ЗМІСТ

ТЕОРЕТИЧНА ЧАСТИНА

1. ТЕХНІЧНЕ ЗАВДАННЯ

1.1. Підстави для розробки

Вступ

Разом із плином загальної комп’ютеризації, дедалі більше галузей людської діяльності стають пов’язаними, так чи інакше, з комп’ютерним моделюванням. Архітектура, інженерія, кінематограф, гейм дизайн – це лише декілька із десятків сфер, для яких в сучасному світі невід’ємною частиною є робота з 3D-об’єктами.

Представлення довільних моделей у тривимірному просторі майже завжди може бути задачею неоднозначною: адже квадрат можна задати двома трикутниками, або ж трикутник – двома меншими; а форма збережених даних про модель може фундаментально відрізнятися навіть від джерела їх отримання: чи то розпізнавання образів по фотознімках у поєднанні з алгоритмами тріангуляції, чи то захоплення рухів кіноактора за допомогою сенсорних точок на спеціальному костюмі, чи то рисування макету або плану будинку в архітектурно орієнтованій САП. Вже тільки з цих причин виникає проблема вибору формату збереження даних про змодельований об’єкт.

САП (система автоматизованого проектування) – автоматизована система, призначена для автоматизації технологічного процесу проектування виробу, результатом якого є комплект проектно-конструкторської документації, достатньої для виготовлення та подальшої експлуатації об'єкта проектування. Тенденцією для сучасного ринку САП програм стало запровадження чималої кількості власних комерційних форматів файлів, що разом із невеликою кількістю форматів з відкритою специфікацією (COLLADA, glTF, STL, OBJ, Universal 3D) утворило проблему обміну даними про 3D-модель в різних системах та між різним програмним забезпеченням.

У сфері машинобудування та суміжних галузей найбільш зручним форматом для зберігання даних про 3D-моделі можна вважати 3D PDF[2]. В порівнянні з деякими іншими форматами, зокрема STEP, 3D XML та JT, варто відмітити найбільшу ефективність 3D PDF у наступних областях застосування:

- ✓ перегляд технічних даних;
- ✓ документування та архівування;
- ✓ портативний документ для використання в неінженерних сферах.

Назву 3D PDF носить формат PDF, що забезпечує вбудовану підтримку 3D. 3D PDF може зберігати CAD-геометрію в наступних представленнях:

- PRC (Project Reviewer Compressed);
- U3D (Universal 3D).

Перевагою U3D у порівнянні з PRC є відкритість а також вища ступінь довготривалого архівування даних через свою універсальну форму представлення даних, яка не залежна від поточних особливостей функцій жодної САП.

Можливість легкого вбудовування в PDF документ об'єкта U3D відкриває потребу в засобах швидкого та простого створення моделей цього формату, їх конвертації між іншими типами файлів. Причому максимальна універсальність кінцевої реалізації такого інструменту роботи з U3D дозволить в майбутньому розробити як надбудови до нього самого (наприклад редактор U3D моделі з користувацьким інтерфейсом та сценою відображення засобами OpenGL), так і програми-конвертери з іншими форматами.

На практиці, такий інструмент роботи з U3D реалізується просто: потрібно створити бібліотеку класів, що дають можливість маніпулювати компонентами і типами цього формату згідно специфікації [1]. Загалом, основним класом в такій структурі повинен бути такий, що представляє власне U3D документ. Він має надавати можливість зберігання даних про 3D-об'єкти та збереження документу.

Актуальність задачі

Актуальність бібліотеки для роботи з файлами формату Universal 3D пов'язана із великою кількістю комерційних й відкритих 3D форматів, і потребою конвертації їх даних про тривимірні моделі в форму представлення, що забезпечить швидкий та доступний перегляд а також оптимальне зберігання та архівування.

Подібні завдання є важкими для здійснення з використанням будь-якого закритого 3D формату через проблему дотримання авторських прав, а також

покупку власне документації цього комерційного типу для подальшої розробки необхідних інструментів.

Обрання, для вирішення цих актуальних проблем, інших відкритих форматів файлів таких як OBJ чи STL є недоречним через їх відсутність підтримки анімацій. Формат COLLADA та glTF максимально схожі за можливостями з U3D, однак перший не можна легко переглядати, а другий програє Universal 3D у компактному розмірі файлу при стисненому записі.

Стан проблематики

У мережі можна знайти доволі малу кількість програмного забезпечення що вміє працювати з «.u3d» файлами.

Проаналізувавши знайдені платні та вільні програми, можна виділити ряд недоліків:

- відсутність можливості читання U3D документа;
- створення U3D можливе тільки в візуальному режимі програми;
- відсутність коректного конвертування з іншими форматами.

Для вирішення вищеперерахованих недоліків застосовано наступні кроки в реалізації бібліотеки:

- додано інструмент читання U3D документа;
- структура бібліотеки, зокрема класу, що представляє документ U3D, дозволяє створювати 3D-об’єкт в режимі програмного коду;
- концепція представлення документа, дозволяє реалізувати конвертацію в довільний формат, попередньо реалізувавши клас-посередник що буде обмінювати дані між різними типами файлів.

У результаті, отримаємо бібліотеку класів для роботи з файлами формату Universal 3D в режимі програмного коду, яка дозволятиме їх створення та збереження в U3D чи PDF за вибором.

