші

графічні операції, забезпечуючи відтворення тривимірних моделей.

Обидва ці класи активно взаємодіють для забезпечення рендерингу на графічному процесорі (GPU), використовуючи переваги паралельних обчислень та апаратного прискорення.

3. Управління світлом і камерою (LightSource, Camera):

- LightSource – клас, який відповідає за створення та управління джерелами світла у сцені. Світло є важливим аспектом для коректної візуалізації тривимірних об'єктів, адже без нього сцена виглядала б плоско та невиразно. LightSource дозволяє задавати положення, колір і інші параметри джерел світла;
- Camera – клас для управління камерою в тривимірному просторі. Камера задає точку огляду сцени, і користувач може змінювати її параметри (позицію, кут огляду тощо), щоб переглядати модель під різними ракурсами. Це забезпечує інтерактивність застосунку і гнучкість у вивченні об'єктів.

4. Оптимізація та трасування променів (BVH, Ray). Для прискорення процесу рендерингу використовується BVH (Bounding Volume Hierarchy) – структура даних, що дозволяє оптимізувати пошук перетинів між променями і об'єктами в сцені. BVH організовує моделі так, щоб значно скоротити кількість необхідних обчислень при трасуванні променів.

Клас Ray моделює промені, які використовуються в алгоритмах рейтрейсингу. Цей клас дозволяє проводити точні обчислення для виявлення взаємодії променів з поверхнями моделей, що є важливим для реалізації фотореалістичних ефектів, таких як відображення і заломлення світла.

5. Графічний інтерфейс користувача (MainWindow). Для побудови інтерфейсу користувача використовується бібліотека Qt, зокрема клас MainWindow. Цей клас реалізує головне вікно програми, де користувач взаємодіє з моделями, запускає рендеринг і змінює параметри сцени через графічний інтерфейс. Це дозволяє користувачу легко управляти програмою та вивчати тривимірні об'єкти без потреби в написанні додаткового коду.

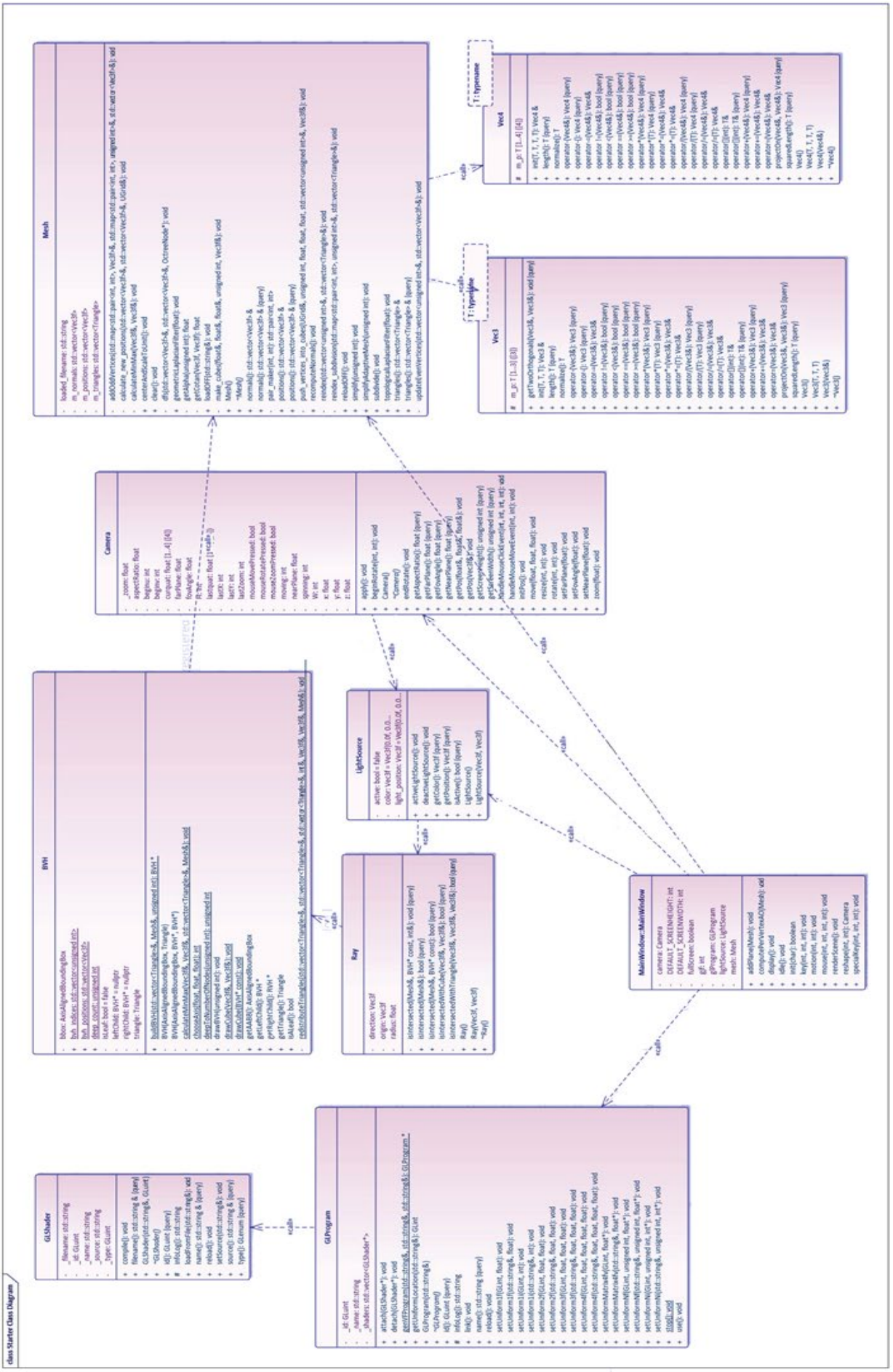


Рисунок 1.16. Діаграма класів застосунку

1.6 Програмування і опис класів застосунку

1.6.1 Створення і опис класу Mesh

Клас Mesh є ключовим компонентом для зберігання та обробки тривимірних моделей у додатку. Він відповідає за збереження координат вершин, побудову полігонів, а також за виконання операцій трансформацій і редагування моделей.

Основні функції класу Mesh:

1. Збереження та обробка вершин: Клас Mesh зберігає координати вершин тривимірної моделі в тривимірному просторі. Для роботи з вершинами використовується допоміжний клас Vec3, який дозволяє виконувати основні векторні операції, такі як додавання, віднімання, масштабування та нормалізація. Окрім цього, Vec3 реалізує перетворення між полярною та декартовою системами координат, що дозволяє легко маніпулювати положенням вершин у просторі.

Для перетворення з полярної в декартову систему координат використовується наступна формула (1.3). Вектори в декартовій і полярній системах координат описуються наступним кодом:

```
class Vec3 {
public:
    float x, y, z;

    Vec3(float x, float y, float z) : x(x), y(y), z(z) {}

    Vec3 operator+(const Vec3& other) const {
        return Vec3(x + other.x, y + other.y, z + other.z);
    }

    Vec3 operator*(float scalar) const {
        return Vec3(x * scalar, y * scalar, z * scalar);
    }

    // Перетворення з полярної в декартову систему
    static Vec3 polarToCartesian(float r, float theta, float phi) {
        float x = r * sin(theta) * cos(phi);
        float y = r * sin(theta) * sin(phi);
        float z = r * cos(theta);
        return Vec3(x, y, z);
    }
};
```

2. Створення полігонів: Після зчитування координат вершин, клас Mesh формує грані моделей шляхом з'єднання відповідних вершин. Для побудови тривимірних об'єктів використовується пошук у глибину (DFS – Depth First Search), який дозволяє ефективно знаходити всі можливі комбінації вершин для побудови трикутників або інших полігонів. Алгоритм пошуку у глибину для створення полігонів реалізується наступним кодом:

```
class Mesh {
public:
    std::vector<Vec3> vertices;
    std::vector<std::vector<int>> faces;  // Зберігаємо індекси вершин для
    // кожної грані
    // Метод для пошуку комбінацій вершин (DFS)
    void createFaces() {
        std::vector<bool> visited(vertices.size(), false);
        for (int i = 0; i < vertices.size(); ++i) {
            if (!visited[i]) {
                dfs(i, visited);
            }
        }
    }
private:
    void dfs(int v, std::vector<bool>& visited) {
        visited[v] = true;
        for (int neighbor : getNeighbors(v)) {
            if (!visited[neighbor]) {
                faces.push_back({v, neighbor});
                dfs(neighbor, visited);
            }
        }
    }
    std::vector<int> getNeighbors(int v) {
        // Повертає сусідів вершини v
        return {}; // Реалізація залежить від топології моделі
    }
};
```

3. Трансформації моделей: Клас Mesh дозволяє виконувати базові трансформації над моделями: масштабування, обертання та переміщення (трансляцію). Ці операції дозволяють змінювати розміри, орієнтацію та положення моделей у просторі.

Для масштабування кожної вершини використовується формула:

$$v' = v \cdot s \tag{1.16}$$

де v — координати вершини, s — коефіцієнт масштабування.

Трансформація моделі, а саме масштабування та трансляція реалізовані наступним кодом:

```
class Mesh {
public:
    void scale(float factor) {
        for (Vec3& vertex : vertices) {
            vertex = vertex * factor;
        }
    }

    void translate(const Vec3& offset) {
        for (Vec3& vertex : vertices) {
            vertex = vertex + offset;
        }
    }
};
```

4. Метод subdivision() — розбиття на підрозділи: Один з важливих методів класу Mesh — це subdivision(), який реалізує алгоритм розбиття на підрозділи. Цей метод дозволяє збільшити кількість контрольних точок на поверхні об'єкта, не змінюючи її загальної форми. Алгоритм базується на формулі для розбиття країв:

$$v_{new} = \frac{v_1 + v_2}{2} \tag{1.17}$$

де v_1 та v_2 — це вершини, що утворюють край, а v_{new} — нова вершина, яка утворюється між ними.

Процес розбиття поверхні на підрозділи реалізовано таким чином:

```
class Mesh {
public:
    void subdivision() {
        std::vector<Vec3> newVertices;
        for (const Vec3& vertex : vertices) {
            // Додаємо нові точки між існуючими вершинами
            newVertices.push_back(vertex); // Початкова вершина
            // Створюємо нові точки для розбиття
```

```
            for (const Vec3& neighbor : getNeighbors(vertex)) {
                newVertices.push_back((vertex + neighbor) * 0.5f);
            }
        }
        vertices = newVertices; // Оновлюємо сітку
    }
};
```

5. Алгоритм освітлення Кука-Торренса: Метод torrance(), реалізований у класі Mesh, відповідає за освітлення моделі за моделлю Кука-Торренса. Цей метод розраховує кількість відбитого світла від поверхні об'єкта з урахуванням шорсткості та інших оптичних характеристик матеріалу. Основна формула для моделі Кука-Торренса виглядає так:

$$L_o = \frac{F \cdot G \cdot D}{4 \cdot (N \cdot L) \cdot (V \cdot H)} \qquad (1.18)$$

де:

- L_o — кількість відбитого світла;
- F — коефіцієнт відбиття за законом Френеля;
- G — геометричний фактор, що враховує мікроструктуру поверхні;
- D — функція розподілу нормалей (roughness distribution);
- $N \cdot L$ — косинус кута між нормаллю N та напрямком на джерело світла L;
- $V \cdot H$ — косинус кута між напрямком на камеру V та напрямком напіввектора H (середнє між напрямком на світло та на камеру).

Для обчислення коефіцієнта відбиття за законом Френеля використовується апроксимація Шліка:

$$F = F_0 + (1 - F_0) \cdot (1 - V \cdot H)^5 \qquad (1.19)$$

де F_0 — коефіцієнт відбиття для перпендикулярного кута падіння (зазвичай це значення при нормальних умовах для матеріалу).

```
float schlickApproximation(float VdotH, float F0) {
    return F0 + (1.0f - F0) * pow(1.0f - VdotH, 5.0f);
}
```

Геометричний фактор враховує те, як мікроскопічна поверхня закриває себе від падаючого світла та відбиваючого напрямку. Для його обчислення можна

використовувати модель Кука-Торренса, яка враховує затінення та самозатінення:

$$G = G_1(N,V) \cdot G_1(N,L) \tag{1.20}$$

Функція G_1 може бути обчислена за допомогою методу Шліка:

$$G_1(N,X) = \frac{2 \cdot (N \cdot X)}{(N \cdot X) + \sqrt{\alpha^2 + (1-\alpha^2) \cdot (N \cdot X)^2}} \tag{1.21}$$

де α — параметр шорсткості поверхні.

```
float geometry(float NdotV, float NdotL, float alpha) {
   float G1_V = 2.0f * NdotV /
(NdotV + sqrt(alpha * alpha + (1.0f - alpha * alpha) * NdotV * NdotV));
   float G1_L = 2.0f * NdotL /
(NdotL + sqrt(alpha * alpha + (1.0f - alpha * alpha) * NdotL * NdotL));
   return G1_V * G1_L;
}
```

Функція шорсткості за Бекманом описує, як орієнтація мікроскопічних поверхонь впливає на розподіл відбиття. Вона визначає ймовірність того, що поверхня з нормаллю N буде відображати світло в напрямку H:

$$DD = \frac{e^{\frac{-(\tan^2 \theta_H)}{\alpha^2}}}{\pi \cdot \alpha^2 \cdot (N \cdot H)^4} \tag{1.22}$$

Де α — параметр, що відповідає за шорсткість поверхні, θ_H — кут між нормаллю та напіввектором H.

```
float beckmannDistribution(float NdotH, float alpha) {
   float tanThetaH2 = (1.0f - NdotH * NdotH) / (NdotH * NdotH);
   return exp(-tanThetaH2 / (alpha * alpha)) / (3.14159f * alpha * alpha *
pow(NdotH, 4.0f));
}
```

Освітлення моделі за допомогою алгоритму Кука-Торренса: вплив шорсткості, розподілу мікронормалей та Френеля на відбите світло:

```
class Mesh {
public:
   float torrance(const Vec3& normal, const Vec3& lightPos, const Vec3&
viewPos, float F0, float alpha) {
      Vec3 h = (lightPos + viewPos).normalize(); // Напрямок напіввектора
      float NdotL = std::max(0.0f, normal.dot(lightPos.normalize()));
      float NdotH = std::max(0.0f, normal.dot(h));
      float VdotH = std::max(0.0f, viewPos.dot(h));
```

```
        // Обчислюємо компоненти Френеля, геометрії та шорсткості
        float F = schlickApproximation(VdotH, F0);
        float G = geometry(NdotL, normal.dot(viewPos.normalize()), alpha);
        float D = beckmannDistribution(NdotH, alpha);
        return (F * G * D) / (4.0f * NdotL * normal.dot(viewPos.normalize()));
    }
};
```

У підсумку, комбінуючи всі ці фактори, можна обчислити кількість світла, що відбивається від поверхні з урахуванням її шорсткості та геометрії.

Щоб не вводити додатковий параметр і не ускладнювати процес обчислення ступеня, можна використовувати спрощену формулу, яка також дає добрий результат:

$$F = \frac{1}{1 + (\vec{V} \bullet \vec{N})} \tag{1.23}$$

Далі описано програмування класів GLShader, GLProgram, GLError, створених для роботи з фреймворком OpenGL.

1.6.2 Створення і опис класу GLShader

Клас GLShader відіграє важливу роль у роботі з графічним конвеєром, оскільки шейдери є основними елементами, що визначають, як обробляються та відображаються 3D-моделі у OpenGL. Шейдери дозволяють програмісту визначати, як саме виглядатиме графічний об’єкт, що включає тіні, освітлення, текстури та інші ефекти. Клас GLShader забезпечує необхідний інтерфейс для роботи з шейдерами у програмі, надаючи механізми для завантаження, компіляції та використання шейдерів у графічному додатку.

Клас GLShader створений для управління шейдерами в OpenGL, зокрема вершинними шейдерами. Основна його функція полягає у завантаженні вихідного коду шейдера, його компіляції та забезпеченні правильного виконання на графічному процесорі.

Структура класу:

1. Завантаження вихідного коду шейдера: Клас GLShader дозволяє завантажувати текст вихідного коду шейдера з файлу або з рядка у програмі. Вихідний код зберігається у змінній _source, яка пізніше передається в OpenGL

для компіляції.

void GLShader::setSource(const std::string & source) { _source = source; }

2. Компіляція шейдера: Після завантаження вихідного коду клас дозволяє компілювати його за допомогою функцій OpenGL. Компіляція перевіряє синтаксичну коректність коду та готує шейдер до виконання на GPU.

```
void GLShader::compile() {
   // Завантаження коду шейдера в OpenGL
   glShaderSource(shaderID, 1, &_source, NULL);
   glCompileShader(shaderID);
   // Перевірка успішності компіляції
   GLint success;
   glGetShaderiv(shaderID, GL_COMPILE_STATUS, &success);
   if (!success) {
      // Обробка помилки компіляції
      GLchar infoLog[512];
      glGetShaderInfoLog(shaderID, 512, NULL, infoLog);
      std::cerr << "Помилка компіляції шейдера: " << infoLog << std::endl;
   }
}
```

3. Використання шейдера: Після успішної компіляції шейдер може бути використаний у графічній програмі. Для цього він повинен бути прив'язаний до відповідної програми OpenGL, що виконує рендеринг об'єктів.

void GLShader::use() { glUseProgram(programID); }

4. Основні методи:

- id(): отримує ідентифікатор шейдера, що був згенерований OpenGL;
- type(): визначає тип шейдера (вершинний, фрагментний чи геометричний);
- source(): отримує вихідний код шейдера;
- setSource(const std::string & source): задає вихідний код шейдера.

```
GLuint GLShader::id() const {
   return shaderID;
}
std::string GLShader::source() const {
   return _source;
}
```

1.6.3 Створення і опис класу GLProgram

Клас GLProgram виконує ключову функцію у графічному додатку, об'єднуючи шейдери та керуючи процесом їхнього використання для рендерингу 3D-моделей. Цей клас безпосередньо взаємодіє з OpenGL, керуючи виконанням графічних операцій, і є основним інтерфейсом для програміста під час роботи з графічними об'єктами, використовуючи шейдери, текстури та буфери.

