МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ НАУК
КАФЕДРА КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Дипломна магістерська робота

на тему

«Програмний засіб для укладки об'єктів у тривимірному просторі»

Виконав: студент групи МгІТ-21
спеціальності 122 Комп'ютерні науки

Студент Денис ОЛІЙНИК

Керівник Володимир ЯХНО

Рецензент

Київ 2022

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

Факультет _Мехатроніки та комп'ютерних наук_

Кафедра Комп'ютерних наук та технологій

Спеціальність 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ

Завідувач кафедри

д.т.н., професор Щербань В.Ю.

" " 2022 року

З А В Д А Н Н Я
НА ДИПЛОМНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ
Олійнику Денису Тімуровичу

1. Тема роботи: Програмний засіб для укладки об'єктів у тривимірному просторі

Науковий керівник роботи к.т.н., доцент Яхно Володимир Михайлович

Затверджені наказом вищого навчального закладу від " " 2022 року №

3. Строк подання студентом роботи

4. Вихідні дані до роботи

5. Зміст дипломної роботи: Розділ 1(Аналіз існуючих методів та засобів взаємодії між об'єктами тривимірного простору); Розділ 2(Алгоритмічне забезпечення програмного засобу); Розділ 3 (Програмне забезпечення засобу).

5. Перелік графічного матеріалу: Презентація, що є додатком до доповіді.

6. Консультанти розділів дипломної магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Вступ	Володимир ЯХНО к.т.н., доц.		
Розділ 1	Володимир ЯХНО к.т.н., доц.		
Розділ 2	Володимир ЯХНО к.т.н., доц.		
Розділ 3	Володимир ЯХНО к.т.н., доц.		
Висновки	Володимир ЯХНО к.т.н., доц.		

7. Дата видачі завдання

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	1.10.2022р.	
2	Розділ 1 (Аналіз існуючих методів та засобів взаємодії між об’єктами тривимірного простору)	10.10.2022р.	
3	Розділ 2 (Алгоритмічне забезпечення програмного засобу)	15.10.2022р.	
4	Розділ 3 (Програмне забезпечення засобу)	20.10.2022р.	
5	Висновки	25.10.2022р.	
6	Оформлення дипломної магістерської роботи (чистовий варіант)	30.10.2022р.	
7	Здача дипломної магістерської роботи на кафедру для рецензування (за 14 днів до захисту)	01.11.2022р.	
8	Перевірка дипломної магістерської роботи на наявність ознак плагіату (за 10 днів до захисту)	04.11.2022р.	
9	Подання дипломної магістерської роботи на затвердження завідувачу кафедри (з 7 днів до захисту)	07.11.2022р	

Студент _____ Денис ОЛІЙНИК

Науковий керівник роботи _____ Володимир ЯХНО

Керівник відділу магістратури _____ Олена ГРИГОРЕВСЬКА

Анотація

Олійник Д.Т. Програмний засіб для укладки об’єктів у тривимірному просторі.

Дипломна магістерська робота за спеціальністю 122- «Комп’ютерні науки та технології» – Київський національний університет технологій та дизайну, Київ, 2022 рік.

Досліджено методи моделювання положення об’єктів у просторі.

Досліджені алгоритм Робертса для відкидання прихованих поверхонь опуклих об’єктів тривимірного простору, а також досліджено алгоритм Z-буфера для схожої задачі.

Описано структуру взаємопов’язаних класів, яка є необхідною для створення зрозумілих для обчислювальної машини об’єктів у тривимірному просторі.

Описано запропонований алгоритм для візуалізації, який є модифікацією поєднання алгоритмів Робертса та Z-буферу.

Описано проблему розміщення об’єкту у просторі та запропоновано алгоритм розміщення об’єкта у просторі сцени, на основі його хіт-боксу.

Створено систему взаємопов'язаних класів, яка формує об'єкт на базі якої формуються об'єкти тривимірного простору.

Створено програмний засіб на базі розробленого алгоритму для пошуку позиції та щільного розміщення об'єктів тривимірного простору за їх хіт-боксами. Для цього використовувалась платформа .NET та її елемент Win Forms, а мовою програмування була обрана C#.

Наведено опис функціоналу та інтерфейсу створеного програмного засобу, а також детально розібраний процес його експлуатації.

Ключові слова: тривимірний простір, простір, клас, переміщення, візуалізація.

Annotation

Oilynyk D.T. Software tool for stacking objects in three-dimensional space.

Master's thesis in specialty 122- "Computer science and technology" - Kyiv National University of Technology and Design, Kyiv, 2022.

Methods of modeling the position of objects in space have been studied.

The Roberts algorithm for discarding hidden surfaces of convex objects in three-dimensional space was studied, as well as the Z-buffer algorithm for a similar problem.

The structure of interrelated classes is described, which is necessary for the creation of computer-understandable objects in three-dimensional space.

The proposed algorithm for visualization is described, which is a modification of the combination of the Roberts and Z-buffer algorithms.

The problem of placing an object in space is described, and an algorithm for placing an object in the space of a scene, based on its hit box, is proposed.

A system of interconnected classes has been created, which forms an object on the basis of which three-dimensional space objects are formed.

A software tool was created based on the developed algorithm for finding the position and dense placement of objects in three-dimensional space by their hit boxes. For this, the .NET platform and its Win Forms element were used, and C# was chosen as the programming language.

A description of the functionality and interface of the created software tool is given, as well as the detailed process of its operation.

Key words: three-dimensional space, space, class, displacement, visualization.

Зміст

Вступ

Актуальність теми: Розміщення об’єктів у тривимірному просторі є досить поширеною задачею як у середовищах для моделювання, так і у іграх і навіть будівництві та плануванні. Проте не всюди алгоритми для виконання цієї задачі виконують свою роботу досконало, через що дуже часто результат потребує корегування з боку користувача.

Мета і завдання виконання дипломної роботи: Дослідження та аналіз методів та засобів комп’ютерної графіки для відтворення тривимірного простору на двовимірній площині. Аналіз взаємодії між хіт-боксами об’єктів у просторі.

Завдання: Провести аналіз методів переміщення об’єктів у просторі та обрати найкращий у рамках виконання задачі. Провести аналіз методів відсічення прихованих частин тривимірних об’єктів та модифікувати обраний. На основі досліджених матеріалів розробити алгоритм, що дозволить чітко, щільно та ефективно розташовувати об’єкти у заданій в просторі точці на базі їх хіт-боксів.

Об’єкт: Чіткість розміщення хіт-боксу об’єкту відносно інших об’єктів у тривимірному просторі.

Предмет: Недосконалість існуючих алгоритмів, що відбуваються під час щільного розміщення об’єктів.

Методи:. Створення та пропонування алгоритму щільного розміщення об’єктів. Модифікація алгоритму Робертса для візуалізації роботи запропонованого алгоритму.

Практичне значення одержаних результатів: Одержані результати надають наглядне розуміння запропонованого алгоритму та його ефективності

Розділ 1. Аналіз існуючих методів та засобів взаємодії між об’єктами тривимірного простору

1.1. 3D-Рендеринг

3D-рендеринг являє собою механізм, що широко використовується в різноманітних, серйозних сферах бізнесу. Наприклад, таких, як маркетинг, будівництво, корпоративний брендинг і реклама. Щоб повністю зрозуміти, що таке тривимірний рендеринг і як саме він використовується, необхідно спочатку визначити чим являється більш загальна концепція 3D-візуалізації.

3D-візуалізація, це поєднання широкого спектру можливостей і навичок, що використовуються, для демонстрації результатів або ж продукту. Основним, що включає в себе ця мультидисциплінарна процедура, є створення концепції, моделювання простору, вибір матеріалів, розташування освітлення, рендеринг, доопрацювання та доставка результату.

За допомогою тривимірної візуалізації та конкретних візуальних завдань, ми маємо можливість забезпечити чітке цифрове представлення того, як буде виглядати продукт візуалізації у реальному житті.

Отже тривимірний рендеринг, це один з останніх етапів тривимірної візуалізації. Це процес, що полягає у створенні докладних і реалістичних зображень цифрової, тривимірної моделі. Наприклад, дизайнер може створити максимально реалістичне зображення, використовуючи найсучасніші програмні засоби для виконання завдання.

3D-моделювання, це процес розробки математичного уявлення про будь-який об’єкт тривимірного простору за допомогою спеціально для цього призначеного програмного забезпечення. Це незамінний етап для візуалізації тривимірних об’єктів у яких є маленькі компоненти(деталі об’єкту).

Існує два основні види тривимірного моделювання – полігональне та параметричне. При використані полігонального виду за основу береться полігон – площина, що складається з набору вершин, які з’єднані між собою лініями, що в свою чергу утворюють ребра. Сукупність подібних полігонів називають сіткою, яку можна редагувати у будь-якій обраній точці та у будь-

який передбачений для цієї точки спосіб. Моделі що створені подібним чином можна поділити за рівнем якості. Чим більше полігонів, тобто площин має створений об’єкт, тим більшу якість у теорії він має і навпаки, чим менше полігонів він має, тим більш незграбнішим буде виглядати. Цей метод тривимірного моделювання є найбільш поширеним. Його використовують для скульптингу, тривимірного друку, спецефектів у кінематографі, створенні та проектуванні ігор, а також у програмах доповненої та віртуальної реальностей. При використанні параметричного виду моделювання використовується якась обрана математична модель, що називають ескізом і в ній задаються параметри. У такому випадку при зміні параметрів для математичної моделі, тривимірна модель буде приймати різний вигляд для кожної комбінації параметрів. Такий вид моделювання являє собою досить простий і вже досить старий спосіб проектування промислових деталей та різноманітних механізмів.

Для створення тривимірних моделей, використовується спеціальне програмне забезпечення, що дозволяє керувати у просторі точками, які належать об’єктам моделювання. Результатом такого моделювання є тривимірний об’єкт, або є сукупність тривимірних об’єктів що складаються є частиною одного проекту.

Перш ніж продукт переходить до наступних етапів, в тому числі і до самого тривимірного рендерингу, зазвичай проводять оцінку розмірів та об’єму даних.

Отже, моделювання надає повну демонстрацію цифрового об’єкту тривимірного простору.

Процес створення результату відображення, тобто візуалізації можна поділити на основні етапи.

Етап 1. Розуміння проекту

Під розумінням проекту мається на увазі точна постановка задачі для подальшого її виконання. На цьому етапі необхідно окреслити конкретні цілі і бажаний фінальний результат.

Етап 2. Тривимірне моделювання

На цьому етапі відбувається створення цифрового макету об’єкта за допомогою обраного розробником програмного засобу і обраного виду тривимірного моделювання.

Етап 3. Вибір найбільш репрезентативних і відповідних матеріалів

На цьому етапі відбувається вибір матеріалу та налаштування вигляду поверхні. Цей етап має безпосередній вплив те як буде виглядати об’єкт, оскільки на ньому задаються багато параметрів поверхні об’єкту.

Етап 4. Вибір освітлення для коректного застосування

На цьому етапі відбувається створення та налаштування джерела, або ж джерел освітлення, якщо їх багато. Цей етап має безпосередній вплив на те, як об’єкт буде сприйматись спостерігачем, оскільки освітлення дає глибину і може або ж підкреслити грані об’єкта, зробивши їх більш помітними, або ж навпаки, приховати їх, зробивши ледь чи зовсім непомітними.