Розробка програмного забезпечення виконується на підставі рішення засідання кафедри МПУіК про затвердження тем дипломних та курсових робіт (протокол № 2 від «06» *вересня* 2018 року).

1.2. Призначення розробки

Призначенням даної розробки є створення бібліотеки для роботи з файлами формату Universal 3D. Класи бібліотеки утворюють інструмент, що дозволяє додавати в документ компоненти 3D-об’єкта, а також зберігати створений документ у наступних форматах на вибір: U3D або 3D PDF.

1.3. Аналіз вимог до програмного забезпечення

1.3.1. Функціональні вимоги

До програмного забезпечення висуваються наступні вимоги:

- класи, необхідні для опису компонентів документа U3D;
- можливість додавання матеріалів;
- можливість додавання текстур;
- можливість додавання шейдерів;
- можливість додавання полігональних сіток;
- можливість додавання вузлів ієрархії моделі;
- можливість збереження документа в U3D форматі;
- можливість збереження документа в PDF форматі;
- інтуїтивно зрозуміла структура бібліотеки.

1.3.2. Вимоги до складу та параметрів технічних засобів

Для нормальної роботи розробленого програмного продукту потрібен комп’ютер/ноутбук із наступними мінімальними характеристиками:

Параметр	Значення
Процесор	Intel Core 2 Duo
ОЗП	2048 МБ
Мінімальна версія	Windows 7
Пам'ять	4,5 Гб

1.3.3. Вимоги до інтерфейсу

Інтерфейс бібліотеки представляє її структура, тому вона має бути зрозумілою для розробника програмного продукту, що використовує бібліотеку.

1.3.4. Вимоги до інформаційної та програмної сумісності

Вимоги до ПК або ноутбуків:

- відповідність мінімальним технічних вимогам;
- встановлена ОС Microsoft Windows 7, 8, 8.1 або 10 та відповідні актуальні оновлення;
- встановлене середовище Microsoft Visual Studio 2017 або вище, з компонентом розробки .NET C#.

1.3.5. Вимоги до тестування програмного забезпечення

Для тестування програмного забезпечення необхідно виконати наступні дії:

1. Відкрити «Samples.sln» за допомогою Microsoft Visual Studio.
2. Запустити на виконання проект «Textured_Rubik_s_Cube».
3. Перейти до каталогу «.\bin\Debug\netcoreapp2.0\» в папці проекту.
4. Відкрити файл «Textured_Rubik_s_Cube.u3d», наприклад за допомогою SAP 3D Visual Enterprise Viewer 9.0, та перевірити відображення 3D моделі.
5. Відкрити файл «Textured_Rubik_s_Cube.pdf», наприклад за допомогою Adobe Acrobat Reader DC, та перевірити відображення 3D моделі.

1.4. Вимоги до програмної документації

Програмне забезпечення постачається разом із супроводжувальною документацією в склад якої входить:

1. Технічне завдання.
2. Опис та обґрунтування обраної архітектури.
3. Функціональна специфікація.
4. Технічна специфікація.
5. Опис програми.
6. Програма та методика випробувань.

1.5. Стадії та етапи розробки

Таблиця 1.5.1

Стадії та етапи розробки

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Одержання технічного завдання.	15.12.18р.
2	Аналіз літератури.	21.12.18р.
3	Розробка інтерфейсу.	06.01.19р.
4	Реалізація введення не продуктивного часу.	12.01.19р.
5	Реалізація введення задач.	20.01.19р.
6	Реалізація аналізу наявних задач.	16.02.19р.
7	Реалізація виведення побудованого плану.	26.03.19р.
8	Тестування та налагодження програми.	11.04.19р.
9	Оформлення програмної документації.	11.04.19р.
10	Представлення готової роботи.	20.04.19р.
11	Захист роботи.	згідно розкладу

1.6. Порядок контролю і приймання

Програмне забезпечення повинне бути випробуване та протестоване виробником на наявність помилок.

Порядок контролю і приймання проекту полягає у наступному:

1. Перевірка правильного функціонування програми керівником проекту.
2. Перевірка відповідності стандартам оформлення документації.
3. Перевірка правильного оформлення програмної документації керівником проекту.
4. Проходження нормоконтролю роботи.
5. Попередній захист курсової роботи.

ПРАКТИЧНА ЧАСТИНА

2. АРХІТЕКТУРА, ФУНКЦІОНАЛЬНІ ТА ТЕХНІЧНІ ПОКАЗНИКИ

2.1. Призначення та область застосування

Призначення даної розробки полягає у наданні зручного інструменту для роботи з файлами формату Universal 3D в режимі програмного коду. Зокрема це бібліотека класів що дає змогу формувати та зберігати відповідний документ в форматі U3D або ж PDF, що містить попередній.

Можливість легкого вбудовування в PDF документ об’єкта U3D відкриває потребу в засобах швидкого та простого створення моделей цього формату, їх конвертації між іншими типами файлів. Причому максимальна універсальність кінцевої реалізації такого інструменту роботи з U3D дозволить в майбутньому розробити як надбудови до нього самого (наприклад редактор U3D моделі з користувацьким інтерфейсом та сценою відображення засобами OpenGL), так і програми-конвертери з іншими форматами.