Головне завдання GLProgram — створення програми OpenGL, що поєднує кілька шейдерів, компілює їх та виконує для відображення моделей на екрані. Клас забезпечує процес ініціалізації програми, її налаштування та малювання об'єктів за допомогою класу Mesh, який містить інформацію про вершини, текстури та інші атрибути моделі. Важливим аспектом є взаємодія з OpenGL для управління рендерингом на екрані, де відображаються 3D-моделі.

Клас GLProgram включає кілька етапів роботи з OpenGL:

1. Створення полотна: на цьому етапі створюється контекст для відображення сцени. Це полотно (буфер кадру) буде використовуватися для відображення моделей на екрані. У OpenGL це відбувається шляхом створення і налаштування буферів, у яких буде рендеритися зображення.

```
void GLProgram::initialize() {
    // Створення буферів кадру та налаштування полотна
    glViewport(0, 0, windowWidth, windowHeight);
    glClearColor(0.1f, 0.1f, 0.1f, 1.0f);  // Колір фону
    glEnable(GL_DEPTH_TEST);  // Активування буфера глибини
}
```

2. Ініціалізація шейдерної програми: клас GLProgram поєднує вершинний, фрагментний та, за потреби, геометричний шейдери у єдину програму. Ця програма компілюється і зв'язується з OpenGL для подальшого використання під час рендерингу.

```
void GLProgram::attachShader(const GLShader& shader) {
    glAttachShader(programID, shader.id());  // Прив’язка шейдера до програми
}
void GLProgram::link() {
    glLinkProgram(programID);  // Лінкування шейдерів у програму
    GLint success;
    glGetProgramiv(programID, GL_LINK_STATUS, &success);
```

```
    if (!success) {
        GLchar infoLog[512];
        glGetProgramInfoLog(programID, 512, NULL, infoLog);
        std::cerr << "Помилка лінкування шейдерної програми: " << infoLog <<
std::endl;
    }
}
```

3. Малювання моделі: після успішного лінкування шейдерів і створення програми, клас GLProgram відповідає за рендеринг моделі. Він взаємодіє з класом Mesh, який надає необхідні дані про вершини, текстури та інші атрибути для рендерингу.

```
void GLProgram::render(const Mesh& mesh) {
    // Використання шейдерної програми
    glUseProgram(programID);
        // Встановлення атрибутів та буферів для малювання
    mesh.bind();
        // Малювання моделі
 glDrawElements(GL_TRIANGLES, mesh.indexCount(), GL_UNSIGNED_INT, 0);
}
```

Основні методи:

- initialize(): ініціалізує контекст OpenGL для відображення;
- attachShader(const GLShader& shader): прив’язує шейдер до програми;
- link(): лінкує шейдери у програму OpenGL;
- render(const Mesh& mesh): виконує рендеринг моделі, використовуючи прив’язані шейдери.

1.6.4 Створення і опис класу GLError

Клас GLError створений для забезпечення відстеження помилок під час взаємодії з OpenGL. Оскільки OpenGL є низькорівневою бібліотекою для рендерингу, будь-які помилки, які виникають під час виконання графічних операцій, можуть залишитися непоміченими без належного моніторингу. Саме для цього і був розроблений клас GLError, який дозволяє програмісту вчасно виявляти проблеми та надавати відповідну інформацію у лог-файл або на консоль.

Клас GLError інтегрований в інші класи, такі як GLProgram та GLShader, для того, щоб перевіряти наявність помилок після кожної взаємодії з OpenGL. Він

відстежує стан помилок, які можуть виникати під час таких операцій, як компіляція шейдерів, лінкування програм або рендеринг. У разі виявлення помилки, клас реєструє повідомлення про її місце виникнення, що значно полегшує процес налагодження додатка.

Основна функція класу полягає у перевірці результату останньої операції OpenGL та виведенні повідомлення про помилку, якщо така виникла. Для цього використовується функція glGetError(), яка повертає код помилки. Клас GLError обробляє цей код і виводить детальну інформацію про помилку.

Основні методи класу GLError:

checkError() — перевіряє наявність помилки після виконання будь-якої OpenGL-операції. У разі виявлення помилки, вона виводиться до лог-файлу або консолі.

```
void GLError::checkError(const std::string& function, const std::string& file,
int line) {
    GLenum error = glGetError();
    if (error != GL_NO_ERROR) {
        std::cerr << "OpenGL Помилка: " << getErrorString(error)
                  << " у функції: " << function
                  << " у файлі: " << file
                  << " на лінії: " << line << std::endl;
    }
}
```

getErrorString() — допоміжний метод для отримання текстового опису помилки на основі її коду. Ця функція перетворює код помилки, отриманий від glGetError(), на зрозумілий для програміста рядок.

```
std::string GLError::getErrorString(GLenum error) {
    switch (error) {
        case GL_INVALID_ENUM:      return "GL_INVALID_ENUM";
        case GL_INVALID_VALUE:     return "GL_INVALID_VALUE";
        case GL_INVALID_OPERATION: return "GL_INVALID_OPERATION";
        case GL_STACK_OVERFLOW:    return "GL_STACK_OVERFLOW";
        case GL_STACK_UNDERFLOW:   return "GL_STACK_UNDERFLOW";
        case GL_OUT_OF_MEMORY:     return "GL_OUT_OF_MEMORY";
        default:                   return "Unknown error";
    }
}
```

invoke — макрос, який спрощує використання методу checkError у коді. Він

автоматично додає інформацію про місце виклику (файл і рядок коду), що робить процес виявлення помилок ефективнішим.

Клас GLError використовується у ключових місцях коду класів GLProgram та GLShader для відстеження стану помилок. Наприклад, після компіляції шейдера або лінкування програми, викликається перевірка помилок:

```
void GLShader::compile() {
    glCompileShader(shaderID);
    GLError::checkError("glCompileShader", __FILE__, __LINE__);
}

void GLProgram::link() {
    glLinkProgram(programID);
    GLError::checkError("glLinkProgram", __FILE__, __LINE__);
}
```

1.6.5 Створення і опис класу Camera

Клас Camera відповідає за управління положенням і орієнтацією камери в просторі 3D-сцени. Його основна мета — забезпечити коректне відображення моделей на екрані, а також забезпечити можливість взаємодії з об'єктами шляхом зміни положення камери, її обертання, масштабування та інших параметрів. У даному класі застосовуються кватерніони для обчислення обертання камери, що дозволяє отримати гладку і точну анімацію.

Клас Camera реалізує основні функції для роботи з тривимірною камерою, такі як обертання, переміщення та зміна масштабу. Основна складність полягає в тому, щоб коректно працювати з обертанням об'єктів у просторі. Існує кілька підходів до реалізації цієї функції, зокрема використання матриць обертання та кутів Ейлера. Проте для гладкої інтерполяції між різними положеннями камери найкраще використовувати кватерніони, оскільки вони дозволяють уникнути проблеми "гімбального замикання" (gimbal lock) і забезпечують плавне обертання.

Клас Camera працює з такими параметрами, як поле зору (FOV), співвідношення сторін екрану (aspect ratio), положення камери в просторі та її орієнтація. Всі ці параметри визначають, як саме об'єкти сцени будуть відображатися на екрані.

Основні методи класу Camera:

1. getFovAngle() — отримує кут огляду камери (FOV), який визначає, яку частину сцени видно у вікні відображення.

float Camera::getFovAngle() const { return fovAngle; }

2. setFovAngle(float angle) — встановлює кут огляду камери. Це дозволяє змінювати масштабування сцени, віддаляючи або наближаючи камеру до об'єкта.

void Camera::setFovAngle(float angle) { fovAngle = angle; }

3. getAspectRatio() — отримує співвідношення сторін екрану, що необхідно для коректного рендерингу сцени без викривлень.

float Camera::getAspectRatio() const { return aspectRatio; }

4. getScreenWidth() і getScreenHeight() — отримують ширину та висоту вікна відображення, що використовується для обчислення проекційної матриці.

```
int Camera::getScreenWidth() const { return screenWidth; }
int Camera::getScreenHeight() const { return screenHeight; }
```

5. rotate() — реалізує обертання камери за допомогою кватерніонів. Це дозволяє забезпечити плавне і безперервне обертання камери навколо об'єкта.

```
void Camera::rotate(float x, float y, float z) {
    Quaternion rotation = Quaternion::fromEuler(x, y, z);
    orientation = rotation * orientation;
}
```

6. move() — переміщує камеру в просторі за вказаними координатами. Це дозволяє користувачеві наблизити або віддалити камеру від об'єкта, а також переміщати її вліво або вправо.

```
void Camera::move(const Vec3& direction, float distance) {
    position += direction * distance;
}
```

7. zoom(float factor) — змінює масштабування камери. Це впливає на кут огляду (FOV), дозволяючи наблизити або віддалити видиму частину сцени.

*void Camera::zoom(float factor) { fovAngle *= factor; }*

Однією з ключових особливостей класу Camera є використання кватерніонів для обертання. Кватерніони дозволяють уникнути проблем, які можуть виникнути при використанні матриць обертання або кутів Ейлера, зокрема проблему

гімбального замикання. Кватерніони забезпечують гладку інтерполяцію обертання та зберігають правильну орієнтацію об'єктів під час обертання.

При обертанні камери кватерніон представляється як четвірка чисел (x, y, z, w), яка кодує вектор осі обертання та кут обертання. Це дозволяє ефективно обчислювати обертання без втрати точності.

Псевдокод для обчислення компонентів кватерніона:

```
// Кут оберту RotationAngle заданий у радіанах
x = RotationAxis.x * sin(RotationAngle / 2);
y = RotationAxis.y * sin(RotationAngle / 2);
z = RotationAxis.z * sin(RotationAngle / 2);
w = cos(RotationAngle / 2);
```

У кватерніоні параметри одиничного вектора (ось обертання) множаться на синус половини кута обертання, а четвертий компонент — на косинус половини кута обертання (табл.1.5).

Таблиця 1.5 Значення кватерніонів для різних обертів

w	*x*	*y*	*z*	*Обертання*
1	0	0	0	-
0	1	0	0	180˚ навколо осі X
√0.5	√0.5	0	0	90˚ навколо осі X
√0.5	-√0.5	0	0	-90˚ навколо осі X

1.6.6 Створення і опис класу Vec4

Клас Vec4 був створений для представлення 4-вимірних векторів, що використовуються у роботі з кватерніонами. Кватерніон можна зручно розглядати як 4D вектор, що дозволяє застосовувати до нього ті ж операції, які використовуються над векторами. Клас Vec4 забезпечує можливість виконувати додавання, віднімання, множення векторів, а також інші операції, специфічні для кватерніонів. Основні операції з кватерніонами:

1. Множення кватерніонів. При роботі з кватерніонами часто виникає необхідність їх множення один на одного. Це важлива операція, оскільки множення двох кватерніонів дає можливість поєднати два обертання: перше обертання буде обернене на друге. Важливо пам'ятати, що множення кватерніонів не є комутативним, тобто порядок операндів має значення. Наприклад:

$$q \times q' \neq q' \times q \quad (1.24)$$

Це означає, що порядок виконання операцій має вирішальний вплив на результат обертання.

2. Додавання кватерніонів. Операцію додавання можна описати як "суміш" обертань, тобто результат додавання обертів знаходиться між двома початковими обертаннями q та q'.

3. Множення на скаляр. Множення кватерніона на скаляр не змінює обертання, якщо скаляр не дорівнює нулю. Кватерніон, помножений на 0, представляє "невизначене" обертання.

```
// Псевдокод для додавання двох кватерніонів q і q'
Vec4 q_sum;
q_sum.w = q.w + q'.w;
q_sum.x = q.x + q'.x;
q_sum.y = q.y + q'.y;
q_sum.z = q.z + q'.z;
```

Клас Vec4 також дозволяє виконувати операції обчислення норми та модуля кватерніона:

- Норма (norm) — це міра квадрата довжини кватерніона:

$$\text{norm}(q) = q.w^2 + q.x^2 + q.y^2 + q.z^2 \quad (1.25)$$

- Модуль (magnitude) або довжина кватерніона — це квадратний корінь з норми:

$$\text{magnitude}(q) = \sqrt{\text{norm}(q)} \quad (1.26)$$

Нормалізація кватерніона полягає у приведенні його довжини до одиниці. Це важливо для забезпечення коректності обертань, так як нормалізований кватерніон зберігає свої властивості під час множення.

Клас Vec4 надає такі методи для роботи з кватерніонами:

- init(T x, T y, T z, T w) — ініціалізація параметрів вектора (кватерніона).
- squaredLength() — отримання квадрата довжини вектора (норми).
- length() — отримання довжини вектора (модуля).
- projectOn(const Vec4 & N, const Vec4 & P) — проекція вектора на інший вектор.

1.6.7 Створення і опис класу Ray

Клас Ray реалізує алгоритм кидання променів (Ray Tracing), який використовується для підсвітки та взаємодії з моделлю під час редагування. Основною метою класу є визначення точок перетину променів з об'єктами сцени, що дозволяє точно визначити, які елементи мають бути підсвічені або виділені для подальшої обробки.

Кидання променів є ефективною технологією рендеринга тривимірної графіки, що базується на моделюванні шляху променя світла від камери до об'єкта. Промені випускаються від камери через кожен піксель площини проєкції і перевіряються на перетин з об'єктами сцени. Основні етапи цього процесу:

1. Генерація первинних променів: Від камери до кожного пікселя площини проєкції проводяться вектори, які називаються первинними променями.

2. Перетин з об'єктами: Для кожного променя алгоритм знаходить найближчий об'єкт сцени, з яким цей промінь перетинається. Якщо такий об'єкт знайдено, визначається точка перетину.

3. Розрахунок кольору пікселя: Кожен піксель площини проєкції зафарбовується в колір, який відповідає точці перетину променя з об'єктом. Повторивши цей процес для кожного пікселя, отримується зображення тривимірної сцени.

Клас Ray використовує такі ключові компоненти:

- Початкова точка променя — це точка, з якої починається промінь, зазвичай вона відповідає позиції камери.
- Напрямок променя — це вектор, який визначає напрямок руху променя від камери до площини проєкції.
- Перетин з об'єктом — алгоритм обчислює, чи перетинає промінь якийсь об'єкт на сцені, і якщо так, визначає точку цього перетину.

Оскільки кожен промінь, що випускається з камери, є незалежним від інших, їх можна обробляти паралельно. Це робить алгоритм кидання променів надзвичайно ефективним для використання у графічних застосунках, де важлива продуктивність.

Однією з головних переваг кидання променів є те, що цей метод не містить обмежень на геометрію об'єктів, що можуть бути представлені у сцені. Це дозволяє використовувати широкий набір геометричних примітивів, таких як площини, сфери, циліндри тощо.

Ray Tracing також дозволяє створювати реалістичні ефекти освітлення та відображення, що практично не відрізняються від реального. Саме тому цей алгоритм є популярним вибором для високоякісної тривимірної графіки та симуляції світла.

Клас Ray надає такі методи:

- init(Vec3 origin, Vec3 direction) — ініціалізація променя з початковою точкою (позицією камери) та напрямком.
- intersect(Object obj) — визначення, чи перетинає промінь даний об'єкт.
- getIntersectionPoint() — отримання точки перетину променя з об'єктом, якщо така точка існує.

1.6.8 Створення і опис класу BVH та алгоритму побудови ієрархії обмежуючих об'ємів

Клас BVH (Bounding Volume Hierarchy) створений для ефективної роботи з класом Ray з метою оптимізації обчислень перетину променів з геометрією сцени. Ієрархія обмежуючих об'ємів (BVH) забезпечує швидкий доступ до геометричних об'єктів і дозволяє ефективно обробляти сцени з великою кількістю трикутників.

BVH використовує структуру дерева, де кожен вузол містить обмежуючий об'єм, що охоплює всі свої підвузли або "дітей". У нашому випадку використовується AABB (Axis-Aligned Bounding Box) — бінарне дерево, де кожен вузол є паралелепіпедом, сторони якого вирівняні вздовж осей координат. Це дозволяє прискорити операції перетину променів з об'єктами сцени, значно скоротивши кількість необхідних обчислень. Алгоритм побудови дерева AABB (рис.1.17) заснований на розбитті набору трикутників на підмножини та рекурсивному створенні вузлів дерева. Основні кроки цього процесу:

1. Ініціалізація вузла: Створюється новий вузол U, що буде охоплювати множину трикутників T;

2. Обчислення AABB: Для множини трикутників T обчислюється обмежуючий паралелепіпед AABB, який запам'ятовується у вузлі U;

3. Розбиття трикутників: Обирається площина розділення, яка проходить через середину найбільшої сторони AABB. Трикутники розбиваються на дві підмножини — Tpositive і Tnegative — за положенням їхніх центрів відносно площини;

4. Рекурсивна побудова дерева: Для кожної з отриманих підмножин трикутників рекурсивно повторюється процес побудови дерева;

5. Призначення листів: Якщо у множині залишився лише один трикутник, він призначається листом дерева, і рекурсія завершується.

Основною перевагою AABB дерев є їх швидкість та економія пам'яті, оскільки паралельність боксу до координатних осей дозволяє спростити обчислення. Метод перевірки перетину променів з об'ємом AABB значно простіший і швидший, ніж для інших типів обмежуючих об'ємів, наприклад, OBB (Oriented Bounding Box), що вимагають обчислення орієнтації.

Однак, через те, що AABB не завжди точно охоплює об'єкти сцени, особливо при роботі зі складними геометричними формами, може виникати більша кількість ітерацій для перевірки перетинів з променями, що зменшує ефективність у порівнянні з OBB. Тим не менше, AABB є оптимальним рішенням у багатьох випадках через свою простоту й ефективність.

