

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»  
Кафедра автоматики та управління в технічних системах

## Курсова робота

### Редактор багатовимірних фігур

З дисципліни «Об'єктно-орієнтоване програмування-2. Поліморфізм об'єктно-орієнтованих моделей»

Керівник :  
ст. викладач  
Хмелюк В.С.

Виконав :  
ст. Мішин О.В.  
зал. книжка № ІТ-6112  
гр. ІТ-61

«Допущений до захисту»

\_\_\_\_\_  
(Особистий підпис керівника)  
" \_\_\_\_ " \_\_\_\_\_ 2018р.

\_\_\_\_\_  
(Особистий підпис виконавця)  
" \_\_\_\_ " \_\_\_\_\_ 2018р.

Захищений з оцінкою

\_\_\_\_\_  
(оцінка)

Члени комісії:

\_\_\_\_\_  
(Особистий підпис)

\_\_\_\_\_  
(Розшифровка підпису)

\_\_\_\_\_  
(Особистий підпис)

\_\_\_\_\_  
(Розшифровка підпису)

## АНОТАЦІЯ

В курсовій роботі розглядається питання багатокористувацької розробки багатовимірних фігур. Розглянуто способи збереження цих фігур для надання можливості доступу різних користувачів. Описано спосіб надання різних рівнів доступу до створених об'єктів і збереження їх в базі даних.

Роботу реалізовано мовою програмування C# в середовищі MS Visual Studio. Документація до програми виконана на 50 сторінках, містить 2 додатки і 16 рисунків.

## ЗМІСТ

ВСТУП.....	6
1 ПОСТАНОВКА ЗАДАЧІ.....	7
2 МОДЕЛЬ ТА СТРУКТУРА ПРОГРАМИ.....	8
2.1 Огляд існуючих рішень .....	8
2.3 Теоретичні матеріали .....	11
2.3.1 Основні поняття: простір, вигляд, трансформація .....	11
2.3.2 Засоби зображення тіл на площині: перспектива, переріз .....	12
2.3.4 Приклади фігур різних розмірностей.....	14
2.4 Загальна структура програми.....	17
3 ПЕРЕЛІК І ПРИЗНАЧЕННЯ РЕЖИМІВ ТА СТРУКТУРА ДІАЛОГУ .....	19
3.1 Режими роботи програми .....	19
3.1.1 Головне вікно, до відкриття простору .....	19
3.1.2 Головне вікно, відкрито простір.....	19
3.1.3 Вікно авторизації.....	19
3.1.4 Вікно підключення до простору .....	20
3.1.5 Вікно створення простору .....	20
3.1.6 Вікно адміністратора.....	20
3.2 Допоміжна інформація.....	20
4 СТРУКТУРА ДАНИХ ТА РЕСУРСІВ ПРОГРАМИ.....	21
4.1 Схема сутностей .....	21
4.2 Схема робочих класів.....	22
4.3 Статичні і допоміжні класи .....	22
5 ОПИС ПРОГРАМИ.....	24
5.1 Архітектура .....	24
5.2 Основні методи.....	24
5.3 Вхідні та вихідні дані .....	24
5.3.1 Простір.....	24
5.3.2 Трансформація .....	25

6 КЕРІВНИЦТВО АДМІНІСТРАТОРА .....	26
6.1 Встановлення програми .....	26
6.2 Підключення до бази даних .....	26
7 КЕРІВНИЦТВО КОРИСТУВАЧА .....	28
7.1 Початок роботи.....	28
7.2 Вигляди та простори .....	30
7.3 Редагування зображення, вигляду і простору .....	33
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ .....	37
ДОДАТОК А (Текст програми «MultiDimEditor») .....	38
ДОДАТОК Б (Детальна діаграма класів).....	50

## ВСТУП

Математичні формули і методи дозволяють розглядати простори з розмірністю вище трьох. Хоча оточуючий нас простір є тривимірним, розв'язання різних задач сучасної науки проводить до конструкцій, які можна зображати як тіла в деякому просторі з вищими розмірностями. Це може бути дослідження фазового простору системи, побудова специфічних графів та інше. Але людині, що все своє життя знаходиться в тривимірному просторі, буває складно уявити такі конструкції чи оперувати ними. Тому постає задача створення середовища для зручного створення і перегляду таких об'єктів з врахуванням особливостей багатовимірного простору.

Така програма має дозволяти користувачу оглядати вже створені фігури чи створювати нові якомога зручнішим способом. Також вона має давати можливість користувачу дозволяти чи блокувати можливість іншим користувачам переглядати та редагувати створені ним фігури.

При організації багатокористувацького доступу постає задача надання можливості керування користувачами: створення і видалення користувачів, надання різним користувачам різних прав доступу та інші. Важливо, щоб користувачі, що створили певний об'єкт, могли самі зазначати, які можливості для роботи з їх об'єктами яким користувачам слід надати, не звертаючись до адміністратора системи.

Оскільки основна робота в програмі виконується саме з зображенням фігур, постає також проблема надання користувачам достатньої кількості інструментів налаштування зовнішнього вигляду таким чином, щоб не перевантажити інтерфейс.

## 1 ПОСТАНОВКА ЗАДАЧІ

Основною задачею є розробка програми, яка дозволяє створювати, переглядати і редагувати фігури довільної розмірності.

Повинна існувати можливість створення якомога більш різноманітних фігур, без обмеження на опуклість, зв'язність, замкненість та інше.

Створені фігури мають зберігатись таким чином, щоб забезпечити можливість керованого багатокористувацького доступу.

Користувач, який створив певну фігуру, повинен мати можливість надавати іншим користувачам права переглядати її, чи переглядати і редагувати, або навпаки блокувати ці права.

При редагуванні чи огляді фігури повинна бути можливість керувати налаштуваннями вигляду та змінювати положення «камери» - точки, з якої ведеться огляд. Мають бути реалізовані операції повороту, зміни масштабу і зсуву як для камери, так і для самої фігури чи її частин.

При роботі з фігурою має бути чіткий поділ можливих дій на дії, що змінюють саму фігуру, та дії, що змінюють вигляд фігури, не змінюючи її саму.

В системі користувачів повинен існувати користувач з правами адміністратора, який може керувати рівнями доступу інших користувачів, а також створювати чи видаляти їх.

## 2 МОДЕЛЬ ТА СТРУКТУРА ПРОГРАМИ

### 2.1 Огляд існуючих рішень

На даний момент тема редагування багатовимірних тіл не є достатньо популярною. Хоча за допомогою сучасних математичних пакетів і можна побудувати необхідну проекцію на екран тіла, заданого математично, такі операції є достатньо складними. Як приклад саме спеціалізованих програм, що розв'язують схожу задачу, можна навести графічний пакет «Novo Spark Visualizer», інтерфейс якої наведений на рисунку 2.1.

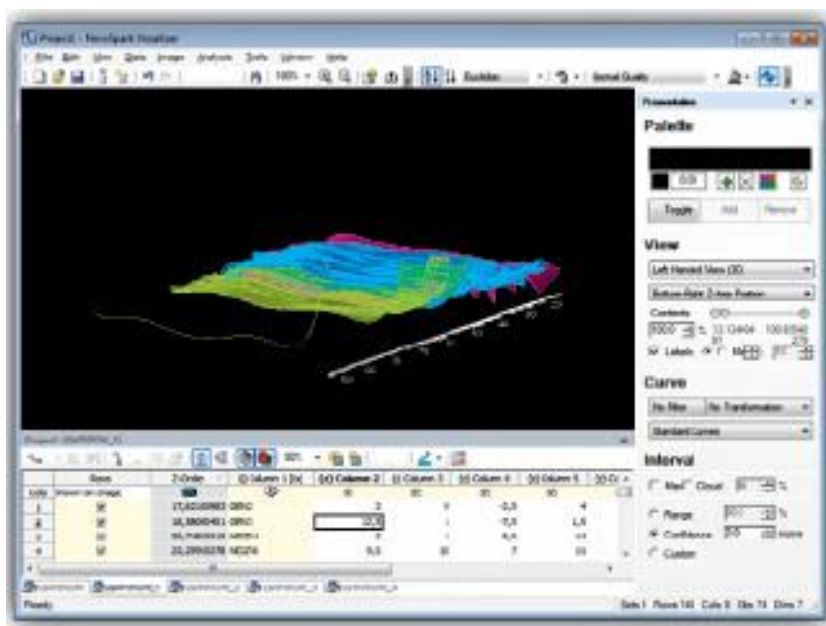


Рисунок 2.1 – Програма «Novo Spark Visualizer»

Цей графічний пакет призначений для наочного зображення багатовимірних систем даних. Дані вводяться з файлів чи баз даних, або заповнюються самим користувачем, далі вони візуалізуються на екрані, після чого їх можна аналізувати. Ця програма є достатньо корисною для своїх цілей, але її спеціалізація досить вузька. Більш детальний опис можна знайти в [1].

В наш час значну популярність має тривимірна комп'ютерна графіка. Через це достатньо поширеними і розвиненими є програми для створення і редагування саме тривимірних об'єктів. Однією із найбільш популярних таких програм є «Blender».

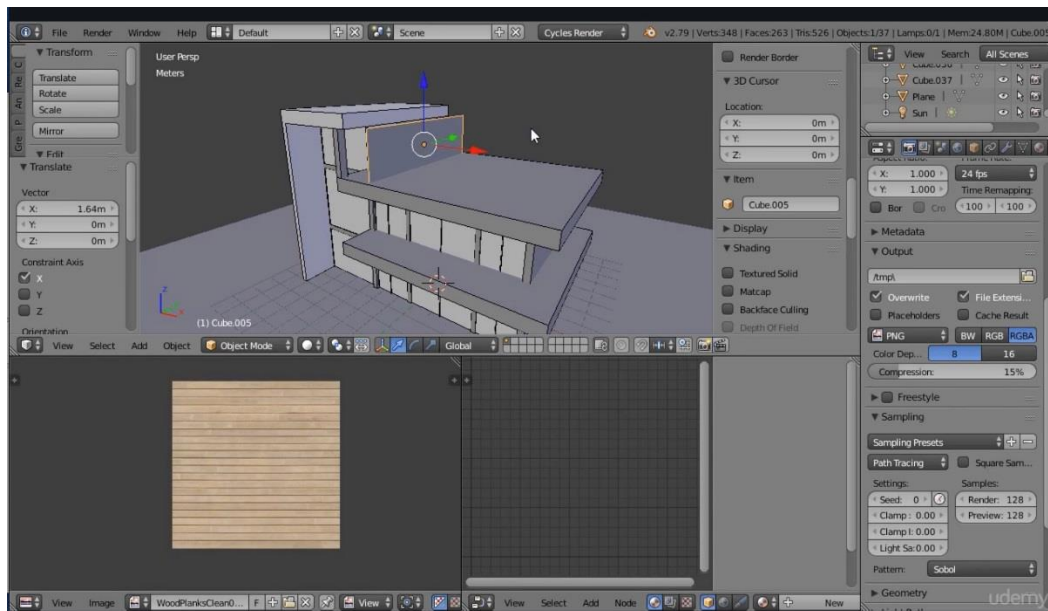


Рисунок 2.2 – Програма «Blender»

Ця програма є дуже потужним інструментом для створення тривимірної графіки, серед її можливостей є різні засоби моделювання, інструменти для створення анімацій, обробка освітлення, розрахунок частинок та динаміки зіткнень та інші.

З іншого боку, ця програма розрахована на спеціалістів з розробки тривимірної графіки, і потребує достатніх навичок для роботи з нею. Детальніше ознайомитись з цією програмою можна в [2].

Як приклад програми, розробленої саме для зручного огляду багатовимірних фігур, наведемо також програму «MultiDimentionalFigures».