2.2. Опис та обґрунтування обраної архітектури

Для даного проекту було обрано об’єктно-орієнтовану архітектуру, мову програмування C#, платформу .NET Standard 2.0, відповідно використовувалося середовище Visual Studio 2017. Підключено бібліотеку FreeSpire.PDF.

Об’єктно-орієнтована архітектура дає змогу проводити абстрагування від деталей реалізації. Відмінністю від процедурного підходу є те, що операції описуються разом та утворюють певну сукупність, не розповсюджуючись по програмному коді. Однією з найбільших переваг ООП є можливість створення систем, що розгортаються. Це означає що систему можна доповнити новими модулями і компонентами без внесення змін до попередніх. Багаторазове використання знижує ймовірність помилок, в зв’язку з численною перевіркою. Варто зазначити наступні переваги даного підходу:

- ✓ зменшення часу на розробку, який можна використати на інші завдання;

- ✓ безліч стандартних компонентів, які можна використовувати для полегшення розуміння і спрощення програми;
- ✓ заміна об’єктів на етапі виконання програми дозволяє адаптувати алгоритм для відповідного об’єкту;
- ✓ наслідування дозволяє узагальнити алгоритми для роботи із декількома видами об’єктів.

.NET Standard являє собою офіційну специфікацію інтерфейсів API .NET, які мають бути у всіх реалізаціях .NET. .NET Standard створена для того, щоб підвищити узгодженість екосистем .NET. ECMA-335 [3] продовжує забезпечувати однорідність для реалізації .NET, але аналогічні специфікації для бібліотек базових класів (BCL) .NET для реалізації бібліотек .NET відсутні.

.NET Standard надає наступні важливі можливості:

- визначає уніфікований набір API-інтерфейсів BCL для реалізації всіма реалізаціями .NET незалежно від робочого навантаження;
- дозволяє розробникам створювати переносні бібліотеки, які можуть використовуватися в різних реалізаціях .NET, за допомогою одного набору API-інтерфейсів;
- дозволяє скоротити або навіть прибрати умовну компіляцію загального джерела через API-інтерфейси .NET (тільки для API операційної системи).

Різні реалізації .NET реалізують конкретні версії .NET Standard. Кожна версія реалізації .NET орієнтована на використання максимальної підтримуваної нею версії .NET Standard. Це також означає, що вона підтримує і попередні версії.

Наприклад, платформа .NET Framework 4.6 реалізовує .NET Standard 1.3, тобто надає всі API-інтерфейси визначені в стандартах .NET Standard версії з 1.0 по 1.3. Аналогічним чином платформа .NET Framework 4.6.1 реалізовує .NET Standard 1.4, а .NET Core 1.0 – .NET Standard 1.6.

FreeSpire.PDF – це бібліотека для .NET, що дозволяє розробникам створювати, писати, редагувати, конвертувати, друкувати, обробляти і читати PDF-файли на будь-яких .NET-програмах. Пакет надає широкі можливості для

створення PDF-файлів з нуля або обробки існуючих документів PDF. Підтримується великий діапазон функцій:

• перетворення HTML-сторінки веб-сторінки, HTML ASPX в PDF;

• перетворення зображення (JPEG, JPG, PNG, BMP, TIFF, GIF, EMF, ICO) на PDF;

• перетворення тексту в PDF;

• перетворення RTF на PDF;

• перетворення PDF на зображення.

Легко можна маніпулювати полями документів і форм:

• об'єднати / розділити PDF-документи;

• накладання документів;

• функція імпорту та штампу;

• функція буклету;

• створити і заповнити поле форми.

Функції безпеки:

• захист документів PDF шляхом встановлення паролів та цифрового підпису;

• розшифрувати документ PDF;

• отримати та перевірити цифровий підпис;

• зміна паролів PDF.

Функції параметрів документа:

• встановлення користувацьких метаданих, властивостей документа, орієнтації сторінки та розміру сторінки;

• встановити положення PDF, відображення заголовка, зміна розміру, режим сторінки та масштабування друку тощо.

Інші особливості:

• відповідність стандарту PDF / A-1b та PDF / x1a: 2001 - можна застосовувати обидва стандарти;

• видобування зображень, тексту, сторінок і вкладень з PDF-документа;

• ручка та кисть для малювання елементів форми, тексту, зображень у PDF-документ;

• шари, прозора графіка, колірний простір і створення штрих-кодів можуть відображати документи PDF;

• додайте скалярні / векторні зображення та маску та розмістіть їх у визначеному місці;

• стиль таблиці та таблиці може підтримуватися FreeSpire.PDF для .NET;

• вставка інтерактивних елементів.

2.3. Функціональна специфікація

2.3.1. Опис функціональних можливостей

Розроблена бібліотека дає можливість створювати U3D документ, додаючи в створений екземпляр відповідного об’єкта такі елементи як: матеріали, текстури, шейдери, сітки полігонів та вузли ієрархії моделі. Всі ці компоненти дають можливість побудувати будь-яку 3D-модель, або конвертувати відповідні дані із файлу іншого формату.

Кожен компонент документа повинен ідентифікуватися за ім’ям всередині свого типу. Тобто неможна допускати двох матеріалів або текстур з однаковим ім’ям. Це вимога специфікації формату Universal 3D, і дотримана вона за рахунок реалізації методів TryAdd класу U3DDocument.

