МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Інформаційна система-тренажер рендерингу моделей трьохвимірного простору для дисциплін напрямку "3D моделювання та візуалізація"»

Завідувач

випускаючої кафедри **Довбиш А.С.**

Керівник роботи **Берест О.Б.**

Студентки групи ІН.м – 92 **Волик Н.А.**

СУМИ 2020

Сумський державний університет

(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*

Спеціальність *«Інформатика»*

Затверджую:

зав.кафедрою _____

_____

"_____" _____ 20___р.

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Волик Надії Андріївні

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна система-тренажер рендерингу моделей трьохвимірного простору для дисциплін напрямку "3D моделювання та візуалізація"*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи)_____

_____

_____

_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Аналіз проблеми. Постановка задачі дослідження. 2) Вибір методу розв'язання задачі. 3) Аналіз функціональних вимог системи. 4) Розробка архітектури системи. 5) Розробка інформаційного та програмного забезпечення системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

_____

_____

_____

_____

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	*Аналіз проблеми. Постановка задачі дослідження*		
2.	*Вибір методу розв'язання задачі*		
3.	*Аналіз функціональних вимог системи*		
4.	*Розробка архітектури системи*		
5.	*Розробка інформаційного та програмного забезпечення системи*		
6.	*Оформлення пояснювальної записки до дипломної роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 79 стор., 16 рис., 2 додатки, 29 джерел.

Об’єкт дослідження — інформаційна система-тренажер вивчення тривимірної графіки та обчислювальної геометрії.

Мета роботи — розробка системи, що полегшить процес навчання та виконання практичних завдань в ході вивчення дисциплін тривимірної графіки та обчислювальної геометрії, а також може використовуватись як основа для майбутньої кастомізації.

Методи дослідження — в процесі виконання програмної реалізації було використано комбінацію технологій для створення десктопних програм з графічним інтерфейсом, яка включає в себе C#, WPF, XAML, Wix#, SharpGL.

Результати — систему-тренажер рендерингу моделей тривимірного простору для дисциплін напрямку "3D моделювання та візуалізація". Дана система надає для користувача можливість завантаження 3D-моделей з файлів, їх рендеринг з урахуванням джерела світла, а також текстур та матеріалів моделі, можливість керувати камерою, підтримка багатомовного інтерфейсу та системи запитів до тривимірної моделі, яка дозволяє її модифікувати або інспектувати. Систему реалізовано у вигляді програмного забезпечення, яке було створене за допомогою мови програмування C# з використанням WPF, XAML, SharpGL та Wix#.

ТРИВИМІРНА МОДЕЛЬ, РЕНДЕРИНГ, OPENGL, C#, ПОЛІГОНАЛЬНА СІТКА, ОБЧИСЛЮВАЛЬНА ГЕОМЕТРІЯ, ДЕСКТОПНА ПРОГРАМА.

ЗМІСТ

ВСТУП

Зараз важко уявити наше життя без тривимірної графіки. Вона увійшла майже в усі сфери життєдіяльності людства. Тривимірна графіка та моделювання продовжує набирати популярність в процесі проектування продуктів у промисловості, архітектурних та дизайнерських рішеннях, кіно, анімації, рекламі, комп'ютерних іграх, в симуляції процесів в науці та робототехніці. Зростаючий попит відображення об'єктів реального об'ємного світу і створення нових віртуальних моделей викликає відповідний попит на спеціалістів для створення нових технологій та розвитку існуючих програмних систем у сфері тривимірної графіки та моделювання.

Важливими дисциплінами для підготовки таких спеціалістів є тривимірна комп'ютерна графіка, моделювання та обчислювальна геометрія, при вивченні яких студенти створюють власні програми для закріплення отриманих теоретичних знань та реалізації вивчених алгоритмів.

Написання програмного забезпечення в процесі вивчення даних предметів з нуля може бути недоцільним та займати багато часу. Доречніше використати програмне забезпечення, яке вже містить певний базовий функціонал та дозволяє додавати новий.

Нині існує досить багато програм для моделювання та рендерингу моделей тривимірного простору. Не всі з них мають можливість розширення вже існуючого функціоналу, і вони мають вже реалізовані всі основні функції та алгоритми, які вивчаються на вказаних дисциплінах, а також вони зазвичай є досить дорогими, складними та вимагають довготривалого попереднього навчання.

Таким чином метою даної роботи є створення системи, що полегшить процес навчання та виконання практичних завдань в ході вивчення дисциплін тривимірної графіки та обчислювальної геометрії, а також може використовуватись як основа для майбутньої кастомізації.

Даний проект є власністю компанії AMC Bridge.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1. Аналіз предметної галузі

Тривимірна графіка - розділ комп'ютерної графіки, присвячений методам створення зображень або відео шляхом моделювання об'ємних об'єктів в тривимірному просторі.

Графічне зображення тривимірних об'єктів включає побудову геометричної проекції тривимірної моделі сцени на площину (наприклад, екран комп'ютера) за допомогою спеціалізованих програм.

Для отримання тривимірного зображення на площині потрібні наступні кроки:

1) Моделювання - створення тривимірної математичної моделі сцени і об'єктів в ній;

2) Текстурування - призначення поверхонь моделей растрових або процедурних текстур, а також налаштування властивостей матеріалів - прозорість, відображення, шорсткість та ін.;

3) Освітлення - встановлення та налаштування джерел світла;

4) Симуляція динаміки - використовується, наприклад, коли потрібно розглянути взаємодію частинок або будь-яких об'єктів, з модельованими силами гравітації, вітру, виштовхування і ін., а також один з одним;

5) Візуалізація - побудова проекції;

6) Виведення отриманого зображення на пристрій виведення - дисплей або принтер.[1]

Важливою складовою тривимірної графіки є 3D моделювання, що являє собою процес створення тривимірної моделі об'єкта за допомогою спеціалізованого програмного забезпечення з метою її подальшого використання. Основним завданням 3D-моделювання є розробка візуального об'ємного образу бажаного об'єкта.

Основним методом моделювання для комп'ютерної графіки в режимі реального часу є полігональне моделювання - підхід для моделювання об’єктів шляхом представлення або апроксимації їх поверхонь за допомогою полігональних сіток. Точки в тривимірному просторі, які називаються вершинами, з’єднані між собою відрізками ліній, утворюють полігональну сітку[2]. Кожна геометрична грань складається щонайменше з трьох вершин, і кожна вершина може бути частиною однієї або декількох граней. Розмір і форма об’єктів змінюються шляхом зміни положення кожної вершини. Чим вищу кількість полігонів має об'єкт, тим вище деталізація і якість. На основі цього поділяють високополігональні (high poly) і низкополігональних (low poly) моделі.

Іншим методом є моделювання поверхні. Він покладається на направляючі лінії для визначення форми та кривизни деталі. Потім програмне забезпечення обчислює гладку поверхню, яка з'єднує напрямні лінії. Використання напрямних ліній - не єдиний варіант. Деякі програми використовують контрольні точки або площини управління, де бажана поверхня слідує за площинами тангенціально. Ця техніка є більш складною і вимагає більш досконалих програм, а також більше навчання та досвіду від дизайнера.[3]

Будучи сукупністю даних (точок та іншої інформації), тривимірні моделі можуть бути створений вручну, алгоритмічно (процедурне моделювання) або шляхом сканування. Їх поверхні можна додатково визначити за допомогою нанесення текстур.[4]

Тривимірна графіка має низку переваг над двовимірною:

1) Візуалізація об'єкта. У 2D-зображення відсутнє уявлення щодо тривимірних взаємодій і зв’язків з іншими об'єктами, про взаємозв'язок власних компонентів конструкції. У випадку 3D моделі об'єкт розгортається під іншим кутом, так що можна побачити його розташування та місцезнаходження інших об'єктів щодо головної фігури.

2) Відсутність потреби в додатковій фізичному моделі, в той час як при використанні двовимірних креслень в деяких випадках проектувальнику доводиться створювати фізичну модель для розуміння повної картини.

3) Можливість автоматизованого розрахунку різних властивостей виробів, таких як розрахунок розподілу тепла, мас-інерційні характеристики і т. ін. [5]

4) Вплив на фізичні реакції глядача. При правильному моделюванні сцени в 3D можна створити ефект дезорієнтації глядача в просторі: наприклад, ефект запаморочливої гонки, падіння, різкого перекидання і т.ін. Людина приміряє на себе цю реальність і стає як би частиною її, сприймаючи як дійсність. Таких ефектів в 2D-графіці досягти майже неможливо.

Тривимірна графіка та моделювання тісно переплітаються з темою обчислювальної геометрією.

Обчислювальна геометрія - розділ інформатики, який передбачає проектування, аналіз та реалізацію ефективних алгоритмів для вирішення геометричних задач. Складність обчислень є архіважливою для обчислювальної геометрії та має велике практичне значення, якщо алгоритми використовуються на дуже великих наборах даних, що містять десятки або сотні мільйонів точок. Для таких наборів різниця між $O(n^2)$ та $O(n \log n)$ може бути різницею між днями та секундами обчислення.

Основним поштовхом для розвитку обчислювальної геометрії як дисципліни був прогрес в галузі комп'ютерної графіки та автоматизованого проектування("computer-aided design", "CAD") та виготовлення ("Computer-aided manufacturing", "CAM")), але багато проблем в обчислювальній геометрії мають класичний характер і можуть з'являтись при математичній візуалізації. Інші важливі застосування обчислювальної геометрії включають робототехніку (проблеми планування руху та видимості), геоінформаційні системи (ГІС) (геометричне розташування та пошук, планування маршруту), проектування інтегральних схем (проектування та перевірка геометрії ІС),

автоматизовану інженерію (CAE) (генерація сіток), комп'ютерний зір (3D-реконструкція).[6]

Оскільки сцени складаються з геометричних об’єктів, геометричні алгоритми відіграють важливу роль у комп’ютерній графіці.

Важливішим кроком у відображенні тривимірної сцени є видалення прихованої поверхні: визначити частину сцени, видиму з певної точки зору, або, іншими словами, відкинути частини, які лежать позаду інших об’єктів. Для створення реалістичних на вигляд сцен ми повинні взяти до уваги світло. Це створює багато нових проблем, таких як обчислення тіней. Отже, реалістичний синтез зображення вимагає складних методів відображення, таких як трасування променів та дифузне відображення. Маючи справу з рухомими об’єктами та у додатках віртуальної реальності, важливо виявляти зіткнення між об’єктами. Усі ці ситуації пов’язані з геометричними проблемами.[7]

Сьогодні 3D-моделі використовуються в самих різних сферах.

Галузь промисловості використовує їх як конструкції нових пристроїв, транспортних засобів та конструкцій, а також для цілого ряду інших потреб.

Сучасні технології дозволяють створити візуалізацію проектованого об'єкта, максимально наближеного до реального пристрою, оцінити його наочно. Тривимірна модель майбутнього механізму прискорює і полегшує роботу інженерів-конструкторів. Тому, особливо важливим для цієї категорії користувачів є не зовнішній вигляд моделі, а можливість застосування формул, роботи з ними, зрізові креслення, графіка, а також перевірка всього механізму на будь-якому етапі розробки.

Отриманий продукт є геометричною сутністю, і, отже, слід очікувати появи всіляких геометричних задач. Справді, пакети CAD повинні мати справу з перетинами та об'єднаннями об'єктів, з декомпозицією об'єктів та меж об'єктів на більш прості форми та з візуалізацією розроблених продуктів. Щоб вирішити, чи відповідає конструкція специфікаціям,

необхідні певні випробування. Часто для цих випробувань не потрібно будувати прототип, і достатньо моделювання.[7]

3D-моделі також можуть бути основою для фізичних пристроїв, побудованих на 3D-принтерах або верстатах з числовим програмним керуванням(ЧПК).

Тривимірна графіка незамінна і для презентації майбутнього виробу. Часто замовники вимагають продумати незвичайний дизайн предметів. В даному випадку процес візуалізації допомагає продемонструвати дизайнерське рішення.

3D-візуалізація також надзвичайно активно використовується в сфері архітектури, будівництва та дизайну інтер’єру. Архітектурна промисловість використовує їх для демонстрації запропонованих будівель та ландшафтів замість традиційних, фізичних моделей архітектури.

За допомогою тривимірної графіки досягається максимально реалістичне моделювання міської архітектури і ландшафтів, з мінімальними витратами. Візуалізація архітектури будівель і ландшафтного оформлення дає можливість інвесторам і архітекторам відчути ефект присутності в спроектованому просторі. Що дозволяє об'єктивно оцінити переваги проекту і усунути недоліки.

Сьогодні 3D-візуалізація будинку - важливий етап перед початком будівельних робіт. Дана технологія дозволяє подивитися на готовий об'єкт і внести зміни при необхідності. Тривимірна візуалізація використовується при створенні реклами інтер'єру або будівель з метою ознайомити споживачів з конкретною пропозицією. Особливість реклами в сфері будівництва та архітектури полягає в тому, що вона пропонує те, чого поки що не існує. Наприклад, реклама житлового комплексу показує результат - споживач бачить зображення готових будинків, в той час як їх зведення може тільки починатися[8]. Архітектурна промисловість використовує їх для демонстрації запропонованих будівель та ландшафтів замість традиційних, фізичних моделей архітектури.

Однак роль тривимірного моделювання в сфері дизайну інтер'єру та будівництва не обмежується демонстрацією об'єкта. 3D моделювання є важливою частиною імітаційного моделювання - це процес створення та аналізу цифрового прототипу фізичної моделі для прогнозування її ефективності в реальному світі.

Важливими функціями програм тривимірного моделювання, які необхідні архітекторам є:

- оцінка екстремальних умов навколишнього середовища, таких як сильний вітер, урагани, висока температура, сильне сонячне світло або повна темрява;
- випробування світлового потоку;
- вивчення ефективності потоку циркуляції повітря всередині та зовні будівлі;
- оптимізація дизайнерської форми для навантажень, ваги та безпечної конструкції.[9]

Тривимірні моделі також використовуються в медичній галузі для створення інтерактивних зображень анатомії. Моделі органів, створені на основі на основі даних томографії, можуть використовуватися хірургами для підготовки до операції. Надзвичайного поширення в медицині набув 3D-друк. За його допомогою створюються імпланти та протези. Активно розвивається біодрук, в якому використовуються живі людські клітини.

Інститут Фраунгофера використовує біодрук на основі органічних стовбурових клітин для виробництва кровоносних судин - трубок, що відтворюють властивості людських артерій і вен.

Університет Ноттінгема Трент за допомогою 3D-друку створив кісткові імплантати, що біологічно розкладаються, які можна використовувати для лікування раку кісток та великих переломів. Синтетичні кістки відтворюють пористу структуру і надруковані з речовини, що містить ті самі мінерали, що й кістки людини, що дозволяє їм з часом розчинятися із зростанням органічної тканини.[10]

Все частіше маркетологи використовують 3D-візуалізацію об'єктів, створюючи анімаційні рекламні ролики через те, що є можливість створити вигаданих героїв, а також 3D-об'єкт виглядає більш привабливо, ніж в реальному житті, краще передає необхідні якості пропонованого продукту.

Без тривимірного моделювання не можуть існувати сфери анімації, комп'ютерних ігор, кінематографу.

Для цих галузей важливим є поняття ріггінгу у 3D моделюванні - це процес розробки і процедура підготовки персонажа до анімації, що включає в себе створення та установку всередині тривимірної моделі віртуального «скелета» - набору «кісток» або «суглобів», процес встановлення систематичного взаємозв'язку між ними і значень всіляких модифікацій і видозмін для кожної з цих кісток.

Скелетна анімація, для якої і використовується ріггінг, зручна насамперед тим, що дозволяє керувати величезною кількістю складових частин фігури, що анімується, - кінцівки, очі, м'язи обличчя, губи і т.ін., за допомогою порівняно невеликої кількості регулюючих частин - тих самих кісток і їх контрольованих характеристик.[11]

3D-моделювання і візуалізація разом з алгоритмами обчислювальної геометрії широко використовується в робототехніці.

Симулятори роботів дозволяють легко і швидко перевірити безліч різних ідей для їх реалізації, протестувати їх, а потім вирішити, який з роботів будувати. Також за допомогою симулятора можливо протестувати поведінку робота в багатьох середовищах. Часто при розробці програмного забезпечення для робота виникають помилки і для їх вирішення доречно також використовувати симуляцію. Виправлення основних помилок реального робота може закінчитися в найкращому випадку до багато витраченого часу, а в гіршому випадку - до поломки робота. Застосувавши симуляцію, є можливість подумати про модифікації робота та протестувати їх. Легко перевірити, чи можливі нові функції, в якому обсязі та що потрібно буде змінити та додати. [12]

Оскільки роботи - це геометричні об’єкти, які діють у тривимірному просторі - реальному світі - очевидно, що геометричні проблеми виникають у багатьох місцях. Планування руху - один із аспектів більш загальної проблеми планування завдань. Це передбачає планування рухів, планування порядку виконання підзадач тощо. [7]

1.2. Огляд існуючих рішень

В даний час існує досить велика кількість програм для моделювання та рендерингу моделей тривимірного простору. До найпопулярніших таких систем можна віднести Autodesk Maya, Autodesk 3ds Max, Blender, SketchUp, Wings3D, Rhino, а також SolidWorks, AutoCAD. Inventor, CATIA, Fusion 360 для інженерного моделювання, та ZBrush і Mudbox для скульптингу.

Розглянемо більш детально кілька таких систем.