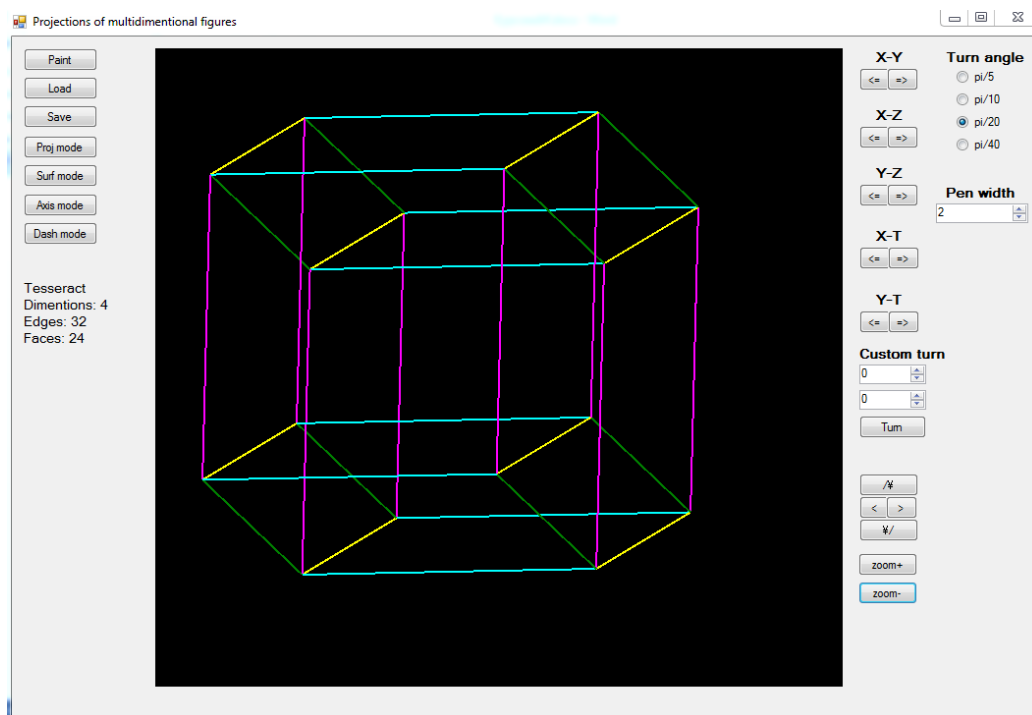


Рисунок 2.3 – Програма «MultiDimentionalFigures»

Ця програма дозволяє завантажувати багатовимірні фігури з файлів, в яких спеціальним чином записані ці фігури. Хоча можливості керування виглядом фігури достатньо потужні, саме створення файлу фігури відбувається вручну, а можливості редагування практично відсутні. Але невелика кількість можливостей робить роботу з програмою більш інтуїтивно зрозумілою

За результатами огляду програм – аналогів побудуємо порівняльну таблицю.

	Підтримка багатовимірності	Можливості редагування	Зручність створення	Простота інтерфейсу
Novo Spark Visualizer	+/-	+	+/-	-
Blender	-	+	+/-	-
MultiDimentionalFigures	+	-	+/-	+

Таблиця 2.1 – Порівняння програм – аналогів

Видно, що крім можливостей роботи з багатовимірними тілами, необхідно також забезпечити зручність створення фігур і не перевантажити при цьому інтерфейс.

## 2.3 Теоретичні матеріали

### 2.3.1 Основні поняття: простір, вигляд, трансформація

Ми розглядаємо  $N$ -вимірний евклідовий простір. В ньому кожна точка задається  $N$  координатами. Оскільки площина екрану є лише 2-вимірною, то безпосередньо побудувати на ній багатовимірне тіло неможливо, але можна побудувати проекцію такого тіла на площину екрану, як описано в [3], аналогічно тому, як це робиться в геометрії з 3-вимірними тілами.

Далі ми будемо розрізняти простір (сукупність точок, а також ребер та граней між ними) і вигляд цього простору (як саме ми оглядаємо цей простір, під яким кутом і з яким масштабом). Це дозволяє нам керувати виглядом, не змінюючи сам простір, а також це дозволить нам одночасно мати декілька незалежних виглядів.

У просторі зберігаються точки і ребра/грані, що поєднують їх. Виявляється досить зручним зберігати кожную точку як набір її координат, а ребра і грані задавати перерахуванням номерів точок, які їм належать. Таким чином будь-яка зміна координат точки автоматично змінить відповідні ребра і грані. Саме таким чином ми і будемо зберігати дані простору.

У вигляді зберігається послідовність дій, які треба виконати з простором перед його зображенням. Стандартний вигляд означає просте проектування всього простору на площину  $Oxy$  і зображення результату на екрані. Але цей вигляд можна змінити, додавши до нього трансформації. Є 3 різні трансформації, які можна виконати для зміни вигляду: поворот, масштабування і зсув.

Повороти в  $N$ -вимірному просторі задаються зазвичай площиною і кутом повороту. Але будь-який поворот можна розбити на елементарні повороти, кожний з яких проводиться в одній з координатних площин. Такі повороти змінюють одночасно лише 2 координати (стандотною матрицею повороту), ніяк не впливаючи на всі інші координати, тому їх можна задавати просто вказанням координатної

площини(тобто двох координатних вісей), напрямком та величиною кута повороту. Детальніше про повороти в багатовимірному просторі див. [4].

В даній роботі трансформація-поворот реалізована за допомогою матриці повороту.

Масштабування – це збільшення чи зменшення координат точок в певну кількість разів, що рівносильно витягненню чи зтисненню простору. Кожна координата множиться на показник розтягу, який може бути різним для різних координат. Ця трансформація також реалізована через множення на матрицю.

Зсув – збільшення чи зменшення всіх координат на певне число, що рівносильно зсуву початку координат. Ця трансформація реалізована через додавання вектора зсуву до всіх векторів точок.

Поворот і масштабування є матричними трансформаціями. Тому сумарна трансформація, що є результатом застосування декількох поворотів і масштабувань, також є матричною трансформацією, і її матриця – добуток матриць цих поворотів і масштабувань. На відміну від них, зсув не є матричною трансформацією, оскільки він змінює початок координат. Але будь-яку послідовність матричних трансформацій і зсувів можна звести до одної «подвійної» трансформації: послідовного застосування одної матричної трансформації і одного зсуву. Саме тому в даній роботі для запису/зчитування трансформацій використовуються саме подвійні трансформації.

### 2.3.2 Засоби зображення тіл на площині: перспектива, переріз

Крім простого проектування є і інші методи зображення багатовимірних тіл на площині. В даній роботі розглянуто також перспективне проектування та побудова перерізу.

Перспектива – зображення об'єктів на площині так, як їх би бачило око, розміщене в певній точці. Оскільки око помічає не лінійні розміри об'єктів, а кутові, більш віддалені об'єкти виглядають зменшеними. А об'єкти, що знаходяться позаду ока чи поза полем зору, зовсім не видимі. В даній роботі при перспективному

проектуванні всі об'єкти, що знаходяться позаду площини проектування (Оху), не зображуються зовсім, а всі інші зображуються з множенням координат  $X$ ,  $Y$  кожної точки на коефіцієнт, обернено пропорційний відстані до цієї точки. При цьому залишається проблема побудови ребер та граней, які частково знаходяться перед площиною і позаду її. Для ребра АВ, у якого точка В знаходиться поза площиною, ця проблема розв'язується знаходженням точки поділу С на ребрі, яка знаходиться перед площиною, але поза полем зору, і тим самим ділить ребро на 2 частини: повністю невидима частина СВ, і частина АС, яка повністю знаходиться перед площиною і може бути зображена.

Переріз – множина точок, що належать одночасно певному тілу та площині перерізу. Зазвичай переріз дозволяє більш детально зобразити внутрішню структуру об'єкта. Побудова перерізу зазвичай є складною задачею, одним з прикладів є відома сукупність конічних перерізів [5]. Але в даній роботі розглядаються лише фігури, що складаються з скінченної кількості точок, прямих ребер між ними і плоских граней між ребрами. Через це задача побудови перерізу значно спрощується. Переріз завжди буде мати вигляд багатокутника (чи сукупністю багатокутників), вершинами якого будуть точки перетину ребер фігури з площиною перерізу, або самі вершини фігури, якщо вони лежать на площині (надалі ребра чи вершини основної фігури, які дають в результаті вершини багатокутника-перерізу, будемо називати первісними цієї вершини). Дві вершини будуть з'єднані лінією в тому випадку, коли відповідні первісні цих вершин лежать на одній грані.

Перетворення, що викликані перспективою чи побудовою перерізу, завжди виконуються після всіх трансформацій камери, і одночасно може бути активне тільки одне таке перетворення.

### 2.3.3 Властивості багатовимірних фігур

Крім операцій над камерою, також можливо проводити операції з самою фігурою чи деякими її точками. Ці операції включають в себе вже розглянуті

трансформації, створення чи видалення нових ребер чи граней, та додавання нових точок.

Оскільки ніяких вимог на зв'язність, замкненість чи опуклість фігур не вводиться, ми можемо розглядати практично будь-яку множину точок і будь-які пари цих точок можуть бути поєднані в ребро. Насправді ж не кожну пару точок можна з'єднати ребром. Так, наприклад у куба не існує ребра між точкою і протилежною до неї. З математичної точки зору ребро існує між двома точками тоді і тільки тоді, коли вони належать двом різним граням, які перетинаються між собою по лінії, що проходить через них.

Грань обов'язково має включати лише точки, що лежать на одній площині. Ці точки на цій площині поєднані між собою в багатокутник, який і є самою гранню. Але в даній програмі це обмеження відсутнє, оскільки перевірку точної належності точок одній площині майже неможливо виконати в умовах того, що дійсні числа при збереженні і виконанні операцій втрачають точність.

### 2.3.4 Приклади фігур різних розмірностей

#### N-вимірний куб

Назвемо точку початку координат 0-вимірним одиничним кубом. Далі будемо виконувати наступну послідовність дій, яка переводить (N-1)-вимірний куб в N-вимірний:

1. Додамо N-й вимір в простір.
2. Всім точкам даного кубу поставимо нову, N-ту координату 0.
3. Для кожної точки даного кубу з координатами  $(\dots, 0)$  створимо парну їй точку з координатами  $(\dots, 1)$  і з'єднаємо їх ребром.
4. Для кожної пари точок даного кубу, з'єднаних ребром, поєднаємо ребром парні їм точки.

Отримана фігура є кубом розмірності на 1 вище за попередню.

Застосовуючи це перетворення до початкового 0-вимірного кубу(точки) отримаємо поступово 1-вимірний куб(відрізок), 2-вимірний куб(квадрат), та 3-вимірний куб(куб). Збільшивши розмірність 3-вимірного кубу, отримаємо 4-вимірний куб, або тесеракт, і далі можна продовжувати до нескінченності.

N-вимірний куб має  $2^N$  вершин. Всі ребра одиничного N-вимірного кубу мають довжину 1, будь-які 2 ребра паралельні, схрещуються під прямим кутом або перетинаються під прямим кутом, будь-які 2 грані паралельні або перетинаються під прямим кутом, всі координати всіх точок приймають лише значення 0 та 1.

### N-вимірний симплекс

Симплекс є узагальненням трикутника і тетраедра. Назвемо точку початку координат 0-вимірним одиничним симплексом. Далі будемо виконувати наступну послідовність дій, яка переводить (N-1)-вимірний симплекс в N-вимірний:

1. Додамо N-й вимір в простір.
2. Всім точкам даного симплексу поставимо нову, N-ту координату 0.
3. Знайдемо точку, віддалену від всіх точок даного симплексу на відстань 1.  
Таких точок завжди дві, і вони відрізняються знаком N-ї координати.
4. Поєднаємо ребрами знайдену точку зі всіма точками даного симплексу.

Застосовуючи це перетворення до 0-вимірного симплексу(точки), отримаємо відрізок, трикутник і тетраедр.

В N-вимірному симплексі N+1 вершина, будь-які 2 вершини одиничного симплексу поєднані ребром довжини 1, будь-які 3 точки лежать на грані, всі грані є правильними трикутниками.

### N-вимірна сфера

Сфера в N-вимірному просторі – це сукупність точок, рівновіддалених від певної точки(центру). Переріз сфери площиною меншої розмірності дасть сферу цієї розмірності і радіуса, що не більший за радіус початкової сфери, або порожню множину точок.

В даній програмі можна побудувати будь-яку фігуру, яка складається з точок, ребер, що є відрізками прямих, і граней, що є плоскими багатокутниками. Побудова кривих ліній і викривлених площин не передбачена.

## 2.4 Загальна структура програми

Розглянемо перелік дій, які має виконувати програма для виконання основних функцій.

Алгоритм авторизації:

- зчитати дані для входу (ім'я користувача і пароль);
- виконати запит в базу даних для перевірки коректності даних;
- якщо дані для входу підтверджені і немає даних про те, що цей користувач вже авторизований, авторизувати користувача.
- вивести користувачу дії, які він може виконати (в залежності від його прав) чи вивести повідомлення про помилку.

Алгоритм відкриття вигляду:

- виконати запит в базу даних для зчитування простору;
- розкодувати рядок із вмістом простору і отримати набори точок, ребер і граней;
- розкодувати рядок трансформації і отримати подвійну трансформацію;
- застосувати трансформацію до простору і зобразити на екрані;
- отримати рівень доступу даного вигляду;
- вивести користувачу дії, які він може виконати (в залежності від рівня доступу вигляду).

Алгоритм підключення до нового простору:

- виконати запит до бази даних і перевірити можливість підключення і необхідність пароля;
- вивести користувачу можливі режими входу і повідомити про необхідність паролю до кожного з них, якщо вона присутня;
- при необхідності пароля отримати пароль від користувача і перевірити його коректність запитом до бази даних;
- отримати від користувача назву створеного вигляду



- створити вигляд за замовченням з відповідним рівнем доступу і підключитись до нього.

Алгоритм створення нового простору:

- отримати від користувача назву простору, його розмірність і правила доступу;
- виконати запит до бази даних, який додасть туди новий простір з відповідними параметрами;
- підключитись до цього простору.

Алгоритм побудови зображення тіла:

- будуємо точки тіла (з врахуванням трансформації вигляду і правил зображення);
- будуємо всі ребра між відповідними точками;
- будуємо всі грані;
- зображуємо координатні осі.