Клас BVH надає такі методи для взаємодії зі структурою дерева:

- getLeftChild() — отримання піддерева, що знаходиться ліворуч від поточного вузла;
- getRightChild() — отримання піддерева, що знаходиться праворуч;
- isALeaf() — перевірка, чи є поточний вузол листом дерева (містить трикутник);
- getTriangle() — отримання трикутника, що зберігається у листовому вузлі дерева.

Ці методи дозволяють ефективно працювати з геометрією сцени, забезпечуючи швидкий пошук та обчислення перетинів для рейтрейсингу.

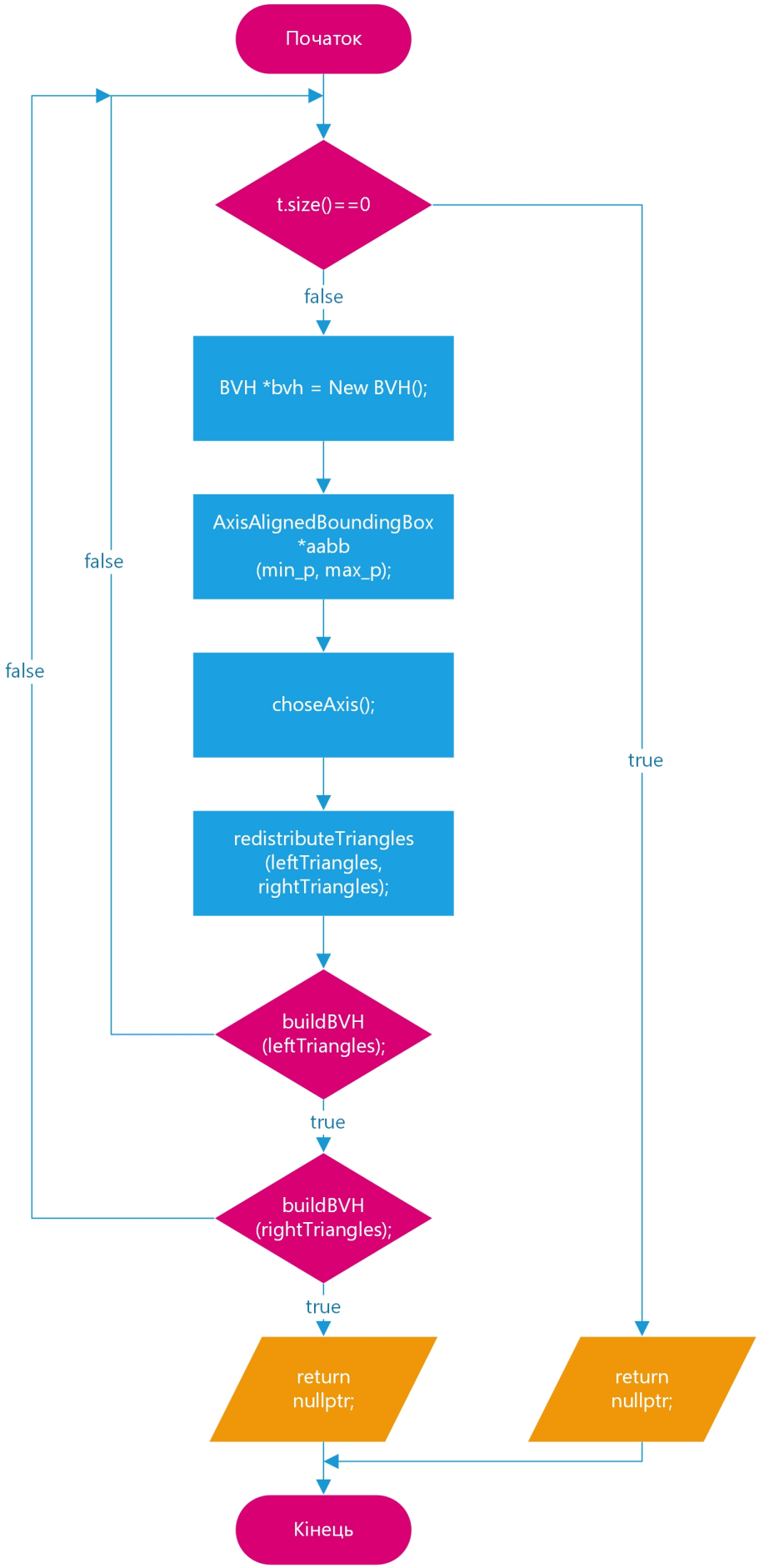


Рисунок 1.17. Блок-схема алгоритму побудови ієрархії обмежуючих об'ємів

1.7 Тестування розробленого застосунку для редагування тривимірних графічних об’єктів

Тестування застосунку для редагування тривимірних моделей було проведено з метою перевірки функціональності, зручності користування та коректності реалізації алгоритмів обробки моделей. Програма була протестована з використанням різних операцій редагування тривимірних об’єктів і налаштувань візуалізації. Текст програми мовою С++, написаний відповідно до створених і описаних у п.1.6 алгоритмів, наведений у Додатку А.

На рис. 1.18 показано головне вікно застосунку, реалізовано за допомогою фреймворку QT. Передбачено кнопку вибору (завантаження з носія) тривимірної моделі у стандартному форматі *.off.

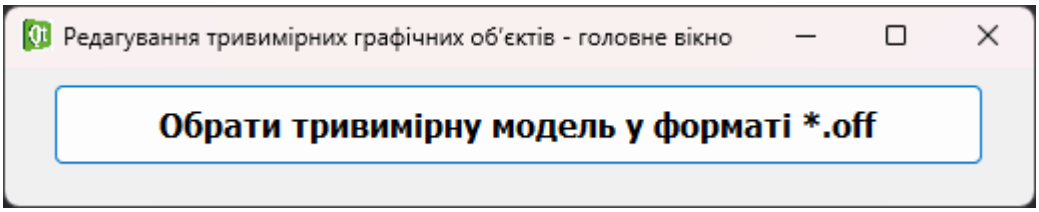


Рисунок 1.18. Головне вікно застосунку

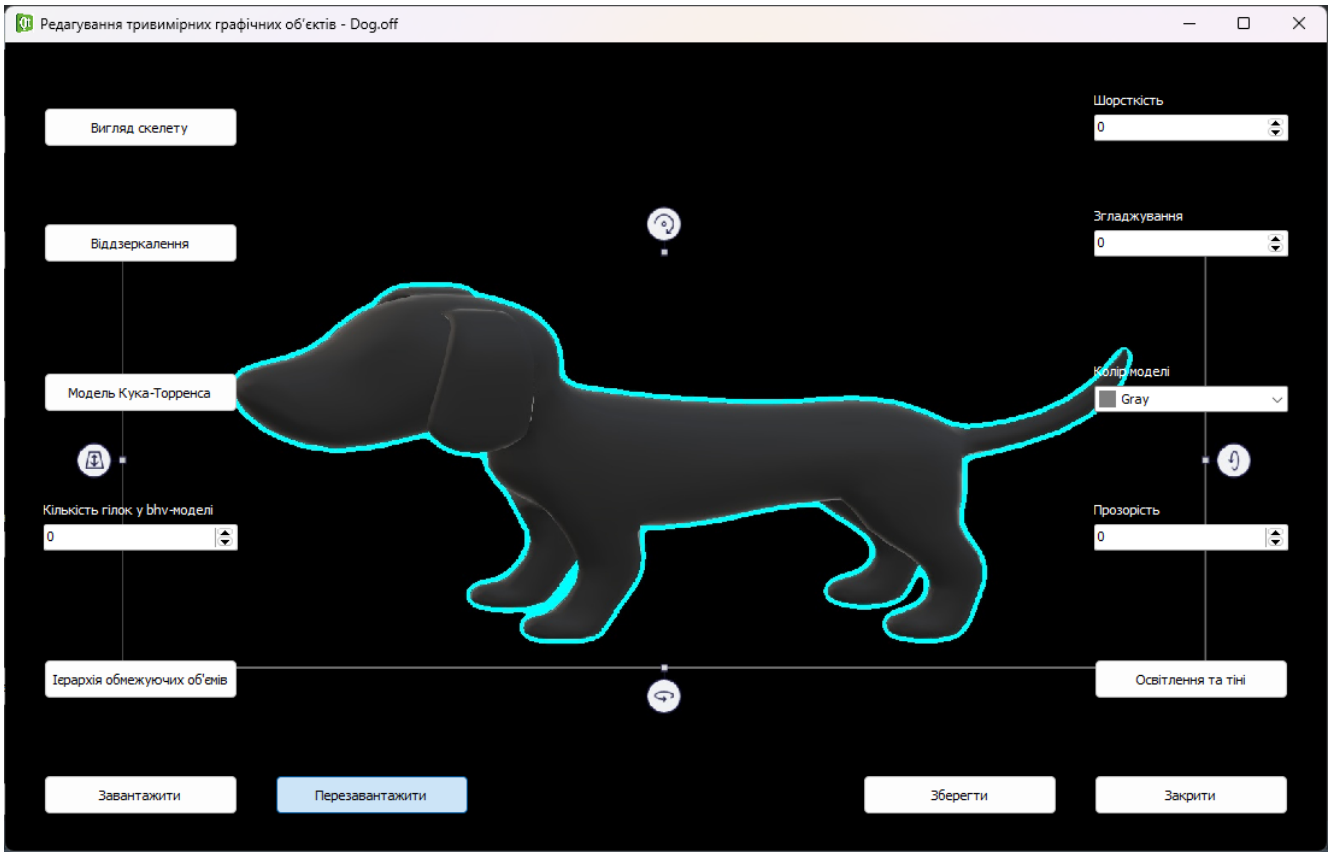


Рисунок 1.19. Вікно застосунку з завантаженою тривимірною моделлю

Після натиснення кнопки “Обрати тривимірну модель у форматі *.off” відкривається вікно програми з тривимірною моделлю (рис.1.19). У центрі екрану розташовано вікно для відображення тривимірної моделі, яка відображається з використанням підсвітки та шейдерів на чорному фоні. Програма дозволяє здійснювати різні маніпуляції з об'єктом за допомогою миші та клавіатури. Зовнішній вигляд моделі на зображенні відповідає базовій моделі з встановленими параметрами шорсткості та згладжування.

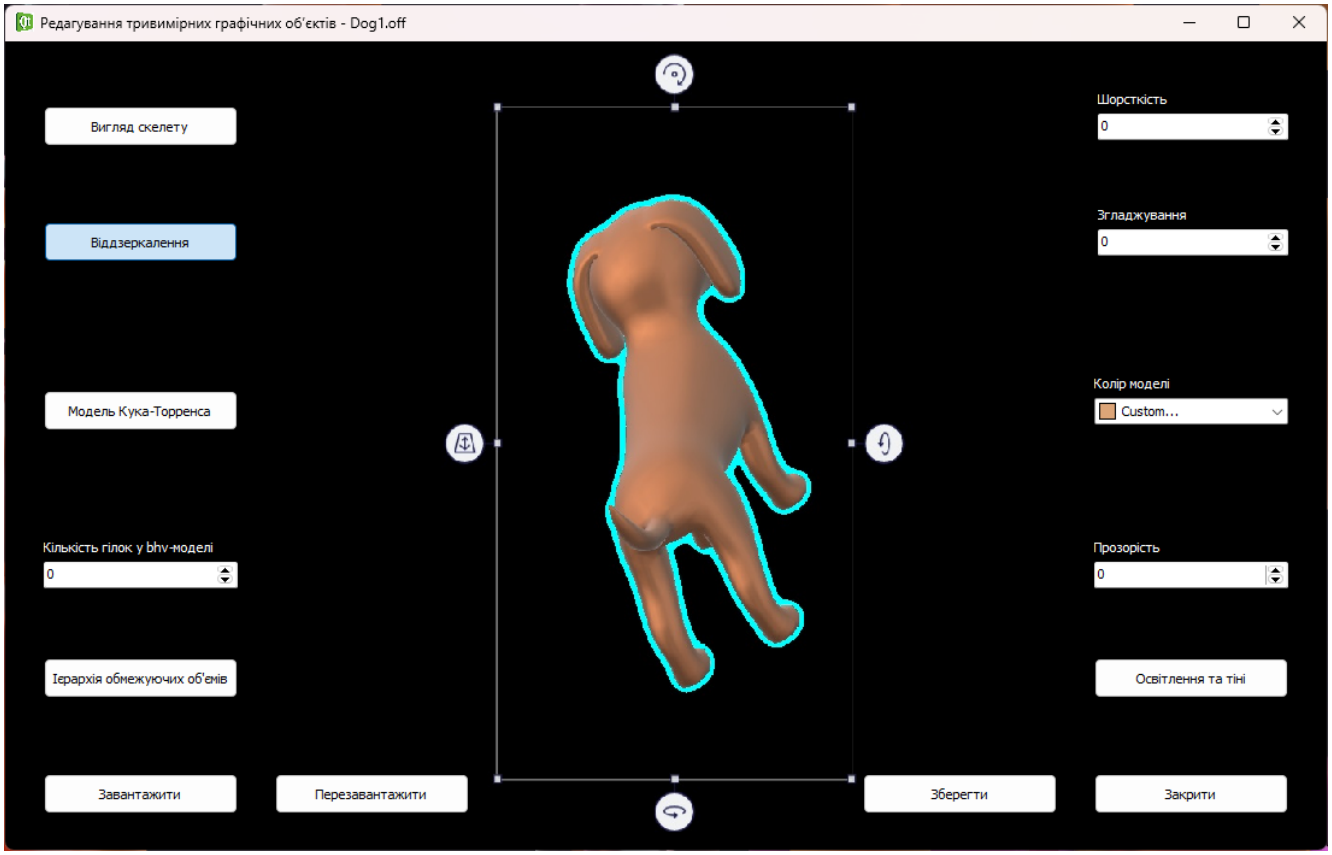


Рисунок 1.20. Режими обертання та віддзеркалення для моделі

Основні елементи керування представлені у вигляді кнопок, розташованих навколо моделі. Ось перелік доступних функцій:

1. Переміщення та обертання моделі:

– Для переміщення моделі використовується миша, відповідні кнопки у інтерфейсі або клавіші зі стрілками. Клавіші W, A, S, D відповідають за обертання моделі навколо осей X і Y (рис.1.20);

– Колесо миші відповідає за масштабування моделі, дозволяючи збільшувати або зменшувати її розмір;

