Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп`ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Інформаційна система для 3D-моделювання на основі OpenGL
Назва теми

КвРІСТ 190120.19.03.05 ПЗ
Шифр

Галузь знань 12 «Інформаційні технології»
Шифр, назва

Спеціальність 126 «Інформаційні системи та технології»
Шифр, назва

Освітня програма «Інформаційні системи та технології»
Назва

Виконав: студент IV курсу, група ІСТ-19-1 _____ А.С. Козира
Підпис Ініціали, прізвище

Керівник _____ К.Ю. Бобровнікова
Підпис, дата Ініціали, прізвище

Нормоконтролер _____ С.М. Лисенко
Підпис, дата Ініціали, прізвище

До захисту допускаю:
Зав. кафедри комп`ютерної інженерії та інформаційних систем _____ Т.О. Говорущенко
Підпис Ініціали, прізвище

«5» червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 126 ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ

Освітня програма «ІНФОРМАЦІЙНІ СИСТЕМИ ТА ТЕХНОЛОГІЇ»

ЗАТВЕРДЖУЮ

Зав. кафедри Т.О.Говорущенко

"11" 01 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Козирі Артуру Сергійовичу

Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Інформаційна система для 3D-моделювання на основі OpenGL.

Керівник проекту (роботи) Бобровнікова К.Ю., к.т.н., доцент

Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 01.03.2023 р. № 5

2. Строк подання студентом проекту (роботи) на кафедру 05.06.2023 р.

3. Вихідні дані до проекту (роботи) Завдання на дипломне проектування

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Дослідження інформаційної системи для 3D-моделювання на основі OpenGL та постановка задачі

Проєктування інформаційної системи для 3D-моделювання

Програмна реалізація та тестування інформаційної системи для 3D-моделювання

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Діаграма класів

Діаграма потоків даних

Візуалізація інструментів 3D-моделювання

6. Консультанти розділів дипломного проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Лисенко С.М., професор кафедри КІІС		
Антиплагіат	Нічепорук А.О., доцент кафедри КІІС		

7. Дата видачі завдання « 11 » 03 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	20.02.2022	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.03.2023	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	10.03.2023	виконано
4	Робота над розділом 2 – проєктування інформаційної системи	20.04.2023	виконано
5	Робота над розділом 3 – програмно-апаратна реалізація інформаційної системи	30.04.2023	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2023	виконано
7	Попередній захист ВКР	26.05.2023	виконано
8	Захист ВКР на засіданні ЕК	Червень 2023 року	

Студент _____ (Підпис) А.С. Козира (Ініціали, прізвище)

Керівник проекту (роботи) _____ (Підпис) К.Ю. Бобровнікова (Ініціали, прізвище)

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРІСТ 190120.19.03.05 ПЗ	Пояснювальна записка	59		
			Графічні матеріали			
2		КвРІСТ.190120.19.03.05 8Е	Діаграма класів	1		
3		КвРІСТ.190120.19.03.05 8Е	Діаграма потоків даних	1		
4		КвРІСТ.190120.19.03.05 8Е	Візуалізація інструментів для 3D-моделювання	1		

Зм	Арк	№ докум	Підпис	Дата	КвРІСТ 190120.19.03.05 ПЗ		
Розробив		Козирда			Відомість проєкту	Літера	Аркуш	Аркушів
Перевір.		Бобровникова				Н	1	65
Н. контр.		Лисенко				ХНУ. ІСТ-19-1		
Затв.		Говорущенко		05.06				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Інформаційна система для 3D-моделювання на основі OpenGL».

Автор роботи: Козира Артур Сергійович.

Керівник роботи: Бобровнікова Кіра Юліївна.

Пояснювальна записка: 66 с., 37 рис., 1 табл., 4 дод., 63 джерел.

Графічна частина: 3 креслення.

ІНФОРМАЦІЙНА СИСТЕМА, 3D-МОДЕЛЮВАННЯ, 3D-ГРАФІКА, OPENGL.

Метою роботи є проєктування та реалізація інформаційної системи для 3D-моделювання з використанням специфікації OpenGL для розробки додатків з використанням графіки.

Метою роботи є зниження системних вимог та забезпечення масштабованості системи для візуалізації 3D-графіки шляхом розроблення інформаційної системи для 3D-моделювання на основі OpenGL.

Об'єктом дослідження є процес 3D-моделювання з використанням специфікації OpenGL.

Предметом дослідження є інформаційна система для 3D-моделювання на основі OpenGL.

Практична цінність роботи полягає в розробленні інформаційної системи та розробленому на її основі програмному забезпеченні для роботи з тривимірною графікою та, зокрема, моделювання.

Підпис студента

05.06.2023

Дата

ЗМІСТ

ВСТУП

Стрімкий розвиток інформаційних технологій значно вплинув на формування сучасного життя. Технічний прогрес та автоматизація дозволили підвищити ефективність виробництва продукції, спростити процеси збору та обробки інформації.

Важливу роль відіграє візуальне представлення інформації. На сьогоднішній день існує чимало галузей людської діяльності, в якій використання графіки є необхідним або допомагає краще зрозуміти подану інформацію. Такими є, наприклад, реклама, мультиплікація, моделювання, створення інтерфейсу для ПЗ, обробка цифрових фотографій тощо.

Використання графічних елементів для рекламних кампаній пов’язують рекламований продукт чи послугу з виробником, його фірмовим логотипом завдяки асоціативним зв’язкам. Мультиплікація та анімація також можуть використовувати тривимірні моделі при створенні навіть 2D проєктів. Візуальні ефекти і тривимірну графіку також застосовують при виробництві фільмів за допомогою технології хромакей.

В комп’ютерних системах тривимірні об’єкти представлені в вигляді математичних моделей, наприклад, скалярні величини. Вершина, яка складається з трьох числових координат проектується на площину монітора, таким чином, створюючи ілюзію 3D-простору.

Проте розроблені моделі можуть бути використані для виготовлення деталей з пластику чи металу. Такі деталі мають безліч застосувань в промисловості (складні деталі та складові моделей збираються за готовою 3D-моделлю), прикладних проєктах, медицині тощо.

Безпосередньо самі цифрові моделі також мають застосування в візуалізації інтер’єрів, фасадів будинків, ландшафту та архітектурі в цілому. Завдяки обчислювальним можливостям процесора та відеокарти отримати бажаний ракурс, наприклад, для будинку, модель якого розроблена за кресленням, можна лише вказавши координати камери та напрям її погляду в програмі для візуалізації

тривимірної графіки, що, безумовно, є перевагою проти ручних креслень кожного окремого ракурсу на папері.

Візуалізація сьогодні швидкий процес. Розробити свою модель безкоштовно і зручно можна використавши безкоштовну програму для моделювання. Широка область застосування на сьогоднішній день визначає актуальність розробки додатку для роботи з графікою, зокрема, тривимірною.

Таким чином, актуальність теми роботи полягає в необхідності розробки ІС для 3D-моделювання, реалізація якої дасть можливість створення та рендеру в реальному часі 3D-моделей.

Завданнями роботи є:

- дослідити процедури функціонування системи 3D-моделювання та візуалізації;
- провести теоретичний аналіз сфери 3D-графіки;
- охарактеризувати структуру предметної області та базову модель системи 3D-моделювання;
- описати існуючі механізми реалізації, виділити наявні проблеми в галузі та шляхи їх вирішення;
- на основі проведених досліджень визначити основні функції системи, сформувати низку функціональних та нефункціональних вимог, розробити модель функцій, які система повинна виконувати;
- реалізувати інформаційну систему для 3D-моделювання на основі OpenGL;
- сформувати об’єкт та мету для наступних досліджень;
- оцінити ступінь виконання поставлених завдань.

Метою роботи є проєктування та реалізація інформаційної системи для 3D-моделювання з використанням специфікації OpenGL для розробки додатків з використанням графіки.

Об’єктом дослідження є програмне забезпечення для моделювання тривимірної графіки.

Практична цінність роботи полягає в розробленні інформаційної системи та розробленому на її основі ПЗ для роботи з тривимірною графікою та моделювання.

1 ДОСЛІДЖЕННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ 3D-МОДЕЛЮВАННЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

3D-моделювання – процес створення об’ємних тривимірних об’єктів за допомогою спеціалізованого ПЗ. Існує два основних види моделювання – полігональне та параметричне. Основними видами полігональних моделей є поверхневі (полігональні) та твердотільні.

Принцип роботи з полігональною моделлю полягає в редагуванні сукупності простих геометричних фігур, кожна з яких є окремим полігоном. Таку сукупність називають сіткою. Для підвищення якості до низькополігональних моделей можна застосувати теселяцію, тобто збільшити кількість полігонів.

Як правило за полігон беруть трикутник або чотирикутник, оскільки вони є досить простими фігурами, яких достатньо для однозначного задання площини і при цьому вони не потребують великих обчислювальних можливостей. Чотирикутникам надають приоритет у випадках, коли потрібно застосовувати анімацію до моделі, спростити роботу з моделлю в редакторі чи застосовувати теселяцію. В той же час, моделі на основі трикутних полігонів швидше обробляються відеокартою, що дозволяє розробляти складніші моделі.

Твердотільна модель не відображає порожнечу всередині моделі в редакторі. Її використовують при створенні моделей для 3D-друку чи виготовленні деталей у промисловості.

Параметричне моделювання представляє об’єкт в вигляді функцій, які визначають криві, по яким вибудовується його форма. При збільшенні масштабу такий об’єкт не втрачатиме точності, що є критичним параметром при виготовленні якісних моделей на підприємствах [1].

Оскільки розроблювана ІС використовуватиме полігональне моделювання, виявимо основні проблеми та завдання цього типу моделювання. Основними з таких є теселяція та тріангуляція.

При виготовленні відповідальних деталей для подальшого використання в більш складних системах фактор точності є критичним. Для таких завдань

використовують параметричний тип моделювання. Хоча ця проблема завжди властива полігональному виду моделювання, вона може бути вирішена за допомогою теселяції. Кілька проходів по моделі з рекурсивним розбиттям полігонів на кілька підрозділів можуть представити модель достатньої точності. Такий підхід змусить форму моделі прагнути до гладкості поверхні, як би це було з використанням параметричного моделювання. Але перепоною стане складність цього алгоритму, особливо при використанні трикутних полігональних сіток. Адже експоненційний ріст кількості полігонів швидко навантажить систему до меж її потужності.

Тріангуляція вирішує проблему закриття порожнеч в моделі, які можуть бути викликані з різних причин. Це не тільки помилка при розробці моделі, а й стандартні операції поділу на напівпростори, бінарні операції та інші. При тріангуляції інколи важливо мати можливість повторити топологію моделі максимально наближено, орієнтуючись на полігони, які знаходяться поруч з порожнечею. Але такі алгоритми непрості в реалізації і виявленні в них помилок. При використанні алгоритмів простої тріангуляції в одній двовимірній площині з'являється інша проблема – вкладеність полігональних сіток. Для її вирішення зручно використовувати структуру даних дерево і обробляти кожну полігональну сітку окремо [2].

Крім вузькоспеціалізованих, є проблеми загального характеру, властиві будь-якому ПЗ для 3D-моделювання:

- конвертація файлів;
- 3D-друк;
- системні вимоги.

Оскільки існує велика кількість програмного забезпечення для роботи з моделями у тривимірному просторі, важливо підтримувати однакову обробку одного й того ж формату різними додатками. Крім того, ліцензовані продукти можуть працювати зі своїми особистими форматами файлів, доступ до яких

неможливо отримати ззовні. Прикладом такого продукту є ліцензована бібліотека Tech Soft 3D HOOPS Visualize.

Моделі для 3D-друку повинні відповідати необхідним критеріям, таким як замкнутість, монолітність, цілісність. Це досягається правилами розробки 3D-моделі та спеціальними програмними інструментами для виявлення та усунення помилок моделі такого роду.

Вимоги до системи є досить важливим параметром при виборі будь-якого додатку. Проте навантаження програмного циклу додатку для роботи з 3D-графікою на систему набагато більше ніж, до прикладу, звичайний рендер інтерфейсу Windows Forms, оскільки величина обчислювальної інформації дуже об’ємна, особливо при роботі зі складними моделями. Оптимізація при розробці таких програм необхідна для оптимальної роботи без помилок.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

На сьогоднішній день існує досить велика кількість ПЗ для 3D-моделювання. Оскільки спектр застосування 3D-графіки досить широкий, такі додатки, зазвичай, мають певну спеціалізацію. Деякі програми краще підходять для розробки моделей і візуальних ефектів для комп’ютерних та мобільних ігор, фільмів та анімації. Інші дають можливість більш ефективно розробляти моделі для друку на 3D-принтері. При виборі готового рішення в першу чергу варто опиратись на поставлене завдання, яке необхідно виконати з його допомогою. Таким чином, можна скласти таблицю з порівняльними характеристиками різних додатків для роботи з 3D-графікою [3-6]:

Таблиця 1.1 – Порівняльні характеристики додатків для 3D-моделювання

Назва	Ціна	Тріал	Швидкість освоєння	Онлайн версія	Функціонал	Вимоги до системи
Blender	-	-	+	-	Повний	Низькі

Продовження таблиці 1.1

Назва	Ціна	Тріал	Швидкість освоєння	Онлайн версія	Функціонал	Вимоги до системи
SolidWorks 3D CAD	4195$	-	+-	+	Повний	Високі
ZBrush	1089$	-	+	-	Повний	Середні
Autodesk 3Ds Max	225$/міс	+	+-	-	Повний	Низькі
TinkerCad	-	-	+	+	Базовий	Низькі
Sculptris	-	-	+	-	Повний	Низькі
Cinema 4D	61€/міс	-	-	-	Повний	Середні

На основі даних, представлених в таблиці 1.1, можна зробити висновок, що існують різнобічні ПЗ для роботи з 3D-графікою. Вони суттєво відрізняються за ціною і простотою в використанні. Ціна та системні вимоги формують доступність, що потенційно впливає на кількість мануалів та форумів, які допоможуть в освоєнні та вирішенні проблем в ході роботи з продуктом. Натомість, ліцензоване ПЗ містить офіційну документацію користувача та технічну підтримку. Зазвичай, такі продукти є вузькоспеціалізованими і використовуються в компаніях та на підприємствах.

Важливим фактором є наявність тріалу – тимчасової ліцензії, як правило, безкоштовної, щоб спробувати продукт і зрозуміти, чи підходить він під цілі, яких необхідно досягнути з його використанням. Таку підписку видають на визначений термін також і студентам в навчальних цілях.

Повнота функціоналу визначає спектр можливостей ПЗ, обмежений його спеціалізацією. Безкоштовні продукти можуть містити базовий функціонал різних напрямків, наприклад, анімації, сумісткість з різними типами моделювання, підтримка скульптингу тощо. Базовий функціонал таких програм достатній для створення 3D-моделей для використання в цифровому вигляді.

1.3 Процес візуалізації об’єктів з використанням OpenGL

3D-модель є математичним представленням поверхні, яка її складає. В випадку полігонального моделювання, таким представленням є набір вершин в тривимірному просторі, які задають трикутники або інші багатокутники – полігони. Кожній вершині, полігону або окремій полігональній сітці можна задати певні властивості – координати, колір, матеріал. Матеріал впливає на результат змішування кольору полігональної сітки та кольору освітлення. Окрім цього, визначивши матеріал можна регулювати відбиваючу здатність поверхні, блискучість [7-10].

Полігональна сітка – основна структура, з якою взаємодіє ПЗ для 3D-моделювання. Проте, лише набір точок в просторі не формує підсумковий вигляд сукупності полігонів на екрані.

Візуалізація геометрії в OpenGL відбувається в нормалізованих координатах пристрою. Тобто координати обмежені значеннями від -1 до 1 і граничні значення знаходяться впритул до країв екрану. Всі координати вершин приводяться до такого вигляду в вершинному шейдері або всередині основного циклу рендеру. Відбувається така нормалізація з використанням матриць проекцій.

Оскільки фізично неможливо представити тривимірний об’єкт використовуючи плоский монітор, застосовують математичні перетворення, які дозволяють моделювати тривимірні об’єкти та виводити їх проекцію на площину.

Одним з етапів переходу скалярних даних до візуального представлення є застосування матриць до кожної вершини. Необхідно помножити вектор координат (одновимірну матрицю, а не геометричний об’єкт) на матрицю перетворень, отримавши в результаті нові координати точки, які використовуються в наступних перемноженнях з іншими матрицями перетворень. Таким чином, поступово відбувається перехід вершини в нові системи координат. Важливими для перетворень системами координат є [11-12]:

- локальний простір – визначається відносно початку моделі;
- світовий простір – визначається відносно деякої глобальної точки відліку, навколо якої будуються всі об’єкти;

- простір перегляду – визначається відносно точки та вектора, які імітують поведінку камери чи спостерігача;

- простір відсічення – застосовує матрицю проекції до вершини, при цьому визначає вершини в межах від -1 до 1;

- екранний простір – перетворює отримані в попередньому кроці координати до координат екрану.

Кожен етап перетворень має власне застосування. Наприклад, редагування однієї моделі зручніше виконувати в локальній системі координат, а редагування сцени, де потрібно враховувати відносне розташування полігональних сіток між собою – в світовій.

Наступним кроком до візуального представлення моделі є растеризація – процес, що перетворює точні математичні координати в наближене растрове зображення. Але, оскільки, пікселі екрану мають фізичні розміри, точність представлення об’єкта погіршуватиметься при масштабуванні, а саме – при зменшенні. З зображення конвеєра візуалізації OpenGL видно спотворення контурів фігури при растеризації (рис 1.2). Одним з рішень, яке пом’якшить ефект спотворення, є згладжування, процес якого відбувається на фінальному етапі обробки зображення перед зміною буферу кольору OpenGL [13].

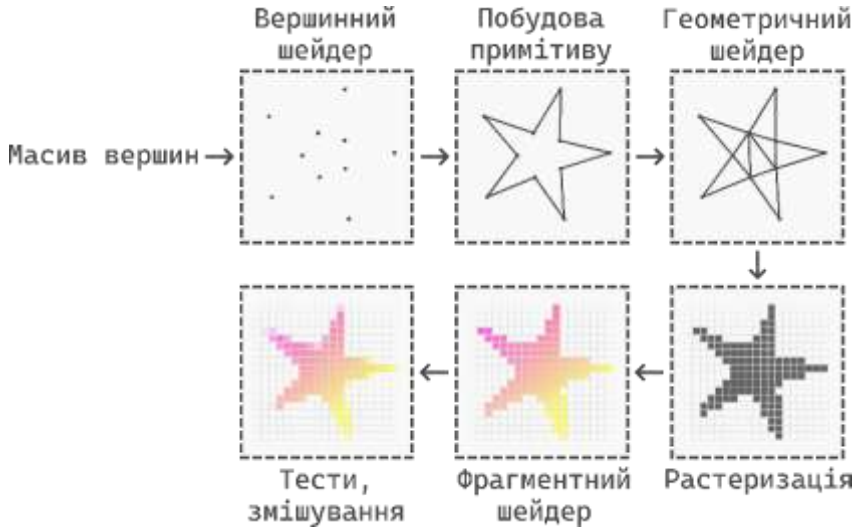


Рис. 1.2 – Конвеєр візуалізації OpenGL

1.4 Концепція полігонального моделювання

Полігональна сітка складається з трьох основних компонентів: вершин, ребер та багатокутників (зазвичай, трикутники та чотирикутники).

Вершина – одновимірний об’єкт, що визначає координати точки в просторі. Дві вершини визначають ребро.

Ребро – двовимірний об’єкт, що визначає відрізок в просторі. Три та більше ребер визначають багатокутник.

Багатокутник – плоский тривимірний об’єкт в просторі.

З багатокутників (полігонів) складаються полігональні сітки шляхом розташування мінімальної кількості полігонів в просторі таким чином, щоб візуально можна було розпізнати модельований об’єкт. Основне навантаження під час рендеру йде на відображення великої кількості полігонів, а не на розрахунки точних значень за формулами сплайнів, до прикладу, за рахунок чого такий вид моделювання вважається одним з найпродуктивніших, якщо не вимагається висока деталізація моделі [14].

Процес створення моделей відбувається з застосуванням операцій над вищезазначеними елементами. Окрім стандартних операцій, таких як переміщення та обертання, застосовні до будь-якого з перелічених елементів, існують додаткові операції, розроблені під кожен окремий елемент. В наступних підрозділах розглядаються основні операції над елементами полігональної сітки.

1.4.1 Операції над вершинами

Екструдування вершин дозволяє витягувати вершини полігональної сітки, створюючи додаткові вершини та ребра.

Формування скоса є одним з випадків екструдування, коли редагована вершина знаходиться на одній площині з вершинами полігональної сітки, формуючи рівну грань. При цьому як і в випадку екструдування формуються нові вершини та ребра, але ті, що ведуть до редагованої вершини і сама вона зникають.

1.4.2 Операції над ребрами

Екструдування та формування скоса також входять до операцій над ребрами. Окрім них, є операції стягування, розрізання та з'єднання. Стягування перетворює ребро в вершину, з'єднуючи дві його вершини в одну. Розрізання дозволяє розділити ребро на дві частини, з'єднавши точки розрізу новим ребром.

1.4.3 Операції над полігонами

Полігони підтримують частину попередньо описаного функціоналу, а саме: екструдування, формування скосу та стягування.

Таким чином, полігональна сітка являє собою певним чином взаєморозташовану групу полігонів, які формують цілісний об'єкт – модель. Оскільки полігональна сітка є окремим складним об'єктом на сцені рендеру, вона також підтримує здійснення деяких, описаних в наступних підрозділах, операцій над собою.

1.5 Визначення апаратних та програмних ресурсів для реалізації ІС для 3D-моделювання на основі OpenGL

Завдання проєкту: розробити ІС для 3D-моделювання на основі OpenGL у вигляді застосунку для ПК, в якому користувач матиме змогу здійснювати базові операції над об'єктами.

Проєктована інформаційна система відносно об'ємна і вимагає достатнього простору для розширення і введення нового функціоналу. Крім того, використовувані алгоритми програми повинні відповідати вимогам продуктивності, що підтримуватимуть високу швидкість роботи програми.

Таким чином, програмна реалізація передбачає велику кількість абстракцій і, як наслідок, об'ємний програмний код, складну архітектуру. При цьому важливо мати можливість підтримувати і оновлювати функціонал програми.

Платформа .NET містить потужні інструменти для розробки програмного забезпечення, включаючи додатки для ПК з ОС Windows. Специфікою складової CLR цієї платформи є менеджмент очищення зайнятої, але не використовуваної пам’яті, що оптимізує роботу програми [15].

Для реалізації ІС для 3D-моделювання було обрано наступні технології:

- мова програмування – C#, тому що об’єктно-орієнтований підхід для реалізації завдання на кваліфікаційну роботу підходить найкраще;
- клієнтська частина – WPF, тому що має простіший синтаксис для розмітки ніж WinForm, а також механізми для керування системою команд;
- контролер рендеру – WinForm UserControl, оскільки він призначений для створення власних елементів інтерфейсу.

Для використання OpenGL на платформі .NET до рішення також буде додано бібліотеку GLFW.NET, яка дозволяє керувати контекстом OpenGL та створювати власні вікна для рендеру графіки. Особливістю GLFW.NET від GLFW є її орієнтованість на вищезазначену платформу і надання програмного інтерфейсу на мові C# [16].

З розширень буде використано Extended WPF Toolkit від Xceed, який містить додаткові компоненти інтерфейсу, необхідні для спрощення взаємодії з користувачем (до прикладу, вибір кольору).

1.6 Визначення основних операцій 3D-моделювання

Трансформації [17-21]. Необхідно зазначити, що описані операції виконуються за допомогою афінних перетворень. Це такі перетворення, при яких зберігаються простір, паралельність та перпендикулярність прямих.

Транспортування – операція, що дозволяє переміщувати елементи полігональної сітки та її саму в просторі за вектором зміщення *T*. Для реалізації завдання на кваліфікаційну роботу необхідно скористатися матрицею переміщення, продемонстровану у формулі 1.3.

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1.3)$$

де T – вектор переміщення.

Обертання – операція, що дозволяє обертати полігональну сітку та її елементи в просторі навколо своєї осі, навколо центру сцени або навколо будь-якої іншої встановленої точки з кутом обертання θ та коефіцієнтами осей обертання R_x, R_y, R_z. Для майбутньої реалізації ІС для 3D-моделювання матриця обертання виглядатиме таким чином:

$$\begin{bmatrix} cos\theta + Rx2(1-cos\theta) & RxRy(1-cos\theta) - Rzsin\theta & RxRz(1-cos\theta) + Rysin\theta & 0 \\ RyRx(1-cos\theta) + Rzsin\theta & cos\theta + Ry2(1-cos\theta) & RyRz(1-cos\theta) - Rxsin\theta & 0 \\ RzRx(1-cos\theta) - Rysin\theta & RzRy(1-cos\theta) + Rxsin\theta & cos\theta + Rz2(1-cos\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1.4)$$

де θ – кут обертання;

R_x, R_y, R_z – коефіцієнти осей обертання.

Масштабування – операція, яка дозволяє збільшувати та зменшувати полігональну сітку, її елементи відносно певної точки за коєфіцієнтами S_1, S_2, S_3, що відповідають осям X, Y та Z. Причому, якщо ця точка не є центром редагованого об’єкта, змінюватиметься також відстань від об’єкта до точки, відносно якої здійснюється операція. Матриця масштабування, яка буде використана в реалізації завдання, має наступний вигляд:

$$\begin{bmatrix} S_1 & 0 & 0 & 0 \\ 0 & S_2 & 0 & 0 \\ 0 & 0 & S_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1.5)$$

де S – коефіцієнти масштабування за осями X, Y, Z.

Для представлення поточного інструмента трансформації часто використовують так звані gizmo. Вони мають певний вигляд і знаходяться разом з

точкою, відносно якої виконуються трансформації. Для транспонування це стрілки, обертання – кола, масштабування – квадрати. Причому вони напрямлені по визначеним осям і мають їхній колір (для осі X – червоний, Y – зелений, Z – синій).