Autodesk Maya - редактор тривимірної графіки, доступний для Windows, macOS та Linux. В основному розглядається як галузевий стандарт для створення та моделювання персонажів у 3D. Autodesk Maya може похвалитися неперевершеним набором інструментів та функцій для 3D-анімації, моделювання та візуалізації. Її великий набір функцій включає частинки, волосся, фізику твердого тіла, тканину, імітацію рідин та анімацію персонажів. Набір інструментів цього редактору надзвичайно складний і вимагає часу на вивчення. Цей рівень потужності також має свою ціну - передплата на Maya коштує недешево($1620 на рік). Широко застосовується в кіно, телебаченні та індустрії комп’ютерних ігор.[13]

Важливою особливістю Maya є відкритість для сторонніх розробників, які можуть додавати необхідний для них функціонал. У Maya є вбудована інтерпретована платформо-незалежна мова: Maya Embedded Language (MEL), дуже схожа на Tcl та C. За її допомого, наприклад, користувач може записати свої дії, як скрипт на MEL, з якого можна швидко зробити зручний макрос. Для написання зовнішніх розширень на мовах C++, C# та Python є детально задокументований API. Мова MEL не прив'язується до платформи, тому код,

написаний на ній, буде виконуватися в будь-якій операційній системі, в якій працює Maya.[14]

SolidWorks - це пропрієтарне програмне забезпечення твердотільного моделювання систем автоматизованого проектування та автоматизованої інженерії. По суті, це повноцінний набір для конструювання виробів в цифровому вигляді, який містить в собі безліч додаткових інструментів, що дозволяють виробляти над моделлю віртуальні технічні випробування. На сьогоднішній день програма доступна для роботи лише на операційних системах Windows. Розроблена та опублікована Dassault Systèmes. SolidWorks використовує параметричний підхід на основі функцій. Програмне забезпечення написане на ядрі Parasolid. Включає широкий спектр функцій, таких як твердотільне 3D моделювання, розробку зварних конструкцій, розрахунки на міцність, прорахунок гідро / аеродинаміки, можливість створення креслень, візуалізацію, роботу з даними 3D сканування та багато інших. Як правило, SolidWorks використовується в промисловому дизайні та машинобудуванні.[15]

SolidWorks підтримує розширення функціоналу за рахунок доповнень, які являють собою COM-об’єкти, які можна розробити за допомогою будь-яких COM-сумісних мов, таких як C ++, C #, VB.NET, VB6.[16]

Blender - це безкоштовне програмне забезпечення для створення 3D із відкритим кодом. Він підтримує величезну кількість функцій - 3D-моделювання, текстурування, скульптинг, ріггінг, анімація, універсальні вбудовані механізми рендерингу і інтеграція з зовнішніми рендерерами YafRay, LuxRender і багатьма іншими, симуляція рідини, вогню, диму, частинок, твердого тіла, редагування відео та багато іншого.[17]

Багатство функціональних можливостей має свою ціну: навіть незважаючи на те, що це програмне забезпечення для 3D-моделювання є безкоштовним, освоювати його далеко не просто. В даний час користується великою популярністю серед безкоштовних 3D-редакторів в зв'язку з його швидким стабільним розвитком, завдяки великому співтовариству

розробників, які постійно розширюють функціональність Blender, та технічною підтримкою. Характерною особливістю пакету Blender є його невеликий розмір в порівнянні з іншими популярними пакетами для 3D-моделювання. Blender підтримується на платформах Windows, Mac та Linux.[18]

Це програмне забезпечення також дає можливість використовувати Python API, завдяки якому можна редагувати будь-які дані, які може редагувати користувацький інтерфейс (сцени, сітки, частинки тощо); змінювати гарячі клавіши та теми; створювати елементи інтерфейсу користувача, такі як меню, заголовки та панелі; створювати нові інструменти, створювати нові рендери, які інтегруються з Blender та інше.[17]

Wings 3D - це безкоштовне програмне забезпечення для 3D-моделювання, написане на Erlang з відкритим вихідним кодом, доступний для Windows, Linux та Mac OS X. Програма пропонує широкий спектр інструментів для моделювання, настроюваний інтерфейс, підтримку освітлення та матеріалів та вбудований автогенератор текстурних координат AutoUV. Представляючи спрощений вступ до 3D-моделювання, Wings 3D використовує контекстний інтерфейс користувача, який демонструє лише відповідні параметри відповідно до обраного інструменту. Саме цей інтерфейс робить цю завантажену програму зрозумілою для початківців.[19]

Wings не підтримує анімацію і має лише базові засоби рендерингу OpenGL, хоча він може експортувати зовнішнє програмне забезпечення для рендерингу, таке як POV-Ray та YafRay. Дана програма має менеджер плагінів для додавання та видалення плагінів, написаних мовою Erlang.[20]

Іншим варіантом системи для вивчення тривимірної графіки є фреймворк, створений в університеті Штутгарта. Він дозволяє користувачу створювати власні плагіни та динамічно їх завантажувати до програми. Користувачі можуть реалізувати лише один плагін за раз. Плагіни завантажуються динамічно і можуть інтерактивно перемикатися під час виконання. Для працюючого плагіна користувач успадковує базовий клас, що

вимагає реалізації принаймні чотирьох віртуальних методів для ініціалізації, активації, деактивації плагіну та метод для рендерингу, який викликається із фреймворка і відображає сцену. Для підвищення зручності також є два сценарії (PowerShell та Perl), які створюють екземпляр нового плагіна з обраною назвою, представляючи користувачам порожній клас, в якому їм потрібно лише імплементувати раніше перелічені методи. Дана система призначена для вивчення лише OpenGL та має досить обмежений інтерфейс, а також не має можливості завантажувати різні моделі під час роботи програми.[21]

Також власний фреймворк для вивчення комп'ютерної графіки є в Віденському технологічному університеті. Цей фреймворк реалізує рендерер з підтримкою шейдерів та інтерактивний 3D-редактор. На початку семестру студенти отримують неповну версію рендереру, який вони потім доповнюють виконуючи практичні завдання. Студенти можуть протестувати своє рішення та взаємодіяти з ним за допомогою простого тривимірного редактора, який використовує їх реалізацію для візуалізації для рендеринга в реальному часі[22]. На жаль, матеріли даного фреймворку наразі недоступні, тому неможливо ознайомитися з ним більш детально.

1.3. Постановка задачі

Метою проекту є розробка системи, що може використовуватись як основа для майбутньої кастомізації, а також полегшить процес навчання та виконання практичних завдань в ході вивчення дисциплін тривимірної графіки та обчислювальної геометрії. Система має надавати можливості завантаження 3D-моделей з файлів, їх рендеринг з урахуванням джерела світла, а також текстур та матеріалів моделі, можливість керувати камерою, тобто повертати, масштабувати та переміщувати вид сцени, підтримка багатомовного інтерфейсу. Важливою є також система запитів до тривимірної моделі, яка дозволяє її модифікувати або інспектувати. Основними запитами є переміщення, поворот, масштабування моделі, розріз

моделі площиною, об’єднання вершин, побудова мінімальної випуклої оболонки, знаходження розривів полігональної сітки моделей. Також запит може складатися з декількох команд, які вкладені одна в одну, тобто результат однієї команди може слугувати вхідними даними для іншої. Розроблений фреймворк має мати зрозумілу структуру для подальшого розширення та налаштування під конкретні вимоги.

Основними завданнями роботи, вирішення яких сприятиме досягненню мети, є:

1. огляд та вибір програмного забезпечення для майбутнього додатка;
2. розробка моделей бізнес-процесів;
3. розробка архітектури системи;
4. розробка та реалізація дизайну інтерфейсу користувача;
5. створення модулю завантаження та збереження моделей;
6. реалізація рендерингу моделей;
7. реалізація модулю інтерпретації запитів;
8. реалізація алгоритмів базових трансформацій моделей;
9. тестування.

2 ВИБІР МЕТОДІВ ВИРІШЕННЯ

2.1. Мова програмування

Одним із найважливіших аспектів розробки системи є програмування, яке виконується за допомогою певної мови програмування.

Найпопулярнішими мовами для розробки десктопних додатків є C# і Java. Розглянемо їх більш докладно.

C# - це сучасна, об'єктно-орієнтована та безпечна для програмування мова програмування. C# сягає своїм корінням з сімейства мов C. Кілька функцій C# допомагають створювати надійні та довговічні програми. Збір сміття автоматично відновлює пам’ять, зайняту недосяжними невикористаними об’єктами. Обробка винятків забезпечує структурований та розширюваний підхід до виявлення та відновлення помилок. Лямбда-вирази підтримують методи функціонального програмування. Підтримка асинхронних операцій забезпечує синтаксис для побудови розподілених систем.[23]

C# є частиною технології .Net, яка має декілька реалізацій. Найактуальнішою з них є .NET Core - це міжплатформна реалізація .NET для Windows, Linux та macOS. .NET підтримує чотири крос-платформні сценарії: веб-програми ASP.NET Core; програми командного рядка; бібліотеки; та універсальні програми Windows Platform. До .NET Core 3.0 не реалізовувалися Windows Forms або Windows Presentation Foundation (WPF), які відображають стандартний графічний інтерфейс для настільного програмного забезпечення в Windows. Однак зараз .NET Core 3 підтримує настільні технології Windows Forms, WPF та Universal Windows Platform (UWP). .NET підтримує використання пакетів NuGet. На відміну від .NET Framework, який обслуговується за допомогою Центру оновлення Windows, .NET покладається на менеджер пакетів для отримання оновлень. На мові C# можна створювати веб-сервіси, ігри, мобільні та десктопні додатки, хмарні сервіси. [24]

Java є універсальною мовою програмування загального призначення, яка використовується для створення крос-платформних додатків. Java - одна з найбільш широко використовуваних комп’ютерних мов. Java - проста, загальноприйнята, об'єктно-орієнтована, надійна, безпечна, нейтральна до архітектури, портативна, високопродуктивна, багатопоточна комп'ютерна мова. Вона призначена для того, щоб дозволити розробникам додатків "писати один раз, запускати де завгодно", що означає, що код, який працює на одній платформі, не потрібно перекомпілювати для роботи на іншій. Код Java працює на будь-якій машині, яка не потребує встановлення спеціального програмного забезпечення, але на машині має бути присутнім JVM. Оскільки Java VM доступна у багатьох різних операційних системах, то програми написані на Java можуть працювати в Microsoft Windows, операційній системі Solaris, Linux або Mac OS.[25]

2.2. Графічний інтерфейс

Платформами для створення програм с графічним інтерфейсом для мови C# є Windows Forms та WPF.

Windows Forms(Winforms) - платформа для керованих програм Windows із легкою моделлю інтерфейсу та доступом до .NET Core або повної .NET Framework. Він відмінно допомагає розробникам швидко розпочати створення додатків, навіть для розробників, що не мають попереднього досвіду використання платформи. Це швидка платформа для розробки додатків на основі форм із великою вбудованою колекцією візуальних та невізуальних елементів керування. [26]

У програмі Windows Forms Windows забезпечують обгортку, що складається з набору класів C ++ для розробки програм Windows, і кожен елемент керування у програмі форми Windows є конкретним екземпляром класу. Він надає різноманітні елементи керування, такі як текстові поля, кнопки, мітки та веб-сторінки, а також параметри створення спеціального елемента керування. Для цього у Visual Studio доступний інструмент

конструктора вікон для обробки елементів керування у формі та упорядкування їх відповідно до бажаного макета для додавання коду для обробки подій.[27]

WPF - це платформа для керованих програм Windows з доступом до .NET Core або повної .NET Framework, а також вона використовує розмітку XAML для відокремлення інтерфейсу від коду. За допомогою XAML у WPF програмісти можуть працювати паралельно з дизайнерами. Ця платформа призначена для настільних додатків, які потребують складного інтерфейсу, налаштування стилів та графічних сценаріїв.[26]

Обидві платформи - Winforms та WPF - популярний вибір на ринку, тому слід зазначити деякі основні відмінності між ними.

Winforms легше використовувати під час розробки програм, тоді як WPF вимагає розуміння повного потоку елементів управління та дизайнерської частини. Проте Windows Forms не використовуються для розробки нових програм, для цього використовується WPF. Також Winforms елементи керування важко налаштувати, тоді як у WPF елементи керування легко налаштовуються, оскільки вони повністю написані з нуля та піддаються обробці або тематизації, для інтерфейсу можна використовувати різні обкладинки або теми, підтримується плавне масштабування компонентів інтерфейсу, не маючи проблем зі спотворенням розміру.[27]

Зважаючи усі переваги та недоліки платформ для виконання проекту кращим вибором є WPF.

2.3. Візуалізація тривимірної графіки

SharpGL - це проект, який дозволяє легко використовувати OpenGL у програмах Windows Forms або WPF. За своєю суттю SharpGL - це не що інше, як обгортка API OpenGL. Зазвичай API OpenGL використовується безпосередньо в додатках C або C ++, але код налаштування може бути досить складним - особливо, якщо потрібно використовувати сучасні розширення OpenGL. SharpGL пропонує всі функції OpenGL та розширення

безпосередньо. Будь-яка функція OpenGL, яка починається на 'gl' або 'glu', є функцією-членом об'єкта SharpGL.OpenGL, при цьому видаляється 'gl' або 'glu'. Усі константи OpenGL визначаються як постійні члени класу SharpGL.OpenGL і мають однакові імена. Усі основні функції OpenGL повністю задокументовані, тобто ви отримуєте необхідну інформацію під час набору тексту. SharpGL дозволяє покращити читабельність та зменшити кількість помилок, використовуючи безпечні перелічення(enum) основних констант. [27]

Open Toolkit(OpenTK) - це набір швидких, низькорівневих прив'язок C# для OpenGL, OpenGL ES та OpenAL. Він працює на всіх основних платформах і використовується для різних додатків, ігор та наукових досліджень. OpenTK надає кілька бібліотек службових програм, включаючи пакет математичної / лінійної алгебри, систему вікон та обробку вводу. Відкритий набір інструментів відображає необроблений API OpenGL з додаванням безпечних для переліку перелічень. Це забезпечує високу продуктивність, простий переклад коду C/C++, і крім того, відповідає філософії OpenTK - це інструментарій, а не фреймворк або движок. Основним недоліком є відсутність елемента контролю для WPF. Це можна обійти тільки використовуючи разом з WindowsFormsHost, що тягне за собою додавання зайвих залежностей. [28]

Обидві бібліотеки є лише обгортками для OpenGL, проте SharpGL краще підходить для використання в WPF додатках, тому для виконання проекту буде використовуватися саме SharpGL.

2.4. Інсталятор

Для встановлення та поширення програми необхідно створити інсталятор. Для цього буде використовуватися фреймворк для створення інсталяційного пакету MSI – Wix#. Він дозволяє створювати повноцінний MSI-дистрибутив, виконуючи файли скриптів, написані простим синтаксисом C#. Механізм Wix# використовує структуру класу C# для

імітації сутностей WiX та їх взаємозв’язків. Wix# використовує вихідний код більш керованого синтаксису (C#), щоб створити бажаний вихідний код із менш керованим синтаксисом (WiX). "Більш керований синтаксис" у цьому контексті означає менш надмірний і більш читабельний код, кращу перевірку помилок під час компіляції та наявність більш досконалих інструментів.[29]

2.5. Аналіз варіантів використання системи

Для того, щоб зобразити основні функції, була створена діаграма бізнес-прецедентів, яка відображає погляд на діяльність системи з точки зору користувача та функціональні вимоги до неї.

Варіанти використання системи представлені в діаграмі (Рисунок 2.1).

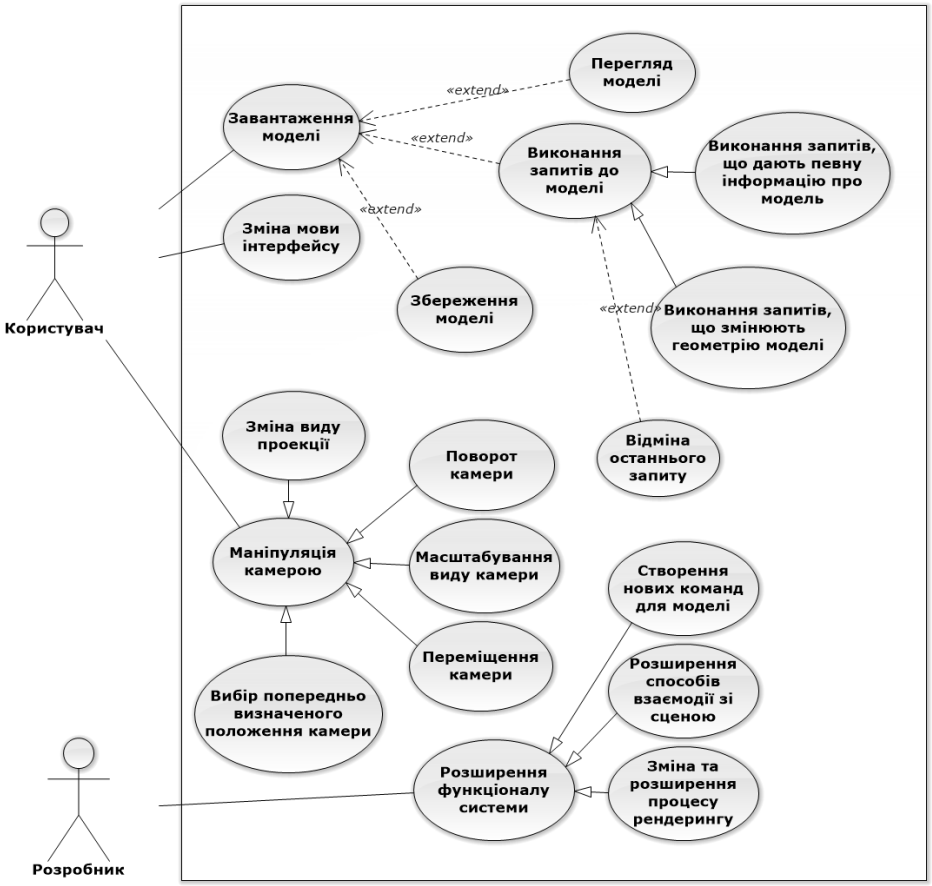

Рисунок 2.1 Діаграма варіантів використання системи

Задіяні актори:

- Користувач – актор, який використовує створений додаток;
- Розробник – актор, який займається безпосередньою розробкою та розширенням нового функціоналу системи.

2.6. Архітектура системи

Структура й взаємозв'язок компонентів інформаційної системи зображені нижче(Рисунок 2.2).