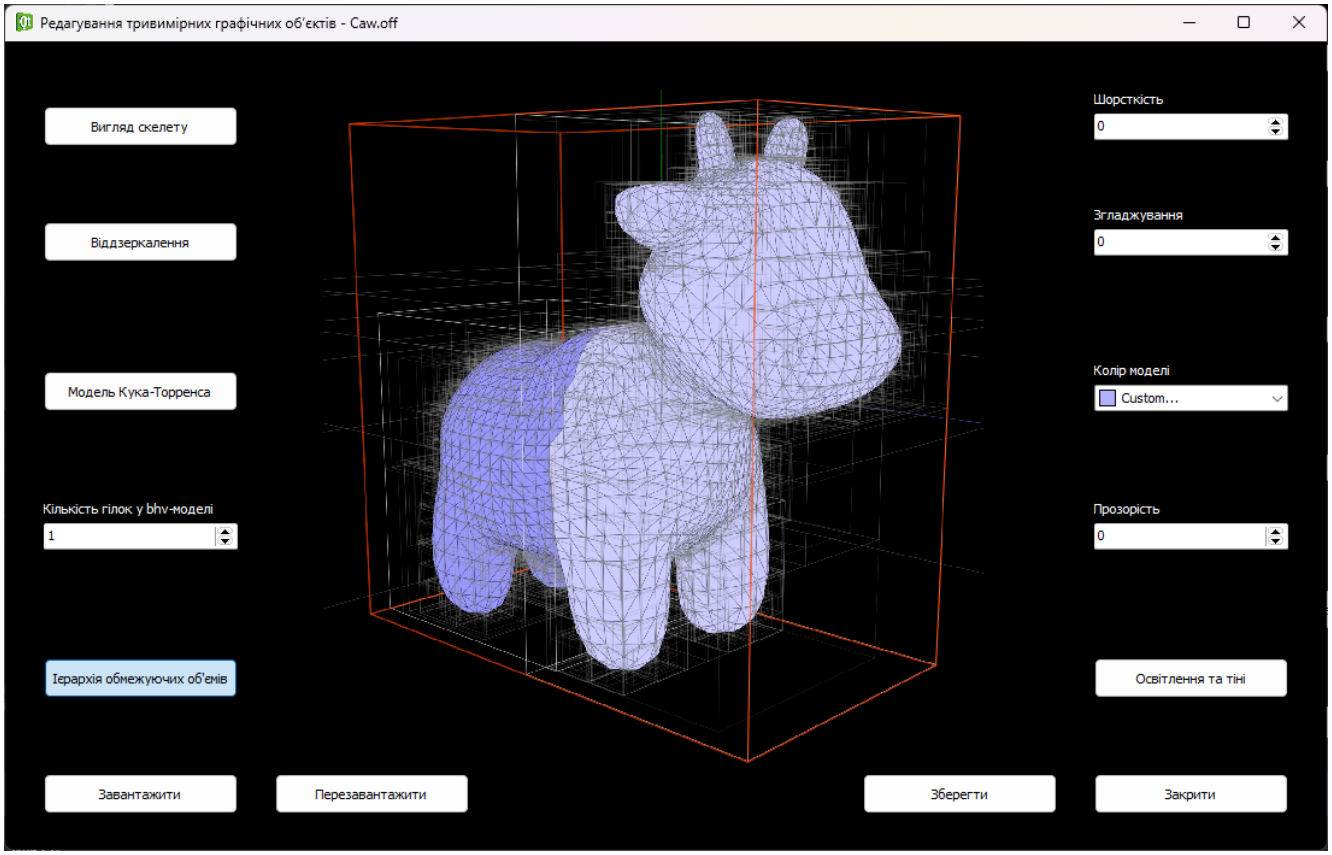


Рисунок 1.21. Режим побудови ієрархії обмежуючих об’ємів з кількістю гілок 1

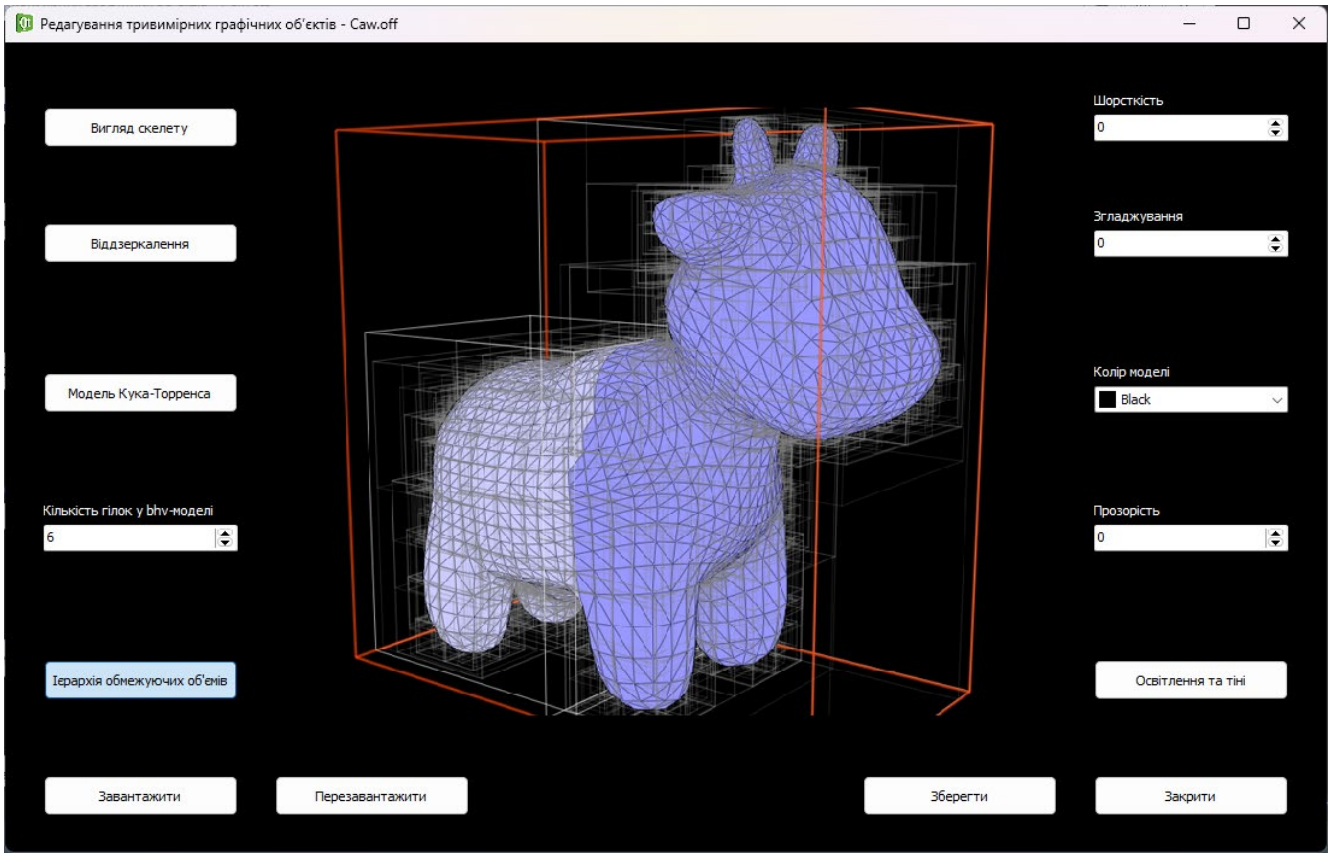


Рисунок 1.22. Режим побудови ієрархії обмежуючих об’ємів з кількістю гілок 6

2. Завантаження і збереження моделей:

– Кнопки "Завантажити" та "Зберегти" дозволяють завантажити нову 3D-модель у форматі *.off, а також зберегти відредаговану модель після застосування змін. Після редагування користувач може зберегти модель у новому форматі, або перезавантажити вихідну модель для подальших змін. Також відповідними кнопками можна перезавантажити файл заново або вийти із застосунку;

3. Відображення структури BVH:

– Натиснувши кнопку "Ієрархія обмежуючих об'ємів", користувач може активувати відображення BVH (Bounding Volume Hierarchy), що дозволяє візуалізувати структуру об'єктів сцени, спрощуючи пошук та вибір окремих елементів для редагування. При цьому за допомогою відповідного поля вводу можна обрати необхідну кількість гілок у BVH-моделі (рис.1.21, 1.22). Цей режим відображає спрощену геометрію об’єкта для швидшої обробки та рендерингу;

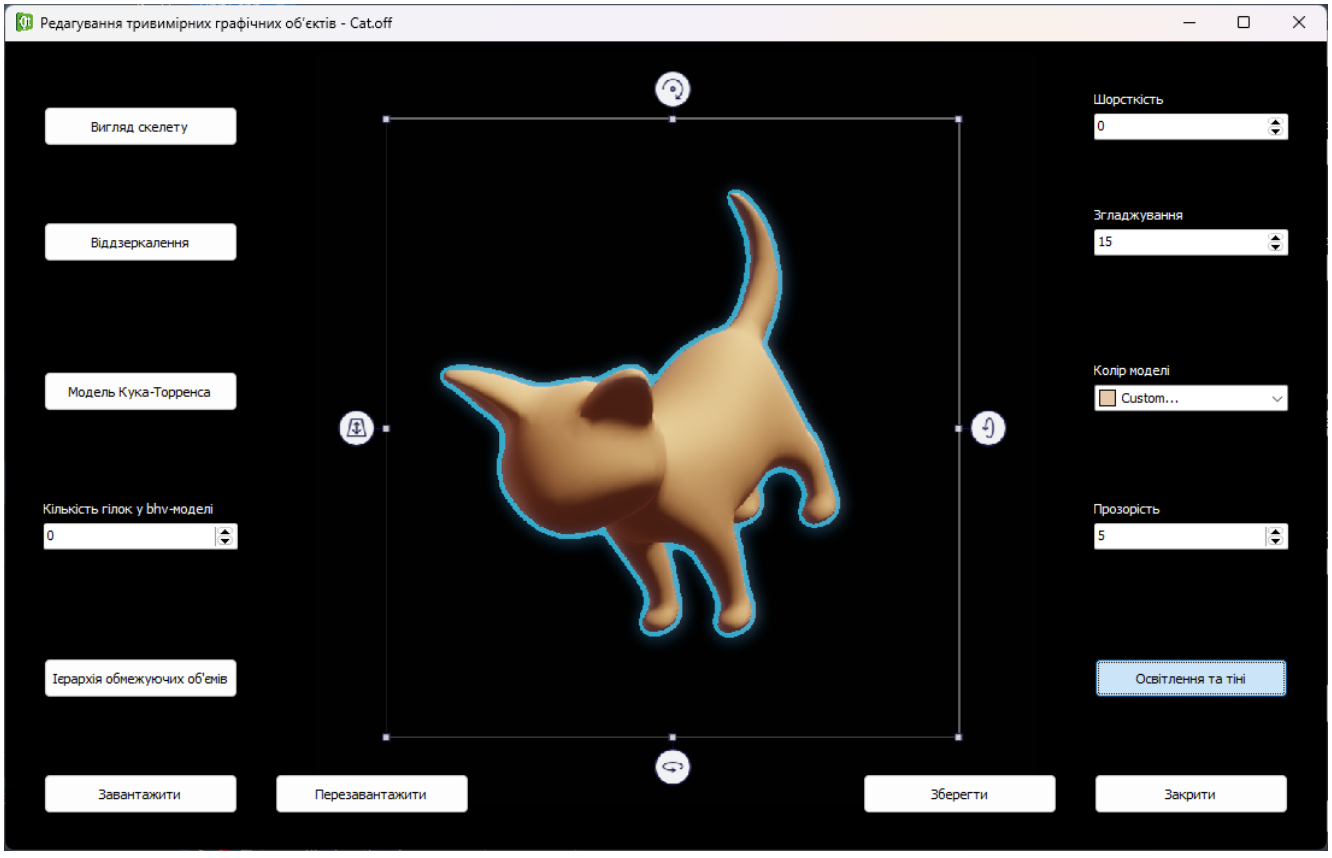


Рисунок 1.23. Режим освітлення та тіні для моделі

4. Шейдери та моделі освітлення:

– Кнопка "Модель Кука-Торренса" активує застосування шейдерів

освітлення за допомогою моделі Кука-Торренса, яка відповідає за реалістичне відображення світла та матеріалів (рис.1.23, 1.24). Це дозволяє користувачу бачити, як модель буде виглядати при різних налаштуваннях освітлення та блиску поверхонь;

5. Згладжування та шорсткість:

– Для налаштування параметрів відображення моделі, користувач може ввести значення шорсткості, згладжування та прозорості у відповідних полях вводу. Наприклад, згладжування з параметром 10 значно покращує зовнішній вигляд об’єкта, згладжуючи кути та поверхні моделі, роблячи їх більш природними та плавними (рис.1.23, 1.24);

6. Віддзеркалення об'єкта:

– Кнопка "Віддзеркалення" дозволяє здійснювати дзеркальне відображення моделі відносно однієї з осей. Це особливо корисно при редагуванні симетричних об’єктів, коли користувачеві потрібно створити дзеркальну копію об'єкта або його частини (рис.1.20);

7. Освітлення та тіні:

– Активувавши функцію "Освітлення та тіні", користувач може додати до сцени тіні, які автоматично генеруються на основі положення джерела світла. Це допомагає візуалізувати модель більш реалістично, показуючи, як вона виглядатиме в умовах різного освітлення (рис.1.23, 1.24);

8. Скелет моделі:

– Кнопка "Вигляд скелету" дозволяє увімкнути режим перегляду внутрішньої структури моделі. У цьому режимі об'єкт показується у вигляді його каркасної сітки, що допомагає детальніше розглянути його геометрію та структуру;

9. Колір моделі:

– Кнопка "Колір моделі" дозволяє обрати колір та перефарбувати модель, використовуючи колір з палітри кольорів (рис.1.24).

Тестування програми показало, що всі функції працюють коректно. Застосунок дозволяє з легкістю редагувати тривимірні моделі, змінювати їх

зовнішній вигляд та взаємодіяти з різними параметрами шейдерів і освітлення.

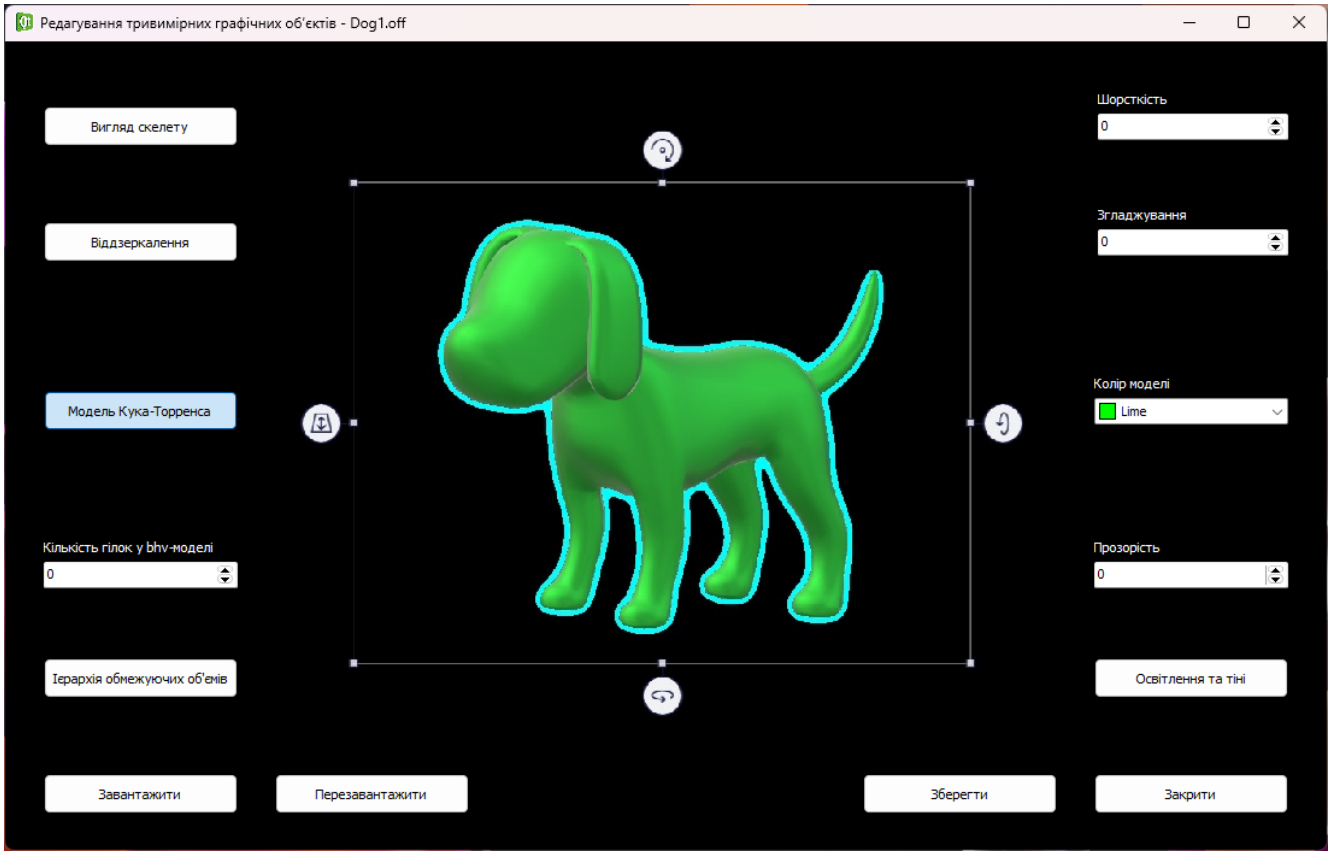


Рисунок 1.24. Змінення кольору моделі та активація режиму Кука-Торренса

Процес тестування також підтвердив високу ефективність алгоритмів рендерингу та обробки об'єктів, таких як BVH та згладжування. Завдяки використанню бібліотеки OpenGL для візуалізації та Qt для побудови інтерфейсу, програма працює стабільно і дозволяє обробляти навіть складні моделі без суттєвих затримок.

Крім того, інтерфейс програми є інтуїтивно зрозумілим і не вимагає від користувача попередніх знань у галузі тривимірної графіки, що робить її доступною для широкого кола користувачів.

2 ЕКОНОМІЧНИЙ РОЗДІЛ

2.1 Резюме

У ході виконання дипломного проєкту було розроблено та протестовано програмне забезпечення, призначене для редагування тривимірних графічних об’єктів. Реалізація проєкту здійснювалася з використанням мови програмування C++, бібліотеки OpenGL для графічного відображення та бібліотеки Qt для створення користувацького інтерфейсу..

Ефективність програмного забезпечення залежить не лише від його внутрішньої якості, але й від успішності процесу його створення. Якість ПЗ оцінюють за кількома критеріями: з позиції користувача, з точки зору ефективного використання ресурсів та відповідності встановленим вимогам. З погляду користувача, якість програмного продукту включає аналіз витрат, пов’язаних з його розробкою, зокрема – оцінку трудомісткості та вартості виробництва. У цьому розділі проведено розрахунок собівартості створеного програмного забезпечення.

2.2 Визначення трудомісткості розробки програмного забезпечення

Тривалість створення програмного продукту визначається його масштабом, рівнем трудомісткості, кваліфікацією виконавців та встановленими ринковими термінами. Використовуючи метод структурної аналогії та аналіз відповідних каталогів аналогічного ПЗ, можна встановити обсяг програмного засобу, що виражається у тисячах умовних машинних команд для аналога

Таблиця 2.1. Каталог аналогів

Найменування ПП	**Обсяг функції ПП – V_o, усл. машинних командах.**
1. ПП автоматизації засобів по каталогу	680 – 7000
2. ПП автоматизованих розрахунків	1300 – 8600
3. ПП оптимізації розрахунків	1300 – 4200

У таблиці 2.1 представлені аналоги програмного забезпечення, функції

яких, у більшому або меншому ступені, виконує розроблений програмний продукт. Для нашого варіанта виділено сірим кольором.

Вибравши аналог ПЗ, що містить Vo в умовних машинних командах, трудомісткості визначати на основі табл.2.2

Таблиця.2.2. Норма часу

Обсяг ПЗ, тис.умов.машинних команд	**Норма часу, люд/год**
1.00	229
2.00	244
3.00	262

На підставі отриманого значення, по довіднику, визначається укрупнена норма часу на розробку аналога програмного забезпечення (коректується поправочним коефіцієнтом враховуючої умови розробки ПП, тобто в умовах комп'ютера, Кк=0,7÷0,8): Тар = 229 х 0,8 = 183,2 (люд/годин).

Трудомісткість програмного продукту визначається окремо для кожного етапу розробки, виходячи з показників трудомісткості відповідного аналога. При цьому враховують рівень складності розробки, ступінь інноваційності та частку використання стандартних модулів. Розрахунки проводяться згідно з наступними формулами:

$$T_{ТЗ} = T^a p \times L_1 \times K_H \qquad (2.1)$$

$$T_{ТП} = T^a p \times L_2 \times K_H \qquad (2.2)$$

$$T_{РП} = T^a p \times L_3 \times K_H \times K_T \qquad (2.3)$$

Для розрахунку необхідні наступні коефіцієнти:

Li – питома вага і-го етапу розробки (див. табл. 2.3.);

Кн – поправочний коефіцієнт, що враховує ступінь новизни (див. табл. 2.4.);

Кт – поправочний коефіцієнт, що враховує ступінь використання в розробці типових програм (див. табл. 2.5)

Таблиця 2.3. Значення питомих коефіцієнтів трудомісткості стадії в загальній трудомісткості розробки ПП

Код стадії	**Ступінь новизни**		
	А	Б	В
ТЗ (L_1)	0,15	0,12	0,12
ТП (L_2)	0,16	0,15	0,11
РП (L_3)	0,55	0,58	0,61

Для нашого варіанта виділено сірим кольором.