Етап 5. Візуалізація тривимірного об’єкта

Після завершення всіх минулих етапів, настає етап візуалізації. Його суть полягає у тому, щоб створити натуралістичне двовимірне зображення тривимірного об’єкту, або ж сукупності тривимірних об’єктів. Це можна порівняти з фотографування у реальному житті. Швидкість виконання цього процесу дуже залежить від складності зображення і може тривати від менше ніж однієї секунди, до кількох діб. Зазвичай цей процес може бути виконаний лише а допомогою високотехнологічного комп’ютерного апаратного та програмного забезпечень.

Етап 6. Уточнення

На цьому етапі проводиться порівняння отриманого у результаті процесу візуалізації зображення з тим, що планувалось під час постановки задачі. Якщо все чітко і правильно, а результат співпадає з бажаним, то виконується перехід до наступного етапу. У іншому

випадку проводиться робота над помилками і за найгіршого сценарію все повністю перероблюється, починаючи зі створення об’єкту.

Етап 7. Отримання остаточного результату

На цьому етапі отримується остаточний результат і готовий продукт.

1.2. Структура об’єктів тривимірного простору

Будь-яку фігуру або ж поверхню можна представити у вигляді безлічі інших, простіших фігур. Наприклад сфера, у комп'ютерній графіці у будь-якому випаду наразі представляється через трикутники і чотирикутники. Чим їх більше, тим більш гладкішою буде виглядати її поверхня, оскільки це напряму залежить від ступеня наближення, який в свою чергу можна розрахувати за допомогою кількості поверхонь, на яку розбивається базова.

Зазвичай тривимірна модель складається з вершин, граней, полігонів, текстур та карт нормалей.

Вершина – це перша функціональна складова, що представлена у вигляді координат по X,Y і Z. Іншими словами, це точка у просторі.

Грань – це прямий відрізок від однієї вершини до іншої. Іншими словами, це лінія, що задана і обмежена двома точками у просторі.

Полігон – це друга функціональна складова, що представлена у вигляді плоскої багатокутної фігури. Те, що вона є плоскою каже про те, що всі її вершини знаходяться на одній площині. Всі вершини полігону послідовно пов’язані між собою, утворюючи грані. Для кожної вершини може існує лише два сусіди на конкретному полігоні.

Текстура – це зображення, що натягується на всю поверхню тривимірного об’єкта, або ж на один з його полігонів.

Нормаль і карта нормалей – це, зазвичай вектор, що є перпендикулярним до площини, але у цьому випадку, це зображення, кожний піксель якого відповідає за направленість вектору в кожній точці на полігоні. За допомогою карт нормалей, імітується складність поверхні.

1.3. Методи для переміщення об’єктів у просторі

Однією з найосновніших та фундаментальних частин комп'ютерної графіки завжди були геометричні перетворення. Вони здатні змінювати розмір, орієнтацію і форму об'єктів (фігур, ліній, точок). Переміщення та перетворення у двовимірному просторі називається двовимірним, або ж 2D ("D" від англійської "Dimensional", що у перекладі на українську означає "Просторовий") переміщеннями та перетвореннями. За подібним принципом, перетворення у тривимірному просторі називаються тривимірними, або ж 3D переміщеннями та перетворенням. Надалі розглянемо два типи перетворення та переміщень: ті які відбуваються при звертанні до афінних методів і ті, що відбуваються при звертанні до значень на пряму.

1.3.1. Звертаючись до афінних перетворень

Афінні перетворення, являють собою метод лінійного відображення, який дозволяє трансформувати вершини об'єктів у просторах. При їх виконанні цілісність вершин одна відносно одної зберігається. Таким чином, наприклад, якщо прямі були паралельні до трансформації, то і після неї вони збережуть свою паралельність одна відносно одної, але тільки за тієї умови, якщо над вершинами, що описують початки та кінці цих ліній були виконані одні і ті ж самі перетворення. Подібно цьому, також зберігаються і інші взаємні відношення, наприклад перпендикулярність і т.п.

Під час роботи над просторовими задачами, часто згадуються терміни евклідової геометрії. Вони базуються на головних основах тривимірного простору, які складаються з осей X, Y та Z координат. Проте у широкому спектрі задач, більш доцільним є мислення з точки зору проекційної геометрії, яка має додатковий (четвертий) вимір W. Цей вимір слугує доповненням до основних трьох. Такий, чотиривимірний, простір називається "проекційним простором". Координати у проекційному просторі мають назву "однорідні".

Основою Афінних перетворень є матриці. Усі переміщення та перетворення у них відбуваються за рахунок перемноження матриці тіла

об’єкту, яка записана у однорідних координатах на матрицю, яка відповідає за конкретну трансформацію (основні матриці трансформацій будуть розглянуті у подальших підпунктах).

З усіх перетворень можна виділити основні і це:

- Переміщення
- Обертання
- Масштабування

Важливою також є пам’ятка про те, що обертання об’єкту відносно точки, яка слугує якорем обертання і не лежить у початку координат, складається з наступних етапів, що виконуються покроково, один за одним:

1) Множення матриці тіла об’єкту на помножене на міну один матрицю переміщення, яка являтиме собою вектор для переміщення об’єкту відносно початку координат, що утворений в результаті віднімання від точки, що слугує якорем, точки початку координатних осей;

2) Множення матриці тіла об’єкту на матрицю повороту із заданим кутом.

3) Повторне Множення матриці тіла об’єкту на матрицю переміщення.

Далі буде наведено приклад того, як виглядають матриці для переміщень та перетворень у загальному випадку:

- Зсув(переміщення)
  - У двох вимірах:

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 1 \end{matrix}$$

  - У трьох вимірах:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{matrix}$$

- Масштабування
  - У двох вимірах:

$$\begin{matrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{matrix}$$

- У трьох вимірах:

$$\begin{matrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

- Повороти
  - У двох вимірах:

$$\begin{matrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{matrix}$$

  - Навколо осі X у трьох вимірах:

$$\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

  - Навколо осі Y у трьох вимірах:

$$\begin{matrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

  - Навколо осі Z у трьох вимірах:

$$\begin{matrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{matrix}$$

1.3.2. Звертаючись на пряму до координат

Звернення на пряму, це звернення безпосередньо до об'єкту тривимірного простору, а саме до координат кожної з його вершин у просторі. Цей вид звернення потребує окремого звернення до кожного елементу об’єкту. Проте на відміну від подібних звернень у Афінних перетвореннях він менше навантажує комп'ютер. Це досягається завдяки тому, що в ньому відсутні багаторазові пусті операції по типу множення на нуль, які дуже часто виникають під час роботи з матрицями.

Далі буде наведено приклад того, як виглядають формули для переміщень та перетворень у загальному випадку:

- Зсув(переміщення)

- У двох вимірах:

$$X = X + Tx;\ Y = Y + Ty$$

- У трьох вимірах:

$$X = X + Tx;\ Y = Y + Ty;\ Z = Z + Tz$$

- Масштабування
  - У двох вимірах:

$$X = X * Sx;\ Y = Y * Sy;$$

  - У трьох вимірах:

$$X = X * Sx;\ Y = Y * Sy;\ Z = Z * Sz$$

- Повороти
  - У двох вимірах:

$$X = \cos(\alpha) * x + \sin(\alpha) * y;\ Y = -\sin(\alpha) * x + \cos(\alpha) * y$$

  - Навколо осі X у трьох вимірах:

$$Y = \cos(\alpha) * y - \sin(\alpha) * z;\ Z = \sin(\alpha) * y + \cos(\alpha) * z$$

  - Навколо осі Y у трьох вимірах:

$$X = \cos(\alpha) * x + \sin(\alpha) * z;\ Z = -\sin(\alpha) * x + \cos(\alpha) * z$$

  - Навколо осі Z у трьох вимірах:

$$X = \cos(\alpha) * x + \sin(\alpha) * y;\ Y = -\sin(\alpha) * x + \cos(\alpha) * y$$

1.4. Алгоритми відкидання прихованих елементів об’єктів тривимірного простору

1.4.1. Алгоритм Робертса

Суть алгоритму Робертса полягає у визначенні векторів нормалей для кожної з граней фігури. Це є необхідний алгоритм, оскільки, якщо користувач бачить усі грані фігури, у нього виникає ефект двоякості зображення, коли стає незрозумілим ні глибина зображення, ні позиція з якої він спостерігає об’єкт.

Алгоритм може використовуватись тільки для випуклих багатокутників. Якщо фігура не представляє собою випуклий багатогранник, тоді її

необхідно розбити на окремі частини і вже потім розглядати їх всі разом, як єдине ціле.

У просторі, випуклий багатогранник, представляє собою набір площин, що формують його сторони. Виходячи з цього, вхідними для алгоритму є опуклі багатогранники, які задані переліком своїх вершин, або ж граней, що задаються у вигляді площин, які записані у канонічному вигляді у об’єктній системі координат:

$$Ax + By + Cz + D = 0$$

рівняння площини у тривимірній системі координат

Виходячи з попереднього, кожна з площин задається чотиривимірним вектором, а кожна її точка, задається в однорідних координатах, які так само складають собою чотиривимірний вектор:

$$P = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}, \text{ p} = [x \quad y \quad z \quad 1]$$

Впевнитись у тому, що точка належить площині, можна за допомогою скалярного перемноження двох векторів (P * p). Якщо точка належить площині, то результат їх множення буде дорівнювати нулю. У інших випадках, ми отримуємо два варіанти. Це або ж позитивне значення, тобто додатне число, або ж негативне, тобто від’ємне.

Результат такого множення показують, по яку сторону від площини лежить точка.

З векторів площин будується матриця, що має назву узагальнена матриця опису багатогранника або ж тіло багатокутнка:

$$\text{M} = \begin{bmatrix} a(1) & b(1) & c(1) & d(1) \\ a(2) & b(2) & c(2) & d(2) \\ a(3) & b(3) & c(3) & d(3) \\ \cdots & \cdots & \cdots & \cdots \\ a(n) & b(n) & c(n) & d(n) \end{bmatrix}$$

У своєму алгоритмі Робертс розглядad відрізки, які є пересіченням граней багатогранника.

З тіла багатогранника можна отримати певну інформацію, про те, які саме грані перетинаються у вершинах.

Якщо вершина $v = [x \quad y \quad z \quad 1]$ належить декільком граням одночасно, то вона буде задовільнювати рівності:

$$v * Q = e$$

Q – матриця із векторів-стовпців, $e - [0 \quad 0 \quad 0 \quad 1]$

Виходячи з останнього твердження, координати вершини знаходяться за співвідношенням:

$$v = e * Q^{(-1)}$$

З усього цього можна зробити визначення, що якщо для будь-яких площин існує обернена матриця, тоді ці площини мають спільну вершину.

Цей алгоритм насамперед видаляє з кожного багатогранника полігони, вершини, або ж ребра, які екрануються їхнім власним тілом. Для перевірки свого алгоритму, можна виконати дуже простий тест. Суть тесту полягала у тому, що якщо одна, або дві суміжні грані обернені своєю зовнішньою стороною до точки спостереження, то полігон є видимим. Цей тест виконується за допомогою обчислення скалярного добутку зовнішнього вектору нормалі площини на координаи позиції спостерігача.

Якщо існує необхідність визначення прихованих елементів у великої кількості фігур, тоді кожна з видимих площин для кожного об’єкту порівнюється з кожним із інших об’єктів.