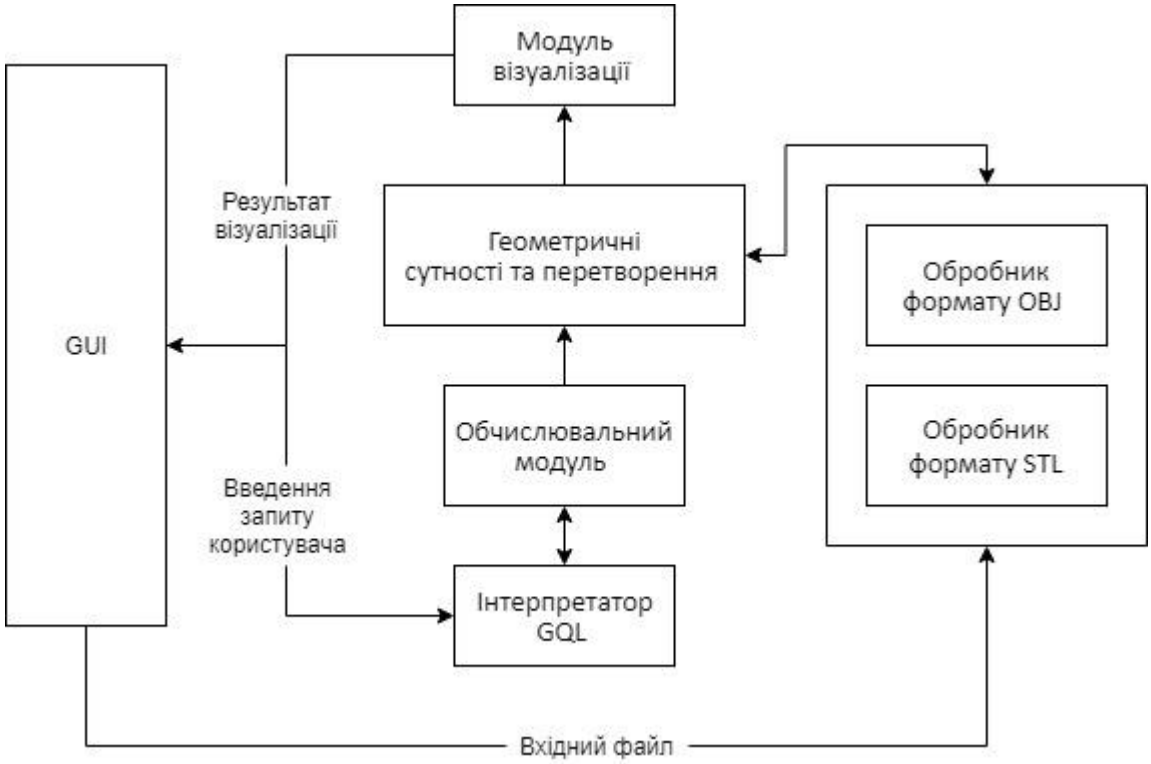

Рисунок 2.2 – Архітектура системи

Наведена структура містить такі компоненти:

1. Обробник формату OBJ та обробник формату STL - це сутності, які служать для серіалізації та десеріалізації даних у файлах форматів OBJ та STL.
2. GUI – графічний інтерфейс користувача.
3. Геометричні сутності та перетворення - модуль, що містить моделі та класи обслуговування для зберігання та виконання геометричних перетворень.
4. Інтерпретатор GQL - спеціальний інтерпретатор мови запитів геометрії("Geometry query language", скорочено "GQL"), яку користувач використовуватиме для модифікації або інспектування геометрії моделі.

5. Обчислювальний модуль - виконує запити до моделі, беручи початкове представлення геометрії та список команд, отриманих із запиту за допомогою модулю інтерпретатору, як вхідні дані та надаючи отриману геометрію як вихідний результат.
6. Модуль візуалізації - керує бібліотекою візуалізації (OpenGL), щоб візуалізувати результуючу геометрію, враховуючи її представлення з модуля “Геометричні сутності та перетворення”.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ

3.1. Програмна реалізація

Для реалізації системи буде використано паттерн розробки MVVM, який дозволяє відокремити логіку додатку від візуальної частини (представлення). Даний паттерн складається з трьох функціональних частин(див. **Ошибка! Источник ссылки не найден.**):

1. Model - основна логіка програми (робота з даними, обчислення, запити і так далі).

2. View – вид або представлення, відповідає за структуру, макет та вигляд того, що користувач бачить на екрані. В ідеалі кожен вигляд визначається в XAML файл з обмеженим кодом, який не містить бізнес-логіки.

3. ViewModel - модель представлення, яка служить прошарком між View і Model та пов'язує їх через механізм прив'язки даних. ViewModel також містить логіку по отриманню даних з моделі, які потім передаються в представлення. І також VewModel визначає логіку по оновленню даних в моделі.

Такий поділ дозволяє прискорити розробку і підтримку програми - можна змінювати один компонент, не зачіпаючи код іншого.

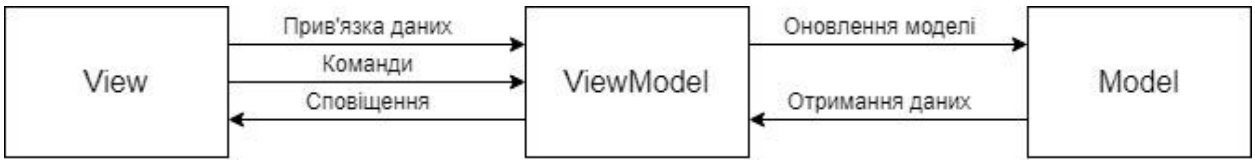

Рисунок 3.1 – Схема паттерну MVVM

ViewModel має реалізувати інтерфейс INotifyPropertyChanged. Цей інтерфейс реалізує систему повідомлень, яка активується, коли значення властивості змінюється. Це потрібно в моделі-уявлення, щоб зробити механізм прив'язки призначеного для користувача інтерфейсу XAML динамічним. Інтерфейс містить подію PropertyChanged, яка має викликатися при зміні даних, які мають бути відображені в представленні.

Оскільки елементи представлення, тобто візуальні компоненти типу кнопок, не використовують події, то представлення взаємодіє з ViewModel за допомогою команд. Команди - це об'єкти, що реалізують інтерфейс ICommand, за допомогою яких передаються повідомлення з View до ViewModel. Коли відбувається подія елемента керування викликається метод Execute у команді. Команди також можуть вказувати, коли вони можуть бути виконані. Це дозволяє елементу управління вмикати або вимикати себе залежно від того, чи можна виконати його команду.

Спочатку необхідно створити основне вікно програми. На ньому будуть відображатися область перегляду 3D-моделей, панель меню програми, поле для вводу запиту для моделі та кнопка для власне виконання введеного запиту.

Для відображення моделей за допомогою OpenGL слугує елемент OpenGLControl, який містить в собі об’єкт OpenGL. Даний елемент керування має 3 події, для яких необхідно створити обробники подій: OpenGLDraw, OpenGLInitialized и Resized.

Головною ViewModel в системі є MainViewModel, яка реалізує інтерфейс INotifyPropertyChanged та є контекстом даних для головного вікна. Тут будуть знаходитися обробники для вище зазначених подій OpenGL.

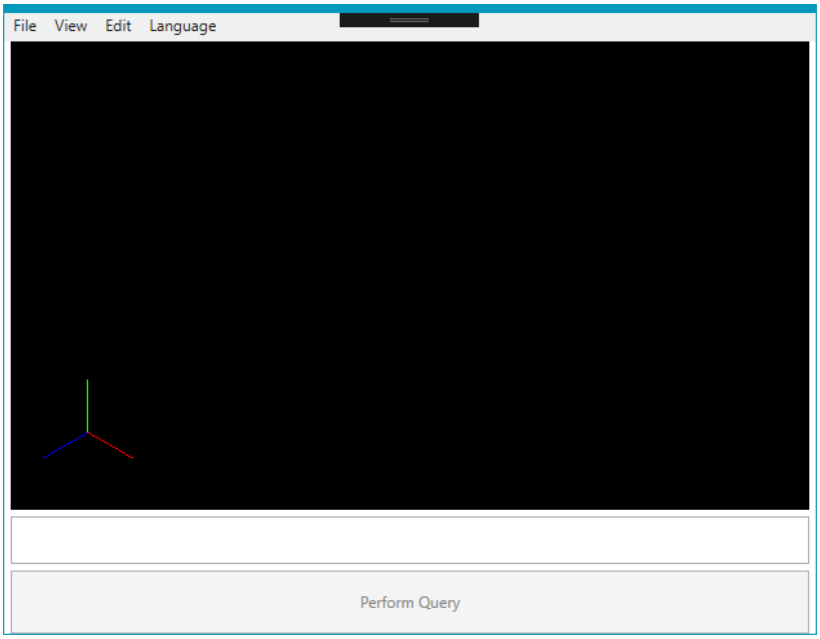

Рисунок 3.2 – Вікно програми

Для подальшого завантаження та відображення моделей спочатку необхідно створити класи для геометричних сутностей, в яких надалі буде зберігатися завантажена геометрія. Ці класи знаходяться в папці GeometricalModels.

Таблиця 3.1Описання геометричних структур

Назва класу	Опис
Point	3D точка в просторі
Vector	3D вектор в просторі
Vertex	Вершина моделі. Містить точку, нормаль та текстурні координати вершини
Edge	Ребро моделі. Складається з двох вершин.
Face	Грань моделі. Містить список вершин, нормаль та матеріал
Mesh	Представляє меш, має набір граней, ім’я та обмежувальні рамки
Model	Набір мешів
Ray	Промінь, що має початкову точку та напрямок
Surface	Площина, що описана вектором нормаллю та точкою що лежить на цій площині.
BoundingBox	Обмежувальні рамки для геометрії. Представляє собою паралелепіпед, якій містить в собі усі точки відповідної геометрії. Описується мінімальною та максимальною точками.
BoundingBoxBuilder	Клас, який створює BoundingBox для геометричних сутностей

Меню “File” має опції для завантаження, збереження файлів та виходу з програми. Підтримується такі формати файлів як “.obj” та “.stl”. Формат файлів OBJ - це простий загальноприйнятий формат даних, який містить тільки 3D геометрію та може містити ім’я файла матеріалів в форматі “.mtl”.

Формат STL зберігає інформацію лише про трикутні грані без матеріалів та може бути текстовим або бінарним. В розроблюваній системі підтримуються обидва варіанти.

Для кожного з форматів створений окремий клас, який відповідає за його завантаження та збереження, він має реалізовувати інтерфейс IFileFormatHandler. Для управління обробниками форматів файлів слугує клас GeometryFileService, який реалізує паттерн проектування “Одинак “(Singleton). Цей клас обирає необхідний обробник для файлу в залежності від необхідного формату.

```
/// <summary>
/// Loads model from file.
/// </summary>
/// <param name="path">Path to file.</param>
/// <returns>Loaded model.</returns>
public Model LoadFromFile(string path)
{
    foreach (var handler in typeHandlers.Values)
    {
        if (handler.CanLoad(path))
        {
            return handler.Load(path);
        }
    }

    return null;
}
```

Рисунок 3.3 – Метод вибору обробника формату файлу для завантаження моделі

Наступним етапом після завантаження моделі є її візуалізація. Для цього в OpenGL використовується графічний конвеєр - це послідовність кроків, які OpenGL робить під час рендерингу об’єктів. Для кожного кроку конвеєра використовуються шейдери - невеликі програми, які виконуються на графічному процесорі. Деякі з них можуть бути написані розробником.

Для використання шейдерів під час рендерингу потрібно спершу їх завантажити, скомпілювати та завантажити до шейдерної програми. Для цього створені класи Shader і ShaderProgram.

Необхідно створити вершинний та фрагментний шейдери. Вершинні шейдери зазвичай виконують перетворення в простір проекції. Для цього

необхідно передати в вершинний шейдер позицію вершини, матриці проекції, виду та моделі, а також усі параметри які використовуватиме фрагментний шейдер. Фрагментний шейдер обчислює підсумковий колір пікселя з урахуванням матеріалу, текстур, освітлення.

Реалізуємо всередині фрагментного шейдеру модель освітлення Phong, яка складається з 3 компонентів: навколишнє освітлення (ambient), розсіяне освітлення (diffuse), яке змінюється в залежності від кута падіння променів світла на поверхню за законом косинуса, та бликова (specular) компонента освітлення - світло від джерела, відбите після попадання на об’єкт, яке видно, якщо воно потрапляє в камеру(0).

```
vec3 ambient = lightAmbient * MaterialAmbient;

vec3 normal = normalize(Normal);
vec3 lightDirection = normalize(lightPosition - Position);
float diff = max(0.0, dot(normal, lightDirection));
vec3 diffuse = lightDiffuse * (diff * MaterialDiffuse);

float spec;
if (dot(normal, lightDirection) < 0.0)
{
    spec = 0;
}
else
{
    vec3 viewDir = normalize(viewPosition - Position);
    vec3 reflectDir = reflect(-lightDirection, normal);
    spec = pow(max(dot(viewDir, reflectDir), 0.0), MaterialShininess);
}

vec3 specular = lightSpecular * (spec * MaterialSpecular);
vec3 color = MaterialColor + ambient + diffuse + specular;
FragColor = vec4(color, 1.0);
```

Рисунок 3.4 – Код фрагментного шейдеру

Для роботи з матеріалами слугують класи Color, Material, Texture. Модель може мати не лише однорідний колір, але й текстуру, яка може бути задана для кожної характеристики матеріалу.

Так завдяки властивості матеріалу SpecularTexture можна задати які частини грані моделі можуть віддзеркалювати світло, наприклад металеві елементи(0).

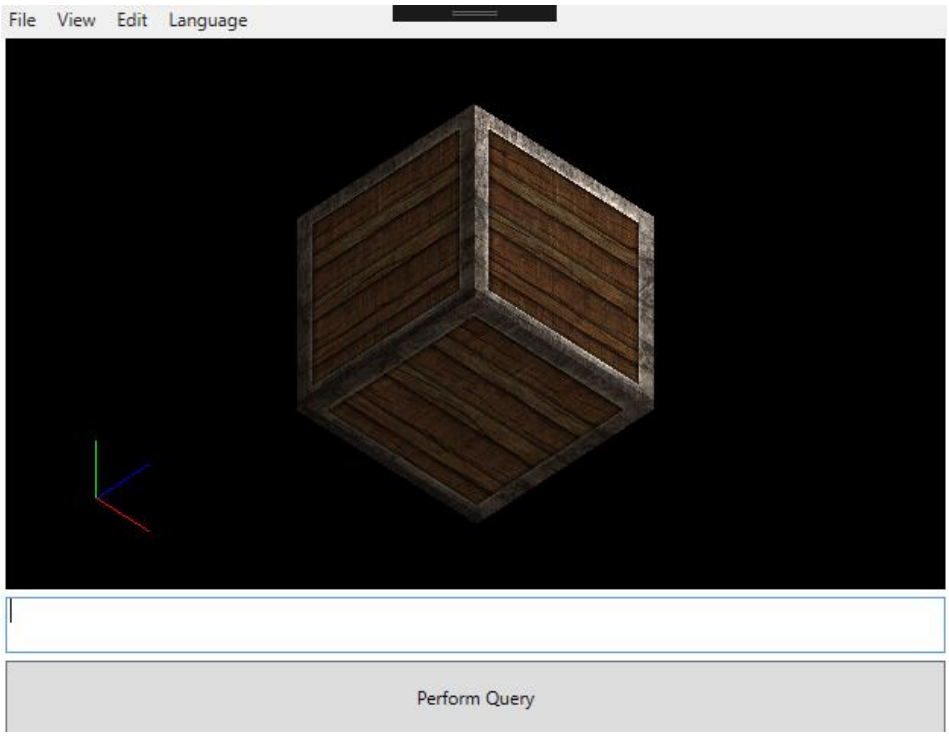

Рисунок 3.5 – Модель з текстурою

Для перегляду моделі також потрібно встановити камеру, яка визначається її положенням у світовому просторі, напрямком, на який вона дивиться, вектором, спрямованим праворуч, і вектором, спрямованим вгору від камери. Для роботи з камерами є базовий абстрактний клас Camera, та 2 його імплементації – OrthographicCamera та PerspectiveCamera. Клас камери має абстрактний метод для встановлення проекції, який має бути реалізованим в дочірніх класах. Також даний клас містить методи для встановлення позиції та вектору вверх камери, перенесення позиції, повороту, масштабування камери на певну дельту, розрахунку масштабування камери, щоб модель цілком помістилась в область перегляду.

Для повороту, масштабування та панорамування камери огляду використовуються ліва кнопка миші(LMB), правка кнопка миші(RMB) та Ctrl + RMB відповідно. Для перемикання між видами камер використовуються відповідні опції в меню “View”. У цьому меню також є опції для

встановлення передвизначених положень камери відносно моделі: зверху, знизу, зліва, справа, ззаду, спереду та ізометричний вид. Для розуміння того, в якому положенні знаходиться камера, в нижньому лівому кутку відображається тріада з координатних осей(червона ось – x, зелена – y, синя – z).

Наступним етапом створення системи є реалізація модулю інтерпретатору та обчислювальних алгоритмів для виконання запитів для моделі.

Кожен запит має вхідні параметри та значення, яке він повертає.

Типи, з якими запити працюють, включають ціле число, номер з плаваючою комою, вектор / точка у форматі (x, y, z), меш.

Запит може мати вкладену структуру, що означає, що результат запиту може бути вхідним параметром іншого запиту. Наприклад, запит:

“translate(rotate(getMeshById(0), 45, (0,1,0)), (1,0,1))”

візьме меш з ідентифікатором 0, поверне його на 45 градусів навколо осі (0,1,0), а потім перенесе на вектор (1,0,1).

Як основна структура даних, що представляє запит, вибрано впорядковане дерево, завдяки чому підтримується вкладений характер запитів.

Для дерева існує два типи вузлів:

1. CommandNode. Вузол, що представляє інший (вкладений) запит. Командний вузол завжди має дочірні елементи, оскільки будь-який запит має параметри.

2. ValueNode. Вузли значень представляють постійні значення, такі як ID мешу, що передається як параметр для команди getMeshById, або вектор перенесення для команди translate. Вузли значень - це завжди листя.