Очевидно, що виконання великої кількості множень матриць для кожної операції трансформації має велику вартість. Для цього існує матриця TRS, яка дозволяє за одне перемноження виконати всі вказані перетворення. Для застосування перетворень необхідно перемножити кожну вершину об'єкта на матрицю, і такий порядок важливо зберігати, оскільки операція множення над матрицею не комутативна.

Отже, матриця TRS, використовувана в реалізації ІС, матиме наступний вигляд [22-23]:

$$\begin{bmatrix} a & b & c & d \\ e & f & d & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{1.6}$$

де $\vec{T}(d, h, l)$ – вектор переміщення;

$\vec{S}(|a, e, i|, |b, f, j|, |c, d, k|)$ – вектор масштабування.

Операція обертання декомпозується з матриці TRS у вигляді нової матриці:

$$\begin{bmatrix} a/S_x & b/S_y & c/S_z & 0 \\ e/S_x & f/S_y & d/S_z & 0 \\ i/S_x & j/S_y & k/S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{1.7}$$

де S – вектор масштабування.

Всі матриці трансформації для тривимірного простору мають розмірність 4x4. Це пояснюється тим, що для перетворень об'єкта без його змін в тривимірному просторі недостатньо матриці 3x3, а тому додається вектор з нулевими значеннями та 1 для параметра w. Цей параметр зазвичай відповідає за належність вершини до точки в просторі чи вектора, що не має конкретної позиції. Координати матриці

такого виду називаються однорідними і дозволяють виконувати перетворення над об’єктами незалежно від змін, утворених проєкціями [24].

Процес згладжування дозволяє збільшити роздільну здатність моделі і разом з тим її деталізацію. Але натомість додаткові елементи зроблять модель важчою, ускладнивши, до прикладу, процес розробки анімації. Правильне застосування згладжування дозволяє отримати більш деталізовану і гладку модель без гострих кутів під час її візуалізації. Повертаючись до анімації, згладжування можна застосувати на етапі рендеру, коли анімація вже готова, але буде працювати з більшою кількістю полігонів на виході.

Згладжування методом підподілу також має місце, коли під час розробки моделі необхідно збільшити кількість вершин, що деталізує модель та дозволить більш гнучко обробляти поверхню полігональної сітки.

В основі процесу згладжування лежить інтерполяція. Форма згладжуваного об’єкта стає більш обтічною та наближається до форм, визначених іншими, більш точними типами моделювання. Створюються додаткові проміжні полігони, які з’єднують вже існуючі між собою, таким чином відсікаючи гострі кути. Експоненційна складність алгоритму унеможливлює застосування алгоритму велику кількість разів, але зазвичай достатньо виконати її до трьох разів для отримання оптимального співвідношення деталізації та складності моделі. Процес згладжування за два кроки представлений на рисунку 1.8 [25-27].

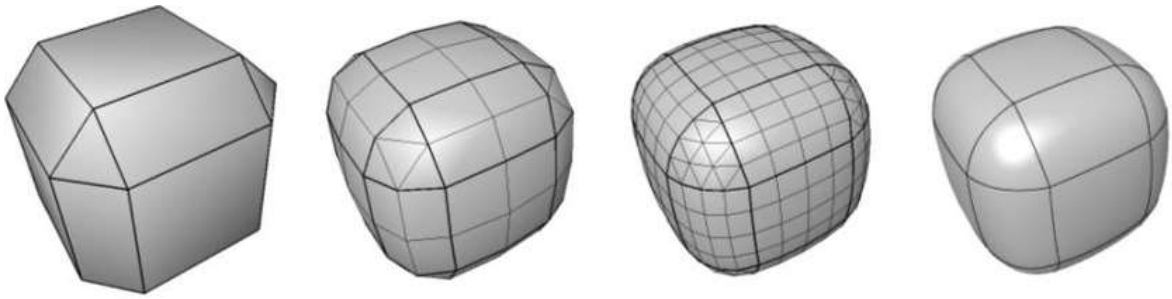


Рис. 1.8 – Процес згладжування полігональної сітки [26]

Як було зазначено, суть процесу моделювання полягає в створенні складних моделей, починаючи з простих геометричних тривимірних фігур. Наприклад, початковим об’єктом створюється куб чи сфера і модифікується рядом операцій,

які ускладнюють його форму. Гарним потенціалом володіють булеві операції (рис. 1.9), за допомогою яких можна отримати безліч різноманітних форм, застосовуючи їх до двох об’єктів, один з яких A, інший B. В залежності від того, над яким об’єктом виконується операція можна отримати різний результат. Булевими операціями є перетин (∩), об’єднання (∪) та різниця (\) [28].

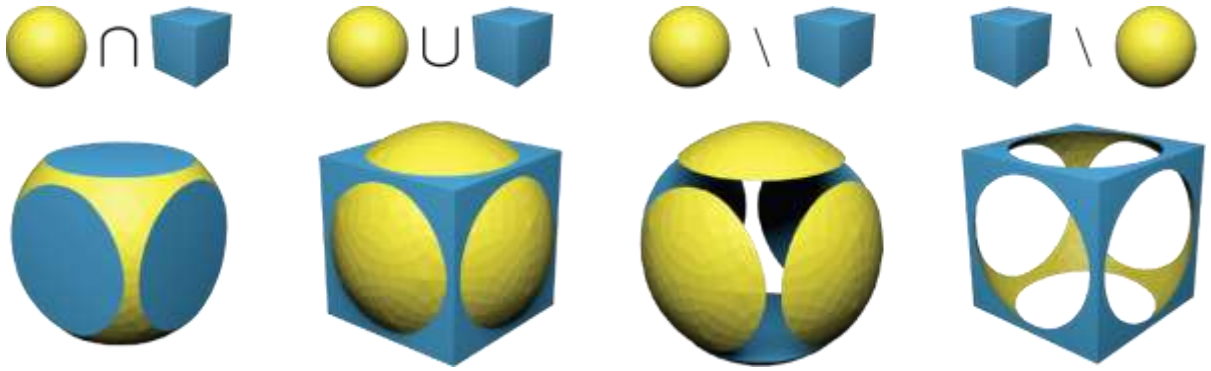


Рис. 1.9 – Булеві операції над полігональною сіткою [28]

Алгоритм відсікання Грейнера-Хормана базується на визначенні положення полігонів одного об’єкта відносно полігонів іншого.

На першому етапі потрібно визначити перетини між ребрами полігонів двох об’єктів. В точках перетину додаються нові ребра. Далі, використовуючи правило парності-непарності необхідно визначити приналежність вершини до одного з об’єктів і встановити прапорці приналежності з обох боків відносно створеного двома об’єктами кордонів. В результаті буде отримано поділений об’єкт, з якого можна обрати потрібну частину відповідно до обраної булевої функції.

1.7 Постановка задачі

Отже, необхідно реалізувати інформаційну систему для 3D-моделювання на основі OpenGL. Очевидно, що для створення моделей також необхідно інтегрувати візуалізацію графіки для більш наочного представлення моделей та підвищення зручності їх обробки.

Визначимо основний функціонал інформаційної системи для 3D-моделювання на основі OpenGL.

Робота з об'єктами. Об'єктами сцени є полігональна сітка, джерело освітлення та камера. Вони групуються в шари, які належать сцені. Складові полігональної сітки – трикутники (та складові трикутника – вершини) – також є об'єктами сцени. Кожен об'єкт має властивості та ім'я. Об'єкт можна приховати та показати. Властивостями об'єкта, в залежності від його типу, може бути, наприклад, позиція, колір, матеріал, налаштування освітлення.

Полігональна сітка може бути створена за допомогою конструктора, продубльована з існуючої полігональної сітки та завантажена з файлу. Додаток працює з розширеннями *.stl (binary / ASCII), *.obj. Полігональні сітки також можна експортувати в файл з одним з вищезазначених розширень. При цьому, якщо вибраним об'єктом є шар, він може бути експортований в один файл як одна полігональна сітка, або в кілька файлів окремо для кожної полігональної сітки, присутньої в шарі.

До полігональних сіток можна застосовувати основні трансформації – переміщення, обертання та масштабування. Застосування трансформації до шару застосує її до кожного її об'єкта з опцією, яка забороняє центрування. Тобто, таким чином, наприклад, обертання буде відбуватись навколо осі всього шару, а не навколо осі кожного об'єкта окремо.

Джерело освітлення в сцені присутнє як умовна точка. Крім позиції, воно може мати напрямок – в залежності від типу освітлення, яких є 3: напрямлене, точкове та прожекторове.

Рендеринг. Візуалізація відбувається з використанням бібліотеки GLFW – кросплатформна бібліотека для розробки з використанням OpenGL. В якості хостингу для цикла роботи бібліотеки використовується вбудований компонент WinForms UserControl, з якого вибудовується її контекст. Вона працює напряму з відеокартою через вказівник на комірку пам'яті, яка містить контролер компонента і надає доступ до стандартних функцій малювання.

Інтерфейс користувача. В користувацькому інтерфейсі сцена представлена деревоподібною структурою, яка дозволяє згортати та розгортати об'єкти, вибирати та застосовувати операції до об'єктів, наприклад, об'єднання кількох полігональних сіток в одну.

Крім візуального представлення тригерів команд додаток підтримує активацію команд комбінацією клавіш. Керування камерою відбувається за допомогою миші (та клавіатури в випадку вільного типу переміщення камери).

Типовий інтерфейс програмного забезпечення для 3D-моделювання зображено на рисунку 1.10.

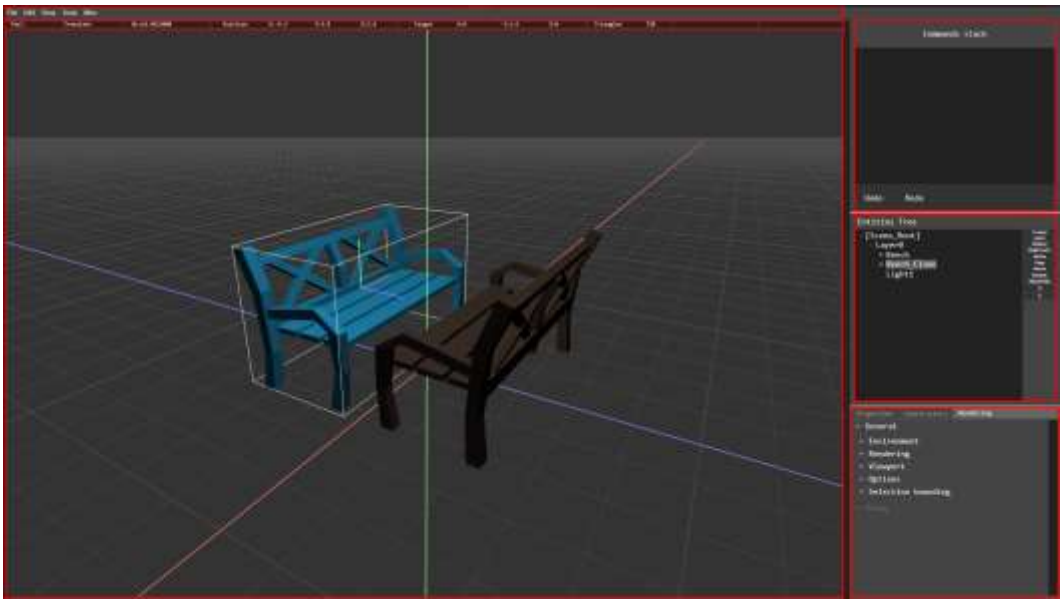


Рис. 1.10 – Типовий інтерфейс додатку для 3D-моделювання [29]

Червоною рамкою виділені основні елементи, які містяться в вікні клієнта:

Стрічкове меню з вкладками для роботи з файлами, редагування зовнішнього вигляду інтерфейсу, вибору інструментів, та кілька інших опцій (ввімкнення системи освітлення, фіксація освітлення за камерою, інструмент візуалізації структури напівребер тощо).

Рядок з інформацією про поточну позицію та напрям погляду камери, час рендеру фрейма, кількість трикутників на сцені, поточний інструмент; інша, обрана в налаштуваннях, інформація.

Вікно рендеру, в якому відбувається візуалізація математичних представлень об’єктів, а також застосовуються шейдери освітлення та матеріалів.

Стек викликів команд, за допомогою якого можна відслідкувати послідовність виклику команд, а також скасувати / повторити скасовану команду.

Дерево сцени – представлення сцени в вигляді вкладеного списку, який описує об’єкти сцени від шару до вершини, а також окремі елементи – джерело освітлення, камера.

Меню налаштувань, яке містить налаштування сцени, рендеру, конструктор полігональних сіток, властивості обраного об’єкта.

Для спрощення розробки створено план проєкту, в якому описаний базовий функціонал, послідовність та строки виконання. Переваги використання плану в ході розробки полягають в наочності прогресу виконання і відповідності запланованого та існуючого функціоналу додатку.

1.8 Висновки

В даному розділі роботи було здійснено теоретичний аналіз, вивчено недоліки існуючих рішень та сформовано задачу на дослідження шляхів їх вирішення. Відбулось ознайомлення з предметною областю 3D-графіки та моделювання.

З даних, представлених в розділі, можна зробити висновок, що основним завданням ПЗ для 3D-моделювання є оптимізація, оскільки візуалізація об’єктів досить ресурсозатратна задача для комп’ютера.

Було визначено основний функціонал, притаманний ІС для роботи з тривимірною графікою, та, зокрема, моделюванням.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ 3D-МОДЕЛЮВАННЯ НА ОСНОВІ OPENGL

2.1 Архітектура інформаційної системи для 3D-моделювання на основі OpenGL

Для отримання більш наочного представлення про структуру та функціонал ІС для 3D-моделювання на основі OpenGL, а також для демонстрації принципів її роботи були створені UML-діаграми.

Спроєктована ІС може бути представлена у вигляді моделі, графічне представлення якої подано на рис. 2.1

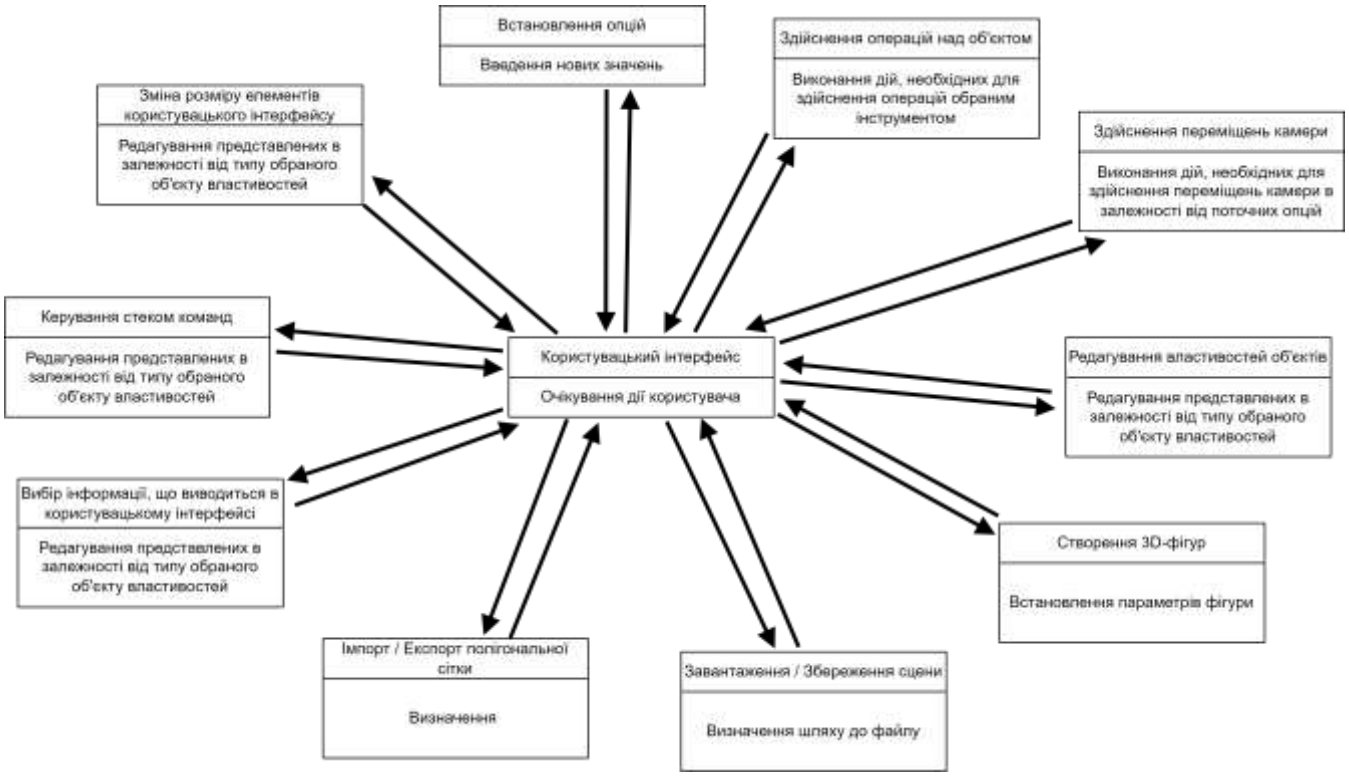


Рис. 2.1 – Модель ІС для 3D-моделювання на основі OpenGL

З метою визначення головних елементів, що відповідатимуть за виконання функціоналу, описаного в автоматній моделі, була створена діаграма компонентів (рис. 2.2).

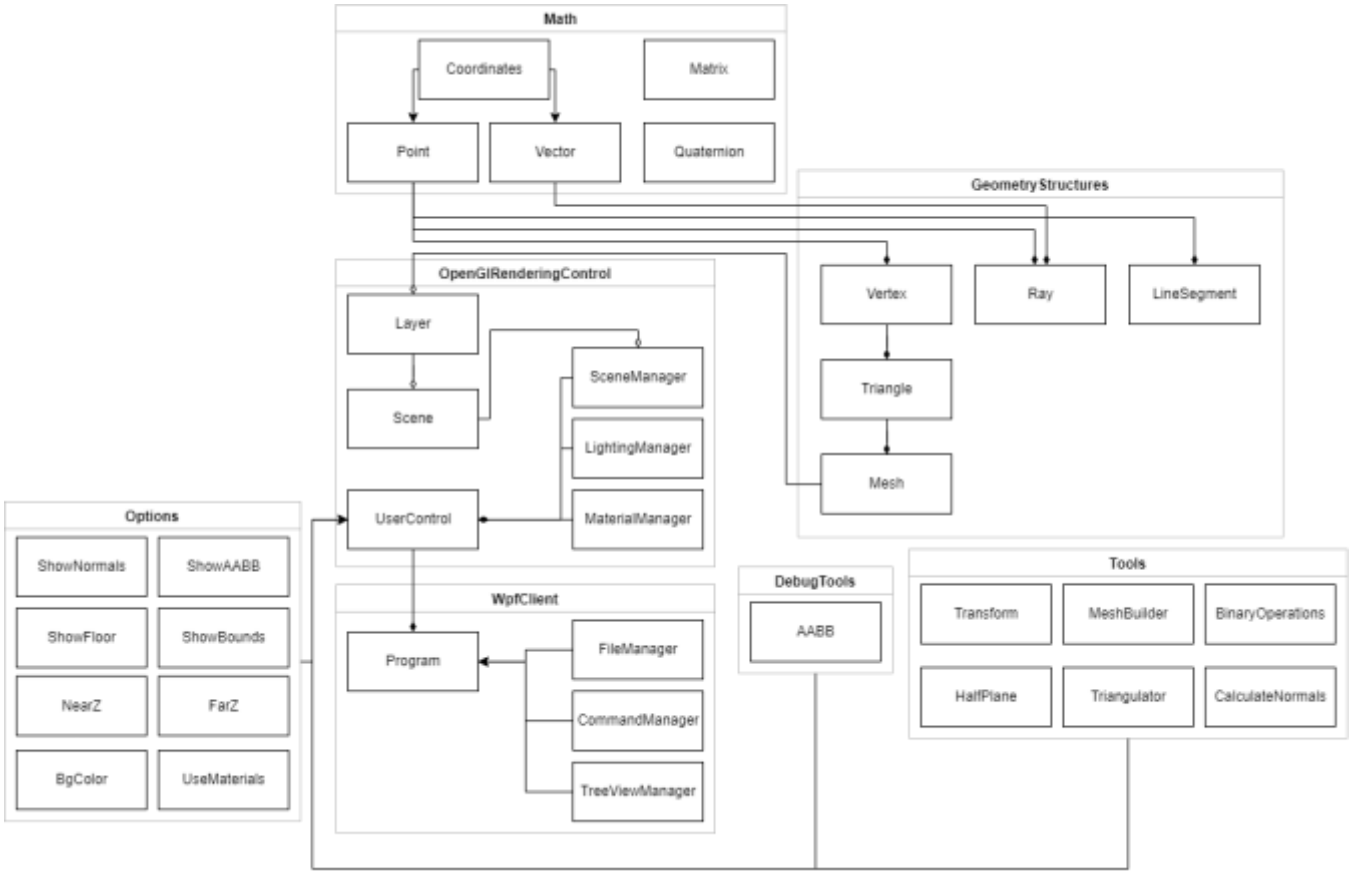


Рис. 2.2 – Діаграма компонентів ІС для 3D-моделювання на основі OpenGL

Спрощена структура ІС за діаграмою компонентів складається з:

- WpfClient – клієнт, з якого відбувається запуск системи. Містить користувацький інтерфейс, налаштування сцени, проєкту, інструменти для роботи з файлами, а також контролер рендеру графіки;
- OpenGlRenderingControl – контролер рендеру, який використовує бібліотеку OpenGL для візуалізації об’єктів. Використовує всі інші структури та класи, які необхідні для відображення сцени, налаштування її освітлення, присвоєння матеріалів об’єктам;
- Math – проєкт, що містить структури для математичних обчислень;
- GeometryStructures – проєкт, що містить основні примітиви, які можна візуалізувати та над якими проводяться операції, в тому числі, комплексну структуру полігональної сітки;
- Options – загальні опції проєкта;
- Tools – інструменти, які використовуються як всередині вихідного коду для обчислень, так і для здійснення операцій над об’єктами в системі.

З метою підвищення деталізації структури окремих компонентів було створено діаграму класів (додаток А), яка описує основні властивості та функціонал, необхідний для реалізації поставленої задачі. В процесі створення діаграми класів було додано кілька нових компонентів, що підвищать практичну цінність майбутньої реалізації.

До прикладу, ІС для 3D-моделювання повинна вміти працювати з файлами для збереження всієї інформації з якою вона працює. Сцена містить всю інформацію про розташування об’єктів, їх властивості та деякі налаштування, пов’язані з функціоналом та візуалізацією (напр. система освітлення). Таким чином, необхідно визначити структуру сцени в реалізовуваній ІС з метою створення підсистеми роботи з файлами. Групування об’єктів в сцені зображено на рисунку 2.3.

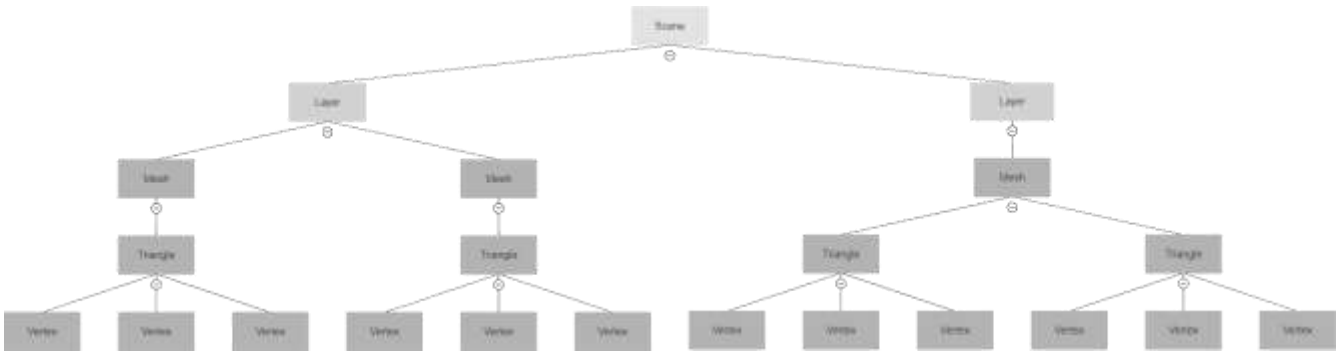


Рис. 2.3 – Структура об’єктів сцени

Так, компонент CommonStructures містить основні структури, які використовуватимуться в інших проєктах рішення. Це необхідний крок з огляду на особливості створення посилань між проєктами в середовищі розробки Visual Studio, а саме того, що посилання можуть бути виключно односторонніми. Діаграма класів компонента CommonStructures зображена на рисунку 2.4.