## 3 ПЕРЕЛІК І ПРИЗНАЧЕННЯ РЕЖИМІВ ТА СТРУКТУРА ДІАЛОГУ

### 3.1 Режими роботи програми

#### 3.1.1 Головне вікно, до відкриття простору

Після того, як була запущена програма, вона очікує від користувача авторизацію. При успішній авторизації користувачу надається список можливих дій. Для користувача з рівнем доступу 1(звичайний користувач) надається список виглядів, що належать цьому користувачу, і список просторів, до яких можливо підключитись. Для користувача з рівнем доступу 2(автор) чи 3(адміністратор) також надається можливість створювати власний простір. Кнопки редагування фігур чи керування зображенням при цьому мають бути заблоковані.

#### 3.1.2 Головне вікно, відкрито простір

Аналогічно попередньому пункту, але розблоковано кнопки керування зображенням, в залежності від рівня доступу вигляду. Для вигляду рівня 0(заблокований перегляд) розблоковано лише кнопки зміни режиму зображення. Для вигляду рівня 1(вільний перегляд) доступні також кнопки керування виглядом, а для вигляду рівня 2(редагування) доступні також кнопки редагування фігури. При виході з простору внесені зміни мають зберігатися.

#### 3.1.3 Вікно авторизації

У цьому вікні зчитується інформація для авторизації, перевіряється її правильність і, у випадку успішної авторизації, дані про користувача повертаються в головне вікно. Крім цього, в цьому вікні може зареєструватись новий користувач, і, якщо його ім'я не зайняте, за замовченням він отримує рівень доступу 1(звичайний користувач).

### 3.1.4 Вікно підключення до простору

У цьому вікні зображуються варіанти підключення і необхідність вводу пароля. У випадку успішного підключення, дані про створений вигляд повертаються в головне вікно.

### 3.1.5 Вікно створення простору

У цьому вікні зчитуються назва та розмірність нового простору, а також можливі варіанти підключення і паролі. Створений простір зберігається в базі даних.

### 3.1.6 Вікно адміністратора

У цьому вікні, яке відкривається автоматично при авторизації адміністратора, виводиться список користувачів, і для кожного користувача(крім адміністратора) надається можливість редагувати його рівень доступу чи видалити його.

## 3.2 Допоміжна інформація

Будь-які дії, пов'язані зі створенням нових користувачів, просторів чи виглядів, відразу заносять відповідні зміни до бази даних і оновлюють списки дій користувача.

Адміністратор не має додаткових можливостей перегляду чи редагування просторів, але він може видалити користувача, що видаляє всі його простори і вигляди.

Автор простору може підключатись до свого простору в будь-якому режимі без пароля.

Вигляд за замовченням зображує фігуру без трансформацій.

## 4 СТРУКТУРА ДАНИХ ТА РЕСУРСІВ ПРОГРАМИ

### 4.1 Схема сутностей

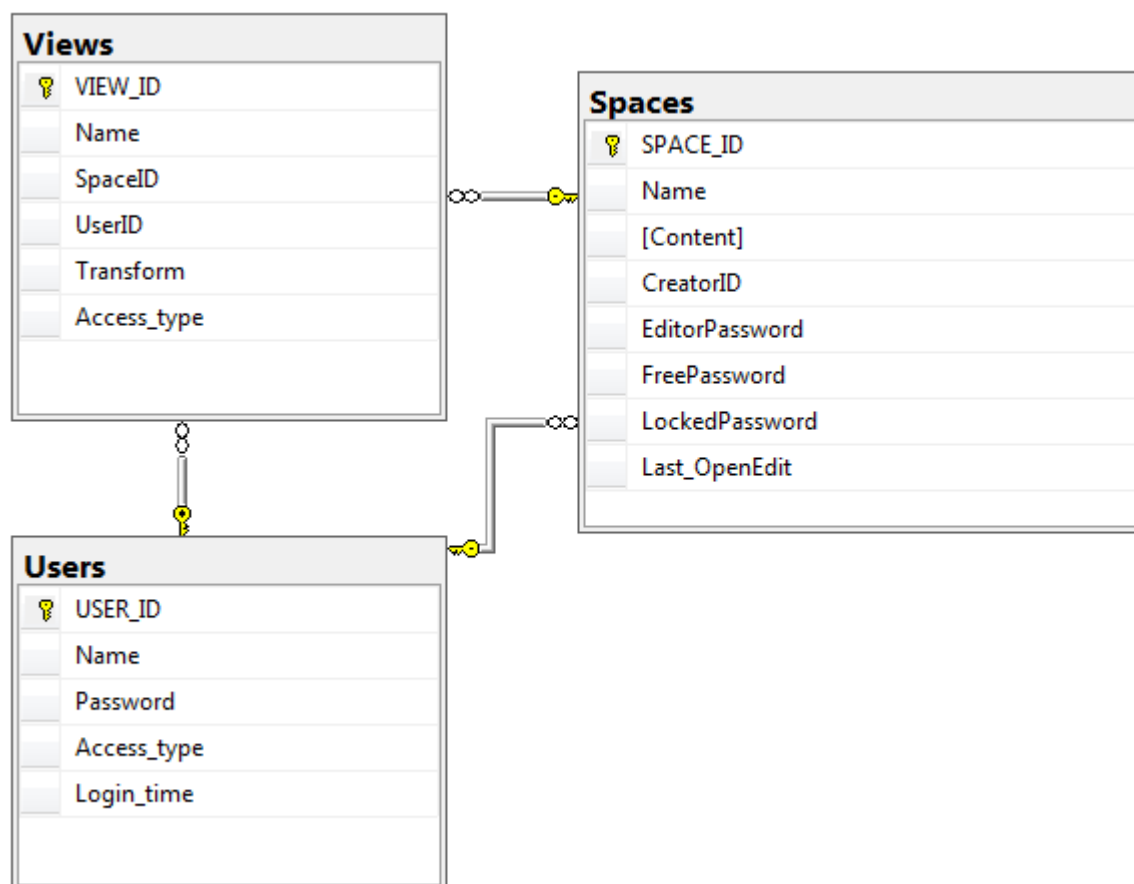


Рисунок 4.1 Схема класів основних сутностей

Клас Users відповідає за користувачів. У користувача є ім'я, пароль, тип доступу і час останньої авторизації.

Клас Spaces відповідає за простори. У простору є назва, вміст(рядок, який вміщує дані про власне простір – точки, ребра, грані), автор, паролі, і час останнього відкриття для редагування.

Клас Views відповідає за вигляди. У вигляду є назва, власник, простір, тип доступу і трансформація(рядок, який вміщує дані про власне вигляд – подвійну трансформацію).

## 4.2 Схема робочих класів

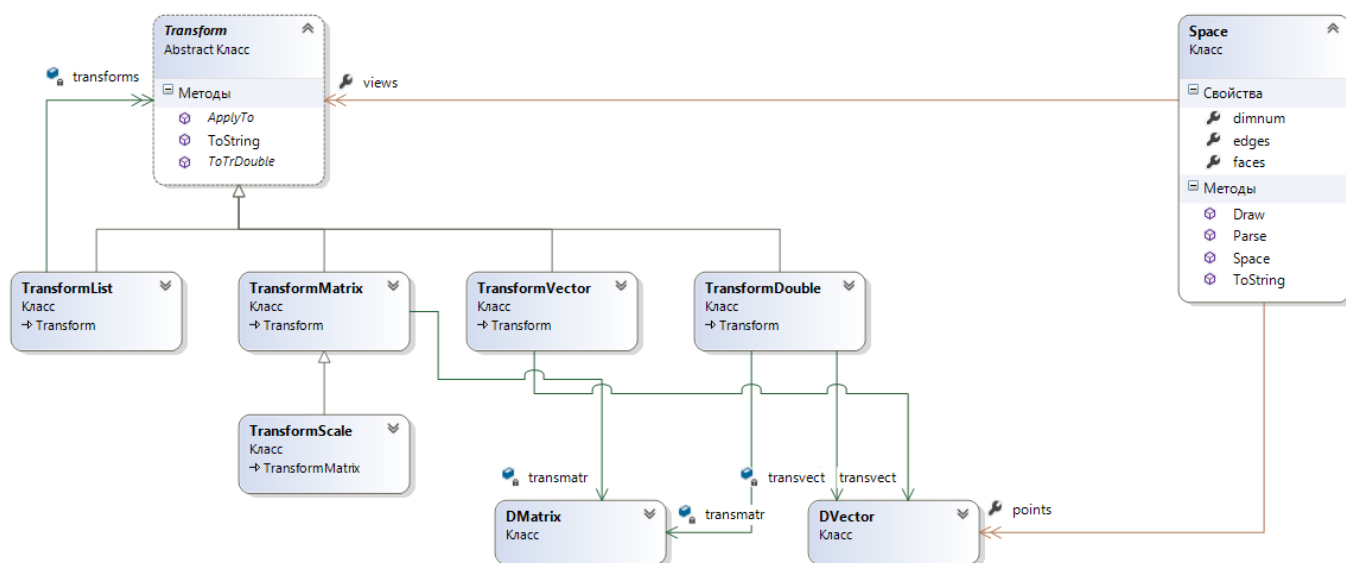


Рисунок 4.2 Схема робочих класів

- class DVector-клас, в якому зберігається точка (вектор): її координати, розмірність і методи роботи;
- class DMatrix-клас, в якому зберігається матриця: її координати, розмірність і методи роботи;
- class Edge і class Face – класи, в яких зберігаються відповідно грані і ребра;
- class Transform і похідні – класи, в яких зберігаються різні типи трансформацій;
- class Space – клас, в якому зберігається простір;

Нумерація точок починається з нуля.

Більш детальна діаграма розміщена в додатках.

## 4.3 Статичні і допоміжні класи

DBMethods – в цьому класі описані методи для операцій з базою даних, які виконуються в різних місцях програми, для уникнення дублювання;

DVectorConverter – в цьому класі описано перетворення DVector в Point, з врахуванням розташування початку координат в центрі рисувальної області і інверсії координати Y.

DrawingRules – клас, що відповідає за зберігання сукупності правил зображення. Об'єкт цього класу передається в метод Draw простору.

VisualHost – клас-контейнер для результуючого зображення.

## 5 ОПИС ПРОГРАМИ

### 5.1 Архітектура

В програмі можна виділити 3 частини: робота з базою даних, робота з простором і інтерфейсна частина. Робота з базою даних виконується через класи сутностей, створені за допомогою LINQ to SQL. Робота з простором виконується через методи простору. Інтерфейс виконано за допомогою технології WPF, і при роботі з ним використовуються такі особливості WPF як прив'язка даних, шаблони даних та автоматичні контейнери елементів інтерфейсу.

### 5.2 Основні методи

Функція Draw() – перерисовує все зображення, і викликається при відкритті вигляду, зміні правил зображення, виконанні трансформації вигляду чи редагуванні простору. При виконанні цієї функції використовуються змінні, що відповідають за режими рисування.

Функції SwitchControls() та SwitchDrawControls() вмикають чи вимикають елементи керування основного вікна, відповідно для рівня доступу користувача та вигляду.

Для кожної операції над простором чи виглядом призначена власна функція, яка належним чином виконує цю операцію.

### 5.3 Вхідні та вихідні дані

#### 5.3.1 Простір

При збереженні простору в базі даних він зберігається в спеціальному форматі. Цей формат виглядає наступним чином:

```
<dimnum>|<Npoints>|<point1.x> <point1.y>...|<pointN.x>...|<Nedges>|<ep11>
```

$\langle ep12 \rangle | \langle ep21 \rangle \langle ep22 \rangle | \dots | \langle Nfaces \rangle | \langle ef11 \rangle \langle ef12 \rangle \langle ef13 \rangle | \dots | \dots$

Тут  $dimnum$  – розмірність простору;

$Npoints$  – кількість точок;

$pointK.x$  – X координата K-ї точки;

$Nedges$  – кількість ребер;

$epK1$  і  $epK2$  – два номери точок, поєднаних ребром;

$Nfaces$  – кількість граней;

$ef11, ef12, ef13 \dots$  - номери точок, що лежать на грані.

Приклад: 3|4|0 0 0|0 0 4|0 1 0|1 0 0|6|0 1|0 2|0 3|1 2|1 3|2 3|4|0 1 2|0 1 3|0 2 3|1 2 3

### 5.3.2 Трансформація

Будь-яка трансформація перед записом до бази даних зводиться до подвійної трансформації. Вона зберігається в наступному форматі:

$\langle dimnum \rangle | \langle matr11 \rangle \langle matr12 \rangle \dots | \langle vect1 \rangle \dots$

Тут  $dimnum$  – розмірність трансформації;

$matrMN$  – елемент матричної складової трансформації;

$vectM$  – елемент векторної складової трансформації.