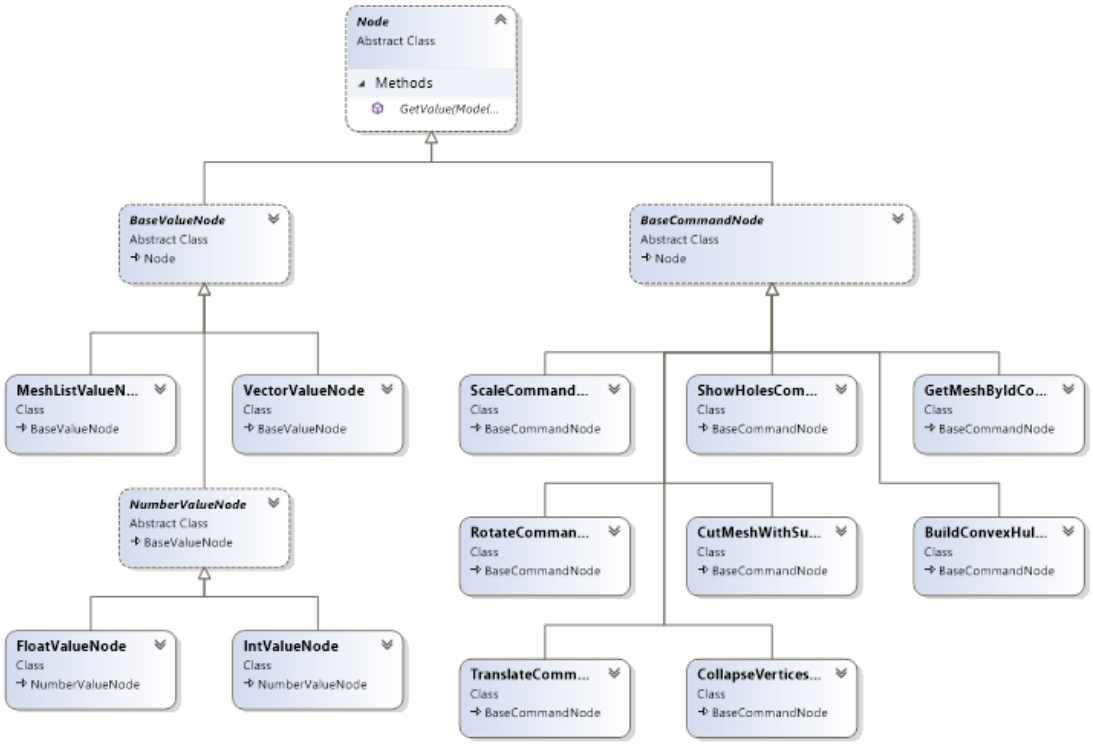

Рисунок 3.6 – Ієрархія класів вузлів дерева запитів

Дерево запиту будується у методі Parse класу Interpreter, який попередньо проводить валідацію рядку запиту. Класи CommandNodeFactory та ValueNodeFactory відповідають за вибір необхідного класу вузлів дерева.

Для виконання запиту необхідно виконати зворотний порядок обходу дерева (LRN) та виконати команди в кожному вузлі дерева.

Система також надає можливість відмінити або заново виконати відмінений запит. За даний функціонал відповідає клас QueryHistory, який керує стеками виконаних та відмінених запитів.

Списки команд, які підтримує система вказані в таблицях 3.2 та 3.3.

Таблиця 3.2 - Список команд, що не змінюють геометрію мешів

Ім’я команди	Призначення	Аргументи
getMeshById	Повертає та підсвічує меш за індексом	індекс(від 0) мешу в завантаженій моделі (тип: ціле число)
showHoles	Підсвічує отвори меша – ребра які належіть лише одній грані	меш для виділення отворів (тип: меш)

Таблиця 3.3 - Список команд, що змінюють геометрію мешів

Ім’я команди	Призначення	Аргументи
translate	Перенесення мешу на вектор	меш для перенесення (тип: меш)
		вектор перенесення (тип: вектор)
rotate	Поворот мешу	меш для обертання (тип: меш)
		кут повороту (тип: число з плаваючою крапкою)
		вісь обертання (тип: вектор)
scale	Масштабування мешу	меш для масштабування (тип: меш)
		вектор масштабування (тип: вектор)
collapseVertices	Об’єднання двох вершин мешу в одну	меш для модифікації (тип: меш)
		перша точка, що об’єднується (тип: точка)
		друга точка, що об’єднується (тип: точка)
cutMeshWithSurface	Розрізання мешу площиною	меш для розрізання (тип: меш)
		точка на ріжучій поверхні (тип: точка)
		нормаль поверхні різання (тип: вектор)
buildConvexHull FromMesh	Побудова мінімальної випуклої оболонки мешу	меш для якого будується випукла оболонка (тип: меш)

Для роботи системи були імплементовані також такі допоміжні методи:

– Векторні та матричні операції;

– Пошук перетинів променю з гранями, площини з ребрами, лініями, гранями;

– Тріангуляція граней за допомогою алгоритму відсікання «вух»(Ear clipping);

– Побудова випуклого многокутника за допомогою алгоритму монотонного ланцюгу(алгоритм Ендрю);

– Знаходження груп з’єднаних граней з використанням алгоритму перевірки зв’язності графу.

Для подальшого використання програми було створено інсталятор. Основними вимогами до нього є:

– Мінімалістичний дизайн;

– Можливість змінити шлях встановлення програми;

– Надання ліцензійної угоди з кінцевим користувачем(EULA);

– Створення ярлику програми в меню “Пуск” та папці установки;

– Створення ярлику для видалення програми.

```
var productNameWithVersion = $"{productName}_{version}";
var uninstallTitle = $"Uninstall {productName}";
var uninstallShortcutTarget = "[System64Folder]msiexec.exe";
var uninstallShortcutTargetArguments = "/x [ProductCode]";

var project = new Project(
                  productName,
                  new Dir(
                      productNameWithVersion,
                      new Files("*.*", f => !f.EndsWith(".pdb") && !f.EndsWith(".xml")),
                      new ExeFileShortcut(uninstallTitle, uninstallShortcutTarget, uninstallShortcutTargetArguments)),
                  new Dir(Path.Combine(@"%ProgramFiles%", companyName)),
                  new Dir(
                      Path.Combine(@"%ProgramMenu%", productNameWithVersion),
                      new ExeFileShortcut(uninstallTitle, uninstallShortcutTarget, uninstallShortcutTargetArguments),
                      new ExeFileShortcut(productName, $"[INSTALLDIR]{executableFileName}", arguments: string.Empty)),
                  new Dir(
                      @"%Desktop%",
                      new ExeFileShortcut(productName, $"[INSTALLDIR]{executableFileName}", arguments: string.Empty)));

project.GUID = new Guid(guid);
project.OutFileName = productNameWithVersion;
project.OutDir = Path.Combine(solutionDir, "Installer");
project.LicenceFile = Path.Combine(Environment.CurrentDirectory, "License.rtf");
project.UI = WUI.WixUI_InstallDir;
project.SourceBaseDir = Path.GetFullPath(Path.Combine(solutionDir, @"GQL\bin\Release"));
project.Version = version;
project.ControlPanelInfo.Manufacturer = companyName;

AutoElements.SupportEmptyDirectories = CompilerSupportState.Disabled;
project.BuildMsi();
```

Рисунок 3.7 – Скрипт створення інсталятора за допомогою Wix#

3.2. Використання системи

Після відкриття програми користувач бачить основні елементи інтерфейсу, проте область перегляду є пустою та деякі елементи керування неактивні. Після завантаження моделі з файлу стає доступною можливість зберегти цю модель та виконувати запити до моделі. Сама модель відображається в області перегляду в ізометричному виді з використанням ортографічної проекції та попередньо масштабується так, щоб повністю в неї поміститися(Рисунок 3.8).

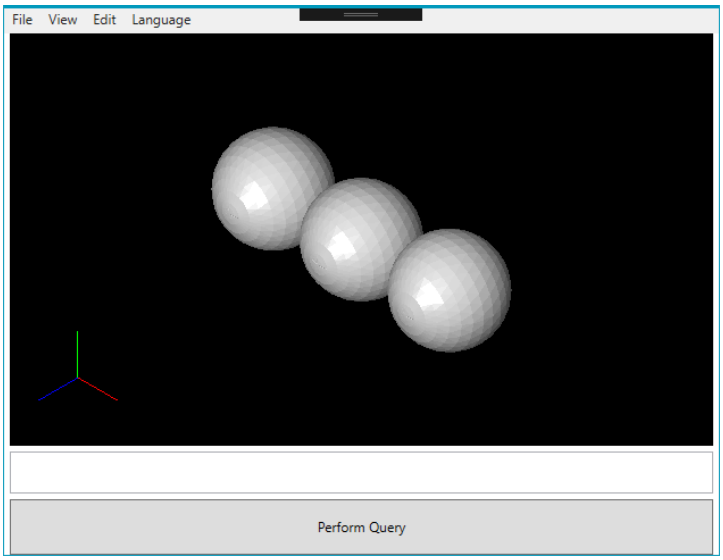

Рисунок 3.8 – Відображення моделі з ортографічною проекцією

Для зміни виду проекції необхідно вибрати бажану в меню “View”.

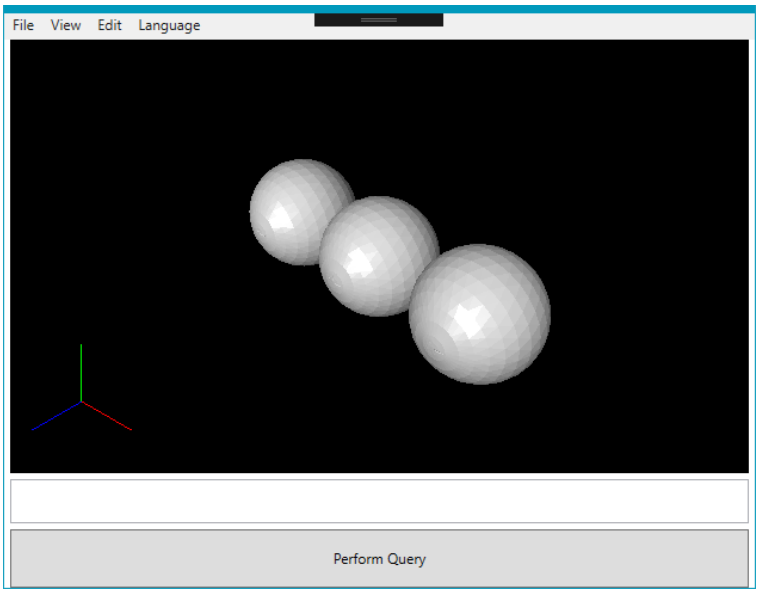

Рисунок 3.9 – Відображення моделі з перспективною проекцією

Завантажені моделі також можуть мати різні варіанти затінення - плоске та гладке (“flat and smooth shading”), які зображені на рисунках 3.10 та 3.11 відповідно.

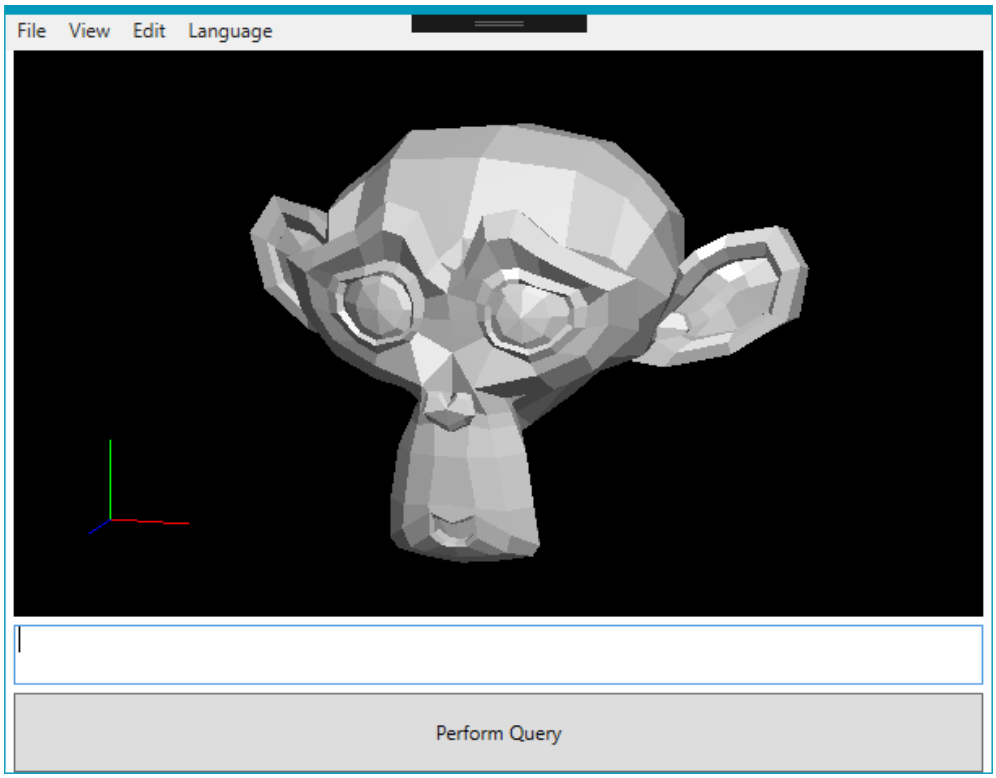

Рисунок 3.10 – Модель з плоским затіненням

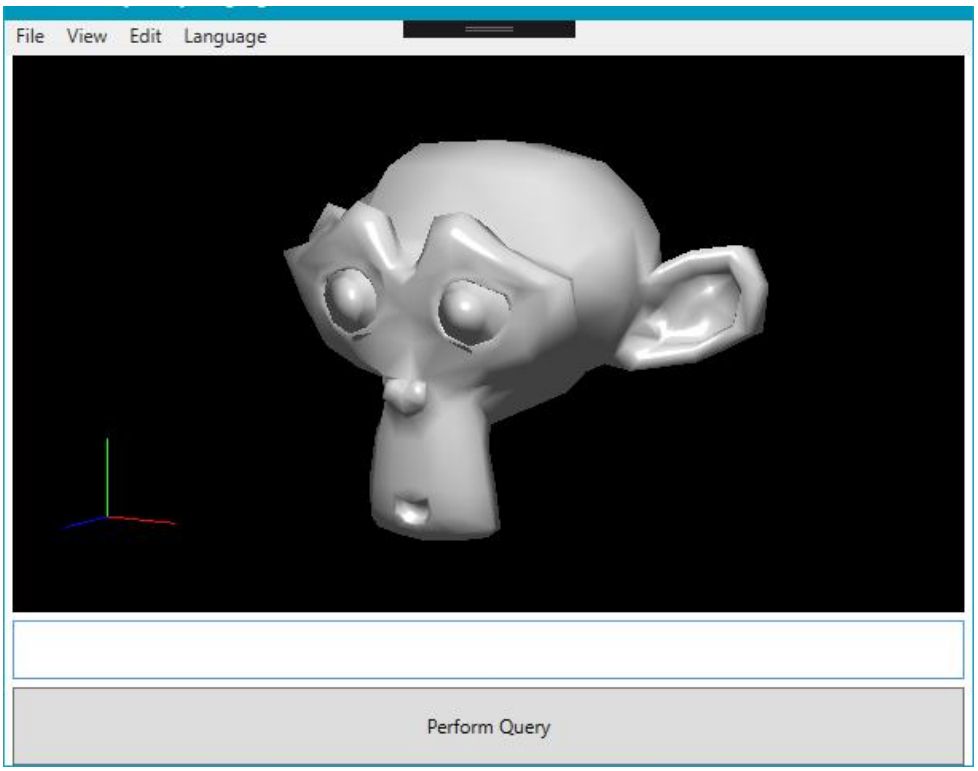

Рисунок 3.11 – Модель з гладким затіненням

Для виконання запитів необхідно внести запит в відповідне текстове поле під областю перегляду та натиснути кнопку “Perform Query”. Якщо запит э невалідним, тобто містить недопустимі символи, незбалансовані дужки, невідомі команди, неправильні типи та порядок параметрів команди, то з’явиться вікно з відповідним повідомленням(Рисунок 3.12)

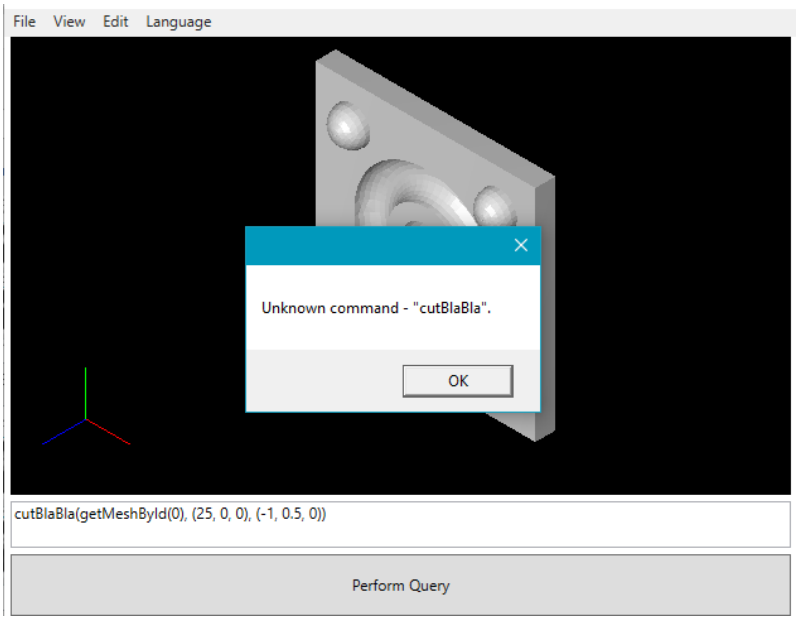

Рисунок 3.12 – Попередження про помилку в запиті

Так як не усі запити мають візуальний результат, то для розуміння того, що запит виконано успішно, буде відображено відповідне повідомлення(Рисунок 3.13).