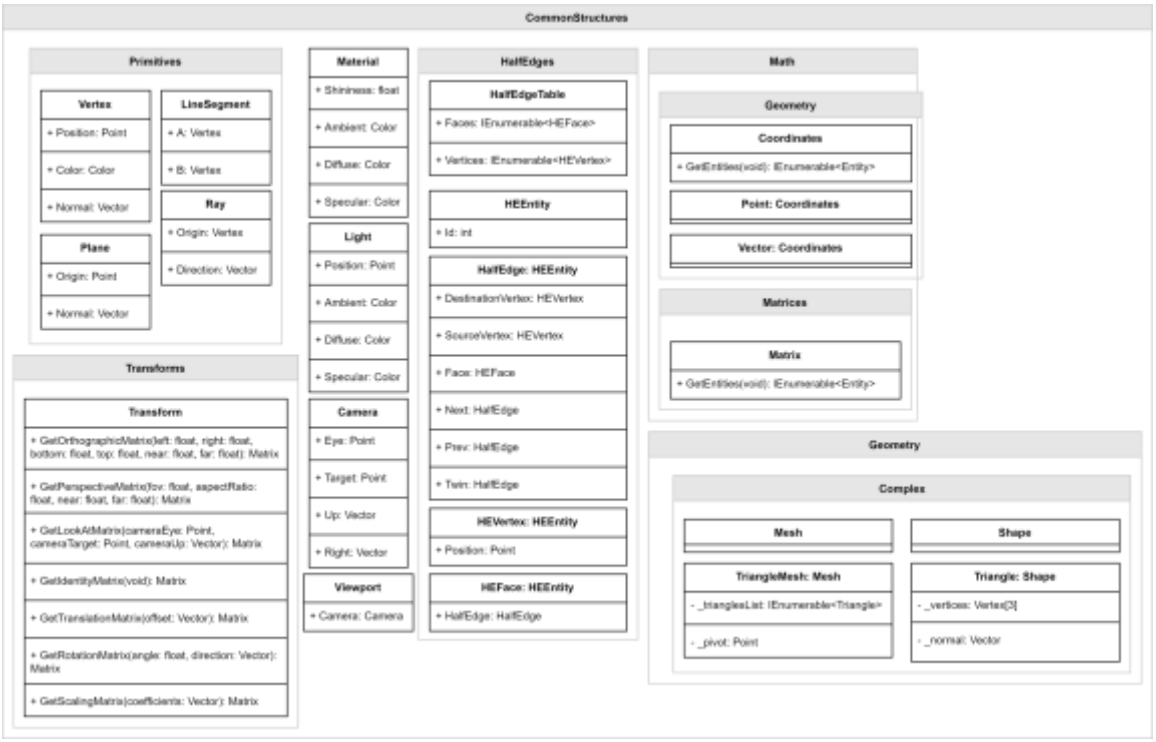


Рис. 2.4 – Діаграма класів компонента CommonStructures

Діаграма класів компонента WpfClient зображена на рисунку 2.5.

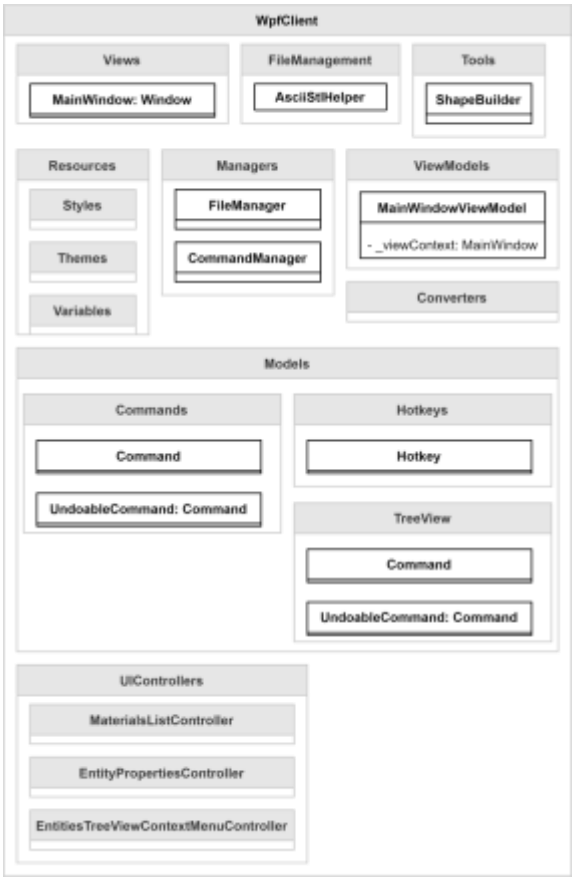


Рис. 2.5 – Діаграма класів компонента WpfClient

Компонент OpenGLCore, що відповідає за процеси візуалізації об’єктів сцени, зображений на рисунку 2.6.

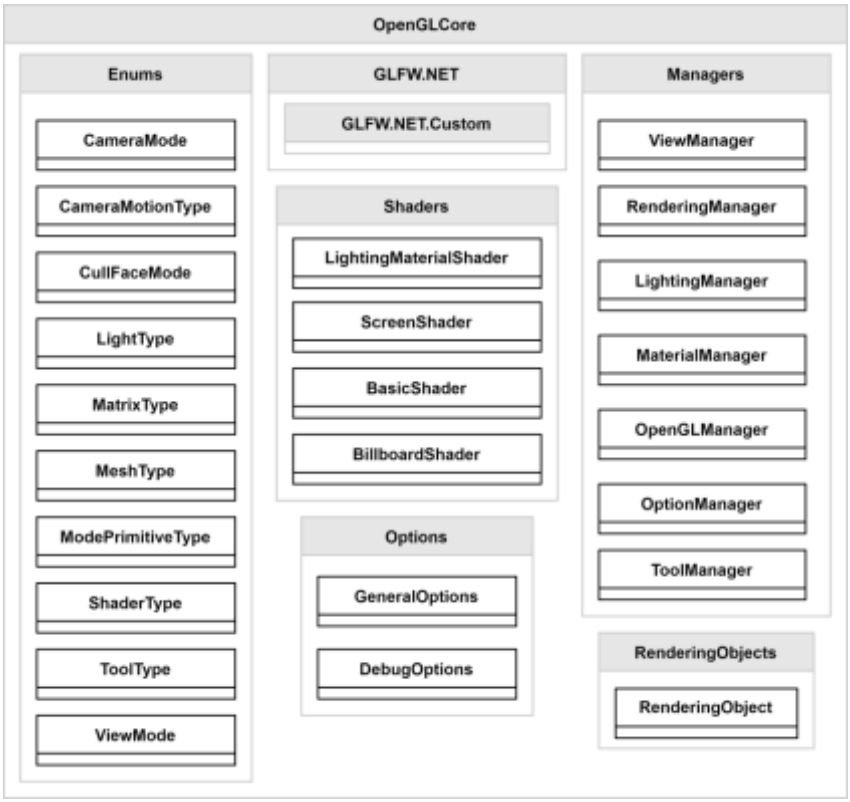


Рис. 2.6 – Діаграма класів компонента OpenGLCore

Допоміжні компоненти (рис. 2.7) були створені для спрощення процесу розробки та підтримки майбутньої реалізації ІС для 3D-моделювання на основі OpenGL завдяки вибудованій архітектурі інтерфейсів, реалізація яких надає змогу змінювати окремі компоненти без втручання до основного коду програмної реалізації.

З них виділяється компонент WorldObjects, який є важливим для реалізації завдання з огляду на необхідність структурованої подачі об’єктів сцени в інтерфейсі користувача (деревоподібний список), а також для зручної обробки даних сцени під час розроблення ПЗ. Об’єкт Entity виступатиме головною сутністю інформаційної системи, від якої наслідуватимуться об’єкти сцени, що дозволяє згрупувати їх для запису в файл. Оскільки реалізація надаватиме можливість роботи з файлами, така ієрархія класів спростить написання програмного коду для обробки об’єктів сцени.

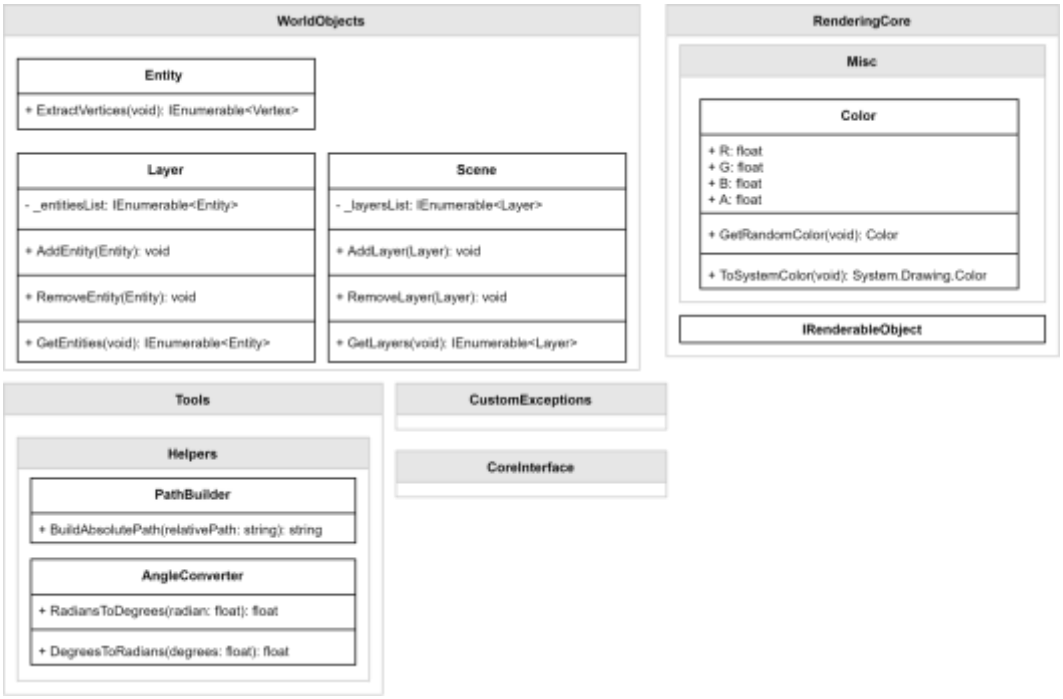


Рис. 2.7 – Діаграми класів допоміжних компонент

Діаграма варіантів використання зображена на рисунку 2.8.

Рис. 2.8 – Діаграма варіантів використання

Діаграма станів відображає зміну станів об’єктів в часі. Для визначення станів об’єктів користувацького інтерфейсу була створена окрема діаграма станів (рис. 2.9) для більш зручної орієнтації, оскільки користувацький інтерфейс цілком складається з набору опцій, перемикачів підсистем і функцій для роботи з файлами.

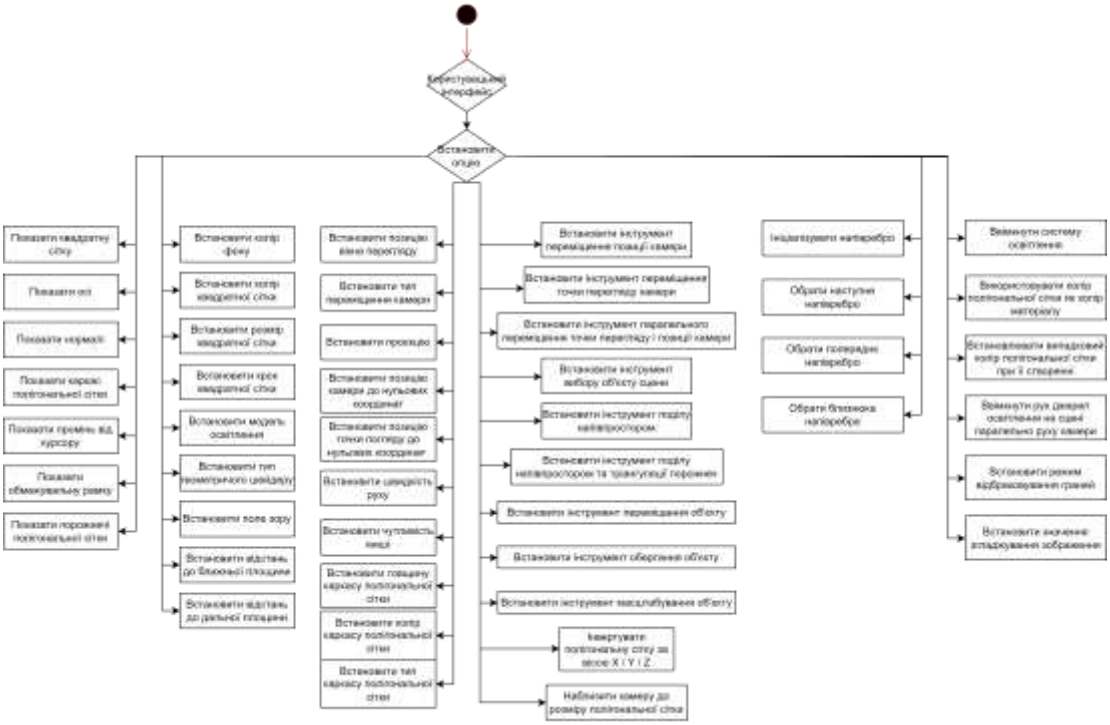


Рис. 2.9 – Діаграма станів користувацького інтерфейсу ПЗ

Діаграма станів для здійснення операцій над об’єктами в майбутній реалізації ІС для 3D-моделювання зображена на рисунку 2.10.

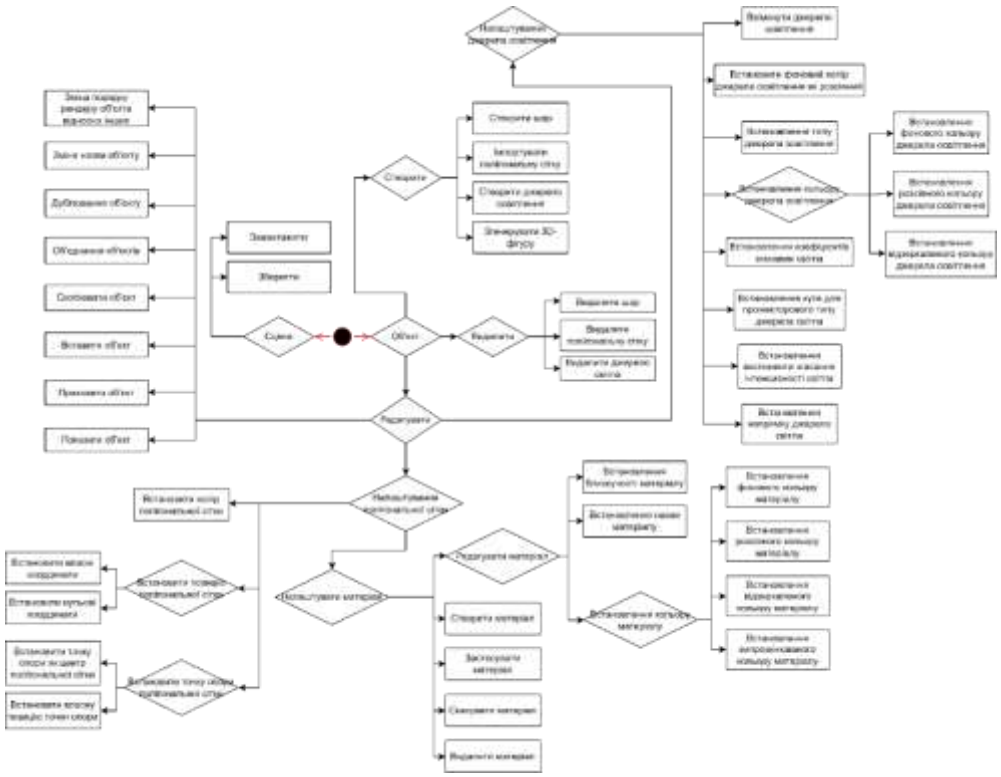


Рис. 2.10 – Діаграма станів здійснення операцій в реалізації ІС

Компоненти системи повинні вносити зміни в строго визначеній послідовності за вимогою принципів роботи OpenGL, оскільки при порушенні порядка підключень функціоналу чи встановлення значень до буферів функціонал бібліотеки візуалізації може викинути помилку чи просто не виконатись. До прикладу, для рендеру одного об’єкта поверх іншого потрібно вимкнути тест глибини, намалювати об’єкт і ввімкнути тест глибини. Порушивши цей порядок буде отримано результат, що відрізняється від запланованого [30-32].

Послідовність виконання ключових частин обробки та візуалізації об’єктів на сцені [32]:

1) ініціалізація початкових значень, ввімкнення необхідних функцій OpenGL;
2) початок циклу GLFW;
    2.1) передача об’єктів для візуалізації до списку менеджера рендеру;
    2.2) налаштування OpenGL;
    2.3) виклик метода рендеру;
    2.4) обробка подій;
3) виклик методу завершення програми.

Для більш конкретизованого представлення передачі даних між частинами програми була створена діаграма інформаційних потоків. На розробленій діаграмі інформаційних потоків (додаток Б) продемонстровано основний цикл обробки даних в інформаційній системі. Джерелом даних є список, розташований в пам’яті програми під час її виконання, до якого додаються об’єкти при генерації тривимірних фігур, при завантаженні сцени чи при використанні функції імпорту полігональної сітки.

В результаті моделювання ІС було отримано ключові для системи компоненти та функціонал. Для спрощення процесу розробки було створено план проєкту ІС для 3D-моделювання на основі OpenGL (рис. 2.11), в якому описаний базовий функціонал, послідовність та строки виконання. З плану робіт можна визначити головні етапи розроблення ІС та її реалізації, відправною точкою якого є отримання завдання на кваліфікаційну роботу. Переваги використання плану в ході розробки полягають в наочності прогресу виконання і відповідності

запланованого та існуючого функціоналу майбутньої реалізації інформаційної системи для 3D-моделювання на основі OpenGL [33].

Name	Duration	Start
Отримання завдання	1 day?	2/13/23 4:00 PM
Аналіз існуючих ІС	**1.5 days**	**2/14/23 11:00 AM**
Пошук кількох існуючих ІС для обраної предметної області	0.625 days	2/14/23 11:00 AM
Визначення основного функціоналу	0.25 days	2/15/23 8:00 AM
Визначення переваг та недоліків	0.25 days	2/15/23 10:00 AM
Створення таблиці з порівняльними характеристиками систем	0.25 days	2/15/23 1:00 PM
Висновок, визначення обов'язкового функціоналу	0.125 days	2/15/23 3:00 PM
Складання специфікації вимог до ІС	**5.75 days?**	**2/16/23 9:00 AM**
Загальний опис	**4.25 days?**	**2/16/23 9:00 AM**
Загальний опис ІС	1 day?	2/16/23 9:00 AM
Визначення середовища функціонування	0.5 days	2/21/23 11:00 AM
Визначення обмежень реалізації	0.5 days	2/21/23 4:00 PM
Складання діаграм	**1 day**	**2/22/23 4:00 PM**
UML-діаграма класів	1 day	2/22/23 4:00 PM
Здача ІС	0.125 days	3/3/23 4:00 PM
Розробка ПЗ	**30.75 days?**	**3/7/23 8:00 AM**
Створення скелету програми	**30.75 days?**	**3/7/23 8:00 AM**
Створення основних класів	2 days	3/7/23 8:00 AM
Підключення бібліотеки GLFW	0.5 days	3/9/23 8:00 AM
Побудова основного циклу програми	0.25 days	3/9/23 1:00 PM
Тестування	**0.25 days**	**3/10/23 8:00 AM**
Виявлення та виправлення критичних помилок	0.125 days	3/10/23 8:00 AM
Рендер простої фігури	0.125 days	3/10/23 9:00 AM
Реалізація складової рендеру	**8.5 days**	**3/10/23 10:00 AM**
Реалізація вершини	0.25 days	3/10/23 10:00 AM
Реалізація вектора	0.25 days	3/10/23 1:00 PM
Реалізація матриці	0.5 days	3/10/23 3:00 PM
Реалізація проекції	0.5 days	3/13/23 10:00 AM
Реалізація камери	0.25 days	3/13/23 3:00 PM
Реалізація вікна перегляду	0.25 days	3/14/23 8:00 AM
Реалізація мешу	1.5 days	3/14/23 10:00 AM
Реалізація матеріалу	1.5 days	3/15/23 3:00 PM
Реалізація джерела освітлення	1.5 days	3/17/23 10:00 AM
Реалізація структури напівграні	2 days	3/20/23 3:00 PM
Реалізація складової інструментів	**9 days?**	**3/22/23 3:00 PM**
Реалізація трансформацій	0.75 days	3/22/23 3:00 PM
Реалізація бінарних операцій	0.75 days	3/23/23 1:00 PM
Реалізація напівпросторів	1.5 days	3/24/23 10:00 AM
Реалізація тріангуляції	1.75 days	3/27/23 3:00 PM
Реалізація AABB	0.75 days	3/29/23 1:00 PM
Реалізація показу нормалей	0.5 days	3/30/23 10:00 AM
Реалізація готових мешей	**5 days?**	**3/28/23 3:00 PM**
Куб	1 day?	3/28/23 3:00 PM
Тор	1 day?	3/29/23 3:00 PM
Піраміда	1 day?	3/30/23 3:00 PM
Стрілка	1 day?	3/31/23 3:00 PM
Куля	1 day?	4/3/23 3:00 PM
Реалізація функціоналу програми	**10 days**	**4/4/23 3:00 PM**
Реалізація команд	3 days	4/4/23 3:00 PM
Реалізація роботи з файлами	5 days	4/7/23 3:00 PM
Реалізація дерева сцени	2 days	4/14/23 3:00 PM

Рис. 2.11 – План проєкту в ProjectLibre

2.2 Математична модель камери, області перегляду та проєкцій

Оскільки OpenGL не передбачає функціоналу камери, моделювати її поведінку потрібно з використанням власного об’єкту [34].

Результуюча проєкція на область перегляду залежатиме від кількох параметрів камери, які об’єкт камери буде містити в собі. Такими є:

1) позиція камери;
2) напрямок погляду камери;
3) вектор, напрямлений праворуч від камери;
4) вектор, напрямлений догори відносно камери.

Позиція камери в просторі визначається аналогічно точці, задаванням вектора координат камери.

Для знаходження вектора напрямку погляду камери необхідно встановити додаткову властивість камері – координати точки, на яку напрямлений її погляд. В результаті віднімання вектору цієї точки від вектора координат позиції буде отримано шуканий напрямок погляду. Проте, для позитивного значення координати Z в системі координат, визначеної матрицею області перегляду, необхідно поміняти ці вектори місцями, так як за замовчуванням в OpenGL камера напрямлена вздовж негативної осі Z.

Вектор вправо представляє позитивну вісь абсцис. Для його отримання потрібно перемножити вектор напрямку погляду на вектор, що вказує догори відносно світових координат.

Для отримання вектора догори необхідно помножити вектор напрямку погляду на вектор вправо. Всі отримані вектори нормалізовуються для уникнення спотворень значень, які можуть з’явитися при обрахунках з використанням цих векторів.

Маючи ці значення можна створити матрицю, яка перетворює простір для вершин зі світового на простір камери. Оскільки вона буде використана в реалізації завдання, необхідно визначити її формулу. Таку матрицю називають матрицею погляду камери і вона визначається за формулою:

$$\begin{bmatrix} R_x & R_y & R_z & 0 \\ U_x & U_y & U_z & 0 \\ D_x & D_y & D_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -P_x \\ 0 & 1 & 0 & -P_y \\ 0 & 0 & 1 & -P_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (2.12)$$

де R – вектор праворуч;

U – вектор догори;

D – вектор напрямку погляду камери;

P – вектор позиції камери.

Для здійснення обертання камери навколо точки достатньо змінювати її позицію за наступним псевдокодом:

```
x = camera.DistanceToOrigin * sin(camera.phi) * cos(camera.theta);
y = camera.DistanceToOrigin * cos(camera.phi);
z = camera.DistanceToOrigin * sin(camera.phi) * sin(camera.theta);
```

Значення phi та theta беруться з різниці координат при руху курсора миші. Таким чином, камера, завжди напрямлена на задану точку напрямку погляду, переміщується по уявній сфері навколо неї, з урахуванням відстані до неї. Одразу можна внести поправку на те, що координати точки перегляду не завжди будуть в центрі сцени, тому необхідно переміщувати вектор напрямку перегляду на нульові координати перед здійсненням обертання, після чого повертати її на вихідне положення.

Паралельне переміщення камери разом з вектором напрямку погляду реалізується наступним чином:

```
Vector zAxis = Vector.Normalize(Eye - Target);
Vector xAxis = Vector.Normalize(Vector.Cross(Up, zAxis));
Vector yAxis = Vector.Cross(zAxis, xAxis);
Eye.X +=
     xAxis.X * -offset.X + (offset.Y * yAxis.X);
Eye.Y +=
     yAxis.Y * offset.Y;
Eye.Z +=
     xAxis.Z * -offset.X + (offset.Y * yAxis.Z);
```

При цьому аналогічні операції як до вектора позиції камери потрібно застосувати і до вектора напрямку погляду.

Потрібно визначити коефіцієнти для осей (необхідно для врахування кута нахилу камери, що скоригує паралельність її руху відносно початкової позиції) і

застосувати однакові перетворення для координат позиції камери та точки, на яку вона напрямлена.

Переміщення камери навколо своєї осі схоже до переміщення навколо точки, з різницею в тому, що замість камери рухається точка, на яку напрямлений її погляд. Для коректного обертання також потрібно прибрати врахування відстані від камери до цієї точки [35].

Основними механізмами, які працюють для обертання навколо точки є параметри pitch та yaw, які відповідають за кут оберту камери по осям абсцис та ординат. Параметр roll використовується для додаткової осі оберту навколо осі аплікат, що може бути корисним при розробці авіа-симуляторів або іншого програмного забезпечення, що потребує довільного переміщення в тривимірному просторі.

Область перегляду захоплює ділянку сцени, яка буде відображена на дисплеї. Вона має параметри близької та дальної границь, фрагменти поза межами яких відсікатимуться на одному з етапів підготовки до візуалізації. Також, область перегляду може мати форму куба або зрізаного конусу, в залежності від обраної проєкції, що імітує перспективу чи ортогональність розміщення об’єктів відносно один одного, а також відносно камери.

Матриці проєкцій дозволяють імітувати певну поведінку об’єктів на двовимірній площині, представленій дисплеєм. Матриці проєкцій перспективи та ортогональності розглядаються далі.

Матриця перспективної проєкції, використовуваній в реалізації ІС, може бути представлена наступним чином [35-36]:

$$\begin{bmatrix} \frac{1}{aspect*\tan\left(\frac{fov}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{fov}{2}\right)} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2*far*near}{far-near} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad (2.13)$$

де *aspect* – співвідношення ширини та висоти області перегляду;

fov – поле зору;

far – відстань до дальньої площини відсічення;

near – відстань до ближньої площини відсічення.

Перспектива враховує параметр поля зору, значення якого чим вище, тим більшу область сцени захопить область перегляду, при цьому об’єкти будуть спотворюватися. Співвідношення сторін зображення описує формат зображення.