Якщо спостерігач розташований у точці u, а задані кінці відрізку r та s, то відрізок задається рівністю:

$$v = r + t * (s - r) = r + t * d, \; 0 <= t <= 1$$

Пряма, яка прямує у точку, відповідно до параметру t, задається рівністю:

$$w = u + T * g = r + t * d + T * g$$

Для визначення частини відрізку, що теоретично закривається будь-яким існуючим на сцені тілом, достатньо знайти значення t і T. За цих значень добуток вектору w на узагальнену матрицю є позитивним. Для кожної площини записується нерівність:

$$Q = (v * P) + t(d * P) + T(g * P) > 0$$

Ці умови мають виконуватись для всіх площин.

Припускаючи, що q = 0, отримуємо систему рівностей, вирішення якої дають нам точки зміни видимості відрізку. Результат можна отримати шляхом рішення різних пар рівностей з цієї системи. Число різноманітних рішень при N площинах, дорівнює N * (N - 1) / 2.

Так як об’єм обчислень росте із збільшенням числа багатокутників, то бажано по можливості скорочувати їх кількість. Наприклад розраховувати площини для багатокутників, які не є трикутниками. Але при цьому виникає інша проблема, суть якої полягає у тому, що погрішності при розрахунках віднімають можливість побудувати чітку площину через чотири, або більше точок.

1.4.2. Алгоритм Z-буфера

Алгоритм свого часу був запропонований Едом Кетмулем. Він являє собою узагальнення буферу для кадру. Стандартний буфер кадру зберігає в собі коди кольору для кожного пікселя в усьому просторі зображення. Сама ідея цього алгоритму полягає в тому, щоб для кожного пікселя, окрім його власний параметрів додатково зберігати параметри координати по Z, тобто глибину. При оновленні даних кожного пікселя пікселя в буфері кадру, значення глибина порівнюється з тим, яке вже знаходиться в буфері. Якщо ж глибина нового пікселя є меншою, ніж координата старого, тоді це означає, що він знаходиться ближче до спостерігача. У такому випадку параметри нового пікселя та його Z-координата заносяться в буфер. У іншому ж випадку, нічого не робиться.

Цей алгоритм є найпростішим з усіх алгоритмів, що призначені для видалення прихованих елементів об’єктів, але він вимагає великого обсягу пам'яті комп’ютера. Дані параметра, який відповідає за глибину, зазвичай можна тримати у розрядності близько 20 біт. У такому випадку при зображенні трохи застарілого телевізійного розміру в 768 × 576 пікселів для зберігання глибини необхідний обсяг пам'яті дорівнює приблизно одному

Мбайту. А от загальний обсяг пам'яті при трьох байтах для кожного із значень RGB складе більше двох з половиною Мбайтів.

Швидкість виконання даного алгоритму не залежить від складності самої сцени. Усі складові сцени, можуть оброблятися у довільному порядку. Для скорочення витрат часу на його роботу, нелицьові об’єкти можуть бути передчасно видалені. Якщо використовується покроковий алгоритм заливки, то можна легко зробити покрокове обчислення глибини чергового пікселя, додатково зберігаючи глибину його вершин і обчислюючи приріст dz Z-координати при переміщенні уздовж X на dx, рівне 1. Якщо відомо рівняння площини, в якій лежить полігон, то можна обійтися без зберігання Z-координат вершин.

Нехай рівняння площини має вигляд:

$$A \cdot x + B \cdot y + C \cdot z + D = 0.$$

У такому випадку при C що не дорівнює нулю:

$$z = - (A \cdot x + B \cdot y + D) / C$$

Знайдемо приріст для пікселя по Z при кроці по X на dx, при цьому пам'ятаючи, що Y – це чергова оброблюваного рядка – константа:

$$dz = - (A \cdot (x + dx) + D) / C + (A \cdot x + D) / C = -A \cdot dx / C$$

але dx = 1, тому

$$dz = -A / C$$

Основний недолік цього конкретного алгоритму - це додаткові витрати пам'яті. Для їх зменшення можна розбивати екран, на я який проекціюється зображення, на кілька менших прямокутників. Взагалі, можна використовувати Z-буфер у вигляді одного рядка, але це призведе до збільшення часу, оскільки кожен прямокутник буде оброблятися n-кількість разів, у залежності від того на скільки областей розбитий екран проекціювання. У такому випадку, можна досягти зменшення витрат часу, але тільки за рахунок попереднього сортування прямокутників на площині.

Усі інші недоліки цього алгоритму полягають у тому, що пікселі заносяться в буфер в довільному порядку. При цьому виникають труднощі з

реалізацією ефектів прозорості або просвічування. Також виникає проблема при усуненні ефекту сходинок з використанням предфільтрації. Предфільтрація у цьому випадку, це коли кожен піксель екрану трактується як точка кінцевого розміру, атрибути якого встановлюються в залежності від того яка частина пікселя зображення потрапляє в піксель мпиого екрану. Інший же підхід до усунення ефекту сходинок, заснований на постфільтрації. Його суть полягає у усередненні всіх значень кожного пікселя з використанням зображення з великою роздільною здатністю. Цей метод у порівнянні з предфільтрацією реалізується простіше, так виходить за рахунок збільшення витрат пам'яті і часу. Для його втілення використовуються два методи. Суть першого полягає в тому, аби збільшити розширення тільки кадрового буфера, що зберігає атрибути пікселів, а розширення сітки глибини збігається з розмірами екрану на який проекціюється простір. Глибина зображення у цьому випадку обчислюється для усередненої групи пікселів. Нажаль цей метод є непридатним, коли відстань до спостерігача імітується зміною інтенсивності пікселів. Суть другого методу полягає у тому, що і кадровий буфер і буфер глибини мають збільшене розширення. При цьому усереднення відбувається для кожного з буферів.

Наведення загальної схеми роботи алгоритму Z-буфера (буфера глибини):

- Ініціалізація двох буферів, кадрового і глибини. Перший буфер замальовується обраним користувачем фоном. Другий буфер замальовується мінімальним значенням глибини.
- Перетворення кожного з об’єктів сцени, та приведення їх до растрової форми. При цьому для кожного пікселя екрану обчислюється параметр його глибини. Якщо обчислена глибина є більшою, ніж глибина, яка вже наявна в буфері глибин, в буфері оновлюються дані для конкретного пікселя.
- У разі, якщо є масщтабування, виконати усереднення зображення у пониженому дозволі.

1.5. Висновки до першого розділу

Досліджено метод моделювання положення об’єктів у просторі. Для системи моделювання необхідно розробити систему взаємопов’язаних класів, кожен з яких буде описувати конкретний елемент структури тривимірного об’єкту. Необхідним моментом є реалізація методів, які допоможуть цим класам гармонічно взаємодіяти між собою усередині цієї системи.

Переміщення об’єктів є дуже важливою частиною їх взаємодії у просторі. Матричні Афінні перетворення надають кращий контроль та глибше розуміння того як саме переміщення відбуваються, проте більш доцільною є реалізація звернень на пряму до даних координат вершин об’єктів. Це допоможе полегшити розрахункові навантаження для комп’ютера, відкинувши пусті операції, що часто виникають при множенні матриць під час Афінних перетворень.

Досліджені алгоритм Робертса для відкидання прихованих поверхонь опуклих об’єктів тривимірного простору, а також досліджено алгоритм Z-буфера для схожої задачі. Для візуалізації роботи алгоритму пошуку та розташування об’єкту, необхідним є поєднання цих алгоритмів та модифікація отриманого таким чином нового алгоритму.

Розділ 2. Алгоритмічне забезпечення програмного засобу

2.1. Опис створеної системи класів

Для візуалізації роботи основного алгоритму необхідно створити структуру, за допомогою якої буде описуватись об’єкт тривимірного простору. Для цього було створено наступний перелік класів, сукупність яких описує структуру об'єкту тривимірного простору:

- Coordinates_3D – це клас, який дозволяє створювати об’єкти, які визначають положення у тривимірній системі координат.
- Dot_3D – це клас, який дозволяє створювати об’єкти, які визначають точки у тривимірній системі координат.
- Line_3D – це клас, який дозволяє створювати об’єкти, які визначають лінії, або ж грані об’єктів у тривимірній системі координат.
- Polygon_3D – це клас, який дозволяє створювати об’єкти, які визначають полігони у тривимірній системі координат.
- Object_3D – це клас, який дозволяє створювати об’єкти, які визначають об’єкти у тривимірній системі координат.

Також додатково до цієї структури, був створений клас Vector_3D, який дозволяє створювати об'єкти, що визначають вектори у тривимірному просторі.

Для моделювання та рендерингу отримуваного зображення був створений клас Display_3D, який дозволяє створювати об’єкти, що визначають камеру спостерігача та її положення у просторі. У конкретному випадку створюється камера для користувача засобу.

Також був створений основний керуючий клас, який зберігає поля необхідні для коректної роботи створюваної програми Main_3D.

2.2. Алгоритм для візуалізації

Як було сказано раніше, для візуалізації створеного середовища буде використовуватись модифікований алгоритм, що побудований при поєднанні алгоритмів Робертса та Z-буфера.

Основною задачею модифікації є його узагальнення для конкретного випадку моделювання і рендерингу, а також поєднання його роботи з камерою, програмна реалізація якої описана у класі Display_3D.

Для початку, необхідно розуміти, що площина MainPlane об’єкту камери, це площина на яку будуть проекціюватись усі об’єкти, тобто елементи, або ж ще можна сказати фігури на сцені.

Кожен об’єкт має свій набір вершин, які і перевіряються на те, видно їх, або ж ні. Весь алгоритм можна поділити на кроки, які надалі і будуть розписані:

1. Першим кроком обирається точка у просторі і перевіряється на видимість для користувача.
   1.1. Точка має лежати у просторі так, щоб при підстановці у формулу площини, вона видавала позитивне значення для кожної з них. Для цього на етапі створення камери всі для кожної площини створюється та корегується її рівняння так, що вектори їх нормалей були направлені усередину камери. Якщо точка задовільняє цьому етапу, тоді вона переходить до наступного. У іншому ж випадку, вона відкидається на майбутнє.

$$A*X+B*Y+C*Z+D > 0$$

   1.2. Від ораної точки до площини будується лінія. У цьому може допомогти вектор Z власної системи координат камери, оскільки він завжди направлений на дев’яносто градусів від площини.

$$Line\_3D = (Dot\_3D, Dot\_3D+vector)$$

   1.3. Створена лінія може перетинати площини інших об’єктів і тому необхідним є перевірити чи робить вона це. У випадку, якщо вона перетинає площину, у межах її належності до будь-якого об’єкта окрім камери, тоді точка яка була взята відкидається. Якщо ж лінія не перетинає жодної площини окрім головної площини камери, тоді можна переходити до наступного кроку.

Точка перетину лінії і площини:

$$Dot\_3D = (x^{PV}*t+x^{PZ}, y^{PV}*t+y^{PZ}, z^{PV}*t+z^{PZ)}$$

$$t=(-1)*tT/tB$$

$$tT = A*x^{PZ}+B*y^{PZ}+C*z^{PZ}+D$$

$$tB = A*x^{PV}+B*y^{PV}+C*z^{PV}+D$$

A+B+C+D – взято з рівняння площини

PV, PZ – значення взяті з параметричного рівняння прямої. Перше це вектор, а друге це стартова точка.

2. Оскільки камера представлена у вигляді довільного прямокутника у просторі, який має можливість переміщуватись, то його значення можуть приймати найрізноманітніші величини. Проте це не є проблемою, оскільки можна розрахувати довжину від точки перетину лінією площини, до будь якої зі сторін камери і таким чином отримати індекс пікселя на камері.