2.3.2. Опис інтерфейсу користувача

Робота з бібліотекою передбачає використання розроблених класів у довільних програмних продуктах – як консольних, так і програмах з візуальним інтерфейсом користувача.

Після збереження побудованого документу засобами бібліотеки, можна відкрити збережений U3D або PDF файл, і переглянути 3D-об’єкт та його ієрархію вузлів.

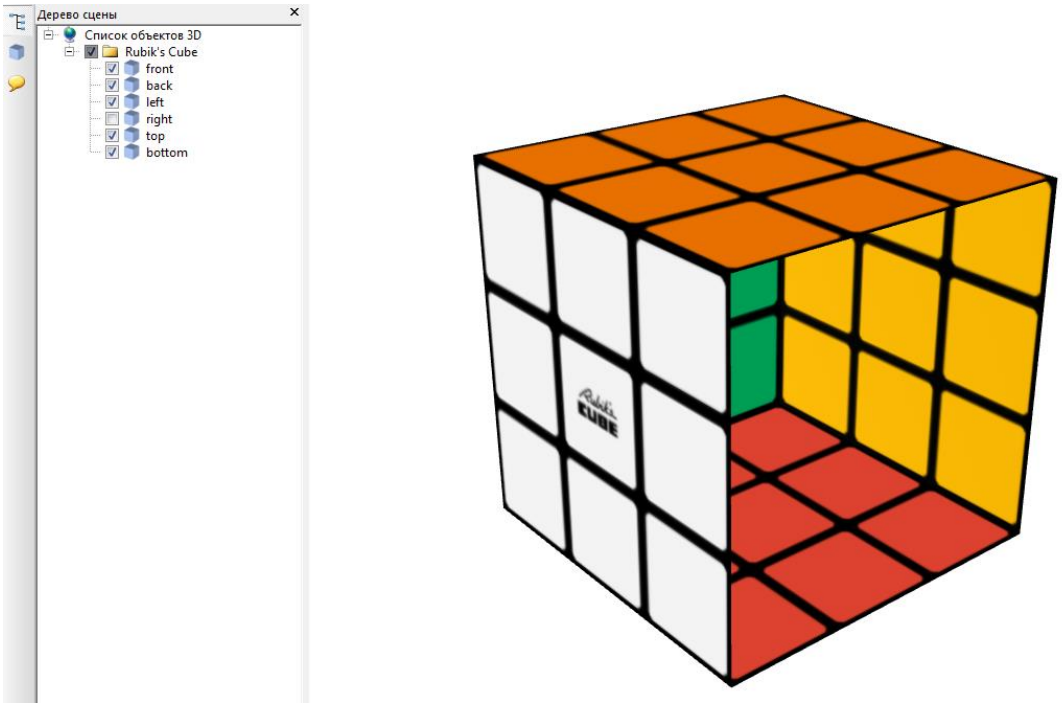

Рис. 2.3.1. Побудований засобами бібліотеки кубик Рубіка із текстурами на гранях. Праву грань приховано через інструмент «Дерево сцени» переглядача

2.4. Технічна специфікація

2.4.1. Опис модулів

Робота розробленого програмного забезпечення реалізується наступними файлами класами:

- U3DDocument – клас, в якому реалізовано представлення документа Universal 3D з основними компонентами для побудови 3D-об’єкта;
- Merger – статичний клас, що дозволяє об’єднувати два об’єкти класу U3DDocument в один, з необхідними трансформаціями масштабування, повороту і зсуву;
- Textured_Rubik_s_Cube – статичний клас, який містить тільки один публічний метод, що повертає екземпляр класу U3DDocument і являє собою зразок використання бібліотеки;
- Shader – клас, що представляє шейдер, який містить ідентифікатор (ім’я) матеріалу обов’язково, а також може містити ідентифікатор текстури;
- Texture – представляє текстуру;
- ImageFormat – тип-перерахування, який є допоміжним для класу Texture;
- Matrix4 – клас, що представляє матрицю розміру 4x4;
- Node – клас, що представляє вузол ієрархії елементів 3D-моделі;
- Vector3 – клас, що представляє тривимірний вектор;
- Vector2 – клас, що представляє двовимірний вектор;
- Corner – клас, що представляє кут грані (полігона, трикутника) і містить індекс вершини, нормалі, та текстурної координати;
- Triangle – клас, що представляє грань (полігон);
- Mesh – клас, що представляє полігональну сітку (масив граней);
- Color – клас, що представляє колір з компонентами RGB;
- Material – клас, що представляє матеріал;
- Block – клас, що представляє блок даних згідно специфікації формату U3D, містить дані та метадані;

- BlockType – тип-перерахування, що зберігає заголовки використовуваних блоків даних;
- BlockReader – клас, що дає можливість читати дані із класу Block;
- BlockWriter – клас, що дає можливість записувати дані в клас Block;
- DocumentReader – клас, що дає можливість читати байтові дані і записувати їх в екземпляр класу U3DDocument;
- DocumentWriter – клас, що дає можливість сформований екземпляр класу U3DDocument записувати у потік даних або файл як в форматі U3D, так і PDF.