Матриця ортогональної проєкції, яка буде використана в реалізації ІС для 3D-моделювання на основі OpenGL виглядає таким чином:

$$\begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right+left}{right-left} \\ 0 & \frac{2}{\text{top}-\text{bottom}} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{-2}{far-near} & -\frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (2.14)$$

де *right* – координати правого краю піраміди огляду;

left – координати лівого краю піраміди огляду;

top – координати верхнього краю піраміди огляду;

bottom – координати нижнього краю піраміди огляду;

far – відстань до дальньої площини відсічення;

near – відстань до ближньої площини відсічення.

Ортогональна проєкція застосовується для перегляду сцени в вигляді креслення по одній з осей.

Загалом існує близько 5 систем координат, якими вершина в тривимірному просторі сцени поступово перетворюється на точку в двовимірному просторі дисплея. Цей поділ умовний і існує для зручності перетворень.

Типи систем координат [37-41]:

1) локальний простір – простір моделі з точкою відліку відносно її центру;

2) світовий простір – простір сцени, яка містить певним чином взаєморозташовані об’єкти;

3) простір камери – простір, що описує вигляд з позиції камери;

4) простір відсічення – простір, який враховує проєкцію;

5) екранний простір – перетворені до NDC координати на моніторі.

Для кожного з просторів існують свої матриці перетворень, які були описані вище. З їх допомогою можна в одному потоці програми відображати як рендер моделі так і елементи інтерфейсу, які завжди знаходяться в одному місці монітору.

2.3 Математичний опис кватерніону

Обертання за допомогою кутів Ейлера досить просте в реалізації, але має істотний недолік. При виконанні обертань положення об'єкта такий метод може виявитись непідходящим для повороту по певній осі, оскільки виникає явище, яке називають шарнірним замком. Основною проблемою виникнення такого явища є необхідність у зазначенні строгої послідовності осей при виконанні операції обертання, оскільки порядок прямо впливає на кінцевий результат. Для вирішення цієї проблеми можна використати в якості основи обертання кватерніони. Вони використовують комплексні числа для побудови векторного простору, які можна перетворити в матрицю обертання. З кватерніонами процес обертання виконується в будь-яку сторону з будь-яким кутом та легко інтерполюється, що може бути особливо корисним при створенні анімацій [40, 42].

Кватерніон складається з вектора та числа, а також з трьох додаткових уявних чисел, добуток квадратів яких рівний -1. Кватерніон для реалізації ІС виглядає так:

$$q = w + xi + yj + zk, \tag{2.15}$$

де w, x, y, z – дісні числа;

i, j, k – уявні числа.

Таким чином, кватерніон є складною структурою, яка містить комплексні числа, що ускладнює стандартні математичні операції над кватерніоном.

Операція повороту вектора, яка буде використана в реалізації завдання, за допомогою кватерніона виконується за наступною формулою:

$$q \cdot v = ((qi, qj, qk) \cdot v,\ (qx, qy, qz) \cdot v + (qi, qj, qk) \times v), \tag{2.16}$$

де *qi*, *qj*, *qk* – уявні числа;

v – вектор;

qx, *qy*, *qz* – дійсні числа.

Ця операція важлива, оскільки саме вона є основним застосуванням кватерніона для поворотів вершин, які можуть бути представлені у вигляді вектора.

Кватерніон також можна отримати з формули кутів Ейлера (формула 2.8), що буде корисним в майбутній реалізації поставленої задачі, оскільки повороти в ній здійснюватимуться за допомогою кутів Ейлера [42-44].

$$q = \begin{bmatrix} \cos(x) \cdot \cos(y) \cdot \cos(z) + \sin(x) \cdot \sin(y) \cdot \sin(z) \\ \sin(x) \cdot \cos(y) \cdot \cos(z) - \cos(x) \cdot \sin(y) \cdot \sin(z) \\ \cos(x) \cdot \sin(y) \cdot \cos(z) + \sin(x) \cdot \cos(y) \cdot \sin(z) \\ \cos(x) \cdot \cos(y) \cdot \sin(z) - \sin(x) \cdot \sin(y) \cdot cos(z) \end{bmatrix}, \quad (2.17)$$

де *x* – поділений навпіл коефіцієнт осі X;

y – поділений навпіл коефіцієнт осі Y;

z – поділений навпіл коефіцієнт осі Z.

В результаті буде одержано кватерніон виду $\vec{Q}$(w, x, y, z).

Кватерніон можна перевести до матриці обертання 4x4, яка є складовою матриці TRS, що застосовуватиметься в реалізації ІС для 3D-моделювання на основі OpenGL і має безпосередній вплив на матрицю моделі об’єкта:

$$M = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy - 2sz & 2xz + 2wy & 0 \\ 2xy + 2sz & 1 - 2x^2 - 2z^2 & 2yz - 2wx & 0 \\ 2xz - 2wy & 2yz + 2wx & 1 - 2x^2 - 2y^2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (2.18)$$

де *y*, *z*, *x*, *w* – дійсні числа.

2.4 Математичний опис проєкціювання курсора

Проєкціювання дозволяє користувачеві здійснювати операції над об’єктами з урахуванням перспективи, ортографічної проєкції тощо. Якщо функція, яка обробляє подію руху миші передбачає знаходження курсора всередині простору

модельованого середовища (наприклад, коли курсором потрібно вибрати об’єкт в просторі), необхідно імітувати цю поведінку. Для цього можна скористатися операцією проєкціювання курсора, яка переведе координати курсора до тривимірних і врахує поточну проєкцію. Щоб виконати проєкціювання курсора достатньо перемножити координати позиції курсора на матрицю проєкції.

Для вибору об’єкта на сцені потрібно створити промінь від курсора в напрямку погляду з врахуванням проєкції і визначити пересічення променя з найближчим об’єктом. Наприклад, щоб обрати полігональну сітку потрібно перевірити перетин променя з трикутниками на сцені, батьком найближчого буде обрана полігональна сітка.

2.5 Математичний опис функції поділу напівпростором

Напівпростором називають геометричну фігуру в просторі, яка включає в себе площину, яка містить напівпростір з однієї із сторін. Поділ сцени, полігональної сітки чи окремого полігона напівпростором дозволяє отримати два об’єкта з одного, відсікаючи тим самим, до прикладу, непотрібний елемент моделі.

Реалізувати таку поведінку досить просто. Потрібно визначити площину і її нормаль. Площину можна визначити трикутником, вершинами якого є позиція камери, точка в якій користувач натиснув кнопку миші і точка в якій користувач її відпустив. Щоб точки положення курсора не знаходились на одній прямій з камерою необхідно здійснити операцію проєкціювання над ними і перемістити на певну відстань за вектором руху, отриманого за допомогою матриці проєкції. Знаходимо нормаль одержаного трикутника векторним добутком двох отриманих за точками векторів [45-46].

Процес поділу відбувається за рахунок знаходження точок перетину площини з кожним окремим полігоном. До полігонів, в яких перетин відбувається в двох точках, додаються ребра, що з’єднують точки перетину і ділять полігон навпіл. В решті випадків полігон цілком потрапляє до одного з напівпросторів, залежно від положення відносно нормалі площини.

Необхідним кроком також є побудова додаткових граней, які повинні забезпечити зберігання полігонів в вигляді обраного багатокутника. В випадку, якщо багатокутником є трикутник можна скористатись тріангуляцією.

Після проведення операцій буде отримано два набори об’єктів, розташованих по різні боки площини напівпростору (рис. 2.19).

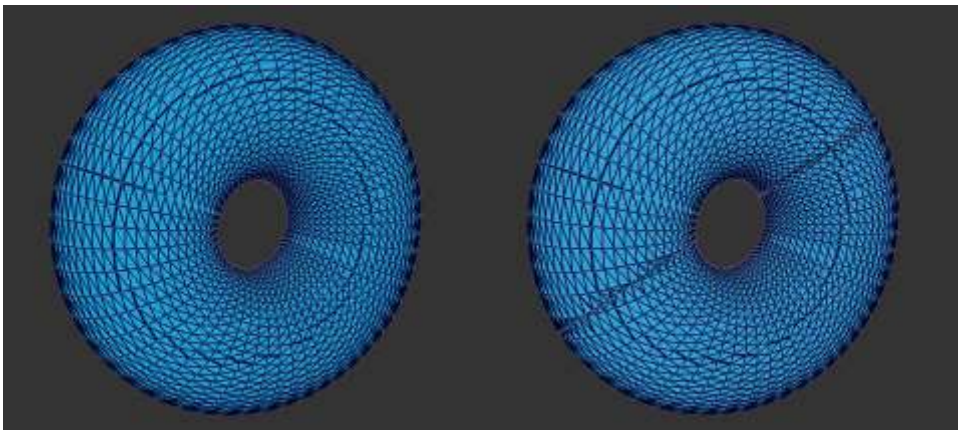


Рис. 2.19 – Ціла та поділена навпіл полігональна сітка тора [46]

2.6 Опис операції тріангуляції

Тріангуляція – процес розбиття геометричної фігури на трикутники. Тріангуляція використовується для закриття порожнеч в моделі. Більш розвинені алгоритми дозволяють імітувати топологію моделі, на відміну від описаних нижче алгоритмів, які закривають порожнечі плоскими фігурами.

Одним з найпростіших алгоритмів є ear clipping [47-48]. Його суть полягає в визначенні опуклих вершин полігона та їх відсікання. Орієнтовна блок-схема алгоритму одного з варіантів реалізації представлена на рисунку 2.20.

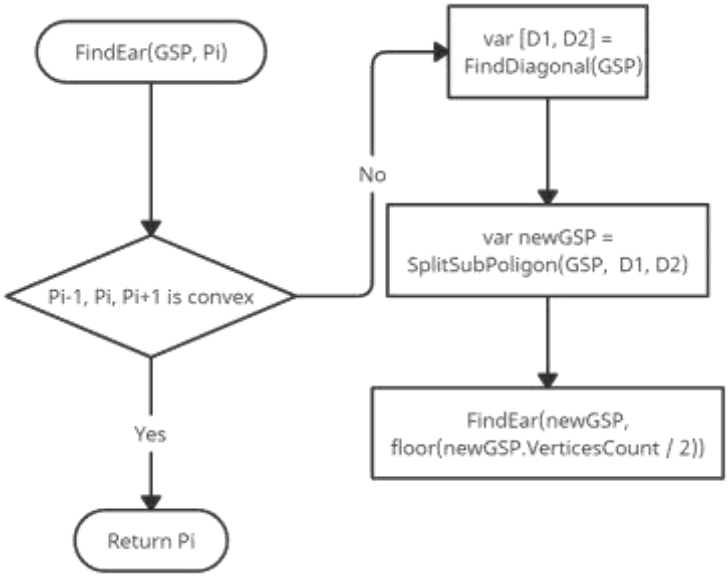


Рис. 2.20 – Блок-схема алгоритму ear clipping

Цей алгоритм має складність O(n).

В даному методі тріангуляції представлені такі сутності:

- вухо – опукла вершина полігона;
- простий підполігон – полігон, який є частиною всього полігона, що підлягає тріангуляції, і при цьому є простим;
- діагональ полігона – відрізок, що з’єднує дві вершини і не перетинається з іншими ребрами полігона.

Недоліками цього методу, окрім ігнорування топології, є погана якість тріангуляції та неврахування вкладеності полігональних сіток.

Якість тріангуляції визначається середнім значенням кутів всіх трикутників. Очевидно, що менше значення кутів означає наявність видовжених трикутників, включаючи трикутники з кутом 0°, які візуально не відображаються і над якими неможливо проводити операції. Також, як наслідок прямих послідовних з’єднань ребер між собою, відсутність проміжних вершин та ребер унеможливлюють редагування топології зони тріангуляції, дозволяючи зачіпати лише грані утвореного полігона.

Вкладеність полігональних сіток є важливим фактором, що впливає на процес тріангуляції. При спробі застосувати операцію до моделі тора, тріангуляція

зачепить максимально можливу територію, наклавши трикутники один поверх одного та повністю закривши западину разом з порожнечею.

Разом з тим, виявляється наступна проблема – відсутність контролю кожної окремої порожнечі, алгоритм вважає всі отримані точки одним полігоном. Хоча це не проблема тріангуляції, її необхідно враховувати під час реалізації ІС.

Покращений алгоритм представленого методу тріангуляції дозволяє обробляти вкладені полігональні сітки коректно, подаючи полігони і порожнечі в вигляді структури даних дерева. Кожен полігон може містити безліч порожнеч, які в свою чергу можуть містити безліч полігонів. Таким чином, можна скорегувати поведінку програми в залежності від поточного оброблюваного елемента.

Приклад заповнених порожнеч алгоритмом відсічення вух у розрізі полігональної сітки наведено на рисунку 2.21.

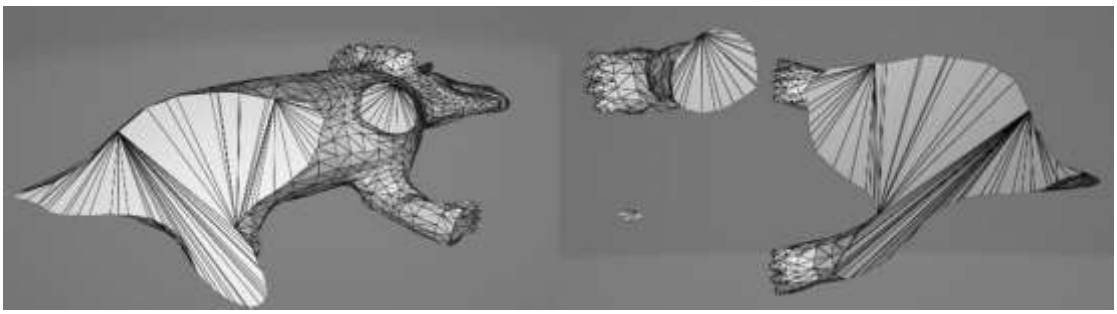


Рис. 2.21 – Тріангульовані полігони, отримані з перерізу полігональної сітки [49]

2.7 Опис структури напівребер

Структура даних напівребер дозволяє ефективно працювати з полігональними сітками, ігноруючи їхню топологію, створюючи планарний граф з її вершин та ребер [50-51].

Складовими частинами структури напівребер є:

- вершина, що містить єдиний вказівник на вихідне з неї напівребро, напівребер може бути будь-яка кількість;
- напівребро, яке містить вказівники на вершину його початку, попереднє та наступне напівребро, а також вказівник на протилежне напівребро (twin);
- грань, містить вказівники на формуючі його напівребра.

Вся структура напівребра, яка буде реалізована в програмному коді, міститиме списки її елементів, кожен з яких буде мати призначені відповідно вказівники на інші елементи.

Два напівребра одного ребра утворюють цикл, вказуючи на протилежні вершини. За рахунок цього, зовнішня ітерація навколо грані та внутрішня відбуваються проти та за годинниковою стрілкою відповідно (або навпаки). Такий механізм обходу граней дозволяє знаходити порожнечі в моделі. Використання структури напівребер дозволяє за O(1) визначити, чи з'єднані два ребра, а також здійснювати деякі операції обходу, описані нижче.

Ітерація навколо грані відбувається за наступним алгоритмом:

1) початкове напівребро зберігається в окрему змінну, яка відповідатиме за початкове напівребро;

2) це ж ребро зберігається в додаткову змінну, що відповідатиме за поточне напівребро;

3) здійснюється обхід, переходячи до наступного напівребра поки поточне напівребро не дорівнюватиме початковому.

Ітерація навколо вершини здійснюється майже так само, як і навколо грані, за виключенням останнього кроку. В залежності від того, в яку сторону здійснюється обхід (по часовій або проти часової стрілки) відбувається перехід до попереднього або до наступного напівребра, а потім до протилежного напівребра.

Алгоритм виявлення порожнеч за допомогою напівребер може бути представлений наступним чином:

1) обираються напівребра, у яких порожні вказівники на близнюків;

2) поки їх кількість більша нуля, необхідно визначити дві змінні, що вказують на те саме напівребро; одне з них, ітерується, додаючи до списку ребер порожнечі поточні напівребра та видаляється списку порожніх напівребер поки поточне напівребро не рівне початковому.

2.8 Математичний опис системи освітлення та матеріалів

Освітлення в OpenGL моделює фізичні процеси з реального життя за спрощеними формулами, якість яких, втім, достатня для сприйняття людиною. Далі розглядається модель освітлення Фонга.

Модель освітлення Гуро інтерполює значення освітленості поверхні між вершинами. Очевидним недоліком такого методу є погана якість згладжування світла на поверхні об’єкта при малій кількості полігонів. Однак при збільшенні роздільної здатності полігональної сітки якість наближається до якості моделі освітлення Фонга [52-57].

Для створення освітлення моделі Фонга необхідно сумістити три складові: фонове освітлення, розсіяне та відзеркалене. Фонове освітлення задає базовий колір об’єкта при відсутності освітлення. Розсіяне створює візуальний об’єм об’єкта, відтіняючи його рельєфну поверхню. Відзеркалене освітлення імітує блиск гладкої поверхні, і чим нижче його значення тим поверхня наближається до матової текстури. При визначенні освітлення важливим параметром є нормаль об’єкта, який підлягає освітленню. Для коректного відображення світла необхідно завжди застосовувати нормалізацію до векторів нормалей в процесі здійснення операцій над моделлю.

Для підвищення впливу кольору на освітленість об’єкта можна створити для нього матеріал з кількома властивостями, що представляють кольори окремих компонентів освітлення і множаться на значення кольорів освітлення відповідно до їх типів. Тобто, фоновий колір матеріалу множиться на фоновий колір освітлення.

Джерелами освітлення можуть бути різні математичні моделі. Імітація освітлення від сонця і від ручного ліхтарика відрізнятимуться. Тому потрібно розділити джерела освітлення за типами.

Напрямлене світло передбачає паралельність променів світла, що падають на об’єкти сцени, освітлюючи їх по всій поверхні з однаковою інтенсивністю, незалежно від їх розташування.

Точкове джерело світла випускає промені в усі сторони навколо себе, причому з відстанню яскравість освітлення згасає.

Величина згасання інтенсивності використовується для реалізації підсистеми освітлення в реалізації завдання. Її можна знайти за формулою:

$$F = \frac{1.0}{K_c + K_l * d + K_q * d^2}, \quad (2.22)$$

де K_c, K_l, K_q – константа, лінійний член і квадратичний член відповідно;

d – відстань від фрагменту до джерела світла.

Параметри K_c, K_l і K_q відповідають за згасання світла на відстані. Константа – мінімальне значення знаменника, зазвичай рівне одиниці, яке попереджає збільшення яскравості освітленості з відстанню. Лінійний і квадратичний члени зменшують інтенсивність з відповідною швидкістю.

Інтенсивність світла зменшуватиметься лінійно доти, поки відстань не стане досить великою для того, щоб значення квадратичного члена стало більшим за значення лінійного члена і швидкість згасання помітно зросла. Таким чином зі збільшенням відстані світло втрачатиме інтенсивність більш плавно. На рисунку 2.23 зображений приклад такого згасання.

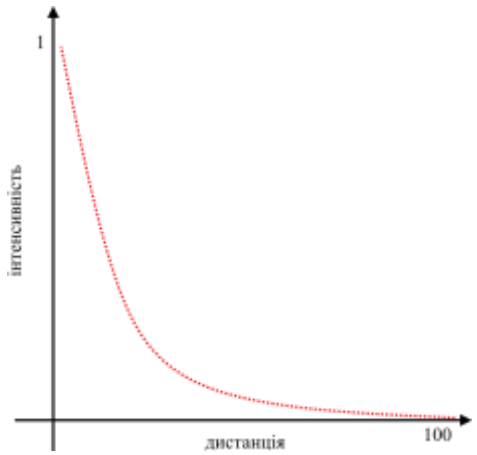


Рис. 2.23 – Графік згасання інтенсивності світла з відстанню [53]

Вибір значень залежить трьох параметрів залежить від характеристик освітлення, яке потрібно отримати в результаті. Залежно від оточення, яке потрібно змоделювати, рівня яскравості, потрібного для освітлення та інших параметрів потрібно підібрати значення. Або можна скористатися таблицею готових значень, підібравши звідти оптимальні для ситуації аргументи.

Джерело освітлення типу прожектора має вектор напрямку погляду, в якому розповсюджується світло за певним радіусом. Для більш плавного переходу на кордонах освітлення можна спроєктувати прожектор з двома конусами – внутрішнім та зовнішнім. Кут розповсюдження зовнішнього конуса повинен бути більшим за кут внутрішнього, за рахунок чого значення інтенсивності в проміжку різниці між конусами можна інтерполювати від 1 до 0, регулючи таким чином ширину лінії плавного переходу.

2.9 Опис елемента Axis-Aligned Bounding Box

За рахунок дешевизни обчислень, AABB активно використовується для виявлення зіткнень між об’єктами. Цей об’єкт являє собою паралелепіпед, що обмежує певну сукупність вершин в просторі. Він завжди розташований таким чином, щоб його поверхні були перпендикулярні відповідним осям координат. Проте, при зміні позиції обмежених вершин так, щоб вони виходили за межі паралелепіпеда, він мінімально змінюватиме свій розмір відповідно до вершин. Одним з варіантів реалізації AABB може бути побудова куба з довжиною ребра, рівною максимальному розміру обмежуваного об’єкта. Очевидно, що при такому способі обчислення втрачатиметься точність, але разом з тим зменшаться витрати на розрахунки.

2.10 Опис алгоритму побудови сітки

Координатна сітка в програмах для 3D-моделювання слугує орієнтиром для правильного взаєморозташування об’єктів на сцені.

Для створення сітки необхідно визначити кількість рядків та стовбців, якщо сітка не повинна досягати границь зони видимості, а також значення кроку між стовбців окремо необхідно визначити їхні позиції: для рядків координати нової точки будуть (-posX, 0, posY – (i * floorLineStep)) для першої точки та з позитивним posX для другої; для стовбців (posY - (i * floorLineStep), 0, -posX)), (posY - (i * floorLineStep), 0, posX) відповідно рядками та стовбцями сітки. За початкові

координати позиції береться кількість рядків або стовбців розділених на крок (posX, posY).

Також потрібно врахувати додаткові умови, до прикладу, уникнення Z-fight (зшивки) сітки з осями, якщо ті ввімкнені. Таке явище створює неприємні для ока мерехтіння об’єктів, що знаходяться на однаковій глибині для буфера глибини, і вони постійно намагаються перекрити одне одного під час рендеру. Для врахування такої умови в циклі перед створенням нових точок необхідно встановити умову, за якої при поточному значенні ітератора рівному центру сітки, а також ввімкнені осьові вказівники, ітерація пропускається.

2.11 Алгоритми побудови простих тривимірних геометричних фігур

Для генерації куба відносно заданої точки обирається довільний кут куба, який буде знаходитися на цій точці. Потрібно сформувати від точки два відрізки довжиною ребра куба, один з яких обернути навколо точки відліку донизу на 90°, після чого отриманий кут обернути навколо кінцевої вершини одного з відрізків. Сформований квадрат необхідно скопіювати і транспонувати на відстань довжини ребра квадрата і з’єднати додатковими ребрами з попереднім квадратом. Куб транспонується відносно точки на відстань в половину довжини його ребра відносно всіх осей таким чином, щоб задана точка знаходилася в центрі куба.

Для початку необхідно визначити, який з видів кулі необхідно згенерувати. Основними видами полігональних сіток, що представляють кулі є: ікосаедр, квадратна сфера і UV-сфера. Ікосаедр – сфера з трикутників. Квадратна сфера формується з куба, додаючи на кожній ітерації проміжні ребра та точки, буквально згладжуючи куб відсіканням гострих кутів. UV-сфера імітує меридіани та паралелі і при мінімальній роздільній здатності все ще імітує кулю, на відміну від квадратної сфери, початок якої формується кубом. Для побудови UV-сфери потрібно сформувати всі вершини кулі та згенерувати полігони окремо для верхнього та нижнього шарів (оскільки там використовуються трикутники) та для решти тіла кулі (незалежно від обраного полігона для тіла необхідно сформувати прямокутні грані).

Формування циліндра схоже до генерації сфери, для закриття верхньої та нижньої поверхностей будуються трикутники навколо центральної точки, а нижні ребра одержаних зверху та знизу трикутників з'єднуються прямокутниками.

Для побудови конуса достатньо прибрати з верхньої грані циліндра нижні ребра трикутників, з'єднавши їх напряму з центральною точкою.

Стрілка сформована з конуса і циліндра, в якому відсутня верхня грань, а ребра трикутників виступають місцем з'єднання нижньої грані конуса замість її центру.

Приклад згенерованих за частково представленими алгоритмами тривимірних фігур продемонстровано на рисунку 2.24.

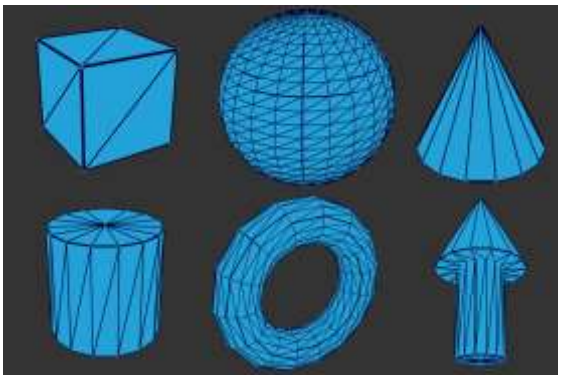


Рис. 2.24 – Скріншот згенерованих полігональних сіток [55]

2.12 Висновки

В даному розділі роботи було здійснено теоретичний аналіз різних елементів тривимірної графіки, визначено мінімальний функціонал, необхідний для обробки полігональних моделей, їх недоліки та переваги.

Було описано математичне представлення операцій над об'єктами полігонального моделювання та деяких процесів, властивих тривимірній графіці і, зокрема, специфікації OpenGL, таких як освітлення, проєкції та камера.