   Відстань від точки до площини:

$$Abs(A*X+B*Y+C*Z+D)/Sqrt(A^{\wedge 2}+B^{\wedge 2}+C^{\wedge 2})$$

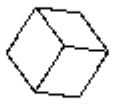
   A+B+C+D – взято з рівняння площини

   X, Y, Z – координати точки

3. Індекс пікселя замальовується кольором у залежності від обраного, або ж заданого.

$$SetPixel(X, Y, Color)$$

Результатом роботи алгоритму є зображення на якому були відкинуті всі приховані об’єкти (див. рис. 2.2.1):



Результат роботи алгоритму рис.2.2.1

2.3. Алгоритм пошуку місця та розташування

Уявімо собі n-розмірну сцену, де n це кількість тривимірних об’єктів, на якій нам необхідно розмістити новий об’єкт. Оскільки майже завжди при вирішенні подібних задач, усі об’єкти мають хіт-бокс, за основу для взаємодії, беремо

його. Хіт-бокс, це можна сказати, коробка навколо об’єкта, яка відповідає за його взаємодію з цифровим навколишнім середовищем. Межі хіт-боксу визначаються мінімальними та максимальними координатами по кожній з осей. У двовимірній графіці, це лише X і Y, а от у тривимірній це все X,Y і Z.

При додаванні до сцени нового об'єкту є необхідним задати його положення, у іншому випадку, об'єкт має додатись до точки початку координат. Другий варіант можна відкинути і завжди задавати точку розміщення.

Існує два варіанти розвитку подій під час розміщення. Перший, це якщо новий об'єкт розміщується у просторі не зачіпаючи при цьому інші об'єкти, які вже існують. А от другий, це коли об'єкт все ж таки пересікається з вже існуючими об'єктами. Саме у другому випадку з'являється необхідність пошуку найліпшого розташування, яке буде найближче до заданої точки.

Розроблений алгоритм можна записати у вигляді послідовних кроків, які необхідно виконати для пошуку рішення. Отож:

1. Перевірка колізії розміщеного об’єкта з вже існуючими.
   1.1. Якщо об’єкт зайняв вільний простір, то він додається і додаткових дій не потребує.
   1.2. У іншому випадку відбувається перехід до наступного кроку.
2. Пошук відстані на яку необхідно перемістити новий об’єкт у кожному з шести напрямків, щоб він залишив межі вже існуючого об’єкту.
3. Переміщення об’єкта у сторону, довжина до якої є найменшою.
4. Далі є два розвитки подій. Або об’єкт зайняв вільний простір і може бути спокійно розміщений, або ж він знов потрапив до простору іншого об’єкта.
   4.1. У першому випадку об’єкт додається до сцени (тільки, якщо це перша ітерація).
   4.2. У другому випадку всі дії починаючи з другого кроку, але проводяться вже для наступного об’єкту і так до тих пір, поки не буде знайдено вільну позицію, де можна буде розташувати об’єкт.
5. Знайшовши вільну позицію, розраховується довжина від заданої для розміщення точки до точки, яка була знайдена і запам’ятовується.

6. Повернення до другого кроку і переміщення по всім іншим напрямкам, від найменшого до найбільшого.
7. У разі, якщо результатом переміщення стає точка, відстань до якої є більшою ніж до вже існуючої, тоді далі шлях не розглядається і відкидається.

2.4. Висновок до другого розділу

Описано структуру взаємопов’язаних класів, яка є необхідною для створення зрозумілих для обчислювальної машини об’єктів у тривимірному просторі.

Описано запропонований алгоритм для візуалізації, який є модифікацією поєднання алгоритмів Робертса та Z-буферу.

Описано проблему розміщення об’єкту у просторі та запропоновано алгоритм розміщення об’єкта у просторі сцени, на основі його хіт-боксу.

На базі розроблених класів та алгоритмів стає можливим створення програмного засобу, що дозволить щільно розміщувати об’єкти у тривимірному просторі.

Розділ 3. Програмне забезпечення засобу

У якості середовища для розробки програми було обрано програмне забезпечення Visual Studio 2022, яке є повнофункціональне інтегрованим середовищем розробки.

Мовою програмування для створення програмного забезпечення була обрана C#. На сьогоднішній день, це одна з найпотужніших мов. Вона є C-подібною, тобто такою, яка за своїм синтаксисом дуже схожа на C++, та інші C-подібні мови. Також вона є об'єктно-орієнтованою, що дозволяє їй підтримувати поліморфізм, наслідування, перевантаження операторів, статичну типізацію та ін. А також об'єктно-орієнтований підхід дозволяє вирішувати задачі зі створення великих за об'єктом, але у той самий час гнучких програмних додатків.

Платформа, яку було обрано для розробки, стала .NET та її елемент Windows Forms. Серед її переваг, є підтримка багатомовного коду, адже основою платформи є загальномовне середовище виконання Common Language Runtime (CLR), завдяки чому вона може підтримувати велику кількість мов одночасно. Серед мов програмування, які підтримує CLR, окрім C# є VB.NET, C++, F#. Під час компіляції коду на будь якій них, компілюється збірка на загальній мові Common Intermediate Language (CIL), яка у певній мірі є асемблером для .NET. Тому за необхідності, можна різні модулі програми реалізувати на різних мовах. Також вона є кросплатформеною, а останні її версії підтримуються на більшості сучасних операційних системах. Ще одною великою перевагою для платформи є її велика бібліотека класів. Вона, має єдину для всіх мов, що підтримуються бібліотеку класів. CLR середовище, та базова бібліотека класів являють собою основу для цілого стеку технологій. Наприклад, для роботи з базами даних у стеці передбачена технологія ADO.NET та Entity Framework Core. Для побудови графічних застосунків з потужним інтерфейсом існує WFP та UWP, а для створення більш простих графічних додатків є Windows Forms. Також, платформа має в собі технології для розробки мобільних застосунків у вигляді Xanmarin і платформи для

створення WEB-додатків у вигляді технології ASP.NET. І що беззаперечно є дуже важливим при роботі з графікою, платформа .NET є високопродуктивною та добре оптимізованою.

.NET та C# також мають функцію автоматичного очищення сміття. Це позитивно вливає на хід розробки програмного застосунку, адже позбавляє від необхідності постійно турбуватись про очищення пам’яті. Середовище CLR саме виконує очистку пам’яті під час виконання програми.

Зазвичай, програми, які написані на мові програмування C#, називають керованим кодом, оскільки вони керуються загальномовним середовищем. Наприклад C++, компілюється у звичайний машинний код і через це .NET не керує застосунком.

3.1. Опис створених класів

Створений клас Coordinates_3D

Клас Coordinates_3D містить в собі: три поля, два конструктори і десять методів:

Поля класу Coordinates_3D:

- Публічні поля X, Y і Z, які оголошують змінні типу числа з плаваючою точкою, які в свою чергу відповідають за координати по осям X, Y і Z відповідно.

Скріншот реалізації полів

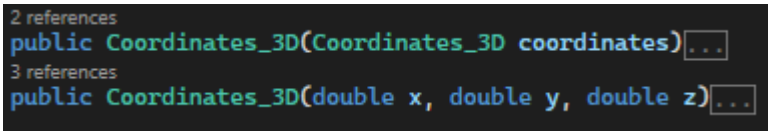
```
17 references
public double X { get; set; }
17 references
public double Y { get; set; }
17 references
public double Z { get; set; }
```

Конструктори класу Coordinates_3D:

- Перший конструктор приймає об’єкт поточного класу і присвоює значення його полів, полям створюваного об’єкту. Таким чином дублюючи прийнятий об’єкт.

- Другий конструктор приймає три різні змінні типу числа з плаваючою точкою і присвоює їх значення відповідним полям створюваного об’єкту.

Скріншот реалізації конструкторів



Методи класу Coordinates_3D:

- Методи для переміщення координат у тривимірній системі координат:
  - Публічний метод Translation приймає в якості параметра, об’єкт класу Vector_3D. Він відповідає за перенесення координат у просторі на заданий об'єктом, який приймається, вектор. Процес відбувається за рахунок додавання до кожного поля, відповідних даних вектору. Напрямок переміщення визначається напрямком самого вектору.
  - Публічні методи MoveX, MoveY і MoveZ приймають в якості параметрів змінну типу числа з плаваючою точкою. Вони відповідають за перенесення координат тільки по одній з осей, відповідно до назви кожного, по X, Y та Z. Вони додають до значення відповідного поля, значення яке приймають.
  - Публічні методи RotateXByDegree, RotateYByDegree і RotateZByDegree приймають два параметри, перший, це змінна типу числа з плаваючою точкою, а другий у вигляді об’єкту классу Vector_3D. Беручи за основу змінну яку вони приймають, методи розраховують кут у радіанах і звертаються до методів RotateXByRadians, RotateYByRadians і RotateZByRadians.
  - Публічні методи RotateXByRadians, RotateYByRadians і RotateZByRadians приймають два параметри, перший, це змінна типу числа з плаваючою точкою, що відповідає за кут оберту а другий у вигляді об’єкту класу Vector_3D, поля якого

відповідають за точку, навколо якої відбуватиметься обертання. Вони звертаються до методу Translation для встановлення об’єкту у позицію для початку обертання, після чого виконується саме обертання і наприкінці знову виконується звертання до методу Translation для встановлення об’єкту у фінальну позицію.

Скріншот реалізації методів

```
8 references
public void Translation(Vector_3D vector)...
1 reference
public void MoveX(double x)...
1 reference
public void MoveY(double y)...
1 reference
public void MoveZ(double z)...
2 references
public void RotateXByDegrees(double degree, Vector_3D anchor)...
2 references
public void RotateYByDegrees(double degree, Vector_3D anchor)...
2 references
public void RotateZByDegrees(double degree, Vector_3D anchor)...
1 reference
public void RotateXByRadians(double radian, Vector_3D anchor)...
1 reference
public void RotateYByRadians(double radian, Vector_3D anchor)...
1 reference
public void RotateZByRadians(double radian, Vector_3D anchor)...
```

Створений клас Dot_3D

Клас Dot_3D містить в собі: чотири поля, два конструктори, три визначення для операторів і тринадцять методів.

Поля класу Dot_3D:

- Приватне поле Coordinates, оголошує об’єкт класу Coordinates_3D, що відповідає за параметри для розташування точки у тривимірному просторі.
- Відкриті поля X, Y і Z звертаються до об’єкту класу Coordinates_3D і повертають, або ж задають у ньому значення.

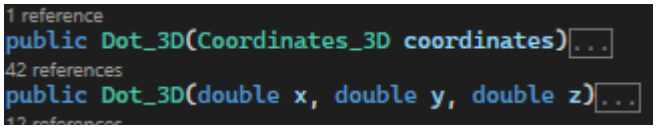
Скріншот реалізації полів

```
16 references
private Coordinates_3D Coordinates { get; set; }
66 references
public double X { get { return Coordinates.X; } set { Coordinates.X = value; } }
66 references
public double Y { get { return Coordinates.Y; } set { Coordinates.Y = value; } }
66 references
public double Z { get { return Coordinates.Z; } set { Coordinates.Z = value; } }
```

Конструктори класу Dot_3D:

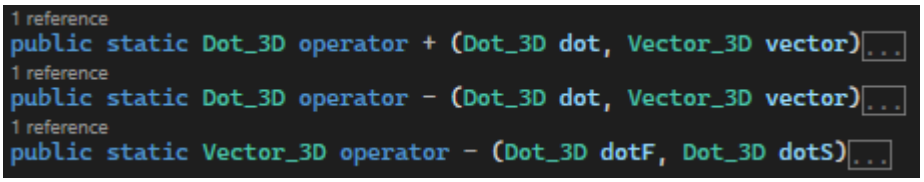
- Перший конструктор приймає об’єкт класу Coordinates_3D і звертається до конструктору класу Coordinates_3D.
- Другий конструктор приймає три параметри типу числа з плаваючою точкою і звертається до конструктору класу Coordinates_3D.