2.4.2. Опис і обґрунтування вхідних і вихідних даних

Вхідними даними для розробленої бібліотеки класів можуть бути залежно від сценарію використання або файли u3d, що необхідно прочитати, об’єднати чи змінити; або дані про 3D-модель із якої необхідно сформувати U3DDocument для подальшої маніпуляції.

Вихідними даними можуть виступати або екземпляри класу U3DDocument з компонентами його наповнення, або збережені на диск чи в потік даних документи у форматі U3D або PDF.

3. КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1. Опис і обґрунтування обраних програмних засобів

Для реалізації бібліотеки класів використано мову програмування C# та середовище Microsoft Visual Studio 2017 Community.

Microsoft Visual Studio – це інтегроване середовище розробки із широкими можливостями для створення програм для ОС Windows, Android, iOS, а також сучасних додатків для вебу і хмарних сервісів. VS містить спеціальне середовище розробки із графічним інтерфейсом характерним для Windows-програм, що дозволяє додавати нові функціональні елементи, змінювати їх розмір, розташування та спроектувати інтуїтивно зрозумілий інтерфейс. Виділяють наступні основні елементи:

- ✓ панель інструментів із командами для роботи у середовищі;
- ✓ оглядач рішень, що відображає проекти і файли, що стосуються проекту;
- ✓ панель елементів, що використовується для побудови рішень;
- ✓ панель властивостей для зміни об’єктів.

Дане середовище має ряд переваг:

- ✓ підтримує декілька мов програмування;
- ✓ забезпечує безпечне передавання даних;
- ✓ забезпечує легке перенесення на інші платформи, завдяки технології CLR;
- ✓ дозволяє реалізувати програми зі зручним інтерфейсом;
- ✓ володіє можливістю встановлення розширень;
- ✓ містить безліч бібліотек та стандартних компонентів, що не лише пришвидшують роботу програми, а й полегшують процес написання коду.

3.2. Опис програми

3.2.1. Функціональні можливості

У створеному програмному забезпеченні розроблені такі функції:

– створення екземпляра класу U3DDocument із необхідних вхідних даних в режимі програмного коду, при цьому забезпечивши коректність документу перевіркою компонентів при додаванні;
– коректне збереження документу у файл, який пізніше можна відкрити наприклад в Adobe Acrobat Reader, якщо файл було збережено в форматі PDF;
– об’єднання двох документів із всіма необхідними перетвореннями – масштабування, поворот і зсув;
– читання всіх коректних даних із файлу в екземпляр класу U3DDocument.

3.2.2. Опис логічної структури

Структуру даного програмного забезпечення зручно подати у вигляді дерева об’єктів середовища Visual Studio (рис.3.2.1):

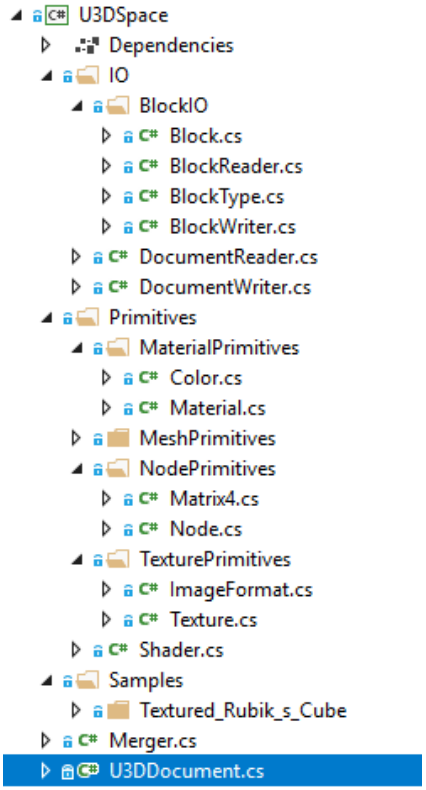

Рис. 3.2.1. Структура розробленої бібліотеки класів

Розглянемо приклад використання бібліотеки для створення 3D-моделі «Кубик Рубіка», на грані якого нанесено підготовлені текстури.