Розглянуто алгоритми побудови основних тривимірних геометричних фігур, які необхідно реалізувати в ІС для 3D-моделювання на основі OpenGL.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ ДЛЯ 3D-МОДЕЛЮВАННЯ НА ОСНОВІ OPENGL

3.1 Архітектура програмного забезпечення для 3D-моделювання

Структура програми загалом визначена структурою розробленої ІС за кількома винятками. До прикладу, створено додаткові проєкти в рішенні .NET, які спрощують процес розробки застосунку та його подальшого розвитку за рахунок використання класів-обгорток, що містять використовуваний у всіх проєктах рішення код.

Головним класом об’єкта для візуалізації є Entity, який наслідує інтерфейс IRenderableObject. Це дає більше контролю над об’єктами що передбачають можливість візуалізації, можливість побудувати більш абстраговану архітектуру, не прив’язуючись ко конкретного об’єкта, а також спростило створення дерева структури сцени (TreeView).

Точкою входу програмної реалізації є вікно з інтерфейсом WPF, яке ініціалізує стандартні значення, керує завантаженням контролера візуалізації і запускає подію початку цикла рендеру при повному її завантаженні.

Керування різними комплексними складовими програми відбувається шляхом використання системи менеджерів. Кожен менеджер проводить ініціалізацію власних змінних та змінних компонент, якими вони керують. Така абстракція спрощує програмний інтерфейс і дозволяє уникнути нагромадження коду та стандартних помилок через неправильне створення та ініціалізацію об’єкта.

З метою поліпшення масштабованості ІС логіка роботи інтерфейсу відділена від логіки процесів візуалізації за рахунок створення групи методів, які дозволяють їм обмінюватись необхідними даними. Такий підхід ускладнює обмін інформацією, оскільки доступ до властивостей одного компонента обмежений лише ним, але, разом з тим, дозволяє ПЗ проводити розрахунки без візуальної частини, звільнюючись від жорсткої прив’язки до конкретної реалізації бібліотеки рендеру. З точки зору ООП така архітектура позитивно сприяє інкапсульованості та абстрагованості між інтерфейсом та логікою роботи ІС.

Схожий принцип дії має проєкт RenderingCore, що цілком складається з інтерфейсів C#, які визначають головні точки процесу візуалізації. Припускається, що реалізація цього інтерфейсу може бути написана для різних бібліотек рендеру тривимірної графіки (наприклад, DirectX), що дозволяє їх змінювати не редагуючи внутрішній код, а додаючи інший проєкт, реалізуючий інтерфейс.

В проєкті присутні прості класи, які створюють абстракцію над логічно згрупованими діями. UIEventController абстрагує підписки на події в окремі методи. UIKeyController відповідає за обробку натиснутих клавіш.

EntityPropertiesController керує вмістом вкладки властивостей поточного об’єкта, відкриваючи доступ до певного набору функцій в залежності від його типу. Також, для полігональних сіток існує властивість матеріалу, який керується окремим контролером MaterialsListController.

EntitiesTreeViewContextMenuController аналогічно до попереднього визначає вміст контекстного меню дерева сцени. Приклад залежності вмісту контекстного меню від обраного об’єкта зображено на рисунку 3.1.

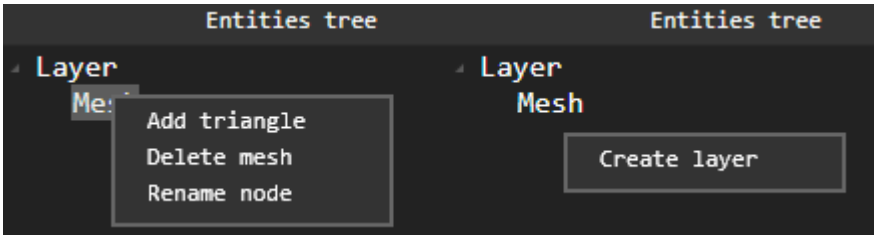


Рис. 3.1 – Демонстрація вмісту контекстного меню в залежності від цільового об’єкта

3.2 Інтеграція OpenGL до додатку на платформі .NET

OpenGL за принципом виконання схожа на скінченний автомат, тобто приймає на вхід певне число параметрів і на виході дає один результат для одного такого набору аргументів.

Для використання цієї технології в реалізації ІС на платформі .NET було встановлено бібліотеку GLFW.NET – обгортки для бібліотеки Glfw на мові

програмування C#, яка дозволяє керувати контекстом вікна рендеру, а також обробляти користувацький ввід з миші та клавіатури.

Щоб додати бібліотеку до проєкту потрібно завантажити файл glfw.dll з офіційного сайту і встановити обгортку GLFW.NET для проєктів, де вона використовуватиметься за допомогою пакетного менеджера NuGet. Далі, для основного проєкту потрібно визначити параметр архітектури процесора, оскільки їх неспівпадіння призведе до помилки.

Створимо контекст рендеру OpenGL в WinForm UserControl (далі – контролер рендеру):

```
Glfw.Init();
Window = Glfw.CreateWindow(800, 600, "", Monitor.None, Window.None);
_hwNative = Native.GetWin32Window(Window);
SetParent(_hwNative, _handle);
ShowWindow(_hwNative, 5);
Glfw.MakeContextCurrent(Window);
```

Ця частина коду ініціалізує бібліотеку, створює нове вікно і встановлює його як дочірнє до контролера рендеру. Передача вікна відбувається за допомогою його дескриптора, унікального для кожного вікна значення. В C# він має тип даних вказівника (IntPtr).

Щоб прибрати зайві елементи, властиві вікнам в ОС Windows, потрібно застосувати кілька стилів, які їх вимкнуть:

```
long style = GetWindowLong(_hwNative, -16);
style &= ~0x80000000;
style |= 1073741824;
SetWindowLong(_hwNative, -16, style);
```

Далі в батьківському вікні були створені обробники подій контролера рендеру, які змінюють відповідні параметри (наприклад, зміна розмірів вікна).

Під час роботи програми можна помітити рваність та миготіння зображення. Це спричинено тим, що OpenGL малює новий кадр прямо на екрані. Кількість кадрів на одиницю часу прямо впливають на видимість процесу малювання та скидання об’єктів на екрані. Для вирішення цієї проблеми можна ввімкнути систему подвійного буферу, яка віддає готове зображення з проміжної точки, що

складається з двох буферів. Одна з них містить візуалізовану сцену і подається на відеокарту для утворення зображення на екрані, в той час як інша використовується бібліотекою візуалізації для процесу малювання. Таким чином, користувач завжди буде бачити лише готовий результат. Бібліотека GLFW.NET за замовчуванням використовує подвійний буфер, тому ніяких додаткових дій для її використання не вимагається.

В процесі розробки додатку виникає необхідність у включенні нових опцій OpenGL до конвеєру обробки зображення. Так, після реалізації системи матеріалів з'являється потреба у прозорості кольору полігональної сітки (імітація скла). Опції бібліотеки рендеру визначаються після створення його контексту. Розглянемо використовувані параметри OpenGL.

Змішування вмикається викликом функції glEnable() з переданою константою GL_BLEND в якості аргумента. Для малювання прозорих об'єктів використовуючи змішування необхідно намалювати спочатку всі непрозорі об'єкти, відсортувати і намалювати прозорі. Сортування є невід'ємним кроком через специфіку роботи змішування в OpenGL – приймач має бути позаду джерела. Нижче наведено формулу розрахунку результуючого кольору на перетині двох об'єктів, яка була використана в реалізації підсистеми матеріалів:

$$\begin{bmatrix} R_r \\ R_g \\ R_b \\ R_a \end{bmatrix} \cdot R_a + \begin{bmatrix} S_r \\ S_g \\ S_b \\ S_a \end{bmatrix} \cdot (1 - R_a), \tag{3.2}$$

де R – приймач кольору;

S – джерело кольору.

Приклад відображення прозорих об'єктів на фоні інших прозорих та непрозорих об'єктів продемонстровано на рисунку 3.3.

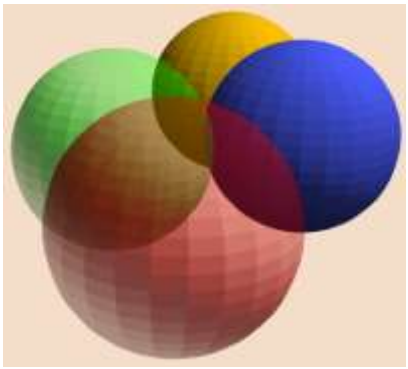


Рис. 3.3 – Прозорі та непрозорі матеріали полігональних сіток

Тест глибини спрацьовує після фрагментного шейдера і перевіряє дальність від екранного простору до кожного фрагмента. Якщо на позиції фрагмента стоїть кілька об’єктів, тест перезаписуватиме значення, доки не знайде найближчого. Для його активації необхідно викликати функцію glEnable(GL_DEPTH_TEST), а також оновлювати буфер глибини разом з буфером кольору в циклі рендеру, додавши новий прапорець GL_DEPTH_BUFFER_BIT. Тоді метод матиме наступний вигляд:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Відбраковуванням граней називається процес, при якому невидимі полігони не передаються до фрагментного шейдеру, тим самим оптимізуючи процес роботи програми. Невидимими полігонами вважаються ті, що мають протилежну орієнтацію відносно видимих. Визначається орієнтація за результатом визначення детермінанта матриці, в якій стовпці є вершинами трикутника. Поняття орієнтації трикутника схематично зображено на рисунку 3.4.

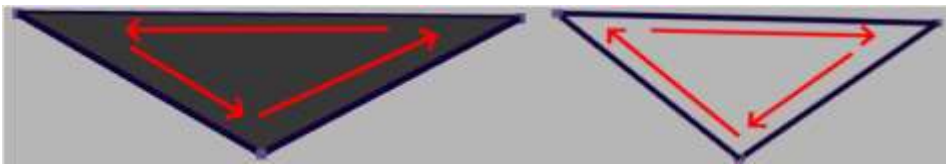


Рис. 3.4 – Орієнтація вершин полігона

Основний цикл OpenGL запускається з зовнішнього компонента (вікна WPF) в обробнику події завантаження контролера рендеру, оскільки для ініціалізації

контексту рендеру необхідно мати дескриптор вже існуючого і повністю завантаженого вікна.

3.3 Реалізація додаткових підсистем для покращення якості візуалізації

Вершини одержаних полігональних сіток можуть мати різний колір, проте цього недостатньо для створення ефекту тривимірного простору та комфортної роботи з моделями. Щоб надати моделям об’єм необхідно створити освітлення, яке відтінюватиме рельєфну поверхню полігонів. При цьому необов’язково створювати фізичну модель тіні. На рисунку 3.5 продемонстрована різниця між освітленою та неосвітленою сценою.

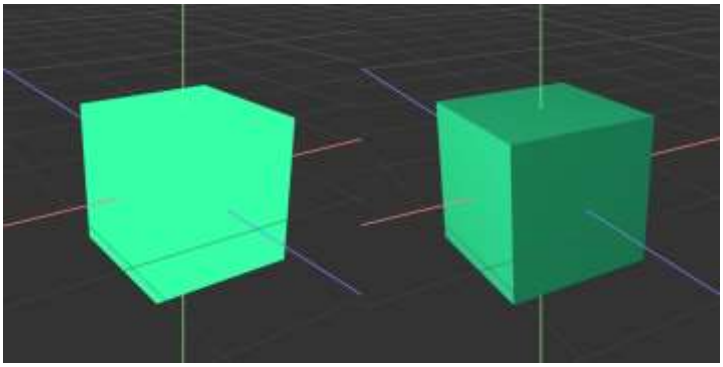


Рис. 3.5 – Сцена без джерел освітлення та сцена з одним джерелом освітлення

Для застосування освітленості до поверхні полігональних сіток в OpenGL використовують шейдери. Шейдером називається програма, яка виконується відеокартою в процесі растеризації об’єктів.

OpenGL дозволяє працювати з фрагментним шейдером за допомогою шейдерної мови GLSL (OpenGL Shading Language). Ця мова підтримує базовий функціонал роботи з векторами та матрицями. Код, що описує шейдерну програму мовою GLSL виконується відеокартою. Файли з розширенням *.vert та *.frag підключаються до основної програми, їх код містить вхідні та вихідні параметри, а також універсальні параметри, які можна передавати минуючи вершинний шейдер прямо до фрагментного шейдера.

Після визначення координат вершин, їх кольорів та будь-яких інших даних, необхідно передати їх до вершинного шейдера. Для цього необхідно виділити пам’ять на відеокарті, де вони будуть збережені та виділимо головні елементи, оскільки дані в основному представлені в вигляді масивів. Об’єкт, що містить дані про організацію елементів в буфері вершин називається об’єктом масиву вершин.

Відбувається така організація наступним чином.

Є масив з 9 елементів, кожен з яких є числом (рис. 3.6).

```
[1, 2, 1, 0, 2, 0, 3, 1, 3]
```

Рис. 3.6 – Масив з 9 елементів

Кожні 3 елементи представлятимуть координати точки в просторі (рис 3.7).

```
[1, 2, 1, 0, 2, 0, 3, 1, 3]
```

Рис. 3.7 – Представлення резервування індексів в VBO

Звідси можна отримати координати трьох вершин.

В одному масиві можна вказувати значення для кількох різних параметрів. Інтепретація масиву відбувається завдяки створенню атрибутів, які визначають:

- початковий індекс, з якого починається читання масиву;
- кількість елементів, які необхідно прочитати послідовно для отримання даних одного об’єкта;
- кількість елементів, які необхідно пропустити до наступного фрагменту масиву що містить дані про об’єкт, визначений атрибутом.

Прикладом інтерпретації координат вершини та її кольору показано на рисунку 3.8.

```
[5, 5, 5, 0.1, 0.2, 0.3]
```

Рис. 3.8 – Представлення резервування індексів в VBO

Для того, щоб вказати шейдеру як сприймати ті чи інші дані, потрібно пов’язати масив з вершинними атрибутами. Необхідно визначити розмір в байтах одного числа, зміщення відносно початку масива, крок і кількість зчитуваних чисел масива для кожного атрибута.

Код створення буферів вершин:

```
VAO = OpenGL.glGenVertexArray();
VBO = OpenGL.glGenBuffer();
```

Буфери вершин необхідно прив’язати до об’єктів масиву вершин OpenGL, створивши об’єкт буферу вершин та визначивши його розмір:

```
OpenGL.glBindVertexArray(VAO);
OpenGL.glBindBuffer(GlfwConstants.GL_ARRAY_BUFFER, VBO);
float[] verts = PointsListToArray(ObjectVertices);
fixed (float* v = &verts[0])
{
    OpenGL.glBufferData(GlfwConstants.GL_ARRAY_BUFFER, sizeof(float) *
verts.Length, v, GlfwConstants.GL_STATIC_DRAW);
}

OpenGL.glVertexAttribPointer(0, 3, GlfwConstants.GL_FLOAT, false, 3 *
sizeof(float), (void*)0);
OpenGL.glEnableVertexAttribArray(0);
```

Далі розглядається робота складових освітлення на прикладі шейдерного коду. Файл ../LightingMaterialShader/shader.frag містить код фрагментного шейдера, який обробляє джерела освітлення і матеріали. Всі наступні дії виконуються в циклі, доки значення *i* менше кількості джерел світла.

Код для обробки фонового освітлення:

```
vec4 ambient = Lights[i].AmbientColor * AppliedMaterial.AmbientColor;
```

Код для обробки розсіяного світла:

```
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(Lights[i].Position - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec4 diffuse = diff * AppliedMaterial.DiffuseColor *
Lights[i].DiffuseColor;
```

Схематичне зображення розрахунку розсіяного світла продемонстровано на рисунку 3.9.

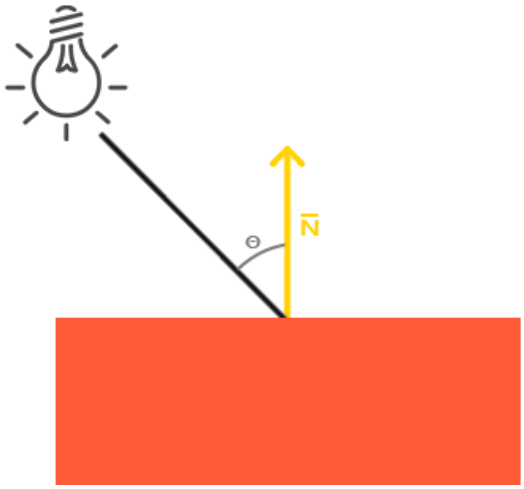


Рис. 3.9 – Розсіяне світло [56]

Код для обробки відзеркаленого світла:

```
vec3 viewDir = normalize(ViewPosition - Lights[i].Position);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0),
AppliedMaterial.Shininess);
vec4 specular = specularStrength * (spec * AppliedMaterial.SpecularColor) *
Lights[i].SpecularColor;
```

Схематичне зображення розрахунку відзеркаленого світла продемонстровано на рисунку 3.10.

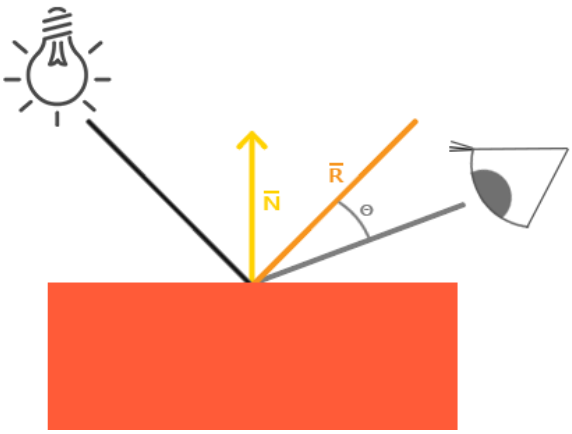


Рис. 3.10 – Відзеркалене світло [56]

Генерація коефіцієнтів згасання здійснюється за наступним кодом:

```
float distance = length(Lights[i].Position - FragPos);
float attenuation = 1.0 / (Lights[i].ConstAttenuation +
Lights[i].LinearAttenuation * distance +
    Lights[i].QuadAttenuation * (distance * distance));
```

Програмний код, в якому присвоєно значення випромінюваного матеріалом світло в окрему змінну для зручності внесення змін:

```
vec4 emission = AppliedMaterial.EmissiveColor;
```

Код для застосування коефіцієнтів згасання:

```
ambient *= attenuation;
diffuse *= attenuation;
specular *= attenuation;
```

Програмний код для об’єднання отриманих параметрів для створення результуючого кольору:

```
if(i == 0)
    result = (ambient + diffuse + specular + emission) * ObjectColor;
else
    result += (ambient + diffuse + specular + emission) * ObjectColor;
```

Недоліком простого освітлення за нормалями є відсутність урахування перекриття об’єктами джерела світла, через що світло буде проходити скрізь полігональні сітки та освітлювати поверхню інших полігональних сіток.

Додатковою функцією є закріплення джерела світла за позицією камери, що може бути більш зручним ніж виставляти статичні джерела освітлення в кількох точках для перегляду полігональної сітки. Для цього необхідно оновлювати позицію джерела світла в кожному кадрі, встановлюючи її значення як позицію камери. Код цієї процедури наведено далі:

```
public void UpdateLightingPosition(Point newPos()
{
    Position = newPos;
}
```

Варто враховувати специфіку будови пам’яті в .NET і передавати екземпляри класів клонуючи їх (для чого всі класи наслідують інтерфейс ICloneable) або створюючи новий екземпляр.

За допомогою коефіцієнтів згасання можна імітувати точкове та прожекторове освітлення. Для прожекторового освітлення також застосовують окремий коефіцієнт – кут напрямленого основою конуса за певним вектором. Такий тип джерела освітлення, зазвичай, створює коло з жорсткими межами на рівній поверхні полігональної сітки. Жорсткість меж можна регулювати способом, описаним в розділі 2.8 [60-63]. На рисунку 3.11 продемонстровано кілька варіантів точкового освітлення з різними значеннями коефіцієнтів згасання.

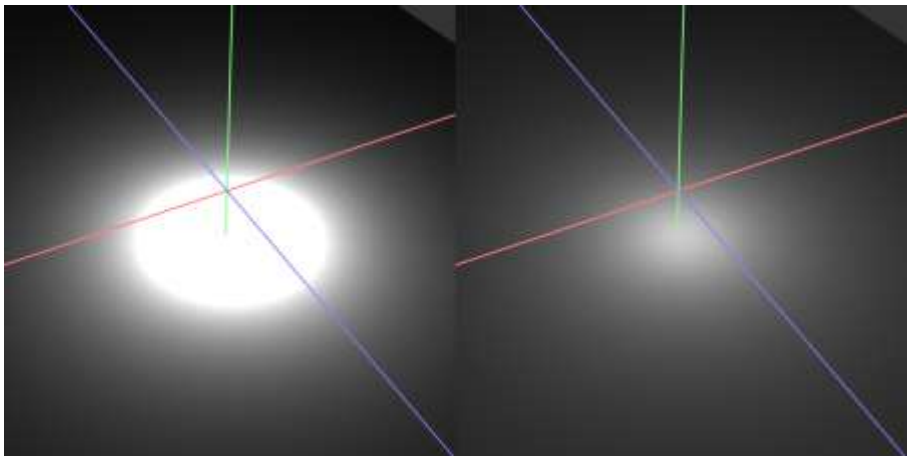


Рис. 3.11 – Вплив коефіцієнтів згасання на освітленість поверхні

Лише освітлення недостатньо для імітації різної поверхні матеріалів та утворення результуючого кольору при змішуванні кольору світла та кольору об’єкта. Для реалізації такого механізму передбачена система матеріалів, застосування в шейдері якої було продемонстровано вище, як до результуючого кольору фрагмента додаються відповідні властивості матеріалу.

Прикладом впливу матеріалу на освітлення поверхні полігональної сітки є параметр блискучості, візуалізація з різними значеннями якого зображена на рисунку 3.12.

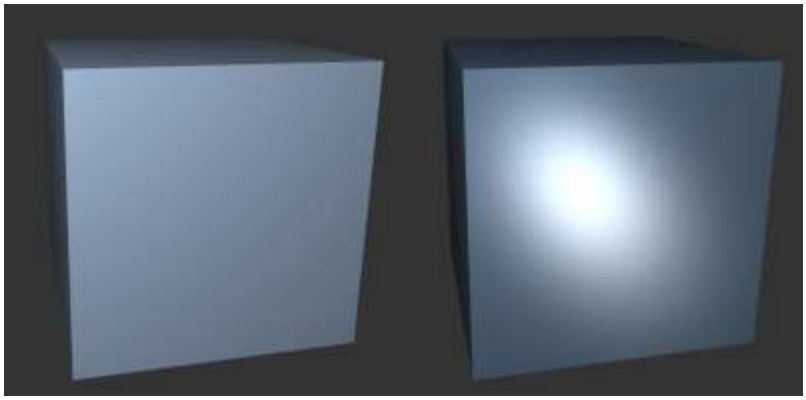


Рис. 3.12 – Вплив блискучості матеріалу на освітлення поверхні

3.4 Опис допоміжного функціоналу для реалізації інформаційної системи

Для спрощення процесу розробки був створений додатковий проєкт, який містить широковикористовуваний функціонал для всього рішення. Для реалізації ІС було створено проєкт, що містить помічників зі створення шляхів до файлів та бібліотеки з додатковими математичними функціями. Основна функція, яка будує абсолютний шлях до файлу на основі відносного описана в наступному коді:

```
public static string GetPath(string relativePath, BaseDirectory
baseDirectory = BaseDirectory.Solution)
{
    string sCurrentDirectory = AppDomain.CurrentDomain.BaseDirectory;
    sCurrentDirectory = System.IO.Path.Combine(sCurrentDirectory,
SetBaseDirectory(baseDirectory));
    string sFile = System.IO.Path.Combine(sCurrentDirectory, relativePath);

    return System.IO.Path.GetFullPath(sFile);
}
```

Оскільки ІС має вбудований функціонал з математичними операціями, до додаткової бібліотеки був доданий лише помічник з конвертування міри кута градуса до радіан та навпаки.

В реалізації ІС була застосована формула перетворення радіан до градусів [58-59]:

$$D = R \cdot (\frac{\pi}{180}), \tag{3.13}$$

де R – радіан.

Проєкт CustomExceptions містить власну базу помилок, які можуть виникнути в процесі растеризації зображення, завантаження об’єкта та інші процеси, які класифікуються окремо від стандартних процесів платформи .NET.

CoreInterface представляє інтерфейс частини додатку, яка відповідає за, безпосередньо, візуалізацію об’єктів, а також за інші процеси, які мають вплив на кінцевий результат рендеру. Такими є, наприклад, система освітлення та матеріалів. Створення такої надбудови над основним проєктом, відповідальним за формування зображення дозволяє абстрагуватись від конкретної її реалізації. Таким чином, можна замінити бібліотеку рендеру на іншу (наприклад, DirectX), або змінити версію поточної бібліотеки, реалізувавши цей інтерфейс.

Стандартна реалізація бібліотеки Glfw.NET вже містить множину імпортованого до .NET функціоналу OpenGL. Проте, імпортовані методи вимагають дотримання відповідності сигнатур, які приймають лише прості значення, що не дуже зручно, коли робота здійснюється над великими об’єктами. З метою спрощення синтаксису сигнатур методів було створено власну обгортку, яка приймає об’єкти, визначені в реалізації IC. До нього, також, додані перелічувані типи даних, які покликані замінити беззнакове ціле число в ролі показника на тип чи ідентифікатор (в залежності від функції, яка приймає таке значення) на більш зрозумілі людині слова і обмежує значення лише валідними даними, на відміну від uint, діапазон значень якого строго визначений, завжди однаковий і завжди перевищує необхідну кількість, що може призвести до помилки при вказанні значення, дії при якому не визначені всередині функції.

3.5 Опис допоміжного функціоналу для використання програмного забезпечення

Прикладне програмне забезпечення повинно мати зручний інтерфейс і гнучку систему налаштувань для комфортної роботи людини з ним. З цією метою до рішення було введено кілька додаткових елементів, які не є необхідними для процесу візуалізації об’єктів, але дозволяють здійснювати додаткові операції, які не відносяться до цього процесу напряму.