Скріншот реалізації конструкторів



Визначення для операторів у класі Dot_3D:

- Оператор додавання, який дозволяє додавати до об'єкту поточного класу, об'єкт класу Vector_3D. Задачею даного оператора є виконання перенесення точки на заданий вектор.
- Оператор віднімання, який дозволяє віднімати від об'єкту поточного класу, об'єкт класу Vector_3D. Задачею даного оператора є виконання перенесення точки на заданий вектор.
- Оператор віднімання, який дозволяє віднімати один від одного два об'єкти поточного класу. Задачею даного оператора є створення нового об’єкту класу Vector_3D.



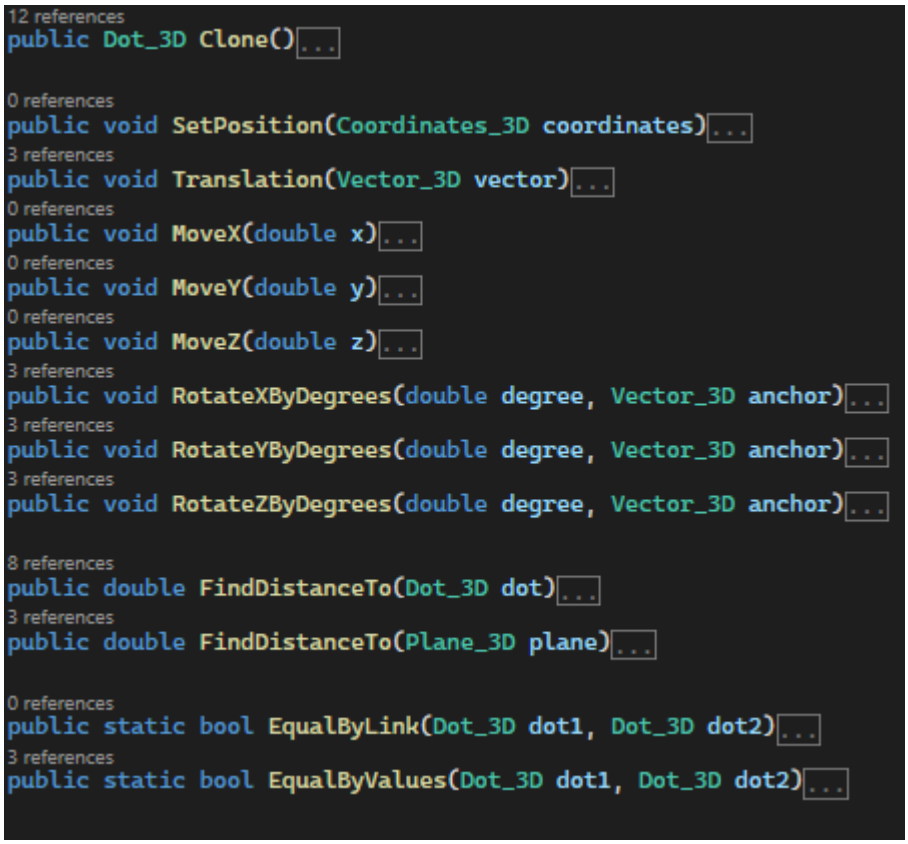
Методи класу Dot_3D:

- Публічний метод Clone, який призначений для створення копії об’єкта. Він повертає об’єкт з ідентичними параметрами, таким чином дублюючи оригінал.
- Методи для переміщення координат точки:
  - Метод SetPosition, який якості параметру приймає об'єкт класу Coordinates_3D. Він замінює значення параметрів полів, таким чином змінюючи позицію точки у просторі.

- Метод Translation, який якості параметру приймає об’єкт класу Vector_3D. Він звертається до методу Translation класу Coordinates_3D.
- Методи MoveX, MoveY і MoveZ, які у якості параметрів, кожен приймають зміну типу числа з плаваючою точкою. Він звертається до методів MoveX, MoveY та MoveZ класу Coordinates_3D.
- Методи RotateXByDegree, RotateYByDegree і RotateZByDegree, які у якості параметрів, кожен приймають змінну типу числа з плаваючою точкою, а також об’єкт класу Vector_3D. Він звертається до методів RotateXByDegree, RotateYByDegree та RotateZByDegree класу Coordinates_3D.

- Група методів FindDostanceTo:
  - Метод, який у якості параметра приймає об’єкт класу Dot_3D, який являється точкою у просторі. Результатом розрахунків при виконанні методу є змінна типу числа плаваючою точкою, що характеризує відстань між точками у просторі.
  - Метод, який у якості параметра приймає об’єкт класу Plane_3D, який являється площиною у тривимірному просторі. Результатом розрахунків у методі, є змінна типу числа з плаваючою точкою, яка характеризує відстань від площини до точки у просторі.
- Статичні методи для порівняння об'єктів класу
  - Метод EqualsByLink, суть якого полягає у порівнянні посилань двох об’єктів класу. Він приймає два об’єкти класу Dot_3D і аналізує чи є вони одним і тим самим об’єктом у пам’яті комп’ютера. Результатом аналізу та розрахунків є логічна змінна.
  - Метод EqualsByValue, що виконує порівняння точок у просторі по їх координатам. Він також приймає два об’єкти класу Dot_3D, але аналізує вже не їх розташування у пам’яті, а величини їх координат. Результатом аналізу та розрахунків також є логічна змінна.

Скріншот реалізації методів



Створений клас Line_3D

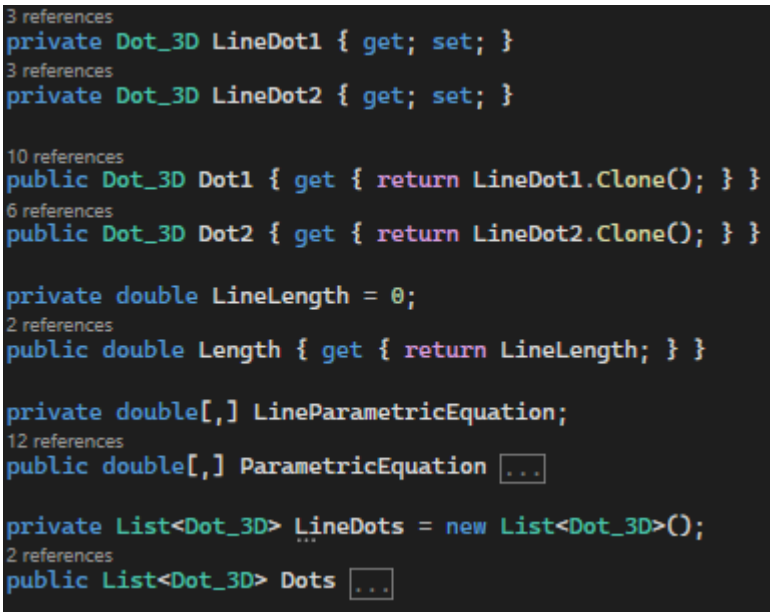
Клас Line_3D містить в собі: десять полів, один конструктор і п'ять методів.

Поля класу Line_3D:

- Два приватні поля LineDot1 і LineDot2, які оголошують об’єкти класу Dot_3D, які визначають початок і кінець заданої грані чи лінії.
- Два публічних поля Dot1 і Dot2, що повертають копії об’єктів класу Dot_3D.
- Приватне поле LineLength, яке оголошує змінну типу числа з плаваючою точкою, яка відповідає за параметр довжини лінії або ж грані.
- Публічне поле Length, яке повертає значення змінної довжини лінії.
- Приватне поле LineParametricEquation, яке оголошує двовимірний масив даних типу чисел з плаваючою точкою, який зберігає в собі значення параметричного рівняння створюваної прямої.
- Публічне поле ParametricEquation, яке повертає копію двовимірного масиву, що відповідає за параметричне рівняння лінії.

- Приватне поле LineDots, яке оголошує список з об’єктів класу Dot_3D, який зберігає в собі точки на лінії, для її майбутнього відображення.
- Публічне поле Dots, яке повертає копію списку точок на лінії.

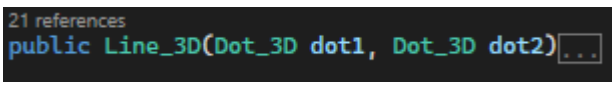
Скріншот реалізації полів



Конструктор класу Line_3D:

- Конструктор приймає у якості параметрів два об'єкти класу Dot_3D і присвоює посилання на них своїм відповідним змінним. Також конструктор звертається до методу Update, який буде описаний нижче.

Скріншот реалізації конструктора



Методи класу Line_3D:

- Публічний метод Update, який відповідає за оновлення даних об’єкту класу. Результатом його виклику є виклик методів FindParametricalEquation, FindLength та FindDots.
- Публічний метод FindParametricalEquation, який відповідає за розрахунок параметричного рівняння прямої. Результатом його виклику є розрахунок даних для кожного елементу масиву, що відповідає за параметричне рівняння прямої а також.
- Публічний метод FindLength, який відповідає за розрахунок довжини лінії або ж грані. Результатом його виклику є виклик методу

FindDistanceTo класу Dot_3D і отримання довжини лінії, а також присвоєння значення цієї довжини відповідному полю.

- Публічний метод FindDots, який відповідає за розрахунок точок на лінії. Розрахунок виконується за принципом, що кожен крок береться нова точка і додається до списку. Крок розраховується через довжину лінії. Результатом розрахунків є отримання списку із точок та присвоєння його відповідному полю для подальшого їх використання з метою демонстрації цієї лінії користувачеві.
- Публічний метод PlaneInstersection, який приймає у якості параметру об’єкт класу Plane_3D. Він відповідає за розрахунок точки у якій пересікаються промінь утворений лінією і площина. Результатом розрахунків цього методу є об’єкт класу Dot_3D, у випадку, коли лінія пересікає площину, або ж null, якщо лінія паралельна площині чи лежить на ній.

Скріншот реалізації методів

```
2 references
public void Update()...
2 references
public void FindParametricEquation()...
2 references
public void FindLength()...
1 reference
public void FindDots()...

2 references
public Dot_3D PlaneIntersection(Plane_3D plane)...
```

Створений клас Plane_3D

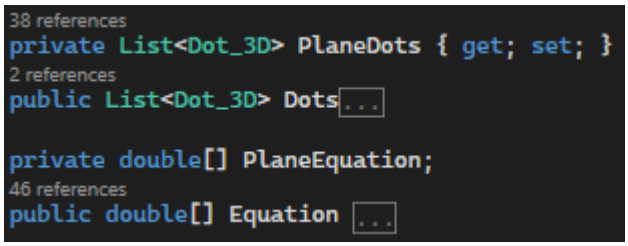
Клас Plane_3D містить в собі: чотири поля, один конструктор і сім методів.

Поля класу Plane_3D:

- Приватне поле PlaneDots, яке оголошує список об’єктів класу Dot_3D, з яких складається фігура на площині.
- Публічне поле Dots, яке повертає копію списку об’єктів класу Dot_3D, для подальшої участі у операціях.
- Приватне поле PlaneEquation, яке оголошує масив чисел з плаваючою точкою, дані якого у майбутньому будуть утворювати рівняння площини у тривимірному просторі.