Приклад: 3|1 0 0 0 1 0 0 0 1|0 0 0



## 6 КЕРІВНИЦТВО АДМІНІСТРАТОРА

### 6.1 Встановлення програми

Дана програма не використовує допоміжних файлів, тому для встановлення достатньо скопіювати файли .exe і .config на комп'ютер. Рекомендується також встановити .NET Framework останньої версії з офіційного сайту [6].

Програма перевірена на наступній конфігурації системи:

ОС MS Windows 7

4 GB RAM

2.2 GHz CPU

### 6.2 Підключення до бази даних

Дана програма використовує в своїй роботі базу даних. Вона має бути розгорнута і на ній мають бути створені таблиці що відповідають схемі сутностей.

Для підключення до бази даних необхідно налаштувати файл .config.

Цей файл виглядає наступним чином:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
  </configSections>
  <connectionStrings>
    <add name="MultiDimEditor.Properties.Settings.courseworkConnectionString"
      connectionString="Data Source=<назва серверу>;Initial Catalog=<назва бази
даних>;Integrated Security=True"
      providerName="System.Data.SqlClient" />
  </connectionStrings>
```

```
<startup>
```

```
  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
```

```
</startup>
```

```
</configuration>
```

Якщо сервер бази даних використовує авторизацію за логіном і паролем, замість Integrated Security=True слід записати User ID=<login>;Password=<password>"

## 7 КЕРІВНИЦТВО КОРИСТУВАЧА

### 7.1 Початок роботи

При запуску програми з'являється головне вікно.

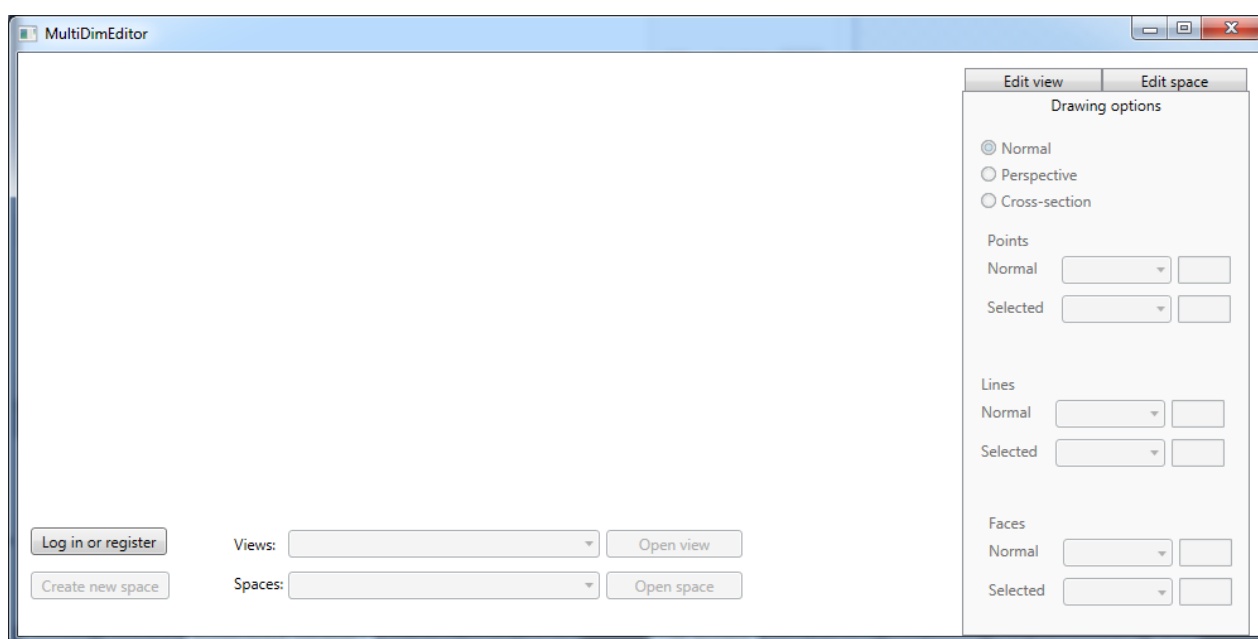


Рисунок 7.1 Головне вікно(не авторизовано)

Відразу після запуску більша частина функцій не діє. В першу чергу користувач має авторизуватися. Для цього необхідно натиснути кнопку “Log in or register”.

Після цього перед користувачем відкривається вікно авторизації.

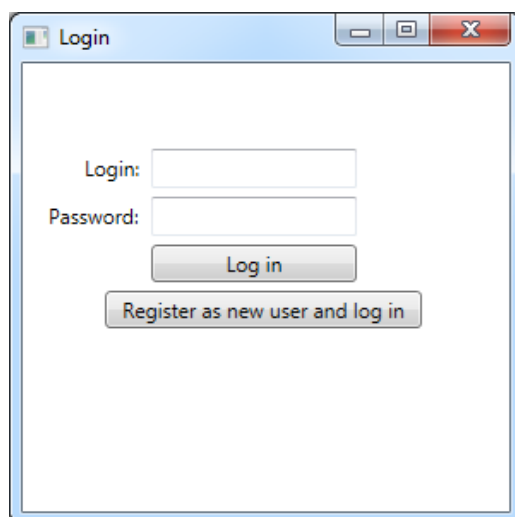


Рисунок 7.2 Вікно авторизації

Це вікно є діалоговим, тобто поки воно активне, взаємодія з головним вікном неможлива.

У вікні авторизації користувач може ввести свої логін і пароль, або зареєструватися як новий користувач. Після успішної авторизації це вікно закривається, а користувач повертається до головного вікна.

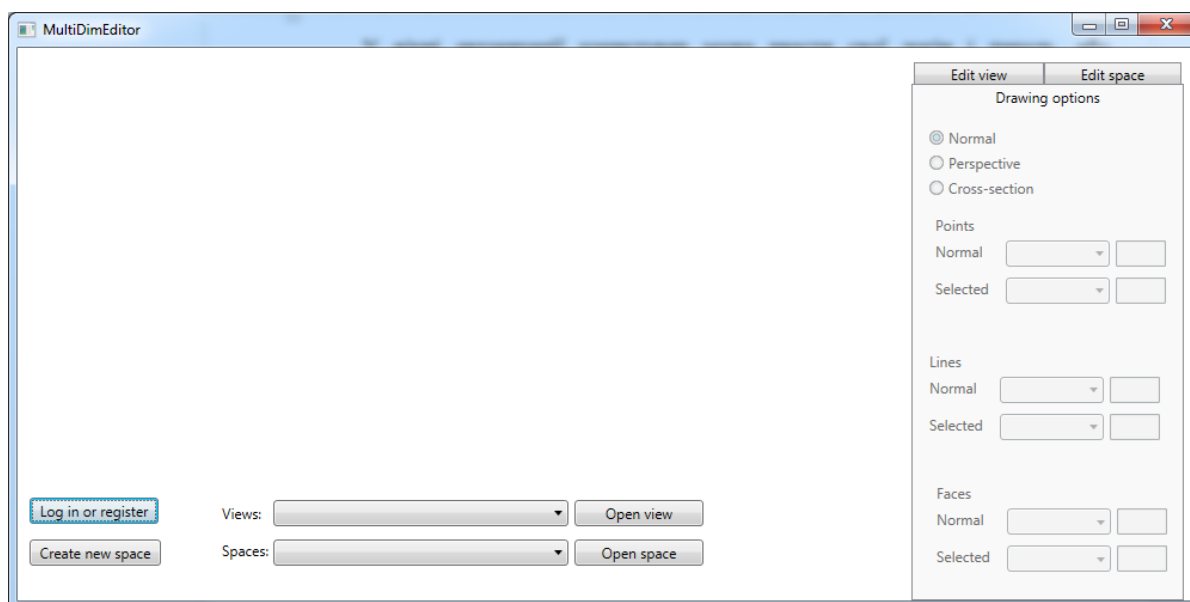


Рисунок 7.3 Головне вікно після успішної авторизації

Якщо авторизувався адміністратор, для нього відкривається додаткове вікно для керування користувачами.

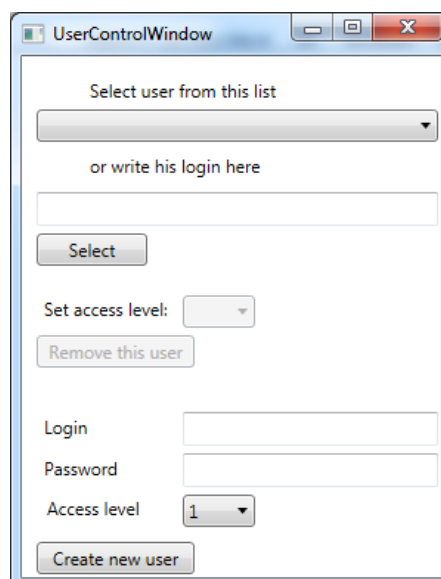


Рисунок 7.4 Вікно керування користувачами

В цьому вікні адміністратор може знаходити користувачів за списком чи логіном, та редагувати їм рівень доступу або видаляти їх. Також тут він може створювати нових користувачів. На відміну від звичайного вікна реєстрації, де рівень доступу нового користувача за замовченням 1(звичайний користувач), вікно адміністратора дозволяє створювати користувачів інших рівнів доступу. Але створити нового адміністратора чи змінити рівень доступу існуючого користувача до адміністраторського він не може, так сам як і змінити свій рівень доступу чи видалити себе.

## 7.2 Вигляди та простори

В даній програмі «простір» - об'єкт, який зберігає точки, а також їх поєднання в ребра і грані. Перегляд простору відбувається через об'єкт «вигляд», в якому зберігається те, як саме слід перетворити простір перед його зображенням.

Після авторизації для користувача, що має хоча б 1й рівень доступу, підбираються 2 списки: список виглядів і список просторів.

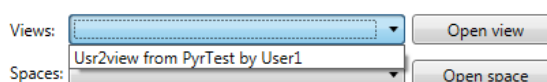


Рисунок 7.5 Список виглядів

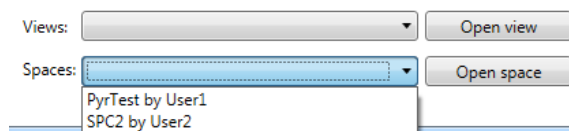


Рисунок 7.6 Список просторів

В списку виглядів користувачу показані всі вигляди, які він створив. Будь-який з них можна відкрити, не вводячи ніякі дані. Але у нового користувача цей список порожній. Вигляди створюються при підключенні до простору.

В списку просторів вказано всі простори, до яких можливо підключитися. Щоб спробувати підключитися до певного простору, необхідно обрати його в списку и натиснути кнопку “Open space”. Це призводить до відкриття вікна підключення до простору.

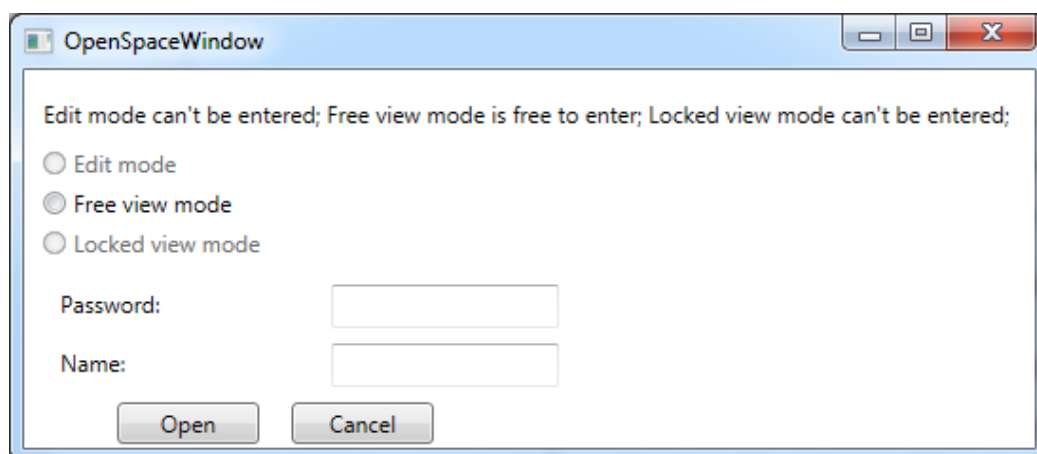


Рисунок 7.7 Вікно підключення до простору

Є три можливих режими підключення:

Режим редагування – дозволяє довільно редагувати як вигляд, так і сам простір.

Режим вільного перегляду – дозволяє змінювати вигляд, але не редагувати простір.

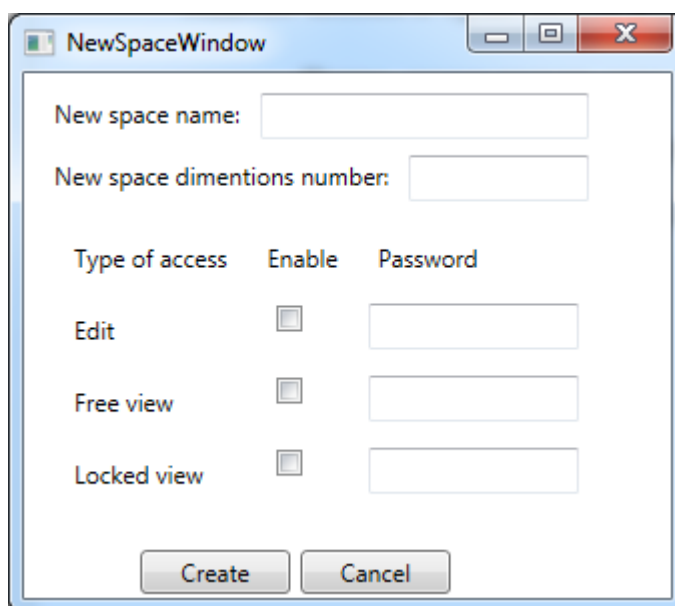
Режим заблокованого перегляду – не дозволяє змінювати вигляд.

Детальніше про ці можливості буде написано в розділі 7.3.

Автор простору може налаштувати кожен з цих 3 режимів доступу трьома різними способами: заборонити режим, дозволити режим або дозволити режим за паролем. У вікні підключення до простору описується, які режими яким способом налаштовані.

Після успішного підключення буде створено вигляд, що належить даному користувачу і відповідає за зображення даного простору. Один користувач може створити декілька виглядів на один і той самий простір. Автор простору може підключатися в будь-якому режимі, незалежно від його налаштувань.

Якщо користувач має 2й рівень доступу(автор) чи вище, йому надається можливість створювати простори. Для створення простору необхідно натиснути кнопку “Create new space”. Після цього відкриється вікно створення нового простору.



The image shows a Windows-style dialog box titled "NewSpaceWindow". It contains the following fields and controls:

- "New space name:" followed by a text input field.
- "New space dimentions number:" followed by a text input field.
- A table with three columns: "Type of access", "Enable", and "Password".
- Row 1: "Edit", a checkbox, and a password input field.
- Row 2: "Free view", a checkbox, and a password input field.
- Row 3: "Locked view", a checkbox, and a password input field.
- At the bottom, there are two buttons: "Create" and "Cancel".

Рисунок 7.8 Вікно створення простору