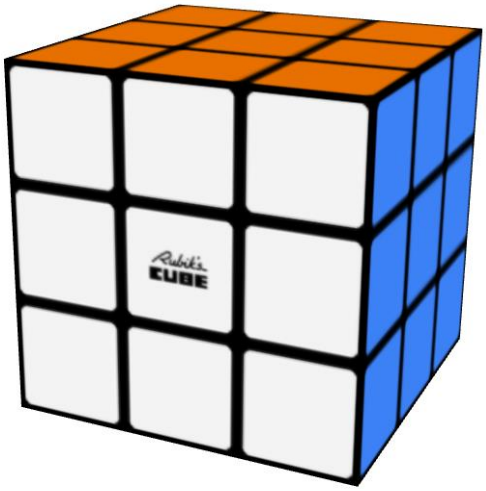

Рис. 3.2.2. Кубик Рубіка із текстурами на гранях

Описуємо методи, що будуть формувати матеріали, текстури і рейдери (Рис. 3.2.3), сітки (Рис. 3.2.4) та вузли (Рис. 3.2.5).

```
private static byte[] GetImageFromResources(string imageName)
{
    Assembly assembly = Assembly.GetExecutingAssembly();
    using (Stream stream = assembly.GetManifestResourceStream(string.Format("U3DSpace.Samples.Textured_Rubik_s_Cube.Images.{0}", imageName)))
    {
        var data = new byte[stream.Length];
        stream.Read(data, 0, (int)stream.Length);
        return data;
    }
}

private static void AddTextures(U3DDocument doc)
{
    doc.TryAddTexture(new Texture("front_texture", ImageFormat.PNG, GetImageFromResources("front.png")));
    doc.TryAddTexture(new Texture("back_texture", ImageFormat.PNG, GetImageFromResources("back.png")));
    doc.TryAddTexture(new Texture("left_texture", ImageFormat.PNG, GetImageFromResources("left.png")));
    doc.TryAddTexture(new Texture("right_texture", ImageFormat.PNG, GetImageFromResources("right.png")));
    doc.TryAddTexture(new Texture("top_texture", ImageFormat.PNG, GetImageFromResources("top.png")));
    doc.TryAddTexture(new Texture("bottom_texture", ImageFormat.PNG, GetImageFromResources("bottom.png")));
}

private static void AddMaterials(U3DDocument doc)
{
    doc.TryAddMaterial(new Material("default_material", new Color(0.1), new Color(1.0), new Color(0.5), new Color(0.0), 1.0f, 1.0f));
}

private static void AddShaders(U3DDocument doc)
{
    doc.TryAddShader(new Shader("front_shader", "default_material", "front_texture"));
    doc.TryAddShader(new Shader("back_shader", "default_material", "back_texture"));
    doc.TryAddShader(new Shader("left_shader", "default_material", "left_texture"));
    doc.TryAddShader(new Shader("right_shader", "default_material", "right_texture"));
    doc.TryAddShader(new Shader("top_shader", "default_material", "top_texture"));
    doc.TryAddShader(new Shader("bottom_shader", "default_material", "bottom_texture"));
}
```

Рис. 3.2.3. Методи для опису даних про матеріали, текстури та шейдери

```
private static void AddMeshes(U3DDocument doc)
{
    var triangles = new List<Triangle> { new Triangle(new Corner(0, 0, 0), new Corner(1, 0, 1), new Corner(2, 0, 2)), new Triangle(new Corner(2, 0, 2),
    var textureCoordinates = new List<Vector2> { new Vector2(0, 0), new Vector2(0, 1), new Vector2(1, 1), new Vector2(1, 0) };
    doc.TryAddMesh(new Mesh(
        "front_mesh", "front_shader",
        new List<Vector3> { new Vector3(-1, -1, 1), new Vector3(-1, 1, 1), new Vector3(1, 1, 1), new Vector3(1, -1, 1) },
        new List<Vector3> { new Vector3(0, 0, 1) },
        textureCoordinates,
        triangles
        ));
    doc.TryAddMesh(new Mesh(
        "back_mesh", "back_shader",
        new List<Vector3> { new Vector3(1, -1, -1), new Vector3(1, 1, -1), new Vector3(-1, 1, -1), new Vector3(-1, -1, -1) },
        new List<Vector3> { new Vector3(0, 0, -1) },
        textureCoordinates,
        triangles
        ));
    doc.TryAddMesh(new Mesh(
        "left_mesh", "left_shader",
        new List<Vector3> { new Vector3(-1, -1, -1), new Vector3(-1, 1, -1), new Vector3(-1, 1, 1), new Vector3(-1, -1, 1) },
        new List<Vector3> { new Vector3(-1, 0, 0) },
        textureCoordinates,
        triangles
        ));
    doc.TryAddMesh(new Mesh(
        "right_mesh", "right_shader",
        new List<Vector3> { new Vector3(1, -1, 1), new Vector3(1, 1, 1), new Vector3(1, 1, -1), new Vector3(1, -1, -1) },
        new List<Vector3> { new Vector3(1, 0, 0) },
        textureCoordinates,
        triangles
        ));
    doc.TryAddMesh(new Mesh(
        "top_mesh", "top_shader",
        new List<Vector3> { new Vector3(-1, 1, 1), new Vector3(-1, 1, -1), new Vector3(1, 1, -1), new Vector3(1, 1, 1) },
        new List<Vector3> { new Vector3(0, 1, 0) },
        textureCoordinates,
        triangles
        ));
    doc.TryAddMesh(new Mesh(
        "bottom_mesh", "bottom_shader",
        new List<Vector3> { new Vector3(-1, -1, -1), new Vector3(-1, -1, 1), new Vector3(1, -1, 1), new Vector3(1, -1, -1) },
        new List<Vector3> { new Vector3(0, -1, 0) },
        textureCoordinates,
        triangles
        ));
}
```

Vector3.Vector3(float x, float y, float z) (+ 5 overloads)

Рис. 3.2.4. Метод для опису даних про полігональні сітки

```
private static void AddNodes(U3DDocument doc)
{
    string rootNodeName = "Rubik's Cube";
    doc.TryAddNode(new Node(rootNodeName, null, null, Matrix4.GetIdentityMatrix()));
    doc.TryAddNode(new Node("front", "front_mesh", rootNodeName, Matrix4.GetIdentityMatrix()));
    doc.TryAddNode(new Node("back", "back_mesh", rootNodeName, Matrix4.GetIdentityMatrix()));
    doc.TryAddNode(new Node("left", "left_mesh", rootNodeName, Matrix4.GetIdentityMatrix()));
    doc.TryAddNode(new Node("right", "right_mesh", rootNodeName, Matrix4.GetIdentityMatrix()));
    doc.TryAddNode(new Node("top", "top_mesh", rootNodeName, Matrix4.GetIdentityMatrix()));
    doc.TryAddNode(new Node("bottom", "bottom_mesh", rootNodeName, Matrix4.GetIdentityMatrix()));
}
```