Так, StlHelper дозволяє експортувати полігональноі сітки у форматі *.stl, що виключає можливість імпортувати таку полігональну сітку зі збереженням будь-яких даних, окрім позицій вершин та їх нормалей. Очевидно, для кожного типу файлу, що зберігає об’єкти тривимірної графіки, збережені дані будуть відрізнятись.

Для уніфікації збереженої інформації про сцену, а також отримання можливості зберегти будь-які інші важливі дані (матеріали, джерела освітлення тощо), введено функціонал, який дозволяє працювати з власним розширенням файлів *.scn. На відміну від помічників, що працюють з файлами стандартизованих форматів файлів, помічник, що містить представлений функціонал обробляє файли *.scn за власними алгоритмами. Тому такі файли неможливо використовувати в іншому ПЗ для 3D-моделювання.

Для динамічного регулювання певних параметрів, а також для початкової їх ініціалізації створено клас GeneralOptions, в якому вони визначені.

Існують опції, що регулюють додатковий функціонал, який спрощує процес розробки, але не є необхідним для кінцевого користувача. Для зручності класифікації, такі опції зібрані в окремому класі DebugOptions.

Часто в програмному забезпеченні можна помітити, що один і той самий функціонал може виконуватись з кількох точок програми. При цьому, важливо мати змогу обмежити можливість виконання функції за певних умов. Патерн команди, реалізований в платформі .NET, вирішує це завдання. Він дозволяє працювати з абстракцією команди, яка є обгорткою над функцією, маючи свої додаткові механізми керування її запуском та передачі параметрів до неї. Також, для інтеграції команд в WPF передбачена однойменна властивість для елементів керування.

Команди використовуються і при визначенні функціоналу гарячих клавіш. Таких механізм дозволяє користувачеві швидко переключатись між інструментами, зберігати чи завантажувати файл, змінювати опції тощо. В реалізації ІС створено окремий клас для керування функціоналом, закріпленим за кожним набором комбінацій клавіш.

Важливим компонентом ПЗ є його інтерфейс, який, в основному, визначає зручність взаємодії між ним та користувачем. WPF, окрім властивостей компонент та їх подій має механізми для керування візуальною складовою додатку.

XAML (Extensible Application Markup Language) спрощує побудову розмітки інтерфейсу, надаючи схожий до HTML формат вкладених елементів, кожен з яких визначає розташування чи функціональний компонент. Задаючи властивості таким компонентам можна визначити додаткові параметри, такі як позицію компонента в розмітці сітки, розміри, колір, шрифт тощо.

Проте, не дуже ефективним з точки зору процесу розробки програмного забезпечення є стилізація компонентів безпосередньо в файлі з інтерфейсом. Стилі займають багато простору і це заважає очам концентруватись на розмітці. Вирішити цю проблему можна створивши додатковий XAML файл, який містить стилі для компонентів. Стилі можуть застосовуватись як при безпосередньому встановленні їх до компонента за назвою стилю, так і автоматично при визначенні стилю за його цільовим типом. В такому випадку, він застосується до всіх компонентів з заданим типом автоматично, що спрощує створення стилів за замовчуванням. Також, файли XAML можуть працювати зі змінними, за рахунок чого можна уникнути зайвої дублікації коду і отримати більш гнучку систему стилізації інтерфейсу. Для підключення такого файлу до проєкту необхідно створити словник ресурсів і встановити на нього посилання.

Фрагмент коду створення змінної в XAML з реалізації завдання:

```
<SolidColorBrush x:Key="Color-15" Color="#fff"/>
```

Фрагмент коду підключення файлу зі змінними з реалізації завдання:

```
<ResourceDictionary.MergedDictionaries>
    <ResourceDictionary Source="../XamlResources/Variables.xaml" />
</ResourceDictionary.MergedDictionaries>
```

Фрагмент коду стандартного стилю з використанням змінної:

```
<Style TargetType="Label">
```

```
        <Setter Property="Foreground" Value ="{StaticResource Color-15}"/>
</Style>
```

Приклади інтерфейсу програмної реалізації представленої ІС зображені на рисунках 3.14-3.16.

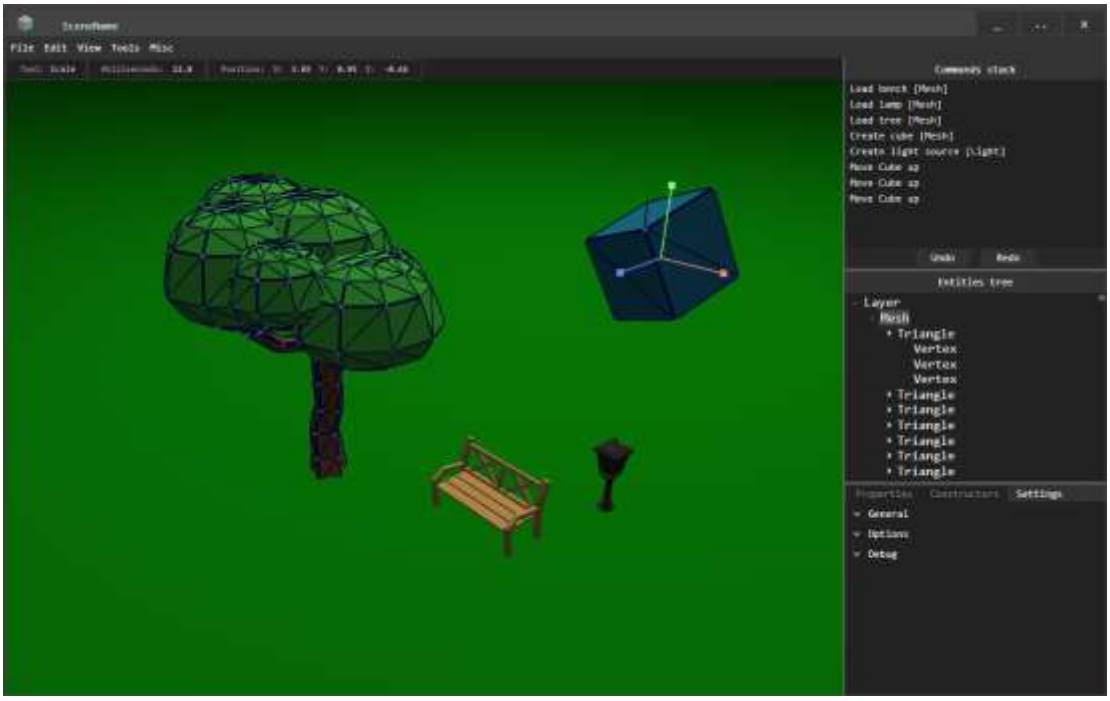


Рис. 3.14 – Користувацький інтерфейс реалізованої ІС для 3D-моделювання

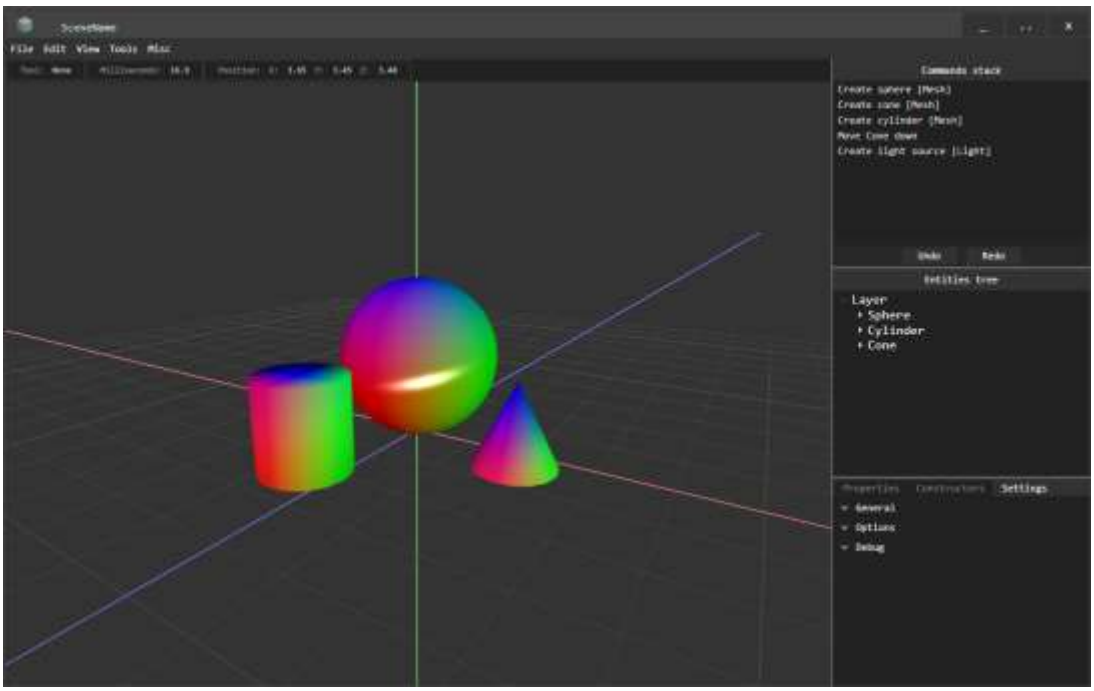


Рис. 3.15 – Користувацький інтерфейс реалізованої ІС для 3D-моделювання

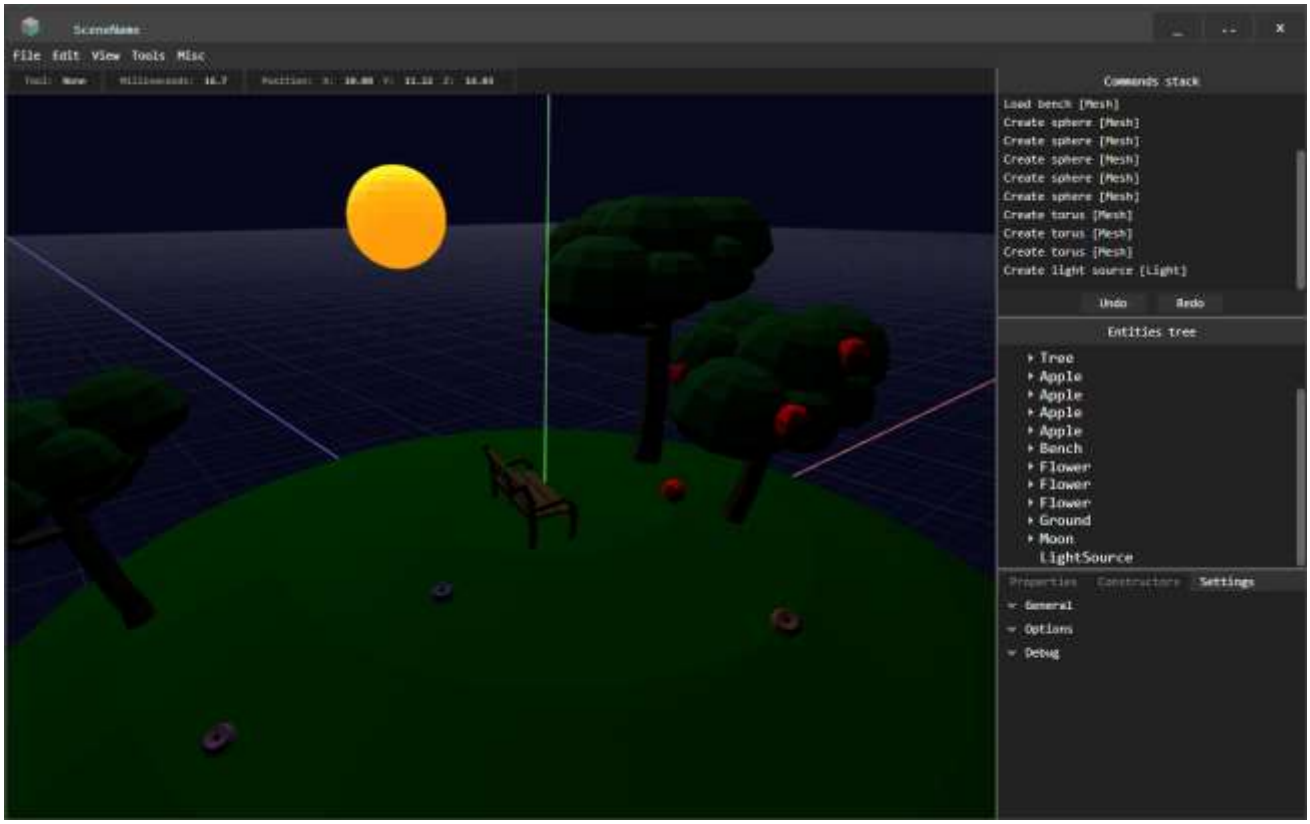


Рис. 3.16 – Користувацький інтерфейс реалізованої ІС для 3D-моделювання

ViewportManager дозволяє фіксувати погляд камери в визначеному напрямку, що широко використовується разом з ортогональною проєкцією для перегляду та редагування сцени в форматі креслення.

3.6 Опис основного функціоналу для використання програмного забезпечення

3.6.1 Алгоритми переміщення камери

В першу чергу для візуального сприйняття об’єкта на сцені має бути можливість розглядати його з різних сторін. Беручи до уваги, що об’єктом може бути як одна полігональна сітка так і кілька шарів, що містять множину полігональних сіток, огляд за допомогою камери є невід’ємною частиною будь-якого ПЗ, яке надає функціонал для роботи з тривимірною графікою.

Для зручного керування положенням камери та напрямком погляду було реалізовано три типи переміщення. Оскільки вільне переміщення використовує алгоритм орбітального переміщення, пояснення принципу його роботи непотрібне.

Першим типом є орбітальний рух камери навколо точки. Реалізується він за допомогою сферичної системи координат, де одне із значень відповідає за рух по осі абсцис, інше – за вісь ординат. Фрагмент коду з реалізації поставленої задачі, який містить метод, що здійснює такий рух, наведений нижче:

```
public void Orbit(Point offset)
{
  SetYaw(Yaw + offset.X);
  SetPitch(Pitch + offset.Y);
  ApplyOrbitRotation();
  SetEye(Eye + Target);
}
```

Панорамний рух камери здійснюється паралельно певній осі. Зазвичай, більш інтуїтивним для користувача є рух по осям системи координат об’єкта руху.

Представлений тип руху також здійснюється при вільному переміщенні камери (зазвичай, клавіші Q та E) догори та вниз. Зображення положення осей руху відносно камери продемонстровано на рисунку 3.15.

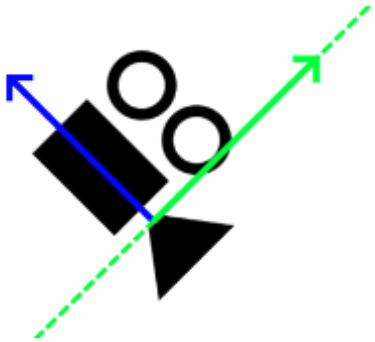


Рис. 3.17 – Схематичне зображення панорамного руху відносно власної системи координат об’єкта руху

Реалізація такого переміщення передбачало застосування наступної формули:

$$\begin{bmatrix} E_x + XA_x \cdot (-O_x) + (O_y \cdot YA_x) \\ E_y + YA_y \cdot O_y \\ E_z + XA_z \cdot (-O_x) + (O_y \cdot YA_z) \end{bmatrix}, \quad (3.18)$$

де *Ex*, *Ey*, *Ez* – позиція камери;

XAx, XAz – коефіцієнти вектора X відносно положення камери;

Ox, Oy – зміщення;

YAx, YAy, YAz – коефіцієнти вектора Y відносно положення камери.

3.6.2 Алгоритми побудови таблиці напівребер та операцій з нею

Ініціалізація таблиці відбувається додаванням всіх полігонів та вершин до таблиці, після чого відбувається процес під'єднання ребер близнюків. Код методу, що реалізує даний процес наведено далі:

```
private void InitHalfEdgeTable()
{
    HalfEdgeTable.Faces.Clear();
    HalfEdgeTable.Vertices.Clear();

    // Add faces, vertices, edges
    foreach (var triangle in Triangles)
      HalfEdgeTable.AddTriangle(triangle);

    // Connect twins
    HalfEdgeTable.ConnectTwins();
}
```

Основна складність реалізації даного алгоритму полягає в пошуку напівребер близнюків. Оскільки напівребра містять посилання на наступнє та попереднє напівребро, реалізований алгоритм пошуку можна умовно розділити на дві частини: пошук близнюків для основних напівребер та пошук близнюків для наступних і попередніх напівребер. Блок-схема алгоритму приєднання близнюків напівребер зображена на рисунку 3.19.

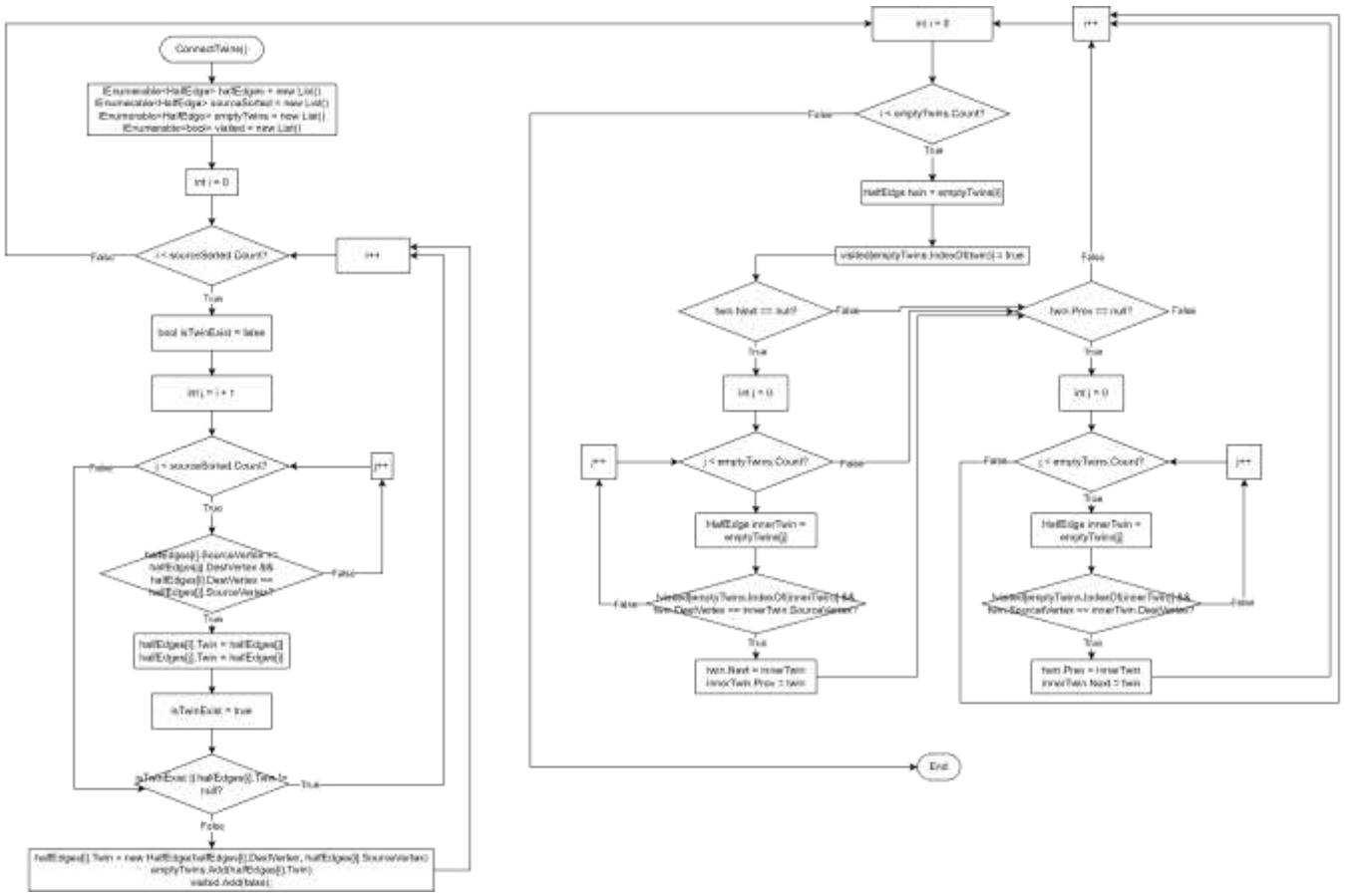


Рис. 3.19 – Блок-схема алгоритму приєднання близнюків напівребер

Фрагмент коду з реалізації ІС для з реалізації завдання для додавання прилеглих до порожнечі напівребер до потоку візуалізації:

```
var emptyTwins = currentMesh.HalfEdgeTable.GetEmptyHalfEdges();
while (emptyTwins.Count > 0)
{
    var f = emptyTwins[0];
    var i = emptyTwins[0];
    while (i.Next != f)
    {
        ObjectVertices.Add(i.SourceVertex.Point);
        ObjectVertices.Add(i.DestVertex.Point);
        emptyTwins.Remove(i);
        i = i.Next;
    }
    ObjectVertices.Add(i.SourceVertex.Point);
    ObjectVertices.Add(i.DestVertex.Point);
    emptyTwins.Remove(i);
}
```

Визначити нормалі вершин можна виконавши обхід ребер, прилеглих до кожної вершини і отримавши середнє значення полігонів, до яких належать ребра.

3.6.3 Алгоритм шейдеру billboard

Шейдери billboard використовуються для відображення плоских об’єктів в тривимірному просторі. Такі об’єкти завжди повернуті до камери за обраною віссю. Фрагмент коду з реалізації ІС для застосування такого ефекту відбувається на етапі обробки вершинного шейдеру:

```
vec4 P = modelView * vec4(center, 1.0) + vec4(aPos, 1.0);
gl_Position = projection * P;
```

В даному випадку застосовується сферична модель білборду, що означає обертання об’єкта як по осі абсцис так і по осі ординат.

Працює цей алгоритм завдяки тому, що матриця камери перемножається лише на центр об’єкта, а решта вершин розставляються відносно нього, що нівелює вплив позиції камери на те, як буде відображено об’єкт.

3.6.4 Алгоритм роботи компонента TreeViewNodeController

В проєкті WpfClient є елемент інтерфейсу, відповідальний за побудову дерева даних. Для її коректної роботи необхідно передбачити однорідність елементів, які будуть передані до цього компонента. Оскільки логіка класу Entity не повинна прив’язуватись до логіки інтерфейсу, було створено новий клас TreeViewNode, який містить Entity і додаткові властивості (ім’я, видимість, розгорнутість тощо), необхідні для побудови дерева на основі його екземплярів. Важливими для вузлів дерева властивостями є вказівник на батьківський вузол і список дочірніх вузлів, що забезпечує можливість реалізації пошуку вузла більш швидкими алгоритмами, спрощує здійснення операцій над вузлами (додавання до вузла, видалення вузла, зміна позиції вузла тощо).

Приклад зовнішнього вигляду дерева сцени наведено на рисунку 3.20.

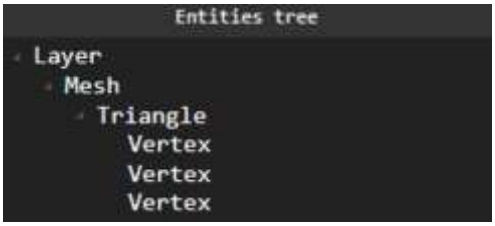


Рис. 3.20 - Зовнішній вигляд дерева об’єктів сцени з трикутником

3.7 Висновки

В даному розділі роботи були описані деталі реалізації інформаційної системи для 3D-моделювання на основі OpenGL, представлено основні алгоритми функціональної частини програмного забезпечення, продемонстровано їх роботу та його інтерфейс. Також, були представлені діаграми класів та потоків даних у реалізованій ІС, в яких зображені ключові елементи керування даними.

В процесі реалізації ІС було встановлено ключові точки програмного забезпечення для 3D-моделювання:

1) налаштування вікна рендеру;
2) ініціалізація початкових даних;
3) цикл рендеру;
    3.1) попереднє встановлення опцій рендеру;
    3.2) процес візуалізації;
    3.3) обробка подій, оновлення даних про об’єкти;
4) коректний вихід з програми шляхом ручного знищення створеного вікна після виходу з циклу рендера.

Також було продемонстровано приклади роботи програмної реалізації ІС для 3D-моделювання на основі OpenGL.

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи є інформаційна система для 3D-моделювання на основі OpenGL, яка надає змогу користувачам здійснювати основні операції над об’єктами сцени в тривимірному просторі.

В першому розділі дипломної роботи було здійснено теоретичний аналіз, вивчено недоліки існуючих рішень та сформовано задачу на дослідження шляхів їх вирішення. Відбулось ознайомлення з предметною областю 3D-графіки та моделювання.

В другому розділі дипломної роботи було здійснено теоретичний аналіз різних елементів тривимірної графіки, визначено мінімальний функціонал, необхідний для обробки полігональних моделей, їх недоліки та переваги.

В ході проєктування ІС були створені UML-діаграми для більш наочного представлення компонентів системи, їх взаємодії та визначення основного функціоналу, який повинна надавати реалізація завдання, сформульовано технічне завдання до роботи та мета розроблення ІС для 3D-моделювання на основі OpenGL. З метою підвищення ефективності та надійності процесу розробки було здійснено планування основних етапів робіт.

В третьому розділі дипломної роботи були описані деталі реалізації інформаційної системи для 3D-моделювання на основі OpenGL, представлено основні алгоритми функціональної частини програмного забезпечення, продемонстровано їх роботу та його інтерфейс. Також, були представлені діаграми класів та потоків даних у реалізованій ІС, в яких зображені ключові елементи керування даними. Також було продемонстровано приклади роботи програмної реалізації ІС для 3D-моделювання на основі OpenGL.

Було розроблено ПЗ для полігонального 3D-моделювання з використанням технологій WPF та WinForms для представлення інтерфейсної частини, бібліотеки GLFW – для керування контекстом вікна та візуалізації графіки за допомогою OpenGL. Було з’ясовано, що в основі процесів візуалізації тривимірної графіки та

здійснення операцій над об’єктами сцени переважне місце займають функції аналітичної геометрії.