В цьому вікні треба вказати назву нового простору, його розмірність, а також налаштувати режими доступу. Щоб дозволити доступ в певному режимі, необхідно поставити прапорець в полі Enable напроти відповідного типу доступу. Щоб при цьому необхідно було вводити пароль, цей пароль слід вписати у відповідне поле. Як вже зазначалося, незалежно від цих налаштувань автор простору може підключатись до нього у будь-якому режимі. Також якщо у простору всі режими доступу

заблоковані, його не буде відображати в списку просторів інших користувачів, що дозволяє зробити його приватним.

### 7.3 Редагування зображення, вигляду і простору

Після відкриття вигляду на головному вікні з'являється зображення простору.

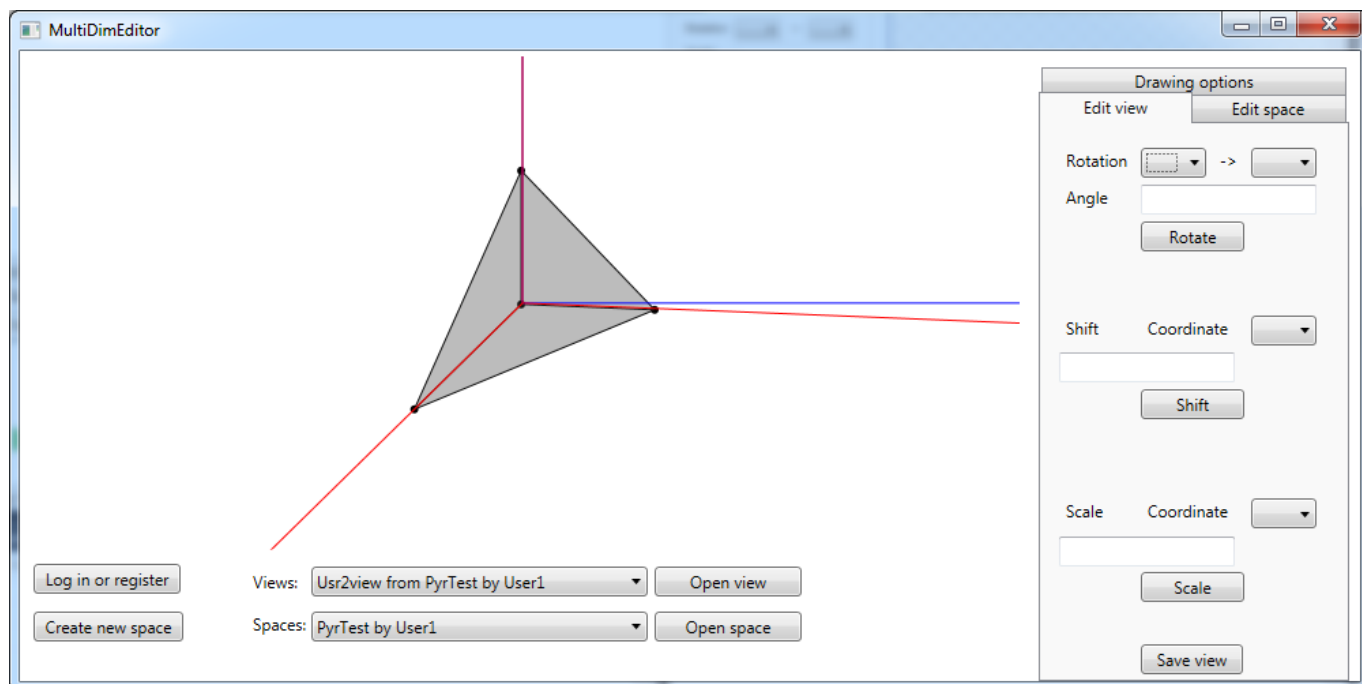


Рисунок 7.9 Головне вікно в режимі зображення

Справа від зображення розташовані керуючі елементи, які розбиті на 3 вкладки.

Вкладка Drawing options відповідає за налаштування зображення: як будувати двовимірне зображення (проекція, перспектива, переріз), яким кольором рисувати точки, ребра та грані, та інші налаштування. Ці налаштування доступні навіть в режимі заблокованого перегляду. Ці опції недоступні в даній версії програми.

Вкладка Edit view відповідає за редагування вигляду. Її функції потребують щонайменше режиму вільного перегляду. Тут можна застосувати поворот, розтяг і зсув.



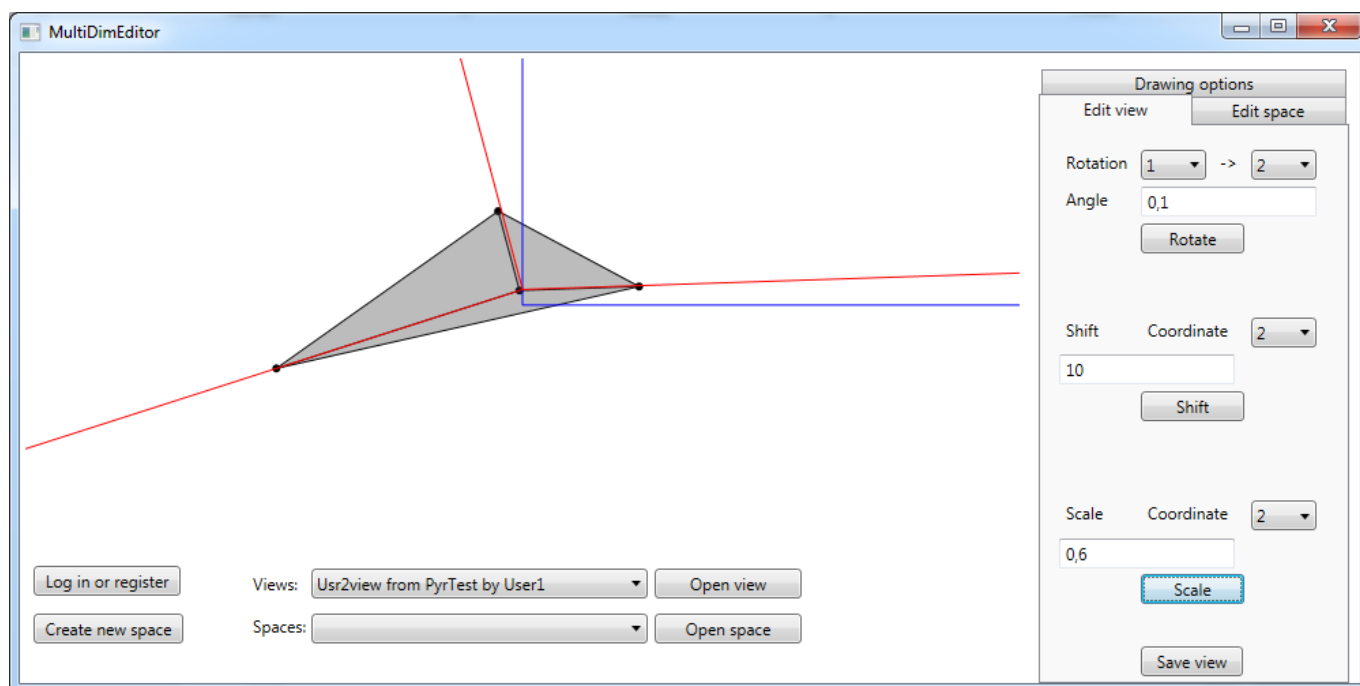


Рисунок 7.10 Результат застосування трансформацій до попереднього вигляду

В багатовимірному просторі поворот задається напрямком повороту і величиною, а напрямок можна задати двома векторами: початковим і кінцевим. Будь-який поворот можна розбити на елементарні повороти, кожний з яких відбувається від одної з координатних осей до іншої з певним кутом повороту. Саме такі елементарні повороти можна тут виконати.

Розтяг виконується по заданій осі на заданий коефіцієнт. Коефіцієнт, що менший 1 за модулем відповідає стисненню. Від'ємний коефіцієнт призведе до розвороту фігури. Використовувати коефіцієнт 0 небажано, оскільки це призведе до того, що відповідна координата всіх точок буде рівна нулю, і надалі ніякі трансформації не зможуть відновити попередні координати.

Зсув виконується по заданій осі на задану відстань.

Кожна трансформація додається в кінець списку трансформацій. В майбутніх версіях програми буде реалізована можливість працювати із самим списком: додавати трансформацію в будь-яке місце списку чи видаляти трансформацію з будь-якого місця списку.

Кнопка “Save view” зберігає зміни вигляду. При цьому список трансформацій перетворюється в подвійну трансформацію, тим самим інформація про послідовність дій втрачається.

Вкладка Edit space відповідає за редагування простору. Для використання її елементів необхідно мати режим редагування. Тут можна створювати нові точки, редагувати та видаляти існуючі, створювати і видаляти ребра та грані.

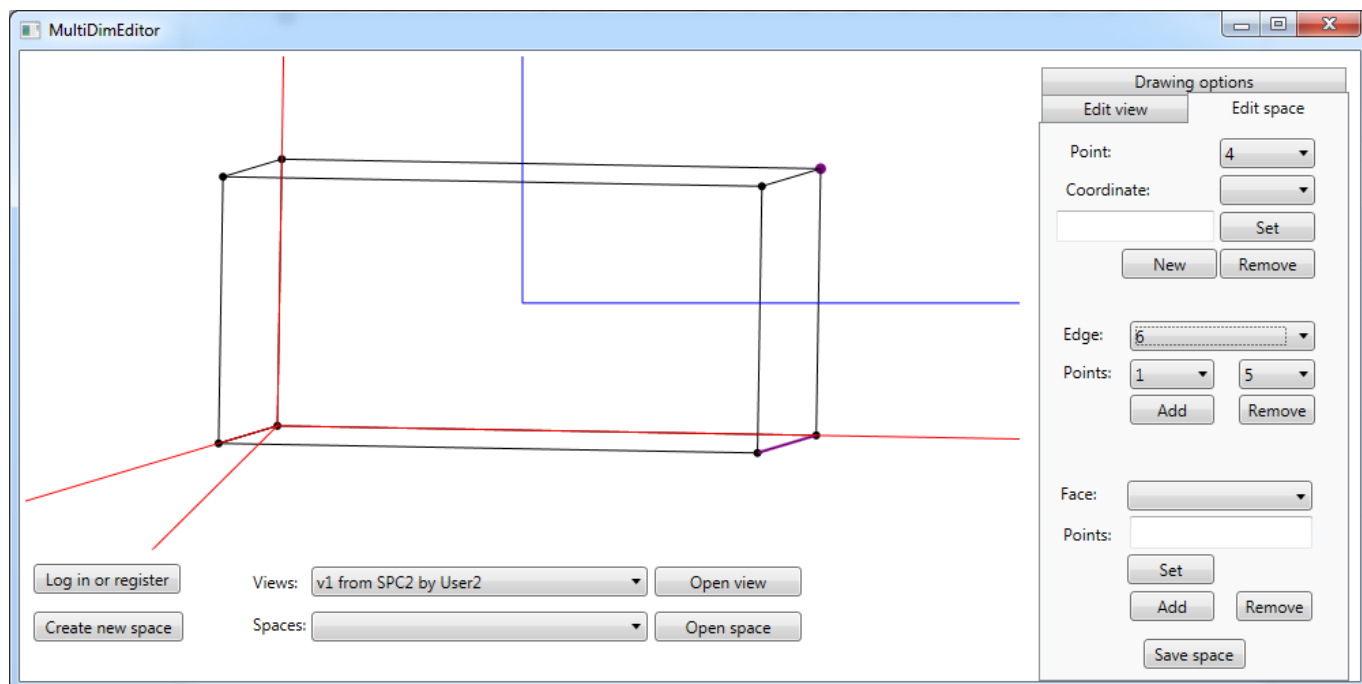


Рисунок 7.11 Простір відкрито в режимі редагування

Ця вкладка розбита на 3 частини: керування точками, ребрами і гранями.

Для точок є можливість обрати точку зі списку(при цьому вона виділяється на зображенні), і можна переглядати чи змінювати її координати, а також створювати нові точки(за замовченням вони мають нульові координати) чи видаляти їх.

Для ребер є можливість обрати ребро зі списку(при цьому воно виділяється на зображенні) і можна переглядати, які саме точки воно з'єднує, а також додати грань між цими точками чи видалити виділену грань.

Кнопка “Save space” зберігає зміни простору. При цьому зміни вигляду не зберігаються, але можуть бути збережені окремо.

## ВИСНОВОК

При виконанні курсової роботи було розроблено додаток для редагування багатовимірних фігур. Цей додаток дозволяє створювати, редагувати і зберігати такі фігури, а також забезпечує можливість надання іншим користувачам різного роду доступу до них. Це дозволяє працювати над однією фігурою декільком користувачам, або надавати певним користувачам можливість переглядати фігуру без редагування. Також передбачена можливість адміністрування – створення і видалення користувачів, та зміна можливостей цих користувачів.

Додаток розроблено із врахуванням особливостей роботи з багатовимірним простором. Передбачено виконання різного роду трансформацій, серед них – елементарні повороти в будь-якій координатній площині.

Для розробки додатку були використані наступні засоби: мова програмування C#, платформа .NET, середовище розробки MS Visual Studio, та технологія розробки користувацького інтерфейсу WPF. Для збереження даних було використано СУБД MS SQL Server.

Подальший розвиток цього додатку передбачає надання користувачу таких можливостей, як: керування налаштуваннями відображення, можливість скасовувати останні дії і коректна обробка одночасного редагування. Після цього планується реалізувати такі необов'язкові, але зручні можливості, як буфер обміну, робота з локальними файлами та макрокоманди.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Visualizer: обзор. [Електронний ресурс]. – Режим доступу:  
<http://www.novospark.com/ru/Products/Visualizer/Overview.aspx>
2. Features – blender.org [Електронний ресурс]. – <https://www.blender.org/features/>
3. Four-dimensional space. [Електронний ресурс]. – Режим доступу:  
[https://en.wikipedia.org/wiki/Four-dimensional\\_space](https://en.wikipedia.org/wiki/Four-dimensional_space)
4. Rotations in 4-dimentional Euclidean space [Електронний ресурс]. – Режим доступу: [https://en.wikipedia.org/wiki/Rotations\\_in\\_4-dimensional\\_Euclidean\\_space](https://en.wikipedia.org/wiki/Rotations_in_4-dimensional_Euclidean_space)
5. Conic section. [Електронний ресурс]. –  
[https://en.wikipedia.org/wiki/Conic\\_section](https://en.wikipedia.org/wiki/Conic_section)
6. Download .NET Framework. [Електронний ресурс]. –  
<https://dotnet.microsoft.com/download/dotnet-framework-runtime>
7. Каталог API (Microsoft) и справочных материалов [Електронний ресурс]. –  
Режим доступу: <https://msdn.microsoft.com/ru-ru/library/>

## ДОДАТОК А (Текст програми «MultiDimEditor»)