Рис. 3.2.5. Метод для опису даних про вузли ієрархії елементів моделі

Записані методи потрібно використати в правильному порядку (Рис. 3.2.6), адже бібліотека не дозволить додати шейдер, що посилається на ще не доданий в документ матеріал.

```
public static U3DDocument GetDocument()
{
    var doc = new U3DDocument();
    AddTextures(doc);
    AddMaterials(doc);
    AddShaders(doc);
    AddMeshes(doc);
    AddNodes(doc);
    return doc;
}
```

Рис. 3.2.5. Використання описаних методів для формування U3D документа

Створений U3D документ можна зберегти в двох форматах: U3D та PDF.

```
public static void Main(string[] args)
{
    U3DDocument doc = U3DSpace.Samples.Textured_Rubik_s_Cube.GetDocument();

    string u3dFileName = "Textured_Rubik_s_Cube.u3d";
    doc.SaveToFile(u3dFileName);
    Console.WriteLine(@"Successfully saved: {0}\{1}", Environment.CurrentDirectory, u3dFileName);

    string pdfFileName = "Textured_Rubik_s_Cube.pdf";
    doc.SaveToPdfFile(pdfFileName);
    Console.WriteLine(@"Successfully saved: {0}\{1}", Environment.CurrentDirectory, pdfFileName);

    Console.ReadKey();
}
```

Рис. 3.2.6. Збереження U3D документа на диск

3.2.3. Використані технічні засоби

Розробка програмного продукту здійснювалась на персональному комп’ютері наступної конфігурації:

1. Процесор – Intel(R) Core(TM) i5-2520M CPU @ 2.50 GHz.
2. ОЗП – 8 Гб.
3. Відеоадаптер – Intel HD Graphics 3000 (32 Мб).
4. SSD – GoodRam CX300 (240GB, SATA III).

Даний комп’ютер працює під ОС Windows 10 Pro 1809.

4. ПРОГРАМА ТА МЕТОДИКА ВИПРОБУВАНЬ

4.1. Об’єкт випробувань

Об’єктом випробування є скомпільований .dll файл бібліотеки, який додається до посилань консольного проекту Visual Studio.

4.2. Використані технічні засоби

Тестування проводилось на пристроях з наступною конфігурацією:

Параметри	Пристрій №1	Пристрій №2
Процесор	Intel Core i5 2520M 2.5 Ghz	Intel Celeron 1.9 Ghz
ОЗП	8192 Мб	2048 Мб
Версія ОС	Windows 10 Pro 1809	Windows 7 Home Premium
Пам'ять	240 ГБ	500 ГБ
Екран	12.5’’ (1366 x 768)	15.6’’ (1366 x 768)

4.3. Порядок та методика випробувань

Порядок проведення випробувань програмного забезпечення:

1) Створення екземпляра класу U3DDocument.
2) Додавання матеріалів та текстур у документ в довільному порядку.
3) Послідовне додавання шейдерів, полігональних сіток і вузлів.
4) Збереження зразка «Textured_Rubik_s_Cube» у .u3d та .pdf файли.
5) Перегляд коректності збереженого результату.

4.4. Результати випробувань

Випробування програмного продукту проводилось на двох різних пристроях на базі ОС Windows. При проведенні випробувань серйозних недоліків не виявлено, увесь заявлений функціонал працює належним чином.

ВИСНОВКИ

У сфері машинобудування та суміжних галузей найбільш зручним форматом для зберігання даних про 3D-моделі можна вважати 3D PDF[2]. Можливість легкого вбудовування в PDF документ об’єкта U3D відкриває потребу в засобах швидкого та простого створення моделей цього формату, їх конвертації між іншими типами файлів.

У результаті виконання курсової роботи розроблена бібліотека для роботи з файлами формату Universal 3D.

Програма доволі функціональна та зручна у використанні.

Тестування програмного забезпечення показало його працездатність і відповідність технічним вимогам.

Розроблене програмне забезпечення має наступні можливості:

1. Створення U3D документу із вхідних даних про 3D-модель;
2. Читання даних із U3D файлу;
3. Об’єднання двох U3D документів із трансформацією геометрії;
4. Можливість простого обміну даними з іншими 3D-форматами завдяки зручній структурі представлення документу;
5. Збереження документу в двох форматах на вибір: U3D та PDF.

В даній роботі використовуються FreeSpire.PDF – це пакет інструментів для розробників, призначення якого полягає в роботі з файлами формату PDF. В цій роботі він використовується для запаковування, в стиснутому форматі, файлу U3D в PDF документ.

Дана розробка в майбутньому може бути розширена додаванням нового функціоналу і вдосконаленням поточного. Проведені усі необхідні роботи по проектуванню архітектури бібліотеки, проаналізовані вимоги до неї, приводиться опис реалізації, кодування, тестування програми.