Практична цінність реалізованої ІС полягає в її застосуванні в області тривимірного моделювання звичайними користувачами, яким необхідно переглянути чи відредагувати тривимірну модель або сцену з додатковими параметрами, такими як освітлення та матеріали. ІС містить велику кількість налаштувань візуалізації графіки OpenGL, а також додаткові інструменти візуалізації, такі як показ векторів нормалей полігональної сітки, виділення порожнеч полігональної сітки та інші. Перевагами системи є її масштабованість та низькі системні вимоги, що робить реалізовану ІС доступним редактором тривимірних моделей.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Planchard D. SOLIDWORKS 2022 Tutorial. *SDC Publications*, 2022. 652 с.
2. Triangulation by Ear Clipping. URL: https://www.geometrictools.com/Documentation/TriangulationByEarClipping.pdf (дата звернення: 01.03.2023).
3. Sung K., Munson M., Pavleas J., Pace J. Build Your Own 2D Game Engine and Create Great Web Games. *Apress*, 2021. 741 с.
4. 3D printing. URL: https://3dprintingindustry.com/3d-printing-basics-free-beginners-guide/ (дата звернення: 26.02.2023).
5. Dasgupta S. 3D Face Reconstruction from Front and Profile Image. *University of Dayton*, 2021. 40 с.
6. Bistacchi A., Massironi M., Viseur S. 3D Digital Geological Models. *Wiley*, 2022. 240 с.
7. Mukundan R. 3D Mesh Processing and Character Animation: With Examples Using OpenGL, OpenMesh and Assimp. *Springer*, 2022. 213 с.
8. Panetta D., Camarlinghi N. 3D Image Reconstruction for CT and PET. *CRC Press*, 2020. 134 с.
9. Howison G. A Treatise on Analytic Geometry. *Bod Third Party Titles*, 2020. 612 с.
10. Patel D. Interactive Data Processing and 3D Visualization of the Solid Earth. *Springer International Publishing*, 2022. 355 с.
11. Begbie C., Horga M., Raywenderlich Tutorial Team. Metal by Tutorials: Beginning Game Engine Development With Metal. *Amazon Digital Services LLC-KDP*, 2022. 808 с.
12. Byl P. Mathematics for Game Programming and Computer Graphics: Explore the essential mathematics for creating, rendering, and manipulating 3D virtual environments. *Packt Publishing*, 2022. 444 с.
13. Gordon S., Clevenger J. Computer Graphics Programming in OpenGL with Java. *Mercury Learning & Information*, 2017. 346 с.

14. Guha S. Computer Graphics Through OpenGL. *CRC Press*, 2022. 702 с.

15. Hallal J. Solution Architecture with .NET. *Packt Publishing*, 2021. 238 с.

16. Grey S. Mind-Melding Unity and Blender for 3D Game Development. *Packt Publishing*, 2021. 460 с.

17. Sunday D. Practical Geometry Algorithms. *Amazon Digital Services LLC-KDP Print US*, 2021. 193 с.

18. Xu J. Practical WebGPU Graphics. *UniCAD Publishing*, 2021. 445 с.

19. Kloks T., Xiao M. A Guide to Graph Algorithms. *Springer Nature Singapore*, 2022. 340 с.

20. Godse D., Godse A. Computer Graphics. *UNICORN Publishing Group*, 2020. 382 с.

21. Mari J., Hetroy-Wheeler F., Subsol G. Geometric and Topological Mesh Feature Extraction for 3D Shape Analysis. *Wiley*, 2020. 194 с.

22. Wilson K. Essential Computer Hardware. *Elluminet Press*, 2021. 180 с.

23. Ahmadian A., Salahshour S. Soft Computing Approach for Mathematical Modeling of Engineering Problems. *CRC Press*, 2021. 222 с.

24. Homogeneous Coordinates. URL: https://yasenh.github.io/post/homogeneous-coordinates/ (дата звернення: 16.02.2023).

25. Karmakar S., Karmakar S. Analytical Geometry. *Levant*, 2022. 316 с.

26. Samanta D. 3D Modeling Using Autodesk 3ds Max with Rendering View. *IGI Global*, 2022. 291 с.

27. Mortenson M. Vectors and Matrices for Geometric and 3D Modeling. *Industrial Press, Incorporated*, 2020. 272 с.

28. Stephanidis C., Antona M. Universal Access in Human-Computer Interaction. Design Methods and User Experience. *Springer International Publishing*, 2021. 660 с.

29. Li Y., Xu M., Zhao P., Ye Z. Advanced Graphic Communication, Printing and Packaging Technology. *Springer Nature Singapore*, 2020. 883 с.

30. Wolff D. OpenGL 4 Shading Language Cookbook: Build high-quality, real-time 3D graphics with OpenGL 4.6, GLSL 4.6 and C++17. *Packt Publishing*, 2018. 472 с.

31. Georgiev S., Zennir K., Boukarou A. Multiplicative Analytic Geometry. *CRC Press*, 2022. 248 с.

32. Hill T. Essential Geometry with Analytic Geometry: A Self-Teaching Guide. *Questing Vole Press*, 2020. 118 с.

33. Orland P. Math for Programmers. *Manning*, 2021. 655 с.

34. Gambetta G. Computer Graphics from Scratch. *No Starch Press*, 2021. 248 с.

35. Vries J. Learn OpenGL. *Kendall & Welling*, 2020. 522 с.

36. Szauer G. Hands-On C++ Game Animation Programming. *Packt Publishing*, 2020. 368 с.

37. Benton J. 3D Models in Motion. *Amazon Digital Services LLC-KDP Print US*, 2020. 110 с.

38. Boufares S. Vector Math for 3D Computer Graphics. *Amazon Digital Services LLC-KDP Print US*, 2020. 324 с.

39. Mortenson M. Vectors and Matrices for Geometric and 3D Modeling. *Industrial Press, Incorporated*, 2020. 272 с.

40. Ringrose P., Bentley M. Reservoir Model Design. *Springer International Publishing*, 2021. 322 с.

41. Li Z., Du P., Ding D. Understanding Quaternions. *Nova Science Publishers*, 2020. 197 с.

42. Kuipers J. Quaternions and Rotation Sequences. *Princeton Univercity Press*, 2020. 400 с.

43. Osborn L. 3D Printing and Intellectual Property. *Camridge University Press*, 2019. 234 с.

44. Fuller G. Algebra and Trigonometry with Analytic Geometry. *Legare Street Press*, 2022. 214 с.

45. Stemkoski L., Cona J. Developing Graphics Frameworks with Java and OpenGL. *CRC Press*, 2022. 298 с.

46. Mendis D., Lemley M., Rimmer M. 3D Printing and Beyond. *Edward Elgar Publishing*, 2019. 432 с.

47. Halladay K. Practical Shader Development: Vertex and Fragment Shaders for Game Developers. *Apress*, 2019. 402 с.

48. Hildenbrand D. Introduction to Geometric Algebra Computing. *Routledge*, 2020. 194 с.

49. Malmberg F., Lindblad J., Sladoje N. Discrete Geometry and Mathematical Morphology. *Springer International Publishing*, 2021. 552 с.

50. Stair R., Reynolds G. Principles of Information System. *Cengage Learning*, 2020. 758 с.

51. Baptista G., Abbruzzese F. Software Architecture with C# 9 and .NET 5. *Packt Publishing*, 2020. 700 с.

52. Godse D., Godse A. Computer Graphics and Multimedia. *UNICORN Publishing Group*, 2020. 688 с.

53. Half-Space (Geometry). URL: https://en.wikipedia.org/wiki/Half-space_(geometry) (дата звернення: 23.02.2023).

54. AxisAlignedBoundingBox. URL: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection (дата звернення: 22.02.2023).

55. LearnOpenGL. URL: https://learnopengl.com/Getting-started/OpenGL (дата звернення: 14.02.2023).

56. OpenGL Constants Translator. URL: https://javagl.github.io/GLConstantsTranslator/GLConstantsTranslator.html (дата звернення: 14.02.2023).

57. Transformation matrix. URL: https://www.cuemath.com/algebra/transformation-matrix/ (дата звернення: 15.02.2023).

58. GLFW Documentation. URL: https://www.glfw.org/documentation.html (дата звернення: 19.02.2023).

59. OpenGL Wiki. URL: https://www.khronos.org/opengl/wiki/ (дата звернення: 21.02.2023).

60. ResearchGate. Example of quad triangle subdivision. URL: https://www.researchgate.net/figure/Example-of-quad-triangle-subdivision-a-Control-mesh-b-c-One-and-two-subdivision_fig2_228631006 (дата звернення: 25.02.2023).

61. PyMesh. Mesh boolean. URL: https://pymesh.readthedocs.io/en/latest/mesh_boolean.html (дата звернення: 02.03.2023).

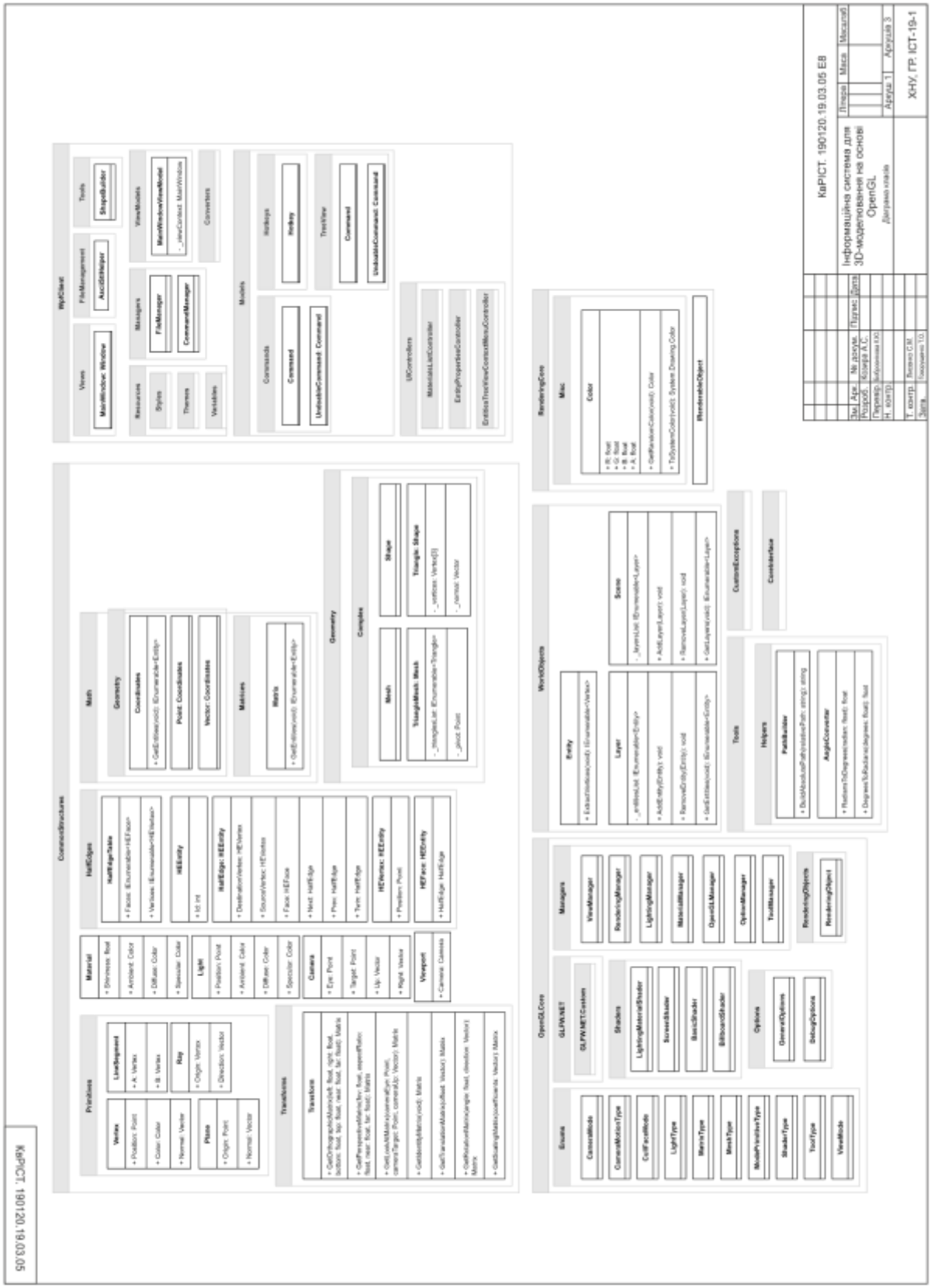
62. Banon C., Raspall F. 3D Printing Architecture. *Springer Nature Singapore*, 2020. 127 с.

63. Horvath J., Cameron R. Mastering 3D Printing. *Apress*, 2020. 347 с.

Додаток А

(обов’язковий)

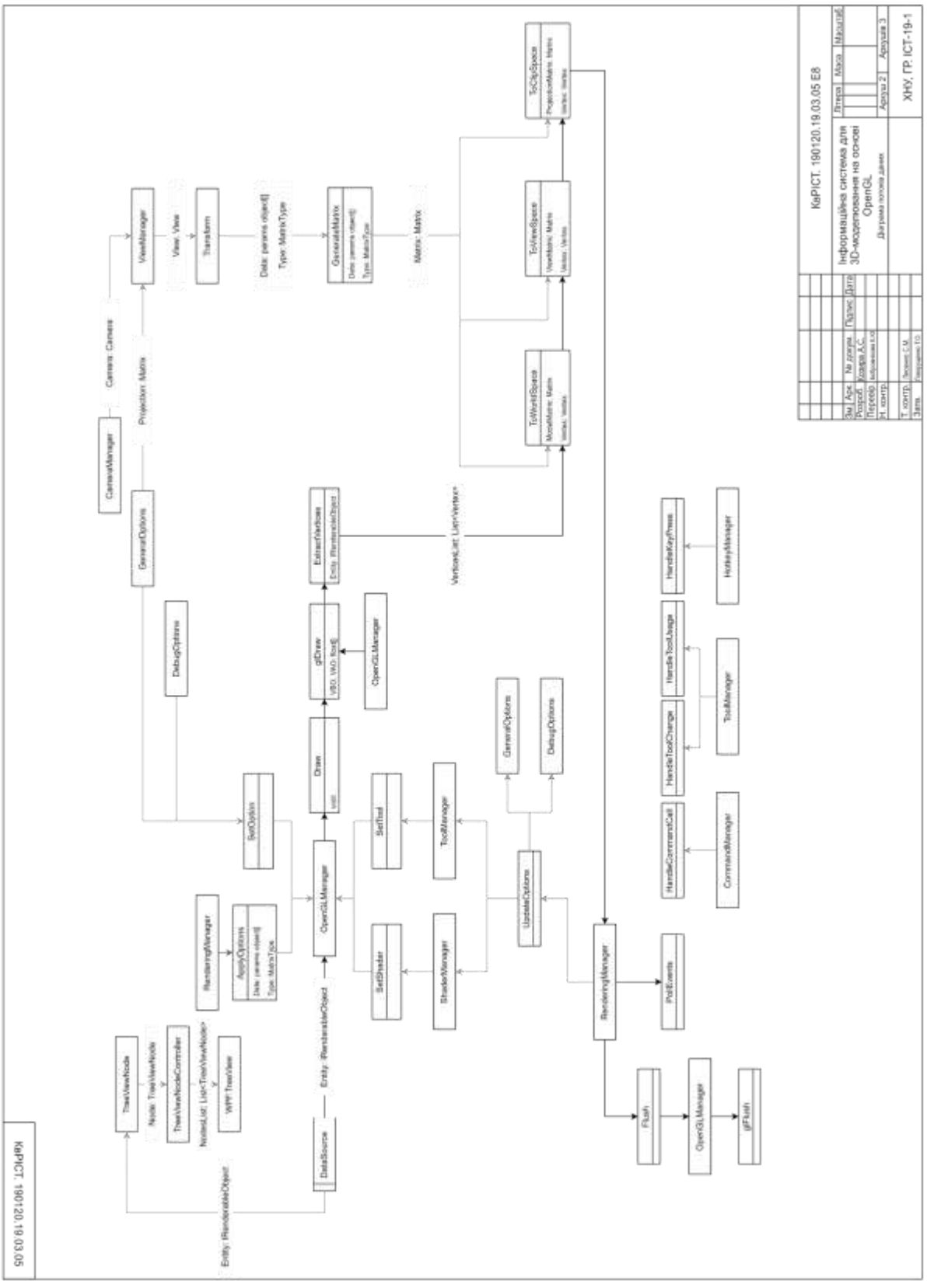
Діаграма класів



Додаток Б

(обов’язковий)

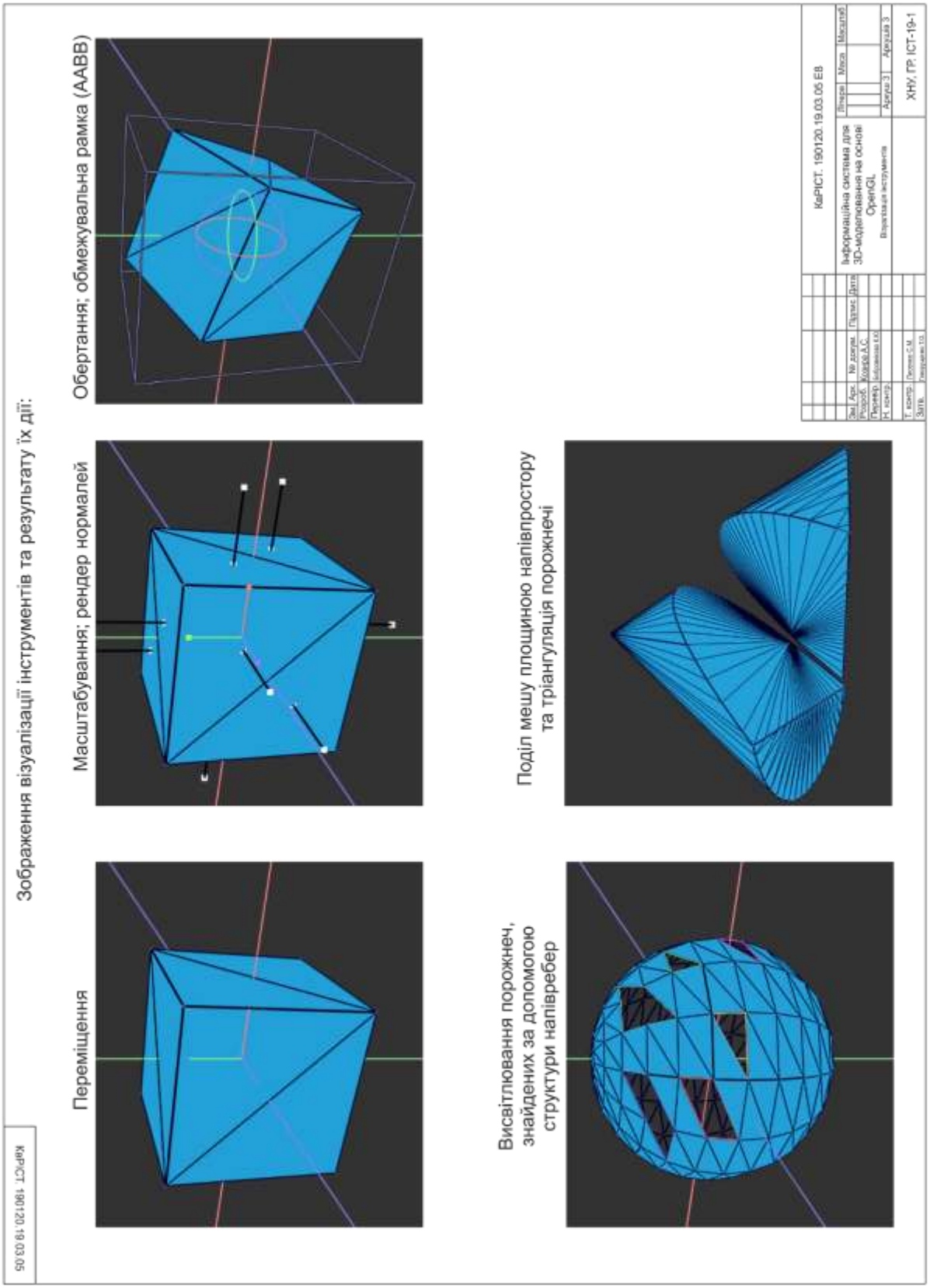
Діаграма потоків даних



Додаток В

(обов’язковий)

Візуалізація інструментів для 3D-моделювання



Додаток Г

Лістинг коду

OpenGLRenderingControl/View/Viewport.cs

```
using System;
using System.Windows.Forms;
using ClassesLib.GeometryStructures.Complex.BoundingBoxes;
using ClassesLib.GeometryStructures.Primitives;
using ClassesLib.Tools;
using OpenGLRenderingControl.MyOpenGL.GlfwWrapper;
using OpenGLRenderingControl.MyOpenGL.GlfwWrapper.Structs;
using RenderingControl.Enumerations;

namespace OpenGLRenderingControl.View
{
  public class Viewport
  {
    #region Constructors

    public Viewport(int width, int height, Control control, CameraMode
cameraMode)
    {
      Width = width;
      Height = height;

      Camera = new Camera(cameraMode, control);

      InitProperties();
    }

    #endregion Constructors

    private void InitProperties()
    {
      VerticalFieldOfViewInDegrees = 60;
      Near = 0.1f;
      Far = 100.0f;
    }

    #region Properties

    public Window Window { get; set; }
    public Camera Camera { get; }
    public float AspectRatio => Width / (float)Height;
    public float VerticalFieldOfViewInDegrees { get; set; }
    public float TargetPlaneWidth => TargetPlaneHeight * AspectRatio;
    public float TargetPlaneHeight =>
      2 * Camera.DistanceFromEyeToTarget *
      (float)Math.Tan
      ((float)AngleConverter.GetRadiansFromDegrees(VerticalFieldOfViewInDegrees
/ 2));
    public int Width { get; set; }
    public int Height { get; set; }
    public float Near { get; set; }
    public float Far { get; set; }
```

```
    #endregion Properties

    #region Public logic

    public void SetWindow(Window pWindow)
    {
      Window = pWindow;
    }

    public void SetSize(int width, int height)
    {
      Width = width;
      Height = height;

      OpenGL.glViewport(0, 0, Width, Height);
      Glfw.SetWindowSize(Window, Width, Height);
    }

    public void SetNearDistance(float near)
    {
      Near = near;
    }

    public void SetFarDistance(float far)
    {
      Far = far;
    }

    public void ZoomToFit(AxisAlignedBoundingBox bBox)
    {
      float coefficient = Camera.CameraMode == CameraMode.Orthographic ? 1.0f :
0.75f;

      Point center = bBox.GetCenter();
      Point size = coefficient * bBox.GetSize();

      float max = Math.Max(Math.Max(size.X, size.Y), size.Z);

      Camera.Target = center;

      float diff = Camera.DistanceFromEyeToTarget - max;

      Camera.DistanceFromEyeToTarget -= diff;
      Camera.UpdatePosition();
    }

    #endregion Public logic
  }
}
```