```

MainWindow.xaml.cs:
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace MultiDimEditor
{
    /// <summary>
    /// Логика взаємодії для MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        TransformList tlist;
        Space currentspace;
        DBModelDataContext dc;
        ObservableCollection<Views> userViews = new ObservableCollection<Views>();
        ObservableCollection<Spaces> userSpaces = new ObservableCollection<Spaces>();
        Users currentuser = null;
        Views currentview = null;
        List<Control> lvl1Controls;
        List<Control> lvl2Controls;
        DrawingRules currules;
        public MainWindow()
        {
            InitializeComponent();
            dc = new DBModelDataContext();
            cbViews.ItemsSource = userViews;
            cbSpaces.ItemsSource = userSpaces;
            lvl2Controls = new List<Control>();
            lvl2Controls.Add(btnNewSpace);

            lvl1Controls = new List<Control>();
            lvl1Controls.Add(cbViews);
            lvl1Controls.Add(cbSpaces);
            lvl1Controls.Add(btnOpenSpace);
            lvl1Controls.Add(btnOpenView);
            lvl1Controls.AddRange(lvl2Controls);
            SwitchControls(0);
            SwitchDrawControls(-1);
            currules = new DrawingRules();
            cbPointNormal.ItemsSource = DrawingRules.AllColors;
            cbPointSelected.ItemsSource = DrawingRules.AllColors;
            cbLineNormal.ItemsSource = DrawingRules.AllColors;
            cbLineSelected.ItemsSource = DrawingRules.AllColors;
            cbFaceNormal.ItemsSource = DrawingRules.AllColors;
            cbFaceSelected.ItemsSource = DrawingRules.AllColors;
        }
        void SwitchControls(int access_type)
        {
            if (access_type == 0)
                foreach (var c in lvl1Controls) c.IsEnabled = false;
            else if (access_type == 1)
            {
                foreach (var c in lvl1Controls) c.IsEnabled = true;
                foreach (var c in lvl2Controls) c.IsEnabled = false;
            }
            else if (access_type == 2)
                foreach (var c in lvl1Controls) c.IsEnabled = true;
        }
        void SwitchDrawControls(int access_type)
        {
            //-1 means no view loaded
        }
    }
}

```

```

        foreach (var ctrl in grDrawOptions.Children) ((Control)ctrl).IsEnabled = (access_type > -1);
        foreach (var ctrl in grViewOptions.Children) ((Control)ctrl).IsEnabled = (access_type > 0);
        foreach (var ctrl in grSpaceOptions.Children) ((Control)ctrl).IsEnabled = (access_type > 1);
    }
    void Draw()
    {
        if (currentview == null) return;
        DrawingVisual dv = new DrawingVisual();
        Point center = new Point(cnvDraw.ActualWidth / 2, cnvDraw.ActualHeight / 2);
        using (DrawingContext dc = dv.RenderOpen())
        {
            currentspace.Draw(dc, center, tlist, currules);
        }
        VisualHost vh = new VisualHost() { visual = dv };
        cnvDraw.Children.Clear();
        cnvDraw.Children.Add(vh);

        cbPickPoint.Items.Clear();
        for (int i = 0; i < currentspace.points.Count; i++)
        {
            cbPickPoint.Items.Add(i);
        }
    }

    private void Button_Click_2(object sender, RoutedEventArgs e)
    {
    }

    private void cbViews_SelectionChanged(object sender, SelectionChangedEventArgs e)
    {
    }

    private void Window_Closing(object sender, System.ComponentModel.CancelEventArgs e)
    {
        dc.Connection.Close();
    }
    void UpdateUserLists()
    {
        foreach (var view in currentuser.Views)
            userViews.Add(view);
        foreach (var sp in dc.Spaces)
            if (sp.Creator.USER_ID == currentuser.USER_ID || sp.EditorPassword != null || sp.FreePassword !=
null || sp.LockedPassword != null)
                userSpaces.Add(sp);
    }
    private void btnLogIn_Click(object sender, RoutedEventArgs e)
    {
        userViews.Clear();
        userSpaces.Clear();
        SwitchControls(0);
        var logwnd = new LoginWindow();
        if(logwnd.ShowDialog()==true)
        {
            var us = logwnd.User;
            currentuser = us;
            int access = us.Access_type;
            SwitchControls(access);
            if(access==0)
            {
                MessageBox.Show("You were banned");
                return;
            }
            if(access>=1)
            {
                UpdateUserLists();
            }
            if(access>=2)
            {
            }
            if(access==3)
            {
                UserControlWindow ucw = new UserControlWindow();
                ucw.Show();
            }
        }
    }

```

```

        SwitchDrawControls(-1);
    }
}
void ViewOpen()
{
    currentspace = Space.Parse(currentview.Space.Content);
    tlist = new TransformList(Space.Parse(currentview.Space.Content).dimnum);
    tlist.AddLast(TransformDouble.Parse(currentview.Transform));
    UpdateUserLists();
    SwitchDrawControls(currentview.Access_type);
    cbRotStart.Items.Clear();
    cbRotEnd.Items.Clear();
    cbScaleCoord.Items.Clear();
    cbShiftCoord.Items.Clear();
    cbPickPointCoord.Items.Clear();
    for (int i = 0; i < currentspace.dimnum; i++)
    {
        cbRotStart.Items.Add(i + 1);
        cbRotEnd.Items.Add(i + 1);
        cbScaleCoord.Items.Add(i + 1);
        cbShiftCoord.Items.Add(i + 1);
        cbPickPointCoord.Items.Add(i + 1);
    }
    Draw();
}
private void btnOpenSpace_Click(object sender, RoutedEventArgs e)
{
    if(cbSpaces.SelectedItem==null)
    {
        MessageBox.Show("No space chosen");
        return;
    }
    OpenSpaceWindow osw = new OpenSpaceWindow(currentuser, (Spaces)cbSpaces.SelectedItem);
    if(osw.ShowDialog()==true)
    {
        Spaces cursp=(Spaces)cbSpaces.SelectedItem;
        //Views view = new Views(){Name=osw.Name, Access_type=osw.Level, SpaceID=cursp.SPACE_ID,
        UserID=currentuser.USER_ID,
        Transform=TransformDouble.GetIdentityTransform(Space.Parse(cursp.Content).dimnum).ToString(),
        VIEW_ID=Guid.NewGuid()};
        Views view = osw.Resultview;
        currentview = view;
        dc.Views.InsertOnSubmit(view);

        dc.SubmitChanges();
        dc.Refresh(System.Data.Linq.RefreshMode.OverwriteCurrentValues, dc.Views);
        dc.Refresh(System.Data.Linq.RefreshMode.OverwriteCurrentValues, dc.Users);
        ViewOpen();
    }
}

private void btnOpenView_Click(object sender, RoutedEventArgs e)
{
    if (cbViews.SelectedItem == null)
    {
        MessageBox.Show("No space chosen");
        return;
    }
    Views view = (Views)cbViews.SelectedItem;
    currentview = view;
    ViewOpen();
}

private void btnNewSpace_Click(object sender, RoutedEventArgs e)
{
    NewSpaceWindow nsw = new NewSpaceWindow();
    nsw.user = currentuser;
    if(nsw.ShowDialog()==true)
    {
        dc.Spaces.InsertOnSubmit(nsw.NewSpace);
        dc.SubmitChanges();
        dc.Refresh(System.Data.Linq.RefreshMode.OverwriteCurrentValues, dc.Views);
        dc.Refresh(System.Data.Linq.RefreshMode.OverwriteCurrentValues, dc.Users);
        UpdateUserLists();
    }
}

```

```

private void Button_Click(object sender, RoutedEventArgs e)
{
    double angle = 0;
    if(!double.TryParse(tbAngle.Text, out angle))
    {
        MessageBox.Show("Turn angle must be a number");
        return;
    }

    if(cbRotStart.SelectedIndex<0||cbRotEnd.SelectedIndex<0||cbRotStart.SelectedIndex==cbRotEnd.SelectedIndex)
    {
        MessageBox.Show("Select a start and end axis. They must not be equal");
        return;
    }
    TransformMatrix tm = new TransformMatrix(DMatrix.GetRotMatrix(currentspace.dimnum,
cbRotStart.SelectedIndex, cbRotEnd.SelectedIndex, angle));
    tlist.AddLast(tm);
    Draw();
}

private void Button_Click_1(object sender, RoutedEventArgs e)
{
    var q = from v in dc.Views
            where v.VIEW_ID == currentview.VIEW_ID
            select v;
    q.First().Transform = tlist.ToString();
    dc.SubmitChanges();
}

private void Button_Click_3(object sender, RoutedEventArgs e)
{
    double scale = 0;
    if (!double.TryParse(tbScaleVal.Text, out scale))
    {
        MessageBox.Show("Scale must be a number");
        return;
    }
    if(cbScaleCoord.SelectedIndex<0)
    {
        MessageBox.Show("Incorrect coordinate for scaling");
        return;
    }
    TransformScale ts = new TransformScale(currentspace.dimnum, cbScaleCoord.SelectedIndex,scale);
    tlist.AddLast(ts);
    Draw();
}

private void Button_Click_4(object sender, RoutedEventArgs e)
{
    double shift = 0;
    if (!double.TryParse(tbShiftVal.Text, out shift))
    {
        MessageBox.Show("Shift must be a number");
        return;
    }
    if (cbShiftCoord.SelectedIndex < 0)
    {
        MessageBox.Show("Incorrect coordinate for shifting");
        return;
    }
    TransformVector tv = new TransformVector(DVector.GetOrt(cbShiftCoord.SelectedIndex,
currentspace.dimnum)*shift);
    tlist.AddLast(tv);
    Draw();
}

private void Button_Click_5(object sender, RoutedEventArgs e)
{
    double val = 0;
    if (!double.TryParse(tbSetPointCoord.Text, out val))
    {
        MessageBox.Show("Coordinate value must be a number");
        return;
    }
    if (cbPickPointCoord.SelectedIndex < 0)

```