- Публічне поле Equation, яке повертає копію масиву чисел з плаваючою точкою, який відповідає за рівняння площини у просторі.

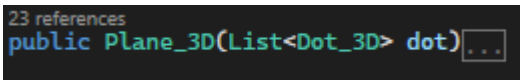
Скріншот реалізації полів



Конструктор класу Plane_3D:

- Конструктор приймає у якості параметрів список об’єктів класу Dot_3D і присвоює посилання на нього відповідному списку який відповідає за точки з яких складається фігура. Також конструктор звертається до методу Update, який буде описаний нижче.

Скріншот реалізації конструктора



Методи класу Plane_3D:

- Методи для різноманітних переміщень площини у просторі:
    - Публічний метод Translation, який відповідає за перенесення площини фігури. Результатом його виклику є звренення кожного об'єкту зі списку об’єктів класу Dot_3D, що відповідають за точки у просторі до відповідного методу Translation їхнього класу.
    - Публічні методи RotateXByDegrees, RotateYByDegrees і RotateZByDegrees, які відповідають за обертання по кожній з осей відповідно до їх назв на заданий. Результатом розрахунків є звернення кожного об'єкту зі списку об’єктів класу Dot_3D, що відповідають за точки у тривимірній системі координат до методів RotateXByDegrees, RotateYByDegrees та RotateZByDegrees.
- Публічний метод Update, який відповідає за оновлення даних об’єкту класу. Результатом його виклику є звернення до методу FindEquation.

- Публічний метод FindEquation, який відповідає за розрахунок рівняння площини по першим трьом точкам зі списку вершин. Результатом його виклику є розрахунок коефіцієнтів рівняння і присвоєння їх елементам одновимірного масиву, який відповідає за рівняння площини.
- Публічний метод EquationCorrection, який відповідає за корегування направленості вектору нормалі через позицію точки у просторі. Метод приймає як параметр об’єкт класу Dot_3D і проводить аналіз її позиції відносно площини. У випадку, якщо точка знаходиться зі сторони у бік якої направлений вектор нормалі, нічого не відбувається. У протилежному випадку, коли точка знаходиться з протилежного боку від направленості вектору нормалі, усі коефіцієнти рівняння площини множаться на мінус один і таким чином рівняння корегується.

Скріншот реалізації методів

```
5 references
public void Translation(Vector_3D vector)...
5 references
public void RotateXByDegrees(double degree, Vector_3D anchor)...
5 references
public void RotateYByDegrees(double degree, Vector_3D anchor)...
5 references
public void RotateZByDegrees(double degree, Vector_3D anchor)...

11 references
public void Update()...
1 reference
public void FindEquation()...
6 references
public void EquationCorrection(Dot_3D dot)...
```

Створений клас Object_3D

Клас Object_3D містить в собі: тридцять одне поля, два конструктори і шістнадцять методів.

Поля класу Object_3D:

- Приватне поле DotValue, яке оголошує список з об’єктів класу Dot_3D, який відповідає за вершини тривимірного об’єкту.
- Приватне поле LineValue, яке оголошує список з об’єктів класу Line_3D, який відповідає за грані тривимірного об’єкту.

- Приватне поле PlaneValue, яке оголошує список з об’єктів класу Plane_3D, який відповідає за полігони тривимірного об’єкту.
- Приватне поле CenterValue, яке оголошує об’єкт класу Dot_3D, який відповідає за координати точки геометричного центру тривимірного об’єкту.
- Приватні поля MinXIndexValue, MinYIndexValue і MinZIndexValue, які оголошують списки з цілочисельних змінних, які відповідають за індекси точок у списку вершин, що є мінімальними по кожній з осей відповідно до назви поля.
- Приватні поля MaxXIndexValue, MaxYIndexValue і MaxZIndexValue, які оголошують списки з цілочисельних змінних, які відповідають за індекси точок у списку вершин, що є максимальними по кожній з осей відповідно до назви поля.
- Приватне поле SizeValue, яке оголошує об’єкт класу Dot_3D, який відповідає за розміри об’єкта у просторі.
- Приватне поле VolumeValue, яке оголошує змінну типу числа з плаваючою точкою, яка відповідає за об’єм тривимірного об’єкта.
- Публічне поле Color, яке оголошує колір для об’єкту.
- Публічне поле Dots, яке повертає копію списку класу Dot_3D з вершинами тривимірного об’єкта.
- Публічне поле Lines, яке повертає копію списку класу Line_3D з гранями тривимірного об’єкта.
- Публічне поле Planes, яке повертає копію списку класу Plane_3D з полігонами тривимірного об’єкта.
- Публічне поле Center, яке повертає копію об’єкта класу Dot_3D, який відповідає за координати центру тривимірного об’єкта.
- Публічні поля MinX, MinY і MinZ, які створюють список з копій вершин, що відповідають мінімальним значенням по кожній з осей відповідно до назви поля і повертають їх.

- Публічні поля MaxX, MaxY і MaxZ, які створюють список з копій вершин, що відповідають максимальним значенням по кожній з осей відповідно до назви поля і повертають їх.
- Публічні поля MinXIndex, MinYIndex і MinZIndex, які створюють копію списку з індексами мінімальних значень вершин тривимірного об’єкта відповідно до назви поля і повертають їх.
- Публічні поля MaxXIndex, MaxYIndex і MaxZIndex, які створюють копію списку з індексами максимальних значень вершин тривимірного об’єкта відповідно до назви поля і повертають їх.
- Публічне поле Size, яке повертає копію об’єкта класу Dot_3D, який відповідає за розміри тривимірного об'єкта.
- Публічне поле Volume, яке повертає значення числа з плаваючою точкою, що відповідає за об’єм тривимірного об’єкта.

Скріншот реалізації полів

```
private List<Dot_3D> DotValue = new List<Dot_3D>();
private List<Line_3D> LineValue = new List<Line_3D>();
private List<Plane_3D> PlaneValue = new List<Plane_3D>();
private Dot_3D CenterValue = new Dot_3D(0, 0, 0);
private List<int> MinXIndexValue;
private List<int> MinYIndexValue;
private List<int> MinZIndexValue;
private List<int> MaxXIndexValue;
private List<int> MaxYIndexValue;
private List<int> MaxZIndexValue;
private Dot_3D SizeValue = new Dot_3D(0, 0, 0);
private double VolumeValue = 0;
public Color Color = Color.Black;
71 references
public List<Dot_3D> Dots ...
7 references
public List<Line_3D> Lines ...
18 references
public List<Plane_3D> Planes ...
16 references
public Dot_3D Center ...
4 references
public List<Dot_3D> MinX...
4 references
public List<Dot_3D> MinY...
4 references
public List<Dot_3D> MinZ...
3 references
public List<Dot_3D> MaxX...
3 references
public List<Dot_3D> MaxY...
3 references
public List<Dot_3D> MaxZ...
0 references
public List<int> MinXIndex ...
0 references
public List<int> MinYIndex ...
0 references
public List<int> MinZIndex ...
0 references
public List<int> MaxXIndex ...
0 references
public List<int> MaxYIndex ...
0 references
public List<int> MaxZIndex ...
6 references
public Dot_3D Size ...
0 references
public double Volume ...
```

Конструктори класу Object_3D:

- Перший конструктор приймає у якості параметра об’єкт класу Object_3D і на його базі створює новий об'єкт.
- Другий конструктор приймає у якості параметрів три списки з об'єктами Dot_3D, Line_3D, Plane_3D і на їх базі створює новий об’єкт.

Скріншот реалізації конструкторів

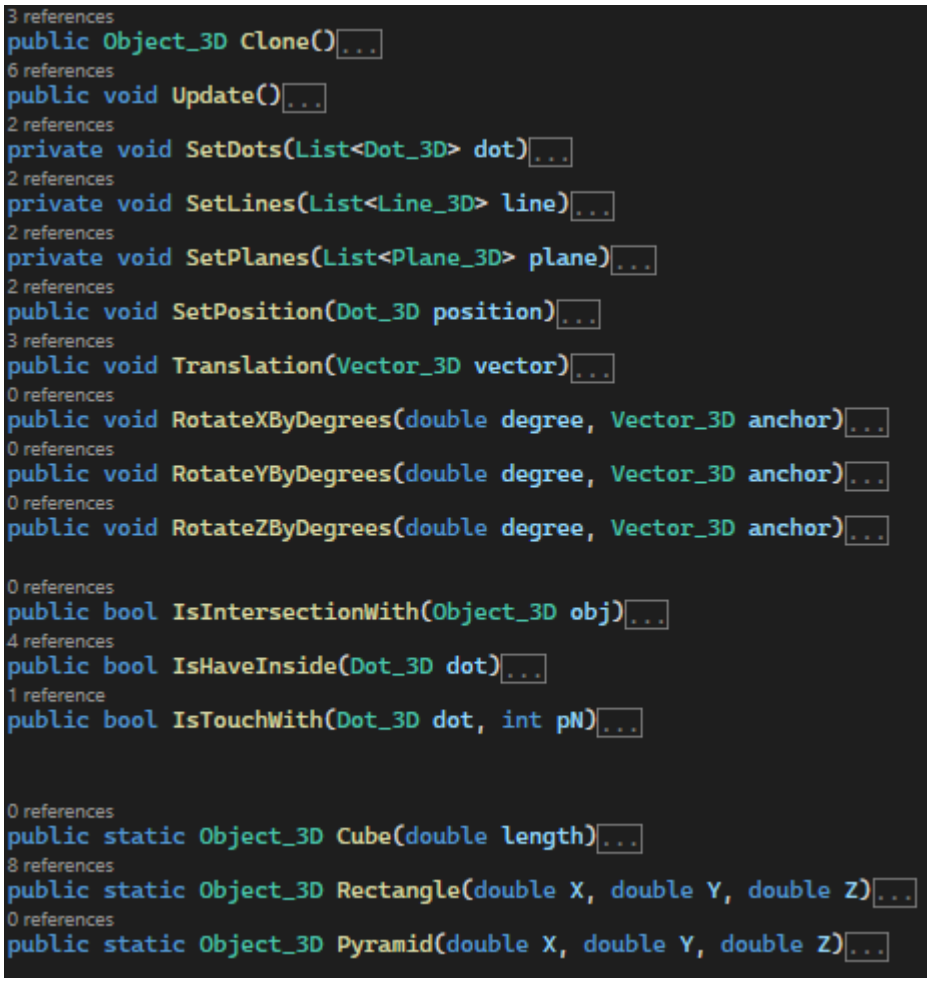
```
1 reference
public Object_3D(Object_3D object_3D)...
3 references
public Object_3D(List<Dot_3D> dot, List<Line_3D> line, List<Plane_3D> plane)...
```

Методи:

- Приватний метод SetDots, який відповідає за присвоєння об’єктів класу Dot_3D до списку, який відповідає за вершини тривимірного об’єкту.
- Приватний метод SetLines, який відповідає за присвоєння об’єктів класу Line_3D до списку, який відповідає за грані тривимірного об’єкту.
- Приватний метод SetPlanes, який відповідає за присвоєння об’єктів класу Plane_3D до списку, який відповідає за полігони тривимірного об’єкту.
- Публічний метод Clone, який створює і повертає точну копію поточного об’єкту.
- Публічний метод Update, який оновлює значення всіх параметрів об’єкту тривимірного простору та залежності між ними.
- Публічний метод IsIntersectionWith, який приймає у якості параметра об’єкт класу Object_3D і виконує перевірку на колізію двох об’єктів. Тобто перевіряє чи входять вони один в одного чи ні і в залежності від результату повертає логічне значення.
- Публічний метод IsHaveInside, який приймає у якості параметра об’єкт класу Dot_3D і виконує перевірку на те, чи належить точка зоні об'єкта. У залежності від результату перевірки метод повертає відповідне логічне значення.
- Публічний метод IsTouchWith, який приймає у якості параметрів об’єкт класу Dot_3D, цілочисельну змінну і виконує перевірку на те, чи належить точка полігону індекс якого дорівнює прийнятому значенню. Результатом перевірки є логічне значення, яке і повертається. В залежності від результату це true, або false.