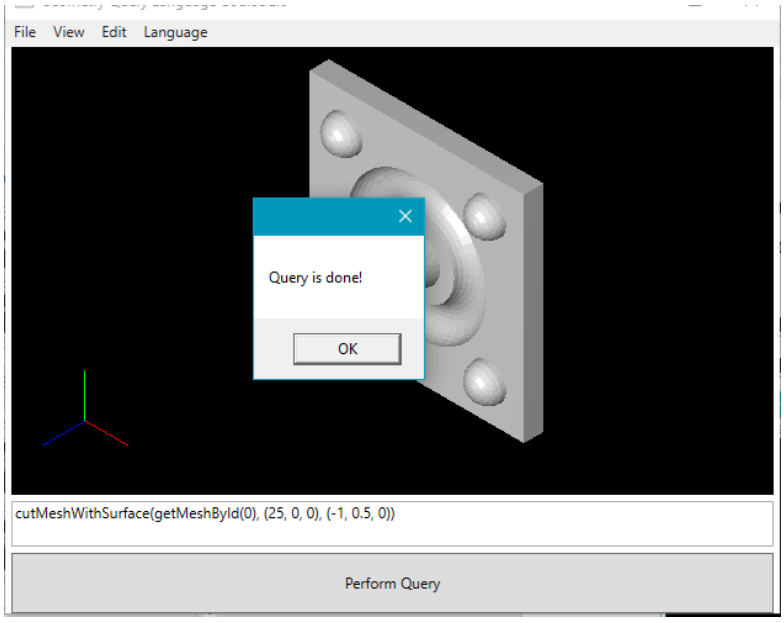

Рисунок 3.13 – Повідомлення про успішне виконання запиту

Прикладом використання вкладених запитів можуть бути наступні команди:

– cutMeshWithSurface(getMeshById(0), (25, 0, 0), (-1, 0.5, 0)) – запит бере меш з ідентифікаторм 0 та розрізає його площиною, яка проходить через точку(25, 0, 0), та має нормаль з координатами (-1, 0.5, 0).

– showHoles(translate(getMeshById(0), (-25, 0, 0))) – запит бере меш з ідентифікатором 0, переносить його на вектор (-25, 0, 0) та показує усі отвори моделі(Рисунок 3.14).

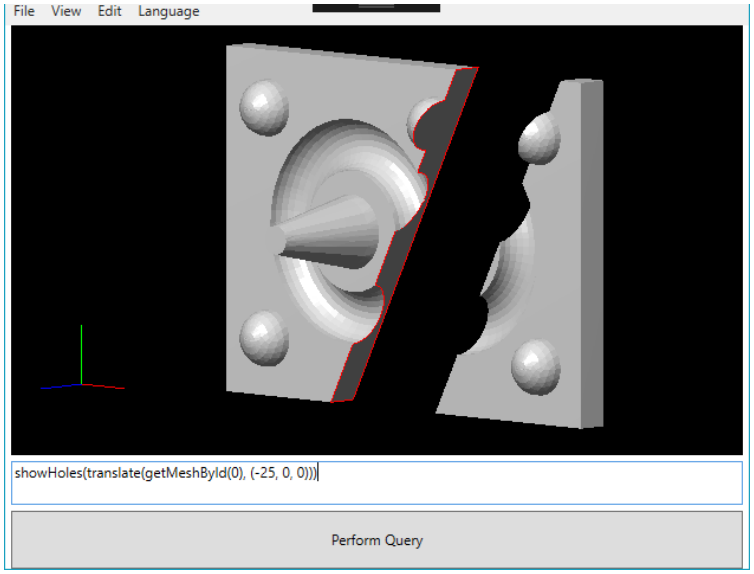

Рисунок 3.14 – Результат виконання запиту підсвічування отворів

– buildConvexHullFromMesh(getMeshById(2)) – запит бере меш з ідентифікатором 2 та будує з нього опуклу оболонку(Рисунок 3.15).

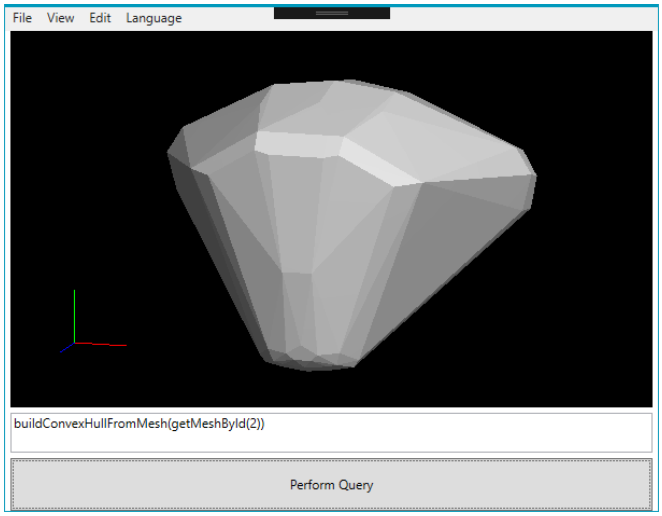

Рисунок 3.15 – Результат виконання запиту побудови опуклої оболонки

Після виконання запиту можна його відмінити за допомогою опції “Undo” в меню “Edit”. Після відміни запиту можна його виконати знову використовуючи опцію “Redo” в тому ж меню.

Система підтримує українську та англійську мову, яку можна змінити в меню “Language”.

Розроблений фреймворк має зручну та зрозумілу структуру для подальшого розширення та налаштування під конкретні вимоги. Увесь код детально задокументовано, що дозволяє легко зрозуміти призначення будь-якого класу чи методу. Система містить певний базовий функціонал та може використовуватись як основа для майбутньої кастомізації. Проект має підключені правила аналізу коду, завдяки яким забезпечується підтримка єдиного стилю та узгодженності коду.

Також система може бути використана як окрема програма для перегляду та редагування тривимірних моделей. Встановлений додаток займає мало пам’яті(728 КБ) та легко інсталюється за допомогою створеного інсталятора.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було зроблено інформаційний огляд предметної галузі та проаналізовано існуючі аналоги. За результатами аналізу була визначена актуальність розробки системи та створений перелік вимог до створюваної інформаційної системи, що полегшить процес виконання завдань в процесі вивчення предметів тривимірної графіки та обчислювальної геометрії, а також може використовуватись як основа для майбутньої кастомізації.

Було створено діаграму варіантів використання, яка дозволяє наочно відобразити попередньо визначені функціональні вимоги та типові взаємодії між користувачем та самою системою. Також спроектовано структуру системи та взаємозв’язок її компонентів.

Крім того, було проведено дослідження інструментів та засобів реалізації, в результаті якого був сформований стек технологій для розробки даної системи.

Було розроблено та програмно реалізовано систему-тренажер рендерингу моделей тривимірного простору для дисциплін напрямку "3D моделювання та візуалізація". Запланована система була реалізована згідно з визначених функціональних вимог.

Завершена система може бути використана для спрощення процесу вивчення предметів тривимірної графіки та обчислювальної геометрії, що підвищить рівень ефективності навчання в зв’язку зі скороченням трудових і часових витрат на процес створення середовища для виконання практичних завдань.

СПИСОК ЛІТЕРАТУРИ

1. Трёхмерная графика [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Трёхмерная_графика.
2. Polygonal modeling [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Polygonal_modeling.
3. Ortiz L. Types of 3D Modeling: Which Is Best for Your Needs? | All3DP [Електронний ресурс]. – Режим доступу: https://all3dp.com/2/types-of-3d-modeling/.
4. 3D modeling [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/3D_modeling.
5. Землянов Г.С., Ермолаева В.В. 3D-моделирование // Молодой ученый. Издательство Молодой ученый, 2015. № 91. P 186–189.
6. Computational geometry [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Computational_geometry.
7. De Berg M. et al. Computational geometry: Algorithms and applications // Computational Geometry: Algorithms and Applications. Springer Berlin Heidelberg, 2008. 1–386 p.
8. Сферы применения 3D-визуализации [Електронний ресурс]. – Режим доступу: https://klona.ua/blog/3d-modelirovanie/sfery-primeneniya-3d-vizualizacii.
9. Omar O., El-Messeidy R., Youssef M. Impact of 3D simulation modeling on architectural design education. 2016.
10. Negru N. et al. A new approach on 3D scanning-printing technologies with medical applications // IOP Conference Series: Materials Science and Engineering. Institute of Physics Publishing, 2019. Vol 572, № 1. P 012049.
11. Кизилов Е.Е. Применение 3D-моделирования в кино и видео-индустрии [Електронний ресурс]. 2017. – Режим доступу: http://web.snauka.ru/issues/2017/01/77658.
12. Tellez R. Why Robotics Companies MUST Use Simulators - The Construct [Електронний ресурс]. 2015. – Режим доступу:

https://www.theconstructsim.com/why-robotics-companies-must-use-simulators/.

13. Jarratt S. The best 3D modelling software in 2020 [Електронний ресурс]. 2020. – Режим доступу: https://www.creativebloq.com/features/best-3d-modelling-software.
14. Autodesk Maya [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Autodesk_Maya.
15. SOLIDWORKS 2016: краткий обзор программы [Електронний ресурс]. – Режим доступу: https://3ddevice.com.ua/blog/3d-printer-obzor/obzor-programmy-solidworks/.
16. How to develop add-ins for SOLIDWORKS automation via API [Електронний ресурс]. – режим доступу: https://www.codestack.net/solidworks-api/getting-started/add-ins/.
17. Офіційний сайт Blender [Електронний ресурс]. – Режим доступу: https://www.blender.org/.
18. Blender [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Blender.
19. Wings 3D Tutorial: Prepare your model for 3D Printing with Wings 3D [Електронний ресурс]. – Режим доступу: https://www.sculpteo.com/en/tutorial/prepare-your-model-3d-printing-wings3d/.
20. Wings 3D - Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Wings_3D.
21. Reina G., Müller T., Ertl T. Incorporating modern OpenGL into computer graphics education // IEEE Comput. Graph. Appl. 2014. Vol 34, № 4.
22. Fink H., Weber T., Wimmer M. Teaching a modern graphics pipeline using a shader-based software renderer // Comput. Graph. Elsevier Ltd, 2013. Vol 37, № 1–2. P 12–20.
23. A Tour of C# - C# Guide | Microsoft Docs [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.

24. NET Core - Wikipedia [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/.NET_Core (accessed: 08.12.2020).
25. Java Overview | CoreJavaGuru [Електронний ресурс]. – Режим доступу: http://www.corejavaguru.com/java/basic/overview.
26. Choose your Windows app platform - Windows applications | Microsoft Docs [Електронний ресурс]. – Режим доступу: https://docs.microsoft.com/en-us/windows/apps/desktop/choose-your-platform.
27. Winforms vs WPF | Learn The Top 6 Most Awesome Differences [Електронний ресурс]. – Режим доступу: https://www.educba.com/winforms-vs-wpf/.
28. GitHub - OpenTK [Електронний ресурс]. – Режим доступу: https://github.com/opentk/opentk.
29. Shilo O. GitHub - wixsharp: Framework for building a complete MSI or WiX source code by using script files written with the C# syntax [Електронний ресурс]. – Режим доступу: https://github.com/oleg-shilo/wixsharp.

ДОДАТКИ

Current project is the property of the AMC Bridge company.

Додаток А. Шейдери

Shader.vert – вершинний шейдер

```
#version 330
layout(location = 0) in vec3 position;
layout(location = 1) in vec3 normal;
layout(location = 2) in vec2 textureCoordinates;
layout(location = 3) in vec3 materialColor;
layout(location = 4) in vec3 materialAmbient;
layout(location = 5) in vec3 materialDiffuse;
layout(location = 6) in vec3 materialSpecular;
layout(location = 7) in float materialShininess;
layout(location = 8) in int appliedTextures;

uniform mat4 V;
uniform mat4 P;

out vec3 Position;
out vec3 Normal;
out vec2 TextureCoordinates;
out vec3 MaterialColor;
out vec3 MaterialAmbient;
out vec3 MaterialDiffuse;
out vec3 MaterialSpecular;
out float MaterialShininess;
flat out int AppliedTextures;

void main()
{
       Position = vec3(V * vec4(position, 1.0));
    Normal = normal;
       TextureCoordinates = textureCoordinates;
       MaterialColor = materialColor;
       MaterialAmbient = materialAmbient;
       MaterialDiffuse = materialDiffuse;
       MaterialSpecular = materialSpecular;
       MaterialShininess = materialShininess;
       AppliedTextures = appliedTextures;

    gl_Position =  P * V * vec4(position,1);
       }
```

Shader.frag - Фрагментний шейдер

```
#version 330
out vec4 FragColor;

in vec3 Position;
in vec3 Normal;
in vec2 TextureCoordinates;
in vec3 MaterialColor;
in vec3 MaterialAmbient;
in vec3 MaterialDiffuse;
in vec3 MaterialSpecular;
in float MaterialShininess;
flat in int AppliedTextures;

uniform vec3 lightPosition;
```

```
uniform vec3 lightAmbient;
uniform vec3 lightDiffuse;
uniform vec3 lightSpecular;

uniform vec3 viewPosition;

uniform sampler2D ambientTexture;
uniform sampler2D diffuseTexture;
uniform sampler2D specularTexture;
const int ambientTextureFlag = 1;
const int diffuseTextureFlag = 2;
const int specularTextureFlag = 4;

uniform int mode;

void main()
{
	if (mode == 0)
	{
		FragColor = vec4(MaterialColor, 1.0);
	}
	else
	{
		vec3 ambient = lightAmbient * MaterialAmbient;

		vec3 normal = normalize(Normal);
		vec3 lightDirection = normalize(lightPosition - Position);
		float diff = max(0.0, dot(normal, lightDirection));
		vec3 diffuse = lightDiffuse * (diff * MaterialDiffuse);

		float spec;
		if (dot(normal, lightDirection) < 0.0)
		{
			spec = 0;
		}
		else
		{
			vec3 viewDir = normalize(viewPosition - Position);
			vec3 reflectDir = reflect(-lightDirection, normal);
			spec = pow(max(dot(viewDir, reflectDir), 0.0), MaterialShininess);
		}

		vec3 specular = lightSpecular * (spec * MaterialSpecular);

		if ((AppliedTextures & ambientTextureFlag) == ambientTextureFlag)
		{
			ambient = ambient * vec3(texture2D(ambientTexture,
TextureCoordinates));
		}

		if ((AppliedTextures & diffuseTextureFlag) == diffuseTextureFlag)
		{
			diffuse = diffuse * vec3(texture2D(diffuseTexture,
TextureCoordinates));
		}

		if ((AppliedTextures & specularTextureFlag) == specularTextureFlag)
		{
			specular = specular * vec3(texture2D(specularTexture,
TextureCoordinates));
		}

		vec3 color = MaterialColor + ambient + diffuse + specular;
		FragColor = vec4(color, 1.0);
```

```
        }
        }
```

Додаток Б. Код системи мовою C#

1. IFileFormatHandler.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GeometryQueryLanguage.GeometricalModels;

namespace GeometryQueryLanguage.FileFormatHandlers
{
    /// <summary>
    /// Handles file save and load operations.
    /// </summary>
    public interface IFileFormatHandler
    {
        /// <summary>
        /// Returns whether model can be loaded from file by filehandler.
        /// </summary>
        /// <param name="path">Path to file.</param>
        /// <returns>Whether file can be loaded by filehandler.</returns>
        bool CanLoad(string path);

        /// <summary>
        /// Saves model to file.
        /// </summary>
        /// <param name="path">Path to file.</param>
        /// <param name="model">Model to be saved.</param>
        void Save(string path, Model model);

        /// <summary>
        /// Loads model from file.
        /// </summary>
        /// <param name="path">Path to file.</param>
        /// <returns>Loaded model.</returns>
        Model Load(string path);
    }
}
```

2. ObjFormatHandler.cs

```
// ***********************************************************************
// Copyright FileName - ObjFormatHandler.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
// FileType - Visual C# Source File
// ***********************************************************************
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GeometryQueryLanguage.GeometricalModels;
```

```
using GeometryQueryLanguage.Materials;

namespace GeometryQueryLanguage.FileFormatHandlers
{
    /// <summary>
    /// Handles save and load operations with obj file format.
    /// </summary>
    public class ObjFormatHandler : IFileFormatHandler
    {
        #region IFileFormatHandler implementation
        /// <summary>
        /// Returns whether model can be loaded from file by filehandler.
        /// </summary>
        /// <param name="path">Path to file.</param>
        /// <returns>Whether file can be loaded by filehandler.</returns>
        public bool CanLoad(string path)
        {
            return Path.GetExtension(path) == FILE_TYPE;
        }

        /// <summary>
        /// Loads model from .obj file.
        /// </summary>
        /// <param name="path">Path to .obj file.</param>
        /// <returns>Loaded model.</returns>
        public Model Load(string path)
        {
            loadPath = path;
            loadedModel = new Model();
            loadedMesh = new Mesh();
            allPoints = new List<Point>();
            allNormals = new List<Vector>();
            allTexturesCoordinates = new List<TextureCoordinate>();
            materials = new Dictionary<string, Material>();
            currentMaterial = new Material();
            var loadActions = new Dictionary<string, Action<string>>
            {
                { VERTEX_KEYWORD, AddPoint },
                { TEXTURE_COORDINATES_KEYWORD, AddTextureCoordinates },
                { NORMAL_KEYWORD, AddNormal },
                { OBJECT_KEYWORD, AddMesh },
                { FACE_KEYWORD, AddFace },
                { MTLLIB_KEYWORD, LoadMaterials },
                { USEMTL_KEYWORD, UseMaterial },
            };

            foreach (var line in File.ReadLines(path))
            {
                var tokens = line.Split(new char[] { ' ' }, 2,
StringSplitOptions.RemoveEmptyEntries);
                if (tokens.Length > 0 && loadActions.ContainsKey(tokens[0]))
                {
                    loadActions[tokens[0]](tokens[1]);
                }
            }


loadedModel.Meshes.AddRange(ComputationalModule.MeshTransformation.GetSeparateMeshes(load
edMesh));
            loadedModel.BoundingBox = new
BoundingBoxBuilder().AddModel(loadedModel).ToBoundingBox();
            return loadedModel;
        }
```

```
        /// <summary>
        /// Saves model to file.
        /// </summary>
        /// <param name="path">Path to file.</param>
        /// <param name="model">Model to be saved.</param>
        public void Save(string path, Model model)
        {
            using (var writer = new StreamWriter(path))
            {
                var points = new List<Point>();
                var normals = new List<Vector>();
                var textureCoordinates = new List<TextureCoordinate>();
                var pointsIndices = new Dictionary<Point, int>();
                var normalsIndices = new Dictionary<Vector, int>();
                var textureCoordinatesIndices = new Dictionary<TextureCoordinate, int>();
                var facesElements = new List<string>();
                var meshesFacesElements = new List<List<string>>();

                var mtlFilename = Path.GetFileNameWithoutExtension(path) + MTL_FILE_TYPE;
                var mtlFilePath =
Path.Combine(Path.GetDirectoryName(Path.GetFullPath(path)), mtlFilename);
                writer.WriteLine($"{MTLLIB_KEYWORD} {mtlFilename}");
                new MtlFormatHandler().SaveMaterials(mtlFilePath, model.GetMaterials());
                string currentMaterialName = string.Empty;
                foreach (var mesh in model.Meshes)
                {
                    foreach (var face in mesh.Faces)
                    {
                        var faceStringBuilder = new StringBuilder();
                        if (!face.Material.Name.Equals(currentMaterialName))
                        {
                            faceStringBuilder.AppendLine($"{USEMTL_KEYWORD}
{face.Material.Name}");
                            currentMaterialName = face.Material.Name;
                        }