ВИКОРИСТАНА ЛІТЕРАТУРА

1. ECMA-363. Universal 3D File Format [Електронний ресурс] // ECMA International. – 2004. – Режим доступу до ресурсу: https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-363%201st%20edition.pdf.
2. Dr. Arnulf Frohlich. Сравнение 3D-форматов [Електронний ресурс] / Dr. Arnulf Frohlich // PROSTEP AG. – 2011. – Режим доступу до ресурсу: http://www.cadcamcae.lv/hot/PROSTEP_n64_p53.pdf.
3. ECMA-335. Common Language Infrastructure (CLI) [Електронний ресурс] // ECMA International. – 2012. – Режим доступу до ресурсу: https://www.ecma-international.org/publications/files/ECMA-ST/ECMA-335.pdf.

ДОДАТКИ

Додаток А

Код програми

```
using System.Collections.Generic;
using System.IO;
using System.Text;
using U3DSpace.IO;
using U3DSpace.Primitives;
using U3DSpace.Primitives.MaterialPrimitives;
using U3DSpace.Primitives.MeshPrimitives;
using U3DSpace.Primitives.NodePrimitives;
using U3DSpace.Primitives.TexturePrimitives;

namespace U3DSpace
{
   public class U3DDocument
   {
      #region Constructors

      public U3DDocument()
      {
         Shaders = new Dictionary<string, Shader>();
         Materials = new Dictionary<string, Material>();
         Meshes = new Dictionary<string, Mesh>();
         Nodes = new Dictionary<string, Node>();
         Textures = new Dictionary<string, Texture>();
         TextEncoding = Encoding.UTF8;
      }

      #endregion Constructors

      #region Properties

      public Dictionary<string, Material> Materials { get; internal set; }
      public Dictionary<string, Mesh> Meshes { get; internal set; }
      public Dictionary<string, Node> Nodes { get; internal set; }
      public Dictionary<string, Shader> Shaders { get; internal set; }
      public Dictionary<string, Texture> Textures { get; internal set; }
      public Encoding TextEncoding { get; }

      #endregion Properties

      #region Methods

      public bool TryAddNode(Node node)
      {
         if ((node == null) || string.IsNullOrEmpty(node.Name) || Nodes.ContainsKey(node.Name))
         {
            return false;
         }
         if (!string.IsNullOrEmpty(node.Mesh) && !Meshes.ContainsKey(node.Mesh))
         {
            return false;
         }
         if (!string.IsNullOrEmpty(node.Parent) && !Nodes.ContainsKey(node.Parent))
         {
```

```
                return false;
            }
        Nodes.Add(node.Name, node);
        return true;
    }

    public bool TryAddMesh(Mesh mesh)
    {
        if ((mesh == null) || string.IsNullOrEmpty(mesh.Name) || Meshes.ContainsKey(mesh.Name))
        {
            return false;
        }
        if (string.IsNullOrEmpty(mesh.Shader) || !Shaders.ContainsKey(mesh.Shader))
        {
            return false;
        }
        if (!mesh.IsTrianglesCorrect())
        {
            return false;
        }
        Meshes.Add(mesh.Name, mesh);
        return true;
    }

    public bool TryAddMaterial(Material material)
    {
        if ((material == null) || string.IsNullOrEmpty(material.Name) || Materials.ContainsKey(material.Name))
        {
            return false;
        }
        Materials.Add(material.Name, material);
        return true;
    }

    public bool TryAddShader(Shader shader)
    {
        if ((shader == null) || string.IsNullOrEmpty(shader.Name) || Shaders.ContainsKey(shader.Name))
        {
            return false;
        }
        if (string.IsNullOrEmpty(shader.Material) || !Materials.ContainsKey(shader.Material))
        {
            return false;
        }
        if (!string.IsNullOrEmpty(shader.Texture) && !Textures.ContainsKey(shader.Texture))
        {
            return false;
        }
        Shaders.Add(shader.Name, shader);
        return true;
    }

    public bool TryAddTexture(Texture texture)
    {
        if ((texture == null) || string.IsNullOrEmpty(texture.Name) || Textures.ContainsKey(texture.Name))
        {
            return false;
        }
        if (texture.Image.Length == 0)
        {
            return false;
        }
        if (texture.ImageFormat == ImageFormat.Invalid)
        {
            return false;
```

```
                                                    }
                                                    Textures.Add(texture.Name, texture);
            return true;
        }

        public void Save(Stream stream, bool leaveOpen = false) => DocumentWriter.Save(this, stream,
leaveOpen);

        public void SaveToFile(string filePath)
        {
            using (var stream = new FileStream(filePath, FileMode.Create))
            {
                Save(stream);
            }
        }

        public void SavePdf(Stream stream) => DocumentWriter.SavePdf(this, stream);

        public void SaveToPdfFile(string filePath)
        {
            using (var stream = new FileStream(filePath, FileMode.Create))
            {
                SavePdf(stream);
            }
        }

        public void Read(Stream stream, bool leaveOpen = false) => DocumentReader.Read(this, stream,
leaveOpen);

        public void ReadFromFile(string filePath)
        {
            using (var stream = new FileStream(filePath, FileMode.Open))
            {
                Read(stream);
            }
        }

        #endregion Methods
    }
}
```