```

    {
        MessageBox.Show("Incorrect coordinate selection");
        return;
    }
    if (cbPickPoint.SelectedIndex < 0)
    {
        MessageBox.Show("Incorrect point selection");
        return;
    }
    currentspace.points[cbPickPoint.SelectedIndex][cbPickPointCoord.SelectedIndex] = val;
    Draw();
}

private void cbPickPointCoord_SelectionChanged(object sender, SelectionChangedEventArgs e)
{
    if (cbPickPoint.SelectedIndex < 0)
        tbSetPointCoord.Text =
currentspace.points[cbPickPoint.SelectedIndex][cbPickPointCoord.SelectedIndex].ToString();
}

private void Button_Click_6(object sender, RoutedEventArgs e)
{
    currentspace.points.Add(new DVector(currentspace.dimnum));
    Draw();
}
}
}

MainWindow.xaml:
<Window x:Class="MultiDimEditor.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:xctk="http://schemas.xceed.com/wpf/xaml/toolkit"
    Title="MultiDimEditor" Height="500" Width="1000" Closing="Window_Closing">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="4*" />
            <RowDefinition Height="1*" />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="3*" />
            <ColumnDefinition Width="1*" />
        </Grid.ColumnDefinitions>
        <Canvas x:Name="cnvDraw" HorizontalAlignment="Stretch" Margin="4,4,4,4" VerticalAlignment="Stretch"
ClipToBounds="True"/>
        <ComboBox x:Name="cbViews" IsEnabled="False" HorizontalAlignment="Left" Margin="214,8,0,0"
VerticalAlignment="Top" Width="247" SelectionChanged="cbViews_SelectionChanged" Grid.Row="1">
            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal" >
                        <TextBlock Text="{Binding Path=Name}" /></TextBlock>
                        <TextBlock Text=" from " /></TextBlock>
                        <TextBlock Text="{Binding Path=Space.Name}" /></TextBlock>
                        <TextBlock Text=" by " /></TextBlock>
                        <TextBlock Text="{Binding Path=Space.Creator.Name}" /></TextBlock>
                    </StackPanel>
                </DataTemplate>
            </ComboBox.ItemTemplate>
        </ComboBox>
        <Button x:Name="btnLogIn" Content="Log in or register" HorizontalAlignment="Left" Margin="10,6,0,0"
VerticalAlignment="Top" Width="108" Click="btnLogIn_Click" Grid.Row="1"/>
        <Label Content="Views:" HorizontalAlignment="Left" Margin="166,6,0,0" VerticalAlignment="Top" Grid.Row="1"/>
        <ComboBox x:Name="cbSpaces" IsEnabled="False" HorizontalAlignment="Left" Margin="214,41,0,0"
VerticalAlignment="Top" Width="247" Grid.Row="1">
            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Horizontal" >
                        <TextBlock Text="{Binding Name}" />
                        <TextBlock Text=" by " />
                        <TextBlock Text="{Binding Creator.Name}" />
                    </StackPanel>
                </DataTemplate>
            </ComboBox.ItemTemplate>
        </ComboBox>
        <Label Content="Spaces:" HorizontalAlignment="Left" Margin="166,37,0,0" VerticalAlignment="Top"
Grid.Row="1"/>

```

```

        <Button x:Name="btnNewSpace" IsEnabled="False" Content="Create new space" HorizontalAlignment="Left"
Margin="10,41,0,0" VerticalAlignment="Top" Width="110" Grid.Row="1" Click="btnNewSpace_Click"/>
        <Button x:Name="btnOpenSpace" Content="Open space" HorizontalAlignment="Left" Margin="466,41,0,0"
VerticalAlignment="Top" Width="108" Grid.Row="1" Click="btnOpenSpace_Click"/>
        <Button x:Name="btnOpenView" Content="Open view" HorizontalAlignment="Left" Margin="466,8,0,0"
VerticalAlignment="Top" Width="108" Grid.Row="1" Click="btnOpenView_Click"/>
        <TabControl HorizontalAlignment="Left" Height="Auto" Margin="10,10,0,0" VerticalAlignment="Stretch"
Width="228" SelectedIndex="0" Grid.Column="1" Grid.RowSpan="2">
            <TabItem Header="Drawing options">
                <Grid x:Name="grDrawOptions">
                    <Grid.RowDefinitions>
                        <RowDefinition Height="Auto"/>
                        <RowDefinition Height="*/>
                        <RowDefinition Height="*/>
                        <RowDefinition Height="*/>
                    </Grid.RowDefinitions>
                    <RadioButton x:Name="rbNormalMode" Content="Normal" HorizontalAlignment="Left"
Margin="10,10,0,0" VerticalAlignment="Top" IsChecked="True"/>
                    <RadioButton x:Name="rbPerspectiveMode" Content="Perspective" HorizontalAlignment="Left"
Margin="10,31,0,0" VerticalAlignment="Top"/>
                    <RadioButton x:Name="rbCSectionMode" Content="Cross-section" HorizontalAlignment="Left"
Margin="10,52,0,0" VerticalAlignment="Top"/>
                    <Label Content="Points" HorizontalAlignment="Left" Margin="10,10,0,0" Grid.Row="1"
VerticalAlignment="Top"/>
                    <ComboBox x:Name="cbPointNormal" HorizontalAlignment="Left" Margin="74,36,0,0" Grid.Row="1"
VerticalAlignment="Top" Width="87"/>
                    <TextBox HorizontalAlignment="Left" Height="22" Margin="166,36,0,0" Grid.Row="1"
TextWrapping="Wrap" VerticalAlignment="Top" Width="42"/>
                    <Label Content="Normal" HorizontalAlignment="Left" Margin="10,32,0,0" Grid.Row="1"
VerticalAlignment="Top"/>
                    <ComboBox x:Name="cbPointSelected" HorizontalAlignment="Left" Margin="74,67,0,0" Grid.Row="1"
VerticalAlignment="Top" Width="87"/>
                    <TextBox HorizontalAlignment="Left" Height="22" Margin="166,67,0,0" Grid.Row="1"
TextWrapping="Wrap" VerticalAlignment="Top" Width="42"/>
                    <Label Content="Selected" HorizontalAlignment="Left" Margin="10,63,0,0" Grid.Row="1"
VerticalAlignment="Top"/>
                    <Label Content="Lines" HorizontalAlignment="Left" Margin="5,13,0,0" Grid.Row="2"
VerticalAlignment="Top"/>
                    <ComboBox x:Name="cbLineNormal" HorizontalAlignment="Left" Margin="69,39,0,0" Grid.Row="2"
VerticalAlignment="Top" Width="87"/>
                    <TextBox HorizontalAlignment="Left" Height="22" Margin="161,39,0,0" Grid.Row="2"
TextWrapping="Wrap" VerticalAlignment="Top" Width="42"/>
                    <Label Content="Normal" HorizontalAlignment="Left" Margin="5,35,0,0" Grid.Row="2"
VerticalAlignment="Top"/>
                    <ComboBox x:Name="cbLineSelected" HorizontalAlignment="Left" Margin="69,70,0,0" Grid.Row="2"
VerticalAlignment="Top" Width="87"/>
                    <TextBox HorizontalAlignment="Left" Height="22" Margin="161,70,0,0" Grid.Row="2"
TextWrapping="Wrap" VerticalAlignment="Top" Width="42"/>
                    <Label Content="Selected" HorizontalAlignment="Left" Margin="5,66,0,0" Grid.Row="2"
VerticalAlignment="Top"/>
                    <Label Content="Faces" HorizontalAlignment="Left" Margin="11,13,0,0" Grid.Row="3"
VerticalAlignment="Top"/>
                    <ComboBox x:Name="cbFaceNormal" HorizontalAlignment="Left" Margin="75,39,0,0" Grid.Row="3"
VerticalAlignment="Top" Width="87"/>
                    <TextBox HorizontalAlignment="Left" Height="22" Margin="167,39,0,0" Grid.Row="3"
TextWrapping="Wrap" VerticalAlignment="Top" Width="42"/>
                    <Label Content="Normal" HorizontalAlignment="Left" Margin="11,35,0,0" Grid.Row="3"
VerticalAlignment="Top"/>
                    <ComboBox x:Name="cbFaceSelected" HorizontalAlignment="Left" Margin="75,70,0,0" Grid.Row="3"
VerticalAlignment="Top" Width="87"/>
                    <TextBox HorizontalAlignment="Left" Height="22" Margin="167,70,0,0" Grid.Row="3"
TextWrapping="Wrap" VerticalAlignment="Top" Width="42"/>
                    <Label Content="Selected" HorizontalAlignment="Left" Margin="11,66,0,0" Grid.Row="3"
VerticalAlignment="Top"/>
                </Grid>
            </TabItem>
            <TabItem Header="Edit view">
                <Grid x:Name="grViewOptions" >
                    <Grid.RowDefinitions>
                        <RowDefinition Height="*/>
                        <RowDefinition Height="*/>
                        <RowDefinition Height="*/>
                        <RowDefinition Height="Auto"/>
                    </Grid.RowDefinitions>
                    <Label Content="Rotation" HorizontalAlignment="Left" Margin="10,10,0,0"
VerticalAlignment="Top"/>

```

```

        <ComboBox x:Name="cbRotStart" HorizontalAlignment="Left" Margin="70,14,0,0"
VerticalAlignment="Top" Width="48"/>
        <ComboBox x:Name="cbRotEnd" HorizontalAlignment="Left" Margin="151,14,0,0"
VerticalAlignment="Top" Width="48"/>
        <Label Content="->" HorizontalAlignment="Left" Margin="123,10,0,0" VerticalAlignment="Top"/>
        <TextBox x:Name="tbAngle" HorizontalAlignment="Left" Height="22" Margin="70,41,0,0"
TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="129"/>
        <Label Content="Angle" HorizontalAlignment="Left" Margin="10,37,0,0" VerticalAlignment="Top"/>
        <Button Content="Rotate" HorizontalAlignment="Left" Margin="70,68,0,0" VerticalAlignment="Top"
Width="76" Click="Button_Click"/>
        <Label Content="Shift" HorizontalAlignment="Left" Margin="10,10,0,0" Grid.Row="1"
VerticalAlignment="Top"/>
        <ComboBox x:Name="cbShiftCoord" HorizontalAlignment="Left" Margin="151,14,0,0"
VerticalAlignment="Top" Width="48" Grid.Row="1"/>
        <Label Content="Coordinate" HorizontalAlignment="Left" Margin="70,10,0,0" Grid.Row="1"
VerticalAlignment="Top"/>
        <TextBox x:Name="tbShiftVal" HorizontalAlignment="Left" Height="22" Margin="10,41,0,0"
TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="129" Grid.Row="1"/>
        <Button Content="Shift" HorizontalAlignment="Left" Margin="70,68,0,0" VerticalAlignment="Top"
Width="76" Grid.Row="1" Click="Button_Click_4"/>
        <Label Content="Scale" HorizontalAlignment="Left" Margin="10,22,0,0" Grid.Row="2"
VerticalAlignment="Top"/>
        <ComboBox x:Name="cbScaleCoord" HorizontalAlignment="Left" Margin="151,26,0,0"
VerticalAlignment="Top" Width="48" Grid.Row="2"/>
        <Label Content="Coordinate" HorizontalAlignment="Left" Margin="70,22,0,0" Grid.Row="2"
VerticalAlignment="Top"/>
        <TextBox x:Name="tbScaleVal" HorizontalAlignment="Left" Height="22" Margin="10,54,0,0"
TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="129" Grid.Row="2"/>
        <Button Content="Scale" HorizontalAlignment="Left" Margin="70,80,0,0" VerticalAlignment="Top"
Width="76" Grid.Row="2" Click="Button_Click_3"/>
        <Button Content="Save view" HorizontalAlignment="Left" Margin="70,10,0,0" Grid.Row="3"
VerticalAlignment="Top" Width="75" Click="Button_Click_1"/>
    </Grid>
</TabItem>
<TabItem Header="Edit space">
    <Grid x:Name="grSpaceOptions" Margin="0,0,0,1">
        <Grid.RowDefinitions>
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>
        <ComboBox HorizontalAlignment="Left" Margin="128,7,0,0" VerticalAlignment="Top" Width="70"/>
        <ComboBox HorizontalAlignment="Left" Margin="128,34,0,0" VerticalAlignment="Top" Width="70"/>
        <Label Content="Point: " HorizontalAlignment="Left" Margin="13,3,0,0" VerticalAlignment="Top"/>
        <Label Content="Coordinate: " HorizontalAlignment="Left" Margin="10,31,0,0"
VerticalAlignment="Top"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="8,60,0,0" TextWrapping="Wrap" Text=""
VerticalAlignment="Top" Width="116"/>
        <Button Content="Set" HorizontalAlignment="Left" Margin="128,61,0,0" VerticalAlignment="Top"
Width="70"/>
        <ComboBox HorizontalAlignment="Left" Margin="62,17,0,0" VerticalAlignment="Top" Width="136"
Grid.Row="1"/>
        <Label Content="Edge: " HorizontalAlignment="Left" Margin="8,13,0,0" VerticalAlignment="Top"
Grid.Row="1"/>
        <Button Content="New" HorizontalAlignment="Left" Margin="56,88,0,0" VerticalAlignment="Top"
Width="70"/>
        <ComboBox HorizontalAlignment="Left" Margin="62,45,0,0" VerticalAlignment="Top" Width="62"
Grid.Row="1"/>
        <ComboBox HorizontalAlignment="Left" Margin="142,45,0,0" VerticalAlignment="Top" Width="56"
Grid.Row="1"/>
        <Label Content="Points: " HorizontalAlignment="Left" Margin="8,41,0,0" VerticalAlignment="Top"
Grid.Row="1"/>
        <Button Content="Add" HorizontalAlignment="Left" Margin="62,71,0,0" VerticalAlignment="Top"
Width="62" Grid.Row="1"/>
        <Button Content="Remove" HorizontalAlignment="Left" Margin="142,71,0,0" VerticalAlignment="Top"
Width="56" Grid.Row="1"/>
        <ComboBox HorizontalAlignment="Left" Margin="60,11,0,0" VerticalAlignment="Top" Width="136"
Grid.Row="2"/>
        <Label Content="Face: " HorizontalAlignment="Left" Margin="6,7,0,0" VerticalAlignment="Top"
Grid.Row="2"/>
        <Label Content="Points: " HorizontalAlignment="Left" Margin="8,37,0,0" VerticalAlignment="Top"
Grid.Row="2"/>
        <TextBox HorizontalAlignment="Left" Height="24" Margin="62,37,0,0" TextWrapping="Wrap" Text=""
VerticalAlignment="Top" Width="134" Grid.Row="2"/>

```