                        faceStringBuilder.Append(FACE_KEYWORD);
                        foreach (var vertex in face.Vertices)
                        {
                            var pointIndex = FindElementIndex(points, pointsIndices,
vertex.Point);
                            var normalIndex = FindElementIndex(normals, normalsIndices,
vertex.Normal ?? face.Normal);

                            faceStringBuilder.Append($" {pointIndex}/");
                            if (vertex.TextureCoordinates != null)
                            {
                                var textureCoordinateIndex =
FindElementIndex(textureCoordinates, textureCoordinatesIndices,
vertex.TextureCoordinates);
                                faceStringBuilder.Append($"{textureCoordinateIndex}");
                            }

                            faceStringBuilder.Append($"/{normalIndex}");
                        }

                        facesElements.Add(faceStringBuilder.ToString());
                    }

                    meshesFacesElements.Add(facesElements);
                    facesElements = new List<string>();
                }
```

```
                points.ForEach(point => writer.WriteLine($"{VERTEX_KEYWORD} {point.X}
{point.Y} {point.Z}"));
                textureCoordinates.ForEach(textureCoordinate =>
writer.WriteLine($"{TEXTURE_COORDINATES_KEYWORD} {textureCoordinate.U}
{textureCoordinate.V}"));
                normals.ForEach(normal => writer.WriteLine($"{NORMAL_KEYWORD} {normal.X}
{normal.Y} {normal.Z}"));
                for (var i = 0; i < model.Meshes.Count; i++)
                {
                    writer.WriteLine($"{OBJECT_KEYWORD} {model.Meshes[i].Name}");
                    meshesFacesElements[i].ForEach(face => writer.WriteLine(face));
                }
            }
        }
        #endregion

        #region Private Logic
        private static float[] ParseCoords(string s)
        {
            return s
                .Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries)
                .Select(x => float.Parse(x))
                .ToArray();
        }

        private static int[] ParseIndices(string s)
        {
            return s
                .Split(new char[] { '/' })
                .Select(x =>
                {
                    int.TryParse(x, out int result);
                    return result;
                })
                .ToArray();
        }

        private static int ConvertIndex(int index, int elementCount)
        {
            if (index > 0)
            {
                return index - 1;
            }
            else if (index < 0)
            {
                return elementCount + index;
            }
            else
            {
                return -1;
            }
        }

        private static int FindElementIndex<T>(ICollection<T> list, IDictionary<T, int>
elementsIndices, T element)
        {
            if (!elementsIndices.ContainsKey(element))
            {
                elementsIndices.Add(element, list.Count);
                list.Add(element);
            }

            var index = elementsIndices[element] + 1;
            return index;
```

```
        }

        private void AddPoint(string pointCoords)
        {
            var pointCoordsArray = ParseCoords(pointCoords);
            if (pointCoordsArray.Length == 3)
            {
                var point = new Point(pointCoordsArray[0], pointCoordsArray[1],
pointCoordsArray[2]);
                allPoints.Add(point);
            }
        }

        private void AddNormal(string normalCoords)
        {
            var normalCoordsArray = ParseCoords(normalCoords);
            if (normalCoordsArray.Length == 3)
            {
                var normal = new Vector(normalCoordsArray[0], normalCoordsArray[1],
normalCoordsArray[2]);
                allNormals.Add(normal);
            }
        }

        private void AddTextureCoordinates(string textureCoordinatesString)
        {
            var textureCoordinatesArray = ParseCoords(textureCoordinatesString);
            if (textureCoordinatesArray.Length == 2)
            {
                var textureCoordinates = new
TextureCoordinate(textureCoordinatesArray[0], textureCoordinatesArray[1]);
                allTexturesCoordinates.Add(textureCoordinates);
            }
        }

        private void AddMesh(string meshName)
        {
            if (loadedMesh.Faces.Count > 0)
            {

loadedModel.Meshes.AddRange(ComputationalModule.MeshTransformation.GetSeparateMeshes(load
edMesh));
                loadedMesh = new Mesh();
            }

            loadedMesh.Name = meshName;
        }

        private void AddFace(string vertexSet)
        {
            var vertexTriplets = vertexSet.Split(new char[] { ' ' },
StringSplitOptions.RemoveEmptyEntries);
            var face = new Face();
            for (var i = 0; i < vertexTriplets.Length; i++)
            {
                int[] vertexIndices = ParseIndices(vertexTriplets[i]);
                if (vertexIndices.Length > 0)
                {
                    face.Vertices.Add(ParseVertex(vertexIndices));
                }
            }

            face.GenerateNormal();
            face.Material = currentMaterial;
```

```
            loadedMesh.Faces.Add(face);
        }

        private Vertex ParseVertex(int[] vertexIndices)
        {
            var vertex = new Vertex();
            var pointIndex = ConvertIndex(vertexIndices[0], allPoints.Count);
            vertex.Point = allPoints[pointIndex];

            if (vertexIndices.Length >= 2)
            {
                var textureCoordinatesIndex = ConvertIndex(vertexIndices[1],
allTexturesCoordinates.Count);
                if (textureCoordinatesIndex >= 0 && textureCoordinatesIndex <
allTexturesCoordinates.Count)
                {
                    vertex.TextureCoordinates =
allTexturesCoordinates[textureCoordinatesIndex];
                }
            }

            if (vertexIndices.Length == 3)
            {
                var normalIndex = ConvertIndex(vertexIndices[2], allNormals.Count);
                if (normalIndex >= 0 && normalIndex < allNormals.Count)
                {
                    vertex.Normal = allNormals[normalIndex];
                }
            }

            return vertex;
        }

        private void AddMaterialsToDictionary(IEnumerable<Material> list,
IDictionary<string, Material> materials)
        {
            foreach (var material in list)
            {
                if (!materials.ContainsKey(material.Name))
                {
                    materials.Add(material.Name, material);
                }
            }
        }

        private void LoadMaterials(string mtlFilename)
        {
            var filenames = mtlFilename.Split(new char[] { ' ' },
StringSplitOptions.RemoveEmptyEntries);
            var mtlFormatHandler = new MtlFormatHandler();
            foreach (var filename in filenames)
            {
                var mtlPath =
Path.Combine(Path.GetDirectoryName(Path.GetFullPath(loadPath)), filename);
                AddMaterialsToDictionary(mtlFormatHandler.LoadMaterials(mtlPath),
materials);
            }
        }

        private void UseMaterial(string materialName)
        {
            currentMaterial = !string.IsNullOrWhiteSpace(materialName) &&
materials.ContainsKey(materialName)
                                        ? materials[materialName] : new Material();
```

```
        }

        #endregion

        #region Fields
        private const string FILE_TYPE = ".obj";
        private const string VERTEX_KEYWORD = "v";
        private const string TEXTURE_COORDINATES_KEYWORD = "vt";
        private const string NORMAL_KEYWORD = "vn";
        private const string FACE_KEYWORD = "f";
        private const string OBJECT_KEYWORD = "o";
        private const string MTLLIB_KEYWORD = "mtllib";
        private const string USEMTL_KEYWORD = "usemtl";
        private const string MTL_FILE_TYPE = ".mtl";

        private string loadPath;
        private Model loadedModel;
        private Mesh loadedMesh;
        private List<Point> allPoints;
        private List<Vector> allNormals;
        private List<TextureCoordinate> allTexturesCoordinates;
        private Dictionary<string, Material> materials;
        private Material currentMaterial;
        #endregion
    }
}
```

3. MtlFormatHandler.cs

```
// ***********************************************************************************
// Copyright FileName - MtlFormatHandler.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
// FileType - Visual C# Source File
// ***********************************************************************************
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GeometryQueryLanguage.GeometricalModels;
using GeometryQueryLanguage.Materials;

namespace GeometryQueryLanguage.FileFormatHandlers
{
    /// <summary>
    /// Handles save and load operations with mtl file format.
    /// </summary>
    public class MtlFormatHandler
    {
        #region Public Logic
        /// <summary>
        /// Loads materials from .mtl file.
        /// </summary>
        /// <param name="path">Path to .mtl file.</param>
        /// <returns>Loaded materials.</returns>
        public IList<Material> LoadMaterials(string path)
        {
            if (!File.Exists(path))
            {

System.Windows.MessageBox.Show(string.Format(Resources.Strings.MtlFileNotFound, path));
```

```
                return loadedMaterials;
            }

            foreach (var line in File.ReadLines(path))
            {
                var tokens = line.Replace('\t', ' ').Split(new char[] { ' ' }, 2,
StringSplitOptions.RemoveEmptyEntries);
                if (tokens.Length < 2)
                {
                    continue;
                }

                if (colorSetters.ContainsKey(tokens[0]))
                {
                    AddColor(tokens[0], tokens[1]);
                }
                else if (textureSetters.ContainsKey(tokens[0]))
                {
                    LoadTexture(tokens[0], tokens[1], path);
                }
                else
                {
                    LoadData(tokens);
                }
            }

            loadedMaterials.Add(currentMaterial);
            return loadedMaterials;
        }

        /// <summary>
        /// Saves materials to file.
        /// </summary>
        /// <param name="path">Path to .mtl file.</param>
        /// <param name="materials">Materials to be saved.</param>
        public void SaveMaterials(string path, IEnumerable<Material> materials)
        {
            using (var writer = new StreamWriter(path))
            {
                foreach (var material in materials)
                {
                    var materialStringBuilder = new StringBuilder();
                    materialStringBuilder.AppendLine($"{NEW_MATERIAL_KEYWORD}
{material.Name}");
                    materialStringBuilder.AppendLine($"{AMBIENT_KEYWORD}
{material.Ambient}");
                    materialStringBuilder.AppendLine($"{DIFFUSE_KEYWORD}
{material.Diffuse}");
                    materialStringBuilder.AppendLine($"{SPECULAR_KEYWORD}
{material.Specular}");
                    materialStringBuilder.AppendLine($"{SHININESS_KEYWORD}
{material.Shininess}");

                    AddTexture(material.AmbientTexture, materialStringBuilder, path,
AMBIENT_TEXTURE_MAP_KEYWORD);
                    AddTexture(material.DiffuseTexture, materialStringBuilder, path,
DIFFUSE_TEXTURE_MAP_KEYWORD);
                    AddTexture(material.SpecularTexture, materialStringBuilder, path,
SPECULAR_TEXTURE_MAP_KEYWORD);

                    writer.Write(materialStringBuilder.ToString());
                }
            }
        }
```

```
        #endregion

        #region Private Logic
        private static void AddTexture(Texture texture, StringBuilder
materialStringBuilder, string path, string textureKeyword)
        {
            if (texture != null)
            {
                texture.Save(Path.GetDirectoryName(Path.GetFullPath(path)));
                materialStringBuilder.AppendLine($"{textureKeyword} {texture.Filename}");
            }
        }

        private void LoadTexture(string textureKeyword, string textureFilename, string
path)
        {
            if (!textureSetters.ContainsKey(textureKeyword))
            {
                return;
            }

            var textureImagePath =
Path.Combine(Path.GetDirectoryName(Path.GetFullPath(path)), textureFilename);
            try
            {
                var texture = new Texture(textureImagePath);
                textureSetters[textureKeyword](currentMaterial, texture);
            }
            catch (Exception)
            {

System.Windows.MessageBox.Show(string.Format(Resources.Strings.CantLoadTexturefromFile,
textureImagePath));
            }
        }

        private void AddColor(string colorKeyword, string value)
        {
            if (colorSetters.ContainsKey(colorKeyword) && Color.TryParse(value, out Color
color))
            {
                colorSetters[colorKeyword](currentMaterial, color);
            }
        }

        private void AddMaterial(string materialName)
        {
            if (currentMaterial != null)
            {
                loadedMaterials.Add(currentMaterial);
            }

            currentMaterial = new Material();
            if (!string.IsNullOrWhiteSpace(materialName))
            {
                currentMaterial.Name = materialName;
            }
        }

        private void LoadData(string[] tokens)
        {
            switch (tokens[0])
            {
```

```
                case NEW_MATERIAL_KEYWORD:
                    AddMaterial(tokens[1]);
                    break;
                case SHININESS_KEYWORD:
                    if (currentMaterial != null && float.TryParse(tokens[1], out float
shininess))
                    {
                        currentMaterial.Shininess = shininess;
                    }

                    break;
                default:
                    break;
            }
        }
        #endregion

        #region Fields
        private const string NEW_MATERIAL_KEYWORD = "newmtl";
        private const string AMBIENT_KEYWORD = "Ka";
        private const string DIFFUSE_KEYWORD = "Kd";
        private const string SPECULAR_KEYWORD = "Ks";
        private const string SHININESS_KEYWORD = "Ns";
        private const string AMBIENT_TEXTURE_MAP_KEYWORD = "map_Ka";
        private const string DIFFUSE_TEXTURE_MAP_KEYWORD = "map_Kd";
        private const string SPECULAR_TEXTURE_MAP_KEYWORD = "map_Ks";

        private readonly IList<Material> loadedMaterials = new List<Material>();

        private readonly IDictionary<string, Action<Material, Color>> colorSetters = new
Dictionary<string, Action<Material, Color>>
        {
            { AMBIENT_KEYWORD, (material, color) => material.Ambient = color },
            { DIFFUSE_KEYWORD, (material, color) => material.Diffuse = color },
            { SPECULAR_KEYWORD, (material, color) => material.Specular = color },
        };

        private readonly IDictionary<string, Action<Material, Texture>> textureSetters =
new Dictionary<string, Action<Material, Texture>>
        {
            { AMBIENT_TEXTURE_MAP_KEYWORD, (material, texture) => material.AmbientTexture
= texture },
            { DIFFUSE_TEXTURE_MAP_KEYWORD, (material, texture) => material.DiffuseTexture
= texture },
            { SPECULAR_TEXTURE_MAP_KEYWORD, (material, texture) =>
material.SpecularTexture = texture },
        };

        private Material currentMaterial;
        #endregion
    }
}
```

4. Model.cs

```
// ***********************************************************************
// Copyright FileName - Model.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
// FileType - Visual C# Source File
// ***********************************************************************
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GeometryQueryLanguage.Materials;

namespace GeometryQueryLanguage.GeometricalModels
{
    /// <summary>
    /// The Model class is a set of Mesh objects.
    /// </summary>
    public class Model
    {
        #region Properties
        /// <summary>
        /// Gets or sets meshes.
        /// </summary>
        public List<Mesh> Meshes { get; set; } = new List<Mesh>();

        /// <summary>
        /// Gets or sets bounding box.
        /// </summary>
        public BoundingBox BoundingBox { get; set; }

        /// <summary>
        /// Gets or sets holes edges.
        /// </summary>
        public List<Edge> HolesEdges { get; set; }
        #endregion

        #region Constructors
        /// <summary>
        /// Initializes a new instance of the <see cref="Model"/> class.
        /// </summary>
        public Model()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Model"/> class.
        /// </summary>
        /// <param name="model">Model to be copied.</param>
        public Model(Model model)
        {
            if (model != null)
            {
                Meshes = model.Meshes.Select(mesh => new Mesh(mesh)).ToList();
                BoundingBox = model.BoundingBox;
                HolesEdges = model.HolesEdges;
            }
        }
        #endregion

        #region Public Logic
        /// <summary>
        /// Copies properties of another model.
        /// </summary>
        /// <param name="anotherModel">Another model.</param>
        public void CopyProperties(Model anotherModel)
        {
            if (anotherModel != null)
            {
                Meshes = anotherModel.Meshes.Select(mesh => new Mesh(mesh)).ToList();
```

```
                BoundingBox = anotherModel.BoundingBox;
                HolesEdges = anotherModel.HolesEdges;
            }
        }

        /// <summary>
        /// Restore highlighting state.
        /// </summary>
        /// <param name="highlightingState">The highlighting state.</param>
        public void RestoreHighlight(HighlightingState highlightingState)
        {
            HolesEdges = highlightingState.HolesEdges;
            for (var i = 0; i < Meshes.Count; i++)
            {
                Meshes[i].IsHighlighted = highlightingState.HighlightedMeshes[i];
            }
        }

        /// <summary>
        /// Gets all materials that are used in model.
        /// </summary>
        /// <returns>All materials that are used in model.</returns>
        public IEnumerable<Material> GetMaterials()
        {
            return Meshes
                .SelectMany(mesh => mesh.Faces)
                .Select(face => face.Material)
                .GroupBy(material => material.Name)
                .Select(group => group.First());
        }

        /// <summary>
        /// Gets all textures that are used in model.
        /// </summary>
        /// <returns>All textures that are used in model.</returns>
        public IEnumerable<Texture> GetTextures()
        {
            return GetMaterials()
                .SelectMany(material => material.GetTextures())
                .Where(texture => texture != null);
        }
        #endregion
    }
}
```

5. Mesh.cs

```
// ***********************************************************************
// Copyright FileName - Mesh.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
// FileType - Visual C# Source File
// ***********************************************************************
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GeometryQueryLanguage.GeometricalModels
{
    /// <summary>
    /// A mesh is a set of faces.
    /// </summary>
```

```
    public class Mesh
    {
        #region Properties
        /// <summary>
        /// Gets or sets mesh faces.
        /// </summary>
        public List<Face> Faces { get; set; } = new List<Face>();

        /// <summary>
        /// Gets or sets mesh name.
        /// </summary>
        public string Name { get; set; } = "default";

        /// <summary>
        /// Gets or sets bounding box.
        /// </summary>
        public BoundingBox BoundingBox { get; set; }

        /// <summary>
        /// Gets or sets mesh color.
        /// </summary>
        public Vector Color { get; set; } = new Vector(0.7f, 0.7f, 0.7f);

        /// <summary>
        /// Gets or sets a value indicating whether mesh is highlighted.
        /// </summary>
        public bool IsHighlighted { get; set; }
        #endregion