- Методи для різноманітних переміщень об’єкту у просторі. Кожен метод викликає відповідний метод для кожного об’єкту класу Dot_3D зі списку вершин об’єкта, а після виконання своєї основної задачі, викликає метод Update для оновлення параметрів тривимірного об’єкта:
    - Публічний метод SetPosition, який приймає у якості параметру об’єкт класу Dot_3D і переміщує фігуру на вектор, який утворюється у результаті віднімання прийнятої точки, від точки центру тривимірного об’єкту.
    - Публічний метод Translation, який у якості параметра приймає об’єкт класу Vector_3D і переміщує об’єкт у заданому напрямі.
    - Публічні методи RotateXByDegrees, RotateYByDegrees і RotateZByDegrees, які у якості параметрів приймають число з плаваючою точкою, об’єкт класу Vector_3D і виконує обертання по відповідній до назв методів осі навколо точки, що задана вектором.
- Статичні методи для створення геометричних фігур:
    - Публічний метод Cube, який приймає в якості параметра одну змінну типу числа з плаваючою точкою і на базі цієї інформації створює тривимірний об’єкт, що відповідає розумінню куба.
    - Публічний метод Rectangle, який приймає в якості параметрів три змінні типу числа з плаваючою точкою і на їх базі створює геометричний тривимірний об’єкт, що відповідає розумінню прямокутника.
    - Публічний метод Pyramid, який приймає в якості параметрів три змінні типу числа з плаваючою точкою і на їх базі створює геометричний тривимірний об’єкт, що відповідає розумінню піраміди.

Скріншот реалізації методів



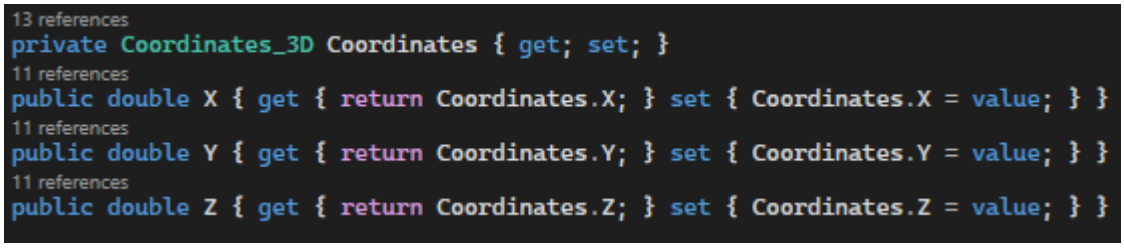
Створений клас Vector_3D

Клас Vector_3D містить в собі: чотири поля, два конструктори, чотири визначення для операторів і п'ять методів.

Поля класу Vector_3D:

- Приватне поле Coordinates, яке оголошує собою об’єкт класу Coordinates_3D, що відповідає за параметри направленості вектору у тривимірній системі координат.
- Відкриті поля X, Y і Z звертаються до об’єкту класу Coordinates_3D і повертають, або ж задають у ньому значення вектору.

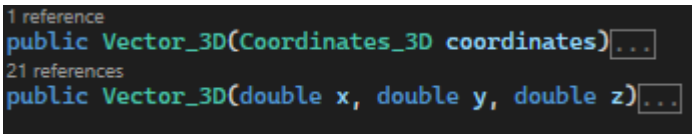
Скріншот реалізації полів



Конструктори класу Vector_3D:

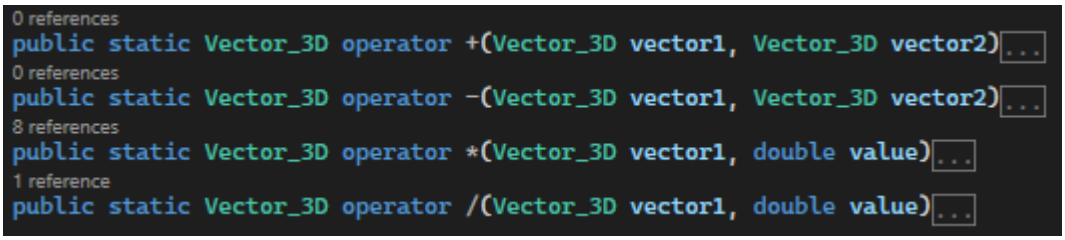
- Перший конструктор приймає об’єкт класу Coordinates_3D і звертається до першого конструктора класу Coordinates_3D, створюючи таким чином координати для розміщення точки у просторі.
- Другий конструктор приймає три параметри типу числа з плаваючою точкою і звертається до другого конструктора класу Coordinates_3D, створюючи координати направленості вектору у трьох вимірах.

Скріншот реалізації конструкторів



Визначення для операторів у класі Vector_3D:

- Оператор додавання, який дозволяє додавати одне до одного об'єкти класу, таким чином утворюючи новий вектор направлення.
- Оператор віднімання, який дозволяє віднімати один від одного об'єкти класу, таким чином утворюючи новий вектор направлення.
- Оператор множення, який дозволяє помножити об'єкт класу на значення числа з плаваючою точкою. За рахунок цієї операції можна змінити довжину вектора.
- Оператор ділення, який дозволяє поділити об'єкт класу на значення числа з плаваючою точкою і так само як і при множенні, змінити довжину вектора.

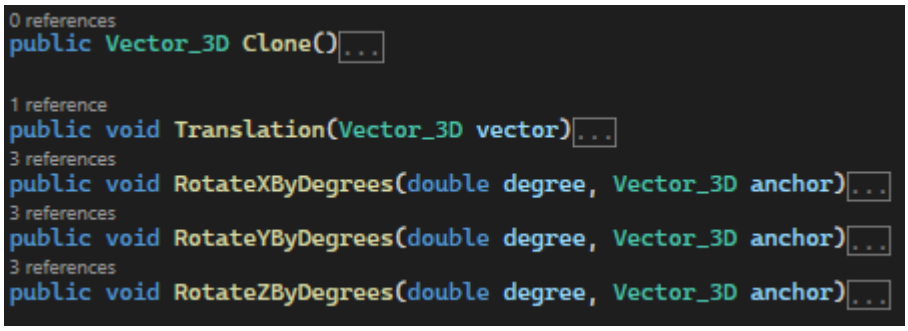


Методи класу Vector_3D:

- Публічний метод Clone, який призначений для копіювання об’єкту. Він повертає об’єкт з ідентичними параметрами, таким чином створюючи копію оригінального об’єкта.
- Методи для переміщення координат вектору:

- Публічний метод Translation, який в якості параметру приймає об’єкт класу Vector_3D. Він звертається до методу Translation класу Coordinates_3D для поля цього ж класу.
- Публічні методи RotateXByDegree, RotateYByDegree і RotateZByDegree. У якості параметрів, кожен з перелічених методів приймає змінну типу числа з плаваючою точкою, що відповідає за кут для оберту у градусах, а також об’єкт класу Vector_3D, який виступає у ролі якоря, навколо якого виконуватиметься обертання. Обертання відбувається по осі, відповідно до назви кожного метода.

Скріншот реалізації методів



Display_3D

Клас Display_3D містить в собі: дванадцять полів, один конструктор і тринадцять методів.

Поля класу Display_3D:

- Публічне поле DotsVisibility, яке оголошує та зберігає в собі логічну змінну, яка відповідає за те, чи видно вершини при відображенні, чи ні.
- Публічне поле Lines Visibility, яке оголошує та зберігає в собі логічну змінну, яка відповідає за те, чи видно користувачеві грані(лінії) тривимірних об’єктів, чи ні.
- Приватні поля VectorX, VectorY, VectorZ, які оголошують та зберігають в собі об’єкти класу Vector_3D, сукупність яких створює власну систему координат для переміщення камери у тривимірному просторі.

- Приватні поля MainPlane, LeftPlane, RightPlane, TopPlane і BottomPlane, які оголошують і зберігають у собі площини, що являють собою зону покриття камери, у тривимірній системі координат.
- Приватне поле ZoneAnchor, яке оголошує та зберігає об’єкт класу Dot_3D, який відповідає за точку на яку спирається камера для визначення векторів нормалей площин з яких вона утворена.
- Приватне поле RotationAnchor, яке оголошує та зберігає об’єкт класу Vector_3D, який відповідає за точку опору, навколо якої обертається камера.

Скріншот реалізації полів

```
private Vector_3D VectorX = new Vector_3D(10, 0, 0);
private Vector_3D VectorY = new Vector_3D(0, 10, 0);
private Vector_3D VectorZ = new Vector_3D(0, 0, 10);
public bool DotsVisibility = true;
public bool LinesVisibility = true;
private Dot_3D ZoneAnchor = new Dot_3D(0, 0, 1);
private Vector_3D RotationAnchor = new Vector_3D(0, 0, 0);
private Plane_3D MainPlane;
private Plane_3D LeftPlane;
private Plane_3D RightPlane;
private Plane_3D TopPlane;
private Plane_3D BottomPlane;
```

Конструктор класу Display_3D:

- Конструктор приймає в якості параметрів дві змінні типу числа з плаваючою точкою, які відповідають за висоту, та ширину камери, що створюватиметься.

Скріншот реалізації конструктора

```
2 references
public Display_3D(double width, double height)...
```

Методи класу Display_3D:

- Публічний метод Update, який оновлює і корегує всі параметри поточного об’єкту.
- Методи для переміщення камери у просторі:

    - Публічний метод Translation, який приймає у якості параметра об’єкт класу Vector_3D, який відповідає за вектор, на який буде перенесено камеру і всі супутні параметри.
    - Публічні методи TranslationX, TranslationY і TranslationZ, які у якості параметру приймають змінну типу числа з плаваючою точкою, яка відповідає за крок, на який буде перенесено камеру у напрямі відповідного до назви метода вектора внутрішньої системи координат камери.
    - Публічні методи RotateXByDegrees, RotateYByDegrees і RotateZByDegrees, які приймають у якості параметру змінну типу числа з плаваючою точкою, яка відповідає за кут, на який обертатиметься камера навколо точки опору.
- Публічний метод SetAnchor, який приймає у якості параметру об’єкт класу Dot_3D і замінює значення точки опору для обертання, на значення об’єкту, який був прийнятий.
- Публічний метод ShowImage, який приймає у якості параметрів об’єкт класу PictureBox і список об’єктів класу Object_3D. Метод призначений для відображення на екрані користувача візуальної проекції тривимірного простору, а саме того, як об’єкти розташовані між собою.
- Приватний метод DrawBorders, який в якості параметрів приймає об’єкт класу Graphics і дві змінні типу числа з плаваючою точкою, що відповідають за висоту та ширину камери. Метод малює рамку для зони відображення.
- Приватний метод DrawDot, який приймає об’єкт класу Bitmap, об’єкт класу Dot_3d і список об’єктів класу Object_3D. Цей метод є основою логіки відображення. В ньому розраховуються дані, на основі яких приймається рішення, малювати точку на екрані користувача або ж ні.
- Публічний метод GetCameraPosition, який повертає значення позиції камери у просторі по трьом координатам X, Y і Z.