```

        <Button Content="Set" HorizontalAlignment="Left" Margin="60,65,0,0" VerticalAlignment="Top"
Width="64" Grid.Row="2"/>
        <Button Content="Add" HorizontalAlignment="Left" Margin="62,92,0,0" VerticalAlignment="Top"
Width="62" Grid.Row="2"/>
        <Button Content="Remove" HorizontalAlignment="Left" Margin="140,92,0,0" VerticalAlignment="Top"
Width="56" Grid.Row="2"/>
        <Button Content="Remove" HorizontalAlignment="Left" Margin="128,88,0,0" VerticalAlignment="Top"
Width="70"/>

    </Grid>
</TabItem>
</TabControl>

</Grid>
</Window>

```

```

DVector.cs:
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MultiDimEditor
{
    class DVector //1*n
    {
        public int DimNumber { get; private set; }
        private double[] coordinates;
        public DVector(int dimnum)
        {
            DimNumber = dimnum;
            coordinates = new double[DimNumber];
        }
        public DVector(params double[] coords) :this(coords.Length)
        {
            for (int i = 0; i < DimNumber; i++) coordinates[i] = coords[i];
        }
        public double this[int index]
        {
            get { return coordinates[index]; }
            private set { coordinates[index] = value; }
        }
        public static DVector operator +(DVector a, DVector b)
        {
            if (a.DimNumber != b.DimNumber) throw new ArgumentException();
            DVector res = new DVector(a.DimNumber);
            for (int i = 0; i < a.DimNumber; i++) res[i] = a[i] + b[i];
            return res;
        }
        public static DVector operator *(DVector a, double b)//multiplication by scalar
        {
            DVector res = new DVector(a.DimNumber);
            for (int i = 0; i < a.DimNumber; i++) res[i] = a[i]*b;
            return res;
        }
        public static DVector operator *(double b, DVector a)//multiplication by scalar
        {
            DVector res = new DVector(a.DimNumber);
            for (int i = 0; i < a.DimNumber; i++) res[i] = a[i] * b;
            return res;
        }
        public static DVector operator -(DVector a)
        {
            return a * (-1);
        }
        public static DVector operator -(DVector a, DVector b)
        {
            return a + (-b);
        }
        public static double operator *(DVector a, DVector b)//scalar multiplication
        {
            if (a.DimNumber != b.DimNumber) throw new ArgumentException();
            double res = 0;
            for (int i = 0; i < a.DimNumber; i++) res+= a[i] * b[i];
            return res;
        }
    }
}

```

```

    }
    public static DVector GetOrt(int dim, int dimnum)
    {
        DVector res = new DVector(dimnum);
        res[dim] = 1;
        return res;
    }
    public override string ToString()
    {
        string res = "";
        foreach (var c in coordinates)
            res += c + " ";
        return res.Remove(res.Length - 1);
    }
}

class DMatrix //n*n
{
    public int DimNumber { get; private set; }
    private double[,] coordinates;
    public DMatrix(int dimnum)
    {
        DimNumber = dimnum;
        coordinates = new double[DimNumber, DimNumber];
    }
    public DMatrix(double[,] coords):this(coords.GetUpperBound(0)+1)
    {
        for (int row = 0; row < DimNumber; row++)
            for (int col = 0; col < DimNumber; col++)
                coordinates[row, col] = coords[row, col];
    }
    double this[int row, int col]
    {
        get { return coordinates[row, col]; }
        set { coordinates[row, col] = value; }
    }
    DVector GetRow(int row)
    {
        double[] res = new double[DimNumber];
        for (int i = 0; i < DimNumber; i++) res[i] = coordinates[row, i];
        return new DVector(res);
    }
    DVector GetCol(int col)
    {
        double[] res = new double[DimNumber];
        for (int i = 0; i < DimNumber; i++) res[i] = coordinates[i, col];
        return new DVector(res);
    }
    public static DMatrix Get1Matrix(int dimnum)
    {
        DMatrix res = new DMatrix(dimnum);
        for (int i = 0; i < dimnum; i++) res[i, i] = 1;
        return res;
    }
    public static DMatrix GetRotMatrix(int dimnum, int axis1, int axis2, double angle)//rotation from axis1 to
axis2 by angle
    {
        DMatrix res = DMatrix.Get1Matrix(dimnum);
        res[axis1, axis1] = Math.Cos(angle);
        res[axis2, axis2] = Math.Cos(angle);
        res[axis1, axis2] = -Math.Sin(angle);
        res[axis2, axis1] = Math.Sin(angle);
        return res;
    }
    public static DMatrix GetDiagMatrix(double[] diagelems)
    {
        DMatrix res = DMatrix.Get1Matrix(diagelems.Length);
        for (int i = 0; i < res.DimNumber; i++)
            res[i, i] = diagelems[i];
        return res;
    }
    public static DMatrix GetDiagMatrix(DVector diagelems)
    {
        DMatrix res = DMatrix.Get1Matrix(diagelems.DimNumber);
        for (int i = 0; i < res.DimNumber; i++)
            res[i, i] = diagelems[i];
    }
}

```

```

        return res;
    }
    public static DMatrix GetScaleMatrix(int dimnum, int dim, double scale)
    {
        DMatrix res = DMatrix.Get1Matrix(dimnum);
        for (int i = 0; i < res.DimNumber; i++)
            res[i, i] = i==dim?scale:1;
        return res;
    }
    public static DMatrix operator *(DMatrix a, DMatrix b)//matrix multiplication
    {
        if (a.DimNumber != b.DimNumber) throw new ArgumentException();
        DMatrix res = new DMatrix(a.DimNumber);
        for(int row=0;row<a.DimNumber;row++)
            for (int col = 0; col < a.DimNumber; col++)
            {
                double sum=0;
                for (int i = 0; i < a.DimNumber; i++) sum += a[row, i] * b[i, col];
                res[row, col] = sum;
            }
        return res;
    }
    public static DVector operator *(DMatrix a, DVector b)//matrix*vector
    {
        if (a.DimNumber != b.DimNumber) throw new ArgumentException();
        double[] res = new double[a.DimNumber];
        for (int i = 0; i < a.DimNumber; i++) res[i] = a.GetRow(i) * b;
        return new DVector(res);
    }
    public static DMatrix operator *(DMatrix a, double b)//multiplication by scalar
    {
        DMatrix res = new DMatrix(a.DimNumber);
        for (int i = 0; i < a.DimNumber; i++)
            for (int j = 0; j < a.DimNumber; j++)
                res[i, j] = a[i, j] * b;
        return res;
    }
    public override string ToString()
    {
        string res = "";
        for (int i = 0; i < DimNumber; i++)
            for (int j = 0; j < DimNumber; j++)
                res += coordinates[i,j] + " ";
        return res.Remove(res.Length - 1);
    }
}
}

```

```

Space.cs:
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Media;
using System.Windows;

namespace MultiDimEditor
{
    class Space
    {
        public List<DVector> points{get; private set;}
        public List<Edge> edges { get; private set; }
        public List<Face> faces { get; private set; }
        public List<Transform> views { get; private set; }
        public int dimnum { get; private set; }
        public Space(int dimnum)
        {
            points = new List<DVector>();
            edges = new List<Edge>();
            faces = new List<Face>();
            views = new List<Transform>();
            this.dimnum = dimnum;
        }
        public static Space Parse(string content)
        {

```

```

string[] lines = content.Split('|');
int dimnum = int.Parse(lines[0]);
int pointnum = int.Parse(lines[1]);
int curstr = 2;
List<DVector> points = new List<DVector>();
for(int i=0;i<pointnum;i++)
{
    string[] strcoords = lines[curstr++].Split(' ');
    double[] coords = strcoords.Select(x => double.Parse(x)).ToArray();
    points.Add(new DVector(coords));
}
int edgenum = int.Parse(lines[curstr++]);
List<Edge> edges = new List<Edge>();
for (int i = 0; i < edgenum; i++)
{
    string[] strpoints = lines[curstr++].Split(' ');
    int p1 = int.Parse(strpoints[0]);
    int p2 = int.Parse(strpoints[1]);
    edges.Add(new Edge(p1, p2));
}
int facenum = int.Parse(lines[curstr++]);
List<Face> faces = new List<Face>();
for (int i = 0; i < facenum; i++)
{
    string[] strfpoints = lines[curstr++].Split(' ');
    int[] fpoints = strfpoints.Select(x => int.Parse(x)).ToArray();
    faces.Add(new Face(fpoints));
}
Space s = new Space(dimnum);
s.points = points;
s.edges = edges;
s.faces = faces;
return s;
}
public override string ToString()
{
    string res = "";
    res += dimnum+"|";
    res += points.Count+"|";
    foreach (var p in points) res += p + "|";
    res += edges.Count + "|";
    foreach (var e in edges) res += e + "|";
    res += faces.Count + "|";
    foreach (var f in faces) res += f + "|";

    res.Remove(res.Length - 1);
    return res;
}
public void Draw(DrawingContext dc, Point center, Transform view, DrawingRules dr)
{
    List<Point> _2dpoints=new List<Point>();
    foreach(var p in points)
    {
        var newp = view.ApplyTo(p);
        _2dpoints.Add(newp.ConvertTo2D(center));
    }
    foreach (var p in _2dpoints) dc.DrawEllipse(dr.PointBrush, null, p, dr.PointWidth, dr.PointWidth);
    foreach (var e in edges) dc.DrawLine(dr.LinePen, _2dpoints[e.point1], _2dpoints[e.point2]);
    foreach(var f in faces)
    {
        PathFigure pf=new PathFigure();
        for(int i=1;i<f.points.Count;i++)
        {
            pf.Segments.Add(new LineSegment(_2dpoints[f.points[i]], true));
        }
        pf.StartPoint=_2dpoints[f.points[0]];
        pf.IsClosed = true;
        PathGeometry g = new PathGeometry(new[]{pf});
        dc.DrawGeometry(dr.FaceBrush, null, g);
    }

    //coordinate lines
    //base
    dc.DrawLine(dr.BaseCoordLinePen, center, new Point(1000+center.X, center.Y));
    dc.DrawLine(dr.BaseCoordLinePen, center, new Point(center.X, center.Y - 1000));
    //orig

```

```
        for(int i=0;i<dimnum;i++)
        {
            dc.DrawLine(dr.CoordLinePen, (view.ApplyTo(new DVector(dimnum)).ConvertTo2D(center)),
            (view.ApplyTo(1000 * DVector.GetOrt(i, dimnum)).ConvertTo2D(center)));
        }
    }
}
```



## ДОДАТОК Б (Детальна діаграма класів)