        #region Constructors
        /// <summary>
        /// Initializes a new instance of the <see cref="Mesh"/> class.
        /// </summary>
        public Mesh()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Mesh"/> class.
        /// </summary>
        /// <param name="mesh">Mesh to be copied.</param>
        public Mesh(Mesh mesh)
        {
            Name = mesh.Name;

            var boundingBoxBuilder = new BoundingBoxBuilder();

            foreach (var face in mesh.Faces)
            {
                var newFace = new Face(face);
                Faces.Add(newFace);
                boundingBoxBuilder.AddFace(newFace);
            }

            BoundingBox = boundingBoxBuilder.ToBoundingBox();
            IsHighlighted = mesh.IsHighlighted;
        }
        #endregion
    }
}
```

6. Face.cs

```
// ****************************************************************************
// Copyright FileName - Face.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
// FileType - Visual C# Source File
// ****************************************************************************
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GeometryQueryLanguage.Materials;

namespace GeometryQueryLanguage.GeometricalModels
{
    /// <summary>
    /// A Face is a set of vertices and a normal.
    /// </summary>
    public class Face
    {
        #region Properties
        /// <summary>
        /// Gets or sets the vertices.
        /// </summary>
        public List<Vertex> Vertices { get; set; } = new List<Vertex>();

        /// <summary>
        /// Gets or sets the face normal.
        /// </summary>
        public Vector Normal { get; set; }

        /// <summary>
        /// Gets or sets the face material.
        /// </summary>
        public Material Material { get; set; } = new Material();
        #endregion

        #region Constructors
        /// <summary>
        /// Initializes a new instance of the <see cref="Face"/> class.
        /// </summary>
        public Face()
        {
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Face"/> class.
        /// </summary>
        /// <param name="points">Face points.</param>
        public Face(IEnumerable<Point> points)
        {
            foreach (var point in points)
            {
                Vertices.Add(new Vertex(point));
            }

            GenerateNormal();
        }

        /// <summary>
        /// Initializes a new instance of the <see cref="Face"/> class.
        /// </summary>
```

```
/// <param name="otherFace">Face to be copied.</param>
public Face(Face otherFace)
{
    foreach (var vertex in otherFace.Vertices)
    {
        var newVertex = new Vertex(vertex);
        Vertices.Add(newVertex);
    }

    GenerateNormal();
    Material = otherFace.Material;
}
#endregion

#region Public Logic
/// <summary>
/// Generates face normal.
/// </summary>
public void GenerateNormal()
{
    if (Vertices.Count >= 3)
    {
        var v1 = new Vector(Vertices[0].Point, Vertices[1].Point);
        var v2 = new Vector(Vertices[0].Point, Vertices[2].Point);
        Normal = v1.CrossProduct(v2);
        Normal.Normalize();
    }
}

/// <summary>
/// Returns whether the face contains given point.
/// </summary>
/// <param name="point">Point.</param>
/// <returns>true if the face contains given point; otherwise, false.</returns>
public bool Contains(Point point)
{
    for (var i = 0; i < Vertices.Count; i++)
    {
        var firstEdgePoint = Vertices[i].Point;
        var secondEdgePoint = Vertices[(i + 1) % Vertices.Count].Point;
        var edgeVector = new Vector(firstEdgePoint, secondEdgePoint);
        var vector = new Vector(firstEdgePoint, point);
        var cross = edgeVector.CrossProduct(vector);
        if (cross.FindAngleBetween(Normal) > 0.00001)
        {
            return false;
        }
    }

    return true;
}

/// <summary>
/// Gets ray intersection point.
/// </summary>
/// <param name="ray">Ray.</param>
/// <returns>Intersection point if found; otherwise, null.</returns>
public Point GetRayIntersectionPoint(Ray ray)
{
    if (ray.Direction.IsPerpendicularTo(Normal))
    {
        return null;
    }
```

```
            var surface = new Surface(Vertices[0].Point, Normal);
            var intersectionPoint = surface.GetLineIntersection(ray.Direction,
ray.StartPoint);
            var directionToIntersectionPoint = new Vector(ray.StartPoint,
intersectionPoint);
            if (directionToIntersectionPoint.FindAngleBetween(ray.Direction) < 0.00001 &&
Contains(intersectionPoint))
            {
                return intersectionPoint;
            }

            return null;
        }
        #endregion
    }
}
```

7. Camera.cs

```
// ***********************************************************************************
// Copyright FileName - Camera.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
// FileType - Visual C# Source File
// ***********************************************************************************
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GeometryQueryLanguage.GeometricalModels;
using SharpGL;

namespace GeometryQueryLanguage
{
    /// <summary>
    /// Represents camera for 3D scene.
    /// </summary>
    public abstract class Camera
    {
        #region Properties

        /// <summary>
        /// Gets the OpenGL instance.
        /// </summary>
        public OpenGL OpenGl { get; }

        /// <summary>
        /// Gets camera position.
        /// </summary>
        public Point Position { get; private set; }

        /// <summary>
        /// Gets or sets rotation angle around X axis.
        /// </summary>
        public double RotationAngleX { get; set; }

        /// <summary>
        /// Gets or sets rotation angle around Y axis.
        /// </summary>
        public double RotationAngleY { get; set; }

        /// <summary>
```

```
        /// Gets or sets near clipping plane.
        /// </summary>
        public double Near { get; set; } = 0.5;

        /// <summary>
        /// Gets or sets far clipping plane.
        /// </summary>
        public double Far { get; set; } = 250;

        /// <summary>
        /// Gets or sets aspect ratio.
        /// </summary>
        public virtual double AspectRatio { get; set; }

        /// <summary>
        /// Gets center of coordinates.
        /// </summary>
        protected Point CenterOfCoordinates { get; } = new Point(0, 0, 0);

        /// <summary>
        /// Gets or sets scale to fit.
        /// </summary>
        protected float Scale { get; set; } = 1;
        #endregion

        #region Constructors

        /// <summary>
        /// Initializes a new instance of the <see cref="Camera"/> class.
        /// </summary>
        /// <param name="openGl">The OpenGL object.</param>
        protected Camera(OpenGL openGl)
        {
            OpenGl = openGl;
            AspectRatio = (double)OpenGl.RenderContextProvider.Width /
OpenGl.RenderContextProvider.Height;
        }
        #endregion

        #region Public Logic

        /// <summary>
        /// Applies camera projection.
        /// </summary>
        public void Project()
        {
            OpenGl.MatrixMode(OpenGL.GL_PROJECTION);
            OpenGl.LoadIdentity();
            SetProjection();
            OpenGl.MatrixMode(OpenGL.GL_MODELVIEW);
        }

        /// <summary>
        /// Sets projection.
        /// </summary>
        public abstract void SetProjection();

        /// <summary>
        /// Sets up camera.
        /// </summary>
        /// <param name="initialCameraPosition">Initial camera position.</param>
        /// <param name="up">Up vector.</param>
        public void SetUp(Point initialCameraPosition, Vector up)
        {
```

```
            Position = new Point(initialCameraPosition);
            var cameraDirection = new Vector(CenterOfCoordinates, Position);

            cameraFront = new Vector(
                                -Position.X,
                                -Position.Y,
                                -Position.Z);
            cameraFront.Normalize();
            cameraRight = up.CrossProduct(cameraFront);
            cameraRight.Normalize();
            cameraUp = cameraFront.CrossProduct(cameraRight);
            cameraUp.Normalize();

            translation = new Vector();

            var directionProjectionXZ = new Vector(cameraDirection.X, 0,
cameraDirection.Z);
            var directionProjectionXY = new Vector(cameraDirection.X, cameraDirection.Y,
0);

            RotationAngleX = new Vector(1, 0, 0).FindAngleBetween(directionProjectionXY);
            RotationAngleY = new Vector(1, 0, 0).FindAngleBetween(directionProjectionXZ);

            if (cameraDirection.Y < 0)
            {
                RotationAngleX += Math.PI;
            }

            if (cameraDirection.X < 0 || cameraDirection.Z < 0)
            {
                RotationAngleY += Math.PI;
            }
        }

        /// <summary>
        /// Transforms the projection matrix so that it looks forward from camera
position.
        /// </summary>
        public void Look()
        {
            OpenGl.LookAt(
                        Position.X,
                        Position.Y,
                        Position.Z,
                        Position.X + cameraFront.X,
                        Position.Y + cameraFront.Y,
                        Position.Z + cameraFront.Z,
                        cameraUp.X,
                        cameraUp.Y,
                        cameraUp.Z);
        }

        /// <summary>
        /// Makes zoom in if zoomSpeed is greater than 0, zoom out - if zoomSpeed is less
than 0.
        /// </summary>
        /// <param name="zoomSpeed">Zoom speed.</param>
        public virtual void Zoom(float zoomSpeed)
        {
            var zSpeed = zoomSpeed * new Vector(Position, CenterOfCoordinates).Length;
            Position.X = Position.X + (zSpeed * cameraFront.X);
            Position.Y = Position.Y + (zSpeed * cameraFront.Y);
            Position.Z = Position.Z + (zSpeed * cameraFront.Z);
        }
```

```
        /// <summary>
        /// Pans view.
        /// </summary>
        /// <param name="xTranslation">Translation along x axis.</param>
        /// <param name="yTranslation">Translation along y axis.</param>
        public virtual void Pan(float xTranslation, float yTranslation)
        {
            Position.X = Position.X + (cameraRight.X * xTranslation) + (cameraUp.X *
yTranslation);
            Position.Y = Position.Y + (cameraRight.Y * xTranslation) + (cameraUp.Y *
yTranslation);
            Position.Z = Position.Z + (cameraRight.Z * xTranslation) + (cameraUp.Z *
yTranslation);

            translation.X = translation.X - ((cameraRight.X * xTranslation) + (cameraUp.X
* yTranslation));
            translation.Y = translation.Y - ((cameraRight.Y * xTranslation) + (cameraUp.Y
* yTranslation));
            translation.Z = translation.Z - ((cameraRight.Z * xTranslation) + (cameraUp.Z
* yTranslation));
        }

        /// <summary>
        /// Rotates camera for specified angles changes.
        /// </summary>
        /// <param name="xAngleDelta">X angle rotation change.</param>
        /// <param name="yAngleDelta">Y angle rotation change.</param>
        public void Rotate(float xAngleDelta, float yAngleDelta)
        {
            RotationAngleX += xAngleDelta;
            RotationAngleY += yAngleDelta;

            Rotate();
        }

        /// <summary>
        /// Rotates camera for <see cref="RotationAngleX"/> and <see
cref="RotationAngleY"/>.
        /// </summary>
        public void Rotate()
        {
            var distanceToCenter = new Vector(CenterOfCoordinates, Position).Length;

            Position.X = (float)(Math.Cos(RotationAngleX) * Math.Cos(RotationAngleY)) *
distanceToCenter;
            Position.Y = (float)Math.Sin(RotationAngleX) * distanceToCenter;
            Position.Z = (float)(Math.Cos(RotationAngleX) * Math.Sin(RotationAngleY)) *
distanceToCenter;

            cameraFront = new Vector(Position, new Point(-translation.X, -translation.Y,
-translation.Z));
            cameraFront.Normalize();

            cameraRight = cameraUp.CrossProduct(cameraFront);
            cameraRight.Y = 0;
            cameraRight.Normalize();

            cameraUp = cameraFront.CrossProduct(cameraRight);
            cameraUp.Normalize();
        }

        /// <summary>
        /// Calculate zoom to fit parameters.
```

```
        /// </summary>
        /// <param name="model">Model.</param>
        public void CalculateZoomToFit(Model model)
        {
            translationToCenterCoordinates.X = -model.BoundingBox.Center.X;
            translationToCenterCoordinates.Y = -model.BoundingBox.Center.Y;
            translationToCenterCoordinates.Z = -model.BoundingBox.Center.Z;
            CalculateScaleToFit((float)model.BoundingBox.BoundingSphereRadius);
        }

        /// <summary>
        /// Calculate zoom to fit parameters assuming the object is at the origin.
        /// </summary>
        /// <param name="boundingSphereRadius">Bounding sphere radius.</param>
        public virtual void CalculateScaleToFit(float boundingSphereRadius)
        {
            Scale = (float)Math.Min(AspectRatio, 1);
        }

        /// <summary>
        /// Zoom to fit model size.
        /// </summary>
        public void ZoomToFit()
        {
            OpenGl.Scale(Scale, Scale, Scale);
            OpenGl.Translate(translationToCenterCoordinates.X,
translationToCenterCoordinates.Y, translationToCenterCoordinates.Z);
        }
        #endregion

        #region Fields
        private readonly Vector translationToCenterCoordinates = new Vector();
        private Vector cameraFront;
        private Vector cameraUp;
        private Vector cameraRight;
        private Vector translation;
        #endregion
    }
}
```

8. OrthographicCamera.cs

```
// ***********************************************************************************
// Copyright FileName - OrthographicCamera.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
// FileType - Visual C# Source File
// ***********************************************************************************
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GeometryQueryLanguage.GeometricalModels;
using SharpGL;

namespace GeometryQueryLanguage
{
    /// <summary>
    /// Represents camera with orthographic projection for 3D scene.
    /// </summary>
    public class OrthographicCamera : Camera
    {
```

```
        #region Properties

        /// <summary>
        /// Gets or sets left clipping plane.
        /// </summary>
        public double Left { get; set; }

        /// <summary>
        /// Gets or sets right clipping plane.
        /// </summary>
        public double Right { get; set; }

        /// <summary>
        /// Gets or sets top clipping plane.
        /// </summary>
        public double Top { get; set; }

        /// <summary>
        /// Gets or sets bottom clipping plane.
        /// </summary>
        public double Bottom { get; set; }

        /// <summary>
        /// Gets zoom factor.
        /// </summary>
        public double ZoomFactor { get => zoom; private set => zoom = Math.Max(value,
0.01f); }

        /// <summary>
        /// Gets or sets aspect ratio.
        /// </summary>
        public override double AspectRatio
        {
            get => aspectRatio;
            set
            {
                aspectRatio = value;
                Right = Top * aspectRatio;
                Left = -Right;
            }
        }
        #endregion

        #region Constructors

        /// <summary>
        /// Initializes a new instance of the <see cref="OrthographicCamera"/> class.
        /// </summary>
        /// <param name="openGl">The OpenGL object.</param>
        public OrthographicCamera(OpenGL openGl)
            : base(openGl)
        {
            Top = 1;
            Bottom = -Top;
            Right = Top * AspectRatio;
            Left = -Right;
            Near = -Far;
        }
        #endregion

        #region Public Logic

        /// <summary>
        /// Sets orthographic projection.
```

```
        /// </summary>
        public override void SetProjection()
        {
            OpenGl.Ortho(Left * ZoomFactor, Right * ZoomFactor, Bottom * ZoomFactor, Top
* ZoomFactor, Near, Far * ZoomFactor);
        }

        /// <summary>
        /// Calculate scale to fit parameter.
        /// </summary>
        /// <param name="boundingSphereRadius">Bounding sphere radius.</param>
        public override void CalculateScaleToFit(float boundingSphereRadius)
        {
            base.CalculateScaleToFit(boundingSphereRadius);
            ZoomFactor = 1;
            Scale *= 1 / boundingSphereRadius;
        }

        /// <summary>
        /// Makes zoom in if zoomSpeed is greater than 0, zoom out - if zoomSpeed is less
than 0.
        /// </summary>
        /// <param name="zoomSpeed">Zoom speed.</param>
        public override void Zoom(float zoomSpeed)
        {
            ZoomFactor = ZoomFactor - (zoomSpeed * new Vector(Position,
CenterOfCoordinates).Length);
            base.Zoom(zoomSpeed);
        }

        /// <summary>
        /// Pans view.
        /// </summary>
        /// <param name="xTranslation">Translation along x axis.</param>
        /// <param name="yTranslation">Translation along y axis.</param>
        public override void Pan(float xTranslation, float yTranslation)
        {
            base.Pan(
                (float)(xTranslation * 2 * Right * ZoomFactor),
                (float)(yTranslation * 2 * Top * ZoomFactor));
        }
        #endregion

        #region Fields
        private double zoom = 1;
        private double aspectRatio;
        #endregion
    }
}
```

9. PerspectiveCamera.cs

```
// ***********************************************************************
// Copyright FileName - PerspectiveCamera.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
// FileType - Visual C# Source File
// ***********************************************************************
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using GeometryQueryLanguage.GeometricalModels;
using SharpGL;

namespace GeometryQueryLanguage
{
    /// <summary>
    /// Represents camera with perspective projection for 3D scene.
    /// </summary>
    public class PerspectiveCamera : Camera
    {
        #region Properties

        /// <summary>
        /// Gets angle of view in degrees.
        /// </summary>
        public float ViewAngle { get; }
        #endregion

        #region Constructors

        /// <summary>
        /// Initializes a new instance of the <see cref="PerspectiveCamera"/> class.
        /// </summary>
        /// <param name="openGl">The OpenGL object.</param>
        /// <param name="viewAngle">View angle.</param>
        public PerspectiveCamera(OpenGL openGl, float viewAngle)
            : base(openGl)
        {
            ViewAngle = viewAngle;
        }
        #endregion

        #region Public Logic

        /// <summary>
        /// Sets perspective projection.
        /// </summary>
        public override void SetProjection()
        {
            OpenGl.Perspective(ViewAngle, AspectRatio, Near, Far);
        }

        /// <summary>
        /// Calculate scale to fit parameter.
        /// </summary>
        /// <param name="boundingSphereRadius">Bounding sphere radius.</param>
        public override void CalculateScaleToFit(float boundingSphereRadius)
        {
            base.CalculateScaleToFit(boundingSphereRadius);
            var centerToCameraDistance = new Vector(Position,
CenterOfCoordinates).Length;
            var halfViewAngleInDegrees = ViewAngle * Math.PI / 360;
            Scale *= (float)Math.Abs(centerToCameraDistance *
Math.Cos(halfViewAngleInDegrees)
                                     * Math.Tan(halfViewAngleInDegrees)
                                     / boundingSphereRadius);
        }

        /// <summary>
        /// Pans view.
        /// </summary>
        /// <param name="xTranslation">Translation along x axis.</param>
        /// <param name="yTranslation">Translation along y axis.</param>
        public override void Pan(float xTranslation, float yTranslation)
```