Скріншот реалізації методів

```
3 references
public void Translation(Vector_3D vector)...
1 reference
public void TranslationX(double direction)...
1 reference
public void TranslationY(double direction)...
3 references
public void TranslationZ(double direction)...
2 references
public void RotateXByDegrees(double degree)...
2 references
public void RotateYByDegrees(double degree)...
1 reference
public void RotateZByDegrees(double degree)...

1 reference
public void Update()...
2 references
public void SetAnchor(Dot_3D dot)...

2 references
public void ShowImage(PictureBox PB, List<Object_3D> obj)...
1 reference
public void DrawBorders(Graphics graphics, float height, float width)...
2 references
private void DrawDot(Bitmap display, Dot_3D dot, List<Object_3D> obj)...

3 references
public Coordinates_3D GetCameraPosition()...
```

3.2. Опис програми

3.2.1. Основні функції

Функції основного вікна (див. рис. 3.2.2.1):

Вікно призначене для спостерігання виконаня запропонованого у роботі алгоритму. Воно містить інтерфейс для взаємодії з камерою, яка дозволяє проекціювати тривимірний простір на двовимірну площину. Також вікно містить інтрфейс для виклику інших вікон, які відповідають за створення об'єктів (див. рис. 3.2.2.2) та їх додавання до сцени (див. рис. 3.2.2.3).

Функції вікна створення нового об’єкту (див. рис. 3.2.2.2):

Вікно признаачене для створення нового прямокутного паралілепіпеда, який являє собою хіт-бокс об’єкта у тривимірному просторі

Значення, які можна задати параметрам прямокутника лежать від 10 до 50. Це зроблено для більшої зручності використання програми. У загальному випадку обмеження не накладаються.

Функції вікна для додавання обраного об’єкту до сцени (див. рис. 3.2.2.3):

Вікно призначене для додавання до сцени нового об'єкту обраного зі списку створених прямокутних паралілепіпедів.

При додаванні об'єкта починається проходження по запропонованому у роботі алгоритму щільного об'єкту. Результатом виконання алгоритму є позиція, яка є максимально наближеною до заданої точки роміщеня у просторі, у якій і розміщюється об'єкт. У разі якщо задана позиція була вільною, об'єкт додається до сцени у ній самій, у протилежному випадку, відбувається пошук коректної позиції.

3.2.2. Інтерфейс

Для створення програми було використане сереедовище Win Forms та його елемменти.

Створена програмний засіб має три вікна:

Основне(головне):

Рис. 3.2.2.1. Основне вікно

Елементи:

- PictureBox, який знаходиться у лівому верхньому куті форми, відповідає за відображення тривимірної сцени у вигляді двовимірного зображення користувачеві.

- ListView, який знаходиться у правому верхньому куті та правій стороні форми, відповідає за відображення зображень елементів списку створених об’єктів.

Кнопки:

- “Reset” відповідає за очищення сцени. Тобто за видалення всіх елементів за списку елементів об’єктів сцени.
- “Add Object” відповідає за виклик вікна для створення ногового хіт-боксу, тобто нового об’єкту (див. рис. 3.2.2.2).
- “Delete Object” відповідає за видалення обраного об’єкту зі списку створених об’єктів.

Стрілочки навпроти написів X Move, Y Move та Z Move відповідають за пересування камери у просторі.

Стрілочки навпроти лейблів X Rotation, Y Rotation та Z Rotation відповідають за обертання камери навколо точки опору.

Вікно для створення нових хіт-боксів, тобто нових об’єктів:

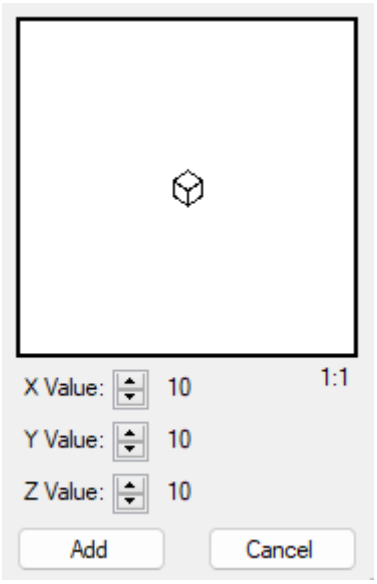


Рис. 3.2.2.2. Вікно для створення та додавання нового об’єкту

Елемент PictureBox, який розташований посередині угорі форми, відповідає за відображення створюваного об’єкта користувачеві.

Кнопки:

- “Add” відповідає за створення нового об'єкту.
- “Cancel” відповідає за вихід із вікна.

Стрілочки навпроти написів X Value, Y Value та Z Value відповідають за збільшення та зменшення значення по відповідній осі.

Напис під правим нижнім кутом елементу PictureBox, відповідає за масштаб зображення.

Вікно додавання обраного об’єкту до сцени:

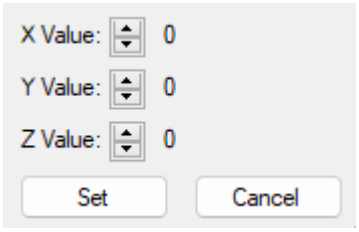


Рис. 3.2.2.3. Вікно додавання обраного об’єкту до сцени.

Кнопки:

- “Set” відповідає за створення точки опору для розташування нового об'єкту до додавання його до сцени.
- “Cancel” відповідає за вихід із вікна.

Стрілочки навпроти написів X Value, Y Value та Z Value відповідають за збільшення та зменшення значення по відповідній осі. Число праворуч від них надає інформацію про те, де буде розташовано новий об’єкт.

3.3. Висновок до третього розділу

Глибоко розглянуто платформу .NET та мову програмування C#.

Створено систему взаємопов'язаних класів, яка формує об'єкт на базі якої формуються об'єкти тривимірного простору.

Розроблений клас Coordinates_3D надає можливість взаємодії з координатами у просторі. А саме, їх переміщення та обертання.

Розроблений клас Dot_3D дозволяє створювати базові елемент фігури, а саме, її вершини.

Розроблений клас Line_3D дозволяє: створювати лінії та грані, розраховувати параметричне рівняння лінії, а також створювати список з точок для відображення їх користувачеві.

Розроблений клас Plane_3D дозволяє: створювати площини, визначати їх рівняння, а також корегувати вектор нормалі.

Розроблений клас Object_3D дозволяє: створювати об’єкти тривимірного простору, знаходити їх центр, розраховувати їх параметри.

Розроблений клас Display_3D дозволяє: створювати камеру, яка досить чітко демонструє користувачеві тривимірне середовище на двовимірній площині, тобто на екрані.

Створено програмний засіб на базі розробленого алгоритму для пошуку позиції та щільного розміщення об'єктів тривимірного простору за їх хіт-боксами. Для цього використовувалась платформа .NET та її елемент Win Forms, а мовою програмування була обрана C#.

Наведено опис функціоналу та інтерфейсу створеного програмного засобу, а також детально розібраний процес його експлуатації.

Висновок

Розміщення об'єктів тривимірного простору є досить розповсюдженою задачею яка виникає під час широкого спектру задач. Серед цього спектру задач є моделювання, створення комп'ютерної графіки та ігор, сам ігровоий процес, планування при будівництві або ж при плануванні дизайну і т.д.

У роботі було досліджено:

Методи моделювання положення об’єктів у просторі.

Алгоритм Робертса для відкидання прихованих поверхонь опуклих об’єктів тривимірного простору, а також досліджено алгоритм Z-буфера для схожої задачі.

У результаті дослідження методів за засобів комп'ютерної графіки був розроблений та запропонований алгоритм для щільного розташування об'єктів у просторі. Який має допомогти у вирішенні задач планування та моделювання у тривимірному середовищі.

У роботі було описано:

Структуру взаємопов’язаних класів, яка є необхідною для створення зрозумілих для обчислювальної машини об’єктів у тривимірному просторі.

Проблему розміщення об’єкту у просторі та запропоновано алгоритм розміщення об’єкта у просторі сцени, на основі його хіт-боксу.

Систему взаємопов'язаних класів, яка формує об'єкт на базі якої формуються об'єкти тривимірного простору.

Також було створено програмний засіб на базі розробленого алгоритму для пошуку позиції та щільного розміщення об'єктів тривимірного простору за їх хіт-боксами. Для цього використовувалась платформа .NET та її елемент Win Forms, а мовою програмування була обрана C#.

Перелік використаної літератури

1) Большаков, В.П. Інженерна і комп'ютерна графіка. Теоретичний курс і тестові завдання / В.П. Большаков. - СПб .: BHV, 2016. - 384 с.
2) Голованов, Д.В. Комп'ютерна нотна графіка: Навчальний посібник / Д.В. Голованов, А.В. Кунгур. - СПб .: Планета Музики, 2018. - 192 с.
3) Бутакова, Н.Г. Комп'ютерна графіка / Н.Г. Бутакова. - М .: МГИУ, 2018. - 216 с.
4) Максимова, І.А. Прийоми образотворчої мови в сучасній архітектурі (ручна і комп'ютерна графіка): Навчальний посібник / І.А. Максимова, А.Е. Винокурова, А.В. Пивоварова. - М .: Инфра-М, 2018. - 264 с.
5) Капранова, М.Н. Macromedia Flash MX. Комп'ютерна графіка та анімація / М.Н. Капранова. - М .: Солон-прес, 2016. - 96 с.
6) Вигодський М.Я. Довідник з вищої математики. - М .: АСТ, 2018. - 991 с.
7) Гусак А.А., Гусак Г.М., Бричкова Е.А. Довідник з вищої математики. - Мінськ. ТетраСистемс, 2015. - 640 с.
8) Міносці В.Б. Курс вищої математики. Частина 2.- М .: 2017. - 517 с
9) Аверін, В.Н. Комп'ютерна інженерна графіка / В.Н. Аверін. - М .: Academia, 2018. - 174 с.
10) Нємцова, Т.І. Практикум з інформатики. Комп'ютерна графіка та Web-дизайн. Практикум: Навчальний посібник / Т.І. Нємцова. - М .: ИД ФОРУМ, НДЦ Инфра-М, 2018. - 288 с.
11) Дегтярьов, В.М. Інженерна і комп'ютерна графіка: Підручник / В.М. Дегтярьов. - М .: Академія, 2017. - 160 с.
12) Нікулін, Е.А. Комп'ютерна графіка. Моделі і алгоритми: Навчальний посібник / Е.А. Нікулін. - СПб .: Лань, 2018. - 708 с.
13) Логіновскій, А.Н. Інженерна 3D-комп'ютерна графіка: Навчальний посібник для бакалаврів / О.М. Логіновскій. - М .: Юрайт, 2014. - 464 с.
14) Строзотт, Т. нефотореалістічниє комп'ютерна графіка: моделювання, рендеринг, анімація / Т. Строзотт. - М .: Кудіц-образ, 2016. - 416 с.

15) Аверін, В.Н. Комп'ютерна інженерна графіка: Навчальний посібник / В.М. Аверін. - М .: Academia, 2018. - 352 с.

16) Чию, Дж. Аналітична теорія S-матриці / Дж. Чию. - М .: 2018. - 724 с.

17) Миронов, Д. Комп'ютерна графіка в дизайні / Д. Миронов. - СПб .: BHV, 2017. - 560 с.