Таблиця 2.4. Значення поправочного коефіцієнта, що враховує ступінь новизни

Код ступеня новизни	**Ступінь новизни**	**Значення $К_н$**
А	Принципово нові ПП	1,75 – 1,2
Б	ПП – розвиток визначеного параметричного ряду	1,0 – 0,8
В	ПП маючий аналог	0,7

Для нашого варіанта виділено сірим кольором.

Таблиця 2.5. Значення коефіцієнта ступеня використання в розробці типових програм

Ступінь охоплення реалізованих функцій розроблювального ПП типовими програмами, %	**Значення $К_т$**
60 і вище	0,6
40-60	0,7
20-40	0,8
До 20	0,9

Для нашого варіанта виділено сірим кольором.

Тепер розраховуємо трудомісткість по кожному етапу окремо:

Трудомісткість технічного завдання

$$Т_{тз}=Т^а * L_1 * К_н = 183,2 * 0,12 * 0,7 = 15,38 \text{ (люд/годин)} \quad (2.4)$$

Трудомісткість розробки технічного проекту

$$Т_{тп}=Т^а * L_2 * К_н = 183,2 * 0,11 * 0,7 = 14,11 \text{ (люд/годин)} \quad (2.5)$$

Трудомісткість розробки робочого проекту

$$Т_{рп}=Т^а * L_3 * К_н * К_т = 183,2 * 0,61 * 0,7 * 0,6 = 46,94 \text{ (люд/годин)} \quad (2.6)$$

Для подальших розрахунків визначили кількість папера, витраченого на

кожен етап: технічне завдання $N_{тз}$= 2 (стр), розробка ТП $N_{тп}$=26 (стр), розробка робочого проекту $N_{рп}$=6(стр), пояснювальна записка відповідно $N_{пз}$=22(стр) Розрахунок зведений у таблицю 2.6.

Таблиця 2.6. Розрахунок трудомісткості ПП

Найменування етапів	**Розрахунок, годин**		
1.ТЗ	$T_{Ртз}$=15,38	$T_{кк}=0,7*N_{тз}$= 0,7*2=1,4	$T_{нк}=0,15*N_{тз}=0,15*2=0,3$
2.Розробка ТП	$T_{Ртп}$=14,11	$T_{кк}=0,7*N_{тп}=0,7*30=21,0$	$T_{нк}=0,15*N_{тп}=0,15*30=4,5$
3.Розробка РП	$T_{Ррп}$= 46,94	$T_{кк}=0,7*N_{рп}=0,7*15=10,5$	$T_{нк}=0,15*N_{рп}=0,15*15=2,25$
4.Розробка ПЗ	$T_{пз}=1,5*N_{пз}$= 1,5*28=42	$T_{кк}=0,7*N_{тз}=0,7*28=19,6$	$T_{нк}=0,15*N_{пз}=0,15*28=4,2$
Усього, в т.ч.:	182,18		
- на розробку	$\Sigma T_{р}$=118,43		
- контроль керівника		$\Sigma T_{кк}$=52,5	
- нормоконтроль			$\Sigma T_{нк}$=11,25

2.3 Розрахунок ціни програмного продукту

Для оцінки вартості програмного продукту розглядаються основна заробітна плата виконавців, матеріальні витрати та загальні витрати на розробку ПЗ. Детальний розрахунок основної заробітної плати наведено у таблиці 2.7. Згідно зі статтею 8 «Закону про Державний бюджет України на 2025» з 1 січня 2025 року встановлено мінімальну місячну заробітну плату у розмірі 8000 гривень, а також мінімальну погодинну тарифну ставку – 48,00 грн.

Таблиця 2.7. Розрахунок основної заробітної плати виконавців

Найменування робіт	**Трудомісткість робіт, години**	**Погодинна тарифна ставка, грн.**	**Розрахунок, грн.**
1.Розробка ПП	118,43	48,00	5684,64
2.Контроль керівника	52,5	110,00	5775,00
3.Нормоконтроль	11,25	100,00	1125,00
Усього	-	-	ΣЗо= 12584,64

Зробимо розрахунок матеріальних витрат на розробку ПП. Розрахунок зведемо в таблицю 2.8.

Таблиця 2.8. Розрахунок матеріальних витрат на розробку ПЗ

Найменування матеріальних витрат	Тип, модель	Кількість	Ціна одиниці, грн.	Вартість, грн.
Папір	Лист А4	60	5.0	300,0
Разом	-		-	$В_{м\,і}$=300,0
Транспортно – заготівельні Витрати (10%)				$Втр\_з = 0{,}1 \times Вм1$ = 0,1*300=30,0
Усього				$В_м=В_{мі}+В_{тр\_з=}$ 330,0

На підставі отриманих даних по окремих статтях витрат складена калькуляція планової собівартості в цілому ПП за формою, приведеною в таблиці 2.9.

Таблиця 2.9. Розрахунок статей витрат планової собівартості

Стаття витрат	Значення, грн.	Формула розрахунку
1. Матеріали	330,0	$В_м$ (див. табл. 2.8.)
2. Основна заробітна плата	12584,64	$З_о$ (див. табл. 2.7.)
3.Додаткова заробітна плата	1258,46	$Зд = 0{,}1 \times Зо$ = 12584,64*0,1
4.Відрахування до єдиного фонду соціального внеску	3045,48	$Вє.с.в. = 0{,}22 \times (Зо + Зд)$=0,22* (12584,64+1258,46)
5. Накладні витрати	5033,86	$Внак. = 0{,}4 \times Зо$ =0,4 * 12584,64
6. Повна собівартість	22252,44	$С_{пов}$= $В_м$ +$З_о$ + $З_д$ +Вє.с.в. + Внак.= 330,0+12584,64+1258,46+3045,48 +5033,86

Розмір прибутку, що включається в ціну, визначаємо по наступній формулі:

$П=(С_{пов}*Р)/100=(22252,44*10)/100=2225,24$ грн (2.7)

Де р – плановий рівень рентабельності (10-15%).

Оптова ціна (кошторисна вартість) визначається по формулі:

$Ц_о =С_{пов}+П$ = = 22252,44+2225,24 = 24477,68 грн; (2.8)

Виходячи з отриманих даних, ціна реалізації розробленого програмного забезпечення становитиме:

$Ц_р=Ц_о+ПДВ$ = 24477,68 +24477,68 *0.2 = 29373,22 грн; (2.9)

3 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Сучасні комп’ютерні технології відкривають широкі можливості для автоматизації рутинних процесів, оптимізації обробки даних та швидкого доступу до інформаційних ресурсів. Вони сприяють підвищенню продуктивності праці та ефективності управління, проте їхнє активне використання також створює певні виклики, зокрема збільшене навантаження на здоров’я користувачів через тривалий час роботи з комп’ютером.

У рамках даного дипломного проєкту розглядається розробка програмної моделі для оцінки стійкості комп’ютерних систем і мереж. Ця модель являє собою аналітичний інструмент, заснований на сучасних методах аналізу робочих місць. Вона інтегрується у систему користувача, яка включає комп’ютерне обладнання з відповідним програмним забезпеченням, спеціально створеним для оцінки життєздатності, надійності та безпеки інформаційних систем. При цьому дотримання стандартів безпеки праці залишається ключовим аспектом, оскільки забезпечує комфортну та безпечну роботу користувача в умовах підвищеного навантаження.

Таким чином, створення такої моделі дозволить не лише автоматизувати процес аналізу комп’ютерних систем і мереж, а й врахувати комплекс реальних факторів, що впливають на їхню стабільність та ефективність у практичному застосуванні.

3.1 Аналіз небезпечних і шкідливих факторів, що впливають на користувача ПК

До основних критеріїв забезпечення гігієни робочого середовища належать інтенсивність освітлення, температура повітря, вологість, рівень шумового забруднення, ступінь вібраційного впливу, токсичність, загазованість, а також обмеження загальної м'язової активності (гіподинамія). Крім цього, враховується дія електростатичного поля та вплив як неіонізуючих, так і іонізуючих електромагнітних випромінювань.

3.2 Гігієнічні вимоги до виробничого середовища

Державні санітарні норми, зокрема ДСАНПіН 3.3.2.007-98 «Гігієнічні вимоги до організації роботи з візуальними дисплейними терміналами електронно-обчислювальних машин», спрямовані на запобігання негативного впливу шкідливих чинників, що супроводжують роботу з візуальними дисплейними терміналами, на здоров’я працівників.

3.2.1 Вимоги до приміщення

Розміщення робочих місць із використанням ВДТ, ЕОМ і ПЕОМ заборонено у підвальних приміщеннях та на цокольних поверхах. Для кімнат, призначених для роботи з візуальними дисплейними терміналами, рекомендується орієнтувати вікна у напрямку півночі або північного сходу. На вікнах повинні бути встановлені регульовані жалюзі або штори, що дозволяють їх повністю закривати для забезпечення оптимальних умов освітлення.

Планувальні рішення будівель і приміщень, де розташовано відеодисплейні термінали, мають відповідати вимогам ДСАНПіН 3.3.2.007-98. Для робочого місця програміста передбачено мінімальну площу не менше 6 кв. м та об’єм приміщення не менше 20 куб. м. Крім того, стіни приміщень повинні бути пофарбовані матовою фарбою, а в приміщеннях з ВДТ обов’язково мають бути передбачені зони для відпочинку та психологічного розвантаження.

3.2.2 Освітлення та шум

Для забезпечення належного освітлення приміщення, де працює програміст, застосовується комбінована система, що поєднує природне освітлення із додатковим штучним світлом. Загальне оздоблення простору виконується за допомогою газорозрядних ламп типу ЛД. Згідно з встановленими нормами, для робочого місця, на якому здійснюються високоточні операції (де мінімальний розмір об'єкта розрізнення становить 0,3–0,5 мм), необхідна освітленість рівномірно має досягати 300 лк. В цілому, ці вимоги щодо освітлення забезпечені.

У робочих приміщеннях основним джерелом шумового навантаження є звуки, що генеруються ПЕОМ. Крім того, значну частину шуму створюють джерела електромагнітного походження – це коливання компонентів електромеханічних пристроїв під впливом змінних магнітних полів. До того ж, в приміщеннях виникає структурний шум, який випромінюють поверхні конструктивних елементів (стіни, перекриття, перегородки) у звуковому спектрі частот. Для зниження або усунення негативного впливу шуму доцільно ізолювати робочі зони, розташовуючи їх у частинах будівлі, що знаходяться в глибині та ведуть своїми вікнами у двір – таким чином мінімізується вплив міського шуму. Крім цього, необхідно регулярно перевіряти герметичність корпусів комп'ютерної техніки та своєчасно здійснювати заміну вентиляторів охолодження.

3.3 Вимоги до організації робочого місця працівника

Конструкція робочого місця користувача комп'ютера, з урахуванням розташування сидіння, засобів керування та засобу відображення інформації, розроблена згідно з антропометричними, фізіологічними та психологічними вимогами, а також відповідно до специфіки виконуваної роботи. Робоче меблеве обладнання повинно бути оснащене можливістю індивідуального регулювання, що дозволить адаптувати його під зріст кожного користувача й підтримувати оптимальну, зручну поставу. Робочий стіл рекомендовано обробляти матовим покриттям, що сприяє зменшенню небажаних відблисків. > > Розміщення дисплея організовано таким чином, щоб його верхня межа відповідала рівню очей, а відстань до екрану становила приблизно 70 см – що повністю входить у допустимий інтервал від 60 до 90 см. Частота мерехтіння екрану fмер дорівнює 100 Гц, що значно перевищує мінімальне рекомендоване значення у 70 Гц. Крім цього, робоче місце розташовано перпендикулярно до віконних прорізів, що дозволяє уникнути прямого та відбитого світлового мерехтіння від вікон та джерел штучного освітлення.

3.4 Мікроклімат

Показники мікроклімату, складу іонів у повітрі, а також рівень шкідливих речовин у робочих зонах, де використовуються ПК, мають відповідати вимогам ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень».

Для підтримки нормативних значень мікроклімату та забезпечення оптимального співвідношення позитивних і негативних іонів слід передбачити установку пристроїв зволоження, штучної іонізації або кондиціювання повітря. Крім того, рівні інфрачервоного випромінювання не повинні перевищувати встановлених нормативних меж згідно з ГОСТ 12.1.005. Також вміст озону в робочій зоні не має перевищувати 0,1 мг/м³, оксидів азоту – 5 мг/м³, а концентрація пилу повинна залишатися в межах 4 мг/м³.

3.5 Електробезпека

Приміщення, де використовуються імпульсні джерела живлення згідно з ОНТП24-86 і ПУЕ-87, віднесено до категорії об'єктів, де ризик ураження персоналу електричним струмом не є підвищеним. Це пояснюється тим, що відносна вологість повітря не перевищує 75%, температура залишається нижчою за 35°С, а хімічно агресивні середовища відсутні. Електроживлення обладнання організовано від двофазної мережі з заземленою нейтраллю, при напрузі 220 В і частоті 50 Гц, із застосуванням автоматичних пристроїв токового захисту.

В приміщенні обов’язково має бути встановлена схема заземлення. Ураження електричним струмом може виникнути у випадках: 1) при контакті з відкритими струмоведучими елементами; 2) при торканні неструмоведучих частин обладнання, які, через порушення ізоляції або інші причини, опинилися під напругою.

Відповідно до вимог ГОСТ-12.2.007.0-75 устаткування (за винятком ЕОМ II класу) відноситься до I класу та оснащене робочою ізоляцією згідно з ГОСТ 12.1.009-76. Підключення обладнання здійснено згідно з нормативами ПБЕ та ПУЕ, тому додаткових заходів щодо електробезпеки не вимагається.

3.6 Пожежна безпека

Робоче приміщення, що відповідає вимогам ПБЕ та ОНТП 24–86 у сфері вибухово-пожежної безпеки, класифікується як об’єкт категорії «В».

Основними потенційними причинами виникнення пожежі в такому приміщенні є:

1. Коротке замикання електропроводки;

2. Використання побутових електрорадіоприладів;

3. Недотримання встановлених норм протипожежного захисту.

Відповідно до ПУЕ, для зниження ризику виникнення пожежі необхідно забезпечити комплекс заходів, зокрема: ретельну ізоляцію всіх струмоведучих проводів, що підключені до робочих місць, регулярний огляд та перевірку стану їх ізоляції, а також суворе дотримання норм безпечної експлуатації обладнання.

Для гасіння пожеж на робочому місці користувача ПК застосовують як вуглекислотні, так і порошкові вогнегасники.

– Вуглекислотні вогнегасники випускаються у варіанті ручних пристроїв (наприклад, ВВК-5);

– Порошкові вогнегасники представлені моделями ВП-2, ВП-5, ВП-10 та іншими

Рисунок 3.1. Засоби пожежогасіння

З метою своєчасного оповіщення, на дільниці необхідно встановити протипожежну сигналізацію. Проходи та запасні виходи повинні бути вільними. Пожежний щит повинен розміщуватись в доступному місці та містити первинні засоби пожежогасіння (вогнегасник, лопату, відро, простирадло, ящик з піском)

ВИСНОВКИ

У процесі виконання дипломного проекту було створено та протестовано програмне забезпечення для редагування тривимірних графічних об'єктів. Розробка базувалася на використанні мови програмування C++, бібліотеки OpenGL для візуалізації та бібліотеки Qt для реалізації інтерфейсу користувача.

Програма продемонструвала високу ефективність та стабільність роботи під час тестування на різних моделях формату *.off, що слугували вхідними даними.

Тестування розробки показало, що програма успішно виконує всі основні функції, включаючи зміну вигляду скелету моделі, операцію віддзеркалення та використання моделі Кука-Торренса для коректної симуляції освітлення. У ході тестування був використаний функціонал з побудови ієрархії обмежуючих об'ємів, який чітко продемонстрував роботу алгоритму для складних моделей, що дозволяє прискорити процес рендерингу та обробки. Програма також вдало впоралася із налаштуваннями параметрів освітлення та тіней, що забезпечило реалістичне візуальне відображення об'єктів під різними умовами освітлення.

Під час тестування була перевірена зручність інтерфейсу для користувача, яка полягала у легкому завантаженні моделей, їх перезавантаженні та збереженні відредагованих версій. Поля введення для шорсткості, згладжування та прозорості забезпечили належний контроль над зовнішніми характеристиками моделей, що дало можливість користувачам без спеціальних знань у сфері комп'ютерної графіки досягати бажаних результатів.

Загалом, результати тестування підтвердили, що створене програмне забезпечення відповідає заявленим вимогам і дозволяє виконувати ефективне редагування тривимірних об'єктів. Система демонструє стабільну роботу навіть при обробці складних моделей і забезпечує високий рівень гнучкості для користувача завдяки інтуїтивно зрозумілому інтерфейсу та багатому набору інструментів.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Гудзь В. П. Комп'ютерна графіка: підручник – Київ: Видавничий дім «Кондор», 2017. – 356 с.
2. Дудикевич В. Б., Мельник С. І. Графічні системи та комп'ютерне моделювання: навч. посібник – Львів: Вид. Львівської політ-ки, 2018. – 432 с.
3. Паламарчук І. М. Основи комп'ютерної графіки: навч. посібник – Київ: Видавничий центр КНУ ім. Т. Шевченка, 2019. – 224 с.
4. Черняк В. І. Алгоритми і методи комп'ютерної графіки: підручник – Київ: НТУУ «КПІ», 2020. – 340 с.
5. Пасічник В. В. Основи комп'ютерної графіки та обробки зображень: підручник – Львів: Видавництво Львівської політехніки, 2021. – 276 с.
6. Роговий О. М. Комп'ютерна графіка та 3D-моделювання: навч. посібник – Київ: Освіта України, 2019. – 380 с.
7. Боярчук О. О. Програмування на C++: підручник – Київ: Видавництво «Либідь», 2018. – 384 с.
8. Дейтел Г. Програмування на C++: повний курс – Львів: БаК, 2020. – 650 с.
9. Савінов О. М. Основи об’єктно-орієнтованого програмування на C++: навч. посібник – Харків: Фоліо, 2019. – 280 с.
10. Бутенко С. В. Програмування в середовищі Qt: навч. посібник – Київ: Видавництво КПІ, 2021. – 312 с.
11. Ляшенко О. О. Qt Creator: практичний посібник – Львів: Видавництво «Афіша», 2019. – 210 с.
12. Офіційна документація Qt [Електронний ресурс]. – Режим доступу: https://doc.qt.io.
13. Офіційний сайт бібліотеки OpenGL [Електронний ресурс]. – Режим доступу: https://www.opengl.org.
14. Офіційна документація C++ [Електронний ресурс]. – Режим доступу: https://cplusplus.com.
15. Офіційний сайт по роботі з файлами формату *.off [Електронний ресурс]. – Режим доступу: https://www.geometry.caltech.edu/software.html.