OpenGLRenderingControl/View/Camera.cs

```
using System;
using System.Windows.Forms;
using ClassesLib.GeometryStructures.Primitives;
using RenderingControl.Enumerations;
using ClassesLib.Tools;
using ClassesLib.GeometryStructures;
```

```
namespace OpenGLRenderingControl.View
{
  public class Camera
  {

    #region Private fields

    private float _pitch;
    private float _distanceFromEyeToTarget;

    #endregion Private fields

    #region Constructors

    public Camera(CameraMode cameraMode, Control control)
    {
      Eye = new Point(0.0f, 0.0f, -1.0f);
      Target = new Point(0.0f, 0.0f, 0.0f);
      Up = new Vector(0.0f, 1.0f, 0.0f);
      Front = new Vector(0.0f, 0.0f, 1.0f);
      Right = new Vector(1.0f, 0.0f, 0.0f);
      DistanceFromEyeToTarget = (int)Math.Sqrt(
        Math.Pow(Eye.X - Target.X, 2) +
        Math.Pow(Eye.Y - Target.Y, 2) +
        Math.Pow(Eye.Z - Target.Z, 2));
      ZoomDistanceStep = 0.1f;
      MinDistance = 0.02f;
      Yaw = -90.0f;

      ViewMode = ViewMode.Free;
      CameraMode = cameraMode;
      _control = control;
    }

    #endregion Constructors

    public void UpdatePosition()
    {
      Orbit(new Point());
    }

    public void SetOrbitMotionType()
    {
      DistanceFromEyeToTarget = 1.0f;
      ZoomDistanceStep = 0.1f;
      MinDistance = 0.02f;
    }

    public void SetFreeMotionType()
    {
      DistanceFromEyeToTarget = 0.001f;
      ZoomDistanceStep = 0.0001f;
      MinDistance = 0.0002f;
    }

    #region Properties

    private Control _control { get; set; }
```

```
    public float Yaw { get; private set; }
    public float Pitch
    {
      get
      {
        return _pitch;
      }
      private set
      {
        _pitch = value > 89.0f ? 89.0f
          : value < -89.0f ? -89.0f
          : value;
      }
    }

    public Vector Direction { get; set; }
    public Point Eye { get; private set; }
    public Point Target { get; set; }
    public Vector Up { get; private set; }
    public Vector Front { get; private set; }
    public Vector Right { get; private set; }
    public CameraMode CameraMode { get; set; }
    public ViewMode ViewMode { get; set; }

    public static float MinDistance { get; set; }
    public float ZoomDistanceStep { get; set; }
    public float DistanceFromEyeToTarget
    {
      get
      {
        return _distanceFromEyeToTarget;
      }
      set
      {
        _distanceFromEyeToTarget = value <= MinDistance ? ZoomDistanceStep :
value;
      }
    }

    #endregion Properties

    #region Private logic

    public void ApplyDistanceToTargetCoef()
    {
      Eye = DistanceFromEyeToTarget * Eye;
    }

    #endregion Private logic

    #region Public logic

    public void ResetTarget()
    {
      Target = new Point(0, 0, 0);
    }

    public void ResetEye()
```

```
    {
      Eye = new Point(0, 0, -1);
    }

    public void SetYaw(float value)
    {
      Yaw = value;
    }

    public void SetPitch(float value)
    {
      Pitch = value;
    }

    public void SetEye(Point eye)
    {
      Eye = eye;
    }

    public void SetMode(CameraMode mode)
    {
      CameraMode = mode;
    }

    public void Refresh()
    {
      _control.Invalidate();
    }

    public void MoveEye(Vector direction)
    {
      Eye.X += direction.X;
      Eye.Y += direction.Y;
      Eye.Z += direction.Z;
    }

    public void MoveTarget(Vector direction)
    {
      Target.X += direction.X;
      Target.Y += direction.Y;
      Target.Z += direction.Z;
    }

    public void Rotate(Point pivot, float angle, Vector direction)
    {
      Transform rotationTransform = Transform.GetRotationTransform(angle,
direction);

      var res = rotationTransform.Multiply(new Transform(new float[,]
        { { Eye.X }, { Eye.Y }, { Eye.Z }, { 1 } }
      ));

      Eye.X = res[0, 0];
      Eye.Y = res[1, 0];
      Eye.Z = res[2, 0];
    }

    public Transform GetViewMatrix()
    {
```

```
      return Transform.GetLookAtTransform(Eye, Target, Up);
    }

    public void ApplyFreeRotation()
    {
      // Rotate target around eye
      Target.X =
        (float)Math.Cos(AngleConverter.GetRadiansFromDegrees(Yaw)) *
(float)Math.Cos(AngleConverter.GetRadiansFromDegrees(Pitch));
      Target.Y =
        (float)Math.Sin(AngleConverter.GetRadiansFromDegrees(Pitch));
      Target.Z =
        (float)Math.Sin(AngleConverter.GetRadiansFromDegrees(Yaw)) *
(float)Math.Cos(AngleConverter.GetRadiansFromDegrees(Pitch));

    }
    public void ApplyOrbitRotation()
    {
      Eye.X =
        (float)Math.Cos(AngleConverter.GetRadiansFromDegrees(Yaw)) *
(float)Math.Cos(AngleConverter.GetRadiansFromDegrees(Pitch));
      Eye.Y =
        (float)Math.Sin(AngleConverter.GetRadiansFromDegrees(Pitch));
      Eye.Z =
        (float)Math.Sin(AngleConverter.GetRadiansFromDegrees(Yaw)) *
(float)Math.Cos(AngleConverter.GetRadiansFromDegrees(Pitch));
      ApplyDistanceToTargetCoef();
    }

    public void Free(Point offset)
    {
      SetYaw(Yaw + offset.X);
      SetPitch(Pitch + offset.Y);

      ApplyFreeRotation();

      Target = Eye + Target;
    }

    public void Orbit(Point offset)
    {
      SetYaw(Yaw + offset.X);
      SetPitch(Pitch + offset.Y);

      ApplyOrbitRotation();

      SetEye(Eye + Target);
    }

    public void Pan(Point offset)
    {
      Vector zAxis = Vector.Normalize(Eye - Target);
      Vector xAxis = Vector.Normalize(Vector.Cross(Up, zAxis));
      Vector yAxis = Vector.Cross(zAxis, xAxis);

      Eye.X +=
        xAxis.X * -offset.X + (offset.Y * yAxis.X);
      Eye.Y +=
        yAxis.Y * offset.Y;
```

```
      Eye.Z +=
        xAxis.Z * -offset.X + (offset.Y * yAxis.Z);

      Target.X +=
        xAxis.X * -offset.X + (offset.Y * yAxis.X);
      Target.Y +=
        yAxis.Y * offset.Y;
      Target.Z +=
        xAxis.Z * -offset.X + (offset.Y * yAxis.Z);
    }

    public void SetEyeTargetUp(Point newEye, Point newTarget, Vector newUp)
    {
      Eye = newEye;
      Target = newTarget;
      Up = newUp;
    }

    #endregion Public logic
  }
}
```

OpenGLRenderingControl/Managers/ShaderManager.cs

```
using ClassesLib.GeometryStructures;
using ClassesLib.GeometryStructures.Complex.LightBoxes;
using ClassesLib.GeometryStructures.Primitives;
using ClassesLib.Tools.PathManager;
using OpenGLRenderingControl.Lighting;
using OpenGLRenderingControl.Materials;
using OpenGLRenderingControl.MyOpenGL.GlfwWrapper;
using OpenGLRenderingControl.MyOpenGL.GlfwWrapper.Enums;

namespace OpenGLRenderingControl.Managers
{
  public class ShaderManager
  {
    #region Resources

    private const string BasicVertexShaderPath =
@"Shaders\\BasicShader\\shader.vert";
    private const string BasicFragmentShaderPath =
@"Shaders\\BasicShader\\shader.frag";

    private const string LightMaterialVertexShaderPath =
@"Shaders\\LightingMaterialShader\\shader.vert";
    private const string LightMaterialFragmentShaderPath =
@"Shaders\\LightingMaterialShader\\shader.frag";

    private const string ScreenVertexShaderPath =
@"Shaders\\ScreenShader\\shader.vert";
    private const string ScreenFragmentShaderPath =
@"Shaders\\ScreenShader\\shader.frag";

    #endregion Resources

    #region Properties

    public MaterialLocations MaterialLocation;
```

```
    public LightLocations LightLocation;

    public int CountOfLightsLocation { get; set; }
    public int CountOfMaterialsLocation { get; set; }

    public int ColorLocation { get; set; }
    public int MvpMatrixLocation { get; set; }
    public int ViewPositionLocation { get; set; }

    public uint SelectedShaderProgram { get; set; }

    public uint BasicShaderProgram { get; set; }
    public uint LightMaterialShaderProgram { get; set; }
    public uint ScreenShaderProgram { get; set; }

    public int NormalLocation { get; set; }
    public int ProjectionLocation { get; set; }
    public int ViewLocation { get; set; }
    public int ModelLocation { get; set; }

    #endregion Properties

    #region Public logic

    public void SetShader(uint shader)
    {
      OpenGL.glUseProgram(shader);
      SelectedShaderProgram = shader;

      SetGeneralLocations();
    }

    private void SetGeneralLocations()
    {
      ColorLocation = OpenGL.glGetUniformLocation(SelectedShaderProgram,
"ObjectColor");
      NormalLocation = OpenGL.glGetUniformLocation(SelectedShaderProgram,
"aNormal");
      ViewPositionLocation = OpenGL.glGetUniformLocation(SelectedShaderProgram,
"ViewPosition");
      MvpMatrixLocation = OpenGL.glGetUniformLocation(SelectedShaderProgram,
"mvpMatrix");
      ProjectionLocation = OpenGL.glGetUniformLocation(SelectedShaderProgram,
"projection");
      ViewLocation = OpenGL.glGetUniformLocation(SelectedShaderProgram,
"view");
      ModelLocation = OpenGL.glGetUniformLocation(SelectedShaderProgram,
"model");

      CountOfLightsLocation =
OpenGL.glGetUniformLocation(SelectedShaderProgram, "CountOfLights");
      CountOfMaterialsLocation =
OpenGL.glGetUniformLocation(SelectedShaderProgram, "CountOfMaterials");
    }

    public void ApplyLight(LightBox lightBox)
    {

      if (lightBox.Light.DiffuseAmbient)
```

```
        OpenGL.glUniform4f(LightLocation.AmbientColor,
((Light)lightBox.Light).Diffuse.Red, ((Light)lightBox.Light).Diffuse.Green,
((Light)lightBox.Light).Diffuse.Blue, ((Light)lightBox.Light).Diffuse.Alpha);
      else
        OpenGL.glUniform4f(LightLocation.AmbientColor,
((Light)lightBox.Light).Ambient.Red, ((Light)lightBox.Light).Ambient.Green,
((Light)lightBox.Light).Ambient.Blue, ((Light)lightBox.Light).Ambient.Alpha);

      if (((Light)lightBox.Light).DiffuseSpecular)
        OpenGL.glUniform4f(LightLocation.SpecularColor,
((Light)lightBox.Light).Diffuse.Red, ((Light)lightBox.Light).Diffuse.Green,
((Light)lightBox.Light).Diffuse.Blue, ((Light)lightBox.Light).Diffuse.Alpha);
      else
        OpenGL.glUniform4f(LightLocation.SpecularColor,
((Light)lightBox.Light).Specular.Red, ((Light)lightBox.Light).Specular.Green,
((Light)lightBox.Light).Specular.Blue, ((Light)lightBox.Light).Specular.Alpha);

      OpenGL.glUniform4f(LightLocation.DiffuseColor,
((Light)lightBox.Light).Diffuse.Red, ((Light)lightBox.Light).Diffuse.Green,
((Light)lightBox.Light).Diffuse.Blue, ((Light)lightBox.Light).Diffuse.Alpha);
      OpenGL.glUniform3f(LightLocation.Position,
lightBox.Appearance.Pivot.Position.X, lightBox.Appearance.Pivot.Position.Y,
lightBox.Appearance.Pivot.Position.Z);
      OpenGL.glUniform3f(LightLocation.Direction,
((Light)lightBox.Light).Direction.X, ((Light)lightBox.Light).Direction.Y,
((Light)lightBox.Light).Direction.Z);

      OpenGL.glUniform1f(LightLocation.ConstAttenuation,
lightBox.Light.ConstAttenuation);
      OpenGL.glUniform1f(LightLocation.LinearAttenuation,
lightBox.Light.LinearAttenuation);
      OpenGL.glUniform1f(LightLocation.QuadAttenuation,
lightBox.Light.QuadAttenuation);

      OpenGL.glUniform1f(LightLocation.CutOff, lightBox.Light.Cutoff);
    }


    public void SetLightLocations(LightLocations locations)
    {
      LightLocation = locations;
    }

    public void SetMaterialLocations(MaterialLocations locations)
    {
      MaterialLocation = locations;
    }

    public void InitShaders()
    {
      var basicShaderVertex = CreateShader(GlfwConstants.GL_VERTEX_SHADER,
LoadShader(PathManager.GetPath
        (BasicVertexShaderPath, BaseDirectory.OpenGLRenderingControl)));

      var basicShaderFragment = CreateShader(GlfwConstants.GL_FRAGMENT_SHADER,
LoadShader(PathManager.GetPath
        (BasicFragmentShaderPath, BaseDirectory.OpenGLRenderingControl)));
```

```
      var lightMaterialShaderVertex =
CreateShader(GlfwConstants.GL_VERTEX_SHADER, LoadShader(PathManager.GetPath
        (LightMaterialVertexShaderPath,
BaseDirectory.OpenGLRenderingControl)));

      var lightMaterialShaderFragment =
CreateShader(GlfwConstants.GL_FRAGMENT_SHADER, LoadShader(PathManager.GetPath
        (LightMaterialFragmentShaderPath,
BaseDirectory.OpenGLRenderingControl)));

      var screenShaderVertex = CreateShader(GlfwConstants.GL_VERTEX_SHADER,
LoadShader(PathManager.GetPath
        (ScreenVertexShaderPath, BaseDirectory.OpenGLRenderingControl)));

      var screenShaderFragment = CreateShader(GlfwConstants.GL_FRAGMENT_SHADER,
LoadShader(PathManager.GetPath
        (ScreenFragmentShaderPath, BaseDirectory.OpenGLRenderingControl)));


      BasicShaderProgram = OpenGL.glCreateProgram();
      LightMaterialShaderProgram = OpenGL.glCreateProgram();
      ScreenShaderProgram = OpenGL.glCreateProgram();

      OpenGL.glAttachShader(BasicShaderProgram, basicShaderVertex);
      OpenGL.glAttachShader(BasicShaderProgram, basicShaderFragment);

      OpenGL.glAttachShader(LightMaterialShaderProgram,
lightMaterialShaderVertex);
      OpenGL.glAttachShader(LightMaterialShaderProgram,
lightMaterialShaderFragment);

      OpenGL.glAttachShader(ScreenShaderProgram, screenShaderVertex);
      OpenGL.glAttachShader(ScreenShaderProgram, screenShaderFragment);

      OpenGL.glLinkProgram(BasicShaderProgram);
      OpenGL.glLinkProgram(LightMaterialShaderProgram);
      OpenGL.glLinkProgram(ScreenShaderProgram);

      OpenGL.glDeleteShader(basicShaderVertex);
      OpenGL.glDeleteShader(basicShaderFragment);
      OpenGL.glDeleteShader(lightMaterialShaderVertex);
      OpenGL.glDeleteShader(lightMaterialShaderFragment);
      OpenGL.glDeleteShader(screenShaderVertex);
      OpenGL.glDeleteShader(screenShaderFragment);
    }

    public void SetViewPositionLocation(Point cameraPos)
    {
      OpenGL.glUniform3f
        (ViewPositionLocation, cameraPos.X, cameraPos.Y, cameraPos.Z);
    }
    public void SetProjectionLocation(Transform projection)
    {
      OpenGL.glUniformMatrix4fv(ProjectionLocation, 1, false,
projection.ToArray());
    }
    public void SetViewLocation(Transform view)
    {
      OpenGL.glUniformMatrix4fv(ViewLocation, 1, false, view.ToArray());
```

```
    }
    public void SetModelLocation(Transform model)
    {
      OpenGL.glUniformMatrix4fv(ModelLocation, 1, false, model.ToArray());
    }

    public void SetCountOfLights(int count)
    {
      OpenGL.glUniform1i(CountOfLightsLocation, count);
    }

    #endregion Public logic

    #region Private logic

    private static uint CreateShader(int type, string source)
    {
      var shader = OpenGL.glCreateShader(type);
      OpenGL.glShaderSource(shader, source);
      OpenGL.glCompileShader(shader);
      return shader;
    }

    private static string LoadShader(string path)
    {
      return System.IO.File.ReadAllText(path);
    }

    #endregion Private logic
  }
}
```

ClassesLib/GeometryStructures/Transform.cs

```
using System;
using System.Linq;
using ClassesLib.GeometryStructures.Primitives;
using ClassesLib.Tools;
using RenderingControl.GeometryStructures;

namespace ClassesLib.GeometryStructures
{
  [Serializable]
  public class Transform : ICloneable, ITransform
  {
    #region Constructors

    public Transform(float[,] matrix)
    {
      Matrix = matrix;
    }

    static Transform()
    {
      Identity = new Transform(new float[,]
      {
        { 1, 0, 0, 0 },
        { 0, 1, 0, 0 },
        { 0, 0, 1, 0 },
```

```
        { 0, 0, 0, 1 },
      });
    }

    #endregion Constructors

    #region Properties

    public static Transform Identity { get; set; }

    public float[,] Matrix { get; set; }

    #endregion Properties

    #region Public logic

    public float this[int i, int j]
    {
      get => Matrix[i, j];
      set => Matrix[i, j] = value;
    }

    public static Transform Inverse(ITransform transform)
    {
      float[,] m = transform.Matrix;

      float det =
        m[0, 0] * m[1, 1] * m[2, 2] - m[1, 2] * m[2, 1] -
        m[0, 1] * m[1, 0] * m[2, 2] - m[1, 2] * m[2, 0] +
        m[0, 2] * m[1, 0] * m[2, 1] - m[1, 1] * m[2, 0];

      float[,] inverse = new float[,]
      {
        { (m[1, 1] * m[2, 2] - m[2, 1] * m[1, 2]) / det, -(m[1, 0] * m[2, 2] -
m[2, 0] * m[1, 2]) / det, (m[1, 0] * m[2, 1] - m[2, 0] * m[1, 1]) / det, 0 },
        { -(m[0, 1] * m[2, 2] - m[2, 1] * m[0, 2]) / det, (m[0, 0] * m[2, 2] -
m[2, 0] * m[0, 2]) / det, -(m[0, 0] * m[2, 1] - m[2, 0] * m[0, 1]) / det, 0 },
        { (m[0, 1] * m[1, 2] - m[1, 1] * m[0, 2]) / det, -(m[0, 0] * m[1, 2] -
m[1, 0] * m[0, 2]) / det, (m[0, 0] * m[1, 1] - m[1, 0] * m[0, 1]) / det, 0 },
        { 0, 0, 0, 1 }
      };

      return new Transform(inverse);
    }

    public float[] ToArray()
    {
      return Matrix.Cast<float>().ToArray();
    }

    public static Transform GetOrthographic(float left, float right, float
bottom, float top, float near, float far)
    {
      float[,] res = new float[,]
      {
        { 2 / (right - left), 0, 0, 0 },
        { 0, 2 / (top - bottom), 0, 0 },
        { 0, 0, -2 / (far - near), 0 },
```

```
        { -((right + left) / (right - left)), -((top + bottom) / (top -
bottom)), -((far + near) / (far - near)) , 1 },
      };

      return new Transform(res);
    }

    public static Transform GetPerspective(float fovy, float aspect, float
zNear, float zFar)
    {
      var range = (float)Math.Tan(AngleConverter.GetRadiansFromDegrees(fovy /
2)) * zNear;
      var left = -range * aspect;
      var right = range * aspect;
      var bottom = range;
      var top = -range;

      float[,] res = new float[,]
      {
        { (2 * zNear) / (right - left), 0, 0, 0 },
        { 0, -(2 * zNear) / (top - bottom), 0, 0 },
        { 0, 0, -(zFar + zNear) / (zFar - zNear), -1 },
        { 0, 0, -(2 * zFar * zNear) / (zFar - zNear), 0 },
      };

      return new Transform(res);
    }

    public static Transform GetTranslationTransform(Vector offset)
    {
      return new Transform(new float[,]
      {
        { 1, 0, 0, offset.X },
        { 0, 1, 0, offset.Y },
        { 0, 0, 1, offset.Z },
        { 0, 0, 0, 1 }
      });
    }

    public static Transform GetRotationTransform(float angle, Vector direction)
    {
      return new Transform(new float[,]
      {
        { (float)(Math.Cos(angle) + Math.Pow(direction.X, 2) * (1 -
Math.Cos(angle))),
          (float)(direction.X * direction.Y * (1 - Math.Cos(angle)) -
direction.Z * Math.Sin(angle)),
          (float)(direction.X * direction.Z * (1 - Math.Cos(angle)) +
direction.Y * Math.Sin(angle)), 0 },

        { (float)(direction.Y * direction.X * (1 - Math.Cos(angle)) +
direction.Z * Math.Sin(angle)),
          (float)(Math.Cos(angle) + Math.Pow(direction.Y, 2) * (1 -
Math.Cos(angle))),
          (float)(direction.Y * direction.Z * (1 - Math.Cos(angle)) -
direction.X * Math.Sin(angle)), 0 },

        { (float)(direction.Z * direction.X * (1 - Math.Cos(angle)) -
direction.Y * Math.Sin(angle)),
```

```
          (float)(direction.Z * direction.Y * (1 - Math.Cos(angle)) +
direction.X * Math.Sin(angle)),
          (float)(Math.Cos(angle) + Math.Pow(direction.Z, 2) * (1 -
Math.Cos(angle))), 0 },

        { 0, 0, 0, 1 }
      });
    }

    public static Transform GetScalingTransform(Vector coeffs)
    {
      return new Transform(new float[,]
      {
        { coeffs.X, 0, 0, 0 },
        { 0, coeffs.Y, 0, 0 },
        { 0, 0, coeffs.Z, 0 },
        { 0, 0, 0, 1 }
      });
    }

    public static Transform GetLookAtTransform(Point eye, Point target, Vector
up)
    {
      Vector zAxis = Vector.Normalize(eye - target);
      Vector xAxis = Vector.Normalize(Vector.Cross(up, zAxis));
      Vector yAxis = Vector.Cross(zAxis, xAxis);

      float[,] viewMatrix = new float[4, 4]
      {
        {        xAxis.X,                    yAxis.X,                zAxis.X,
         0},
        {        xAxis.Y,                    yAxis.Y,                zAxis.Y,
         0},
        {        xAxis.Z,                    yAxis.Z,                zAxis.Z,
         0},
        {    -Vector.Dot(xAxis, eye), -Vector.Dot(yAxis, eye), -
Vector.Dot(zAxis, eye),  1}
      };

      return new Transform(viewMatrix);
    }

    public Transform Multiply(Transform transform)
    {
      int rA = Matrix.GetLength(0);
      int cA = Matrix.GetLength(1);
      int rB = transform.Matrix.GetLength(0);
      int cB = transform.Matrix.GetLength(1);
      if (cA != rB)
        throw new Exception();

      float temp = 0;
      float[,] res = new float[rA, cB];

      for (int i = 0; i < rA; i++)
      {
        for (int j = 0; j < cB; j++)
        {
          temp = 0;
```

```
            for (int k = 0; k < cA; k++)
            {
              temp += this[i, k] * transform[k, j];
            }
            res[i, j] = temp;
          }
        }

        return new Transform(res);
      }

      public object Clone()
      {
        return new Transform(Matrix);
      }

      #endregion Public logic
    }
}
```

UNICHECK

Ім'я користувача:
Кафедра КІ

ID перевірки:
1015421222

Дата перевірки:
05.06.2023 08:41:42 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
05.06.2023 08:42:16 EEST

ID користувача:
100005591

Назва документа: **Козира_Інформаційна система для 3D-моделювання на основі OpenGL**

Кількість сторінок: **74** Кількість слів: **13708** Кількість символів: **109241** Розмір файлу: **6.65 MB** ID файлу: **1015083752**

3.27%
Схожість

Найбільша схожість: **0.68%** з джерелом з Бібліотеки (ID файлу: **1014883050**)

2.91% Джерела з Інтернету	83	Сторінка 76
1.67% Джерела з Бібліотеки	115	Сторінка 77

1.51% Цитат

Цитати	11	Сторінка 78

Не знайдено жодних посилань

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	18

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 17.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 11%**

	Документ		Сумарний збіг по Базі Даних	
ID: 114667 Назва: БКР Інформаційна система для 3D-моделювання на основі OpenGL Додано в БД: 2023-06-05 Автора: А.С. Козира Керівники: К.Ю. Бобровнікова Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	88111	773	15577 (18%)	142 (18%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
111987	Назва: ЗВІТ з переддипломної практики "Інформаційна система для 3D-моделювання на основі OpenGL" Додано в БД: 2023-03-10 Автора: А. С. Козира Керівники: Гнатчук Є.Г. Консультанти: Опоненти:	14821 (17.0%)	134 (17.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Козира Артур Сергійович

Тема: Інформаційна система для 3D-моделювання на основі OpenGL

Спеціальність: 126 «Інформаційні системи та технології»

Обсяг кваліфікаційної роботи:

Кількість листів креслень 3 ; кількість сторінок записки 65

1. Короткий зміст роботи та прийнятих рішень: У кваліфікаційній роботі було досліджено і проаналізовано предметну область. Був проведений аналіз відомих комерційних рішень, розглянуто їх переваги та недоліки і доведено актуальність розроблення нової інформаційної системи. Розглянуто інструменти для реалізації спроєктованих рішень, в результаті чого створено інформаційну систему.

2. Висновок про відповідність роботи дипломному завданню: Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: В першому розділі дипломної роботи було здійснено теоретичний аналіз, вивчено недоліки відомих рішень та сформовано задачу на дослідження шляхів їх вирішення. Відбулось ознайомлення з предметною областю 3D-графіки та моделювання.

В другому розділі дипломної роботи було здійснено теоретичний аналіз різних елементів тривимірної графіки, визначено мінімальний функціонал, необхідний для обробки полігональних моделей, їх недоліки та переваги.

В третьому розділі дипломної роботи були описані деталі реалізації інформаційної системи для 3D-моделювання на основі OpenGL, представлено основні алгоритми функціональної частини програмного забезпечення, продемонстровано їх роботу та інтерфейс інформаційної системи.

4. Позитивні сторони роботи: Тематика кваліфікаційної роботи є актуальною, оскільки має широке застосування в області моделювання тривимірної графіки. Розроблене програмне забезпечення має низькі системні вимоги та легко масштабується.

5. Негативні сторони роботи: Було б доцільно розширити функціонал інформаційної системи (напр. текстурування, анімація, прив'язка до сітки).

6. Оцінка графічного оформлення та пояснювальної записки роботи: Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про роботу в цілому: Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проєктування. Графічний матеріал надає можливість наочно побачити деталі проєктування системи.

8. Інші зауваження:

9. Оцінка дипломної роботи: відмінно (4.75/A)

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Гурман Іван Васильович, к.т.н., доцент кафедри інженерії програмного забезпечення

"5" червня 2023 р. (підпис)

Завідувачу кафедри КІС
д-р.техн.наук, проф. Говорущенко Т. О.

Козири Артура Сергійовича

ПІБ здобувача вищої освіти

ФІТ, 4 курсу, групи ІСТ-19-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

22 квітня 2023 року

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОМАЦІЙНИХ СИСТЕМ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Інформаційна система для 3D-моделювання на основі OpenGL

Автор: Козира Артур Сергійович

Спеціальність: 126 – Інформаційні системи і технології

Освітня програма: освітньо-професійна

Науковий керівник: Бобровнікова К.Ю., к.т.н, доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) запозичення розміщені в розділах є збіг зі звітом з науково-дослідної практики автора Артура Козири "Інформаційна система для 3D-моделювання на основі OpenGL", який було додано в репозитарій ХНУ 20 березня 2023 року;
2) усі запозичення фрагментарні, або мають належним чином оформлені посилання;
3) окремі виявлені збіги є загальновживаними фразами або виразами, про що свідчить посилання системи на збіг з джерелами на один фрагмент речення;
4) всі зафіксовані системою ознаки модифікації тексту відносяться до індексів в формулах, що не є модифікацією тексту.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості Unicheck, складає 3.27% і адресується до 198 першоджерел; та системою Anti-Plagiarism складає 17%, що, з урахуванням наведених обґрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ К.Ю.Бобровнікова

Гарант ОП _____ Є. Г. Гнатчук

Завідувач кафедри КІІС _____ Т. О. Говорущенко