```
        {
            var centerToCameraDistance = new Vector(Position,
CenterOfCoordinates).Length;
            base.Pan(
                (float)(xTranslation * centerToCameraDistance * Math.Tan(ViewAngle *
Math.PI / 180) * AspectRatio),
                (float)(yTranslation * centerToCameraDistance * Math.Tan(ViewAngle *
Math.PI / 180)));
        }
        #endregion
    }
}
```

10.Texture.cs

```
// ****************************************************************************
// Copyright FileName - Texture.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
// FileType - Visual C# Source File
// ****************************************************************************
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using SharpGL;

namespace GeometryQueryLanguage.Materials
{
    /// <summary>
    /// Represents OpenGL texture.
    /// </summary>
    public class Texture
    {
        #region Properties
        /// <summary>
        /// Gets the id of the texture.
        /// </summary>
        public uint TextureId { get; private set; }

        /// <summary>
        /// Gets texture bitmap.
        /// </summary>
        public Bitmap Bitmap { get; }

        /// <summary>
        /// Gets texture name.
        /// </summary>
        public string Filename { get; }
        #endregion

        #region Constructors
        /// <summary>
        /// Initializes a new instance of the <see cref="Texture"/> class.
        /// </summary>
        /// <param name="path">The path to the texture image file.</param>
        public Texture(string path)
        {
            var loadedBitmap = new Bitmap(path);
```

```
            Bitmap = new Bitmap(loadedBitmap);
            loadedBitmap.Dispose();
            Filename = Path.GetFileName(path);
        }
        #endregion

        #region Public Logic
        /// <summary>
        /// Bind texture to the specified OpenGL instance.
        /// </summary>
        /// <param name="openGl">The OpenGL instance.</param>
        public void Bind(OpenGL openGl)
        {
            openGl.BindTexture(OpenGL.GL_TEXTURE_2D, TextureId);
        }

        /// <summary>
        /// Creates OpenGL texture.
        /// </summary>
        /// <param name="openGl">The OpenGL instance.</param>
        public virtual void Create(OpenGL openGl)
        {
            if (IsCreated())
            {
                return;
            }

            var textureArray = new uint[1];
            openGl.GenTextures(1, textureArray);
            TextureId = textureArray[0];

            Bind(openGl);

            Bitmap.RotateFlip(RotateFlipType.RotateNoneFlipY);
            BitmapData bitmapData = Bitmap.LockBits(
                                        new Rectangle(0, 0, Bitmap.Width, Bitmap.Height),
                                        ImageLockMode.ReadOnly,
                                        PixelFormat.Format32bppArgb);

            openGl.TexImage2D(
                OpenGL.GL_TEXTURE_2D,
                0,
                OpenGL.GL_RGBA,
                Bitmap.Width,
                Bitmap.Height,
                0,
                OpenGL.GL_BGRA,
                OpenGL.GL_UNSIGNED_BYTE,
                bitmapData.Scan0);

            Bitmap.UnlockBits(bitmapData);

            openGl.TexParameter(OpenGL.GL_TEXTURE_2D, OpenGL.GL_TEXTURE_MIN_FILTER,
OpenGL.GL_LINEAR);
            openGl.TexParameter(OpenGL.GL_TEXTURE_2D, OpenGL.GL_TEXTURE_MAG_FILTER,
OpenGL.GL_LINEAR);

            Bitmap.RotateFlip(RotateFlipType.RotateNoneFlipY);
        }

        /// <summary>
        /// Destroys texture.
        /// </summary>
        /// <param name="openGl">The OpenGL instance.</param>
```

```
        public virtual void Destroy(OpenGL openGl)
        {
            if (IsCreated())
            {
                openGl.DeleteTextures(1, new uint[1] { TextureId });
                TextureId = 0;
                // Bitmap.Dispose();
            }
        }

        /// <summary>
        /// Saves texture image to specified directory.
        /// </summary>
        /// <param name="directoryPath">Directory path.</param>
        public void Save(string directoryPath)
        {
            Bitmap.Save(Path.Combine(directoryPath, Filename));
        }
        #endregion

        #region Private Logic
        private bool IsCreated()
        {
            return TextureId != 0;
        }
        #endregion
    }
}
```

11.MeshTransformation.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using GeometryQueryLanguage.Exceptions;
using GeometryQueryLanguage.GeometricalModels;
using GeometryQueryLanguage.Materials;

namespace GeometryQueryLanguage.ComputationalModule
{
    /// <summary>
    /// Provides ability to transform meshes.
    /// </summary>
    public class MeshTransformation
    {
        #region Properties
        /// <summary>
        /// Gets or sets meshes.
        /// </summary>
        public List<Mesh> Meshes { get; set; }
        #endregion

        #region Constructors
        /// <summary>
        /// Initializes a new instance of the <see cref="MeshTransformation"/> class.
        /// </summary>
        /// <param name="meshes">List of meshes.</param>
        public MeshTransformation(List<Mesh> meshes)
        {
            Meshes = meshes;
```

```
        }
        #endregion

/// <summary>
        /// Cuts mesh with a surface.
        /// </summary>
        /// <param name="surface">Surface.</param>
        /// <returns>List of meshes after cut.</returns>
        public List<Mesh> CutMeshWithSurface(Surface surface)
        {
            var resultMeshList = new List<Mesh>();
            foreach (var mesh in Meshes)
            {
                var meshAbovePlane = new Mesh();
                var meshUnderPlane = new Mesh();

                foreach (var face in mesh.Faces)
                {
                    CutFace(face, surface, meshAbovePlane, meshUnderPlane);
                }

                resultMeshList.AddRange(GetSeparateMeshes(meshAbovePlane));
                resultMeshList.AddRange(GetSeparateMeshes(meshUnderPlane));
            }

            return resultMeshList;
        }
/// <summary>
        /// Get holes edges in meshes.
        /// </summary>
        /// <returns>List of holes edges.</returns>
        public List<Edge> GetHolesEdges()
        {
            var holesEdges = new List<Edge>();
            foreach (var mesh in Meshes)
            {
                var edgesCountDictionary = new Dictionary<Edge, int>();

                foreach (var face in mesh.Faces)
                {
                    for (var i = 0; i < face.Vertices.Count; i++)
                    {
                        var thisVertex = face.Vertices[i];
                        var nextVertex = face.Vertices[(i + 1) % face.Vertices.Count];
                        var edge = new Edge(thisVertex, nextVertex);
                        if (edgesCountDictionary.ContainsKey(edge))
                        {
                            edgesCountDictionary[edge]++;
                        }
                        else
                        {
                            edgesCountDictionary.Add(edge, 1);
                        }
                    }
                }

                holesEdges.AddRange(edgesCountDictionary
                                    .Where(pair => pair.Value == 1)
                                    .Select(pair => pair.Key));
            }

            return holesEdges;
        }
```

```
/// <summary>
        /// Builds convex hull from mesh.
        /// </summary>
        /// <returns>Convex hull.</returns>
        public List<Mesh> BuildConvexHull()
        {
            var convexHulls = new List<Mesh>();
            foreach (var mesh in Meshes)
            {
                var convexHull = new Mesh();
                var allPoints = mesh
                    .Faces
                    .SelectMany(face => face.Vertices)
                    .Select(vertex => vertex.Point)
                    .Distinct();

                var initialFace = FindFirstConvexHullFace(allPoints);
                var unconnectedFaces = new List<Face>();
                var usedEdgesCounts = new Dictionary<Edge, int>();

                unconnectedFaces.Add(initialFace);
                convexHull.Faces.Add(initialFace);
                AddUsedEdges(initialFace, usedEdgesCounts);

                while (unconnectedFaces.Any())
                {
                    var face = unconnectedFaces.Last();
                    unconnectedFaces.Remove(face);
                    for (var i = 0; i < face.Vertices.Count; i++)
                    {
                        var thisVertex = face.Vertices[i];
                        var nextVertex = face.Vertices[(i + 1) % face.Vertices.Count];
                        var edge = new Edge(nextVertex, thisVertex);

                        AddConvexHullFace(convexHull, allPoints, face.Normal, edge,
unconnectedFaces, usedEdgesCounts);
                    }

                }

                convexHull.BoundingBox = mesh.BoundingBox;
                convexHulls.Add(convexHull);
            }

            return convexHulls;
        }
private static Dictionary<int, List<int>> GetFaceGraph(IList<Face> faces)
        {
            var connectedFacesGraph = new Dictionary<int, List<int>>();
            foreach (var connectedFaces in GetFaceGraphWithConnectionsCount(faces))
            {
                connectedFacesGraph[connectedFaces.Key] = connectedFaces.Value
                                                          .Where(pair => pair.Value > 1)
                                                          .Select(pair => pair.Key)
                                                          .ToList();
            }

            return connectedFacesGraph;
        }

        private static Dictionary<Point, List<int>> GetFacesConnectedByPoint(IList<Face>
faces)
        {
            var pointToConnectedFaces = new Dictionary<Point, List<int>>();
```

```
            for (var i = 0; i < faces.Count; i++)
            {
                foreach (var point in faces[i].Vertices.Select(v => v.Point))
                {
                    if (!pointToConnectedFaces.ContainsKey(point))
                    {
                        pointToConnectedFaces.Add(point, new List<int>());
                    }

                    pointToConnectedFaces[point].Add(i);
                }
            }

            return pointToConnectedFaces;
        }

        private static Dictionary<int, Dictionary<int, int>>
GetFaceGraphWithConnectionsCount(IList<Face> faces)
        {
            var pointToConnectedFaces = GetFacesConnectedByPoint(faces);

            var faceGraphWithConnectionsCount = new Dictionary<int, Dictionary<int,
int>>();
            foreach (var faceList in pointToConnectedFaces.Values)
            {
                foreach (var faceIndex in faceList)
                {
                    if (!faceGraphWithConnectionsCount.ContainsKey(faceIndex))
                    {
                        faceGraphWithConnectionsCount.Add(faceIndex, new Dictionary<int,
int>());
                    }

                    foreach (var otherFaceIndex in faceList.Where(index => index !=
faceIndex))
                    {

faceGraphWithConnectionsCount[faceIndex].TryGetValue(otherFaceIndex, out int
currentCount);
                        faceGraphWithConnectionsCount[faceIndex][otherFaceIndex] =
currentCount + 1;
                    }
                }
            }

            return faceGraphWithConnectionsCount;
        }

        private static List<List<Face>> GetConnectedFacesGroups(List<Face> allFaces)
        {
            var connectedFacesGraph = GetFaceGraph(allFaces);
            var resultFacesGroups = new List<List<Face>>();
            var visitedFaces = new bool[allFaces.Count];

            for (var i = 0; i < allFaces.Count; i++)
            {
                if (!visitedFaces[i])
                {
                    resultFacesGroups.Add(new List<Face>());
                    VisitFace(i, visitedFaces, allFaces, resultFacesGroups.Last(),
connectedFacesGraph);
                }
            }
```

```
            return resultFacesGroups;
        }

        private static void VisitFace(int faceIndex, bool[] visitedFaces, IList<Face>
allFaces, IList<Face> facesGroup, Dictionary<int, List<int>> graph)
        {
            visitedFaces[faceIndex] = true;
            facesGroup.Add(allFaces[faceIndex]);
            foreach (var connectedFaceIndex in graph[faceIndex])
            {
                if (!visitedFaces[connectedFaceIndex])
                {
                    VisitFace(connectedFaceIndex, visitedFaces, allFaces, facesGroup,
graph);
                }
            }
        }


        private void CutFace(Face face, Surface surface, Mesh meshAbovePlane, Mesh
meshUnderPlane)
        {
            var faceAbovePlane = new Face();
            var faceUnderPlane = new Face();
            for (var i = 0; i < face.Vertices.Count; i++)
            {
                var thisVertex = face.Vertices[i];
                if (surface.Contains(thisVertex.Point))
                {
                    faceAbovePlane.Vertices.Add(new Vertex(thisVertex));
                    faceUnderPlane.Vertices.Add(new Vertex(thisVertex));
                    continue;
                }
                else if (surface.IsPointAbove(thisVertex.Point))
                {
                    faceAbovePlane.Vertices.Add(thisVertex);
                }
                else
                {
                    faceUnderPlane.Vertices.Add(thisVertex);
                }

                var nextVertex = face.Vertices[(i + 1) % face.Vertices.Count];
                if (surface.IsIntersectedByEdge(thisVertex.Point, nextVertex.Point))
                {
                    var newPoint = surface.GetEdgeIntersection(thisVertex.Point,
nextVertex.Point);
                    Vector newNormal = null;
                    var textureCoordinate =
CalculateTextureCoordinateAfterCut(thisVertex, nextVertex, newPoint);

                    if (thisVertex.Normal != null && nextVertex.Normal != null)
                    {
                        newNormal = thisVertex.Normal + nextVertex.Normal;
                        newNormal.Normalize();
                    }

                    faceAbovePlane.Vertices.Add(new Vertex
                    {
                        Point = newPoint,
                        TextureCoordinates = textureCoordinate,
                        Normal = newNormal,
                    });
                    faceUnderPlane.Vertices.Add(new Vertex
```

```
                    {
                        Point = newPoint,
                        TextureCoordinates = textureCoordinate,
                        Normal = newNormal,
                    });
                }
            }

            if (faceAbovePlane.Vertices.Count >= minCountOfVerticesInFace)
            {
                faceAbovePlane.GenerateNormal();
                faceAbovePlane.Material = face.Material;
                meshAbovePlane.Faces.Add(faceAbovePlane);
            }

            if (faceUnderPlane.Vertices.Count >= minCountOfVerticesInFace)
            {
                faceUnderPlane.GenerateNormal();
                faceUnderPlane.Material = face.Material;
                meshUnderPlane.Faces.Add(faceUnderPlane);
            }
               }


private List<Point> BuildFlatConvexHull(IEnumerable<Point> points, Vector normal)
        {
            var sortedPoints = points.OrderBy(point => point.X)
                                     .ThenBy(point => point.Y)
                                     .ThenBy(point => point.Z);

            var upperHull = new List<Point>();
            foreach (var point in sortedPoints)
            {
                AddPointToHull(point, upperHull, normal);
            }

            upperHull.RemoveAt(upperHull.Count - 1);

            var lowerHull = new List<Point>();
            foreach (var point in sortedPoints.Reverse())
            {
                AddPointToHull(point, lowerHull, normal);
            }

            lowerHull.RemoveAt(lowerHull.Count - 1);

            if (upperHull.Count != 1 || !Enumerable.SequenceEqual(upperHull, lowerHull))
            {
                upperHull.AddRange(lowerHull);
            }

            return upperHull;
               }
       }
}
```

1. Program.cs

```
// ***********************************************************************
// Copyright FileName - Program.cs
// Copyright(c) AMC Bridge LLC. All rights reserved.
```

```
// FileType - Visual C# Source File
// ****************************************************************************
using System;
using System.IO;
using System.Linq;
using System.Reflection;

using WixSharp;

namespace Setup
{
    /// <summary>
    /// Creates installer for GeometryQueryLanguage project.
    /// </summary>
    public static class Program
    {
        /// <summary>
        /// Represents script for creating installer.
        /// </summary>
        public static void Main()
        {
            var solutionDir = Path.GetFullPath(Path.Combine(Environment.CurrentDirectory,
@"..\"));

            Compiler.WixLocation = Path.GetFullPath(Path.Combine(solutionDir,
@"packages\WixSharp.wix.bin.3.11.0\tools\bin"));

            var assembly = typeof(GeometryQueryLanguage.MainViewModel).Assembly;

            var productName =
GetAssemblyAttribute<AssemblyProductAttribute>(assembly).Product;
            var companyName =
GetAssemblyAttribute<AssemblyCompanyAttribute>(assembly).Company;
            var guid =
GetAssemblyAttribute<System.Runtime.InteropServices.GuidAttribute>(assembly).Value;
            var versionString = Environment.GetEnvironmentVariable("CURRENT_VERSION");
            var version = (versionString != null) ? new Version(versionString) :
assembly.GetName().Version;

            var executableFileName = assembly.ManifestModule.Name;

            var productNameWithVersion = $"{productName}_{version}";
            var uninstallTitle = $"Uninstall {productName}";
            var uninstallShortcutTarget = "[System64Folder]msiexec.exe";
            var uninstallShortcutTargetArguments = "/x [ProductCode]";

            var project = new Project(
                                productName,
                                new Dir(
                                    productNameWithVersion,
                                    new Files("*.*", f => !f.EndsWith(".pdb")
                                                       && !f.EndsWith(".xml")),
                                    new ExeFileShortcut(uninstallTitle,
uninstallShortcutTarget, uninstallShortcutTargetArguments)),
                                new Dir(Path.Combine(@"%ProgramFiles%", companyName)),
                                new Dir(
                                    Path.Combine(@"%ProgramMenu%", productNameWithVersion),
                                    new ExeFileShortcut(uninstallTitle,
uninstallShortcutTarget, uninstallShortcutTargetArguments),
                                    new ExeFileShortcut(productName,
$"[INSTALLDIR]{executableFileName}", arguments: string.Empty)),
                                new Dir(
                                    @"%Desktop%",
```

```
                                new ExeFileShortcut(productName,
$"[INSTALLDIR]{executableFileName}", arguments: string.Empty)));

            project.Actions = new[]
            {
                new SetPropertyAction
                {
                    PropName = "INSTALLDIR",
                    Value = Path.Combine(@"[ProgramFilesFolder]", companyName,
productNameWithVersion),
                    Sequence = Sequence.InstallUISequence,
                    When = When.Before,
                    Step = Step.CostFinalize,
                },
            };

            project.GUID = new Guid(guid);
            project.OutFileName = productNameWithVersion;
            project.OutDir = Path.Combine(solutionDir, "Installer");
            project.LicenceFile = Path.Combine(Environment.CurrentDirectory,
"License.rtf");
            project.UI = WUI.WixUI_InstallDir;
            project.SourceBaseDir = Path.GetFullPath(Path.Combine(solutionDir,
@"GQL\bin\Release"));
            project.Version = version;
            project.ControlPanelInfo.Manufacturer = companyName;
            AutoElements.SupportEmptyDirectories = CompilerSupportState.Disabled;
            project.BuildMsi();
        }

        private static T GetAssemblyAttribute<T>(System.Reflection.Assembly assembly)
            where T : Attribute
        {
            return assembly
                    .GetCustomAttributes(typeof(T), false)
                    .OfType<T>()
                    .First();
        }
    }
}
```