ДОДАТОК А. Ключові фрагменти коду програми для редагування тривимірних моделей

1. Клас Mesh

Клас Mesh відповідає за зберігання 3D-моделі та її маніпуляцію. Включає методи для завантаження файлів формату *.off, зміни геометрії моделі, її subdivision та згладжування:

```
class Mesh {
public:
    Mesh() {}

    // Завантаження 3D-моделі з файлу .off
    bool loadOFF(const std::string &filename);

    // Метод для поділу сітки на дрібніші трикутники
    void subdivide();

    // Оновлення нормалей після зміни моделі
    void recomputeNormals();

    // Повертає позицію вершин
    const std::vector<Vec3f>& getPositions() const { return m_positions; }

    // Повертає трикутники
    const std::vector<Triangle>& getTriangles() const { return m_triangles; }

private:
    std::vector<Vec3f> m_positions;  // Вершини моделі
    std::vector<Triangle> m_triangles;  // Трикутники моделі

    // Внутрішні методи для subdivision
    void addOddVertices(std::map<std::pair<int, int>, Vec3f>& oddVertices,
std::map<std::pair<int, int>, unsigned int>& oddVertexIndices,
std::vector<Vec3f>& new_positions);
    void updateEvenVertices(std::vector<unsigned int>& valence,
std::vector<Vec3f>& new_positions);
    void reindex_subdivision(const std::map<std::pair<int, int>, unsigned
int>& oddVertexIndices, std::vector<Triangle>& new_triangles);
};

// Метод для завантаження файлу формату .off
bool Mesh::loadOFF(const std::string &filename) {
    std::ifstream ifs(filename);
    if (!ifs.is_open()) {
        std::cerr << "Помилка завантаження файлу: " << filename << std::endl;
        return false;
    }

    std::string header;
    ifs >> header;
    if (header != "OFF") {
        std::cerr << "Файл не підтримується: " << header << std::endl;
        return false;
    }

    int num_vertices, num_faces, num_edges;
    ifs >> num_vertices >> num_faces >> num_edges;

    m_positions.resize(num_vertices);
    for (int i = 0; i < num_vertices; ++i) {
        ifs >> m_positions[i][0] >> m_positions[i][1] >> m_positions[i][2];
    }

    m_triangles.resize(num_faces);
    for (int i = 0; i < num_faces; ++i) {
```

```
            int n;
            ifs >> n;
            if (n == 3) {
                ifs >> m_triangles[i][0] >> m_triangles[i][1] >>
m_triangles[i][2];
            }
        }

        recomputeNormals();
        return true;
}

// Метод для subdivision моделі
void Mesh::subdivide() {
    map<pair<int, int>, Vec3f> oddVertices;
    map<pair<int, int>, unsigned int> oddVertexIndices;
    vector<Vec3f> new_positions = m_positions;
    vector<Triangle> new_triangles;

    addOddVertices(oddVertices, oddVertexIndices, new_positions);
    updateEvenVertices(valence, new_positions);
    reindex_subdivision(oddVertexIndices, new_triangles);

    m_positions.swap(new_positions);
    m_triangles.swap(new_triangles);
    recomputeNormals();
}

// Оновлення нормалей для нових вершин
void Mesh::recomputeNormals() {
    m_normals.resize(m_positions.size(), Vec3f(0.0f, 0.0f, 0.0f));
    for (const Triangle & t : m_triangles) {
        Vec3f normal = crossProduct(m_positions[t[1]] - m_positions[t[0]],
m_positions[t[2]] - m_positions[t[0]]);
        m_normals[t[0]] += normal;
        m_normals[t[1]] += normal;
        m_normals[t[2]] += normal;
    }

    for (Vec3f & n : m_normals) {
        normalize(n);
    }
}
```

2. Клас Camera

Клас Camera забезпечує управління рухом та обертанням камери у просторі 3D-сцени. Він дозволяє переміщати та обертати камеру відносно моделі.

```
class Camera {
public:
    Camera() : fov_angle(45.0f), aspect_ratio(1.0f), znear(0.1f),
zfar(1000.0f) {}

    // Налаштування виду камери
    void setView(float fov, float aspect, float znear, float zfar) {
        this->fov_angle = fov;
        this->aspect_ratio = aspect;
        this->znear = znear;
        this->zfar = zfar;
    }

    // Поворот камери
    void rotate(int u, int v);

    // Переміщення камери вперед-назад
```

```
    void moveForward(float dist) {
        position += dist * (target - position).normalize();
    }

    // Переміщення камери вліво-вправо
    void moveSideways(float dist) {
        Vec3f right = crossProduct(target - position, Vec3f(0, 1,
0)).normalize();
        position += dist * right;
    }

    // Отримання проекційної матриці для рендерингу
    Mat4 getProjectionMatrix() const;

private:
    float fov_angle;
    float aspect_ratio;
    float znear;
    float zfar;
    Vec3f position;
    Vec3f target;
    Quatf curquat, lastquat;

    // Обертання за допомогою trackball
    void trackball(Quatf & q, float p1x, float p1y, float p2x, float p2y);
    void add_quats(const Quatf & q1, const Quatf & q2, Quatf & dest);
};

// Реалізація повороту камери
void Camera::rotate(int u, int v) {
    if (moving) {
        trackball(lastquat, (2.0 * beginu / W) - 1.0, (H - 2.0 * beginv) / H
- 1.0,
                  (2.0 * u / W) - 1.0, (H - 2.0 * v) / H - 1.0);
        beginu = u;
        beginv = v;
        spinning = 1;
        add_quats(lastquat, curquat, curquat);
    }
}

// Отримання матриці проекції
Mat4 Camera::getProjectionMatrix() const {
    return perspective(fov_angle, aspect_ratio, znear, zfar);
}
```

3. Клас Ray

Клас Ray відповідає за рейтрейсинг і перевірку перетину променя з трикутниками моделі.

```
class Ray {
public:
    Ray(Vec3f origin, Vec3f direction) : origin(origin), direction(direction)
{}

    // Перевірка перетину з трикутником
    bool isIntersectedWithTriangle(const Vec3f & v0, const Vec3f & v1, const
Vec3f & v2) const;

private:
    Vec3f origin;
    Vec3f direction;
};

// Метод для перевірки перетину променя з трикутником
```

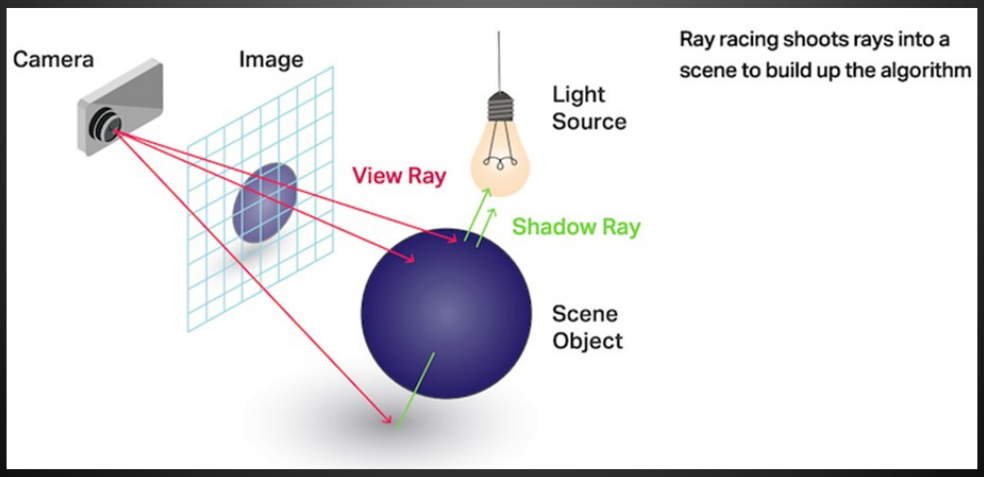
```
bool Ray::isIntersectedWithTriangle(const Vec3f & v0, const Vec3f & v1, const
Vec3f & v2) const {
    Vec3f edge1 = v1 - v0;
    Vec3f edge2 = v2 - v0;
    Vec3f h = crossProduct(direction, edge2);
    float a = dotProduct(edge1, h);

    if (fabs(a) < 0.0001f) return false;

    float f = 1.0f / a;
    Vec3f s = origin - v0;
    float u = f * dotProduct(s, h);

    if (u < 0.0 || u > 1.0) return false;

    Vec3f q = crossProduct(s, edge1);
    float v = f * dotProduct(direction, q);

    if (v < 0.0 || u + v > 1.0) return false;

    float t = f * dotProduct(edge2, q);
    return t > 0.0001f;
}
```

4. Клас BVH

Клас BVH відповідає за побудову ієрархії обмежуючих об'ємів для прискорення рейтрейсингу.

```
class BVH {
public:
    BVH(const AxisAlignedBoundingBox & aabb, const Triangle & triangle)
        : aabb(aabb), triangle(triangle), left_child(nullptr),
right_child(nullptr) {}

    // Рекурсивна побудова BVH
    BVH* buildBVH(const std::vector<Triangle> & triangles, const Mesh & mesh,
unsigned int depth);

private:
    AxisAlignedBoundingBox aabb;
    Triangle triangle;
    BVH* left_child;
    BVH* right_child;
};

// Побудова BVH дерева
BVH* BVH::buildBVH(const vector<Triangle>& triangles, const Mesh& mesh,
unsigned int depth) {
    if (triangles.empty()) return nullptr;

    Vec3f min_p, max_p;
    calculateMinMax(min_p, max_p, triangles, mesh);
    AxisAlignedBoundingBox aabb(min_p, max_p);

    if (triangles.size() == 1) {
        return new BVH(aabb, triangles[0]);
    }

    // Поділ на підмножини для побудови лівого і правого піддерева
    vector<Triangle> left_triangles, right_triangles;
    divideTriangles(triangles, left_triangles, right_triangles);

    return new BVH(aabb, buildBVH(left_triangles, mesh, depth + 1),
buildBVH(right_triangles, mesh, depth + 1));
}
```

ДОДАТОК Б. Слайди мультимедійної презентації



Ілюстрація алгоритму Ray tracing

$$x(\theta,\phi) = r \cdot sin(\theta) \cdot cos(\phi)$$

$$y(\theta,\phi) = r \cdot sin(\theta) \cdot sin(\phi)$$

$$z(\theta,\phi) = r \cdot cos(\theta)$$

де θ і ϕ — кутові координати в сферичній системі координат.

Параметричне рівняння сфери радіуса r в тривимірному просторі

$$T = \arg\max_{T_i \in S} \min \angle(T_i)$$

де T — сукупність трикутників, а $\angle(T_i)$ — кути трикутника T_i.

Математична задача для побудови тріангуляції

$$P_{\text{new}} = \frac{P_{\text{old}} + \sum_{i=1}^{n} P_{\text{neigh}_i}}{n+1}$$

де:

- P_{new} — нова позиція вершини;
- P_{old} — стара позиція вершини;
- P_{neigh_i} — позиція сусідньої вершини;
- n — кількість сусідніх вершин.

Алгоритм	*Кількість операцій*	*Реалістичність рез-ту*	*Продуктивність*
Catmull-Clark	$O(n^2)$	Висока	Помірна
Loop Subdivision	$O(n^2)$	Середня	Висока
Butterfly	$O(n \log n)$	Низька	Висока

Порівняння методів згладжування поверхонь

Математичне представлення NURBS-кривої

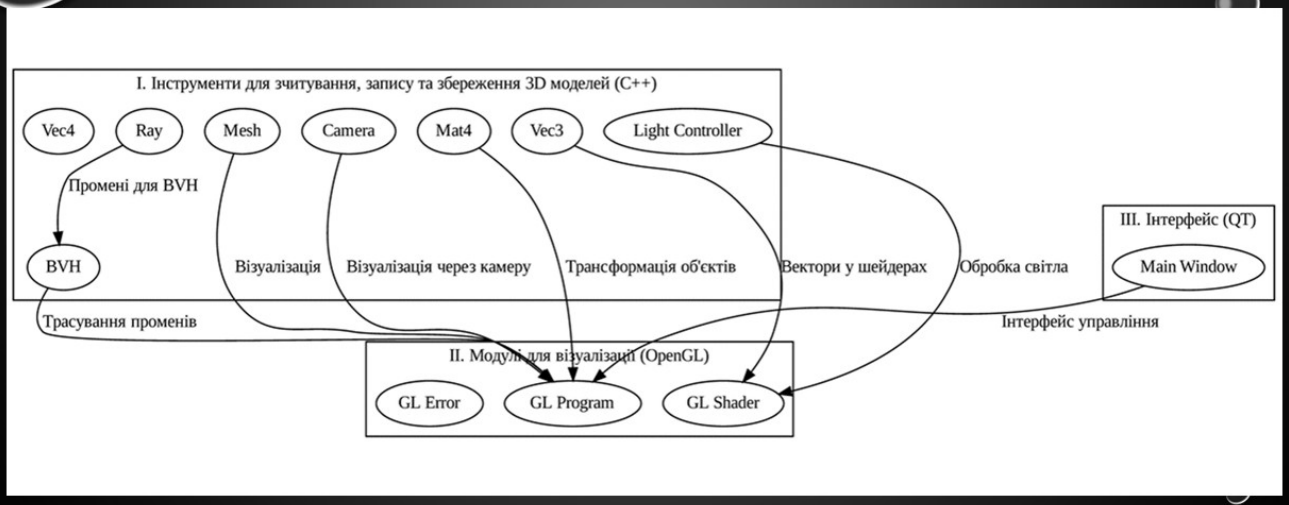
$$C(u) = \frac{\sum_{i=0}^{n} N_{i,p}(u) \cdot P_i \cdot w_i}{\sum_{i=0}^{n} N_{i,p}(u) \cdot w_i}$$

Алгоритм	*Тип об'єкта*	*Рівень складності*	*Реалістичність результату*
Шум Перліна	Ландшафти, текстури	Середній	Висока
L-системи	Дерева, рослинність	Низький	Висока
Фрактали	Гори, гірські масиви	Високий	Висока
Шум Вороново	Камені, поверхні	Середній	Середня

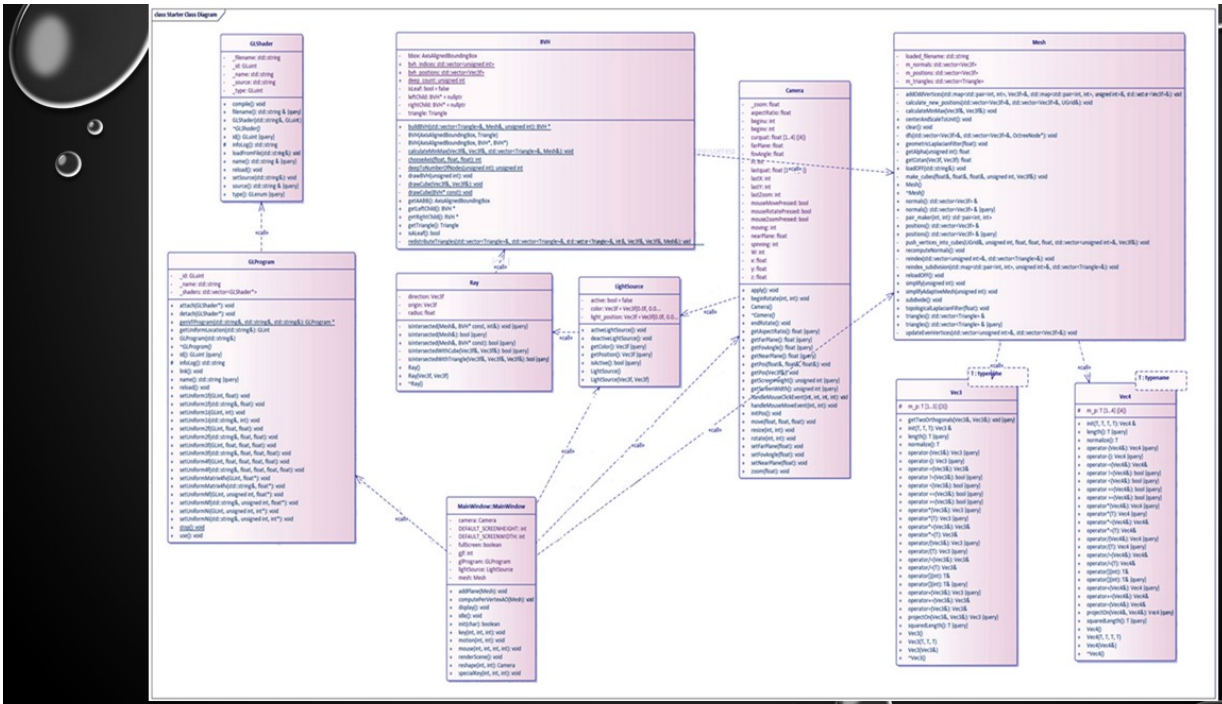
Порівняння процедурних алгоритмів для різних типів об'єктів

Вид сплайна	*Використання*	*Гладкість кривої*	*Гнучкість в керуванні*
Кубічний сплайн	Просте моделювання	Висока	Середня
B-сплайн	Складні криві	Висока	Висока
NURBS	Інженерне моделювання, CAD	Дуже висока	Дуже висока

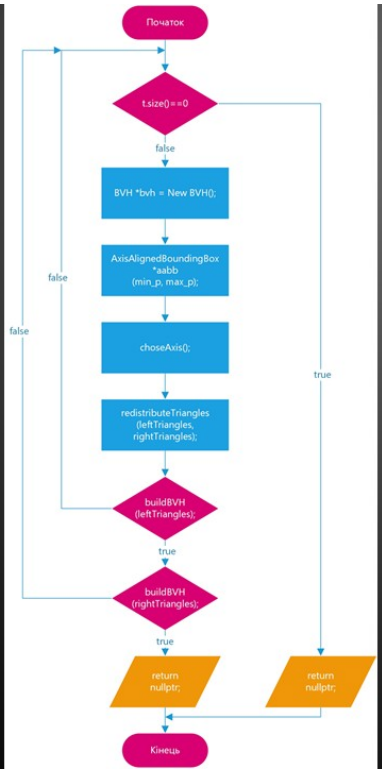
Порівняння видів сплайнів



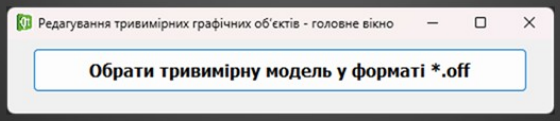
Структурна схема основних компонентів застосунку



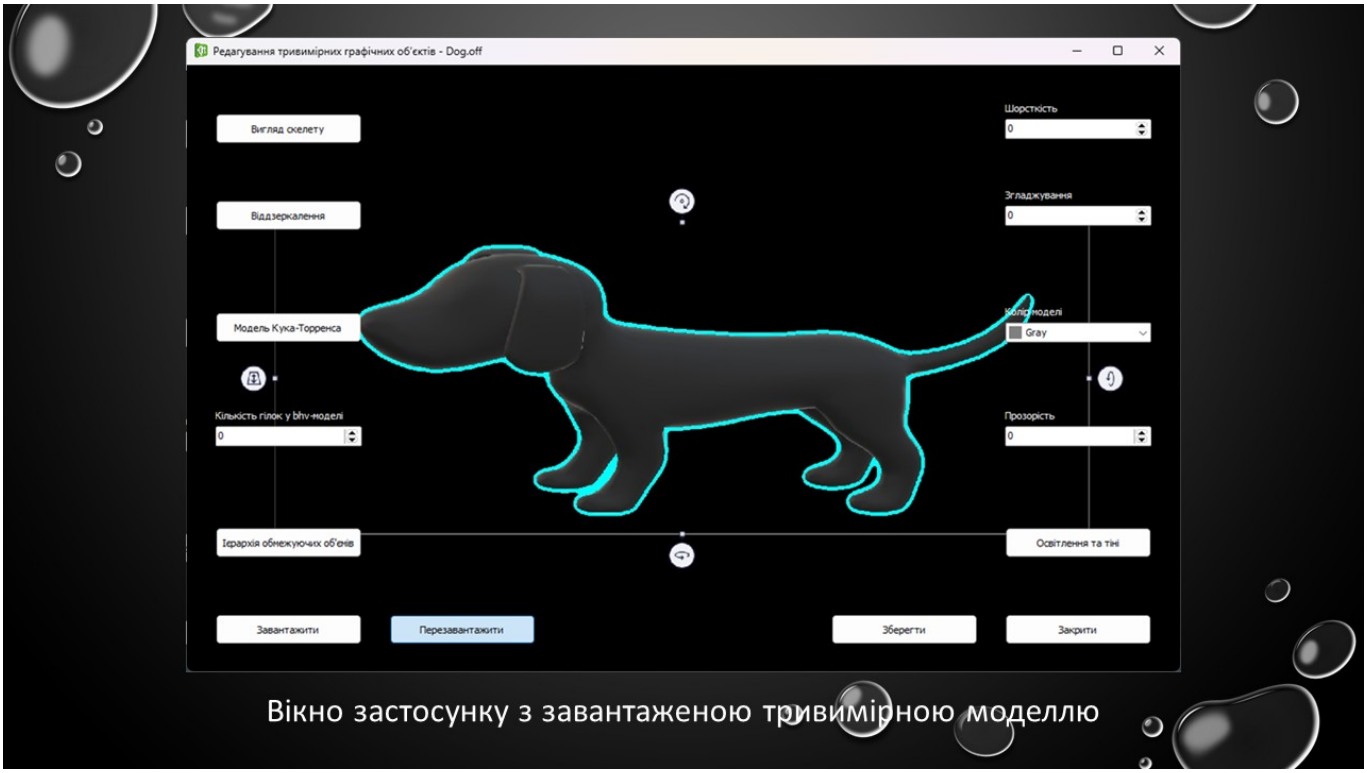
Діаграма класів застосунку



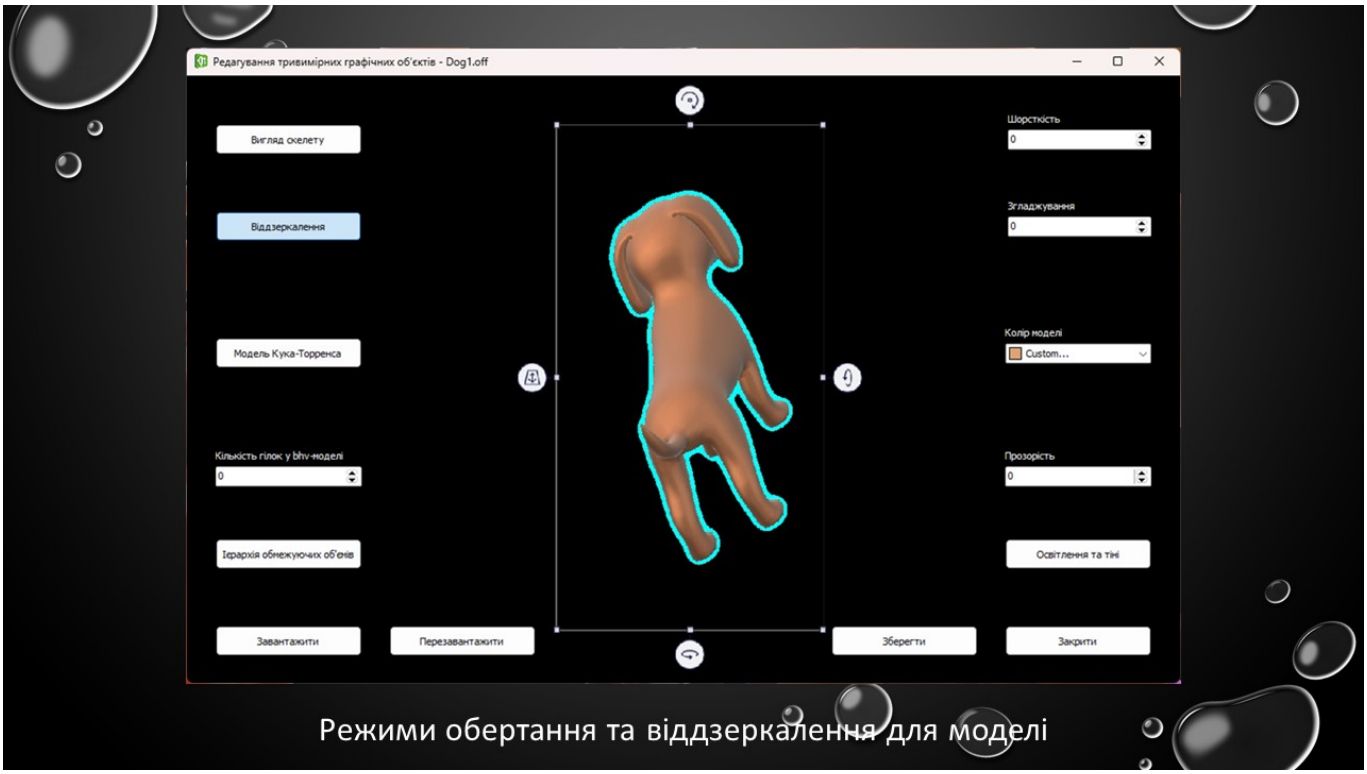
Блок-схема алгоритму побудови ієрархії обмежуючих об'ємів



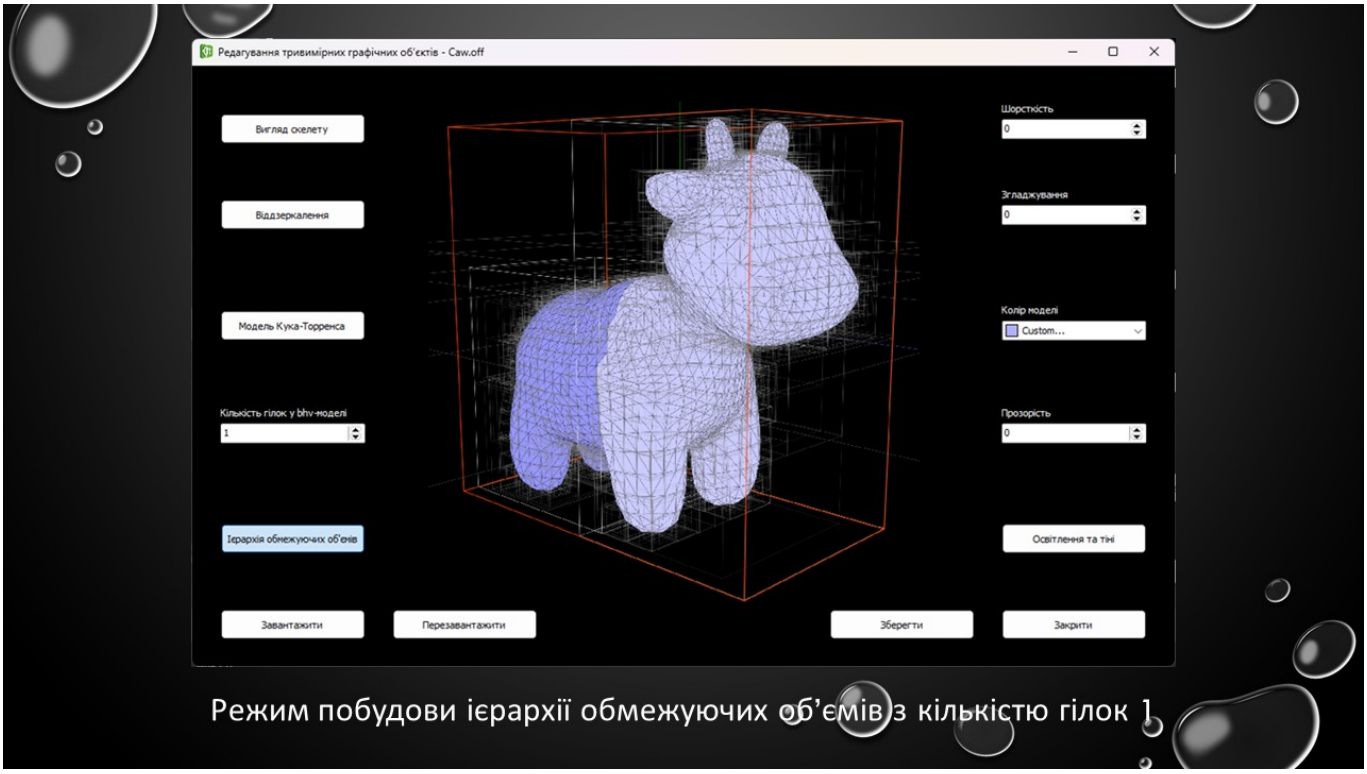
Головне вікно застосунку



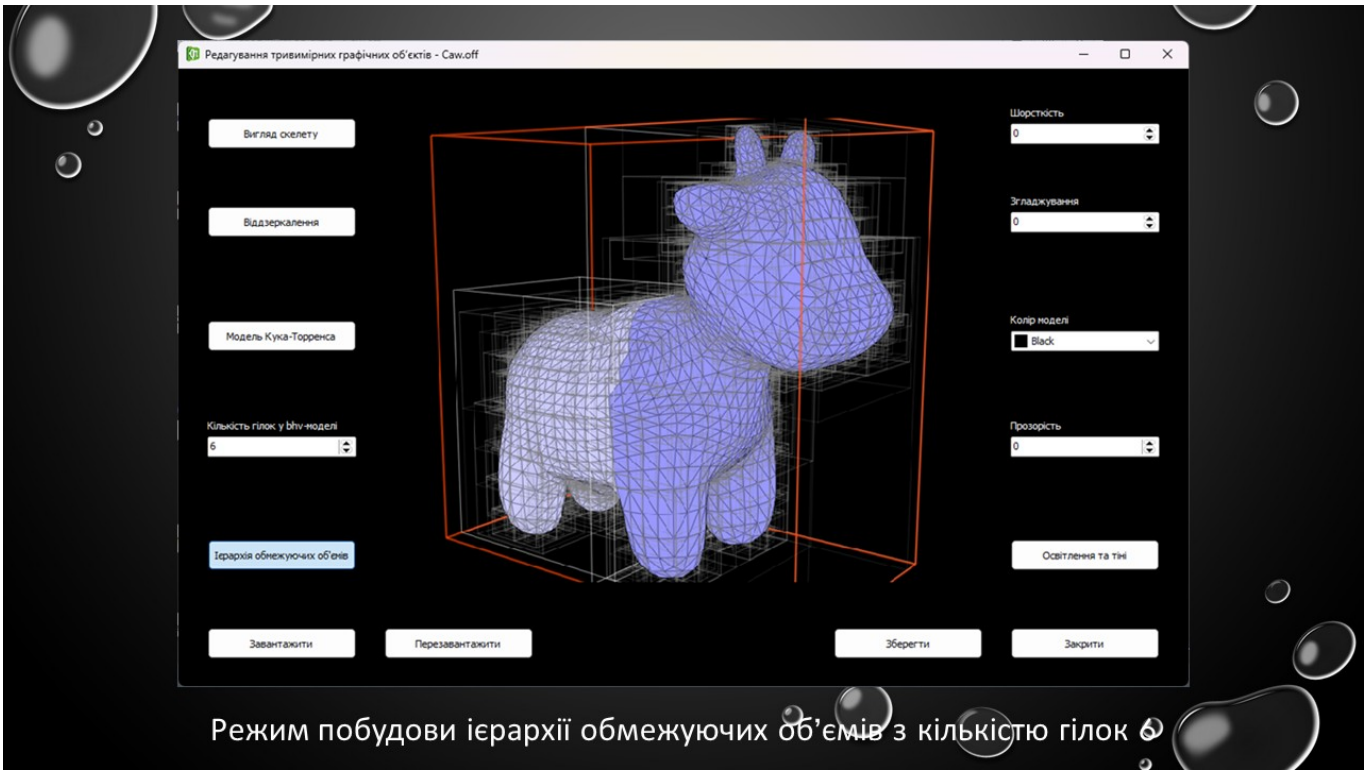
Вікно застосунку з завантаженою тривимірною моделлю



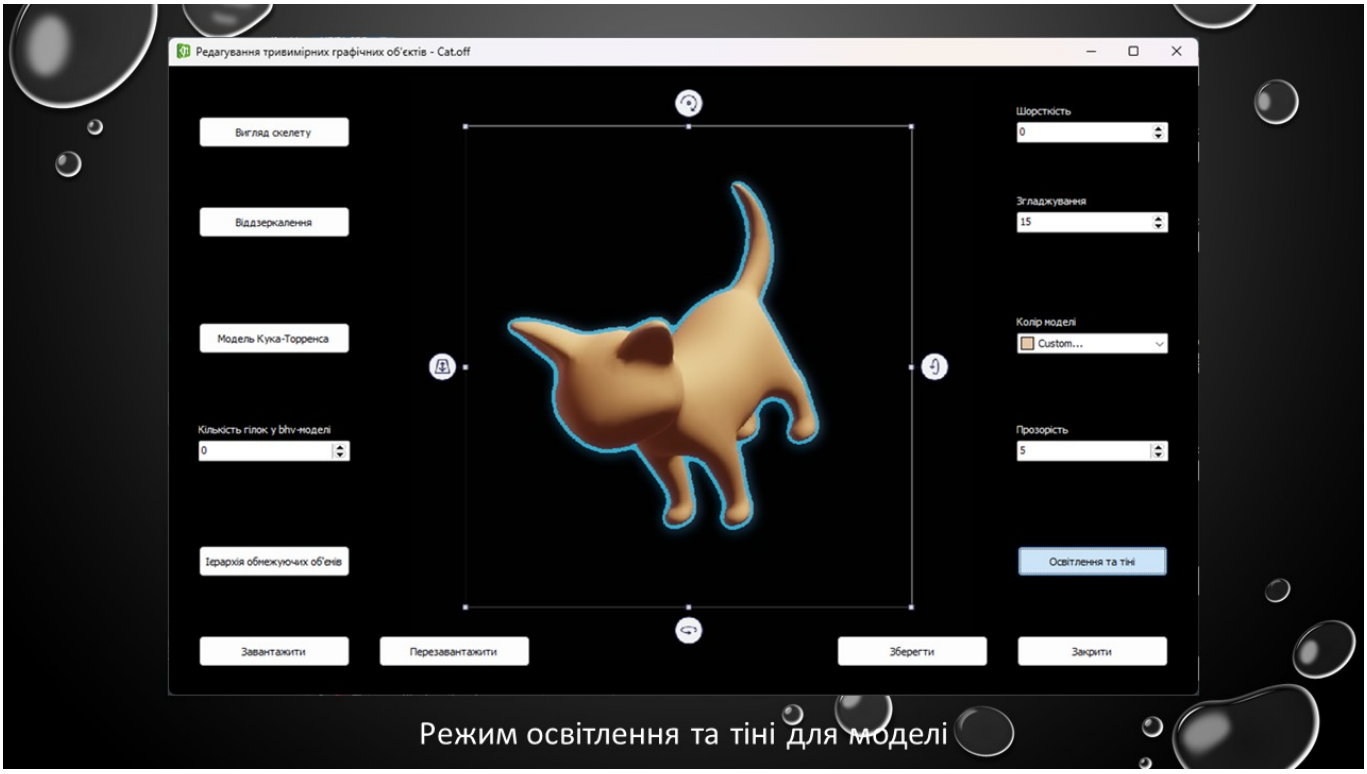
Режими обертання та віддзеркалення для моделі



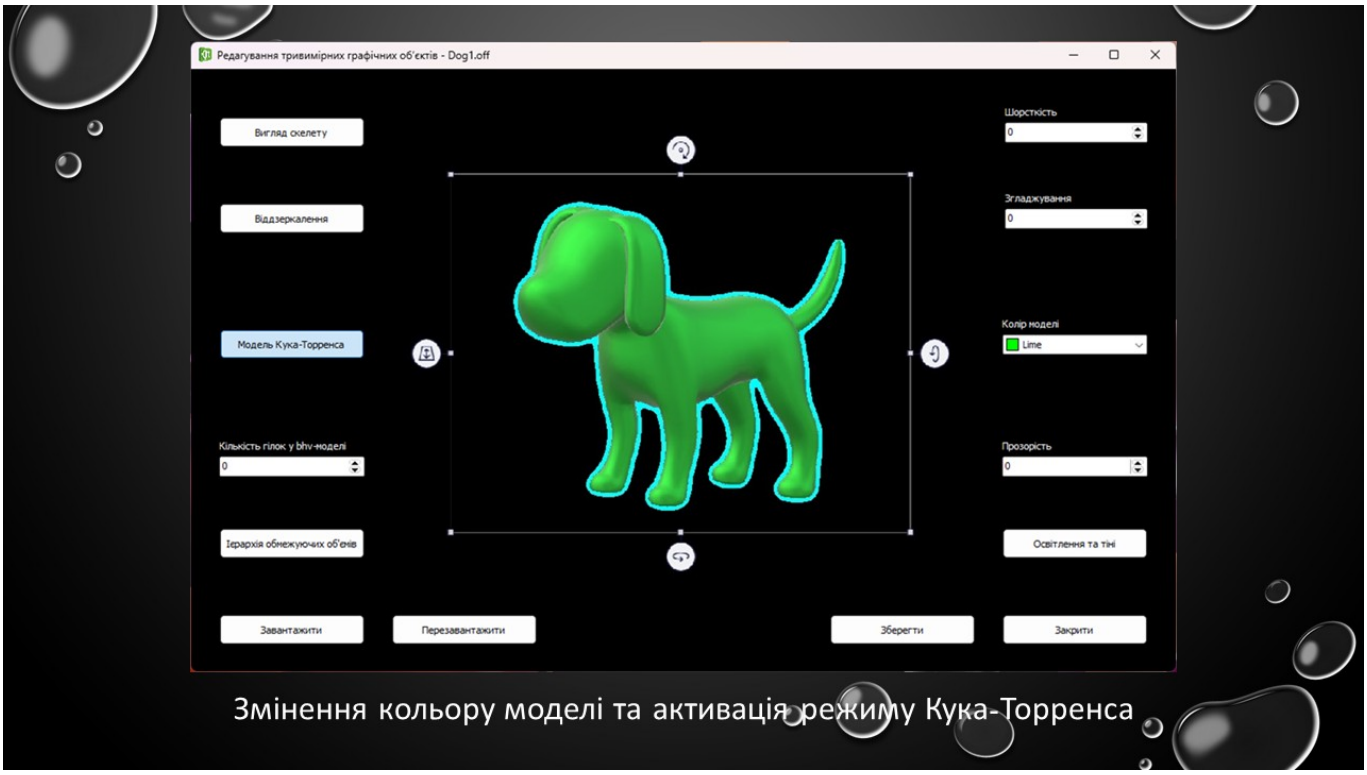
Режим побудови ієрархії обмежуючих об’ємів з кількістю гілок 1



Режим побудови ієрархії обмежуючих об’ємів з кількістю гілок 6



Режим освітлення та тіні для моделі



Змінення кольору моделі та активація режиму Кука-Торренса

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Полтавського Кирила Артемовича

(прізвище, ім`я та по батькові)

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна графіка і Web-дизайн»*

Керівник дипломного проекту (роботи) *Закроєв Юрій Михайлович*

(прізвище, ім`я та по батькові)

Тема дипломного проекту (роботи) *Розробка застосунку для редагування тривимірних графічних об'єктів*

Обсяг розрахунково-пояснювальної записки *89* сторінок

Обсяг графічної (презентаційної) частини *16* аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту завданню

Представлений дипломний проект відповідає затвердженій темі та виконаний відповідно технічному завданню. Дипломний проект присвячено розробці застосунку для редагування тривимірних графічних об'єктів і складається з пояснювальної записки та мультимедійної презентації

б) характеристика виконання кожного розділу дипломного проекту

Пояснювальна записка складається з основного розділу (Аналітичний огляд програмних засобів генерування тривимірної графіки; Огляд основних алгоритмів тривимірного моделювання; Вибір і обґрунтування засобів розробки; Розробка архітектури застосунку та ОО-моделі; Програмування і опис класів застосунку; Тестування розробленого застосунку), економічного розділу, розділу охорони праці та додатків. Перелічені розділи поетапно охоплюють розробку, виконані докладно та обґрунтовано.

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту

Графічна частина складається з 16 слайдів мультимедійної презентації, виконаної у програмному продукті MS PowerPoint, які містять структурні, та функціональні схеми, діаграми, скріншоти, блок-схеми алгоритмів, передбачені технічним завданням. Пояснювальна записка виконана акуратно та у відповідності до норм. Якість виконання пояснювальної записки відмінна, розробку виконано у повному обсязі.

г) перелік позитивних якостей дипломного проекту

Проєкт реалізує NURBS- та B-сплайни, фрактали, L-системи, розбиття на підрозділи (subdivision). Чітко винесені класи Mesh, Camera, Ray, BVH, GLShader/GLProgram, GLError, Vec3/4, що спрощує підтримку та розширення коду. Використання OpenGL для рендерингу + ієрархії обмежуючих об'ємів (BVH) забезпечує прийнятну продуктивність навіть на складних моделях.

д) основні недоліки дипломного проекту

Не подано замірів FPS, швидкості побудови BVH або часу підвантаження моделей. Зчитування й збереження лише OFF-файлів не дає змоги працювати з поширеними у 3D-індустрії OBJ, FBX, glTF тощо. Попри реалістичне освітлення, проект не передбачає роботу з UV-мапами та матеріалами (дифузійними/спекулярними картами).

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові рецензента *к.т.н. Шибаєва Наталя Олегівна*

Місце роботи і посада рецензента *Національний університет «Одеська політехніка», доцент кафедри інформаційних технологій*

Підпис:

« 20 » червня 2025 р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника на дипломний проект здобувача (здобувачки) освіти

відділення комп'ютерних систем

Полтавського Кирила Артемовича

(прізвище, ім`я та по батькові)

Спеціальність: *123 «Комп'ютерна інженерія»*

Освітня програма: *«Комп'ютерна графіка і Web-дизайн»*

Тема дипломного проекту: *Розробка застосунку для редагування тривимірних графічних об'єктів*

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ

а) обсяг і якість виконання проекту (графічного матеріалу і розрахунково-пояснювальної записки) *Дипломний проект виконано відповідно технічному завданню. Пояснювальна записка до дипломного проекту містить 89 сторінок. У пояснювальній записці виконано огляд та аналіз методів створення та редагування тривимірних графічних об'єктів, розроблено алгоритмічне та програмне забезпечення для редагування тривимірних графічних об'єктів. Графічна частина складається з 16 слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра.*

б) самостійність роботи над проектом: *Протягом виконання дипломного проекту здобувач освіти Полтавський Кирил поступово та послідовно виконував всі етапи, проявив ініціативу в створенні загальної концепції та реалізації роботи. Всі роботи здобувач освіти виконував самостійно, з оглядом на рекомендації керівника.*

в) теоретична підготовка випускника (випускниці): *Здобувач освіти Полтавський Кирил під час роботи над дипломним проектом вивчив достатньо багато літературних та інтернет-джерел за даною тематикою.*

Вважаю, що теоретична підготовка дипломника достатня і він готовий до захисту проекту.

г) вміння розв'язувати виробничі та конструкторські питання *Під час виконання дипломного проекту здобувач освіти Полтавський Кирил показав вміння організовано працювати над поставленим завданням, застосовувати знання у галузі комп'ютерної графіки та програмування, розробляти, встановлювати та налаштовувати спеціалізоване програмне забезпечення, користуватись сучасними засобами розробки, такими як QT, Microsoft Visual Studio C++*

Оцінка розрахункової частини *Відмінно*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім'я, по батькові керівника дипломного проекту *Закроєв Юрій Михайлович*

Місце роботи і посада керівника дипломного проекту *директор ТОВ «Біг ВОШ»*

Підпис

«16» 06 2025 р.

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
(ДИПЛОМНОГО ПРОЕКТУ)
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Полтавський К.А.,
здобувач освіти гр. 4КГ-08, та

Закроєв Ю.М.,
керівник дипломного проекту,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до дипломного проекту фахового молодшого бакалавра на тему:

«Розробка застосунку для редагування тривимірних графічних об'єктів» (автор роботи – Полтавський К.А., керівник роботи – Закроєв Ю.М.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2025 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Полтавський К.А. /

Керівник _____ / Закроєв Ю.М. /

«16» червня 2025 р.

ДОВІДКА

циклової комісії КТ та ПІ
про допуск до захисту дипломного проєкту
здобувача (здобувачки) освіти IV курсу
відділення комп'ютерних систем групи 4КГ-08

Полтавського Кирила Артемовича

на тему *Розробка застосунку для редагування тривимірних графічних об'єктів*

Висновок відповідальної особи за проведення нормоконтролю:
пояснювальна записка до дипломного проєкту виконана з несуттєвими порушеннями ДСТУ та оформлена відповідно до вимог Положення про дипломне проєктування

(підпис) *16.06.2025* (дата) *Петрашова В.І.* (П.І.Б.)

Висновок відповідальної особи за перевірку роботи на наявність академічного плагіату *згідно звіту про перевірку від 29.05.2025 р. значення коефіцієнту подібності в роботі становить 16,96%, коефіцієнт цитування – 0,98%.*

(підпис) *16.06.2025* (дата) *Краснокутська К.Г.* (П.І.Б.)

Попередня експертиза (малий захист) дипломного проєкту

здобувача (здобувачки) освіти *Полтавського К.А.* (П.І.Б.)

проведена « *16* » *червня* 2025 р.

Висновки *Пояснювальна записка до дипломного проєкту виконана у повному обсязі. Випускна кваліфікаційна робота (дипломний проєкт) відповідає вимогам Положення про дипломне проєктування та рекомендована до захисту.*

Голова ЦК КТ та ПІ (підпис) *Кривченко Ю.В.* (П.І.Б.)

StrikePlagiarism.com

Дата звіту 5/29/2025
Дата редагування 5/30/2025

Документ прийнятий

Звіт подібності

метадані

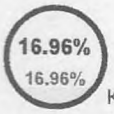
Назва організації
Odesa Technical Professional College of Odesa National University of Technology
Заголовок
Розробка застосунку для редагування тривимірних графічних об'єктів

Автор
Полтавський Кирил Артемович
Науковий керівник / Експерт
Закроєв Юрій Михайлович

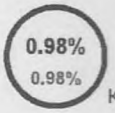
підрозділ
Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету"

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КЦ

25	16204	130199
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	23
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки	ß	0
Парафрази (SmartMarks)	a	112

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	103 0.64 %
2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content	76 0.47 %
3	https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content	60 0.37 %
4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	58 0.36 %

5	Система 3D моделювання [redacted] 3/15/2025 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)	57 0.35 %
6	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	57 0.35 %
7	https://card-file.ontu.edu.ua/bitstreams/5240e379-7721-49f0-8ee8-27140b0b473a/download	55 0.34 %
8	https://card-file.ontu.edu.ua/bitstreams/341a820e-d025-42f3-b7dc-27e831d6c66f/download	49 0.30 %
9	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	43 0.27 %
10	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	37 0.23 %

з домашньої бази даних (0.19 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Створення програмної моделі обробки графічних зображень рентгенівських знімків [redacted] 5/28/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	25 (2) 0.15 %
2	Створення web-застосунку цифрового помічника з використанням Open AI [redacted] 5/28/2025 Odesa Technical Professional College of Odesa National University of Technology (Відокремлений структурний підрозділ "Одеський технічний фаховий коледж Одеського національного технологічного університету")	5 (1) 0.03 %

з програми обміну базами даних (2.99 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Система 3D моделювання [redacted] 3/15/2025 **National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute)**	**430 (32) 2.65 %**
2	lab4.docx [redacted] 10/14/2020 University of International Business (UIB) (University of International Business)	22 (3) 0.14 %
3	Документ Microsoft Word.docx [redacted] 10/8/2020 University of International Business (UIB) (University of International Business)	14 (1) 0.09 %
4	Записка_МКР_Процик [redacted] 5/23/2025 National University "Lviv Politechnika" (NULP2)	12 (2) 0.07 %
5	Магістри П СП ІПСА 2024 [redacted] 12/13/2024 National Technical University of Ukraine Igor Sikorskyi Kyiv Politech Institute (ІПСА, К-ра системного проектування)	6 (1) 0.04 %

з Інтернету (13.79 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://card-file.ontu.edu.ua/bitstreams/1dff552d-7200-49b8-ae1d-ba76a1335685/download	645 (53) 3.98 %

2	https://card-file.ontu.edu.ua/server/api/core/bitstreams/ead3fa83-2e3d-4cd7-bfbd-1d5ed04c1ce4/content	368 (12) 2.27 %
3	https://card-file.ontu.edu.ua/bitstreams/aed610a6-43ef-47e0-9066-e85c89456f3e/download	201 (17) 1.24 %
4	https://card-file.ontu.edu.ua/server/api/core/bitstreams/995bdcec-4e4d-4321-8070-4d6badcb8e49/content	171 (5) 1.06 %
5	https://card-file.ontu.edu.ua/bitstreams/62baa43e-b968-4993-bb54-8cf8761a89b2/download	80 (6) 0.49 %
6	https://card-file.ontu.edu.ua/bitstreams/549ee9fe-7574-4ae5-b500-9fe2711f33e6/download	77 (6) 0.48 %
7	https://card-file.ontu.edu.ua/bitstreams/53ed22ad-8700-4162-b97a-082a1ad472d6/download	70 (4) 0.43 %
8	https://card-file.ontu.edu.ua/bitstreams/29489599-0581-4ce6-8890-c3b13d9f2e0e/download	57 (2) 0.35 %
9	https://card-file.ontu.edu.ua/bitstreams/5240e379-7721-49f0-8ee8-27140b0b473a/download	55 (1) 0.34 %
10	https://card-file.ontu.edu.ua/server/api/core/bitstreams/44c16132-5f53-48e2-b6c0-61e9a2f0fd75/content	55 (5) 0.34 %
11	https://card-file.ontu.edu.ua/bitstreams/341a820e-d025-42f3-b7dc-27e831d6c66f/download	49 (1) 0.30 %
12	https://card-file.ontu.edu.ua/bitstreams/bbaf3f38-16a8-4070-bead-5562769b7c71/download	49 (4) 0.30 %
13	https://www.mathematik.uni-marburg.de/~thormae/lectures/graphics1/graphics_8_1_ger_web.html	41 (4) 0.25 %
14	http://cpsm.kpi.ua/stud/bak/DP_BAK_KARAULOVA_LU.pdf	39 (2) 0.24 %
15	https://card-file.ontu.edu.ua/server/api/core/bitstreams/a141b658-5fa7-4f90-b0bd-7f0ccaed21e5/content	33 (3) 0.20 %
16	https://card-file.ontu.edu.ua/bitstreams/bbed74c8-2ea7-44c5-8d00-0fe3fd9790ee/download	31 (3) 0.19 %
17	https://zh.wikipedia.org/wiki/%E6%A0%B7%E6%9D%A1%E6%8F%92%E5%80%BC	30 (3) 0.19 %
18	https://card-file.ontu.edu.ua/server/api/core/bitstreams/4bb7255e-46d4-4349-9726-9698476da02d/content	27 (4) 0.17 %
19	https://card-file.ontu.edu.ua/bitstreams/c58b0ff5-46e0-49f8-8cbe-65c32256665d/download	26 (2) 0.16 %
20	https://card-file.ontu.edu.ua/bitstreams/0e72a3b9-bdd7-4711-a3c6-dedc1d4287cc/download	26 (3) 0.16 %
21	https://studopedia.su/7_8255_vimogi-elektrobezpeki.html	22 (2) 0.14 %
22	https://www.demo2s.com/cpp/cpp-vector-vector-lerp-const-vector-other-float-alpha-const.html	21 (2) 0.13 %
23	https://card-file.ontu.edu.ua/bitstreams/34a6756b-592f-4b77-a805-183aa03a6a26/download	17 (1) 0.10 %
24	http://genderi.org/microsoft-word-programlasdrma-dilln-v-informatikabenovse-doc.html?page=7	17 (1) 0.10 %
25	http://www.podlzwszach.net/rezultaty.php?nrzgl=79	8 (1) 0.05 %
26	https://card-file.ontu.edu.ua/bitstreams/e4afae26-0a7e-4a4d-afc2-94341838de2a/download	8 (1) 0.05 %
27	https://card-file.ontu.edu.ua/bitstreams/035f6436-20b4-4ee6-8e99-bede670e308b/download	7 (1) 0.04 %
28	https://ohoronapraci.com.ua/articles/666201-vydy-medychnykh-ohlyadiv-pratsivnykiv	5 (1) 0.03 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)